Update on the Voice Model:

For the regression task (predicting total UPDRS), I experimented with various models including multiple neural networks. The best validation R^2 I achieved was around 0.47, but the test R^2 dropped to ~0.39, suggesting some overfitting or data imbalance. I believe there's room for improvement in several areas:

- Data processing, particularly stratified splitting and potential outlier handling
- Hyperparameter tuning, especially learning rate schedules and network depth
- Advanced methods, like ensembling or leveraging pretrained models for feature extraction

For the classification task, my best model reached 66% validation accuracy. There's likely headroom here too, I found a published model that achieved 86% accuracy. https://github.com/mrpintime/Parkinsons-Telemonitoring/tree/main

- Same process as above

Further details and specs are documented throughout the notebook

Voice Processing

Imports

```
!pip install ucimlrepo
!pip install xgboost

import math

import numpy as np
import pandas as pd
import scipy
import seaborn as sns
import matplotlib.pyplot as plt

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, LeakyReLU
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint, ReduceLROnPlateau
from tensorflow.keras.utils import to_categorical

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor,RandomForestClassi
from sklearn.model_selection import train_test_split, GridSearchCV, ParameterGrid
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error, r2_score, classification_report, confusion_matri
```

```
from sklearn.preprocessing import StandardScaler, RobustScaler
from ucimlrepo import fetch_ucirepo
from xgboost import XGBRegressor
```

```
Requirement already satisfied: ucimlrepo in /root/venv/lib/python3.10/site-packages (0.0.7)
Requirement already satisfied: certifi>=2020.12.5 in /root/venv/lib/python3.10/site-packages (fro
Requirement already satisfied: pandas>=1.0.0 in /root/venv/lib/python3.10/site-packages (from uci
Requirement already satisfied: python-dateutil>=2.8.2 in /root/venv/lib/python3.10/site-packages
Requirement already satisfied: pytz>=2020.1 in /root/venv/lib/python3.10/site-packages (from panc
Requirement already satisfied: tzdata>=2022.1 in /root/venv/lib/python3.10/site-packages (from pa
Requirement already satisfied: numpy<2,>=1.22.4 in /root/venv/lib/python3.10/site-packages (from
Requirement already satisfied: six>=1.5 in /root/venv/lib/python3.10/site-packages (from python-d

[notice] A new release of pip is available: 23.0.1 -> 25.2
[notice] To update, run: pip install --upgrade pip
Collecting xgboost
  Downloading xgboost-3.0.3-py3-none-manylinux_2_28_x86_64.whl (253.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 253.8/253.8 MB 2.4 MB/s eta 0:00:00
Collecting nvidia-nccl-cu12
  Downloading nvidia_nccl_cu12-2.27.6-py3-none-manylinux2014_x86_64.manylinux_2_17_x86_64.whl (32
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 322.5/322.5 MB 1.5 MB/s eta 0:00:00
Requirement already satisfied: scipy in /root/venv/lib/python3.10/site-packages (from xgboost) (1
Requirement already satisfied: numpy in /root/venv/lib/python3.10/site-packages (from xgboost) (1
Installing collected packages: nvidia-nccl-cu12, xgboost
Successfully installed nvidia-nccl-cu12-2.27.6 xgboost-3.0.3

[notice] A new release of pip is available: 23.0.1 -> 25.2
[notice] To update, run: pip install --upgrade pip
2025-08-03 00:25:58.008054: I external/local_tsl/tsl/cuda/cudart_stub.cc:31] Could not find cuda
2025-08-03 00:25:58.059756: E external/local_xla/xla/stream_executor/cuda/cuda_dnn.cc:9261] Unabl
2025-08-03 00:25:58.060536: E external/local_xla/xla/stream_executor/cuda/cuda_fft.cc:607] Unable
2025-08-03 00:25:58.061727: E external/local_xla/xla/stream_executor/cuda/cuda_blas.cc:1515] Unab
2025-08-03 00:25:58.069839: I external/local_tsl/tsl/cuda/cudart_stub.cc:31] Could not find cuda
2025-08-03 00:25:58.070577: I tensorflow/core/platform/cpu_feature_guard.cc:182] This TensorFlow
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the
2025-08-03 00:25:59.640362: W tensorflow/compiler/tf2tensorrt/utils/py_utils.cc:38] TF-TRT Warnir
```

xShortcut for all the processing - So after running this block you can go immediately to model development - Not for classifications tho (there's further processing that has to be done)

Feel free to experiment with these since some of the choices were arbitrary

```
seed = 42
np.random.seed(seed)
tf.random.set_seed(seed)
parkinsons_voice = fetch_ucirepo(id=189)
X = parkinsons_voice.data.features
y = parkinsons_voice.data.targets

features_to_drop = [
    'Jitter(%)', 'Jitter:RAP', 'Jitter:DDP',
```

```
    'Shimmer(dB)', 'Shimmer:APQ5', 'Shimmer:DDA',
    'Shimmer', 'NHR',   'age',
    'test_time', 'subject#'
] #to address multicollinearity and also to overcome overfitting based on age as id

X_filtered = X.drop(columns=[col for col in features_to_drop if col in X.columns])

y_target = y["total_UPDRS"]
#y_target = y["motor_UPDRS"]

#this section can definitely be improved to split the data using subject# so that testings wille
#or any other type of good stratification
X_train, X_temp, y_train, y_temp = train_test_split(
    X_filtered, y_target, test_size=0.5, random_state=seed
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=seed
)

#Necessary for NN
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

Downloading Data & Other Stuff

```
seed = 42
np.random.seed(seed)
tf.random.set_seed(seed)
```

```
parkinsons_voice = fetch_ucirepo(id=189)
X = parkinsons_voice.data.features
y = parkinsons_voice.data.targets
```

```
print(parkinsons_voice.metadata)
print(parkinsons_voice.variables)
```

```
{'uci_id': 189, 'name': 'Parkinsons Telemonitoring', 'repository_url': 'https://archive.ics.uci.e
             name     role        type demographic  \
0        subject#       ID     Integer        None
1             age  Feature     Integer         Age
2       test_time  Feature  Continuous        None
3       Jitter(%)  Feature  Continuous        None
4     Jitter(Abs)  Feature  Continuous        None
5      Jitter:RAP  Feature  Continuous        None
6     Jitter:PPQ5  Feature  Continuous        None
7      Jitter:DDP  Feature  Continuous        None
8         Shimmer  Feature  Continuous        None
9     Shimmer(dB)  Feature  Continuous        None
10   Shimmer:APQ3  Feature  Continuous        None
11   Shimmer:APQ5  Feature  Continuous        None
12  Shimmer:APQ11  Feature  Continuous        None
13    Shimmer:DDA  Feature  Continuous        None
14            NHR  Feature  Continuous        None
15            HNR  Feature  Continuous        None
16           RPDE  Feature  Continuous        None
17            DFA  Feature  Continuous        None
18            PPE  Feature  Continuous        None
```

```
19    motor_UPDRS   Target  Continuous        None
20    total_UPDRS   Target  Continuous        None
21            sex  Feature      Binary         Sex

                                        description units missing_values
0       Integer that uniquely identifies each subject  None             no
1                                       Subject age  None             no
2   Time since recruitment into the trial. The int...  None             no
3   Several measures of variation in fundamental f...  None             no
4   Several measures of variation in fundamental f...  None             no
5   Several measures of variation in fundamental f...  None             no
6   Several measures of variation in fundamental f...  None             no
7   Several measures of variation in fundamental f...  None             no
8          Several measures of variation in amplitude  None             no
9          Several measures of variation in amplitude  None             no
10         Several measures of variation in amplitude  None             no
11         Several measures of variation in amplitude  None             no
12         Several measures of variation in amplitude  None             no
13         Several measures of variation in amplitude  None             no
14  Two measures of ratio of noise to tonal compon...  None             no
15  Two measures of ratio of noise to tonal compon...  None             no
16           A nonlinear dynamical complexity measure  None             no
17                     Signal fractal scaling exponent  None             no
18  A nonlinear measure of fundamental frequency v...  None             no
19  Clinician's motor UPDRS score, linearly interp...  None             no
20  Clinician's total UPDRS score, linearly interp...  None             no
21               Subject sex '0' - male, '1' - female  None             no
```

Data Processing

```
df = parkinsons_voice.data.features.copy()
df['total_UPDRS'] = parkinsons_voice.data.targets['total_UPDRS']
print(df.shape)
print(df.columns.tolist())
print(df.head(5))

print(df.info())
print(df.isna().sum())

print(df.describe().T)
```

```
Jitter:DDP      5875.0   0.008962   0.009371   0.000980   0.004730   0.006750
Shimmer         5875.0   0.034035   0.025835   0.003060   0.019120   0.027510
Shimmer(dB)     5875.0   0.310960   0.230254   0.026000   0.175000   0.253000
Shimmer:APQ3    5875.0   0.017156   0.013237   0.001610   0.009280   0.013700
Shimmer:APQ5    5875.0   0.020144   0.016664   0.001940   0.010790   0.015940
Shimmer:APQ11   5875.0   0.027481   0.019986   0.002490   0.015665   0.022710
Shimmer:DDA     5875.0   0.051467   0.039711   0.004840   0.027830   0.041110
NHR             5875.0   0.032120   0.059692   0.000286   0.010955   0.018448
HNR             5875.0  21.679495   4.291096   1.659000  19.406000  21.920000
RPDE            5875.0   0.541473   0.100986   0.151020   0.469785   0.542250
DFA             5875.0   0.653240   0.070902   0.514040   0.596180   0.643600
PPE             5875.0   0.219589   0.091498   0.021983   0.156340   0.205500
sex             5875.0   0.317787   0.465656   0.000000   0.000000   0.000000
total_UPDRS     5875.0  29.018942  10.700283   7.000000  21.371000  27.576000

                      75%         max
age             72.000000   85.000000
test_time      138.445000  215.490000
Jitter(%)        0.006800    0.099990
Jitter(Abs)      0.000053    0.000446
Jitter:RAP       0.003290    0.057540
Jitter:PPQ5      0.003460    0.069560
Jitter:DDP       0.009870    0.172630
Shimmer          0.039750    0.268630
Shimmer(dB)      0.365000    2.107000
Shimmer:APQ3     0.020575    0.162670
Shimmer:APQ5     0.023755    0.167020
Shimmer:APQ11    0.032715    0.275460
Shimmer:DDA      0.061735    0.488020
NHR              0.031463    0.748260
HNR             24.444000   37.875000
RPDE             0.614045    0.966080
DFA              0.711335    0.865600
PPE              0.264490    0.731730
sex              1.000000    1.000000
total_UPDRS     36.399000   54.992000
```

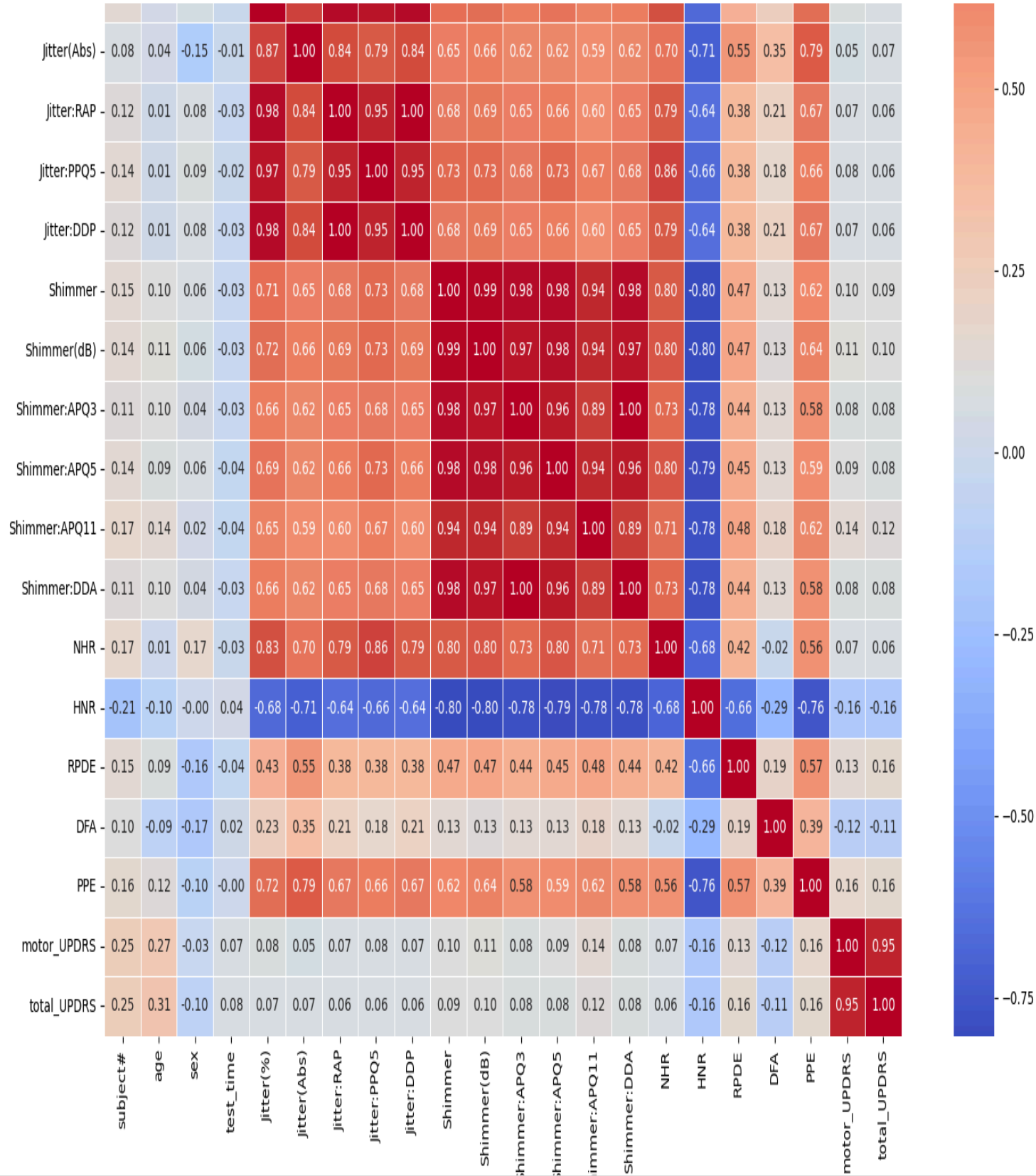
Correlation Heat Maps

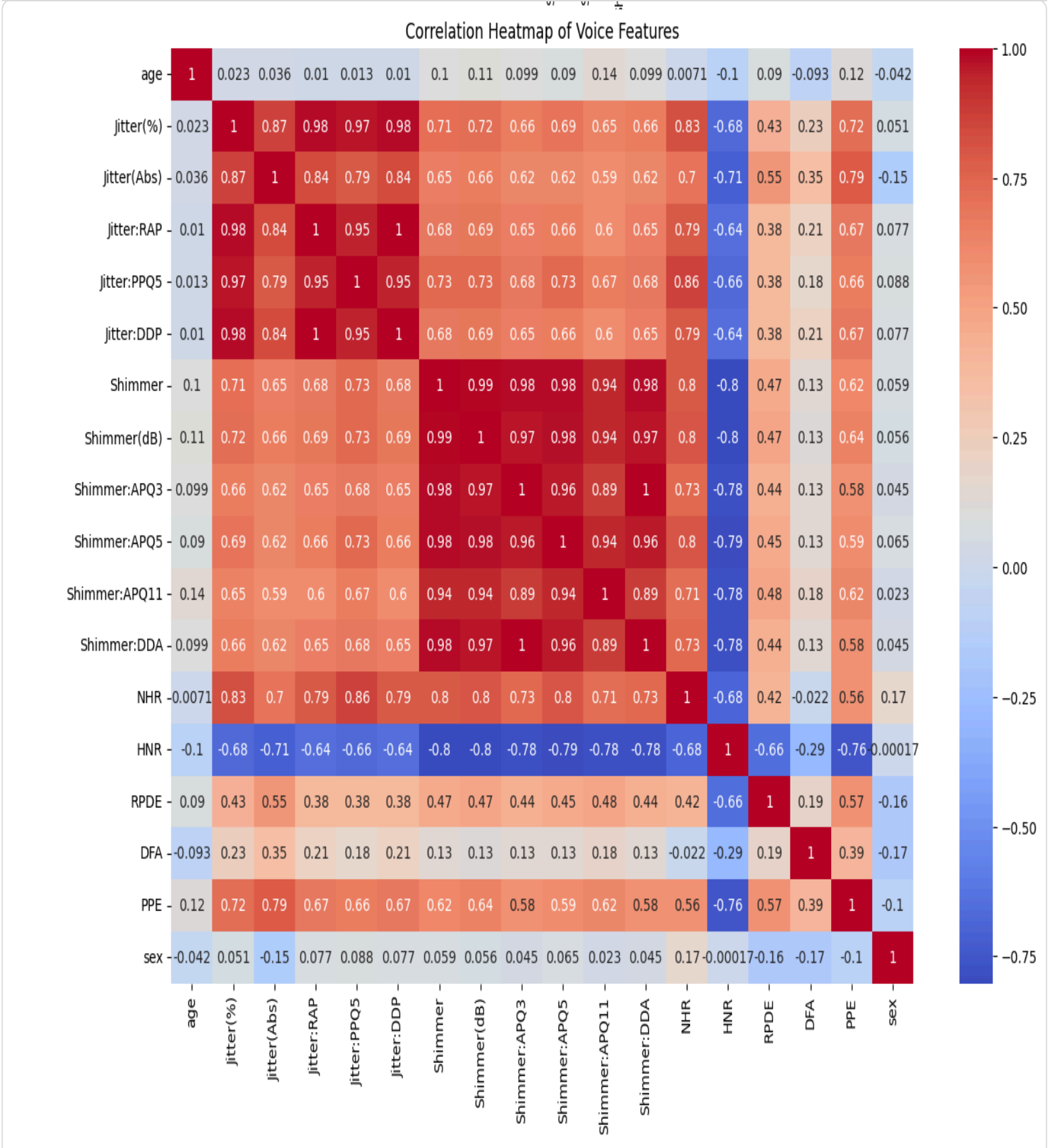
```
df = parkinsons_voice.data.original.copy()
corr_matrix = df.corr()

plt.figure(figsize=(15, 12))
sns.heatmap(corr_matrix, annot=True, fmt=".2f", cmap="coolwarm", square=True, linewidths=0.5)
plt.title("Correlation Heatmap of Parkinson's Voice Dataset Features")
plt.tight_layout()
plt.show()
```

Correlation Heatmap of Parkinson's Voice Dataset Features

```
if 'subject#' in X.columns:
    X = X.drop(columns=['subject#'])
if 'test_time' in X.columns:
    X = X.drop(columns=['test_time'])
```

```
plt.figure(figsize=(14, 10))
sns.heatmap(X.corr(), cmap='coolwarm',annot = True)
plt.title("Correlation Heatmap of Voice Features")
plt.xticks(rotation=90)
plt.show()
```





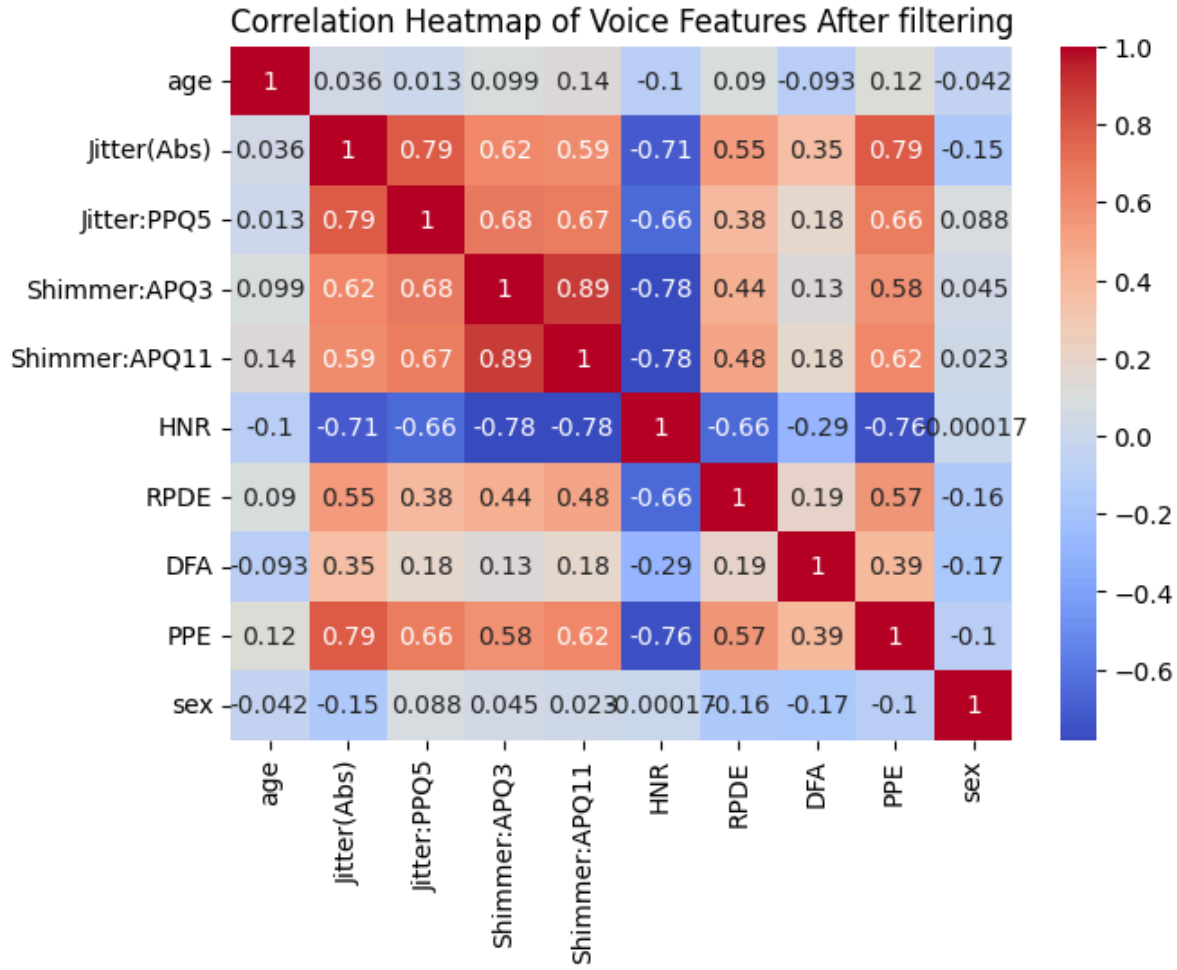

