Import

In [417…
```
import numpy as np
import pandas as pd
import sqlite3
from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.preprocessing import StandardScaler, OneHotEncoder, LabelEncode
from sklearn.impute import SimpleImputer
from sklearn.base import BaseEstimator, TransformerMixin
import pandas as pd
from sklearn.model_selection import train_test_split,learning_curve
import matplotlib.pyplot as plt
from sklearn.metrics import  precision_score, recall_score, f1_score,accura
import seaborn as sns
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline as ImblearnPipeline
from sklearn.model_selection import cross_validate, StratifiedKFold
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import make_pipeline as make_pipeline_imb
from sklearn.metrics import make_scorer
from sklearn.feature_selection import RFE
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from scipy.stats import expon, reciprocal
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import StratifiedKFold
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import StackingClassifier
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.regularizers import l2
from imblearn.pipeline import Pipeline as ImbPipeline
```

In [2]:
```
def plot_learning_curves(model, X, y, cv, train_sizes=np.linspace(0.1, 1.0,
    """
    Plots the learning curve for a given model.

    Parameters:
    - model: sklearn model
    - X: feature matrix
    - y: target vector
    - cv: cross-validation splitting strategy
    - train_sizes: array, list of train sizes to use
    - scoring: scoring strategy

    Returns:
    - None, but plots the learning curve
    """
    train_sizes, train_scores, validation_scores = learning_curve(
        estimator=model,
        X=X,
        y=y,
```

```
        train_sizes=train_sizes,
        cv=cv,
        scoring=scoring,
        n_jobs=-1
    )

    # To calculate mean and standard deviation for training set scores
    train_mean = np.mean(train_scores, axis=1)
    train_std = np.std(train_scores, axis=1)

    # To alculate mean and standard deviation for test set scores
    validation_mean = np.mean(validation_scores, axis=1)
    validation_std = np.std(validation_scores, axis=1)

    plt.plot(train_sizes, train_mean, label='Training score', color='blue',
    plt.fill_between(train_sizes, train_mean - train_std, train_mean + trair

    plt.plot(train_sizes, validation_mean, label='Validation score', color=
    plt.fill_between(train_sizes, validation_mean - validation_std, validat:

    plt.title('Learning Curve')
    plt.xlabel('Training Data Size')
    plt.ylabel('Accuracy Score')
    plt.legend(loc='best')
    plt.grid()
    plt.show()
```

Dataset

In [2]:
```
path = "/Users/mac1/Downloads/data-society-european-soccer-data/original/soc
#database = path + 'database.sqlite'
connection = sqlite3.connect(path)
```

In [3]:
```
# SQL query to select relevant features for match prediction, without the Li
query = """
SELECT
    m.id AS match_id,
    m.date AS match_date,
    m.season,
    ht.team_long_name AS home_team,
    at.team_long_name AS away_team,
    m.home_team_goal,
    m.away_team_goal,
    (SELECT AVG(overall_rating) FROM Player_Attributes
      WHERE player_api_id IN
        (m.home_player_1, m.home_player_2, m.home_player_3, m.home_player_4,
         m.home_player_6, m.home_player_7, m.home_player_8, m.home_player_9,
         AS avg_home_player_rating,
    (SELECT AVG(overall_rating) FROM Player_Attributes
      WHERE player_api_id IN
        (m.away_player_1, m.away_player_2, m.away_player_3, m.away_player_4,
         m.away_player_6, m.away_player_7, m.away_player_8, m.away_player_9,
         AS avg_away_player_rating,
    hta.buildUpPlaySpeed AS home_buildUpPlaySpeed,
    ata.buildUpPlaySpeed AS away_buildUpPlaySpeed,
    hta.defenceAggression AS home_defenceAggression,
    ata.defenceAggression AS away_defenceAggression,
    hta.defencePressure AS home_defencePressure,
    ata.defencePressure AS away_defencePressure
FROM Match m
JOIN League l ON m.league_id = l.id
JOIN Country c ON l.country_id = c.id
```

```
JOIN Team ht ON m.home_team_api_id = ht.team_api_id
JOIN Team at ON m.away_team_api_id = at.team_api_id
LEFT JOIN Team_Attributes hta ON ht.team_api_id = hta.team_api_id AND hta.da
    (SELECT MAX(date) FROM Team_Attributes hta2 WHERE hta.team_api_id = hta2
LEFT JOIN Team_Attributes ata ON at.team_api_id = ata.team_api_id AND ata.da
    (SELECT MAX(date) FROM Team_Attributes ata2 WHERE ata.team_api_id = ata2
WHERE c.name = 'England'
GROUP BY m.id
ORDER BY m.date DESC;

"""
```

In [ ]:

```
# To execute the query and load the data into a DataFrame
df = pd.read_sql_query(query, connection)

# To save the DataFrame to a CSV file
csv_file_path = '/Users/mac1/Downloads/untitled folder 2/match_data3.csv'
df.to_csv(csv_file_path, index=False)

print(f"Data saved to {csv_file_path}")
```

In [3]:

```
df = pd.read_csv('match_data3.csv')
```

In [4]:

```
df.head()
```

Out[4]:

	match_id	match_date	season	home_team	away_team	home_team_goal	away_tea
0	4702	2016-05-17 00:00:00	2015/2016	Manchester United	Bournemouth	3	
1	4699	2016-05-15 00:00:00	2015/2016	Arsenal	Aston Villa	4	
2	4700	2016-05-15 00:00:00	2015/2016	Chelsea	Leicester City	1	
3	4701	2016-05-15 00:00:00	2015/2016	Everton	Norwich City	3	
4	4703	2016-05-15 00:00:00	2015/2016	Newcastle United	Tottenham Hotspur	5	

Feature Engineering

In [5]:

```
def calculate_form(df, team, num_matches):
    """
    Calculate the form of a given team over the last 'num_matches'.
    Form metrics can include win rate, average goals scored, and average goa
    """
    # Filter matches involving the team
    team_matches = df[(df['home_team'] == team) | (df['away_team'] == team)]

    # Calculate win rate, average goals scored/conceded in the last 'num_ma
    recent_matches = team_matches.iloc[-num_matches:]
    wins = recent_matches.apply(lambda row: 1 if ((row['home_team'] == team
                                               (row['away_team'] == tear
    goals_scored = recent_matches.apply(lambda row: row['home_team_goal'] i
    goals_conceded = recent_matches.apply(lambda row: row['away_team_goal']

    win_rate = wins / num_matches
    avg_goals_scored = goals_scored / num_matches
    avg_goals_conceded = goals_conceded / num_matches
```

```
    return win_rate, avg_goals_scored, avg_goals_conceded
```

