California_housing

August 14, 2023

[ ]:
```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.wrappers.scikit_learn import KerasRegressor,␣
↪KerasClassifier
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

import matplotlib.pyplot as plt
import seaborn as sns

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.datasets import fetch_california_housing
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import precision_score, f1_score, accuracy_score,␣
↪recall_score, confusion_matrix
```

1 Regression problem

1.1 Loading dataset

Load the Boston Housing dataset from the Keras library.

Due to the ethical issues surrounding the Boston housing dataset, primarily the inclusion of a racist feature engineered by the dataset's authors, the Boston housing dataset will not be used. Alternatively, the California housing dataset, available in SKlearn, will be used instead.

[ ]:
```
ca_housing = fetch_california_housing(as_frame=True)
```

[ ]:
```
feature_df = ca_housing.data
target_df = ca_housing.target
```

1.2 Explore and preprocess

Explore and preprocess the data (e.g., normalization, one-hot encoding, etc.).

[ ]:
```python
(num_samples, num_features) = feature_df.shape

print(f'Sample size: {num_samples}')
print(f'Number of features: {num_features}')
```

```
Sample size: 20640
Number of features: 8
```

[ ]:
```python
# concatenate all train and test data and targets together
full_df = pd.concat([feature_df, target_df], axis=1)
```

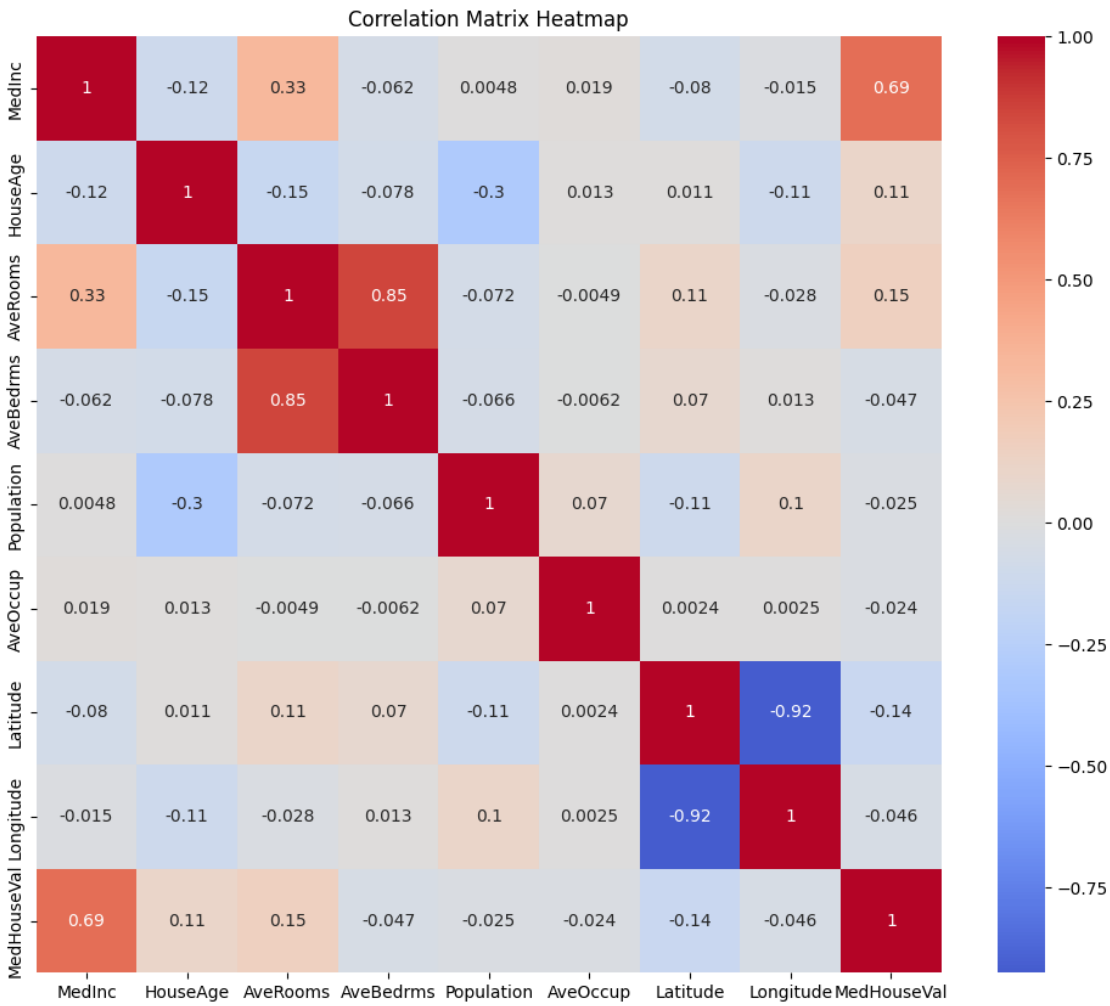
[ ]:
```python
full_df
```

[ ]:
```
       MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
0      8.3252      41.0  6.984127   1.023810       322.0  2.555556     37.88
1      8.3014      21.0  6.238137   0.971880      2401.0  2.109842     37.86
2      7.2574      52.0  8.288136   1.073446       496.0  2.802260     37.85
3      5.6431      52.0  5.817352   1.073059       558.0  2.547945     37.85
4      3.8462      52.0  6.281853   1.081081       565.0  2.181467     37.85
…         …        …        …          …          …        …
20635  1.5603      25.0  5.045455   1.133333       845.0  2.560606     39.48
20636  2.5568      18.0  6.114035   1.315789       356.0  3.122807     39.49
20637  1.7000      17.0  5.205543   1.120092      1007.0  2.325635     39.43
20638  1.8672      18.0  5.329513   1.171920       741.0  2.123209     39.43
20639  2.3886      16.0  5.254717   1.162264      1387.0  2.616981     39.37

       Longitude  MedHouseVal
0        -122.23        4.526
1        -122.22        3.585
2        -122.24        3.521
3        -122.25        3.413
4        -122.25        3.422
…           …          …
20635    -121.09        0.781
20636    -121.21        0.771
20637    -121.22        0.923
20638    -121.32        0.847
20639    -121.24        0.894

[20640 rows x 9 columns]
```

1.2.1 Correlation Matrix

[ ]:
```python
correlation_matrix = full_df.corr()

corr_w_target = correlation_matrix['MedHouseVal']
```

[ ]:
```
plt.figure(figsize=(12,10))

sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', center=0)

plt.title('Correlation Matrix Heatmap')
plt.show()
```



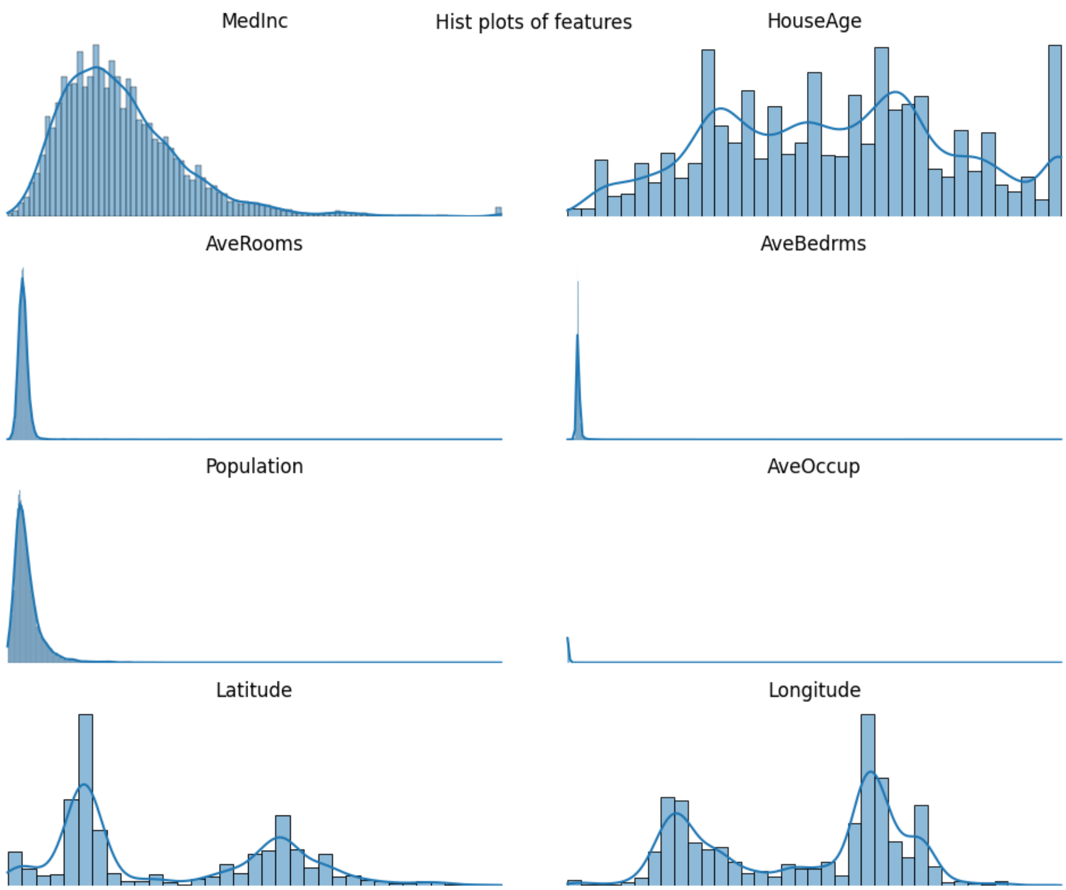
[ ]:
```
print(f'Correlations with the target variable:\n{corr_w_target}')
```

```
Correlations with the target variable:
MedInc          0.688075
HouseAge        0.105623
AveRooms        0.151948
AveBedrms      -0.046701
Population     -0.024650
AveOccup       -0.023737
```

```
Latitude      -0.144160
Longitude     -0.045967
MedHouseVal    1.000000
Name: MedHouseVal, dtype: float64
```

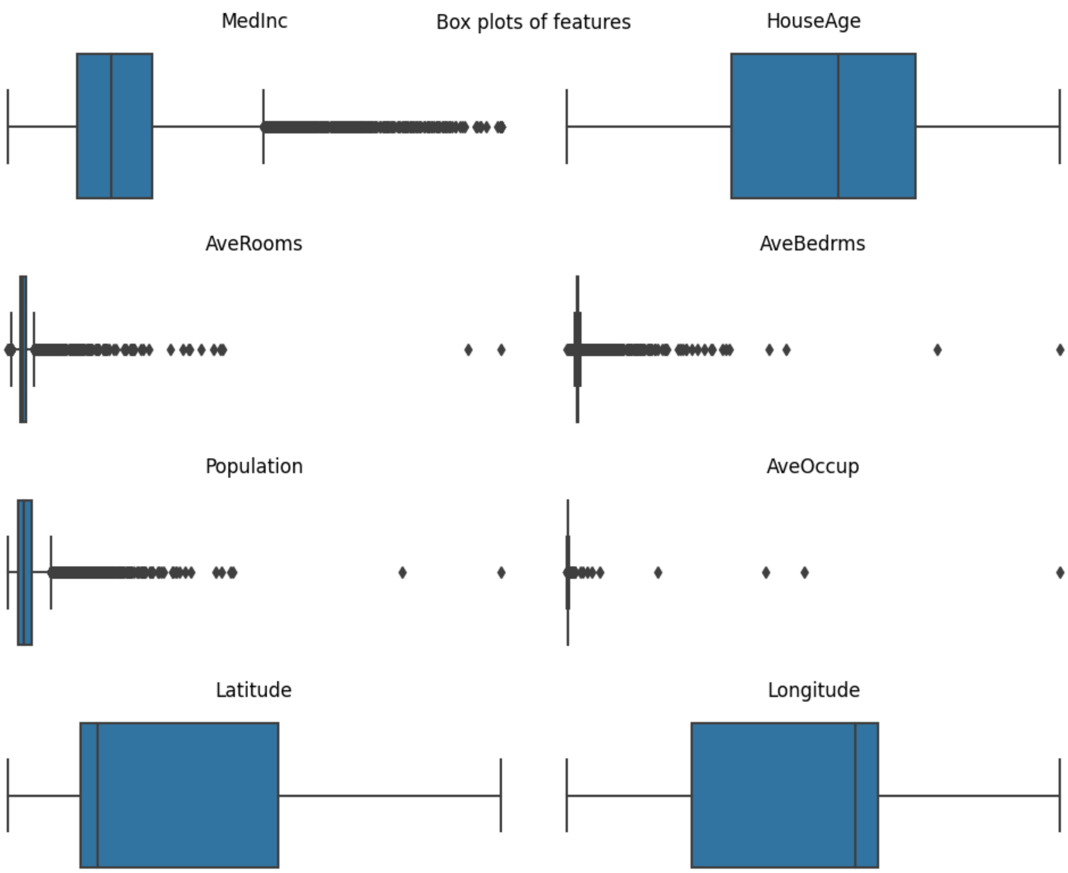
1.2.2 Feature histplots

[ ]:
```
num_rows = 4
num_cols = 2

fig, axes = plt.subplots(num_rows, num_cols, figsize=(10,8))
fig.subplots_adjust(hspace=0.5)

for i in range(num_features):
    row_idx = i // num_cols
    col_idx = i % num_cols

    sns.histplot(feature_df.iloc[:,i], ax=axes[row_idx, col_idx], kde=True)
    axes[row_idx, col_idx].set_title(f'{feature_df.columns[i]}')

    if i >= num_features - (num_rows * num_cols):
        axes[row_idx, col_idx].axis('off')

plt.tight_layout()
plt.suptitle('Hist plots of features')
plt.show()
```



1.2.3 Feature boxplots

[ ]:
```python
fig, axes = plt.subplots(num_rows, num_cols, figsize=(10,8))
fig.subplots_adjust(hspace=0.5)

for i in range(num_features):
    row_idx = i // num_cols
    col_idx = i % num_cols

    sns.boxplot(x=feature_df.iloc[:,i], ax=axes[row_idx, col_idx])
    axes[row_idx, col_idx].set_title(f'{feature_df.columns[i]}')

    if i >= num_features - (num_rows * num_cols):
        axes[row_idx, col_idx].axis('off')

plt.tight_layout()
plt.suptitle('Box plots of features')
plt.show()
```