```
'''sns.pairplot(X, corner=True, plot_kws={"s": 10})
plt.suptitle("All Feature vs Feature Pairwise Plots", y=1.02)
plt.show()'''
#Takes too long - No need to run
```

```
'sns.pairplot(X, corner=True, plot_kws={"s": 10})\nplt.suptitle("All Feature vs Feature Pairwise
Plots", y=1.02)\nplt.show()'
```

Dropping Columns with over 90% correlation

```
features_to_drop = [
    'Jitter(%)', 'Jitter:RAP', 'Jitter:DDP',
    'Shimmer(dB)', 'Shimmer:APQ5', 'Shimmer:DDA',
    'Shimmer', 'NHR'
]
X_filtered = X.drop(columns=[col for col in features_to_drop if col in X.columns])
```

```
plt.figure(figsize=(7, 5))
sns.heatmap(X_filtered.corr(), cmap='coolwarm', annot = True)
plt.title("Correlation Heatmap of Voice Features After filtering")
plt.xticks(rotation=90)
plt.show()
```




```
'''sns.pairplot(X, corner=True, plot_kws={"s": 10})
plt.suptitle("All Feature vs Feature Pairwise Plots", y=1.02)
plt.show()'''
```

```
'sns.pairplot(X, corner=True, plot_kws={"s": 10})\nplt.suptitle("All Feature vs Feature Pairwise
Plots", y=1.02)\nplt.show()'
```

Final Optimized Drop Columns

Drops Age along with other redundant features

```
'''features_to_drop = [
    'Jitter(%)', 'Jitter:RAP', 'Jitter:DDP',
    'Shimmer(dB)', 'Shimmer:APQ5', 'Shimmer:DDA',
    'Shimmer', 'NHR', 'age'
]
X_filtered = X.drop(columns=[col for col in features_to_drop if col in X.columns])
'''
```

```
"features_to_drop = [\n    'Jitter(%)', 'Jitter:RAP', 'Jitter:DDP',\n    'Shimmer(dB)',
'Shimmer:APQ5', 'Shimmer:DDA',\n    'Shimmer', 'NHR', 'age'  \n]\nX_filtered = X.drop(columns=
[col for col in features_to_drop if col in X.columns])\n"
```

Target Selection + Data Split

Used Total_UPDRS because it can be applied more universally

POSSIBLE IMPROVEMENT: Data split should probably utilize the patient label so that validation and testing cols can have patients that the model did not see yet

```
print(df['total_UPDRS'].describe())
```

```
count    5875.000000
mean       29.018942
std        10.700283
min         7.000000
25%        21.371000
50%        27.576000
75%        36.399000
max        54.992000
Name: total_UPDRS, dtype: float64
```

```
y_target = y["total_UPDRS"]
#y_target = y["motor_UPDRS"]

X_train, X_temp, y_train, y_temp = train_test_split(
    X_filtered, y_target, test_size=0.5, random_state=seed
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=seed
)
```

Train & Optimize Random Forest Regression Model

Baseline Model

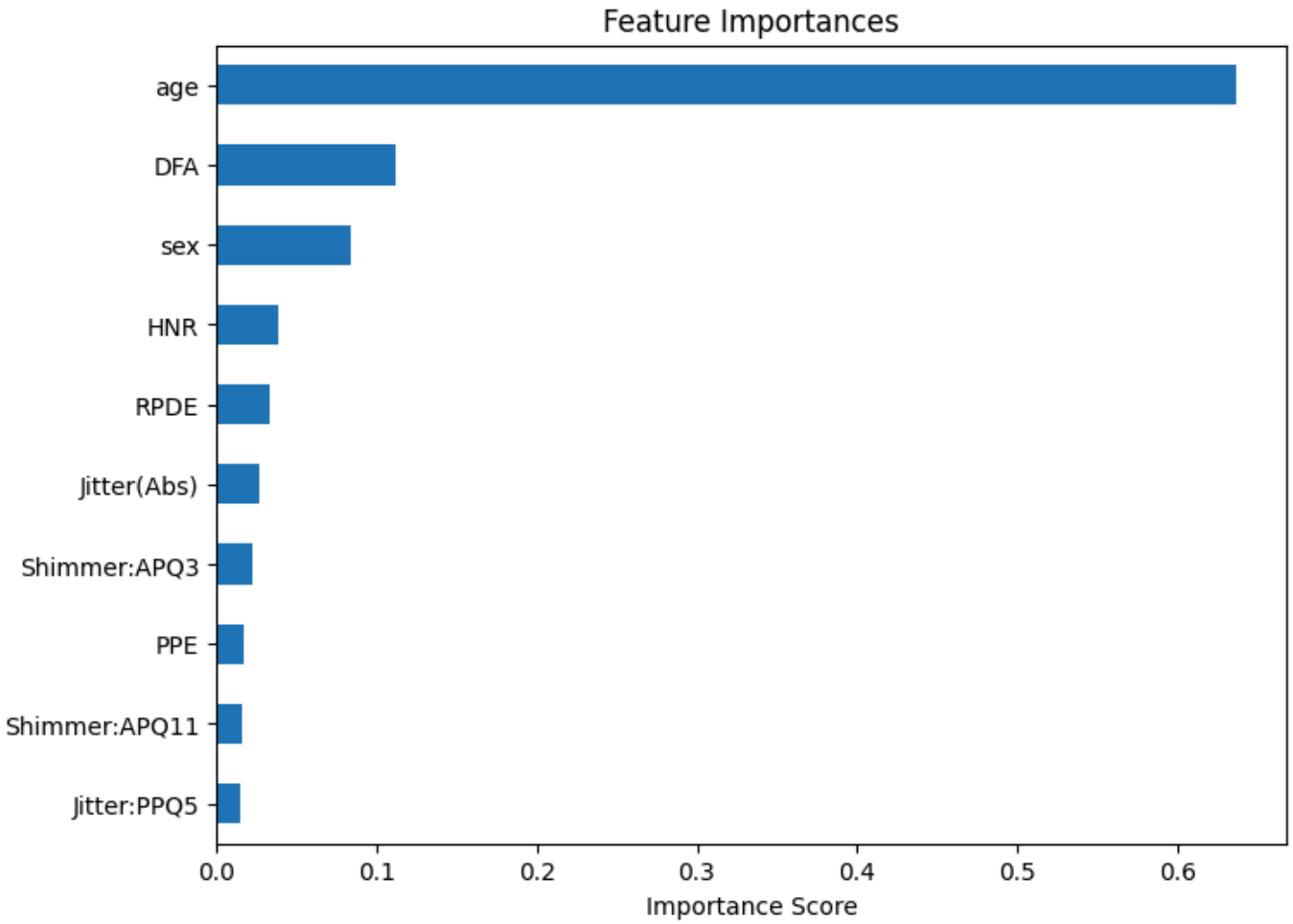
```
model = RandomForestRegressor(n_estimators=100, random_state=seed)
model.fit(X_train, y_train)

y_pred = model.predict(X_val)
```

```
rmse = mean_squared_error(y_val, y_pred, squared=False)
r2 = r2_score(y_val, y_pred)
print(f" RMSE: {rmse:.3f}")
print(f" R^2 Score: {r2:.3f}")
```

```
 RMSE: 3.215
 R^2 Score: 0.906
```

Check for overfitting

```
importances = pd.Series(model.feature_importances_, index=X_filtered.columns)
importances.sort_values().plot(kind='barh', figsize=(8, 6))
plt.title("Feature Importances")
plt.xlabel("Importance Score")
plt.show()
```




```
print(df['age'].value_counts().sort_index())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 1
----> 1 print(df['age'].value_counts().sort_index())

NameError: name 'df' is not defined
```

Overfitting, so removing Age (as it acts as an id) & Split Data Again

```
if 'age' in X.columns:
    X_filtered = X_filtered.drop(columns=["age"])

X_train, X_temp, y_train, y_temp = train_test_split(
    X_filtered, y_target, test_size=0.5, random_state=seed
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=seed
)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[52], line 2
      1 if 'age' in X.columns:
----> 2     X_filtered = X_filtered.drop(columns=["age"])
      4 X_train, X_temp, y_train, y_temp = train_test_split(
      5     X_filtered, y_target, test_size=0.5, random_state=seed
      6 )
      8 X_val, X_test, y_val, y_test = train_test_split(
      9     X_temp, y_temp, test_size=0.5, random_state=seed
     10 )

File ~/venv/lib/python3.10/site-packages/pandas/core/frame.py:5344, in DataFrame.drop(self,
labels, axis, index, columns, level, inplace, errors)
   5196 def drop(
   5197     self,
   5198     labels: IndexLabel | None = None,
   (...)
   5205     errors: IgnoreRaise = "raise",
   5206 ) -> DataFrame | None:
   5207     """
   5208     Drop specified labels from rows or columns.
   5209
   (...)
   5342             weight  1.0     0.8
   5343     """
-> 5344     return super().drop(
   5345         labels=labels,
   5346         axis=axis,
   5347         index=index,
   5348         columns=columns,
   5349         level=level,
   5350         inplace=inplace,
   5351         errors=errors,
   5352     )

File ~/venv/lib/python3.10/site-packages/pandas/core/generic.py:4711, in NDFrame.drop(self,
labels, axis, index, columns, level, inplace, errors)
   4709 for axis, labels in axes.items():
   4710     if labels is not None:
-> 4711         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4713 if inplace:
   4714     self._update_inplace(obj)

File ~/venv/lib/python3.10/site-packages/pandas/core/generic.py:4753, in
NDFrame._drop_axis(self, labels, axis, level, errors, only_slice)
   4751         new_axis = axis.drop(labels, level=level, errors=errors)
   4752     else:
-> 4753         new_axis = axis.drop(labels, errors=errors)
   4754     indexer = axis.get_indexer(new_axis)
   4756 # Case for non-unique axis
   4757 else:

File ~/venv/lib/python3.10/site-packages/pandas/core/indexes/base.py:7000, in Index.drop(self,
labels, errors)
   6998 if mask.any():
   6999     if errors != "ignore":
-> 7000         raise KeyError(f"{labels[mask].tolist()} not found in axis")
   7001     indexer = indexer[~mask]
   7002 return self.delete(indexer)
```

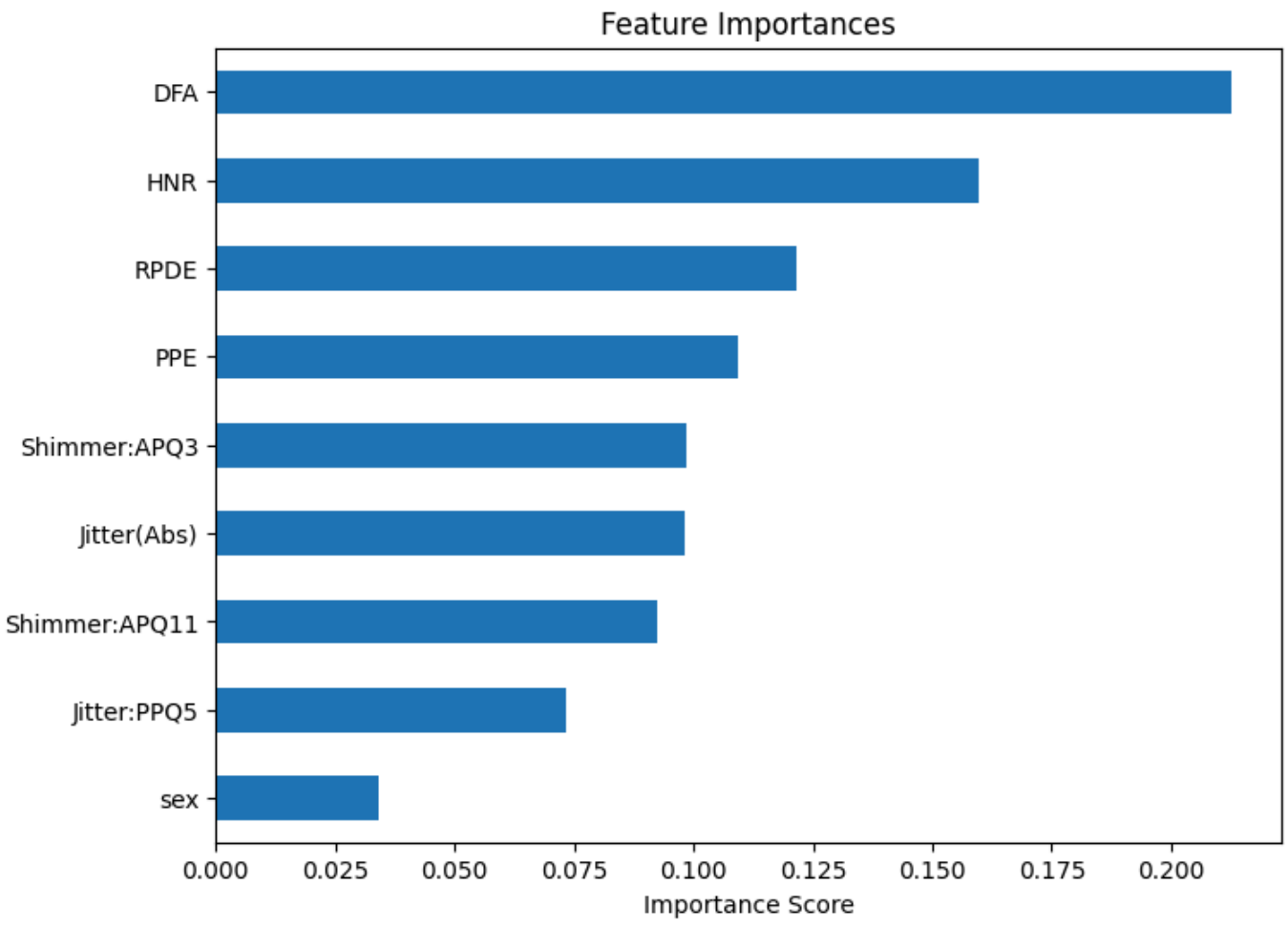
Check for overfitting again

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, explained_varianc
```

```
# Train
model = RandomForestRegressor(n_estimators=100, random_state=seed)
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_val)

# Metrics
rmse = mean_squared_error(y_val, y_pred, squared=False)
mae  = mean_absolute_error(y_val, y_pred)
r2   = r2_score(y_val, y_pred)
ev   = explained_variance_score(y_val, y_pred)

print(f"RMSE:               {rmse:.3f}")
print(f"MAE:                {mae:.3f}")
print(f"R² Score:           {r2:.3f}")
print(f"Explained Variance: {ev:.3f}")
```

```
RMSE: 8.394
R^2 Score: 0.358
```

```
importances = pd.Series(model.feature_importances_, index=X_train.columns)
importances.sort_values().plot(kind='barh', figsize=(8, 6))
plt.title("Feature Importances")
plt.xlabel("Importance Score")
plt.show()
```



Looks like there are no overfitting and the importance appears to be evenly distributed. Did try dropping sex columns but it lowered the R^2 score

Hyper-parameter Optimization

No need to run the Optimization Blocks Again

Grid Search

```
'''param_grid = {
    "n_estimators": [100, 200],
    "max_depth": [None, 10, 20],
    "min_samples_split": [2, 5],
    "min_samples_leaf": [1, 2],
    "max_features": [1.0, "sqrt"]
}

best_r2 = -float("inf")
best_model = None
best_params = None

for params in ParameterGrid(param_grid):
    model = RandomForestRegressor(**params, random_state=42)
    model.fit(X_train, y_train)
    val_pred = model.predict(X_val)
    r2 = r2_score(y_val, val_pred)

    if r2 > best_r2:
        best_r2 = r2
        best_model = model
        best_params = params

print("Best R² on validation set:", best_r2)
print("Best hyperparameters:", best_params)'''
```

```
'param_grid = {\n    "n_estimators": [100, 200],\n    "max_depth": [None, 10, 20],\n
"min_samples_split": [2, 5],\n    "min_samples_leaf": [1, 2],\n    "max_features": [1.0,
"sqrt"]\n}\n\nbest_r2 = -float("inf")\nbest_model = None\nbest_params = None\n\nfor params in
ParameterGrid(param_grid):\n    model = RandomForestRegressor(**params, random_state=42)\n
model.fit(X_train, y_train)\n    val_pred = model.predict(X_val)\n    r2 = r2_score(y_val,
val_pred)\n\n    if r2 > best_r2:\n        best_r2 = r2\n        best_model = model\n
best_params = params\n\nprint("Best R² on validation set:", best_r2)\nprint("Best
hyperparameters:", best_params)'
```

```
'''param_grid = {
    "n_estimators": [200, 400, 600,800,1000],
    "max_depth": [None],
    "min_samples_split": [2],
    "min_samples_leaf": [2],
}

best_r2 = -float("inf")
best_model = None
best_params = None
```

```
for params in ParameterGrid(param_grid):
    model = RandomForestRegressor(**params, random_state=42)
    model.fit(X_train, y_train)
    val_pred = model.predict(X_val)
    r2 = r2_score(y_val, val_pred)

    if r2 > best_r2:
        best_r2 = r2
        best_model = model
        best_params = params

print("Best R² on validation set:", best_r2)
print("Best hyperparameters:", best_params)
'''
```

```
'param_grid = {\n    "n_estimators": [200, 400, 600,800,1000],\n    "max_depth": [None],\n
"min_samples_split": [2],\n    "min_samples_leaf": [2],\n}\n\nbest_r2 = -
float("inf")\nbest_model = None\nbest_params = None\n\nfor params in
ParameterGrid(param_grid):\n    model = RandomForestRegressor(**params, random_state=42)\n
model.fit(X_train, y_train)\n    val_pred = model.predict(X_val)\n    r2 = r2_score(y_val,
val_pred)\n\n    if r2 > best_r2:\n        best_r2 = r2\n        best_model = model\n
best_params = params\n\nprint("Best R² on validation set:", best_r2)\nprint("Best
hyperparameters:", best_params)\n'
```

```
'''param_grid = {
    "n_estimators": [1000, 1500, 2000, 2500],
    "max_depth": [None],
    "min_samples_split": [2],
    "min_samples_leaf": [2],
}

best_r2 = -float("inf")
best_model = None
best_params = None

for params in ParameterGrid(param_grid):
    model = RandomForestRegressor(**params, random_state=42)
    model.fit(X_train, y_train)
    val_pred = model.predict(X_val)
    r2 = r2_score(y_val, val_pred)

    if r2 > best_r2:
        best_r2 = r2
        best_model = model
        best_params = params

print("Best R² on validation set:", best_r2)
print("Best hyperparameters:", best_params)'''
```

```
'param_grid = {\n    "n_estimators": [1000, 1500, 2000, 2500],\n    "max_depth": [None],\n
"min_samples_split": [2],\n    "min_samples_leaf": [2],\n}\n\nbest_r2 = -
float("inf")\nbest_model = None\nbest_params = None\n\nfor params in
ParameterGrid(param_grid):\n    model = RandomForestRegressor(**params, random_state=42)\n
model.fit(X_train, y_train)\n    val_pred = model.predict(X_val)\n    r2 = r2_score(y_val,
val_pred)\n\n    if r2 > best_r2:\n        best_r2 = r2\n        best_model = model\n
best_params = params\n\nprint("Best R² on validation set:", best_r2)\nprint("Best
hyperparameters:", best_params)'
```

```
'''param_grid = {
    "n_estimators": [1250, 1500, 1750],
```

```
    "max_depth": [None],
    "min_samples_split": [2],
    "min_samples_leaf": [2],
}

best_r2 = -float("inf")
best_model = None
best_params = None

for params in ParameterGrid(param_grid):
    model = RandomForestRegressor(**params, random_state=42)
    model.fit(X_train, y_train)
    val_pred = model.predict(X_val)
    r2 = r2_score(y_val, val_pred)

    if r2 > best_r2:
        best_r2 = r2
        best_model = model
        best_params = params

print("Best R² on validation set:", best_r2)
print("Best hyperparameters:", best_params)'''
```

```
'param_grid = {\n    "n_estimators": [1250, 1500, 1750],\n    "max_depth": [None],\n
"min_samples_split": [2],\n    "min_samples_leaf": [2],\n}\n\nbest_r2 = -
float("inf")\nbest_model = None\nbest_params = None\n\nfor params in
ParameterGrid(param_grid):\n    model = RandomForestRegressor(**params, random_state=42)\n
model.fit(X_train, y_train)\n    val_pred = model.predict(X_val)\n    r2 = r2_score(y_val,
val_pred)\n\n    if r2 > best_r2:\n        best_r2 = r2\n        best_model = model\n
best_params = params\n\nprint("Best R² on validation set:", best_r2)\nprint("Best
hyperparameters:", best_params)'
```