In [6]:
```
#  Adding form over the last 5 matches to the dataset
for index, row in df.iterrows():
    home_team = row['home_team']
    away_team = row['away_team']

    # Ensure there are enough past matches to calculate form
    if index >= 5:
        home_win_rate, home_avg_goals_scored, home_avg_goals_conceded = calc
        away_win_rate, away_avg_goals_scored, away_avg_goals_conceded = calc

        # Add calculated form metrics as new columns in your DataFrame (if n
        df.at[index, 'home_win_rate_5'] = home_win_rate
        df.at[index, 'away_win_rate_5'] = away_win_rate
        df.at[index, 'home_avg_goals_scored_5'] = home_avg_goals_scored
        df.at[index, 'away_avg_goals_scored_5'] = away_avg_goals_scored
        df.at[index, 'home_avg_goals_conceded_5'] = home_avg_goals_conceded
        df.at[index, 'away_avg_goals_conceded_5'] = away_avg_goals_conceded
```

In [8]:
```
df
```

Out[8]:

	match_id	match_date	season	home_team	away_team	home_team_goal	aw
0	4702	2016-05-17 00:00:00	2015/2016	Manchester United	Bournemouth	3	
1	4699	2016-05-15 00:00:00	2015/2016	Arsenal	Aston Villa	4	
2	4700	2016-05-15 00:00:00	2015/2016	Chelsea	Leicester City	1	
3	4701	2016-05-15 00:00:00	2015/2016	Everton	Norwich City	3	
4	4703	2016-05-15 00:00:00	2015/2016	Newcastle United	Tottenham Hotspur	5	
...	...	...	...	...	...	...	
3035	1732	2008-08-16 00:00:00	2008/2009	West Ham United	Wigan Athletic	2	
3036	1734	2008-08-16 00:00:00	2008/2009	Everton	Blackburn Rovers	2	
3037	1735	2008-08-16 00:00:00	2008/2009	Middlesbrough	Tottenham Hotspur	2	
3038	1736	2008-08-16 00:00:00	2008/2009	Bolton Wanderers	Stoke City	3	
3039	1737	2008-08-16 00:00:00	2008/2009	Hull City	Fulham	2	

3040 rows × 21 columns

In [9]:
```
# interaction feature
df['home_goal_count_x_buildUpPlaySpeed'] = df['home_team_goal'] * df['home_b
```

In [10]:
```
# interaction feature
df['away_goal_count_x_buildUpPlaySpeed'] = df['away_team_goal'] * df['away_b
```

In [11]:
```
df
```

Out[11]:

	match_id	match_date	season	home_team	away_team	home_team_goal	aw
0	4702	2016-05-17 00:00:00	2015/2016	Manchester United	Bournemouth	3	
1	4699	2016-05-15 00:00:00	2015/2016	Arsenal	Aston Villa	4	
2	4700	2016-05-15 00:00:00	2015/2016	Chelsea	Leicester City	1	
3	4701	2016-05-15 00:00:00	2015/2016	Everton	Norwich City	3	
4	4703	2016-05-15 00:00:00	2015/2016	Newcastle United	Tottenham Hotspur	5	
...	...	...	...	...	...	...	
3035	1732	2008-08-16 00:00:00	2008/2009	West Ham United	Wigan Athletic	2	
3036	1734	2008-08-16 00:00:00	2008/2009	Everton	Blackburn Rovers	2	
3037	1735	2008-08-16 00:00:00	2008/2009	Middlesbrough	Tottenham Hotspur	2	
3038	1736	2008-08-16 00:00:00	2008/2009	Bolton Wanderers	Stoke City	3	
3039	1737	2008-08-16 00:00:00	2008/2009	Hull City	Fulham	2	

3040 rows × 23 columns

In [12]:
```
# Calculating difference in ratings
df['rating_difference'] = df['avg_home_player_rating'] - df['avg_away_playe

# Calculating ratio of build-up play speed
df['buildUpPlaySpeed_ratio'] = df['home_buildUpPlaySpeed'] / df['away_buildU

df
```

Out[12]:

	match_id	match_date	season	home_team	away_team	home_team_goal	aw
0	4702	2016-05-17 00:00:00	2015/2016	Manchester United	Bournemouth	3	
1	4699	2016-05-15 00:00:00	2015/2016	Arsenal	Aston Villa	4	
2	4700	2016-05-15 00:00:00	2015/2016	Chelsea	Leicester City	1	
3	4701	2016-05-15 00:00:00	2015/2016	Everton	Norwich City	3	
4	4703	2016-05-15 00:00:00	2015/2016	Newcastle United	Tottenham Hotspur	5	
...	...	...	...	...	...	...	
3035	1732	2008-08-16 00:00:00	2008/2009	West Ham United	Wigan Athletic	2	
3036	1734	2008-08-16 00:00:00	2008/2009	Everton	Blackburn Rovers	2	
3037	1735	2008-08-16 00:00:00	2008/2009	Middlesbrough	Tottenham Hotspur	2	
3038	1736	2008-08-16 00:00:00	2008/2009	Bolton Wanderers	Stoke City	3	
3039	1737	2008-08-16 00:00:00	2008/2009	Hull City	Fulham	2	

3040 rows × 25 columns

In [13]:
```
df['output'] = df.apply(lambda row: 'H' if row['home_team_goal'] > row['away
                        else ('D' if row['home_team_goal'] == row['away_tear
```

In [14]:
```
df
```

Out[14]:

	match_id	match_date	season	home_team	away_team	home_team_goal	aw
0	4702	2016-05-17 00:00:00	2015/2016	Manchester United	Bournemouth	3	
1	4699	2016-05-15 00:00:00	2015/2016	Arsenal	Aston Villa	4	
2	4700	2016-05-15 00:00:00	2015/2016	Chelsea	Leicester City	1	
3	4701	2016-05-15 00:00:00	2015/2016	Everton	Norwich City	3	
4	4703	2016-05-15 00:00:00	2015/2016	Newcastle United	Tottenham Hotspur	5	
...	...	...	...	...	...	...	
3035	1732	2008-08-16 00:00:00	2008/2009	West Ham United	Wigan Athletic	2	
3036	1734	2008-08-16 00:00:00	2008/2009	Everton	Blackburn Rovers	2	
3037	1735	2008-08-16 00:00:00	2008/2009	Middlesbrough	Tottenham Hotspur	2	
3038	1736	2008-08-16 00:00:00	2008/2009	Bolton Wanderers	Stoke City	3	
3039	1737	2008-08-16 00:00:00	2008/2009	Hull City	Fulham	2	

3040 rows × 26 columns

In [15]:
```
df = df.drop(['match_id', 'match_date', 'season', 'home_team_goal', 'away_te
```

In [16]:
```
df.tail()
```

Out[16]:

	home_team	away_team	avg_home_player_rating	avg_away_player_rating	home_b
3035	West Ham United	Wigan Athletic	75.417722	73.039648	
3036	Everton	Blackburn Rovers	78.427984	73.898058	
3037	Middlesbrough	Tottenham Hotspur	72.429864	77.802326	
3038	Bolton Wanderers	Stoke City	73.025210	69.742857	
3039	Hull City	Fulham	69.246073	74.401786	