1.2.4 Address significant outliers

[ ]:
```
full_df.describe()
```

[ ]:
```
              MedInc     HouseAge     AveRooms    AveBedrms   Population  \
count  20640.000000  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.870671     28.639486      5.429000      1.096675   1425.476744
std        1.899822     12.585558      2.474173      0.473911   1132.462122
min        0.499900      1.000000      0.846154      0.333333      3.000000
25%        2.563400     18.000000      4.440716      1.006079    787.000000
50%        3.534800     29.000000      5.229129      1.048780   1166.000000
75%        4.743250     37.000000      6.052381      1.099526   1725.000000
max       15.000100     52.000000    141.909091     34.066667  35682.000000

           AveOccup      Latitude     Longitude   MedHouseVal
count  20640.000000  20640.000000  20640.000000  20640.000000
mean       3.070655     35.631861   -119.569704      2.068558
std       10.386050      2.135952      2.003532      1.153956
min        0.692308     32.540000   -124.350000      0.149990
```

```
25%       2.429741     33.930000   -121.800000      1.196000
50%       2.818116     34.260000   -118.490000      1.797000
75%       3.282261     37.710000   -118.010000      2.647250
max    1243.333333     41.950000   -114.310000      5.000010
```

[ ]:
```python
full_df[full_df['AveRooms'] > 50]
```

[ ]:
```
       MedInc  HouseAge    AveRooms  AveBedrms  Population  AveOccup  \
1912   4.9750      16.0   56.269231  10.153846        54.0  2.076923
1913   4.0714      19.0   61.812500  11.000000       112.0  2.333333
1914   1.8750      33.0  141.909091  25.636364        30.0  2.727273
1979   4.6250      34.0  132.533333  34.066667        36.0  2.400000
2395   3.8750      23.0   50.837838  10.270270        64.0  1.729730
9676   3.2431      14.0   52.848214  11.410714       265.0  2.366071
11707  1.1912      22.0   52.690476   8.857143        98.0  2.333333
11862  2.6250      25.0   59.875000  15.312500        28.0  1.750000
12447  1.6154      17.0   62.422222  14.111111        83.0  1.844444

       Latitude  Longitude  MedHouseVal
1912      39.01    -120.16      2.06300
1913      39.01    -120.06      4.37500
1914      38.91    -120.10      5.00001
1979      38.80    -120.08      1.62500
2395      37.12    -119.34      1.25000
9676      37.64    -119.02      2.21400
11707     39.15    -120.06      1.70000
11862     40.27    -121.25      0.67500
12447     33.97    -114.49      0.87500
```

[ ]:
```python
# drop outliers, averooms 132 and 141 and AveBedrms 34 and 25
full_df = full_df.loc[full_df['AveRooms'] < 65]
```

[ ]:
```python
full_df[full_df['AveOccup'] > 50]
```

[ ]:
```
        MedInc  HouseAge  AveRooms  AveBedrms  Population     AveOccup  \
3364    5.5179      36.0  5.142857   1.142857      4198.0   599.714286
9172    4.2391       5.0  5.123810   0.933333      8733.0    83.171429
12104   1.6250       8.0  7.600000   0.950000      1275.0    63.750000
13034   6.1359      52.0  8.275862   1.517241      6675.0   230.172414
16420   5.7485      26.0  5.366667   0.900000      1542.0    51.400000
16669   4.2639      46.0  9.076923   1.307692      6532.0   502.461538
19006  10.2264      45.0  3.166667   0.833333      7460.0  1243.333333

       Latitude  Longitude  MedHouseVal
3364      40.41    -120.51        0.675
9172      34.47    -118.59        1.546
12104     33.97    -117.33        1.625
```

```
13034     38.69    -121.15        2.250
16420     37.89    -121.29        1.625
16669     35.32    -120.70        3.500
19006     38.32    -121.98        1.375
```

[ ]:
```
# drop outliers, aveoccup 100+
full_df = full_df.loc[full_df['AveOccup'] < 100]
```

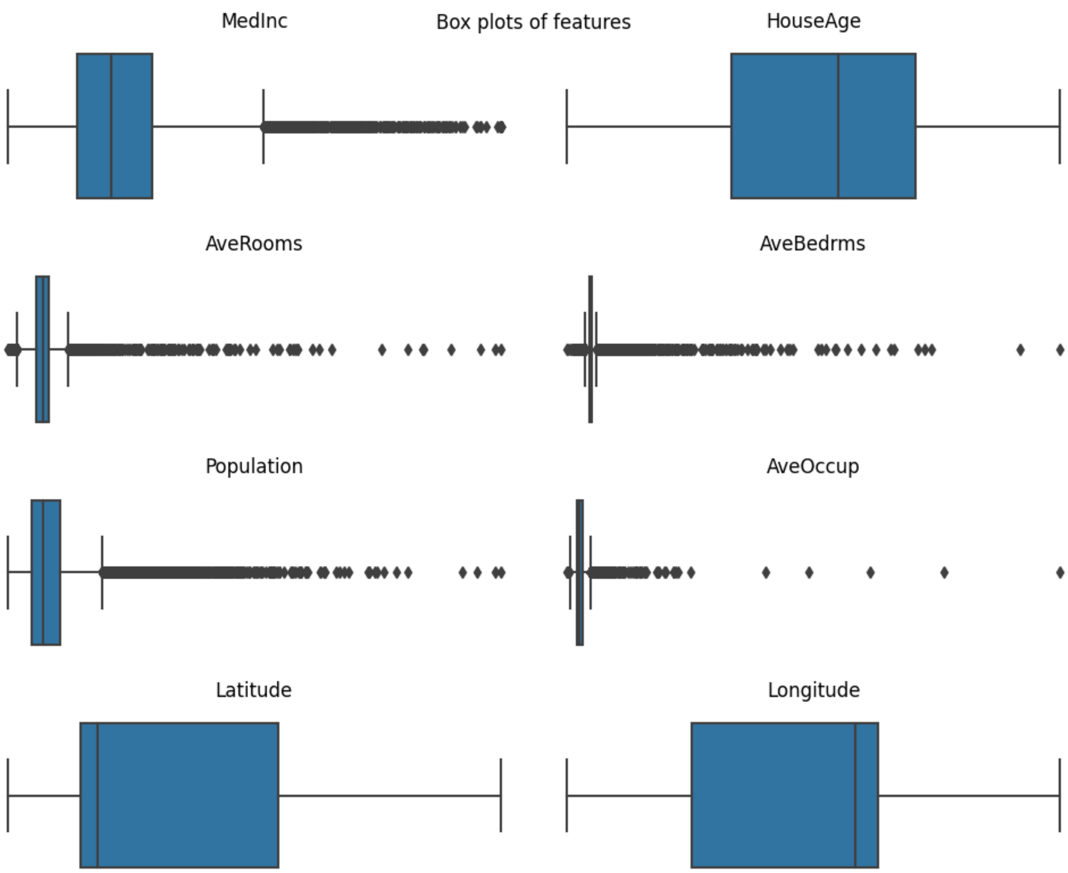
[ ]:
```
full_df[full_df['Population'] > 20000]
```

[ ]:
```
       MedInc  HouseAge  AveRooms  AveBedrms  Population  AveOccup  Latitude  \
9880   2.3087      11.0  5.364518   1.059684     28566.0  4.696810     36.64
15360  2.5729      14.0  5.270497   1.010484     35682.0  7.482072     33.35

       Longitude  MedHouseVal
9880     -121.79        1.188
15360    -117.42        1.344
```

[ ]:
```
# drop outliers, population 20k+
full_df = full_df.loc[full_df['Population'] < 20000]
```

1.2.5 New boxplots with significant outliers removed

[ ]:
```
fig, axes = plt.subplots(num_rows, num_cols, figsize=(10,8))
fig.subplots_adjust(hspace=0.5)

for i in range(num_features):
    row_idx = i // num_cols
    col_idx = i % num_cols

    sns.boxplot(x=full_df.iloc[:,i], ax=axes[row_idx, col_idx])
    axes[row_idx, col_idx].set_title(f'{feature_df.columns[i]}')

    if i >= num_features - (num_rows * num_cols):
        axes[row_idx, col_idx].axis('off')

plt.tight_layout()
plt.suptitle('Box plots of features')
plt.show()
```



1.3 TTS

Split the data into training and testing sets.

```
[ ]: X = full_df.drop(['MedHouseVal'], axis=1)
     y = full_df['MedHouseVal']
```

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
      ↪random_state=97)
```

1.3.1 Feature and target scaling

```
[ ]: scaler_x = MinMaxScaler()

     X_train_scaled = scaler_x.fit_transform(X_train)
     X_test_scaled = scaler_x.transform(X_test)
```

```
[ ]: scaler_y = MinMaxScaler()
```

```
y_train_arr = np.array(y_train)
y_test_arr = np.array(y_test)
y_train_arr = y_train_arr.reshape(-1, 1)
y_test_arr = y_test_arr.reshape(-1, 1)

y_train_scaled = scaler_y.fit_transform(y_train_arr)
y_test_scaled = scaler_y.transform(y_test_arr)
```

1.4 Model definition

Define a deep neural network architecture for regression using Keras.

[ ]:
```
def create_regression_model(optimizer='adam', activation='relu', dropout_rate=0.
↪2):
    model = Sequential()
    model.add(Dense(64, activation=activation, input_shape=(X_train_scaled.
↪shape[1],)))
    model.add(Dense(64, activation=activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1))

    model.compile(optimizer=optimizer, loss='mean_squared_error',␣
↪metrics=['mae'])

    return model
```

[ ]:
```
baseline_regression_model = create_regression_model()
```

1.5 Model training and testing

Train the model on the training set and evaluate its performance on the testing set.

1.5.1 Model training

[ ]:
```
epochs = 50
batch_size = 32

early_stopping = EarlyStopping(monitor='val_loss', patience=5,␣
↪restore_best_weights=True)
model_checkpoint = ModelCheckpoint('baseline_regression.h5',␣
↪save_best_only=True)

history = baseline_regression_model.fit(X_train_scaled, y_train_scaled,␣
↪epochs=epochs, batch_size=batch_size, validation_split=0.2, verbose=1,␣
↪callbacks=[early_stopping, model_checkpoint])
```

```
Epoch 1/50
413/413 [==============================] - 1s 1ms/step - loss: 0.0303 - mae:
```

```
0.1283 - val_loss: 0.0211 - val_mae: 0.1062
Epoch 2/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0235 - mae:
0.1111 - val_loss: 0.0186 - val_mae: 0.0951
Epoch 3/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0204 - mae:
0.1026 - val_loss: 0.0162 - val_mae: 0.0897
Epoch 4/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0187 - mae:
0.0975 - val_loss: 0.0153 - val_mae: 0.0865
Epoch 5/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0177 - mae:
0.0943 - val_loss: 0.0144 - val_mae: 0.0853
Epoch 6/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0172 - mae:
0.0929 - val_loss: 0.0142 - val_mae: 0.0855
Epoch 7/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0164 - mae:
0.0902 - val_loss: 0.0179 - val_mae: 0.1053
Epoch 8/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0163 - mae:
0.0903 - val_loss: 0.0137 - val_mae: 0.0815
Epoch 9/50
413/413 [==============================] - 1s 1ms/step - loss: 0.0157 - mae:
0.0887 - val_loss: 0.0131 - val_mae: 0.0820
Epoch 10/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0155 - mae:
0.0875 - val_loss: 0.0140 - val_mae: 0.0884
Epoch 11/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0153 - mae:
0.0871 - val_loss: 0.0130 - val_mae: 0.0805
Epoch 12/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0154 - mae:
0.0875 - val_loss: 0.0146 - val_mae: 0.0826
Epoch 13/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0151 - mae:
0.0864 - val_loss: 0.0131 - val_mae: 0.0795
Epoch 14/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0150 - mae:
0.0860 - val_loss: 0.0128 - val_mae: 0.0788
Epoch 15/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0147 - mae:
0.0854 - val_loss: 0.0144 - val_mae: 0.0817
Epoch 16/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0146 - mae:
0.0848 - val_loss: 0.0130 - val_mae: 0.0829
Epoch 17/50
413/413 [==============================] - 1s 1ms/step - loss: 0.0148 - mae:
```

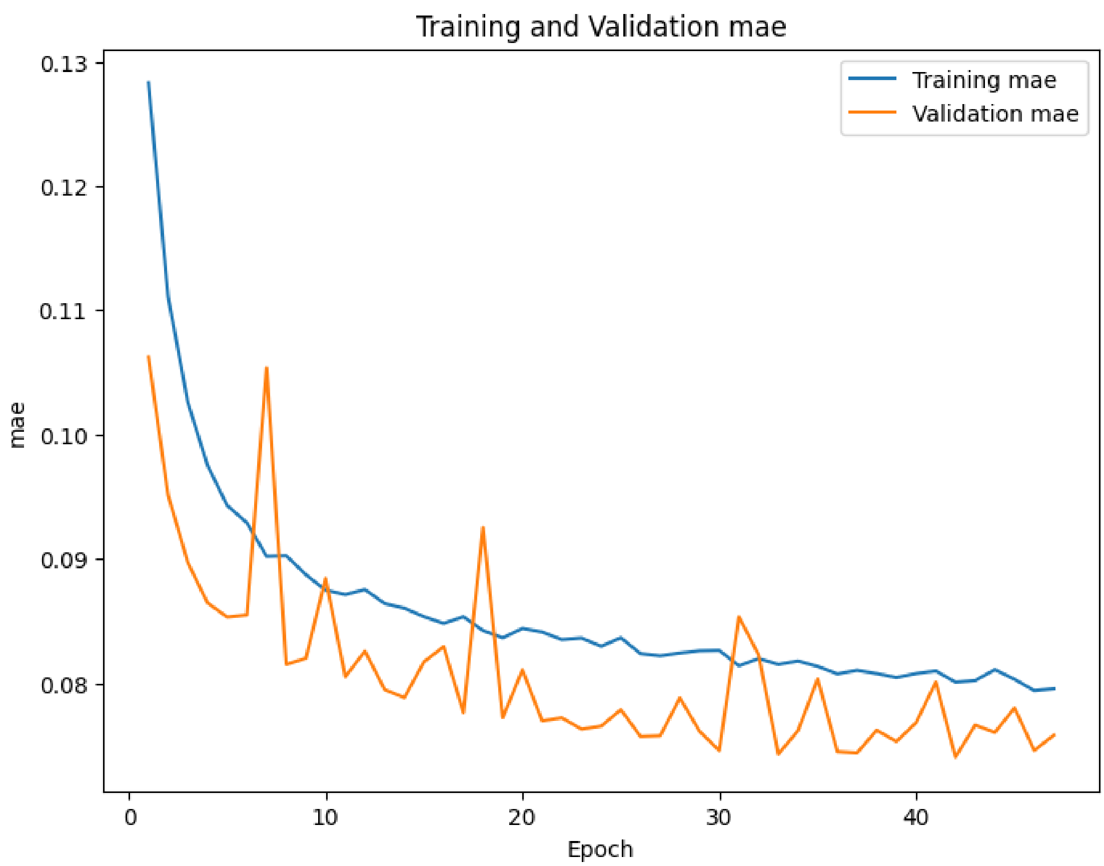
```
0.0854 - val_loss: 0.0133 - val_mae: 0.0776
Epoch 18/50
413/413 [==============================] - 1s 1ms/step - loss: 0.0145 - mae:
0.0842 - val_loss: 0.0156 - val_mae: 0.0925
Epoch 19/50
413/413 [==============================] - 1s 1ms/step - loss: 0.0143 - mae:
0.0837 - val_loss: 0.0123 - val_mae: 0.0772
Epoch 20/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0145 - mae:
0.0844 - val_loss: 0.0127 - val_mae: 0.0811
Epoch 21/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0142 - mae:
0.0841 - val_loss: 0.0122 - val_mae: 0.0770
Epoch 22/50
413/413 [==============================] - 1s 1ms/step - loss: 0.0142 - mae:
0.0835 - val_loss: 0.0130 - val_mae: 0.0772
Epoch 23/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0142 - mae:
0.0836 - val_loss: 0.0123 - val_mae: 0.0763
Epoch 24/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0141 - mae:
0.0830 - val_loss: 0.0126 - val_mae: 0.0766
Epoch 25/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0142 - mae:
0.0837 - val_loss: 0.0123 - val_mae: 0.0779
Epoch 26/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0138 - mae:
0.0824 - val_loss: 0.0122 - val_mae: 0.0757
Epoch 27/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0138 - mae:
0.0822 - val_loss: 0.0122 - val_mae: 0.0758
Epoch 28/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0139 - mae:
0.0824 - val_loss: 0.0136 - val_mae: 0.0788
Epoch 29/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0138 - mae:
0.0826 - val_loss: 0.0122 - val_mae: 0.0761
Epoch 30/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0140 - mae:
0.0826 - val_loss: 0.0120 - val_mae: 0.0746
Epoch 31/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0135 - mae:
0.0814 - val_loss: 0.0134 - val_mae: 0.0853
Epoch 32/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0136 - mae:
0.0820 - val_loss: 0.0150 - val_mae: 0.0823
Epoch 33/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0135 - mae:
```