Best Random Forest Regression Model

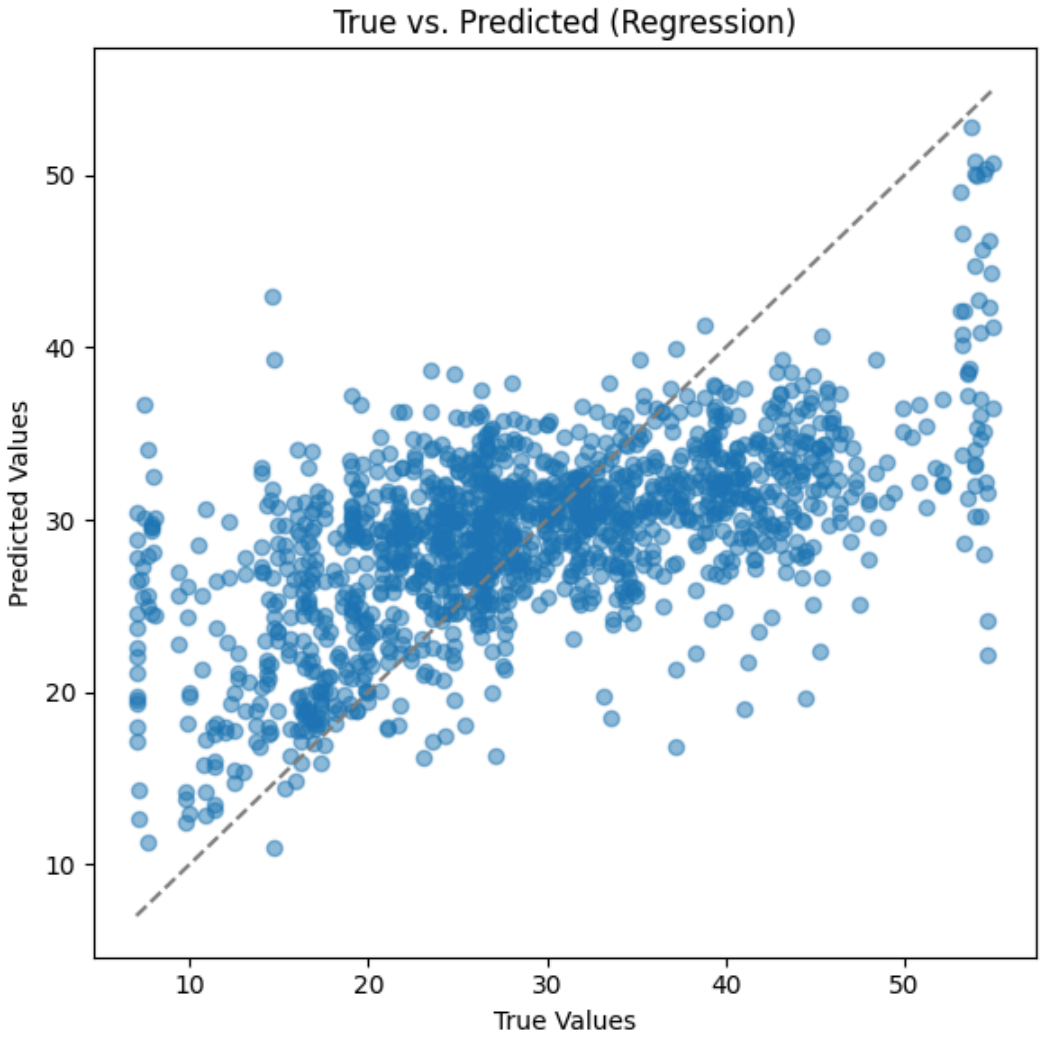
```
model = RandomForestRegressor(
    n_estimators=1250,
    max_depth=None,
    max_features='sqrt',
    min_samples_leaf=2,
    min_samples_split=2,
    random_state=seed
)

model.fit(X_train, y_train)

y_pred = model.predict(X_val)

rmse = mean_squared_error(y_val, y_pred, squared=False)
r2 = r2_score(y_val, y_pred)

print(f"RMSE: {rmse:.3f}")
print(f"R^2 Score: {r2:.3f}")
```

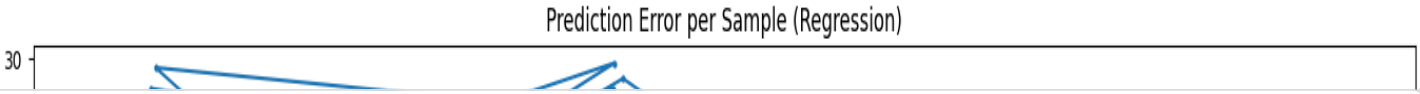
```
RMSE: 8.307
R^2 Score: 0.371
```

```
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

```
# Predict (no argmax)
y_test_pred = model.predict(X_test).flatten()

# Evaluation
print("RMSE:", mean_squared_error(y_test, y_test_pred, squared=False))
print("MAE:", mean_absolute_error(y_test, y_test_pred))
print("R² Score:", r2_score(y_test, y_test_pred))

plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_test_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--', color='gray')
plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.title("True vs. Predicted (Regression)")
plt.tight_layout()
plt.show()

errors = y_test_pred - y_test

plt.figure(figsize=(14, 3))
plt.plot(errors, marker='o', linestyle='-', markersize=2)
plt.axhline(0, color='gray', linestyle='--')
plt.title("Prediction Error per Sample (Regression)")
plt.xlabel("Sample Index")
plt.ylabel("Prediction Error")
plt.tight_layout()
plt.show()

# Compute prediction error
errors = y_test_pred - y_test
abs_errors = np.abs(errors)

# Find the index of the largest error
outlier_idx = np.argmax(abs_errors)

# Remove the outlier from predictions and ground truth
y_test_no_outlier = np.delete(y_test, outlier_idx)
y_pred_no_outlier = np.delete(y_test_pred, outlier_idx)
errors_no_outlier = y_pred_no_outlier - y_test_no_outlier

plt.figure(figsize=(6, 6))
plt.scatter(y_test_no_outlier, y_pred_no_outlier, alpha=0.6)
plt.plot([y_test_no_outlier.min(), y_test_no_outlier.max()],
         [y_test_no_outlier.min(), y_test_no_outlier.max()],
         'k--', lw=2)
plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.title("True vs. Predicted (Regression, Outlier Removed)")
plt.tight_layout()
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 3.5))
plt.plot(errors_no_outlier, linestyle='-', marker='o', markersize=2, alpha=0.8)
plt.axhline(0, color='gray', linestyle='--')
plt.title("Prediction Error per Sample (Outlier Removed)")
plt.xlabel("Sample Index")
plt.ylabel("Prediction Error")
plt.tight_layout()
plt.grid(True)
plt.show()
```

```
RMSE: 8.53154285953858
MAE: 6.6600981418705585
R² Score: 0.35952284526494127
```

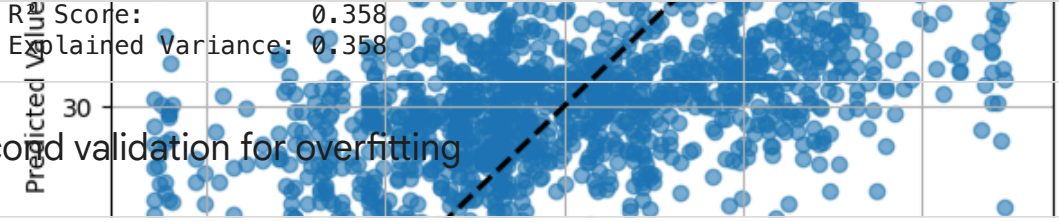





```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, explained_varianc

# Train
model = RandomForestRegressor(n_estimators=100, random_state=seed)
model.fit(X_train, y_train)

# Predict
y_pred = model.predict(X_val)

# Metrics
rmse = mean_squared_error(y_val, y_pred, squared=False)
mae  = mean_absolute_error(y_val, y_pred)
r2   = r2_score(y_val, y_pred)
ev   = explained_variance_score(y_val, y_pred)

print(f"RMSE:               {rmse:.3f}")
print(f"MAE:                {mae:.3f}")
print(f"R² Score:           {r2:.3f}")
print(f"Explained Variance: {ev:.3f}")
```

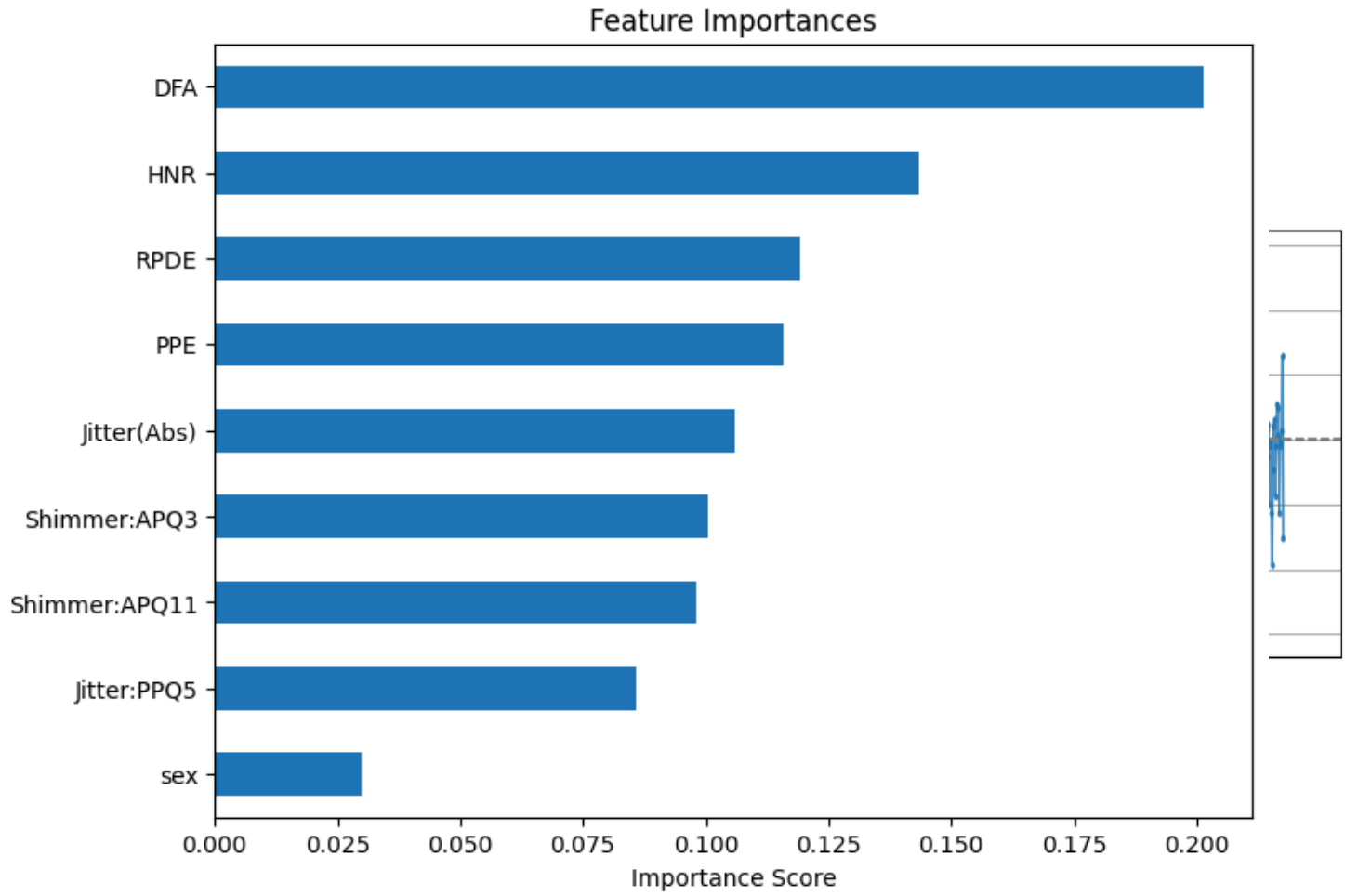
```
RMSE:  8.394
MAE:   6.477
```

```
R² Score:          0.358
Explained Variance: 0.358
```



Second validation for overfitting

```
importances = pd.Series(model.feature_importances_, index=X_filtered.columns)
importances.sort_values().plot(kind='barh', figsize=(8, 6))
plt.title("Feature Importances")
plt.xlabel("Importance Score")
plt.show()
```



Not overfitting, but the model performance is not that great

Train Other Model Architectures

```
# Takes too long so no need to run again
'''models = {
    "Linear Regression": LinearRegression(),
    "Ridge Regression": Ridge(alpha=1.0),
    "Lasso Regression": Lasso(alpha=0.1),
    "ElasticNet": ElasticNet(alpha=0.1, l1_ratio=0.5),
    "Support Vector Regressor": SVR(kernel='rbf', C=1.0, epsilon=0.2),
    "MLP Regressor": MLPRegressor(hidden_layer_sizes=(64, 64), max_iter=1000, random_state=seed)
    "KNN Regressor": KNeighborsRegressor(n_neighbors=5),
    "Random Forest": RandomForestRegressor(
```

```
        n_estimators=1500, max_depth=None, max_features='sqrt',
        min_samples_leaf=2, min_samples_split=2, random_state=seed
    ),
    "Gradient Boosting": GradientBoostingRegressor(n_estimators=300, learning_rate=0.1, random_s
    "XGBoost Regressor": XGBRegressor(n_estimators=300, learning_rate=0.1, random_state=42)
}

results = []
for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_val)
    r2 = r2_score(y_val, y_pred)
    rmse = mean_squared_error(y_val, y_pred, squared=False)
    results.append((name, r2, rmse))

results_df = pd.DataFrame(results, columns=["Model", "Validation R²", "Validation RMSE"])
results_df.sort_values(by="Validation R²", ascending=False, inplace=True)

print(results_df)'''
```

```
'models = {\n    "Linear Regression": LinearRegression(),\n    "Ridge Regression":
Ridge(alpha=1.0),\n    "Lasso Regression": Lasso(alpha=0.1),\n    "ElasticNet":
ElasticNet(alpha=0.1, l1_ratio=0.5),\n    "Support Vector Regressor": SVR(kernel=\'rbf\', C=1.0,
epsilon=0.2),\n    "MLP Regressor": MLPRegressor(hidden_layer_sizes=(64, 64), max_iter=1000,
random_state=seed),\n    "KNN Regressor": KNeighborsRegressor(n_neighbors=5),\n    "Random
Forest": RandomForestRegressor(\n        n_estimators=1500, max_depth=None,
max_features=\'sqrt\',\n        min_samples_leaf=2, min_samples_split=2, random_state=seed\n
),\n    "Gradient Boosting": GradientBoostingRegressor(n_estimators=300, learning_rate=0.1,
random_state=seed),\n    "XGBoost Regressor": XGBRegressor(n_estimators=300, learning_rate=0.1,
random_state=42)\n}\n\nresults = []\nfor name, model in models.items():\n    model.fit(X_train,
y_train)\n    y_pred = model.predict(X_val)\n    r2 = r2_score(y_val, y_pred)\n    rmse =
mean_squared_error(y_val, y_pred, squared=False)\n    results.append((name, r2,
rmse))\n\nresults_df = pd.DataFrame(results, columns=["Model", "Validation R²", "Validation
RMSE"])\nresults_df.sort_values(by="Validation R²", ascending=False,
inplace=True)\n\nprint(results_df)'
```

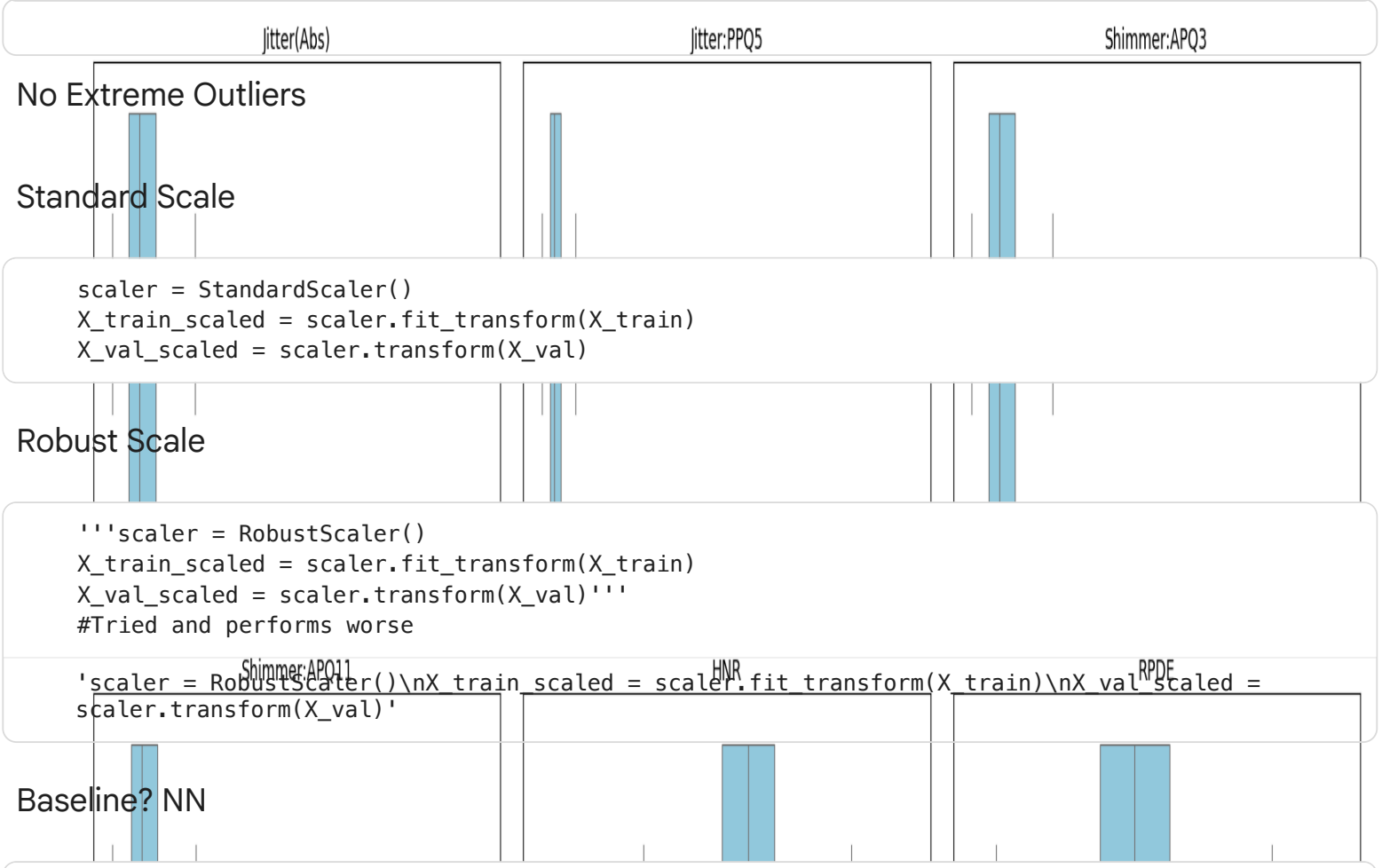
Random Forest is still the best model

Train Neural Network

Process data for NN

```
num_features = X_train.shape[1]
num_cols = 3  # Number of plots per row
num_rows = math.ceil(num_features / num_cols)

fig, axes = plt.subplots(num_rows, num_cols, figsize=(16, 4 * num_rows))
axes = axes.flatten()

for i, col in enumerate(X_train.columns):
    sns.boxplot(x=X_train[col], ax=axes[i], fliersize=2, linewidth=0.8, color="skyblue")
    axes[i].set_title(col)

# Hide unused subplots
for j in range(i + 1, len(axes)):
    axes[j].set_visible(False)
```

```
    plt.tight_layout()
    plt.show()
```



No Extreme Outliers

Standard Scale

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

Robust Scale

```
'''scaler = RobustScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)'''
#Tried and performs worse
```

```
'scaler = RobustScaler()\nX_train_scaled = scaler.fit_transform(X_train)\nX_val_scaled =
scaler.transform(X_val)'
```


Baseline? NN

```
# Build model
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.3),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1)  # Regression output
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='mse')