5 rows × 21 columns

In [17]:
```
df = pd.DataFrame(df)
```

In [18]:
```
%matplotlib inline
```

In [19]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3040 entries, 0 to 3039
Data columns (total 21 columns):
 #   Column                              Non-Null Count  Dtype  
---  ------                              --------------  -----  
 0   home_team                           3040 non-null   object 
 1   away_team                           3040 non-null   object 
 2   avg_home_player_rating              3040 non-null   float64
 3   avg_away_player_rating              3040 non-null   float64
 4   home_buildUpPlaySpeed               2396 non-null   float64
 5   away_buildUpPlaySpeed               2396 non-null   float64
 6   home_defenceAggression              2396 non-null   float64
 7   away_defenceAggression              2396 non-null   float64
 8   home_defencePressure                2396 non-null   float64
 9   away_defencePressure                2396 non-null   float64
 10  home_win_rate_5                     3035 non-null   float64
 11  away_win_rate_5                     3035 non-null   float64
 12  home_avg_goals_scored_5             3035 non-null   float64
 13  away_avg_goals_scored_5             3035 non-null   float64
 14  home_avg_goals_conceded_5           3035 non-null   float64
 15  away_avg_goals_conceded_5           3035 non-null   float64
 16  home_goal_count_x_buildUpPlaySpeed  2396 non-null   float64
 17  away_goal_count_x_buildUpPlaySpeed  2396 non-null   float64
 18  rating_difference                   3040 non-null   float64
 19  buildUpPlaySpeed_ratio              2396 non-null   float64
 20  output                              3040 non-null   object 
dtypes: float64(18), object(3)
memory usage: 498.9+ KB
```

Test/Train Split

In [435…

```
X = df[['home_team', 'away_team', 'avg_home_player_rating','avg_away_player_

Y = df['output']
```

In [436…

```
# encode strings to integer
Ly = LabelEncoder().fit(Y)
y = Ly.transform(Y).astype(int)
```

In [437…

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rar


print('Number of training examples',len(X_train))
print('Number of validation examples',len(X_test))
```

```
Number of training examples 2432
Number of validation examples 608
```

Dataset Preprocessing

In [64]:

```
# Define the custom transformer for casting to float32
class Float32Transformer(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self

    def transform(self, X):
        return X.astype(np.float32)

# Numeric features to be scaled
```

```
numeric_features = ['avg_home_player_rating',
       'avg_away_player_rating', 'home_buildUpPlaySpeed',
       'away_buildUpPlaySpeed', 'home_defenceAggression',
       'away_defenceAggression', 'home_defencePressure',
       'away_defencePressure', 'home_win_rate_5', 'away_win_rate_5',
       'home_avg_goals_scored_5', 'away_avg_goals_scored_5',
       'home_avg_goals_conceded_5', 'away_avg_goals_conceded_5',
       'home_goal_count_x_buildUpPlaySpeed',
       'away_goal_count_x_buildUpPlaySpeed', 'rating_difference',
       'buildUpPlaySpeed_ratio']
# Categorical features to be one-hot encoded
categorical_features = ['home_team', 'away_team']

# preprocessing pipeline for numeric features
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler()),
    ('to_float32', Float32Transformer())])

# preprocessing pipeline for categorical features
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('onehot', OneHotEncoder(handle_unknown='ignore', dtype=np.float32))])

# Combine preprocessing steps
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)])
```

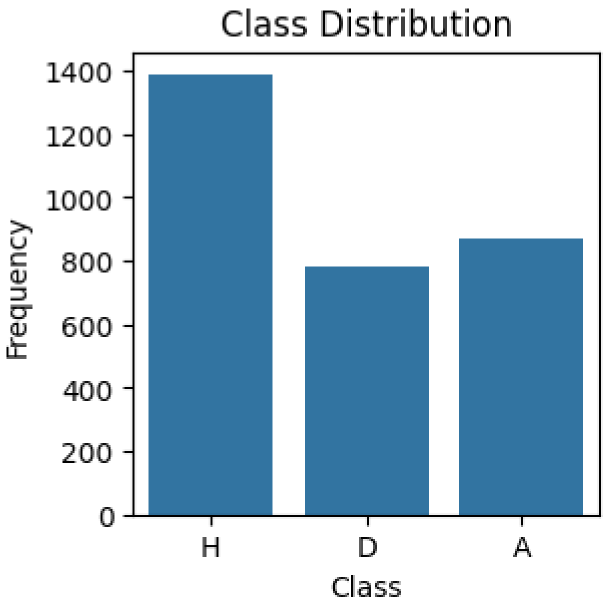
In [66]:
```
class_counts = df['output'].value_counts()
print(class_counts)
```

```
output
H    1390
A     867
D     783
Name: count, dtype: int64
```

In [67]:
```
class_distribution = df['output'].value_counts(normalize=True) * 100
print(class_distribution)
```

```
output
H    45.723684
A    28.519737
D    25.756579
Name: proportion, dtype: float64
```

In [68]:
```
plt.figure(figsize=(3, 3))
sns.countplot(x='output', data=df)
plt.title('Class Distribution')
plt.xlabel('Class')
plt.ylabel('Frequency')
plt.show()
```



Random Forest

In [39]:
```
param_distributions = {
    'classifier__n_estimators': [100, 200],
    'classifier__max_depth': [ None, 100, 200],
    'classifier__min_samples_split': [20, 50, 80],
    'classifier__min_samples_leaf': [20, 60, 80],
    'classifier__max_features': ['sqrt', 'log2', 0.5],
    'classifier__class_weight': ['balanced_subsample', 'balanced'],
    'classifier__bootstrap': [True, False]
}
```

In [41]:
```
# Create a SMOTE object to oversample the minority class
smote = SMOTE(random_state=42)
```

In [42]:
```
# Create the complete pipeline
pipeline_RFE =  ImblearnPipeline([('preprocessor', preprocessor),
                                 ('smote', smote),
                               ('classifier', RandomForestClassifier(random_sta
```

In [43]:
```
# Configure the randomized search
random_search = RandomizedSearchCV(pipeline, param_distributions=param_dist

# Fit the randomized search object to the training data
random_search.fit(X_train, y_train)
```