```
0.0815 - val_loss: 0.0117 - val_mae: 0.0743
Epoch 34/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0136 - mae:
0.0818 - val_loss: 0.0117 - val_mae: 0.0762
Epoch 35/50
413/413 [==============================] - 1s 1ms/step - loss: 0.0135 - mae:
0.0814 - val_loss: 0.0123 - val_mae: 0.0804
Epoch 36/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0134 - mae:
0.0808 - val_loss: 0.0116 - val_mae: 0.0745
Epoch 37/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0135 - mae:
0.0810 - val_loss: 0.0115 - val_mae: 0.0744
Epoch 38/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0134 - mae:
0.0808 - val_loss: 0.0119 - val_mae: 0.0762
Epoch 39/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0132 - mae:
0.0805 - val_loss: 0.0115 - val_mae: 0.0753
Epoch 40/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0134 - mae:
0.0808 - val_loss: 0.0125 - val_mae: 0.0768
Epoch 41/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0133 - mae:
0.0810 - val_loss: 0.0122 - val_mae: 0.0801
Epoch 42/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0131 - mae:
0.0801 - val_loss: 0.0115 - val_mae: 0.0741
Epoch 43/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0133 - mae:
0.0802 - val_loss: 0.0131 - val_mae: 0.0766
Epoch 44/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0132 - mae:
0.0811 - val_loss: 0.0131 - val_mae: 0.0760
Epoch 45/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0131 - mae:
0.0803 - val_loss: 0.0120 - val_mae: 0.0780
Epoch 46/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0129 - mae:
0.0794 - val_loss: 0.0116 - val_mae: 0.0746
Epoch 47/50
413/413 [==============================] - 0s 1ms/step - loss: 0.0130 - mae:
0.0796 - val_loss: 0.0115 - val_mae: 0.0758
```

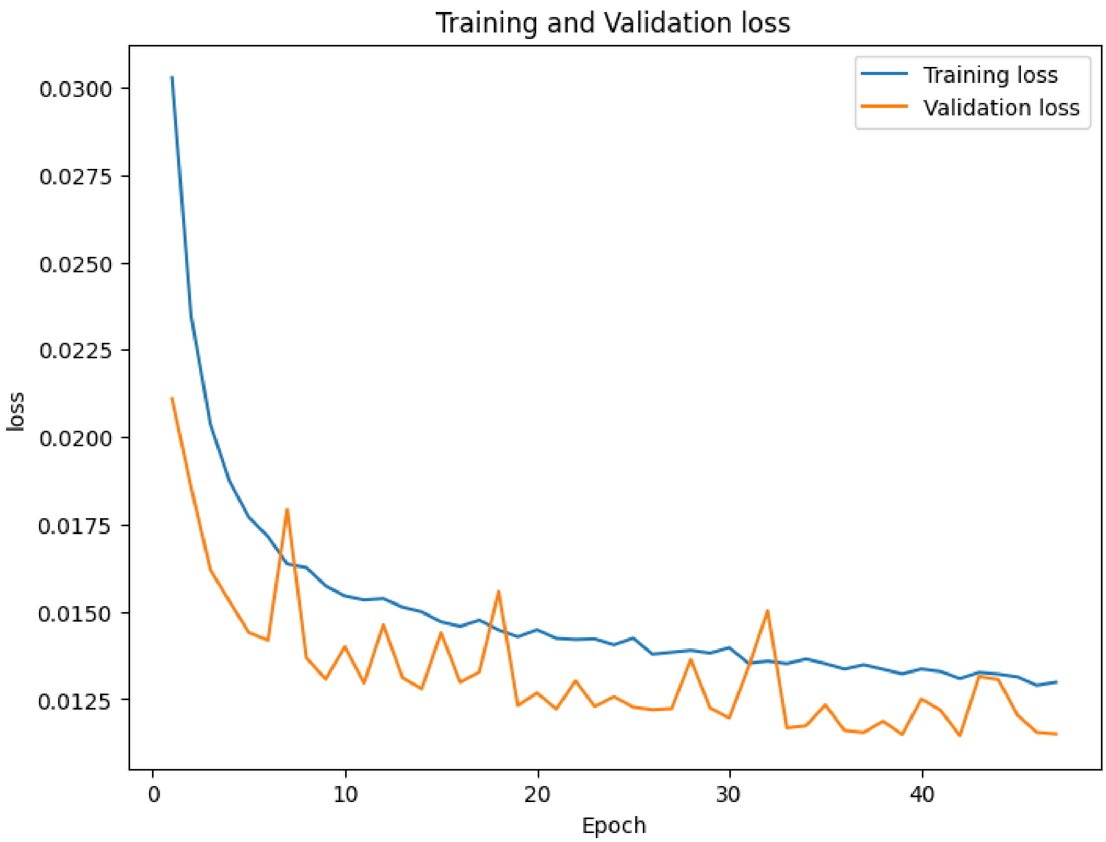
1.5.2 Baseline training perf

[ ]:
```python
def training_plots(history, metric):
    """
    Function to plot training and validation history for a given metric
    inputs: history - model training history
            metric - desired metric for plotting
    returns: -
    """
    epochs = len(history.history[f'{metric}'])
    plt.figure(figsize=(8, 6))
    sns.lineplot(x=range(1, epochs + 1), y=history.history[f'{metric}'],␣
 ↪label=f'Training {metric}')
    sns.lineplot(x=range(1, epochs + 1), y=history.history[f'val_{metric}'],␣
 ↪label=f'Validation {metric}')
    plt.xlabel('Epoch')
    plt.ylabel(f'{metric}')
    plt.title(f'Training and Validation {metric}')
    plt.legend()
    plt.show()
```

[ ]:
```python
training_plots(history, 'mae')
```



[ ]:
```python
training_plots(history, 'loss')
```



1.5.3 Basemodel performance in testing

[ ]:
```python
loss, mae = baseline_regression_model.evaluate(X_test_scaled, y_test_scaled,␣
    ↪verbose=0)
print(f'Test Loss: {loss:.4f}')
print(f'Test MAE: {mae:.4f}')
```

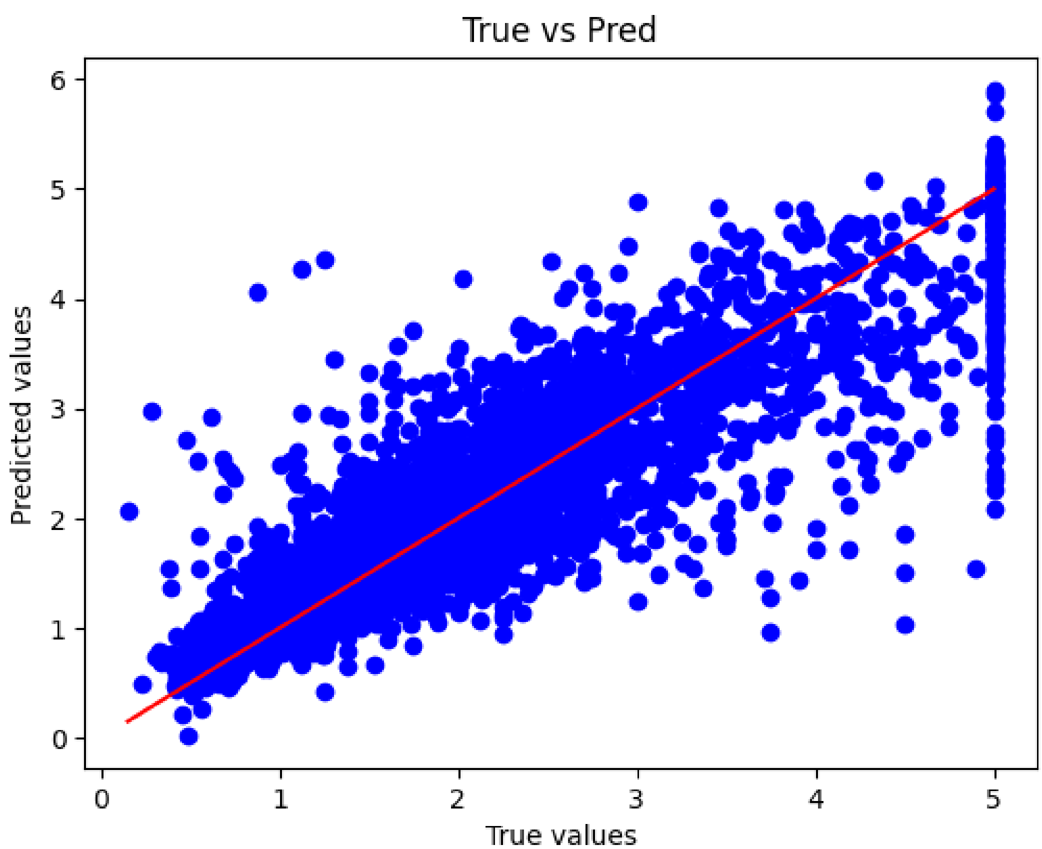
```
Test Loss: 0.0123
Test MAE: 0.0757
```

[ ]:
```python
y_pred = baseline_regression_model.predict(X_test_scaled)
y_pred_inv = scaler_y.inverse_transform(y_pred)
```

```
129/129 [==============================] - 0s 571us/step
```

[ ]:
```python
def testing_plot(y_true, y_pred, tuned=0):
    """
    Function to scatterplot testing performance of a regression model
    Inputs: y_true - Actual targets
            y_pred - Model predicted targets
            tuned - parameter to allow for plotting of 2 scatterplots
    Returns: -
    """
    if tuned:
        plt.scatter(y_true, y_pred, color='green', label='Tuned model')
    else:
        plt.scatter(y_true, y_pred, color='blue', label='Baseline model')
    plt.xlabel('True values')
    plt.ylabel('Predicted values')
    plt.title('True vs Pred')
    plt.plot([min(y_true), max(y_true)], [min(y_true), max(y_true)],␣
 ↪color='red')
```

[ ]:
```python
testing_plot(y_test, y_pred_inv)
plt.show()
```



1.6 Parameter tuning

Tune the hyperparameters of the model to achieve better performance (e.g., number of hidden layers, activation functions, learning rate, number of epochs, etc.).

[ ]:
```
tuning_regression_model = KerasRegressor(build_fn=create_regression_model,␣
↪verbose=0)

param_grid = {
    'optimizer': ['adam', 'sgd'],
    'activation': ['relu', 'tanh'],
    'batch_size': [16, 32, 64],
    'epochs': [20, 50],
    'dropout_rate': [0.2, 0.3]
}
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_16472\1470353211.py:1:
DeprecationWarning: KerasRegressor is deprecated, use Sci-Keras
(https://github.com/adriangb/scikeras) instead. See
https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  tuning_regression_model = KerasRegressor(build_fn=create_regression_model,
verbose=0)
```

[ ]:
```
regressor_grid = GridSearchCV(estimator=tuning_regression_model,␣
↪param_grid=param_grid, cv=3, n_jobs=-1)

grid_result = regressor_grid.fit(X_train_scaled, y_train_scaled,␣
↪callbacks=[early_stopping])
```