# Train
model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=100,
    batch_size=32,
    verbose=1
)

# Evaluate
y_pred = model.predict(X_val_scaled).flatten()
rmse = mean_squared_error(y_val, y_pred, squared=False)
r2 = r2_score(y_val, y_pred)

print(f"RMSE: {rmse:.3f}")
print(f"R^2 Score: {r2:.3f}")
```

```
DFA                                PPE                                sex
Epoch 1/100
92/92 [==============================] - 1s 5ms/step - loss: 458.9861 - val_loss: 522.9216
Epoch 2/100
92/92 [==============================] - 0s 3ms/step - loss: 151.3707 - val_loss: 343.0474
Epoch 3/100
92/92 [==============================] - 0s 3ms/step - loss: 133.2045 - val_loss: 205.1878
Epoch 4/100
92/92 [==============================] - 0s 3ms/step - loss: 140.3216 - val_loss: 136.0538
Epoch 5/100
92/92 [==============================] - 0s 3ms/step - loss: 125.7388 - val_loss: 95.8794
Epoch 6/100
92/92 [==============================] - 0s 2ms/step - loss: 118.8792 - val_loss: 94.8557
Epoch 7/100
92/92 [==============================] - 0s 3ms/step - loss: 124.4692 - val_loss: 80.9595
Epoch 8/100
92/92 [==============================] - 0s 3ms/step - loss: 115.7263 - val_loss: 84.1983
Epoch 9/100
92/92 [==============================] - 0s 3ms/step - loss: 116.0064 - val_loss: 77.0318
Epoch 10/100
92/92 [==============================] - 0s 3ms/step - loss: 114.6971 - val_loss: 75.1258
Epoch 11/100
92/92 [==============================] - 0s 3ms/step - loss: 109.5688 - val_loss: 78.4954
Epoch 12/100
92/92 [==============================] - 0s 3ms/step - loss: 112.5707 - val_loss: 76.9274
Epoch 13/100
92/92 [==============================] - 0s 3ms/step - loss: 107.3336 - val_loss: 77.2956
Epoch 14/100
92/92 [==============================] - 0s 3ms/step - loss: 109.6112 - val_loss: 77.6799
Epoch 15/100
92/92 [==============================] - 0s 4ms/step - loss: 107.3034 - val_loss: 76.3978
Epoch 16/100
92/92 [==============================] - 0s 3ms/step - loss: 106.3569 - val_loss: 70.9348
Epoch 17/100
92/92 [==============================] - 0s 4ms/step - loss: 103.3846 - val_loss: 72.3830
Epoch 18/100
92/92 [==============================] - 0s 3ms/step - loss: 100.9387 - val_loss: 71.5408
Epoch 19/100
92/92 [==============================] - 0s 4ms/step - loss: 100.7458 - val_loss: 73.5242
Epoch 20/100
92/92 [==============================] - 0s 3ms/step - loss: 102.6849 - val_loss: 68.8589
Epoch 21/100
92/92 [==============================] - 0s 3ms/step - loss: 106.8131 - val_loss: 75.3465
Epoch 22/100
92/92 [==============================] - 0s 3ms/step - loss: 97.5184 - val_loss: 74.8204
Epoch 23/100
92/92 [==============================] - 0s 4ms/step - loss: 98.8281 - val_loss: 78.4428
Epoch 24/100
92/92 [==============================] - 0s 3ms/step - loss: 101.4627 - val_loss: 70.0363
Epoch 25/100
92/92 [==============================] - 0s 3ms/step - loss: 101.1452 - val_loss: 72.5694
Epoch 26/100
92/92 [==============================] - 0s 3ms/step - loss: 102.7027 - val_loss: 71.6096
Epoch 27/100
92/92 [==============================] - 0s 3ms/step - loss: 94.9195 - val_loss: 69.0815
Epoch 28/100
92/92 [==============================] - 0s 3ms/step - loss: 95.0526 - val_loss: 71.0179
Epoch 29/100
92/92 [==============================] - 0s 3ms/step - loss: 99.2918 - val_loss: 70.6478
```

Neural Network Optimization

```
model = Sequential([
    Dense(256, input_shape=(X_train_scaled.shape[1],)),
    LeakyReLU(),
```

```
    BatchNormalization(),
    Dropout(0.3),

    Dense(128),
    LeakyReLU(),
    BatchNormalization(),
    Dropout(0.3),

    Dense(64),
    LeakyReLU(),
    Dropout(0.2),

    Dense(1)  # Regression output
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])

early_stop = EarlyStopping(monitor='val_loss', patience=45, restore_best_weights=True)
checkpoint = ModelCheckpoint('best_model.keras', monitor='val_loss', save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=30, verbose=1)

history = model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=300,
    batch_size=32,
    callbacks=[early_stop, checkpoint, reduce_lr],
    verbose=1
)

y_pred = model.predict(X_val_scaled).flatten()
rmse = mean_squared_error(y_val, y_pred, squared=False)
r2 = r2_score(y_val, y_pred)

print(f"RMSE: {rmse:.3f}")
print(f"R^2 Score: {r2:.3f}")
```

```
Epoch 1/300
92/92 [==============================] - 2s 10ms/step - loss: 530.3774 - mae: 19.7700 - val_los
Epoch 2/300
92/92 [==============================] - 1s 10ms/step - loss: 175.3035 - mae: 10.4858 - val_los
Epoch 3/300
92/92 [==============================] - 1s 8ms/step - loss: 140.5132 - mae: 9.5113 - val_loss:
Epoch 4/300
92/92 [==============================] - 1s 9ms/step - loss: 140.7338 - mae: 9.4814 - val_loss:
Epoch 5/300
92/92 [==============================] - 1s 7ms/step - loss: 132.7072 - mae: 9.2162 - val_loss:
Epoch 6/300
92/92 [==============================] - 1s 8ms/step - loss: 128.8177 - mae: 9.1021 - val_loss:
Epoch 7/300
92/92 [==============================] - 1s 8ms/step - loss: 121.9469 - mae: 8.8494 - val_loss:
Epoch 8/300
92/92 [==============================] - 1s 9ms/step - loss: 124.6049 - mae: 8.9217 - val_loss:
Epoch 9/300
92/92 [==============================] - 1s 7ms/step - loss: 118.6902 - mae: 8.7017 - val_loss:
Epoch 10/300
92/92 [==============================] - 0s 3ms/step - loss: 112.3521 - mae: 8.4645 - val_loss:
Epoch 11/300
92/92 [==============================] - 0s 3ms/step - loss: 112.8022 - mae: 8.4783 - val_loss:
Epoch 12/300
92/92 [==============================] - 1s 8ms/step - loss: 113.3292 - mae: 8.4552 - val_loss:
Epoch 13/300
92/92 [==============================] - 0s 3ms/step - loss: 111.3965 - mae: 8.4230 - val_loss:
```

```
Epoch 14/300
92/92 [==============================] - 0s 3ms/step - loss: 108.0789 - mae: 8.2943 - val_loss:
Epoch 15/300
92/92 [==============================] - 1s 8ms/step - loss: 105.6176 - mae: 8.2316 - val_loss:
Epoch 16/300
92/92 [==============================] - 1s 8ms/step - loss: 110.8678 - mae: 8.4086 - val_loss:
Epoch 17/300
92/92 [==============================] - 0s 3ms/step - loss: 107.1818 - mae: 8.3373 - val_loss:
Epoch 18/300
92/92 [==============================] - 1s 7ms/step - loss: 101.2195 - mae: 7.9956 - val_loss:
Epoch 19/300
92/92 [==============================] - 0s 3ms/step - loss: 104.5988 - mae: 8.1809 - val_loss:
Epoch 20/300
92/92 [==============================] - 0s 3ms/step - loss: 102.3594 - mae: 8.0997 - val_loss:
Epoch 21/300
92/92 [==============================] - 1s 8ms/step - loss: 100.5460 - mae: 8.0255 - val_loss:
Epoch 22/300
92/92 [==============================] - 0s 3ms/step - loss: 101.4855 - mae: 8.0166 - val_loss:
Epoch 23/300
92/92 [==============================] - 0s 3ms/step - loss: 94.7940 - mae: 7.7708 - val_loss:
Epoch 24/300
92/92 [==============================] - 0s 3ms/step - loss: 98.6884 - mae: 7.8594 - val_loss:
Epoch 25/300
92/92 [==============================] - 1s 8ms/step - loss: 97.8321 - mae: 7.9378 - val_loss:
Epoch 26/300
92/92 [==============================] - 0s 3ms/step - loss: 101.1776 - mae: 7.9950 - val_loss:
Epoch 27/300
92/92 [==============================] - 0s 4ms/step - loss: 94.0472 - mae: 7.7377 - val_loss:
Epoch 28/300
92/92 [==============================] - 0s 3ms/step - loss: 93.4816 - mae: 7.7228 - val_loss:
Epoch 29/300
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

model = Sequential([
    Dense(128, input_shape=(X_train_scaled.shape[1],)),
    LeakyReLU(),
    BatchNormalization(),
    Dropout(0.3),

    Dense(128),
    LeakyReLU(),
    Dropout(0.3),

    Dense(64),
    LeakyReLU(),
    Dropout(0.2),

    Dense(32),
    LeakyReLU(),
    Dropout(0.2),
    Dense(1)  # Regression output
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])

callbacks = [
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=30, verbose=1),
    EarlyStopping(monitor='val_loss', patience=50, restore_best_weights=True)
]
```

```
history = model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=300,
    batch_size=32,
    callbacks=callbacks,
    verbose=1
)

y_pred = model.predict(X_val_scaled).flatten()
rmse = mean_squared_error(y_val, y_pred, squared=False)
r2 = r2_score(y_val, y_pred)

print(f"RMSE: {rmse:.3f}")
print(f"R^2 Score: {r2:.3f}")
```

```
Epoch 1/300
92/92 [==============================] - 2s 7ms/step - loss: 384.1128 - mae: 15.7390 - val_loss
Epoch 2/300
92/92 [==============================] - 0s 3ms/step - loss: 160.1084 - mae: 10.0367 - val_loss
Epoch 3/300
92/92 [==============================] - 0s 3ms/step - loss: 143.0413 - mae: 9.4051 - val_loss:
Epoch 4/300
92/92 [==============================] - 0s 3ms/step - loss: 140.6411 - mae: 9.4630 - val_loss:
Epoch 5/300
92/92 [==============================] - 0s 3ms/step - loss: 141.4737 - mae: 9.3943 - val_loss:
Epoch 6/300
92/92 [==============================] - 0s 3ms/step - loss: 132.2740 - mae: 9.1397 - val_loss:
Epoch 7/300
92/92 [==============================] - 0s 3ms/step - loss: 124.7969 - mae: 8.9492 - val_loss:
Epoch 8/300
92/92 [==============================] - 0s 3ms/step - loss: 124.1982 - mae: 8.8608 - val_loss:
Epoch 9/300
92/92 [==============================] - 0s 3ms/step - loss: 116.6571 - mae: 8.6364 - val_loss:
Epoch 10/300
92/92 [==============================] - 0s 3ms/step - loss: 117.6663 - mae: 8.6351 - val_loss:
Epoch 11/300
92/92 [==============================] - 0s 3ms/step - loss: 116.3681 - mae: 8.5680 - val_loss:
Epoch 12/300
92/92 [==============================] - 0s 3ms/step - loss: 113.3407 - mae: 8.5099 - val_loss:
Epoch 13/300
92/92 [==============================] - 0s 3ms/step - loss: 115.7587 - mae: 8.6667 - val_loss:
Epoch 14/300
92/92 [==============================] - 0s 3ms/step - loss: 109.1437 - mae: 8.3560 - val_loss:
Epoch 15/300
92/92 [==============================] - 0s 3ms/step - loss: 114.0601 - mae: 8.4920 - val_loss:
Epoch 16/300
92/92 [==============================] - 0s 3ms/step - loss: 110.6792 - mae: 8.3956 - val_loss:
Epoch 17/300
92/92 [==============================] - 0s 3ms/step - loss: 112.3053 - mae: 8.4403 - val_loss:
Epoch 18/300
92/92 [==============================] - 0s 3ms/step - loss: 108.9818 - mae: 8.3062 - val_loss:
Epoch 19/300
92/92 [==============================] - 0s 3ms/step - loss: 107.2067 - mae: 8.2877 - val_loss:
Epoch 20/300
92/92 [==============================] - 0s 4ms/step - loss: 108.3658 - mae: 8.3014 - val_loss:
Epoch 21/300
92/92 [==============================] - 0s 3ms/step - loss: 108.5138 - mae: 8.3243 - val_loss:
Epoch 22/300
92/92 [==============================] - 0s 4ms/step - loss: 106.9734 - mae: 8.2459 - val_loss:
Epoch 23/300
92/92 [==============================] - 0s 4ms/step - loss: 103.1346 - mae: 8.1038 - val_loss:
Epoch 24/300
92/92 [==============================] - 0s 3ms/step - loss: 107.9905 - mae: 8.2880 - val_loss:
Epoch 25/300
```

```
92/92 [==============================] - 0s 3ms/step - loss: 107.6281 - mae: 8.2649 - val_loss:
Epoch 26/300
92/92 [==============================] - 0s 3ms/step - loss: 112.1922 - mae: 8.4105 - val_loss:
Epoch 27/300
92/92 [==============================] - 0s 3ms/step - loss: 106.2902 - mae: 8.1590 - val_loss:
Epoch 28/300
92/92 [==============================] - 0s 3ms/step - loss: 105.2960 - mae: 8.2411 - val_loss:
Epoch 29/300
92/92 [==============================] - 0s 3ms/step - loss: 102.3972 - mae: 8.0636 - val_loss:
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(128, activation='sigmoid'),
    BatchNormalization(),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1)  # Regression output
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])

callbacks = [
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=30, verbose=1),
    EarlyStopping(monitor='val_loss', patience=50, restore_best_weights=True)
]

history = model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=300,
    batch_size=32,
    callbacks=callbacks,
    verbose=1
)

y_pred = model.predict(X_val_scaled).flatten()
rmse = mean_squared_error(y_val, y_pred, squared=False)
r2 = r2_score(y_val, y_pred)

print(f"RMSE: {rmse:.3f}")
print(f"R^2 Score: {r2:.3f}")
```

```
Epoch 1/300
92/92 [==============================] - 2s 5ms/step - loss: 375.7271 - mae: 15.7430 - val_loss
Epoch 2/300
92/92 [==============================] - 0s 3ms/step - loss: 146.0475 - mae: 9.6307 - val_loss:
Epoch 3/300
92/92 [==============================] - 0s 3ms/step - loss: 136.1338 - mae: 9.2368 - val_loss:
Epoch 4/300
92/92 [==============================] - 0s 3ms/step - loss: 130.1717 - mae: 9.0484 - val_loss:
Epoch 5/300
92/92 [==============================] - 0s 3ms/step - loss: 123.6506 - mae: 8.8375 - val_loss:
Epoch 6/300
```

```
92/92 [==============================] - 0s 3ms/step - loss: 114.3619 - mae: 8.4398 - val_loss:
Epoch 7/300
92/92 [==============================] - 0s 3ms/step - loss: 110.5712 - mae: 8.2907 - val_loss:
Epoch 8/300
92/92 [==============================] - 0s 3ms/step - loss: 106.4088 - mae: 8.1271 - val_loss:
Epoch 9/300
92/92 [==============================] - 0s 3ms/step - loss: 107.0042 - mae: 8.1894 - val_loss:
Epoch 10/300
92/92 [==============================] - 0s 3ms/step - loss: 102.2078 - mae: 7.9690 - val_loss:
Epoch 11/300
92/92 [==============================] - 0s 3ms/step - loss: 100.4209 - mae: 7.8586 - val_loss:
Epoch 12/300
92/92 [==============================] - 0s 4ms/step - loss: 103.4199 - mae: 7.9450 - val_loss:
Epoch 13/300
92/92 [==============================] - 0s 4ms/step - loss: 101.6784 - mae: 7.9548 - val_loss:
Epoch 14/300
92/92 [==============================] - 0s 3ms/step - loss: 96.9804 - mae: 7.8415 - val_loss:
Epoch 15/300
92/92 [==============================] - 0s 3ms/step - loss: 97.1508 - mae: 7.7745 - val_loss:
Epoch 16/300
92/92 [==============================] - 0s 3ms/step - loss: 96.0499 - mae: 7.7262 - val_loss:
Epoch 17/300
92/92 [==============================] - 0s 3ms/step - loss: 98.7089 - mae: 7.8105 - val_loss:
Epoch 18/300
92/92 [==============================] - 0s 4ms/step - loss: 94.2651 - mae: 7.6119 - val_loss:
Epoch 19/300
92/92 [==============================] - 0s 3ms/step - loss: 92.4716 - mae: 7.5505 - val_loss:
Epoch 20/300
92/92 [==============================] - 0s 3ms/step - loss: 97.1796 - mae: 7.8021 - val_loss:
Epoch 21/300
92/92 [==============================] - 0s 3ms/step - loss: 94.0567 - mae: 7.6234 - val_loss:
Epoch 22/300
92/92 [==============================] - 0s 3ms/step - loss: 94.1686 - mae: 7.6551 - val_loss:
Epoch 23/300
92/92 [==============================] - 0s 4ms/step - loss: 93.4702 - mae: 7.5757 - val_loss:
Epoch 24/300
92/92 [==============================] - 0s 3ms/step - loss: 91.5190 - mae: 7.5221 - val_loss:
Epoch 25/300
92/92 [==============================] - 0s 3ms/step - loss: 91.8528 - mae: 7.5134 - val_loss:
Epoch 26/300
92/92 [==============================] - 0s 3ms/step - loss: 93.6638 - mae: 7.5736 - val_loss:
Epoch 27/300
92/92 [==============================] - 0s 3ms/step - loss: 91.0284 - mae: 7.4526 - val_loss:
Epoch 28/300
92/92 [==============================] - 0s 3ms/step - loss: 89.4417 - mae: 7.4618 - val_loss:
Epoch 29/300
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

model = Sequential([
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),
    Dense(128, activation='relu'),
    BatchNormalization(),
    Dropout(0.3),
    Dense(128, activation='sigmoid'),
    BatchNormalization(),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.2),
```

```
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1)  # Regression output
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])

callbacks = [
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=20, verbose=1),
    EarlyStopping(monitor='val_loss', patience=50, restore_best_weights=True)
]

history = model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=300,
    batch_size=32,
    callbacks=callbacks,
    verbose=1
)

y_pred = model.predict(X_val_scaled).flatten()
rmse = mean_squared_error(y_val, y_pred, squared=False)
r2 = r2_score(y_val, y_pred)

print(f"RMSE: {rmse:.3f}")
print(f"R^2 Score: {r2:.3f}")
```

```
Epoch 1/300
92/92 [==============================] - 2s 7ms/step - loss: 466.2141 - mae: 17.9260 - val_loss
Epoch 2/300
92/92 [==============================] - 0s 4ms/step - loss: 154.5234 - mae: 9.8692 - val_loss:
Epoch 3/300
92/92 [==============================] - 0s 4ms/step - loss: 138.4125 - mae: 9.3466 - val_loss:
Epoch 4/300
92/92 [==============================] - 0s 4ms/step - loss: 134.0977 - mae: 9.1606 - val_loss:
Epoch 5/300
92/92 [==============================] - 0s 3ms/step - loss: 122.9230 - mae: 8.8087 - val_loss:
Epoch 6/300
92/92 [==============================] - 0s 4ms/step - loss: 117.2660 - mae: 8.5729 - val_loss:
Epoch 7/300
92/92 [==============================] - 0s 4ms/step - loss: 118.3391 - mae: 8.6315 - val_loss:
Epoch 8/300
92/92 [==============================] - 0s 4ms/step - loss: 114.3142 - mae: 8.4556 - val_loss:
Epoch 9/300
92/92 [==============================] - 0s 4ms/step - loss: 112.5823 - mae: 8.3243 - val_loss:
Epoch 10/300
92/92 [==============================] - 0s 4ms/step - loss: 110.6237 - mae: 8.2625 - val_loss:
Epoch 11/300
92/92 [==============================] - 0s 4ms/step - loss: 111.0977 - mae: 8.3265 - val_loss:
Epoch 12/300
92/92 [==============================] - 0s 4ms/step - loss: 104.8734 - mae: 8.0477 - val_loss:
Epoch 13/300
92/92 [==============================] - 0s 4ms/step - loss: 108.3786 - mae: 8.2362 - val_loss:
Epoch 14/300
92/92 [==============================] - 0s 4ms/step - loss: 102.4387 - mae: 7.9263 - val_loss:
Epoch 15/300
92/92 [==============================] - 0s 4ms/step - loss: 103.8232 - mae: 7.9573 - val_loss:
Epoch 16/300
92/92 [==============================] - 0s 4ms/step - loss: 102.7403 - mae: 7.9832 - val_loss:
Epoch 17/300
92/92 [==============================] - 0s 4ms/step - loss: 104.6190 - mae: 8.0877 - val_loss:
Epoch 18/300
92/92 [==============================] - 0s 4ms/step - loss: 101.9364 - mae: 7.9172 - val_loss:
Epoch 19/300
```