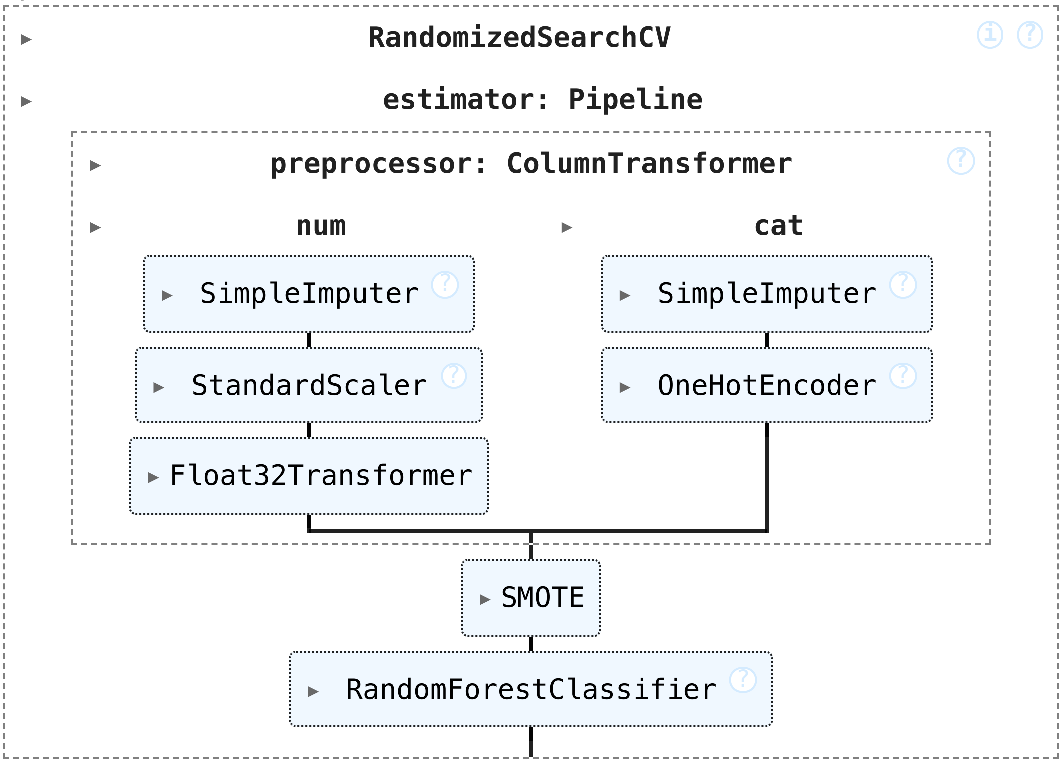
```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=100, classifier__max_features=sqrt, classifier__min_samp
les_leaf=20, classifier__min_samples_split=80, classifier__n_estimators=10
0; total time=   1.6s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=100, classifier__max_features=sqrt, classifier__min_samp
les_leaf=20, classifier__min_samples_split=80, classifier__n_estimators=10
0; total time=   1.6s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=100, classifier__max_features=sqrt, classifier__min_samp
les_leaf=20, classifier__min_samples_split=80, classifier__n_estimators=10
0; total time=   1.5s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=100, classifier__max_features=sqrt, classifier__min_samp
les_leaf=20, classifier__min_samples_split=80, classifier__n_estimators=10
0; total time=   1.3s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=100, classifier__max_features=sqrt, classifier__min_samp
les_leaf=20, classifier__min_samples_split=80, classifier__n_estimators=10
0; total time=   1.4s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced, cl
assifier__max_depth=100, classifier__max_features=0.5, classifier__min_samp
les_leaf=80, classifier__min_samples_split=80, classifier__n_estimators=20
0; total time=   6.2s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced, cl
assifier__max_depth=200, classifier__max_features=0.5, classifier__min_samp
les_leaf=20, classifier__min_samples_split=80, classifier__n_estimators=20
0; total time=   8.6s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced, cl
assifier__max_depth=200, classifier__max_features=0.5, classifier__min_samp
les_leaf=20, classifier__min_samples_split=80, classifier__n_estimators=20
0; total time=   8.8s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced, cl
assifier__max_depth=200, classifier__max_features=0.5, classifier__min_samp
les_leaf=20, classifier__min_samples_split=80, classifier__n_estimators=20
0; total time=   8.8s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced, cl
assifier__max_depth=200, classifier__max_features=0.5, classifier__min_samp
les_leaf=20, classifier__min_samples_split=80, classifier__n_estimators=20
0; total time=   8.8s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced, cl
assifier__max_depth=200, classifier__max_features=0.5, classifier__min_samp
les_leaf=20, classifier__min_samples_split=80, classifier__n_estimators=20
0; total time=   8.8s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced, cl
assifier__max_depth=100, classifier__max_features=0.5, classifier__min_samp
les_leaf=80, classifier__min_samples_split=80, classifier__n_estimators=20
0; total time=   6.3s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced, cl
assifier__max_depth=100, classifier__max_features=0.5, classifier__min_samp
les_leaf=80, classifier__min_samples_split=80, classifier__n_estimators=20
0; total time=   6.3s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced_subs
ample, classifier__max_depth=100, classifier__max_features=log2, classifier
__min_samples_leaf=20, classifier__min_samples_split=20, classifier__n_esti
mators=100; total time=   1.4s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced_subs
ample, classifier__max_depth=100, classifier__max_features=log2, classifier
__min_samples_leaf=20, classifier__min_samples_split=20, classifier__n_esti
mators=100; total time=   1.4s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced, cl
assifier__max_depth=100, classifier__max_features=0.5, classifier__min_samp
les_leaf=80, classifier__min_samples_split=80, classifier__n_estimators=20
```

```
0; total time=   5.9s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced_subs
ample, classifier__max_depth=100, classifier__max_features=log2, classifier
__min_samples_leaf=20, classifier__min_samples_split=20, classifier__n_esti
mators=100; total time=   1.6s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced, cl
assifier__max_depth=100, classifier__max_features=0.5, classifier__min_samp
les_leaf=80, classifier__min_samples_split=80, classifier__n_estimators=20
0; total time=   5.8s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced_sub
sample, classifier__max_depth=100, classifier__max_features=0.5, classifier
__min_samples_leaf=80, classifier__min_samples_split=50, classifier__n_esti
mators=200; total time=   5.8s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced_sub
sample, classifier__max_depth=100, classifier__max_features=0.5, classifier
__min_samples_leaf=80, classifier__min_samples_split=50, classifier__n_esti
mators=200; total time=   5.7s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced_sub
sample, classifier__max_depth=100, classifier__max_features=0.5, classifier
__min_samples_leaf=80, classifier__min_samples_split=50, classifier__n_esti
mators=200; total time=   5.6s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced_sub
sample, classifier__max_depth=100, classifier__max_features=0.5, classifier
__min_samples_leaf=80, classifier__min_samples_split=50, classifier__n_esti
mators=200; total time=   5.9s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced_sub
sample, classifier__max_depth=100, classifier__max_features=0.5, classifier
__min_samples_leaf=80, classifier__min_samples_split=50, classifier__n_esti
mators=200; total time=   5.6s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced_subs
ample, classifier__max_depth=100, classifier__max_features=log2, classifier
__min_samples_leaf=20, classifier__min_samples_split=20, classifier__n_esti
mators=100; total time=   1.3s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced_subs
ample, classifier__max_depth=100, classifier__max_features=log2, classifier
__min_samples_leaf=20, classifier__min_samples_split=20, classifier__n_esti
mators=100; total time=   1.3s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced_sub
sample, classifier__max_depth=200, classifier__max_features=sqrt, classifie
r__min_samples_leaf=80, classifier__min_samples_split=80, classifier__n_est
imators=100; total time=   1.0s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced_sub
sample, classifier__max_depth=200, classifier__max_features=sqrt, classifie
r__min_samples_leaf=80, classifier__min_samples_split=80, classifier__n_est
imators=100; total time=   1.0s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced_sub
sample, classifier__max_depth=200, classifier__max_features=sqrt, classifie
r__min_samples_leaf=80, classifier__min_samples_split=80, classifier__n_est
imators=100; total time=   1.0s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced_sub
sample, classifier__max_depth=200, classifier__max_features=sqrt, classifie
r__min_samples_leaf=80, classifier__min_samples_split=80, classifier__n_est
imators=100; total time=   1.1s
[CV] END classifier__bootstrap=False, classifier__class_weight=balanced_sub
sample, classifier__max_depth=200, classifier__max_features=sqrt, classifie
r__min_samples_leaf=80, classifier__min_samples_split=80, classifier__n_est
imators=100; total time=   1.1s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=None, classifier__max_features=log2, classifier__min_sam
ples_leaf=20, classifier__min_samples_split=20, classifier__n_estimators=20
0; total time=   1.9s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=None, classifier__max_features=log2, classifier__min_sam
ples_leaf=20, classifier__min_samples_split=20, classifier__n_estimators=20
```

```
0; total time=   2.0s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=None, classifier__max_features=log2, classifier__min_sam
ples_leaf=20, classifier__min_samples_split=20, classifier__n_estimators=20
0; total time=   2.0s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=None, classifier__max_features=log2, classifier__min_sam
ples_leaf=20, classifier__min_samples_split=20, classifier__n_estimators=20
0; total time=   1.9s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=None, classifier__max_features=log2, classifier__min_sam
ples_leaf=20, classifier__min_samples_split=20, classifier__n_estimators=20
0; total time=   1.9s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced_subs
ample, classifier__max_depth=200, classifier__max_features=log2, classifier
__min_samples_leaf=20, classifier__min_samples_split=80, classifier__n_esti
mators=200; total time=   2.2s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced_subs
ample, classifier__max_depth=200, classifier__max_features=log2, classifier
__min_samples_leaf=20, classifier__min_samples_split=80, classifier__n_esti
mators=200; total time=   2.2s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced_subs
ample, classifier__max_depth=200, classifier__max_features=log2, classifier
__min_samples_leaf=20, classifier__min_samples_split=80, classifier__n_esti
mators=200; total time=   2.2s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=100, classifier__max_features=sqrt, classifier__min_samp
les_leaf=80, classifier__min_samples_split=50, classifier__n_estimators=20
0; total time=   1.6s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=100, classifier__max_features=sqrt, classifier__min_samp
les_leaf=80, classifier__min_samples_split=50, classifier__n_estimators=20
0; total time=   1.5s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=100, classifier__max_features=sqrt, classifier__min_samp
les_leaf=80, classifier__min_samples_split=50, classifier__n_estimators=20
0; total time=   1.5s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced_subs
ample, classifier__max_depth=200, classifier__max_features=log2, classifier
__min_samples_leaf=20, classifier__min_samples_split=80, classifier__n_esti
mators=200; total time=   2.2s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced_subs
ample, classifier__max_depth=200, classifier__max_features=log2, classifier
__min_samples_leaf=20, classifier__min_samples_split=80, classifier__n_esti
mators=200; total time=   2.2s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=100, classifier__max_features=sqrt, classifier__min_samp
les_leaf=80, classifier__min_samples_split=50, classifier__n_estimators=20
0; total time=   1.5s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=100, classifier__max_features=sqrt, classifier__min_samp
les_leaf=80, classifier__min_samples_split=50, classifier__n_estimators=20
0; total time=   1.5s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=200, classifier__max_features=sqrt, classifier__min_samp
les_leaf=60, classifier__min_samples_split=20, classifier__n_estimators=20
0; total time=   1.6s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=200, classifier__max_features=sqrt, classifier__min_samp
les_leaf=60, classifier__min_samples_split=20, classifier__n_estimators=20
0; total time=   1.4s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=200, classifier__max_features=sqrt, classifier__min_samp
les_leaf=60, classifier__min_samples_split=20, classifier__n_estimators=20
```

```
0; total time=   1.3s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=200, classifier__max_features=sqrt, classifier__min_samp
les_leaf=60, classifier__min_samples_split=20, classifier__n_estimators=20
0; total time=   1.2s
[CV] END classifier__bootstrap=True, classifier__class_weight=balanced, cla
ssifier__max_depth=200, classifier__max_features=sqrt, classifier__min_samp
les_leaf=60, classifier__min_samples_split=20, classifier__n_estimators=20
0; total time=   1.2s
```