```
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
```

```
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
```

```
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
WARNING:tensorflow:Early stopping conditioned on metric `val_loss` which is not
available. Available metrics are: loss,mae
```

[ ]:
```
gs_regression_df = pd.DataFrame(grid_result.cv_results_)
gs_regression_df = gs_regression_df.drop(columns=['mean_fit_time',
 'std_fit_time', 'mean_score_time', 'std_score_time', 'params',
 'split0_test_score', 'split1_test_score', 'split2_test_score'])

gs_regression_df = gs_regression_df.sort_values(by='rank_test_score',
 ascending=True)
gs_regression_df.head(20).round(4)
```

[ ]:
```
   param_activation param_batch_size param_dropout_rate param_epochs  \
10             relu               32                0.2           50
22             relu               64                0.3           50
6              relu               16                0.3           50
14             relu               32                0.3           50
```

```
18              relu              64              0.2             50
2               relu              16              0.2             50
8               relu              32              0.2             20
0               relu              16              0.2             20
12              relu              32              0.3             20
16              relu              64              0.2             20
4               relu              16              0.3             20
20              relu              64              0.3             20
30              tanh              16              0.3             50
26              tanh              16              0.2             50
38              tanh              32              0.3             50
34              tanh              32              0.2             50
3               relu              16              0.2             50
7               relu              16              0.3             50
28              tanh              16              0.3             20
42              tanh              64              0.2             50

   param_optimizer  mean_test_score  std_test_score  rank_test_score
10            adam          -0.0125          0.0006                1
22            adam          -0.0126          0.0004                2
6             adam          -0.0128          0.0003                3
14            adam          -0.0129          0.0007                4
18            adam          -0.0130          0.0005                5
2             adam          -0.0134          0.0006                6
8             adam          -0.0136          0.0008                7
0             adam          -0.0137          0.0016                8
12            adam          -0.0140          0.0011                9
16            adam          -0.0142          0.0008               10
4             adam          -0.0143          0.0011               11
20            adam          -0.0144          0.0009               12
30            adam          -0.0165          0.0010               13
26            adam          -0.0168          0.0006               14
38            adam          -0.0170          0.0008               15
34            adam          -0.0177          0.0009               16
3              sgd          -0.0192          0.0013               17
7              sgd          -0.0194          0.0007               18
28            adam          -0.0196          0.0008               19
42            adam          -0.0196          0.0003               20
```

1.7 Model evaluation

Compare the performance of the tuned model with the baseline model (i.e., the initial model without any hyperparameter tuning).

[ ]:
```
# best params identified via grid search
activation = 'relu'
batch_size = 16
```

```
dropout_rate = 0.2
epochs = 50
optimizer = 'adam'
```

[ ]:
```
tuned_regression_model = create_regression_model(activation=activation,␣
 ↪optimizer=optimizer, dropout_rate=dropout_rate)
```

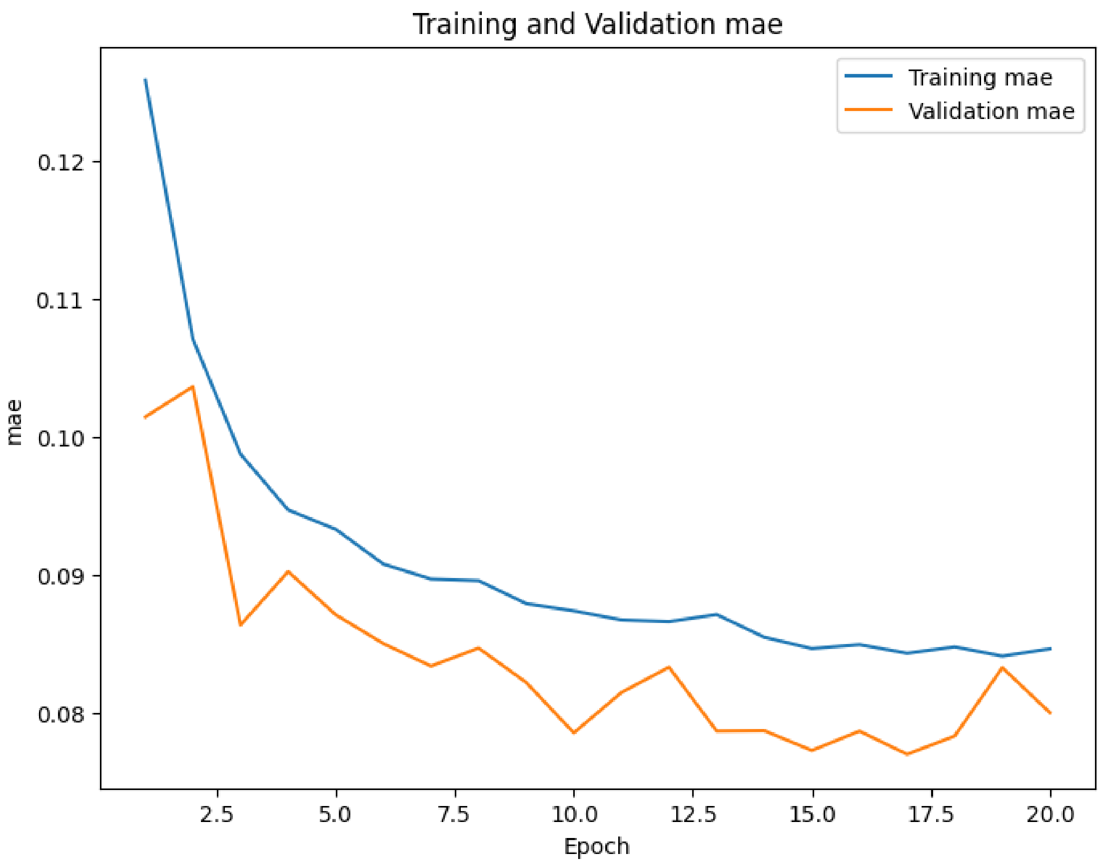
[ ]:
```
model_checkpoint = ModelCheckpoint('tuned_regression.h5', save_best_only=True)

history = tuned_regression_model.fit(X_train_scaled, y_train_scaled,␣
 ↪epochs=epochs, batch_size=batch_size, validation_split=0.2, verbose=1,␣
 ↪callbacks=[early_stopping, model_checkpoint])
```

```
Epoch 1/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0295 - mae:
0.1259 - val_loss: 0.0203 - val_mae: 0.1015
Epoch 2/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0221 - mae:
0.1071 - val_loss: 0.0238 - val_mae: 0.1037
Epoch 3/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0192 - mae:
0.0988 - val_loss: 0.0159 - val_mae: 0.0864
Epoch 4/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0177 - mae:
0.0947 - val_loss: 0.0147 - val_mae: 0.0903
Epoch 5/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0172 - mae:
0.0933 - val_loss: 0.0145 - val_mae: 0.0871
Epoch 6/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0165 - mae:
0.0908 - val_loss: 0.0138 - val_mae: 0.0850
Epoch 7/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0161 - mae:
0.0897 - val_loss: 0.0145 - val_mae: 0.0834
Epoch 8/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0161 - mae:
0.0896 - val_loss: 0.0135 - val_mae: 0.0847
Epoch 9/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0155 - mae:
0.0879 - val_loss: 0.0129 - val_mae: 0.0822
Epoch 10/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0153 - mae:
0.0874 - val_loss: 0.0128 - val_mae: 0.0786
Epoch 11/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0151 - mae:
0.0868 - val_loss: 0.0141 - val_mae: 0.0815
Epoch 12/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0152 - mae:
```

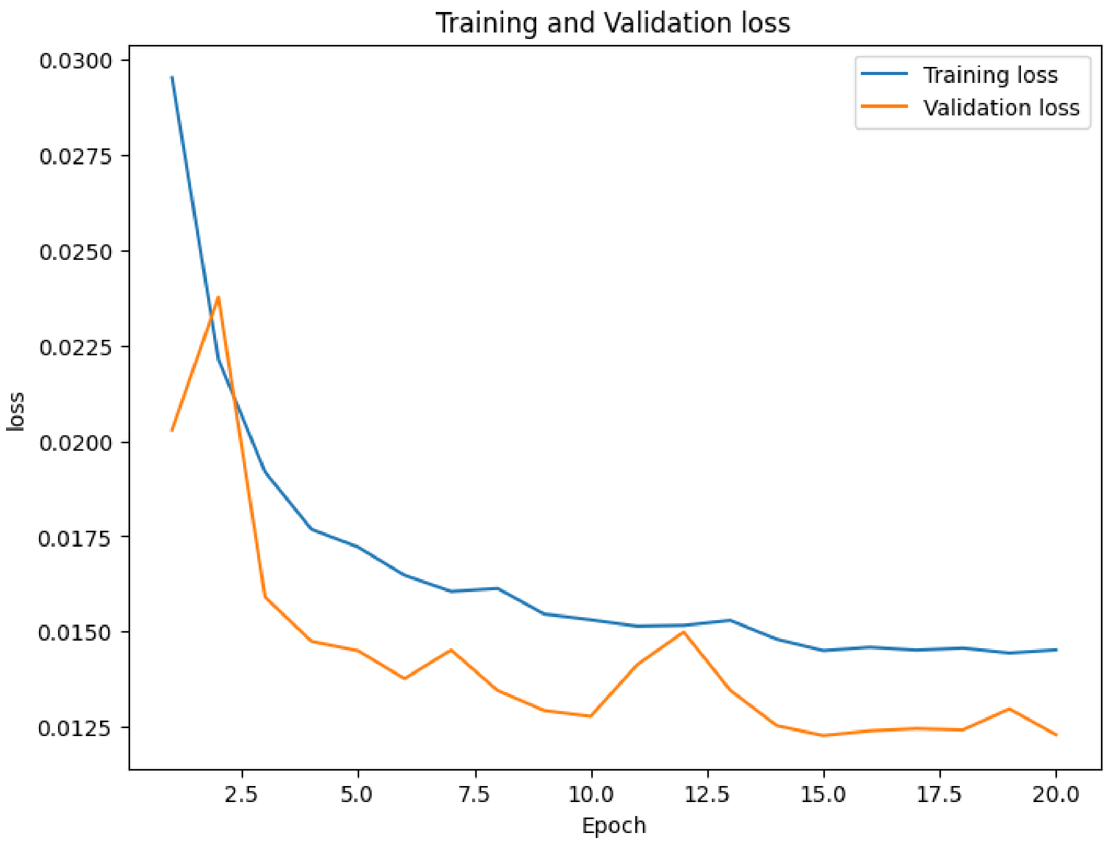
```
0.0866 - val_loss: 0.0150 - val_mae: 0.0833
Epoch 13/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0153 - mae:
0.0871 - val_loss: 0.0135 - val_mae: 0.0787
Epoch 14/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0148 - mae:
0.0855 - val_loss: 0.0125 - val_mae: 0.0787
Epoch 15/50
826/826 [==============================] - 1s 998us/step - loss: 0.0145 - mae:
0.0847 - val_loss: 0.0123 - val_mae: 0.0773
Epoch 16/50
826/826 [==============================] - 1s 998us/step - loss: 0.0146 - mae:
0.0850 - val_loss: 0.0124 - val_mae: 0.0787
Epoch 17/50
826/826 [==============================] - 1s 990us/step - loss: 0.0145 - mae:
0.0843 - val_loss: 0.0125 - val_mae: 0.0770
Epoch 18/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0146 - mae:
0.0848 - val_loss: 0.0124 - val_mae: 0.0783
Epoch 19/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0144 - mae:
0.0841 - val_loss: 0.0130 - val_mae: 0.0833
Epoch 20/50
826/826 [==============================] - 1s 1ms/step - loss: 0.0145 - mae:
0.0847 - val_loss: 0.0123 - val_mae: 0.0800
```

1.7.1 Tuned model training perf

[ ]:
```python
training_plots(history, 'mae')
```




```
[ ]: training_plots(history, 'loss')
```



1.7.2 Tuned model testing perf

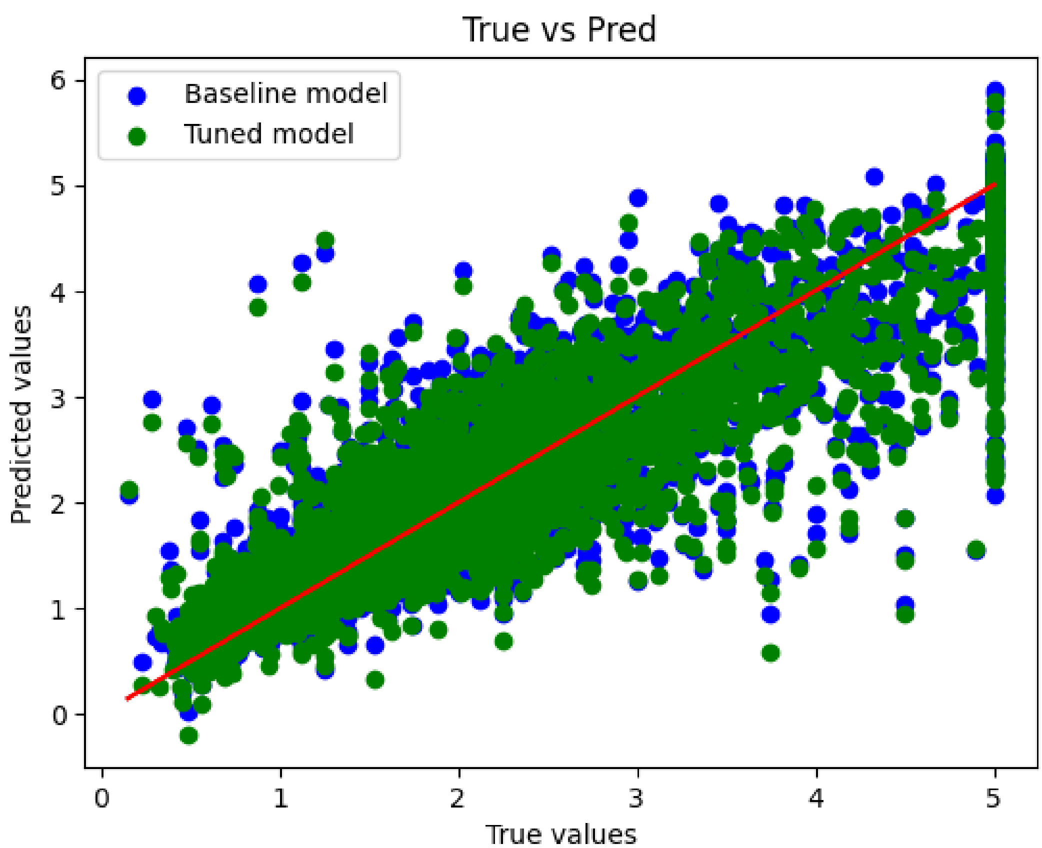
[ ]:
```
baseline_loss, baseline_mae = baseline_regression_model.evaluate(X_test_scaled,␣
 ↪y_test_scaled, verbose=0)
tuned_loss, tuned_mae = tuned_regression_model.evaluate(X_test_scaled,␣
 ↪y_test_scaled, verbose=0)

print(f'Baseline model test loss: {baseline_loss:.4f}')
print(f'Tuned model test loss: {tuned_loss:.4f}\n')

print(f'Baseline model test MAE: {baseline_mae:.4f}')
print(f'Tuned model test MAE: {tuned_mae:.4f}')
```