```
92/92 [==============================] - 0s 4ms/step - loss: 96.4046 - mae: 7.7019 - val_loss:
Epoch 20/300
92/92 [==============================] - 0s 5ms/step - loss: 101.2491 - mae: 7.9061 - val_loss:
Epoch 21/300
92/92 [==============================] - 0s 5ms/step - loss: 101.6445 - mae: 7.8852 - val_loss:
Epoch 22/300
92/92 [==============================] - 0s 4ms/step - loss: 97.0835 - mae: 7.7146 - val_loss:
Epoch 23/300
92/92 [==============================] - 0s 4ms/step - loss: 95.9223 - mae: 7.6731 - val_loss:
Epoch 24/300
92/92 [==============================] - 0s 4ms/step - loss: 96.6472 - mae: 7.7307 - val_loss:
Epoch 25/300
92/92 [==============================] - 0s 4ms/step - loss: 102.4363 - mae: 7.9649 - val_loss:
Epoch 26/300
92/92 [==============================] - 0s 4ms/step - loss: 96.0755 - mae: 7.6760 - val_loss:
Epoch 27/300
92/92 [==============================] - 0s 4ms/step - loss: 95.7446 - mae: 7.6971 - val_loss:
Epoch 28/300
92/92 [==============================] - 0s 4ms/step - loss: 97.1487 - mae: 7.6896 - val_loss:
Epoch 29/300
92/92 [==============================] - 0s 4ms/step - loss: 92.5482 - mae: 7.5656 - val_loss:
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    BatchNormalization(),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1)  # Regression output
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])

callbacks = [
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=30, verbose=1),
    EarlyStopping(monitor='val_loss', patience=50, restore_best_weights=True)
]

history = model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=300,
    batch_size=10,
    callbacks=callbacks,
    verbose=1
)

y_pred = model.predict(X_val_scaled).flatten()
rmse = mean_squared_error(y_val, y_pred, squared=False)
r2 = r2_score(y_val, y_pred)

print(f"RMSE: {rmse:.3f}")
print(f"R^2 Score: {r2:.3f}")
```

```
Epoch 1/300
294/294 [==============================] - 2s 5ms/step - loss: 228.4837 - mae: 11.8314 - val_lo
Epoch 2/300
294/294 [==============================] - 1s 3ms/step - loss: 147.0714 - mae: 9.5899 - val_los
Epoch 3/300
294/294 [==============================] - 1s 3ms/step - loss: 129.6915 - mae: 9.0638 - val_los
Epoch 4/300
294/294 [==============================] - 1s 3ms/step - loss: 127.7317 - mae: 8.8916 - val_los
Epoch 5/300
294/294 [==============================] - 1s 3ms/step - loss: 121.9761 - mae: 8.7968 - val_los
Epoch 6/300
294/294 [==============================] - 1s 3ms/step - loss: 117.4542 - mae: 8.5670 - val_los
Epoch 7/300
294/294 [==============================] - 1s 3ms/step - loss: 117.4068 - mae: 8.5114 - val_los
Epoch 8/300
294/294 [==============================] - 1s 3ms/step - loss: 114.1644 - mae: 8.4313 - val_los
Epoch 9/300
294/294 [==============================] - 1s 3ms/step - loss: 111.3593 - mae: 8.3495 - val_los
Epoch 10/300
294/294 [==============================] - 1s 3ms/step - loss: 108.4239 - mae: 8.1761 - val_los
Epoch 11/300
294/294 [==============================] - 1s 3ms/step - loss: 112.9405 - mae: 8.3002 - val_los
Epoch 12/300
294/294 [==============================] - 1s 3ms/step - loss: 109.0575 - mae: 8.2750 - val_los
Epoch 13/300
294/294 [==============================] - 1s 3ms/step - loss: 105.6618 - mae: 8.0321 - val_los
Epoch 14/300
294/294 [==============================] - 1s 3ms/step - loss: 104.3782 - mae: 8.0377 - val_los
Epoch 15/300
294/294 [==============================] - 1s 3ms/step - loss: 108.1571 - mae: 8.2396 - val_los
Epoch 16/300
294/294 [==============================] - 1s 3ms/step - loss: 103.9547 - mae: 8.0741 - val_los
Epoch 17/300
294/294 [==============================] - 1s 3ms/step - loss: 107.7629 - mae: 8.2335 - val_los
Epoch 18/300
294/294 [==============================] - 1s 3ms/step - loss: 104.5091 - mae: 8.0246 - val_los
Epoch 19/300
294/294 [==============================] - 1s 3ms/step - loss: 99.8518 - mae: 7.9025 - val_loss
Epoch 20/300
294/294 [==============================] - 1s 3ms/step - loss: 100.0096 - mae: 7.8022 - val_los
Epoch 21/300
294/294 [==============================] - 1s 3ms/step - loss: 104.7844 - mae: 7.9963 - val_los
Epoch 22/300
294/294 [==============================] - 1s 3ms/step - loss: 101.4489 - mae: 7.9446 - val_los
Epoch 23/300
294/294 [==============================] - 1s 3ms/step - loss: 102.3550 - mae: 7.9347 - val_los
Epoch 24/300
294/294 [==============================] - 1s 3ms/step - loss: 100.2975 - mae: 7.8754 - val_los
Epoch 25/300
294/294 [==============================] - 1s 3ms/step - loss: 98.9089 - mae: 7.8274 - val_loss
Epoch 26/300
294/294 [==============================] - 1s 3ms/step - loss: 99.5994 - mae: 7.8670 - val_loss
Epoch 27/300
294/294 [==============================] - 1s 3ms/step - loss: 98.2060 - mae: 7.7499 - val_loss
Epoch 28/300
294/294 [==============================] - 1s 3ms/step - loss: 96.4957 - mae: 7.7362 - val_loss
Epoch 29/300
294/294 [==============================] - 1s 3ms/step - loss: 96.5161 - mae: 7.6792 - val_loss
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
```

```
    BatchNormalization(),
    Dropout(0.3),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1)  # Regression output
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])

callbacks = [
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=30, verbose=1),
    EarlyStopping(monitor='val_loss', patience=50, restore_best_weights=True)
]

history = model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=300,
    batch_size=32,
    callbacks=callbacks,
    verbose=1
)

y_pred = model.predict(X_val_scaled).flatten()
rmse = mean_squared_error(y_val, y_pred, squared=False)
r2 = r2_score(y_val, y_pred)

print(f"RMSE: {rmse:.3f}")
print(f"R^2 Score: {r2:.3f}")
```

```
Epoch 1/300
92/92 [==============================] - 2s 6ms/step - loss: 397.1013 - mae: 16.0452 - val_loss
Epoch 2/300
92/92 [==============================] - 0s 4ms/step - loss: 142.4626 - mae: 9.5119 - val_loss:
Epoch 3/300
92/92 [==============================] - 0s 4ms/step - loss: 125.6553 - mae: 8.9057 - val_loss:
Epoch 4/300
92/92 [==============================] - 0s 4ms/step - loss: 127.0565 - mae: 8.9111 - val_loss:
Epoch 5/300
92/92 [==============================] - 0s 3ms/step - loss: 126.8338 - mae: 8.8161 - val_loss:
Epoch 6/300
92/92 [==============================] - 0s 4ms/step - loss: 117.9635 - mae: 8.5812 - val_loss:
Epoch 7/300
92/92 [==============================] - 0s 3ms/step - loss: 115.2151 - mae: 8.5498 - val_loss:
Epoch 8/300
92/92 [==============================] - 0s 3ms/step - loss: 114.7484 - mae: 8.4548 - val_loss:
Epoch 9/300
92/92 [==============================] - 0s 4ms/step - loss: 109.2851 - mae: 8.2458 - val_loss:
Epoch 10/300
92/92 [==============================] - 0s 4ms/step - loss: 111.3360 - mae: 8.3690 - val_loss:
Epoch 11/300
92/92 [==============================] - 1s 5ms/step - loss: 106.6861 - mae: 8.1103 - val_loss:
Epoch 12/300
92/92 [==============================] - 0s 3ms/step - loss: 106.2308 - mae: 8.1711 - val_loss:
Epoch 13/300
92/92 [==============================] - 0s 4ms/step - loss: 106.6165 - mae: 8.1521 - val_loss:
Epoch 14/300
92/92 [==============================] - 0s 3ms/step - loss: 103.5234 - mae: 8.0020 - val_loss:
Epoch 15/300
92/92 [==============================] - 0s 3ms/step - loss: 101.7917 - mae: 7.9213 - val_loss:
```

```
Epoch 16/300
92/92 [==============================] - 0s 3ms/step - loss: 103.8179 - mae: 8.1038 - val_loss:
Epoch 17/300
92/92 [==============================] - 0s 3ms/step - loss: 104.8302 - mae: 8.0107 - val_loss:
Epoch 18/300
92/92 [==============================] - 0s 4ms/step - loss: 103.3243 - mae: 7.9337 - val_loss:
Epoch 19/300
92/92 [==============================] - 0s 4ms/step - loss: 93.9768 - mae: 7.6110 - val_loss:
Epoch 20/300
92/92 [==============================] - 0s 4ms/step - loss: 99.7106 - mae: 7.9183 - val_loss:
Epoch 21/300
92/92 [==============================] - 0s 4ms/step - loss: 100.7598 - mae: 7.9412 - val_loss:
Epoch 22/300
92/92 [==============================] - 0s 3ms/step - loss: 100.2587 - mae: 7.9125 - val_loss:
Epoch 23/300
92/92 [==============================] - 0s 4ms/step - loss: 94.4039 - mae: 7.6528 - val_loss:
Epoch 24/300
92/92 [==============================] - 0s 4ms/step - loss: 98.4942 - mae: 7.7847 - val_loss:
Epoch 25/300
92/92 [==============================] - 0s 4ms/step - loss: 100.0936 - mae: 7.8436 - val_loss:
Epoch 26/300
92/92 [==============================] - 0s 3ms/step - loss: 101.3727 - mae: 7.8653 - val_loss:
Epoch 27/300
92/92 [==============================] - 0s 3ms/step - loss: 94.8514 - mae: 7.6487 - val_loss:
Epoch 28/300
92/92 [==============================] - 0s 3ms/step - loss: 97.0766 - mae: 7.7561 - val_loss:
Epoch 29/300
```

Key Findings - not like 100% tho since there are a lot more combination of everything

128 - 128 -64-32 works the best

Best activation function: relu

Batch size:16

But theres probably a lot of ways to optimize this further

Best Model So Far

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau, EarlyStopping

model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train_scaled.shape[1],)),
    BatchNormalization(),
    Dropout(0.2),
    Dense(128, activation='relu'),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(32, activation='relu'),
    Dropout(0.2),
    Dense(1)  # Regression output
])
```

```
model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])

callbacks = [
    ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=30, verbose=1),
    EarlyStopping(monitor='val_loss', patience=50, restore_best_weights=True)
]

history = model.fit(
    X_train_scaled, y_train,
    validation_data=(X_val_scaled, y_val),
    epochs=300,
    batch_size=16,
    callbacks=callbacks,
    verbose=1
)

y_pred = model.predict(X_val_scaled).flatten()
rmse = mean_squared_error(y_val, y_pred, squared=False)
r2 = r2_score(y_val, y_pred)

print(f"RMSE: {rmse:.3f}")
print(f"R^2 Score: {r2:.3f}")
```

```
Epoch 1/300
184/184 [==============================] - 3s 6ms/step - loss: 277.2464 - mae: 13.1391 - val_lo
Epoch 2/300
184/184 [==============================] - 1s 4ms/step - loss: 139.8604 - mae: 9.4062 - val_los
Epoch 3/300
184/184 [==============================] - 1s 4ms/step - loss: 126.8023 - mae: 8.9479 - val_los
Epoch 4/300
184/184 [==============================] - 1s 4ms/step - loss: 116.1606 - mae: 8.5220 - val_los
Epoch 5/300
184/184 [==============================] - 1s 4ms/step - loss: 112.5897 - mae: 8.3828 - val_los
Epoch 6/300
184/184 [==============================] - 1s 4ms/step - loss: 112.2560 - mae: 8.3869 - val_los
Epoch 7/300
184/184 [==============================] - 1s 4ms/step - loss: 106.4367 - mae: 8.1189 - val_los
Epoch 8/300
184/184 [==============================] - 1s 4ms/step - loss: 109.0179 - mae: 8.2494 - val_los
Epoch 9/300
184/184 [==============================] - 1s 5ms/step - loss: 104.6643 - mae: 8.0558 - val_los
Epoch 10/300
184/184 [==============================] - 1s 7ms/step - loss: 103.1133 - mae: 7.9766 - val_los
Epoch 11/300
184/184 [==============================] - 1s 8ms/step - loss: 104.1293 - mae: 8.0418 - val_los
Epoch 12/300
184/184 [==============================] - 1s 6ms/step - loss: 104.1923 - mae: 8.0578 - val_los
Epoch 13/300
184/184 [==============================] - 1s 7ms/step - loss: 103.9447 - mae: 8.0123 - val_los
Epoch 14/300
184/184 [==============================] - 1s 5ms/step - loss: 97.2755 - mae: 7.7318 - val_loss
Epoch 15/300
184/184 [==============================] - 1s 5ms/step - loss: 96.6005 - mae: 7.7443 - val_loss
Epoch 16/300
184/184 [==============================] - 1s 7ms/step - loss: 99.0170 - mae: 7.8589 - val_loss
Epoch 17/300
184/184 [==============================] - 1s 8ms/step - loss: 99.9593 - mae: 7.8569 - val_loss
Epoch 18/300
184/184 [==============================] - 1s 8ms/step - loss: 99.4562 - mae: 7.8219 - val_loss
Epoch 19/300
184/184 [==============================] - 2s 10ms/step - loss: 98.1293 - mae: 7.7576 - val_los
Epoch 20/300
184/184 [==============================] - 1s 7ms/step - loss: 97.9583 - mae: 7.8055 - val_loss
Epoch 21/300
```

```
184/184 [==============================] - 2s 8ms/step - loss: 99.4280 - mae: 7.8616 - val_loss
Epoch 22/300
184/184 [==============================] - 2s 9ms/step - loss: 95.4526 - mae: 7.6891 - val_loss
Epoch 23/300
184/184 [==============================] - 1s 6ms/step - loss: 97.6467 - mae: 7.7663 - val_loss
Epoch 24/300
184/184 [==============================] - 1s 7ms/step - loss: 93.1341 - mae: 7.5593 - val_loss
Epoch 25/300
184/184 [==============================] - 1s 6ms/step - loss: 92.9692 - mae: 7.5925 - val_loss
Epoch 26/300
184/184 [==============================] - 1s 7ms/step - loss: 93.0909 - mae: 7.5180 - val_loss
Epoch 27/300
184/184 [==============================] - 1s 6ms/step - loss: 91.2219 - mae: 7.4616 - val_loss
Epoch 28/300
184/184 [==============================] - 2s 10ms/step - loss: 93.1689 - mae: 7.5991 - val_los
Epoch 29/300
```

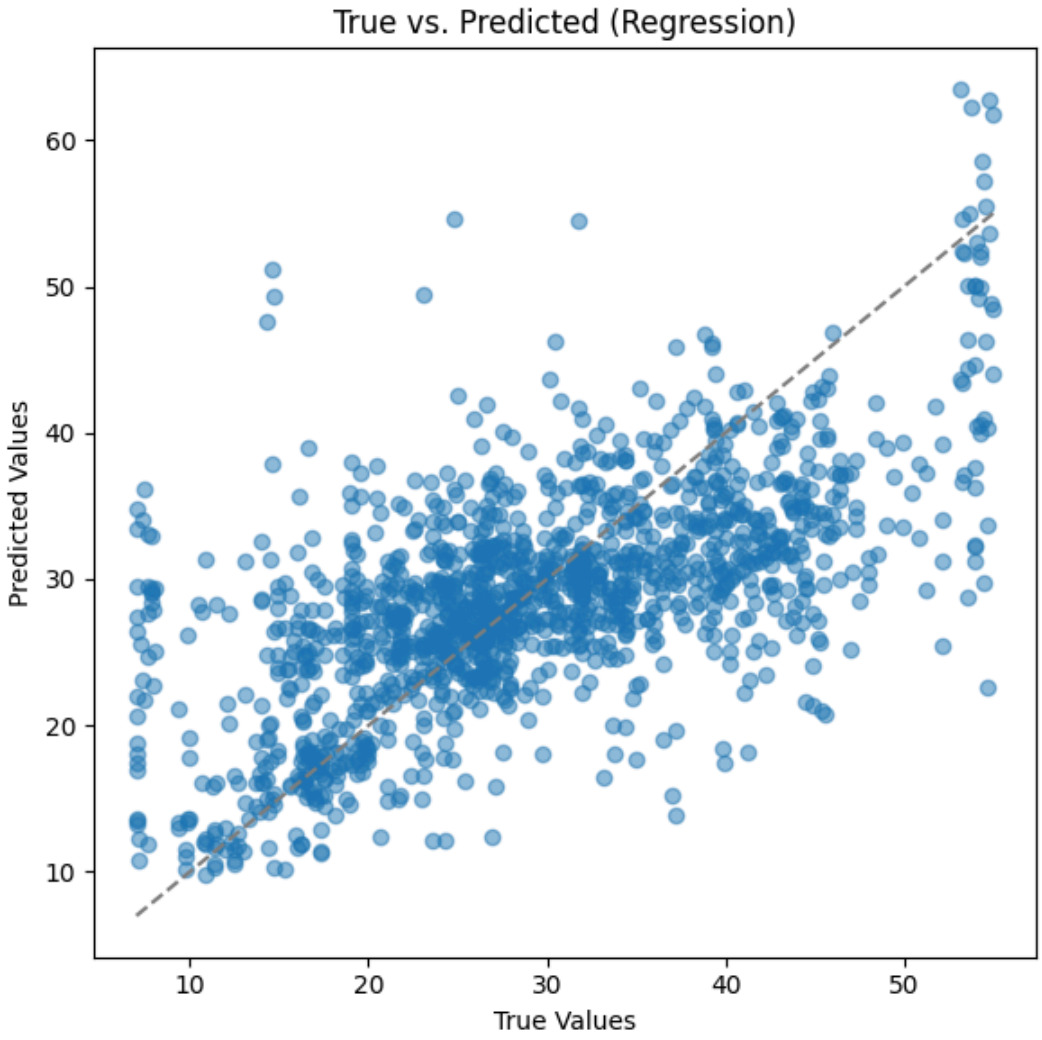
```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, explained_varianc

# After your existing fit and predict:
y_pred = model.predict(X_val_scaled).flatten()

# Compute metrics
rmse = mean_squared_error(y_val, y_pred, squared=False)
mae  = mean_absolute_error(y_val, y_pred)
r2   = r2_score(y_val, y_pred)
ev   = explained_variance_score(y_val, y_pred)

print(f"RMSE:              {rmse:.3f}")
print(f"MAE:               {mae:.3f}")
print(f"R² Score:          {r2:.3f}")
print(f"Explained Variance: {ev:.3f}")
```

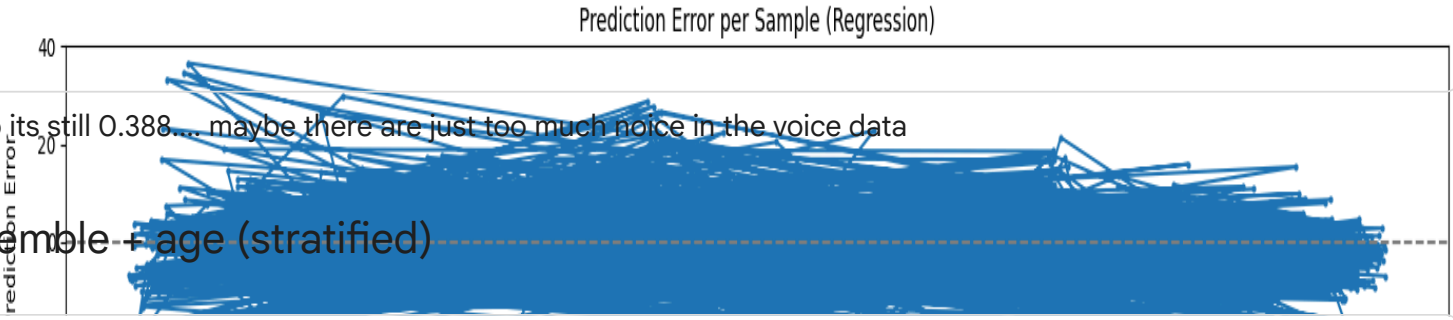
```
46/46 [==============================] - 0s 2ms/step
RMSE:              7.677
MAE:               5.836
R² Score:          0.463
Explained Variance: 0.466
```

```
X_test_scaled = scaler.transform(X_test)

y_test_pred = model.predict(X_test_scaled).argmax(axis=1)

from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error

# Predict (no argmax)
y_test_pred = model.predict(X_test_scaled).flatten()

# Evaluation
print("RMSE:", mean_squared_error(y_test, y_test_pred, squared=False))
print("MAE:", mean_absolute_error(y_test, y_test_pred))
print("R² Score:", r2_score(y_test, y_test_pred))

plt.figure(figsize=(6, 6))
plt.scatter(y_test, y_test_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], '--', color='gray')
plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.title("True vs. Predicted (Regression)")
plt.tight_layout()
plt.show()
```

```
errors = y_test_pred - y_test

plt.figure(figsize=(14, 3))
plt.plot(errors, marker='o', linestyle='-', markersize=2)
plt.axhline(0, color='gray', linestyle='--')
plt.title("Prediction Error per Sample (Regression)")
plt.xlabel("Sample Index")
plt.ylabel("Prediction Error")
plt.tight_layout()
plt.show()

# Compute prediction error
errors = y_test_pred - y_test
abs_errors = np.abs(errors)

# Find the index of the largest error
outlier_idx = np.argmax(abs_errors)

# Remove the outlier from predictions and ground truth
y_test_no_outlier = np.delete(y_test, outlier_idx)
y_pred_no_outlier = np.delete(y_test_pred, outlier_idx)
errors_no_outlier = y_pred_no_outlier - y_test_no_outlier

plt.figure(figsize=(6, 6))
plt.scatter(y_test_no_outlier, y_pred_no_outlier, alpha=0.6)
plt.plot([y_test_no_outlier.min(), y_test_no_outlier.max()],
         [y_test_no_outlier.min(), y_test_no_outlier.max()],
         'k--', lw=2)
plt.xlabel("True Values")
plt.ylabel("Predicted Values")
plt.title("True vs. Predicted (Regression, Outlier Removed)")
plt.tight_layout()
plt.grid(True)
plt.show()

plt.figure(figsize=(12, 3.5))
plt.plot(errors_no_outlier, linestyle='-', marker='o', markersize=2, alpha=0.8)
plt.axhline(0, color='gray', linestyle='--')
plt.title("Prediction Error per Sample (Outlier Removed)")
plt.xlabel("Sample Index")
plt.ylabel("Prediction Error")
plt.tight_layout()
plt.grid(True)
plt.show()
```

```
46/46 [==============================] - 0s 912us/step
46/46 [==============================] - 0s 1ms/step
RMSE: 8.213510985950949
MAE: 6.076341098191772
R² Score: 0.4063832122969444
```