Out[43]:



In [44]:
```
print("Best parameters found: ", random_search.best_params_)
print("Best cross-validation score: ", random_search.best_score_)

# Evaluate on the test set
test_score = random_search.score(X_test, y_test)
print("Test set score: ", test_score)
# Evaluate on the train set
train_score = random_search.score(X_train, y_train)
print("Train set score: ", train_score)
```

```
Best parameters found:  {'classifier__n_estimators': 200, 'classifier__min_
samples_split': 80, 'classifier__min_samples_leaf': 20, 'classifier__max_fe
atures': 0.5, 'classifier__max_depth': 200, 'classifier__class_weight': 'ba
lanced', 'classifier__bootstrap': False}
Best cross-validation score:  0.843334938863116
Test set score:  0.8552631578947368
Train set score:  0.8951480263157895
```

In [45]:
```
y_pred = random_search.predict(X_test)

# Calculate accuracy, precision, recall, and F1 score
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f"Precision: {precision}")
print(f"Recall: {recall}")
print(f"F1 Score: {f1}")
print(f"Accuracy:{accuracy}")
```

```
Precision: 0.8594247766256548
Recall: 0.8552631578947368
F1 Score: 0.8564794521233697
Accuracy:0.8552631578947368
```

Use all data with CV

In [32]:
```
# Incorporating SMOTE to handle class imbalance
smote = SMOTE(random_state=42)

classifier = RandomForestClassifier(random_state=42, class_weight='balanced_
# Define pipeline
pipeline = make_pipeline_imb(
    preprocessor,
    smote,
    classifier
)

# Define scoring metrics
scoring = {'accuracy': 'accuracy',
           'precision': make_scorer(precision_score, average='weighted'),  #
           'recall': make_scorer(recall_score, average='weighted'),  # Adju
           'f1_score': make_scorer(f1_score, average='weighted')}

# Use StratifiedKFold for cross-validation to maintain the percentage of sam
cv = StratifiedKFold(n_splits=6, shuffle=True, random_state=42)

# Perform cross-validation

scores = cross_validate(pipeline, X, y, cv=cv, scoring=scoring, return_trai

#print("Accuracy scores for each fold:", scores)
#print("Mean cross-validation accuracy:", scores.mean())
```

In [33]:
```
# Print scores for each metric
for metric in scoring:
    metric_scores = scores[f'test_{metric}']
    print(f"{metric.capitalize()} scores for each fold:", metric_scores)
    print(f"Mean {metric.capitalize()}:", metric_scores.mean())
```

```
Accuracy scores for each fold: [0.78500986 0.82248521 0.81854043 0.78303748
0.82213439 0.82608696]
Mean Accuracy: 0.8095490536962108
Precision scores for each fold: [0.78587442 0.81996826 0.81527497 0.7792513
8 0.82241683 0.82410654]
Mean Precision: 0.8078154007414079
Recall scores for each fold: [0.78500986 0.82248521 0.81854043 0.78303748
0.82213439 0.82608696]
Mean Recall: 0.8095490536962108
F1_score scores for each fold: [0.78464701 0.81965656 0.81445972 0.7805142
0.82207285 0.82450086]
Mean F1_score: 0.8076418664467769
```