```
Baseline model test loss: 0.0123
Tuned model test loss: 0.0129

Baseline model test MAE: 0.0757
Tuned model test MAE: 0.0787
```

[ ]:
```
y_pred_baseline = baseline_regression_model.predict(X_test_scaled)
y_pred_tuned = tuned_regression_model.predict(X_test_scaled)

y_pred_baseline_inv = scaler_y.inverse_transform(y_pred_baseline)
y_pred_tuned_inv = scaler_y.inverse_transform(y_pred_tuned)
```

```
  1/129 […] - ETA: 1s129/129
[==============================] - 0s 643us/step
129/129 [==============================] - 0s 585us/step
```

[ ]:
```
testing_plot(y_test, y_pred_baseline_inv)
testing_plot(y_test, y_pred_tuned_inv, tuned=1)
plt.legend()
plt.show()
```



2 Classification problem

2.1 Load data

Load the Boston Housing dataset from the Keras library.

No new data will be loaded as the data loaded for the regression problem is the same as the data required for the classification problem.

2.2 EDA

Explore and preprocess the data (e.g., normalization, one-hot encoding, etc.).

EDA and preprocessing completed within the scope of the regression setup.

2.3 Binary target

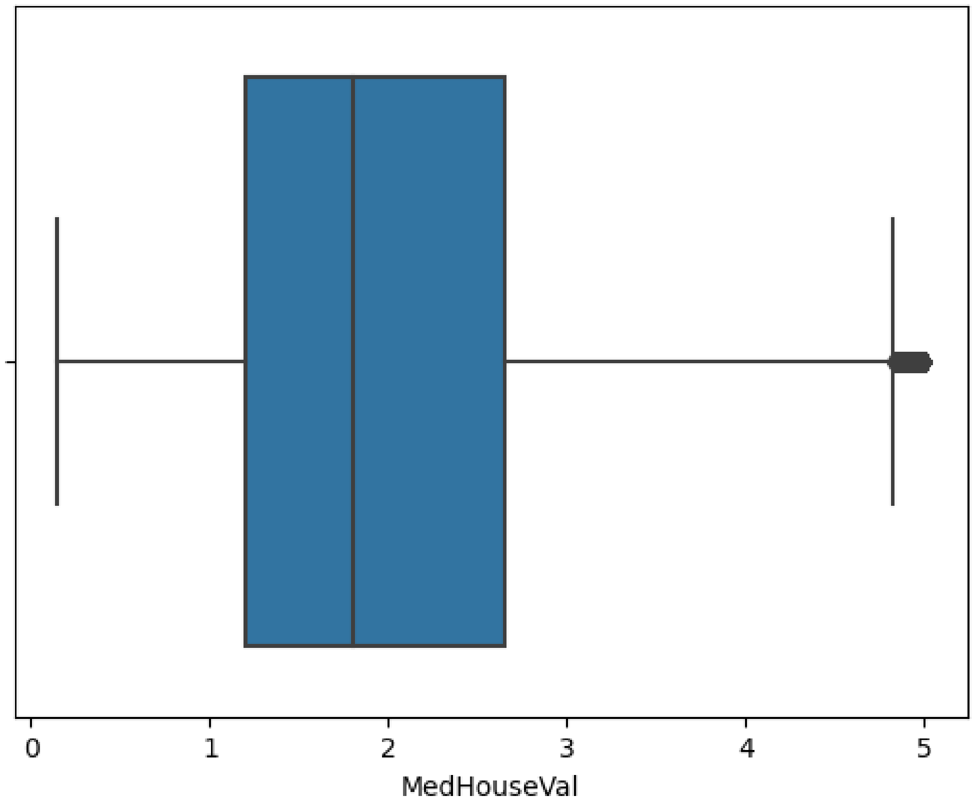
Convert the target variable into a binary variable (i.e., expensive or not expensive).

2.3.1 Investigate the 'breakpoint' for expensive vs non-expensive

[ ]:
```
full_df['MedHouseVal'].describe()
```

[ ]:
```
count    20632.000000
mean         2.068538
std          1.153873
min          0.149990
25%          1.196000
50%          1.797000
75%          2.647250
max          5.000010
Name: MedHouseVal, dtype: float64
```

[ ]:
```
sns.boxplot(x=full_df['MedHouseVal'])
plt.show()
```



Expensive houses will be MedHouseVal > 3

```
[ ]: # create new binary target 'expensive'
expensive = 3

full_df['expensive'] = full_df['MedHouseVal'].apply(lambda x: 1 if x >␣
 ↪expensive else 0)
```

2.4 TTS

Split the data into training and testing sets.

```
[ ]: X = full_df.drop(['MedHouseVal', 'expensive'], axis=1)
y = full_df['expensive']
```

```
[ ]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=97)
```

2.4.1 Feature scaling

[ ]:
```
scaler_x = MinMaxScaler()

X_train_scaled = scaler_x.fit_transform(X_train)
X_test_scaled = scaler_x.transform(X_test)
```

2.5 Define model

Define a deep neural network architecture for classification using Keras.

[ ]:
```
def create_classification_model(optimizer='adam', activation='relu',␣
 ↪dropout_rate=0.2):
    """
    Function to create a DNN classifier
    Inputs: Optimizer - model optimizer, default: adam
            actiavtion - activation function for hidden layers, default: relu
            dropout_rate - dropout rate for hidden layer, default: 0.2
    Returns: compiled model
    """
    model = Sequential()
    model.add(Dense(64, activation=activation, input_shape=(X_train_scaled.
 ↪shape[1],)))
    model.add(Dense(64, activation=activation))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1, activation='sigmoid'))

    model.compile(optimizer=optimizer, loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

    return model
```

2.6 Train and test model

Train the model on the training set and evaluate its performance on the testing set.

[ ]:
```
baseline_classification_model = create_classification_model()
```

[ ]:
```
epochs = 50
batch_size = 32

early_stopping = EarlyStopping(monitor='val_loss', patience=5,␣
 ↪restore_best_weights=True)
model_checkpoint = ModelCheckpoint('baseline_regression.h5',␣
 ↪save_best_only=True)
```

```
history = baseline_classification_model.fit(X_train_scaled, y_train,␣
↪epochs=epochs, batch_size=batch_size, validation_split=0.2, verbose=1,␣
↪callbacks=[early_stopping, model_checkpoint])
```

```
Epoch 1/50
```

```
c:\Users\Reed Oken\AppData\Local\Programs\Python\Python310\lib\site-
packages\keras\engine\data_adapter.py:1700: FutureWarning: The behavior of
`series[i:j]` with an integer-dtype index is deprecated. In a future version,
this will be treated as *label-based* indexing, consistent with e.g. `series[i]`
lookups. To retain the old behavior, use `series.iloc[i:j]`. To get the future
behavior, use `series.loc[i:j]`.
  return t[start:end]
```

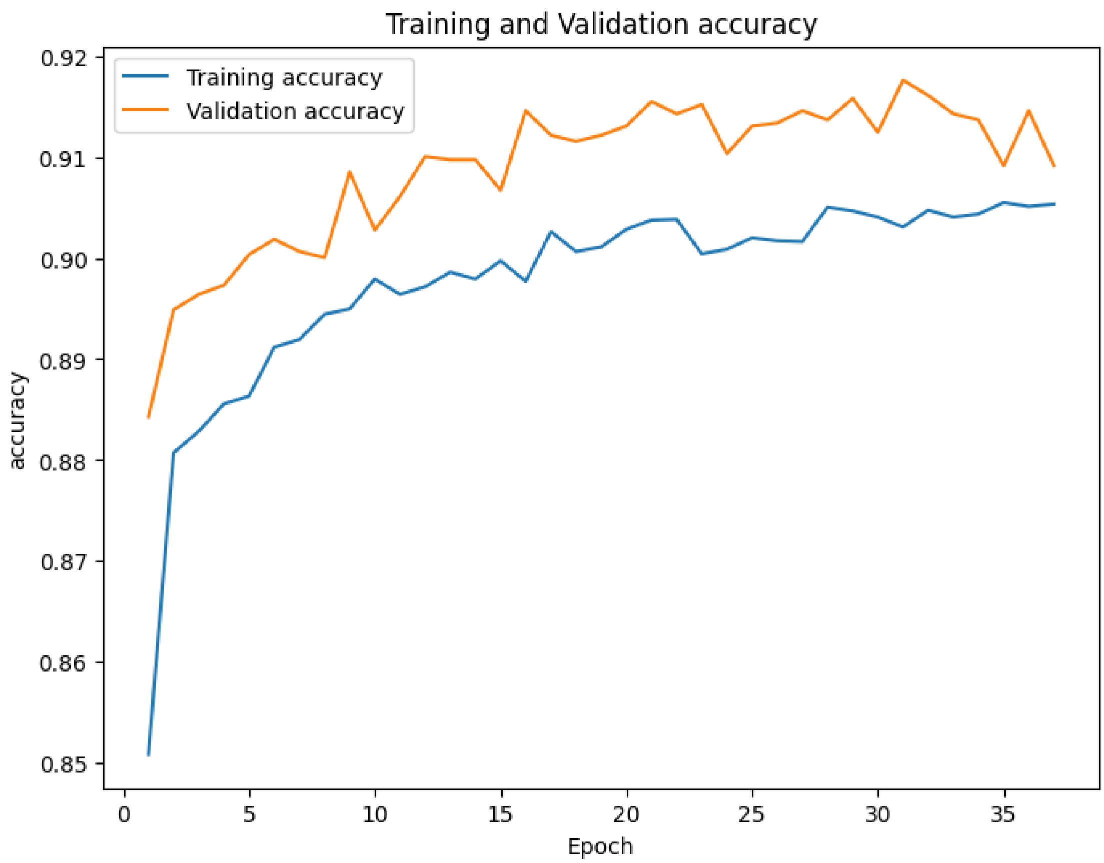
```
413/413 [==============================] - 1s 1ms/step - loss: 0.3683 -
accuracy: 0.8509 - val_loss: 0.2877 - val_accuracy: 0.8843
Epoch 2/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2974 -
accuracy: 0.8807 - val_loss: 0.2733 - val_accuracy: 0.8949
Epoch 3/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2912 -
accuracy: 0.8828 - val_loss: 0.2638 - val_accuracy: 0.8964
Epoch 4/50
413/413 [==============================] - 1s 1ms/step - loss: 0.2817 -
accuracy: 0.8856 - val_loss: 0.2614 - val_accuracy: 0.8973
Epoch 5/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2763 -
accuracy: 0.8863 - val_loss: 0.2476 - val_accuracy: 0.9003
Epoch 6/50
413/413 [==============================] - 1s 1ms/step - loss: 0.2680 -
accuracy: 0.8912 - val_loss: 0.2440 - val_accuracy: 0.9018
Epoch 7/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2595 -
accuracy: 0.8919 - val_loss: 0.2412 - val_accuracy: 0.9006
Epoch 8/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2562 -
accuracy: 0.8944 - val_loss: 0.2488 - val_accuracy: 0.9000
Epoch 9/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2553 -
accuracy: 0.8950 - val_loss: 0.2316 - val_accuracy: 0.9085
Epoch 10/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2510 -
accuracy: 0.8979 - val_loss: 0.2493 - val_accuracy: 0.9028
Epoch 11/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2509 -
accuracy: 0.8964 - val_loss: 0.2358 - val_accuracy: 0.9061
Epoch 12/50
413/413 [==============================] - 1s 1ms/step - loss: 0.2492 -
accuracy: 0.8972 - val_loss: 0.2285 - val_accuracy: 0.9100
```