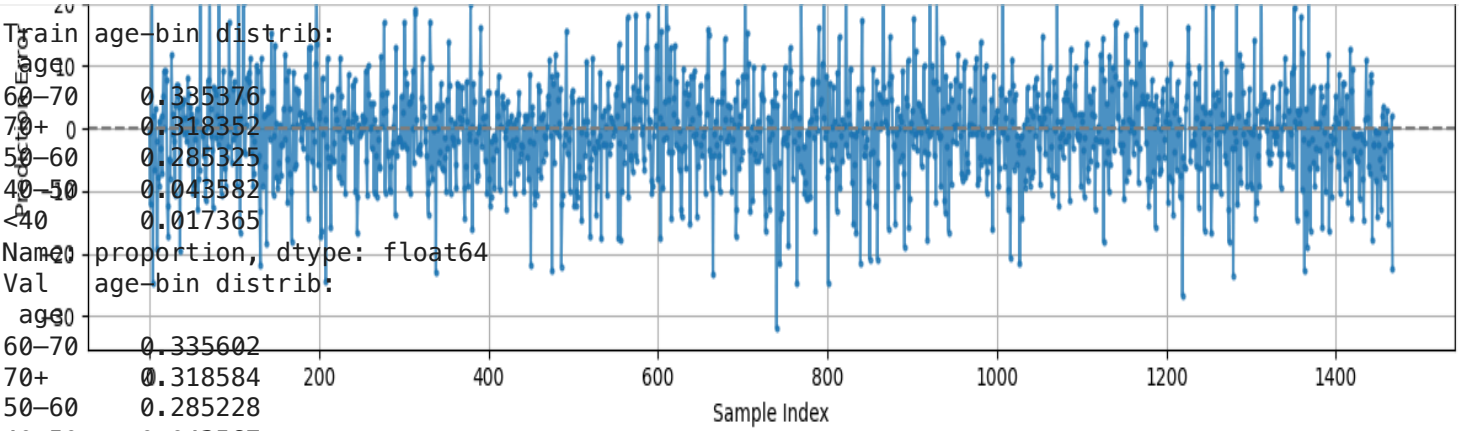


Ok so its still 0.388.... maybe there are just too much noice in the voice data

Ensemble + age (stratified)

```
seed = 42
np.random.seed(seed)
tf.random.set_seed(seed)
parkinsons_voice = fetch_ucirepo(id=189)


age = parkinsons_voice.data.features['age']
X  = parkinsons_voice.data.features.copy()
bins = [age.min() - 1, 40, 50, 60, 70, age.max()]
labels = ['<40', '40–50', '50–60', '60–70', '70+']
age_cat = pd.cut(age, bins=bins, labels=labels)
features_to_drop = [
    'Jitter(%)', 'Jitter:RAP', 'Jitter:DDP',
    'Shimmer(dB)', 'Shimmer:APQ5', 'Shimmer:DDA',
    'Shimmer', 'NHR',
```

```
    'test_time', 'subject#'
]

X = X.drop(columns=[col for col in features_to_drop if col in X.columns])
y = parkinsons_voice.data.targets['total_UPDRS']

X_train, X_temp, y_train, y_temp, age_train, age_temp = train_test_split(
    X, y, age_cat,
    test_size=0.5,         # leaves 50% for temp
    random_state=42,
    stratify=age_cat
)

# 4. Second split: split the temp half into val & test equally
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp,
    test_size=0.5,         # splits temp into 25% each
    random_state=42,
    stratify=age_temp
)

print("Train age-bin distrib:\n", age_train.value_counts(normalize=True))
print("Val   age-bin distrib:\n", age_temp.loc[X_val.index].value_counts(normalize=True))
print("Test  age-bin distrib:\n", age_temp.loc[X_test.index].value_counts(normalize=True))
```




```
Train age-bin distrib:
 age
60–70    0.335376
70+      0.318352
50–60    0.285325
40–50    0.043582
<40      0.017365
Name: proportion, dtype: float64
Val   age-bin distrib:
 age
60–70    0.335602
70+      0.318584
50–60    0.285228
40–50    0.043567
<40      0.017018
Name: proportion, dtype: float64
Test  age-bin distrib:
 age
60–70    0.335602
70+      0.318584
50–60    0.285228
40–50    0.043567
<40      0.017018
Name: proportion, dtype: float64
```

```
#Build a DataFrame with age & target (or whatever y)
df = parkinsons_voice.data.features.copy()
df['target'] = parkinsons_voice.data.targets['total_UPDRS']

#Compute count per age
age_counts = df['age'].value_counts().sort_values(ascending=False)
# This gives a Series: index=age, value=#samples

#Define your target sizes (50%,25%,25%)
total = len(df)
targets = {
    'train': total * 0.7,
```

```
    'val':   total * 0.15,
    'test':  total * 0.15,
}

#  keep track of current sum per split; sort ages by descending count
current = {k: 0 for k in targets}
assignment = {}  # age -> split

for age, count in age_counts.items():
    # compute “fill ratio” = current[split] / targets[split]
    ratios = {s: current[s] / targets[s] for s in targets}
    # pick the split with the smallest ratio (i.e. farthest below its target)
    best_split = min(ratios, key=ratios.get)
    assignment[age] = best_split
    current[best_split] += count

#filter df into X/y for each split
splits = {s: {'X': [], 'y': []} for s in targets}
for split in splits:
    ages_in_split = [age for age, s in assignment.items() if s == split]
    mask = df['age'].isin(ages_in_split)
    splits[split]['X'] = df.loc[mask].drop(columns=['age','target'])
    splits[split]['y'] = df.loc[mask,  'target']

for split in ['train','val','test']:
    print(f"{split.upper():5} | n_samples = {len(splits[split]['X']):4} | "
          f"pct = {100*len(splits[split]['X'])/total:5.1f}%")

for name, split_df in [('Train', splits['train']['X']),
                       ('Val',   splits['val']['X']),
                       ('Test',  splits['test']['X'])]:
    freqs = split_df.copy()
    freqs['age'] = df.loc[split_df.index, 'age']
    print(f"\n{name} age distribution:")
    print(freqs['age'].value_counts(normalize=True).sort_index())

ages = df['age']

Q1, Q3 = np.percentile(ages, [25,75])
IQR = Q3 - Q1
lower_bound = Q1 - 1.5*IQR
print("Lower outlier cutoff =", lower_bound)
```

```
TRAIN | n_samples = 4055 | pct =  69.0%
VAL   | n_samples =  861 | pct =  14.7%
TEST  | n_samples =  959 | pct =  16.3%

Train age distribution:
age
49    0.063132
55    0.065845
56    0.034525
57    0.094945
58    0.105795
59    0.073736
60    0.038471
61    0.036991
62    0.058200
68    0.078175
72    0.076449
73    0.094945
74    0.066831
76    0.035512
```

```
78    0.041430
85    0.035018
Name: proportion, dtype: float64

Val age distribution:
age
63    0.181185
66    0.490128
67    0.328688
Name: proportion, dtype: float64

Test age distribution:
age
36    0.105318
65    0.424400
71    0.172054
75    0.298227
Name: proportion, dtype: float64
Lower outlier cutoff = 37.0
```

```
import numpy as np
import pandas as pd
from sklearn.experimental import enable_halving_search_cv  # noqa
from sklearn.model_selection import PredefinedSplit, HalvingRandomSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error

# 1) Glue train+val and build PredefinedSplit
X_tv = pd.concat([splits["train"]["X"], splits["val"]["X"]], axis=0)
y_tv = pd.concat([splits["train"]["y"], splits["val"]["y"]], axis=0)

folds = np.array(
    [-1] * len(splits["train"]["X"]) +
    [ 0] * len(splits["val"]["X"])
)
ps = PredefinedSplit(test_fold=folds)

# 2) Define RF search space (exclude n_estimators)
rf_param_dist = {
    "max_depth":         [None] + list(range(3, 31, 3)),
    "min_samples_split": list(range(2, 21)),
    "min_samples_leaf":  list(range(1, 11)),
    "max_features":      [1.0, "sqrt", "log2"],
}

# 3) HalvingRandomSearchCV over n_estimators
halving_rf = HalvingRandomSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_distributions=rf_param_dist,
    resource="n_estimators",   # budget parameter
    min_resources=50,          # start with 50 trees
    max_resources=1500,        # up to 1500 trees
    factor=3,                  # keep top third each round
    cv=ps,
    scoring="neg_mean_squared_error",
    random_state=42,
    n_jobs=1,
    verbose=2
)

# 4) Run the search
halving_rf.fit(X_tv, y_tv)
```

```
print("Best RF params (excluding n_estimators):",
      {k: v for k, v in halving_rf.best_params_.items() if k != "n_estimators"})
print("Best n_estimators:", halving_rf.best_params_["n_estimators"])
print("Best CV MSE:", -halving_rf.best_score_)

# 5) Retrain & test
best_rf = halving_rf.best_estimator_
best_rf.set_params(n_estimators=halving_rf.best_params_["n_estimators"])
best_rf.fit(X_tv, y_tv)

X_test, y_test = splits["test"]["X"], splits["test"]["y"]
mse_test = mean_squared_error(y_test, best_rf.predict(X_test))
print("RF Test MSE:", mse_test)
```

```
n_iterations: 4
n_required_iterations: 4
n_possible_iterations: 4
min_resources_: 50
max_resources_: 1500
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 30
n_resources: 50
Fitting 1 folds for each of 30 candidates, totalling 30 fits
[CV] END max_depth=3, max_features=sqrt, min_samples_leaf=6, min_samples_split=7, n_estimators=
[CV] END max_depth=27, max_features=sqrt, min_samples_leaf=4, min_samples_split=15, n_estimator
[CV] END max_depth=27, max_features=1.0, min_samples_leaf=6, min_samples_split=3, n_estimators=
[CV] END max_depth=27, max_features=1.0, min_samples_leaf=4, min_samples_split=6, n_estimators=
[CV] END max_depth=18, max_features=sqrt, min_samples_leaf=9, min_samples_split=12, n_estimator
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=3, min_samples_split=16, n_estimator
[CV] END max_depth=30, max_features=1.0, min_samples_leaf=2, min_samples_split=17, n_estimators
[CV] END max_depth=30, max_features=log2, min_samples_leaf=10, min_samples_split=16, n_estimato
[CV] END max_depth=None, max_features=log2, min_samples_leaf=5, min_samples_split=12, n_estimat
[CV] END max_depth=27, max_features=sqrt, min_samples_leaf=1, min_samples_split=16, n_estimator
[CV] END max_depth=21, max_features=log2, min_samples_leaf=3, min_samples_split=20, n_estimator
[CV] END max_depth=27, max_features=log2, min_samples_leaf=4, min_samples_split=13, n_estimator
[CV] END max_depth=30, max_features=log2, min_samples_leaf=8, min_samples_split=20, n_estimator
[CV] END max_depth=18, max_features=1.0, min_samples_leaf=2, min_samples_split=7, n_estimators=
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=7, min_samples_split=19, n_estimator
[CV] END max_depth=15, max_features=1.0, min_samples_leaf=4, min_samples_split=14, n_estimators
[CV] END max_depth=None, max_features=1.0, min_samples_leaf=7, min_samples_split=18, n_estimato
[CV] END max_depth=6, max_features=log2, min_samples_leaf=9, min_samples_split=15, n_estimators
[CV] END max_depth=3, max_features=sqrt, min_samples_leaf=1, min_samples_split=11, n_estimators
[CV] END max_depth=12, max_features=1.0, min_samples_leaf=6, min_samples_split=18, n_estimators
[CV] END max_depth=27, max_features=log2, min_samples_leaf=6, min_samples_split=8, n_estimators
[CV] END max_depth=12, max_features=1.0, min_samples_leaf=9, min_samples_split=3, n_estimators=
[CV] END max_depth=27, max_features=1.0, min_samples_leaf=10, min_samples_split=12, n_estimator
[CV] END max_depth=24, max_features=log2, min_samples_leaf=6, min_samples_split=18, n_estimator
[CV] END max_depth=6, max_features=1.0, min_samples_leaf=3, min_samples_split=8, n_estimators=5
[CV] END max_depth=21, max_features=log2, min_samples_leaf=10, min_samples_split=16, n_estimato
[CV] END max_depth=15, max_features=log2, min_samples_leaf=9, min_samples_split=5, n_estimators
[CV] END max_depth=21, max_features=1.0, min_samples_leaf=7, min_samples_split=15, n_estimators
[CV] END max_depth=24, max_features=sqrt, min_samples_leaf=5, min_samples_split=19, n_estimator
[CV] END max_depth=15, max_features=1.0, min_samples_leaf=3, min_samples_split=18, n_estimators
----------
iter: 1
n_candidates: 10
n_resources: 150
Fitting 1 folds for each of 10 candidates, totalling 10 fits
[CV] END max_depth=27, max_features=log2, min_samples_leaf=6, min_samples_split=8, n_estimators
[CV] END max_depth=27, max_features=log2, min_samples_leaf=4, min_samples_split=13, n_estimator
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=7, min_samples_split=19, n_estimator
```

```
[CV] END max_depth=21, max_features=log2, min_samples_leaf=3, min_samples_split=20, n_estimator
[CV] END max_depth=21, max_features=log2, min_samples_leaf=10, min_samples_split=16, n_estimato
[CV] END max_depth=24, max_features=log2, min_samples_leaf=6, min_samples_split=18, n_estimator
[CV] END max_depth=30, max_features=log2, min_samples_leaf=10, min_samples_split=16, n_estimato
[CV] END max_depth=15, max_features=log2, min_samples_leaf=9, min_samples_split=5, n_estimators
[CV] END max_depth=None, max_features=log2, min_samples_leaf=5, min_samples_split=12, n_estimat
[CV] END max_depth=27, max_features=sqrt, min_samples_leaf=4, min_samples_split=15, n_estimator
```

```
import numpy as np
import pandas as pd
from sklearn.experimental import enable_halving_search_cv  # noqa
from sklearn.model_selection import PredefinedSplit, HalvingRandomSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.base import BaseEstimator, RegressorMixin

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.optimizers import Adam

# ——— 0) Load or build your DataFrame `df` with features + `age` + target column ———
# e.g. df = your_dataframe.copy()
#      df['target'] = ...

# ——— 1) Create age-stratified splits dict ——————————————————————————————
total = len(df)
age_counts = df['age'].value_counts().sort_values(ascending=False)
targets = {'train': total*0.7, 'val': total*0.15, 'test': total*0.15}
current = {k: 0 for k in targets}
assignment = {}

for age, cnt in age_counts.items():
    # pick split that’s furthest below its target proportion
    frac = {s: current[s]/targets[s] for s in targets}
    pick = min(frac, key=frac.get)
    assignment[age] = pick
    current[pick] += cnt

# build the actual X/y splits
splits = {s: {'X': None, 'y': None} for s in targets}
for s in splits:
    ages = [age for age, grp in assignment.items() if grp == s]
    mask = df['age'].isin(ages)
    splits[s]['X'] = df.loc[mask].drop(columns=['age','target'])
    splits[s]['y'] = df.loc[mask, 'target']

# ——— 2) Glue train+val and build PredefinedSplit ———————————————————————
X_tv = pd.concat([splits['train']['X'], splits['val']['X']], axis=0)
y_tv = pd.concat([splits['train']['y'], splits['val']['y']], axis=0)

# -1 for train rows, 0 for val rows
folds = np.array(
    [-1] * len(splits['train']['X']) +
    [ 0] * len(splits['val']['X'])
)
ps = PredefinedSplit(test_fold=folds)

# ——— 3) Define your 3-layer Keras regressor ————————————————————————————
class KerasRegressor3L(BaseEstimator, RegressorMixin):
    def __init__(self,
                 hidden_units_1=64,
                 hidden_units_2=32,
```

```
                 hidden_units_3=16,
                 lr=1e-3,
                 epochs=50,
                 batch_size=32,
                 verbose=0):
        self.hidden_units_1 = hidden_units_1
        self.hidden_units_2 = hidden_units_2
        self.hidden_units_3 = hidden_units_3
        self.lr = lr
        self.epochs = epochs
        self.batch_size = batch_size
        self.verbose = verbose
        self.model_ = None

    def build_model(self, input_dim):
        m = Sequential([
            Input(shape=(input_dim,)),
            Dense(self.hidden_units_1, activation='relu'),
            Dense(self.hidden_units_2, activation='relu'),
            Dense(self.hidden_units_3, activation='relu'),
            Dense(1)
        ])
        m.compile(optimizer=Adam(self.lr), loss='mse')
        return m

    def fit(self, X, y):
        X, y = np.asarray(X), np.asarray(y)
        self.model_ = self.build_model(X.shape[1])
        self.model_.fit(X, y,
                        epochs=self.epochs,
                        batch_size=self.batch_size,
                        verbose=self.verbose)
        return self

    def predict(self, X):
        return self.model_.predict(np.asarray(X), verbose=0).ravel()

    def score(self, X, y):
        preds = self.predict(X)
        return -mean_squared_error(np.asarray(y), preds)

# ─── 4) Set up HalvingRandomSearchCV over epochs ──────────────────────────
nn_param_dist = {
    "hidden_units_1": [32, 64, 128],
    "hidden_units_2": [16, 32, 64],
    "hidden_units_3": [16, 32, 64],
    "lr":             [1e-4, 1e-3, 1e-2],
    "batch_size":     [16, 32, 64],
}

halving_nn = HalvingRandomSearchCV(
    estimator=KerasRegressor3L(verbose=0),
    param_distributions=nn_param_dist,
    resource="epochs",
    min_resources=20,
    max_resources=200,
    factor=3,
    cv=ps,
    scoring="neg_mean_squared_error",
    random_state=42,
    n_jobs=1,
```

```
    verbose=2
)

# ─── 5) Run the hyperparameter search ──────────────────────────────────────
halving_nn.fit(X_tv, y_tv)

print("Best params (excl. epochs):",
      {k:v for k,v in halving_nn.best_params_.items() if k!="epochs"})
print("Best epochs:", halving_nn.best_params_["epochs"])
print("Best CV MSE:", -halving_nn.best_score_)

# ─── 6) Retrain best model & test ──────────────────────────────────────────
best_nn = halving_nn.best_estimator_
best_nn.set_params(epochs=halving_nn.best_params_["epochs"])
best_nn.fit(X_tv, y_tv)

X_test, y_test = splits['test']['X'], splits['test']['y']
print("NN Test MSE:", mean_squared_error(y_test, best_nn.predict(X_test)))
```

```
n_iterations: 3
n_required_iterations: 3
n_possible_iterations: 3
min_resources_: 20
max_resources_: 200
aggressive_elimination: False
factor: 3
----------
iter: 0
n_candidates: 10
n_resources: 20
Fitting 1 folds for each of 10 candidates, totalling 10 fits
[CV] END batch_size=16, epochs=20, hidden_units_1=32, hidden_units_2=64, hidden_units_3=64, lr=0.
[CV] END batch_size=16, epochs=20, hidden_units_1=32, hidden_units_2=16, hidden_units_3=64, lr=0.
[CV] END batch_size=32, epochs=20, hidden_units_1=128, hidden_units_2=32, hidden_units_3=64, lr=0
[CV] END batch_size=64, epochs=20, hidden_units_1=128, hidden_units_2=32, hidden_units_3=64, lr=0
[CV] END batch_size=64, epochs=20, hidden_units_1=128, hidden_units_2=64, hidden_units_3=32, lr=0
[CV] END batch_size=64, epochs=20, hidden_units_1=32, hidden_units_2=32, hidden_units_3=32, lr=0.
[CV] END batch_size=64, epochs=20, hidden_units_1=64, hidden_units_2=64, hidden_units_3=64, lr=0.
[CV] END batch_size=32, epochs=20, hidden_units_1=128, hidden_units_2=32, hidden_units_3=64, lr=0
[CV] END batch_size=16, epochs=20, hidden_units_1=32, hidden_units_2=32, hidden_units_3=16, lr=0.
[CV] END batch_size=32, epochs=20, hidden_units_1=64, hidden_units_2=16, hidden_units_3=32, lr=0.
----------
iter: 1
n_candidates: 4
n_resources: 60
Fitting 1 folds for each of 4 candidates, totalling 4 fits
[CV] END batch_size=32, epochs=60, hidden_units_1=64, hidden_units_2=16, hidden_units_3=32, lr=0.
[CV] END batch_size=32, epochs=60, hidden_units_1=128, hidden_units_2=32, hidden_units_3=64, lr=0
[CV] END batch_size=64, epochs=60, hidden_units_1=64, hidden_units_2=64, hidden_units_3=64, lr=0.
[CV] END batch_size=64, epochs=60, hidden_units_1=128, hidden_units_2=64, hidden_units_3=32, lr=0
----------
iter: 2
n_candidates: 2
n_resources: 180
Fitting 1 folds for each of 2 candidates, totalling 2 fits
[CV] END batch_size=64, epochs=180, hidden_units_1=64, hidden_units_2=64, hidden_units_3=64, lr=0
[CV] END batch_size=32, epochs=180, hidden_units_1=64, hidden_units_2=16, hidden_units_3=32, lr=0
Best params (excl. epochs): {'lr': 0.01, 'hidden_units_3': 32, 'hidden_units_2': 16, 'hidden_unit
Best epochs: 180
Best CV MSE: 84.73360653507366
NN Test MSE: 302.36672016657633
```

```
from sklearn.ensemble import VotingRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

# --- assume best_rf, best_nn, X_tv, y_tv, splits are already defined and fitted ---

# 1) Build and fit the Voting ensemble
ensemble = VotingRegressor([
    ("rf", best_rf),
    ("nn", best_nn)
])
ensemble.fit(X_tv, y_tv)

# 2) Evaluate on the held-out test set
X_test, y_test = splits["test"]["X"], splits["test"]["y"]
preds = ensemble.predict(X_test)

mse  = mean_squared_error(y_test, preds)
rmse = np.sqrt(mse)
mae  = mean_absolute_error(y_test, preds)
r2   = r2_score(y_test, preds)

print("Voting Ensemble performance on test set:")
print(f"  RMSE = {rmse:.3f}")
print(f"  MAE  = {mae:.3f}")
print(f"  R²   = {r2:.3f}")
```

```
Voting Ensemble performance on test set:
  RMSE = 17.620
  MAE  = 15.576
  R²   = -0.204
```