Feature Selection

In [67]:
```
# parameter grid
param_grid = {'classifier__n_estimators': [100, 200],
              'classifier__max_depth': [None, 10, 20],
              'feature_elimination__n_features_to_select': [5, 10, 15]}

# GridSearchCV object
```

```
grid_search = GridSearchCV(pipeline_RFE, param_grid, cv=5, scoring='accuracy
grid_search.fit(X_train, y_train)

# Best model
print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_)
```

```
Best parameters: {'classifier__max_depth': 20, 'classifier__n_estimators':
100, 'feature_elimination__n_features_to_select': 10}
Best cross-validation score: 0.86
```

In [71]:

```
# best estimator from the grid search
best_model = grid_search.best_estimator_

# Predict on the validation set
y_pred_test = best_model.predict(X_test)

# Evaluate the model
print("Validation Accuracy:", accuracy_score(y_test, y_pred_test))
print(classification_report(y_test, y_pred_test))
```

```
Validation Accuracy: 0.8569078947368421
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       172
           1       0.80      0.73      0.77       150
           2       0.87      0.91      0.89       286

    accuracy                           0.86       608
   macro avg       0.85      0.84      0.84       608
weighted avg       0.86      0.86      0.86       608
```

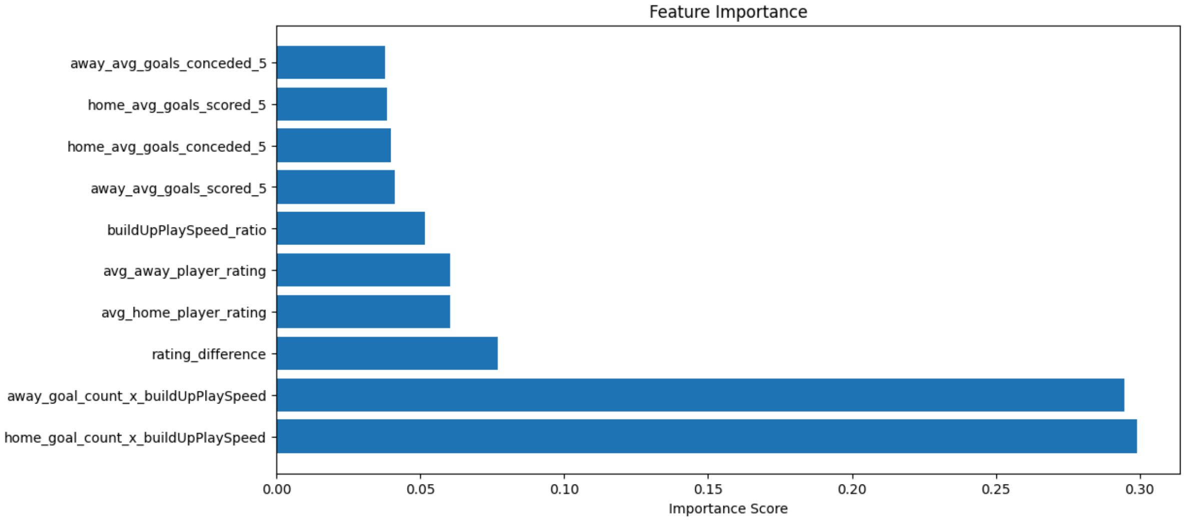
In [72]:

```
importances = best_model.named_steps['classifier'].feature_importances_
selected_features = transformed_feature_names[best_model.named_steps['featu

# Dictionary of features and their importance scores
feature_importance_dict = dict(zip(selected_features, importances))
sorted_feature_importance = sorted(feature_importance_dict.items(), key=laml
print("Sorted Feature Importances:", sorted_feature_importance)
```

```
Sorted Feature Importances: [('home_goal_count_x_buildUpPlaySpeed', 0.29896
66966280148), ('away_goal_count_x_buildUpPlaySpeed', 0.29454061463065856),
('rating_difference', 0.0768690873066816), ('avg_home_player_rating', 0.060
371512567891604), ('avg_away_player_rating', 0.060338540560018146), ('build
UpPlaySpeed_ratio', 0.05171330068819019), ('away_avg_goals_scored_5', 0.041
26784827577079), ('home_avg_goals_conceded_5', 0.03991393909772067 4), ('hom
e_avg_goals_scored_5', 0.038365337556582085), ('away_avg_goals_conceded_5',
0.03765312268847183)]
```

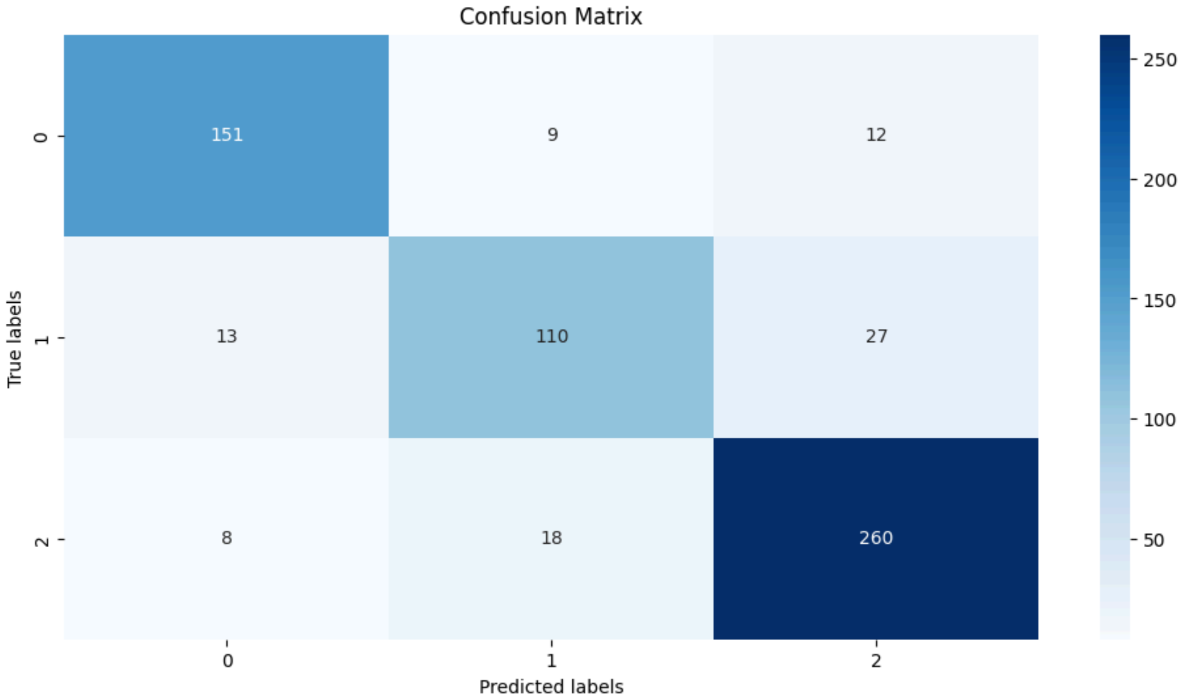
In [73]:

```
# Extracting names and importance scores
features, importance_scores = zip(*sorted_feature_importance)

plt.barh(features, importance_scores)
plt.xlabel('Importance Score')
plt.title('Feature Importance')
plt.show()
```