```
Epoch 13/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2458 -
accuracy: 0.8986 - val_loss: 0.2247 - val_accuracy: 0.9097
Epoch 14/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2438 -
accuracy: 0.8979 - val_loss: 0.2248 - val_accuracy: 0.9097
Epoch 15/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2448 -
accuracy: 0.8997 - val_loss: 0.2377 - val_accuracy: 0.9067
Epoch 16/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2449 -
accuracy: 0.8977 - val_loss: 0.2227 - val_accuracy: 0.9146
Epoch 17/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2413 -
accuracy: 0.9026 - val_loss: 0.2246 - val_accuracy: 0.9121
Epoch 18/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2427 -
accuracy: 0.9006 - val_loss: 0.2259 - val_accuracy: 0.9115
Epoch 19/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2398 -
accuracy: 0.9011 - val_loss: 0.2218 - val_accuracy: 0.9121
Epoch 20/50
413/413 [==============================] - 1s 1ms/step - loss: 0.2385 -
accuracy: 0.9028 - val_loss: 0.2201 - val_accuracy: 0.9131
Epoch 21/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2380 -
accuracy: 0.9037 - val_loss: 0.2194 - val_accuracy: 0.9155
Epoch 22/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2374 -
accuracy: 0.9038 - val_loss: 0.2213 - val_accuracy: 0.9143
Epoch 23/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2399 -
accuracy: 0.9004 - val_loss: 0.2193 - val_accuracy: 0.9152
Epoch 24/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2387 -
accuracy: 0.9009 - val_loss: 0.2231 - val_accuracy: 0.9103
Epoch 25/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2386 -
accuracy: 0.9020 - val_loss: 0.2202 - val_accuracy: 0.9131
Epoch 26/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2352 -
accuracy: 0.9017 - val_loss: 0.2208 - val_accuracy: 0.9134
Epoch 27/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2365 -
accuracy: 0.9016 - val_loss: 0.2179 - val_accuracy: 0.9146
Epoch 28/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2333 -
accuracy: 0.9050 - val_loss: 0.2193 - val_accuracy: 0.9137
```

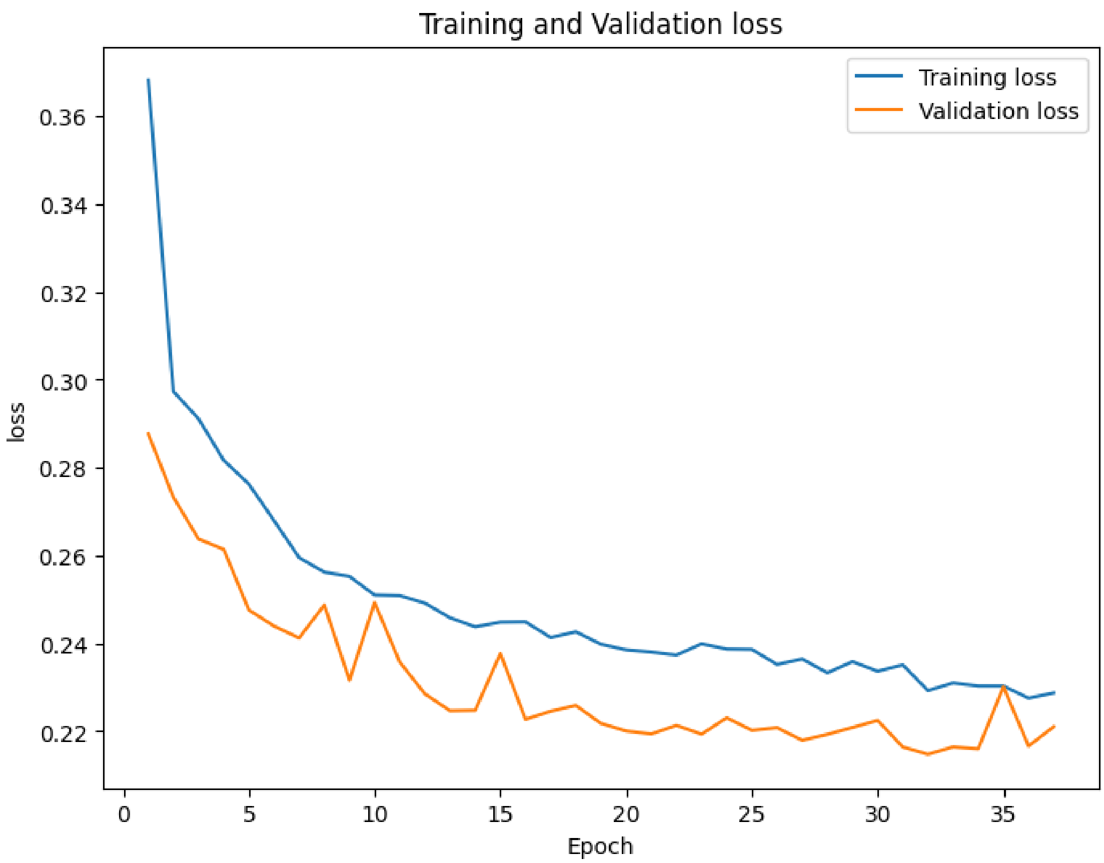
```
Epoch 29/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2359 -
accuracy: 0.9047 - val_loss: 0.2209 - val_accuracy: 0.9158
Epoch 30/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2337 -
accuracy: 0.9040 - val_loss: 0.2225 - val_accuracy: 0.9125
Epoch 31/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2351 -
accuracy: 0.9031 - val_loss: 0.2164 - val_accuracy: 0.9176
Epoch 32/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2293 -
accuracy: 0.9047 - val_loss: 0.2148 - val_accuracy: 0.9161
Epoch 33/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2310 -
accuracy: 0.9040 - val_loss: 0.2164 - val_accuracy: 0.9143
Epoch 34/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2303 -
accuracy: 0.9043 - val_loss: 0.2160 - val_accuracy: 0.9137
Epoch 35/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2303 -
accuracy: 0.9055 - val_loss: 0.2301 - val_accuracy: 0.9091
Epoch 36/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2275 -
accuracy: 0.9051 - val_loss: 0.2166 - val_accuracy: 0.9146
Epoch 37/50
413/413 [==============================] - 0s 1ms/step - loss: 0.2287 -
accuracy: 0.9053 - val_loss: 0.2210 - val_accuracy: 0.9091
```

2.6.1 Baseline training progression plots

[ ]:
```
training_plots(history, 'accuracy')
```



[ ]:
```
training_plots(history, 'loss')
```



2.6.2 Baseline testing performance

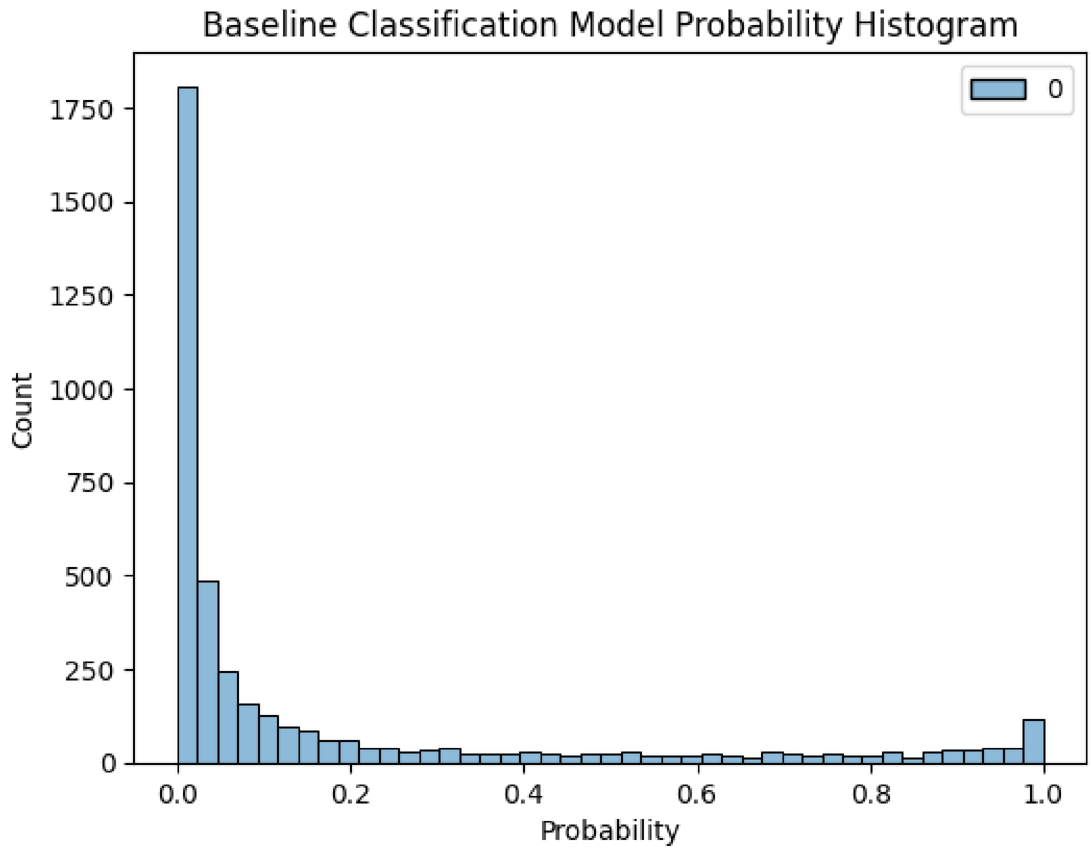
[ ]:
```
loss, accuracy = baseline_classification_model.evaluate(X_test_scaled, y_test,␣
  ↪verbose=0)
print(f'Test Loss: {loss:.4f}')
print(f'Test accuracy: {accuracy:.4f}')
```

```
Test Loss: 0.2132
Test accuracy: 0.9159
```

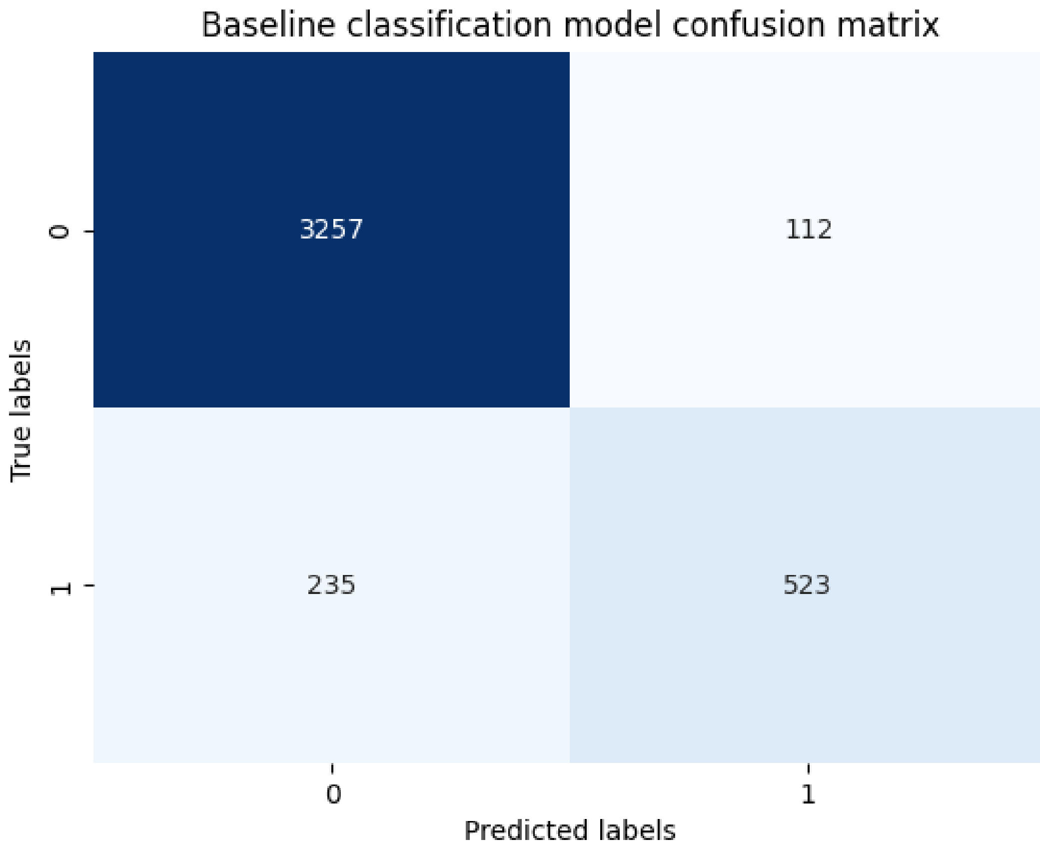
[ ]:
```
predictions = baseline_classification_model.predict(X_test_scaled)
threshold = 0.5
y_pred_baseline = np.where(predictions >= threshold, 1, 0)
```

```
129/129 [==============================] - 0s 580us/step
```

[ ]:
```
sns.histplot(predictions)
plt.title('Baseline Classification Model Probability Histogram')
plt.xlabel('Probability')
plt.show()
```



[ ]:
```
confusion_mat = confusion_matrix(y_test, y_pred_baseline)

sns.heatmap(confusion_mat, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Baseline classification model confusion matrix')
plt.show()
```




```
[ ]: baseline_acc = accuracy_score(y_test, y_pred_baseline)
     baseline_f1 = f1_score(y_test, y_pred_baseline)
     baseline_precision = precision_score(y_test, y_pred_baseline)
     baseline_recall = recall_score(y_test, y_pred_baseline)
```

```
[ ]: print(f'Baseline classification model test accuracy: {baseline_acc:.4f}')
     print(f'Baseline classification model test precision: {baseline_precision:.4f}')
     print(f'Baseline classification model test recall: {baseline_recall:.4f}')
     print(f'Baseline classification model test F1: {baseline_f1:.4f}')
```

```
Baseline classification model test accuracy: 0.9159
Baseline classification model test precision: 0.8236
Baseline classification model test recall: 0.6900
Baseline classification model test F1: 0.7509
```

2.7 Parameter tuning

Tune the hyperparameters of the model to achieve better performance (e.g., number of hidden layers, activation functions, learning rate, number of epochs, etc.).

[ ]:
```
tuning_classification_model =␣
↪KerasClassifier(build_fn=create_classification_model, verbose=0)
```