Very Bad - Will not be used

Converting into a classification task

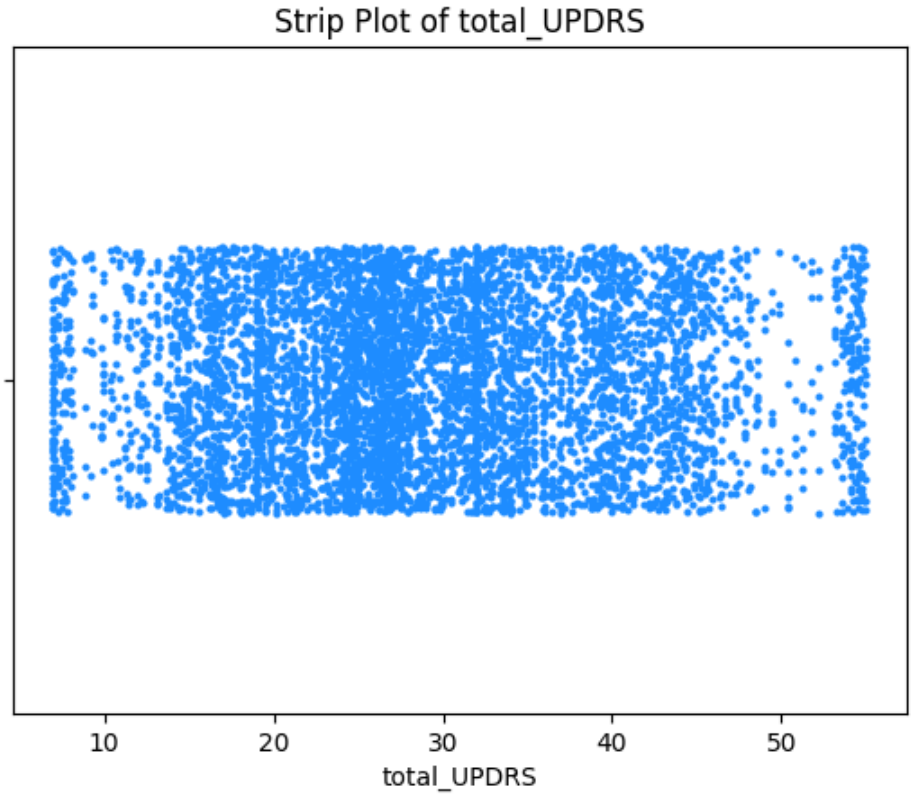
Classifying using total UPDRS

```
y_target = y["total_UPDRS"]
#y_target = y["motor_UPDRS"]

X_train, X_temp, y_train, y_temp = train_test_split(
    X_filtered, y_target, test_size=0.5, random_state=seed
)

X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, random_state=seed
)
```

```
print(y_target.describe())
print(y_target.unique())
print(y_target.head(10))
sns.stripplot(x=y_target, color='dodgerblue', size=3, jitter=0.2)
plt.title("Strip Plot of total_UPDRS")
```

```
plt.xlabel("total_UPDRS")
plt.show()
```

```
count    5875.000000
mean       29.018942
std        10.700283
min         7.000000
25%        21.371000
50%        27.576000
75%        36.399000
max        54.992000
Name: total_UPDRS, dtype: float64
[34.398 34.894 35.389 ... 35.863 36.961 35.401]
0    34.398
1    34.894
2    35.389
3    35.810
4    36.375
5    36.870
6    37.363
7    37.857
8    38.353
9    38.849
Name: total_UPDRS, dtype: float64
```

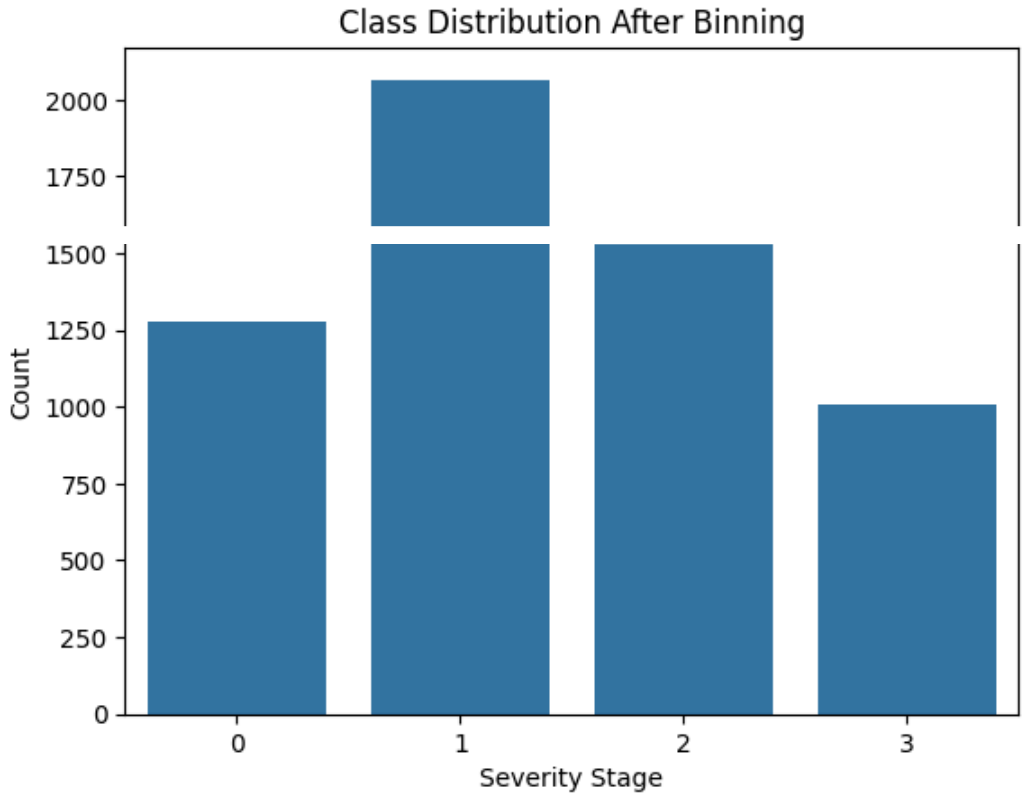


Double-click (or enter) to edit

4-way split

```
#4-way Split
bins = [0, 20, 30, 40, float('inf')]
labels = [0, 1, 2, 3]
y_classify = pd.cut(y_target, bins=bins, labels=labels).astype(int)

### this is just a repeat of making traing/val of X and y
X_train, X_temp, y_train, y_temp = train_test_split(X_filtered, y_classify, test_size=0.5, strat
```

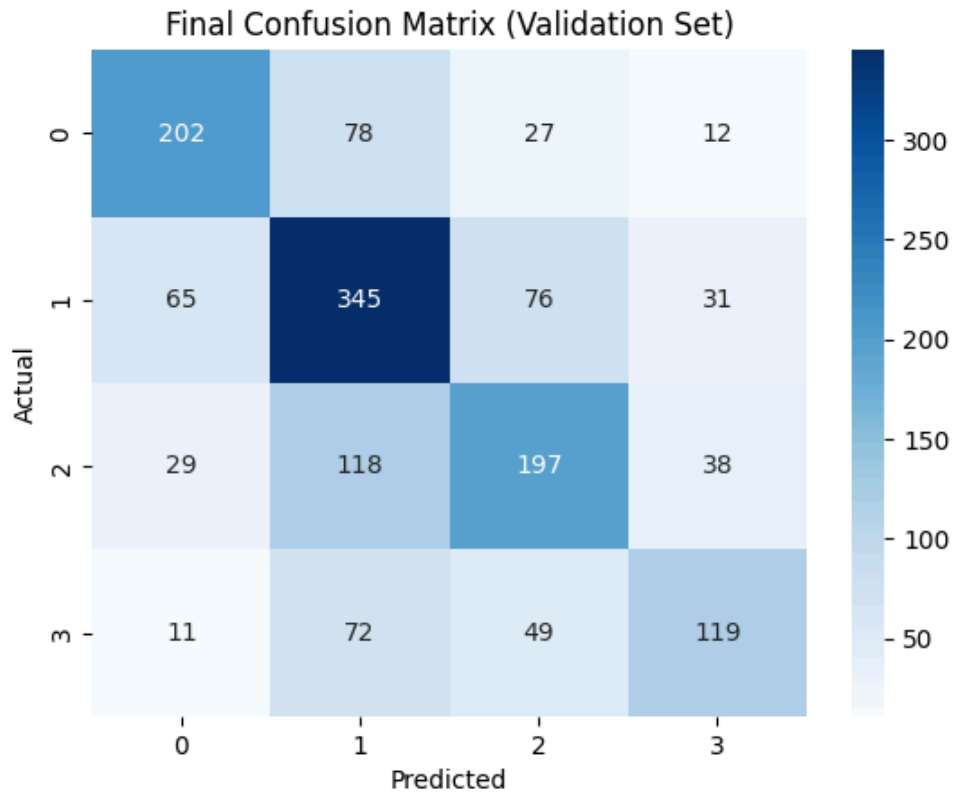
```
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, stratify=y_temp,

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
#making categorical y
y_train_cat = to_categorical(y_train, num_classes=4)
y_val_cat = to_categorical(y_val, num_classes=4)
###

sns.countplot(x=y_classify)
plt.title("Class Distribution After Binning")
plt.xlabel("Severity Stage")
plt.ylabel("Count")
plt.show()
```




```
# Build the final model with best grid parameters
final_model = RandomForestClassifier(
    n_estimators=700,
    max_depth=20,
    max_features='log2',
    min_samples_leaf=2,
    min_samples_split=2,
    class_weight='balanced',
    random_state=42
)

final_model.fit(X_train, y_train)

y_val_pred = final_model.predict(X_val)

print("Final Validation Classification Report:")
print(classification_report(y_val, y_val_pred))

cm = confusion_matrix(y_val, y_val_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
```

```
plt.title("Final Confusion Matrix (Validation Set)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Final Validation Classification Report:
              precision    recall  f1-score   support

           0       0.66      0.63      0.65       319
           1       0.56      0.67      0.61       517
           2       0.56      0.52      0.54       382
           3       0.59      0.47      0.53       251

    accuracy                           0.59      1469
   macro avg       0.60      0.57      0.58      1469
weighted avg       0.59      0.59      0.59      1469
```




```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, LeakyReLU
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam

# Build model
model = Sequential([
    Dense(256, input_shape=(X_train_scaled.shape[1],)),
    LeakyReLU(alpha=0.01),
    BatchNormalization(),
    Dropout(0.4),

    Dense(128),
    LeakyReLU(alpha=0.01),
    BatchNormalization(),
    Dropout(0.3),

    Dense(64),
    LeakyReLU(alpha=0.01),
    BatchNormalization(),
```

```
    Dropout(0.2),

    Dense(32),
    LeakyReLU(alpha=0.01),

    Dense(4, activation='softmax')
])

# Compile model
model.compile(
    optimizer=Adam(learning_rate=0.001),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Callbacks
early_stop = EarlyStopping(monitor='val_loss', patience=30, restore_best_weights=True, verbose=1
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=20, verbose=1)

# Train
history = model.fit(
    X_train_scaled, y_train_cat,
    validation_data=(X_val_scaled, y_val_cat),
    epochs=200,
    batch_size=32,
    callbacks=[early_stop, reduce_lr],
    verbose=1
)

y_val_pred = model.predict(X_val_scaled)
y_val_pred_classes = tf.argmax(y_val_pred, axis=1)

print("Keras MLP Classification Report:")
print(classification_report(y_val, y_val_pred_classes))

cm = confusion_matrix(y_val, y_val_pred_classes)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Final Confusion Matrix (Validation Set)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Epoch 1/200
92/92 [==============================] - 2s 8ms/step - loss: 1.4784 - accuracy: 0.3214 - val_loss
Epoch 2/200
92/92 [==============================] - 0s 4ms/step - loss: 1.2876 - accuracy: 0.4195 - val_loss
Epoch 3/200
92/92 [==============================] - 0s 4ms/step - loss: 1.2407 - accuracy: 0.4474 - val_loss
Epoch 4/200
92/92 [==============================] - 0s 4ms/step - loss: 1.1843 - accuracy: 0.4678 - val_loss
Epoch 5/200
92/92 [==============================] - 0s 4ms/step - loss: 1.1647 - accuracy: 0.4668 - val_loss
Epoch 6/200
92/92 [==============================] - 0s 4ms/step - loss: 1.1554 - accuracy: 0.4866 - val_loss
Epoch 7/200
92/92 [==============================] - 0s 4ms/step - loss: 1.1202 - accuracy: 0.5056 - val_loss
Epoch 8/200
92/92 [==============================] - 0s 4ms/step - loss: 1.1304 - accuracy: 0.5049 - val_loss
Epoch 9/200
92/92 [==============================] - 0s 4ms/step - loss: 1.1085 - accuracy: 0.5131 - val_loss
Epoch 10/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0897 - accuracy: 0.5182 - val_loss
Epoch 11/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0884 - accuracy: 0.5121 - val_loss
Epoch 12/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0734 - accuracy: 0.5264 - val_loss
Epoch 13/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0933 - accuracy: 0.5233 - val_loss
Epoch 14/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0539 - accuracy: 0.5380 - val_loss
Epoch 15/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0541 - accuracy: 0.5349 - val_loss
Epoch 16/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0501 - accuracy: 0.5441 - val_loss
Epoch 17/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0363 - accuracy: 0.5529 - val_loss
Epoch 18/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0436 - accuracy: 0.5407 - val_loss
Epoch 19/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0601 - accuracy: 0.5349 - val_loss
Epoch 20/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0324 - accuracy: 0.5506 - val_loss
Epoch 21/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0190 - accuracy: 0.5499 - val_loss
Epoch 22/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0092 - accuracy: 0.5587 - val_loss
Epoch 23/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0186 - accuracy: 0.5601 - val_loss
Epoch 24/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0130 - accuracy: 0.5618 - val_loss
Epoch 25/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0089 - accuracy: 0.5533 - val_loss
Epoch 26/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0102 - accuracy: 0.5662 - val_loss
Epoch 27/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0018 - accuracy: 0.5594 - val_loss
Epoch 28/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9890 - accuracy: 0.5686 - val_loss
Epoch 29/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9801 - accuracy: 0.5781 - val_loss
Epoch 30/200
92/92 [==============================] - 0s 4ms/step - loss: 1.0005 - accuracy: 0.5577 - val_loss
Epoch 31/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9946 - accuracy: 0.5720 - val_loss
Epoch 32/200
92/92 [==============================] - 0s 5ms/step - loss: 0.9949 - accuracy: 0.5700 - val_loss
Epoch 33/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9884 - accuracy: 0.5747 - val_loss
Epoch 34/200
```

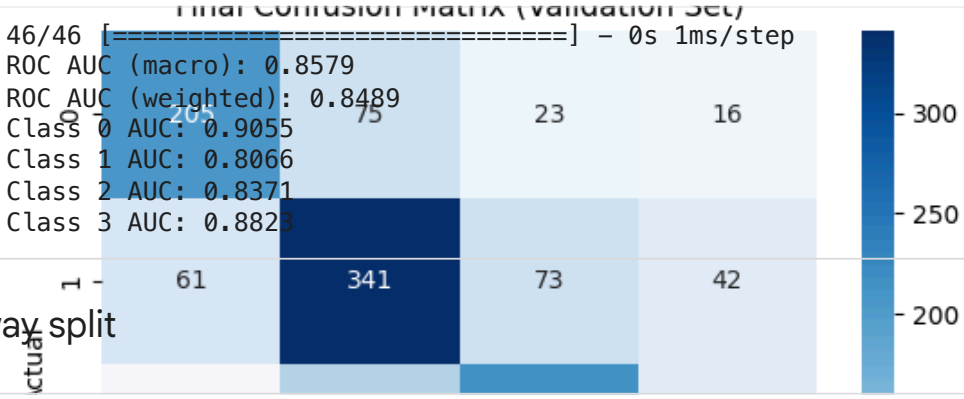
```
92/92 [==============================] - 0s 4ms/step - loss: 0.9762 - accuracy: 0.5904 - val_loss
Epoch 35/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9860 - accuracy: 0.5666 - val_loss
Epoch 36/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9576 - accuracy: 0.5873 - val_loss
Epoch 37/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9752 - accuracy: 0.5724 - val_loss
Epoch 38/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9581 - accuracy: 0.5884 - val_loss
Epoch 39/200
92/92 [==============================] - 0s 5ms/step - loss: 0.9605 - accuracy: 0.5870 - val_loss
Epoch 40/200
92/92 [==============================] - 1s 6ms/step - loss: 0.9555 - accuracy: 0.5935 - val_loss
Epoch 41/200
92/92 [==============================] - 1s 10ms/step - loss: 0.9725 - accuracy: 0.5822 - val_los
Epoch 42/200
92/92 [==============================] - 1s 8ms/step - loss: 0.9575 - accuracy: 0.5890 - val_loss
Epoch 43/200
92/92 [==============================] - 1s 10ms/step - loss: 0.9560 - accuracy: 0.6033 - val_los
Epoch 44/200
92/92 [==============================] - 1s 12ms/step - loss: 0.9457 - accuracy: 0.5993 - val_los
Epoch 45/200
92/92 [==============================] - 1s 10ms/step - loss: 0.9521 - accuracy: 0.5901 - val_los
Epoch 46/200
92/92 [==============================] - 1s 11ms/step - loss: 0.9671 - accuracy: 0.5802 - val_los
Epoch 47/200
92/92 [==============================] - 1s 9ms/step - loss: 0.9461 - accuracy: 0.5996 - val_loss
Epoch 48/200
92/92 [==============================] - 1s 14ms/step - loss: 0.9543 - accuracy: 0.5955 - val_los
Epoch 49/200
92/92 [==============================] - 1s 10ms/step - loss: 0.9356 - accuracy: 0.6013 - val_los
Epoch 50/200
92/92 [==============================] - 1s 9ms/step - loss: 0.9444 - accuracy: 0.5962 - val_loss
Epoch 51/200
92/92 [==============================] - 1s 11ms/step - loss: 0.9437 - accuracy: 0.5935 - val_los
Epoch 52/200
92/92 [==============================] - 1s 7ms/step - loss: 0.9444 - accuracy: 0.5921 - val_loss
Epoch 53/200
92/92 [==============================] - 1s 9ms/step - loss: 0.9418 - accuracy: 0.5989 - val_loss
Epoch 54/200
92/92 [==============================] - 1s 11ms/step - loss: 0.9235 - accuracy: 0.6071 - val_los
Epoch 55/200
92/92 [==============================] - 1s 6ms/step - loss: 0.9365 - accuracy: 0.6020 - val_loss
Epoch 56/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9096 - accuracy: 0.6170 - val_loss
Epoch 57/200
92/92 [==============================] - 0s 5ms/step - loss: 0.9108 - accuracy: 0.6118 - val_loss
Epoch 58/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9089 - accuracy: 0.6088 - val_loss
Epoch 59/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9130 - accuracy: 0.6047 - val_loss
Epoch 60/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9160 - accuracy: 0.6067 - val_loss
Epoch 61/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9160 - accuracy: 0.6023 - val_loss
Epoch 62/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9126 - accuracy: 0.6156 - val_loss
Epoch 63/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9167 - accuracy: 0.6105 - val_loss
Epoch 64/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9221 - accuracy: 0.6153 - val_loss
Epoch 65/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9131 - accuracy: 0.6142 - val_loss
Epoch 66/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9031 - accuracy: 0.6200 - val_loss
Epoch 67/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9079 - accuracy: 0.6166 - val_loss
Epoch 68/200
```

```
92/92 [==============================] - 0s 5ms/step - loss: 0.8906 - accuracy: 0.6146 - val_loss
Epoch 69/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9055 - accuracy: 0.6071 - val_loss
Epoch 70/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8817 - accuracy: 0.6313 - val_loss
Epoch 71/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9191 - accuracy: 0.6057 - val_loss
Epoch 72/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8965 - accuracy: 0.6166 - val_loss
Epoch 73/200
92/92 [==============================] - 0s 4ms/step - loss: 0.9016 - accuracy: 0.6207 - val_loss
Epoch 74/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8752 - accuracy: 0.6306 - val_loss
Epoch 75/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8905 - accuracy: 0.6214 - val_loss
Epoch 76/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8865 - accuracy: 0.6204 - val_loss
Epoch 77/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8984 - accuracy: 0.6163 - val_loss
Epoch 78/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8885 - accuracy: 0.6193 - val_loss
Epoch 79/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8874 - accuracy: 0.6207 - val_loss
Epoch 80/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8816 - accuracy: 0.6241 - val_loss
Epoch 81/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8680 - accuracy: 0.6272 - val_loss
Epoch 82/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8755 - accuracy: 0.6193 - val_loss
Epoch 83/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8885 - accuracy: 0.6255 - val_loss
Epoch 84/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8806 - accuracy: 0.6275 - val_loss
Epoch 85/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8853 - accuracy: 0.6296 - val_loss
Epoch 86/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8625 - accuracy: 0.6377 - val_loss
Epoch 87/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8828 - accuracy: 0.6217 - val_loss
Epoch 88/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8668 - accuracy: 0.6326 - val_loss
Epoch 89/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8763 - accuracy: 0.6272 - val_loss
Epoch 90/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8788 - accuracy: 0.6248 - val_loss
Epoch 91/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8650 - accuracy: 0.6306 - val_loss
Epoch 92/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8635 - accuracy: 0.6316 - val_loss
Epoch 93/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8651 - accuracy: 0.6289 - val_loss
Epoch 94/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8737 - accuracy: 0.6313 - val_loss
Epoch 95/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8742 - accuracy: 0.6241 - val_loss
Epoch 96/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8562 - accuracy: 0.6330 - val_loss
Epoch 97/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8571 - accuracy: 0.6279 - val_loss
Epoch 98/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8691 - accuracy: 0.6207 - val_loss
Epoch 99/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8645 - accuracy: 0.6326 - val_loss
Epoch 100/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8509 - accuracy: 0.6411 - val_loss
Epoch 101/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8571 - accuracy: 0.6374 - val_loss
Epoch 102/200
```

```
Epoch 102/200
86/92 [===========================>..] - ETA: 0s - loss: 0.8620 - accuracy: 0.6344
Epoch 102: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
92/92 [==============================] - 0s 4ms/step - loss: 0.8613 - accuracy: 0.6357 - val_loss
Epoch 103/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8428 - accuracy: 0.6357 - val_loss
Epoch 104/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8526 - accuracy: 0.6360 - val_loss
Epoch 105/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8237 - accuracy: 0.6452 - val_loss
Epoch 106/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8199 - accuracy: 0.6524 - val_loss
Epoch 107/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8264 - accuracy: 0.6473 - val_loss
Epoch 108/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8226 - accuracy: 0.6592 - val_loss
Epoch 109/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8258 - accuracy: 0.6452 - val_loss
Epoch 110/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8282 - accuracy: 0.6476 - val_loss
Epoch 111/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8241 - accuracy: 0.6442 - val_loss
Epoch 112/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8268 - accuracy: 0.6496 - val_loss
Epoch 113/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8306 - accuracy: 0.6527 - val_loss
Epoch 114/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8226 - accuracy: 0.6428 - val_loss
Epoch 115/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8207 - accuracy: 0.6479 - val_loss
Epoch 116/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8262 - accuracy: 0.6452 - val_loss
Epoch 117/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8158 - accuracy: 0.6565 - val_loss
Epoch 118/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8274 - accuracy: 0.6619 - val_loss
Epoch 119/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8113 - accuracy: 0.6479 - val_loss
Epoch 120/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8286 - accuracy: 0.6517 - val_loss
Epoch 121/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8170 - accuracy: 0.6513 - val_loss
Epoch 122/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8134 - accuracy: 0.6510 - val_loss
Epoch 123/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8156 - accuracy: 0.6513 - val_loss
Epoch 124/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8255 - accuracy: 0.6483 - val_loss
Epoch 125/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8092 - accuracy: 0.6602 - val_loss
Epoch 126/200
76/92 [=======================>......] - ETA: 0s - loss: 0.8100 - accuracy: 0.6521
Epoch 126: ReduceLROnPlateau reducing learning rate to 0.0002500000118743628.
92/92 [==============================] - 0s 4ms/step - loss: 0.8099 - accuracy: 0.6541 - val_loss
Epoch 127/200
92/92 [==============================] - 0s 4ms/step - loss: 0.8046 - accuracy: 0.6629 - val_loss
Epoch 128/200
92/92 [==============================] - 0s 4ms/step - loss: 0.7869 - accuracy: 0.6585 - val_loss
Epoch 129/200
92/92 [==============================] - 0s 3ms/step - loss: 0.8045 - accuracy: 0.6691 - val_loss
Epoch 130/200
92/92 [==============================] - 0s 3ms/step - loss: 0.7998 - accuracy: 0.6639 - val_loss
Epoch 131/200
92/92 [==============================] - 0s 4ms/step - loss: 0.7963 - accuracy: 0.6633 - val_loss
Epoch 132/200
92/92 [==============================] - 0s 4ms/step - loss: 0.7867 - accuracy: 0.6680 - val_loss
Epoch 133/200
92/92 [==============================] - 0s 3ms/step - loss: 0.7950 - accuracy: 0.6663 - val_loss
```

```
from sklearn.metrics import roc_auc_score
from sklearn.preprocessing import label_binarize

# Predict softmax probabilities from Keras model
y_val_probs = model.predict(X_val_scaled)

# Binarize the true labels to one-hot (if not already)
y_val_bin = label_binarize(y_val, classes=[0, 1, 2, 3])  # shape: (n_samples, 4)

# ROC AUC Scores
auc_macro = roc_auc_score(y_val_bin, y_val_probs, average='macro', multi_class='ovr')
auc_weighted = roc_auc_score(y_val_bin, y_val_probs, average='weighted', multi_class='ovr')
auc_per_class = [roc_auc_score(y_val_bin[:, i], y_val_probs[:, i]) for i in range(4)]

print(f"ROC AUC (macro): {auc_macro:.4f}")
print(f"ROC AUC (weighted): {auc_weighted:.4f}")
for i, auc in enumerate(auc_per_class):
    print(f"Class {i} AUC: {auc:.4f}")
```