In [99]:

```
# Generating the confusion matrix
cm = confusion_matrix(y_test, y_pred_test)
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

# Printing the classification report
print(classification_report(y_test, y_pred_test))
```




```
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       172
           1       0.80      0.73      0.77       150
           2       0.87      0.91      0.89       286

    accuracy                           0.86       608
   macro avg       0.85      0.84      0.84       608
weighted avg       0.86      0.86      0.86       608
```

SVM

In [52]:
```
pipeline_ = Pipeline([
    ('preprocessor', preprocessor),
    ('poly_features', PolynomialFeatures(degree=2)),
    ('svm', SVC(random_state=42, class_weight='balanced'))
])
```

In [105…
```
# parameter grid
param_distributions = {
    'svm__C': [0.1, 1, 10, 100],
    'svm__gamma': ['scale', 'auto', 0.01, 0.1, 1, 10],
    'svm__kernel': ['linear', 'rbf', 'poly'],
    'svm__degree': [2,3,4]
}
```

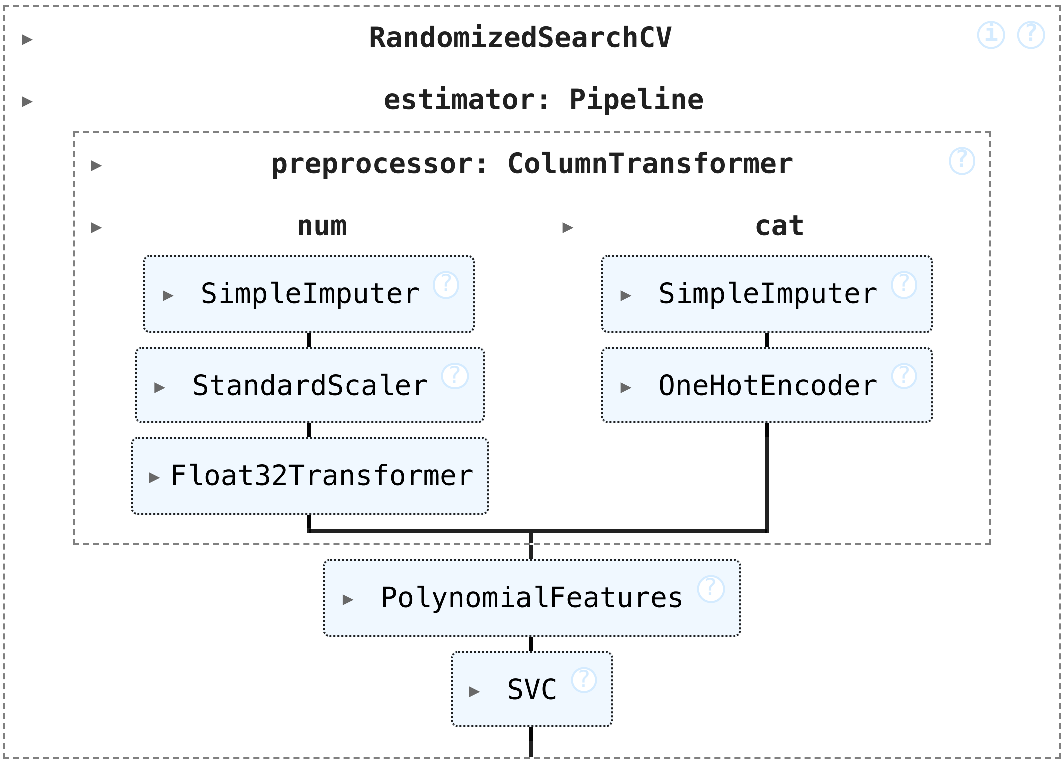
In [106…
```
# RandomizedSearchCV
random_search_SVM = RandomizedSearchCV(pipeline_, param_distributions=param_
```

In [107…
```
# Fit RandomizedSearchCV
random_search_SVM.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END svm__C=100, svm__degree=4, svm__gamma=auto, svm__kernel=linear; to
tal time=   5.2s
[CV] END svm__C=100, svm__degree=4, svm__gamma=auto, svm__kernel=linear; to
tal time=   5.3s
[CV] END svm__C=100, svm__degree=4, svm__gamma=auto, svm__kernel=linear; to
tal time=   5.3s
[CV] END svm__C=100, svm__degree=4, svm__gamma=10, svm__kernel=linear; tota
l time=   5.3s
[CV] END svm__C=100, svm__degree=4, svm__gamma=auto, svm__kernel=linear; to
tal time=   5.3s
[CV] END svm__C=100, svm__degree=4, svm__gamma=10, svm__kernel=linear; tota
l time=   5.3s
[CV] END svm__C=100, svm__degree=4, svm__gamma=10, svm__kernel=linear; tota
l time=   5.4s
[CV] END svm__C=100, svm__degree=4, svm__gamma=auto, svm__kernel=linear; to
tal time=   5.4s
[CV] END svm__C=100, svm__degree=4, svm__gamma=10, svm__kernel=linear; tota
l time=   3.6s
[CV] END svm__C=100, svm__degree=4, svm__gamma=10, svm__kernel=linear; tota
l time=   3.6s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=linear; total
time=   3.7s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=linear; total
time=   3.7s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=linear; total
time=   3.7s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=linear; total
time=   3.8s
[CV] END svm__C=100, svm__degree=2, svm__gamma=10, svm__kernel=linear; tota
l time=   3.7s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=linear; total
time=   3.9s
[CV] END svm__C=100, svm__degree=2, svm__gamma=10, svm__kernel=linear; tota
l time=   3.6s
[CV] END svm__C=100, svm__degree=2, svm__gamma=10, svm__kernel=linear; tota
l time=   3.7s
[CV] END svm__C=100, svm__degree=2, svm__gamma=10, svm__kernel=linear; tota
l time=   3.6s
[CV] END svm__C=100, svm__degree=2, svm__gamma=10, svm__kernel=linear; tota
l time=   3.9s
[CV] END svm__C=0.1, svm__degree=2, svm__gamma=10, svm__kernel=linear; tota
l time=   3.6s
[CV] END svm__C=0.1, svm__degree=2, svm__gamma=10, svm__kernel=linear; tota
l time=   3.8s
[CV] END svm__C=0.1, svm__degree=2, svm__gamma=10, svm__kernel=linear; tota
l time=   3.8s
[CV] END svm__C=0.1, svm__degree=2, svm__gamma=10, svm__kernel=linear; tota
l time=   3.8s
[CV] END svm__C=10, svm__degree=2, svm__gamma=auto, svm__kernel=linear; tot
al time=   3.8s
[CV] END svm__C=0.1, svm__degree=2, svm__gamma=10, svm__kernel=linear; tota
l time=   4.1s
[CV] END svm__C=10, svm__degree=2, svm__gamma=auto, svm__kernel=linear; tot
al time=   4.0s
[CV] END svm__C=10, svm__degree=2, svm__gamma=auto, svm__kernel=linear; tot
al time=   3.9s
[CV] END svm__C=10, svm__degree=2, svm__gamma=auto, svm__kernel=linear; tot
al time=   4.0s
[CV] END svm__C=10, svm__degree=2, svm__gamma=auto, svm__kernel=linear; tot
al time=   4.0s
[CV] END svm__C=100, svm__degree=3, svm__gamma=scale, svm__kernel=poly; tot
al time=   7.4s
[CV] END svm__C=100, svm__degree=3, svm__gamma=scale, svm__kernel=poly; tot
```

```
al time=   8.0s
[CV] END svm__C=100, svm__degree=3, svm__gamma=scale, svm__kernel=poly; tot
al time=   7.2s
[CV] END svm__C=100, svm__degree=3, svm__gamma=scale, svm__kernel=poly; tot
al time=   7.1s
[CV] END svm__C=100, svm__degree=3, svm__gamma=scale, svm__kernel=poly; tot
al time=   7.2s
[CV] END svm__C=1, svm__degree=3, svm__gamma=scale, svm__kernel=rbf; total
time=   7.2s
[CV] END svm__C=1, svm__degree=3, svm__gamma=scale, svm__kernel=rbf; total
time=   7.2s
[CV] END svm__C=1, svm__degree=3, svm__gamma=scale, svm__kernel=rbf; total
time=   7.2s
[CV] END svm__C=1, svm__degree=3, svm__gamma=scale, svm__kernel=rbf; total
time=   6.7s
[CV] END svm__C=1, svm__degree=3, svm__gamma=scale, svm__kernel=rbf; total
time=   6.6s
[CV] END svm__C=100, svm__degree=4, svm__gamma=0.01, svm__kernel=rbf; total
time=   8.0s
[CV] END svm__C=100, svm__degree=4, svm__gamma=0.01, svm__kernel=rbf; total
time=   8.2s
[CV] END svm__C=100, svm__degree=4, svm__gamma=0.01, svm__kernel=rbf; total
time=   8.1s
[CV] END svm__C=100, svm__degree=4, svm__gamma=0.01, svm__kernel=rbf; total
time=   8.3s
[CV] END svm__C=100, svm__degree=4, svm__gamma=0.01, svm__kernel=rbf; total
time=   8.2s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=rbf; total tim
e=   8.2s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=rbf; total tim
e=   7.1s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=rbf; total tim
e=   6.8s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=rbf; total tim
e=   4.9s
[CV] END svm__C=10, svm__degree=3, svm__gamma=1, svm__kernel=rbf; total tim
e=   4.8s
```