```
C:\Users\Reed Oken\AppData\Local\Temp\ipykernel_16472\3533025257.py:1:
DeprecationWarning: KerasClassifier is deprecated, use Sci-Keras
(https://github.com/adriangb/scikeras) instead. See
https://www.adriangb.com/scikeras/stable/migration.html for help migrating.
  tuning_classification_model =
KerasClassifier(build_fn=create_classification_model, verbose=0)
```

[ ]:
```
param_grid = {
    'optimizer': ['adam', 'sgd'],
    'activation': ['relu', 'tanh'],
    'batch_size': [16, 32, 64],
    'epochs': [20, 50],
    'dropout_rate': [0.2, 0.3]
}
```

[ ]:
```
classifier_grid = GridSearchCV(estimator=tuning_classification_model,␣
↪param_grid=param_grid, cv=3, n_jobs=-1)

model_checkpoint = ModelCheckpoint('tuning_classification.h5',␣
↪save_best_only=True)
```

[ ]:
```
grid_result = classifier_grid.fit(X_train_scaled, y_train,␣
↪callbacks=[early_stopping], validation_split=0.2)
```

[ ]:
```
gs_classification_df = pd.DataFrame(grid_result.cv_results_)
gs_classification_df = gs_classification_df.drop(columns=['mean_fit_time',␣
↪'std_fit_time', 'mean_score_time', 'std_score_time', 'params',␣
↪'split0_test_score', 'split1_test_score', 'split2_test_score'])

gs_classification_df = gs_classification_df.sort_values(by='rank_test_score',␣
↪ascending=True)
gs_classification_df.head(20).round(4)
```

[ ]:
```
   param_activation param_batch_size param_dropout_rate param_epochs  \
2              relu               16                0.2           50
6              relu               16                0.3           50
10             relu               32                0.2           50
14             relu               32                0.3           50
18             relu               64                0.2           50
22             relu               64                0.3           50
4              relu               16                0.3           20
26             tanh               16                0.2           50
12             relu               32                0.3           20
42             tanh               64                0.2           50
0              relu               16                0.2           20
```

30	tanh	16	0.3	50
8	relu	32	0.2	20
46	tanh	64	0.3	50
38	tanh	32	0.3	50
20	relu	64	0.3	20
24	tanh	16	0.2	20
34	tanh	32	0.2	50
16	relu	64	0.2	20
36	tanh	32	0.3	20

	param_optimizer	mean_test_score	std_test_score	rank_test_score
2	adam	0.9068	0.0058	1
6	adam	0.9052	0.0054	2
10	adam	0.9038	0.0025	3
14	adam	0.9037	0.0033	4
18	adam	0.9031	0.0043	5
22	adam	0.9027	0.0043	6
4	adam	0.9009	0.0048	7
26	adam	0.9000	0.0052	8
12	adam	0.9000	0.0076	9
42	adam	0.8994	0.0056	10
0	adam	0.8994	0.0073	11
30	adam	0.8991	0.0050	12
8	adam	0.8988	0.0031	13
46	adam	0.8986	0.0064	14
38	adam	0.8985	0.0058	15
20	adam	0.8983	0.0065	16
24	adam	0.8982	0.0065	17
34	adam	0.8982	0.0054	18
16	adam	0.8974	0.0071	19
36	adam	0.8971	0.0061	20

2.8 Model evaluation

Compare the performance of the tuned model with the baseline model (i.e., the initial model without any hyperparameter tuning).

While a batch size of 32 and dropout of .3 yielded marginally better test score, + .0008, over 16 batch size, .2 dropout, it did so with over double the standard deviation in test score. For the tuned model, a dropout of .2 and batch size of 16 will be used for the greater consistency.

[ ]:
```
activation = 'relu'
batch_size = 16
dropout_rate = 0.2
epochs = 50
optimizer = 'adam'
```

[ ]:
```
tuned_classification_model = create_classification_model(activation=activation,␣
    ↪optimizer=optimizer, dropout_rate=dropout_rate)
```

[ ]:
```
model_checkpoint = ModelCheckpoint('tuned_classification.h5',␣
    ↪save_best_only=True)

history = tuned_classification_model.fit(X_train_scaled, y_train,␣
    ↪epochs=epochs, batch_size=batch_size, validation_split=0.2, verbose=1,␣
    ↪callbacks=[early_stopping, model_checkpoint])
```

```
Epoch 1/50
```

```
c:\Users\Reed Oken\AppData\Local\Programs\Python\Python310\lib\site-
packages\keras\engine\data_adapter.py:1700: FutureWarning: The behavior of
`series[i:j]` with an integer-dtype index is deprecated. In a future version,
this will be treated as *label-based* indexing, consistent with e.g. `series[i]`
lookups. To retain the old behavior, use `series.iloc[i:j]`. To get the future
behavior, use `series.loc[i:j]`.
  return t[start:end]
```

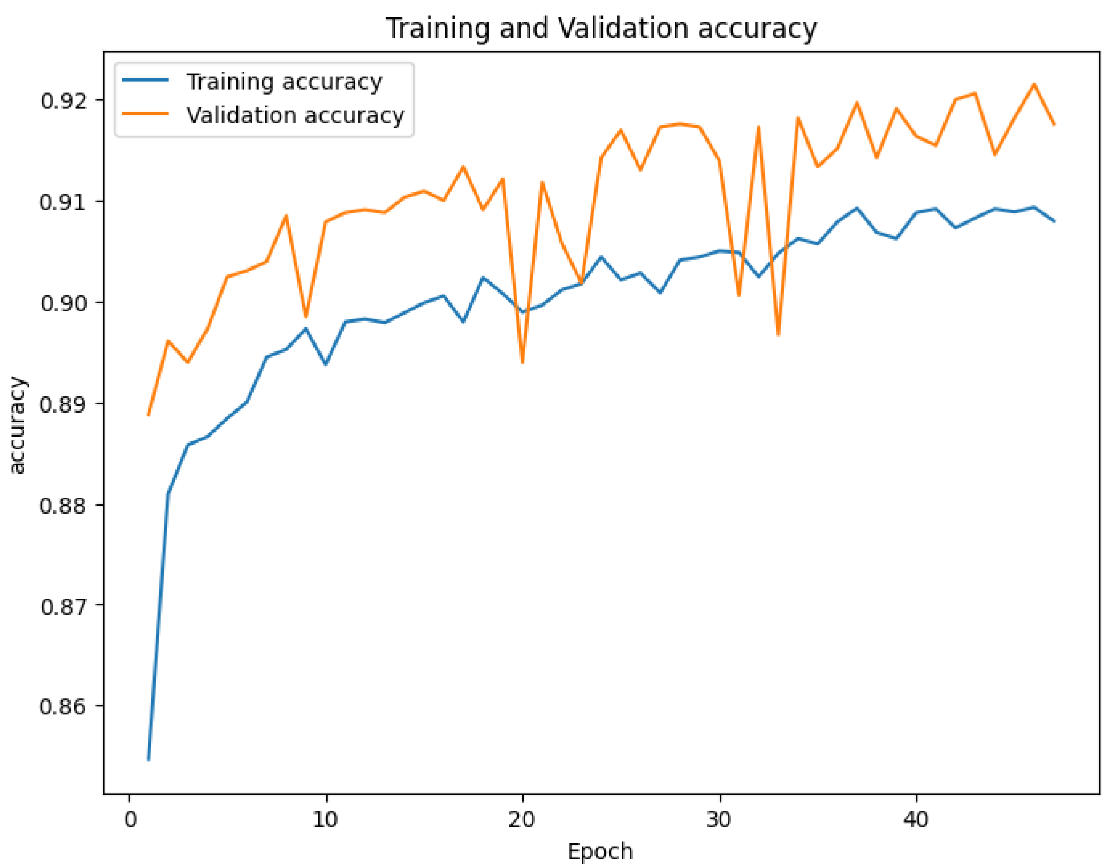
```
826/826 [==============================] - 2s 1ms/step - loss: 0.3517 -
accuracy: 0.8547 - val_loss: 0.2826 - val_accuracy: 0.8888
Epoch 2/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2975 -
accuracy: 0.8809 - val_loss: 0.2690 - val_accuracy: 0.8961
Epoch 3/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2854 -
accuracy: 0.8858 - val_loss: 0.2666 - val_accuracy: 0.8940
Epoch 4/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2788 -
accuracy: 0.8866 - val_loss: 0.2572 - val_accuracy: 0.8973
Epoch 5/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2706 -
accuracy: 0.8884 - val_loss: 0.2449 - val_accuracy: 0.9025
Epoch 6/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2645 -
accuracy: 0.8900 - val_loss: 0.2374 - val_accuracy: 0.9031
Epoch 7/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2577 -
accuracy: 0.8945 - val_loss: 0.2366 - val_accuracy: 0.9040
Epoch 8/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2547 -
accuracy: 0.8953 - val_loss: 0.2336 - val_accuracy: 0.9085
Epoch 9/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2535 -
accuracy: 0.8973 - val_loss: 0.2425 - val_accuracy: 0.8985
Epoch 10/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2534 -
accuracy: 0.8937 - val_loss: 0.2316 - val_accuracy: 0.9079
```

```
Epoch 11/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2510 -
accuracy: 0.8980 - val_loss: 0.2272 - val_accuracy: 0.9088
Epoch 12/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2494 -
accuracy: 0.8983 - val_loss: 0.2371 - val_accuracy: 0.9091
Epoch 13/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2468 -
accuracy: 0.8979 - val_loss: 0.2306 - val_accuracy: 0.9088
Epoch 14/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2485 -
accuracy: 0.8989 - val_loss: 0.2240 - val_accuracy: 0.9103
Epoch 15/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2444 -
accuracy: 0.8999 - val_loss: 0.2240 - val_accuracy: 0.9109
Epoch 16/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2431 -
accuracy: 0.9006 - val_loss: 0.2265 - val_accuracy: 0.9100
Epoch 17/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2441 -
accuracy: 0.8980 - val_loss: 0.2230 - val_accuracy: 0.9134
Epoch 18/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2402 -
accuracy: 0.9024 - val_loss: 0.2251 - val_accuracy: 0.9091
Epoch 19/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2430 -
accuracy: 0.9008 - val_loss: 0.2245 - val_accuracy: 0.9121
Epoch 20/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2400 -
accuracy: 0.8990 - val_loss: 0.2578 - val_accuracy: 0.8940
Epoch 21/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2392 -
accuracy: 0.8997 - val_loss: 0.2225 - val_accuracy: 0.9118
Epoch 22/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2378 -
accuracy: 0.9012 - val_loss: 0.2341 - val_accuracy: 0.9058
Epoch 23/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2382 -
accuracy: 0.9018 - val_loss: 0.2495 - val_accuracy: 0.9018
Epoch 24/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2336 -
accuracy: 0.9044 - val_loss: 0.2172 - val_accuracy: 0.9143
Epoch 25/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2340 -
accuracy: 0.9022 - val_loss: 0.2197 - val_accuracy: 0.9170
Epoch 26/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2366 -
accuracy: 0.9028 - val_loss: 0.2196 - val_accuracy: 0.9131
```

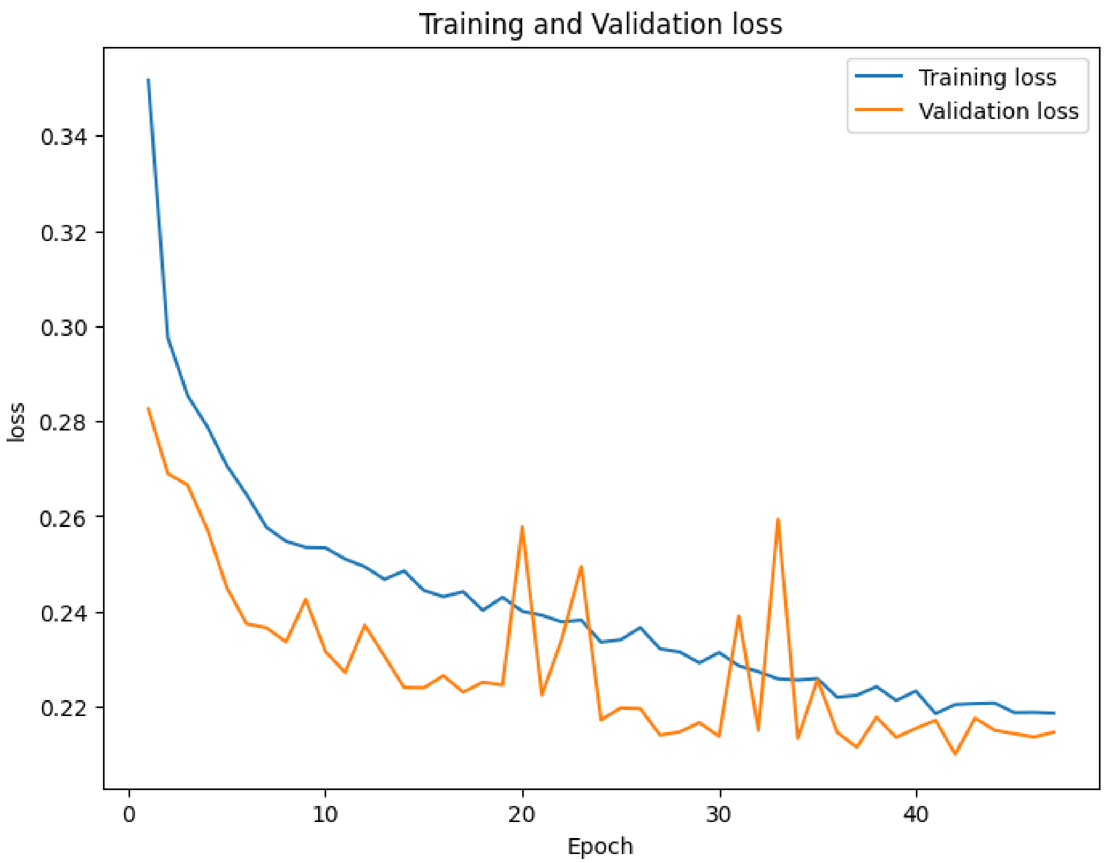
```
Epoch 27/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2321 -
accuracy: 0.9009 - val_loss: 0.2140 - val_accuracy: 0.9173
Epoch 28/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2315 -
accuracy: 0.9041 - val_loss: 0.2147 - val_accuracy: 0.9176
Epoch 29/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2292 -
accuracy: 0.9044 - val_loss: 0.2166 - val_accuracy: 0.9173
Epoch 30/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2314 -
accuracy: 0.9050 - val_loss: 0.2137 - val_accuracy: 0.9140
Epoch 31/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2286 -
accuracy: 0.9049 - val_loss: 0.2390 - val_accuracy: 0.9006
Epoch 32/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2273 -
accuracy: 0.9025 - val_loss: 0.2150 - val_accuracy: 0.9173
Epoch 33/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2258 -
accuracy: 0.9048 - val_loss: 0.2594 - val_accuracy: 0.8967
Epoch 34/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2256 -
accuracy: 0.9062 - val_loss: 0.2133 - val_accuracy: 0.9182
Epoch 35/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2259 -
accuracy: 0.9057 - val_loss: 0.2256 - val_accuracy: 0.9134
Epoch 36/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2220 -
accuracy: 0.9079 - val_loss: 0.2146 - val_accuracy: 0.9152
Epoch 37/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2224 -
accuracy: 0.9093 - val_loss: 0.2114 - val_accuracy: 0.9197
Epoch 38/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2242 -
accuracy: 0.9068 - val_loss: 0.2178 - val_accuracy: 0.9143
Epoch 39/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2213 -
accuracy: 0.9062 - val_loss: 0.2135 - val_accuracy: 0.9191
Epoch 40/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2233 -
accuracy: 0.9088 - val_loss: 0.2154 - val_accuracy: 0.9164
Epoch 41/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2185 -
accuracy: 0.9092 - val_loss: 0.2171 - val_accuracy: 0.9155
Epoch 42/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2204 -
accuracy: 0.9073 - val_loss: 0.2100 - val_accuracy: 0.9200
```