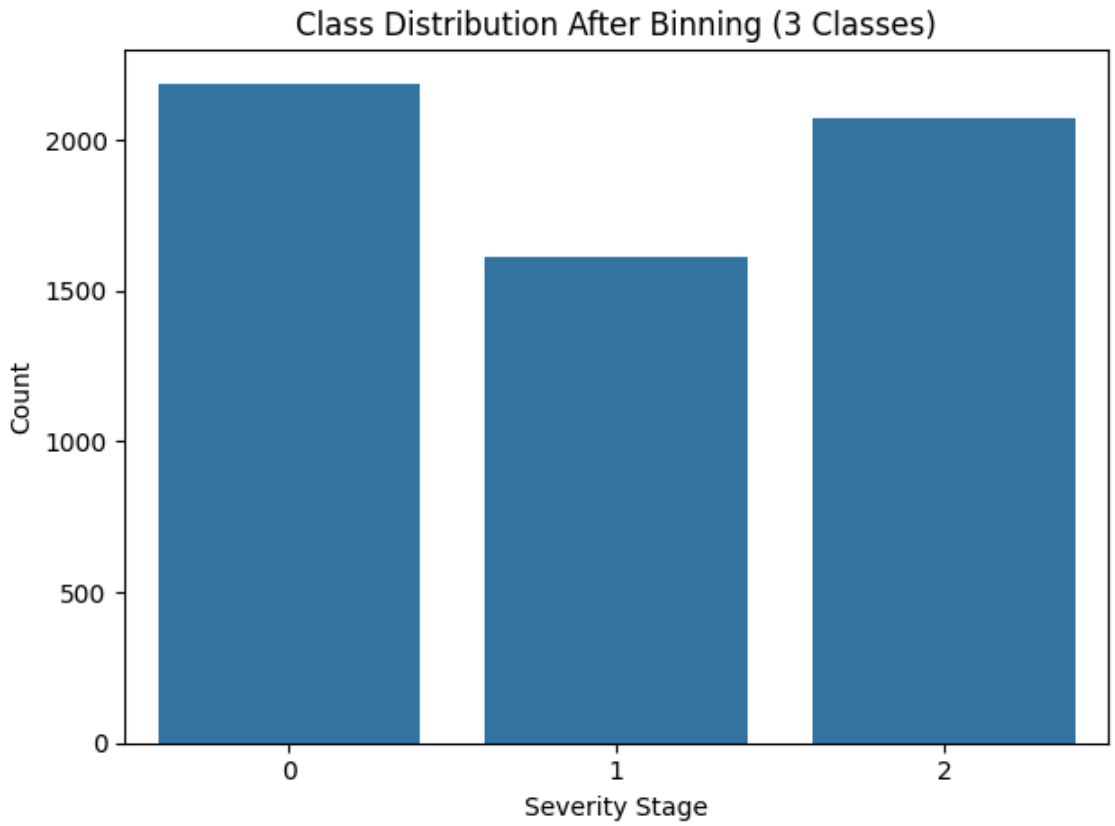
```
46/46 [==============================] - 0s 1ms/step
ROC AUC (macro): 0.8579
ROC AUC (weighted): 0.8489
Class 0 AUC: 0.9055
Class 1 AUC: 0.8066
Class 2 AUC: 0.8371
Class 3 AUC: 0.8823
```



3-way split

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.utils import to_categorical
import seaborn as sns
import matplotlib.pyplot as plt

# --- Binning: 3-class severity levels based on total_UPDRS ---
bins = [0, 25, 32, float('inf')]
labels = [0, 1, 2]
y_classify = pd.cut(y_target, bins=bins, labels=labels).astype(int)

# --- Train/Val/Test split (with stratification on class labels) ---
X_train, X_temp, y_train, y_temp = train_test_split(
    X_filtered, y_classify, test_size=0.5, stratify=y_classify, random_state=42
)
X_val, X_test, y_val, y_test = train_test_split(
    X_temp, y_temp, test_size=0.5, stratify=y_temp, random_state=42
)

# --- Scale input features ---
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# --- One-hot encode class labels for classification ---
num_classes = len(np.unique(y_classify))
```

```
y_train_cat = to_categorical(y_train, num_classes=num_classes)
y_val_cat = to_categorical(y_val, num_classes=num_classes)

# --- Plot class distribution ---
sns.countplot(x=y_classify)
plt.title("Class Distribution After Binning (3 Classes)")
plt.xlabel("Severity Stage")
plt.ylabel("Count")
plt.tight_layout()
plt.show()
```




```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, BatchNormalization, LeakyReLU
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from tensorflow.keras.optimizers import Adam

# Build model
model = Sequential([
    Dense(256, input_shape=(X_train_scaled.shape[1],)),
    LeakyReLU(alpha=0.01),
    BatchNormalization(),
    Dropout(0.4),

    Dense(128),
    LeakyReLU(alpha=0.01),
    BatchNormalization(),
    Dropout(0.3),

    Dense(64),
    LeakyReLU(alpha=0.01),
    BatchNormalization(),
    Dropout(0.2),

    Dense(32),
    LeakyReLU(alpha=0.01),
```

```
    Dense(3, activation='softmax')
])

# Compile model
model.compile(
    optimizer=Adam(learning_rate=0.01),
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

# Callbacks
early_stop = EarlyStopping(monitor='val_loss', patience=50, restore_best_weights=True, verbose=1
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=15, verbose=1)

# Train
history = model.fit(
    X_train_scaled, y_train_cat,
    validation_data=(X_val_scaled, y_val_cat),
    epochs=300,
    batch_size=16,
    callbacks=[early_stop, reduce_lr],
    verbose=1
)

y_val_pred = model.predict(X_val_scaled)
y_val_pred_classes = tf.argmax(y_val_pred, axis=1)

print("Keras MLP Classification Report:")
print(classification_report(y_val, y_val_pred_classes))

cm = confusion_matrix(y_val, y_val_pred_classes)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title("Final Confusion Matrix (Validation Set)")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()
```

```
Epoch 1/300
184/184 [==============================] - 3s 6ms/step - loss: 1.1795 - accuracy: 0.4205 - val_lo
Epoch 2/300
184/184 [==============================] - 1s 4ms/step - loss: 1.0381 - accuracy: 0.4743 - val_lo
Epoch 3/300
184/184 [==============================] - 1s 4ms/step - loss: 1.0018 - accuracy: 0.4923 - val_lo
Epoch 4/300
184/184 [==============================] - 1s 4ms/step - loss: 0.9691 - accuracy: 0.5182 - val_lo
Epoch 5/300
184/184 [==============================] - 1s 4ms/step - loss: 0.9405 - accuracy: 0.5475 - val_lo
Epoch 6/300
184/184 [==============================] - 1s 4ms/step - loss: 0.9262 - accuracy: 0.5495 - val_lo
Epoch 7/300
184/184 [==============================] - 1s 4ms/step - loss: 0.9260 - accuracy: 0.5359 - val_lo
Epoch 8/300
184/184 [==============================] - 1s 4ms/step - loss: 0.9037 - accuracy: 0.5591 - val_lo
Epoch 9/300
184/184 [==============================] - 1s 4ms/step - loss: 0.9162 - accuracy: 0.5485 - val_lo
Epoch 10/300
184/184 [==============================] - 1s 4ms/step - loss: 0.9060 - accuracy: 0.5635 - val_lo
Epoch 11/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8863 - accuracy: 0.5577 - val_lo
Epoch 12/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8778 - accuracy: 0.5778 - val_lo
Epoch 13/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8703 - accuracy: 0.5863 - val_lo
Epoch 14/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8546 - accuracy: 0.5918 - val_lo
Epoch 15/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8713 - accuracy: 0.5754 - val_lo
Epoch 16/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8586 - accuracy: 0.5826 - val_lo
Epoch 17/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8545 - accuracy: 0.5822 - val_lo
Epoch 18/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8625 - accuracy: 0.5877 - val_lo
Epoch 19/300
184/184 [==============================] - 1s 3ms/step - loss: 0.8481 - accuracy: 0.5819 - val_lo
Epoch 20/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8429 - accuracy: 0.5986 - val_lo
Epoch 21/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8558 - accuracy: 0.5894 - val_lo
Epoch 22/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8552 - accuracy: 0.5819 - val_lo
Epoch 23/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8500 - accuracy: 0.5924 - val_lo
Epoch 24/300
184/184 [==============================] - 1s 3ms/step - loss: 0.8245 - accuracy: 0.6010 - val_lo
Epoch 25/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8278 - accuracy: 0.6078 - val_lo
Epoch 26/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8258 - accuracy: 0.6020 - val_lo
Epoch 27/300
184/184 [==============================] - 1s 3ms/step - loss: 0.8255 - accuracy: 0.6098 - val_lo
Epoch 28/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8230 - accuracy: 0.6067 - val_lo
Epoch 29/300
184/184 [==============================] - 1s 3ms/step - loss: 0.8268 - accuracy: 0.6078 - val_lo
Epoch 30/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8166 - accuracy: 0.6166 - val_lo
Epoch 31/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8335 - accuracy: 0.5918 - val_lo
Epoch 32/300
184/184 [==============================] - 1s 4ms/step - loss: 0.8063 - accuracy: 0.6003 - val_lo
Epoch 33/300
184/184 [==============================] - 1s 5ms/step - loss: 0.8141 - accuracy: 0.6132 - val_lo
Epoch 34/300
```

```
184/184 [==============================] - 1s 5ms/step - loss: 0.8037 - accuracy: 0.6214 - val_lo
Epoch 35/300
184/184 [==============================] - 1s 5ms/step - loss: 0.8185 - accuracy: 0.6006 - val_lo
Epoch 36/300
184/184 [==============================] - 1s 5ms/step - loss: 0.8066 - accuracy: 0.6176 - val_lo
Epoch 37/300
184/184 [==============================] - 1s 5ms/step - loss: 0.8059 - accuracy: 0.6224 - val_lo
Epoch 38/300
184/184 [==============================] - 1s 5ms/step - loss: 0.8063 - accuracy: 0.6071 - val_lo
Epoch 39/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7946 - accuracy: 0.6176 - val_lo
Epoch 40/300
184/184 [==============================] - 1s 3ms/step - loss: 0.8016 - accuracy: 0.6156 - val_lo
Epoch 41/300
184/184 [==============================] - 1s 3ms/step - loss: 0.8013 - accuracy: 0.6224 - val_lo
Epoch 42/300
184/184 [==============================] - 1s 3ms/step - loss: 0.8046 - accuracy: 0.6159 - val_lo
Epoch 43/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7941 - accuracy: 0.6156 - val_lo
Epoch 44/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7998 - accuracy: 0.6227 - val_lo
Epoch 45/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7839 - accuracy: 0.6333 - val_lo
Epoch 46/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7827 - accuracy: 0.6224 - val_lo
Epoch 47/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7850 - accuracy: 0.6319 - val_lo
Epoch 48/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7730 - accuracy: 0.6381 - val_lo
Epoch 49/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7829 - accuracy: 0.6309 - val_lo
Epoch 50/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7845 - accuracy: 0.6313 - val_lo
Epoch 51/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7827 - accuracy: 0.6340 - val_lo
Epoch 52/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7763 - accuracy: 0.6299 - val_lo
Epoch 53/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7861 - accuracy: 0.6255 - val_lo
Epoch 54/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7755 - accuracy: 0.6411 - val_lo
Epoch 55/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7726 - accuracy: 0.6350 - val_lo
Epoch 56/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7709 - accuracy: 0.6422 - val_lo
Epoch 57/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7643 - accuracy: 0.6391 - val_lo
Epoch 58/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7586 - accuracy: 0.6319 - val_lo
Epoch 59/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7676 - accuracy: 0.6367 - val_lo
Epoch 60/300
184/184 [==============================] - 1s 5ms/step - loss: 0.7708 - accuracy: 0.6306 - val_lo
Epoch 61/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7566 - accuracy: 0.6411 - val_lo
Epoch 62/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7550 - accuracy: 0.6398 - val_lo
Epoch 63/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7552 - accuracy: 0.6425 - val_lo
Epoch 64/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7601 - accuracy: 0.6445 - val_lo
Epoch 65/300
184/184 [==============================] - 1s 5ms/step - loss: 0.7676 - accuracy: 0.6428 - val_lo
Epoch 66/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7569 - accuracy: 0.6350 - val_lo
Epoch 67/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7616 - accuracy: 0.6462 - val_lo
Epoch 68/300
```

```
184/184 [==============================] - 1s 4ms/step - loss: 0.7611 - accuracy: 0.6391 - val_lo
Epoch 69/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7535 - accuracy: 0.6387 - val_lo
Epoch 70/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7691 - accuracy: 0.6462 - val_lo
Epoch 71/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7556 - accuracy: 0.6469 - val_lo
Epoch 72/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7428 - accuracy: 0.6442 - val_lo
Epoch 73/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7463 - accuracy: 0.6561 - val_lo
Epoch 74/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7399 - accuracy: 0.6500 - val_lo
Epoch 75/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7549 - accuracy: 0.6394 - val_lo
Epoch 76/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7466 - accuracy: 0.6452 - val_lo
Epoch 77/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7594 - accuracy: 0.6323 - val_lo
Epoch 78/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7402 - accuracy: 0.6582 - val_lo
Epoch 79/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7480 - accuracy: 0.6534 - val_lo
Epoch 80/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7491 - accuracy: 0.6456 - val_lo
Epoch 81/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7444 - accuracy: 0.6571 - val_lo
Epoch 82/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7264 - accuracy: 0.6575 - val_lo
Epoch 83/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7355 - accuracy: 0.6571 - val_lo
Epoch 84/300
184/184 [==============================] - 1s 4ms/step - loss: 0.7355 - accuracy: 0.6622 - val_lo
Epoch 85/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7159 - accuracy: 0.6554 - val_lo
Epoch 86/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7381 - accuracy: 0.6507 - val_lo
Epoch 87/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7332 - accuracy: 0.6513 - val_lo
Epoch 88/300
176/184 [===========================>..] - ETA: 0s - loss: 0.7338 - accuracy: 0.6538
Epoch 88: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
184/184 [==============================] - 1s 3ms/step - loss: 0.7344 - accuracy: 0.6503 - val_lo
Epoch 89/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7280 - accuracy: 0.6670 - val_lo
Epoch 90/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7274 - accuracy: 0.6660 - val_lo
Epoch 91/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7156 - accuracy: 0.6639 - val_lo
Epoch 92/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7196 - accuracy: 0.6677 - val_lo
Epoch 93/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6933 - accuracy: 0.6806 - val_lo
Epoch 94/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6991 - accuracy: 0.6786 - val_lo
Epoch 95/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7006 - accuracy: 0.6684 - val_lo
Epoch 96/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6985 - accuracy: 0.6711 - val_lo
Epoch 97/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6926 - accuracy: 0.6830 - val_lo
Epoch 98/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7123 - accuracy: 0.6718 - val_lo
Epoch 99/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6917 - accuracy: 0.6830 - val_lo
Epoch 100/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7179 - accuracy: 0.6745 - val_lo
Epoch 101/300
```

```
Epoch 101/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7111 - accuracy: 0.6687 - val_lo
Epoch 102/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6970 - accuracy: 0.6687 - val_lo
Epoch 103/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7081 - accuracy: 0.6748 - val_lo
Epoch 104/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6949 - accuracy: 0.6759 - val_lo
Epoch 105/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6770 - accuracy: 0.6908 - val_lo
Epoch 106/300
180/184 [============================>.] - ETA: 0s - loss: 0.6940 - accuracy: 0.6743
Epoch 106: ReduceLROnPlateau reducing learning rate to 0.00025000000118743628.
184/184 [==============================] - 1s 3ms/step - loss: 0.6925 - accuracy: 0.6745 - val_lo
Epoch 107/300
184/184 [==============================] - 1s 3ms/step - loss: 0.7026 - accuracy: 0.6799 - val_lo
Epoch 108/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6921 - accuracy: 0.6772 - val_lo
Epoch 109/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6830 - accuracy: 0.6956 - val_lo
Epoch 110/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6788 - accuracy: 0.6912 - val_lo
Epoch 111/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6732 - accuracy: 0.6959 - val_lo
Epoch 112/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6761 - accuracy: 0.6898 - val_lo
Epoch 113/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6778 - accuracy: 0.6803 - val_lo
Epoch 114/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6778 - accuracy: 0.6905 - val_lo
Epoch 115/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6744 - accuracy: 0.6939 - val_lo
Epoch 116/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6681 - accuracy: 0.6902 - val_lo
Epoch 117/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6697 - accuracy: 0.6895 - val_lo
Epoch 118/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6663 - accuracy: 0.6990 - val_lo
Epoch 119/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6711 - accuracy: 0.6977 - val_lo
Epoch 120/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6632 - accuracy: 0.6949 - val_lo
Epoch 121/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6703 - accuracy: 0.6977 - val_lo
Epoch 122/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6743 - accuracy: 0.6966 - val_lo
Epoch 123/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6590 - accuracy: 0.6953 - val_lo
Epoch 124/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6696 - accuracy: 0.6905 - val_lo
Epoch 125/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6686 - accuracy: 0.6912 - val_lo
Epoch 126/300
179/184 [============================>.] - ETA: 0s - loss: 0.6801 - accuracy: 0.6917
Epoch 126: ReduceLROnPlateau reducing learning rate to 0.00012500000059371814.
184/184 [==============================] - 1s 3ms/step - loss: 0.6757 - accuracy: 0.6942 - val_lo
Epoch 127/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6590 - accuracy: 0.6987 - val_lo
Epoch 128/300
184/184 [==============================] - 1s 4ms/step - loss: 0.6679 - accuracy: 0.6973 - val_lo
Epoch 129/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6716 - accuracy: 0.6908 - val_lo
Epoch 130/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6706 - accuracy: 0.6834 - val_lo
Epoch 131/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6720 - accuracy: 0.6898 - val_lo
Epoch 132/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6665 - accuracy: 0.6959 - val_lo
```

```
Epoch 133/300
184/184 [==============================] - 1s 3ms/step - loss: 0.6842 - accuracy: 0.6816 - val_lo
Epoch 134/300
```