```
Epoch 43/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2206 -
accuracy: 0.9083 - val_loss: 0.2176 - val_accuracy: 0.9206
Epoch 44/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2207 -
accuracy: 0.9092 - val_loss: 0.2150 - val_accuracy: 0.9146
Epoch 45/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2187 -
accuracy: 0.9089 - val_loss: 0.2143 - val_accuracy: 0.9182
Epoch 46/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2188 -
accuracy: 0.9093 - val_loss: 0.2136 - val_accuracy: 0.9215
Epoch 47/50
826/826 [==============================] - 1s 1ms/step - loss: 0.2186 -
accuracy: 0.9080 - val_loss: 0.2146 - val_accuracy: 0.9176
```

[ ]:
```
training_plots(history, 'accuracy')
```



[ ]:
```
training_plots(history, 'loss')
```




```
[ ]: loss, accuracy = tuned_classification_model.evaluate(X_test_scaled, y_test,␣
     ↪verbose=0)
     print(f'Test Loss: {loss:.4f}')
     print(f'Test accuracy: {accuracy:.4f}')
```

```
Test Loss: 0.2053
Test accuracy: 0.9193
```

```
[ ]: predictions = tuned_classification_model.predict(X_test_scaled)
     threshold = 0.5
     y_pred = np.where(predictions >= threshold, 1, 0)
```

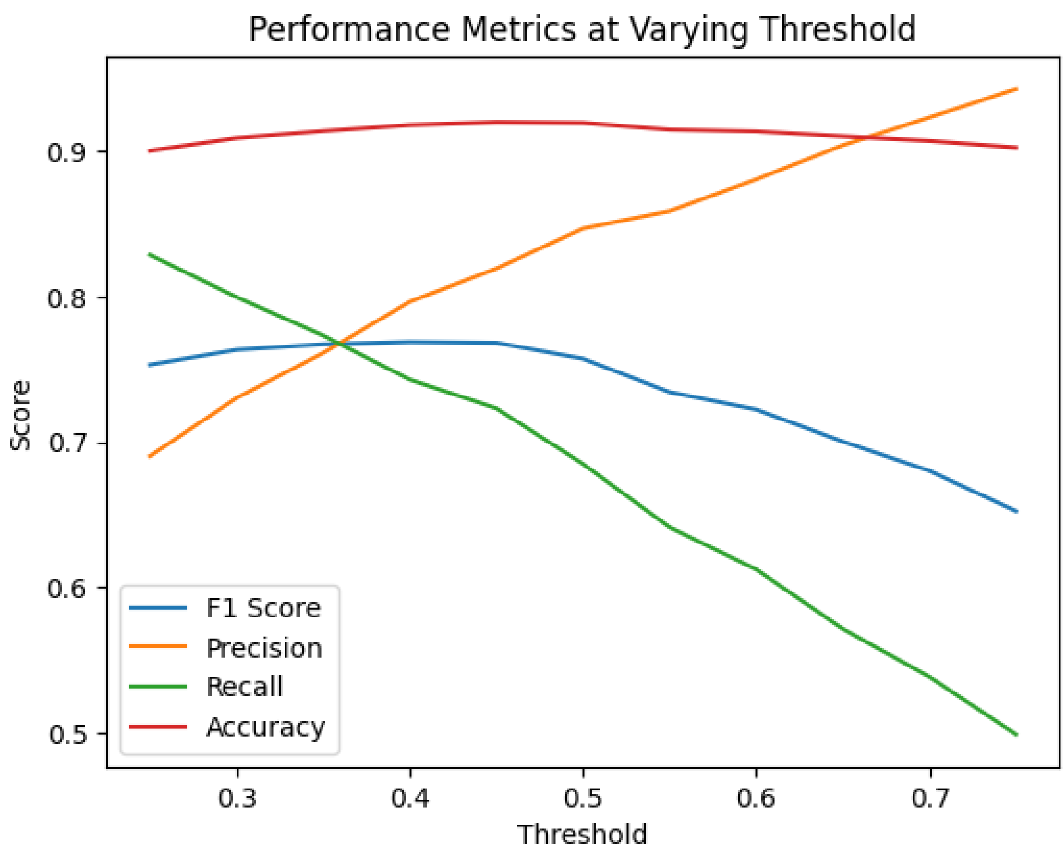
```
129/129 [==============================] - 0s 600us/step
```

2.8.1 Threshold tuning

```
[ ]: thresh_arr = []
     prec_arr = []
     recall_arr = []
     f1_arr = []
```

```
acc_arr = []

for i in range(11):
    thresh = .25 + i*.05
    thresh_arr.append(thresh)
    y_pred = np.where(predictions >= thresh, 1, 0)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    acc = accuracy_score(y_test, y_pred)
    prec_arr.append(precision)
    recall_arr.append(recall)
    f1_arr.append(f1)
    acc_arr.append(acc)
```

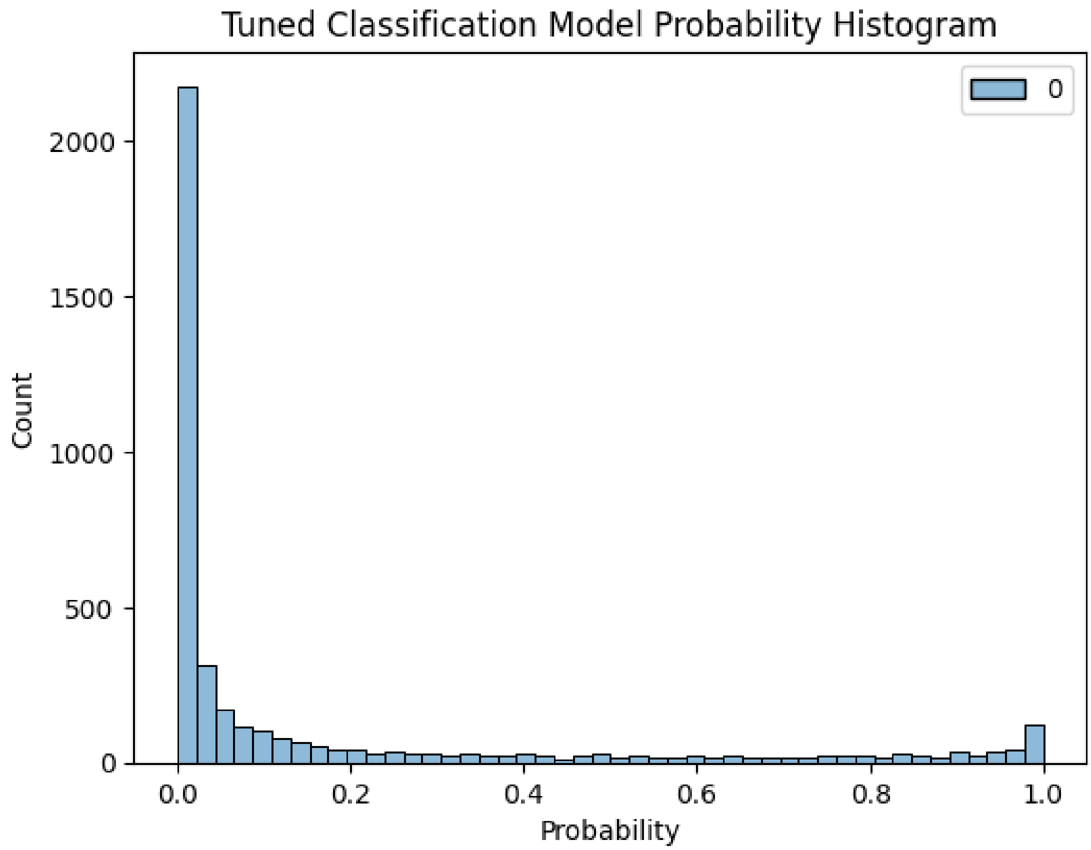
[ ]:
```
sns.lineplot(x=thresh_arr, y=f1_arr, label='F1 Score')
sns.lineplot(x=thresh_arr, y=prec_arr, label='Precision')
sns.lineplot(x=thresh_arr, y=recall_arr, label='Recall')
sns.lineplot(x=thresh_arr, y=acc_arr, label='Accuracy')
plt.legend()
plt.xlabel('Threshold')
plt.ylabel('Score')
plt.title('Performance Metrics at Varying Threshold')
plt.show()
```



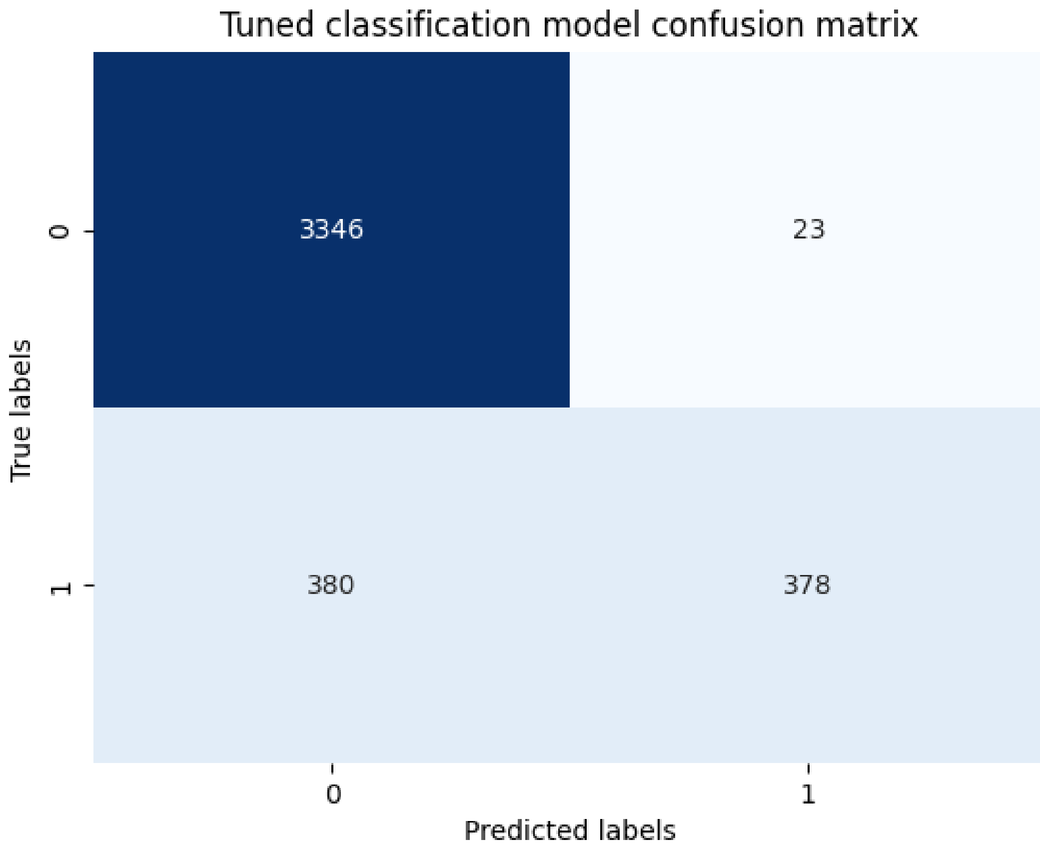
Threshold of .5 appears to provide the best performance across all performance metrics

2.8.2 Test performance on tuned model

[ ]:
```
sns.histplot(predictions)
plt.title('Tuned Classification Model Probability Histogram')
plt.xlabel('Probability')
plt.show()
```



[ ]:
```
confusion_mat = confusion_matrix(y_test, y_pred)

sns.heatmap(confusion_mat, annot=True, cmap='Blues', fmt='g', cbar=False)
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Tuned classification model confusion matrix')
plt.show()
```



[ ]:
```
print(f'Baseline classification model test accuracy: {baseline_acc:.4f}')
print(f'Tuned classification model test accuracy: {tuned_acc:.4f}\n')

print(f'Baseline classification model test precision: {baseline_precision:.4f}')
print(f'Tuned classification model test precision: {tuned_precision:.4f}\n')

print(f'Baseline classification model test recall: {baseline_recall:.4f}')
print(f'Tuned classification model test recall: {tuned_recall:.4f}\n')

print(f'Baseline classification model test F1: {baseline_f1:.4f}')
print(f'Tuned classification model test F1: {tuned_f1:.4f}')
```

```
Baseline classification model test accuracy: 0.9159
Tuned classification model test accuracy: 0.9198

Baseline classification model test precision: 0.8236
Tuned classification model test precision: 0.8280

Baseline classification model test recall: 0.6900
Tuned classification model test recall: 0.7111
```

```
Baseline classification model test F1: 0.7509
Tuned classification model test F1: 0.7651
```