Out[107]:



In [113…

```python
# Evaluate the best model found on the test set
print("Best parameters:", random_search_SVM.best_params_)
```

```
print("Best cross-validation score:", random_search_SVM.best_score_)
print("Test set score:", random_search_SVM.score(X_test, y_test))
```

```
Best parameters: {'svm__kernel': 'linear', 'svm__gamma': 10, 'svm__degree':
2, 'svm__C': 0.1}
Best cross-validation score: 0.8059134196939354
Test set score: 0.8223684210526315
```

In [109…

```
y_pred_SVM = random_search_SVM.predict(X_test)
```

In [110…

```
acc_SVM = accuracy_score(y_test, y_pred_SVM)
```

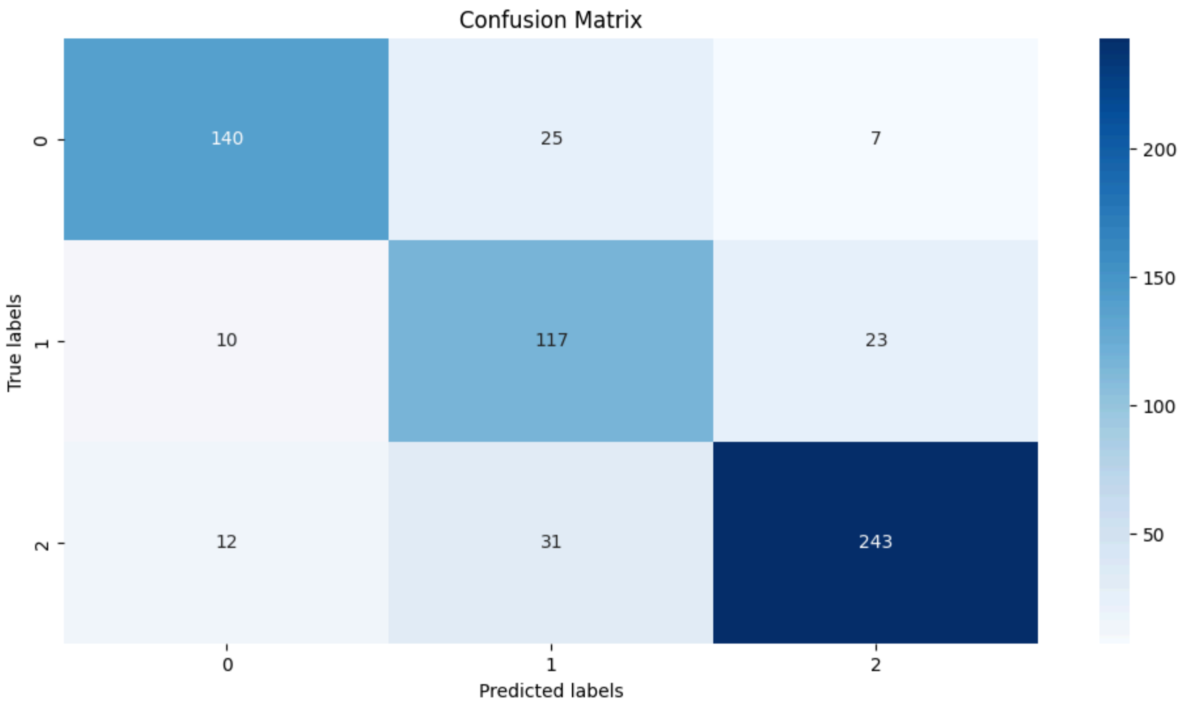
In [111…

```
print ("The accuracy on validation dataset: \t", accuracy_score(y_test, y_pr
print ("Precision on validation dataset: \t", precision_score(y_test, y_pred
print ("Recall on validation dataset : \t", recall_score(y_test, y_pred_SVM,
print ("F1 score on validation dataset: \t", f1_score(y_test, y_pred_SVM, av
```

```
The accuracy on validation dataset:      0.8223684210526315
Precision on validation dataset:         0.8300304121468303
Recall on validation dataset :   0.8223684210526315
F1 score on validation dataset:          0.8248544575778283
```

In [112…

```
# Generating the confusion matrix
cm = confusion_matrix(y_test, y_pred_SVM)
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

# Printing the classification report
print(classification_report(y_test, y_pred_SVM))
```



```
              precision    recall  f1-score   support

           0       0.86      0.81      0.84       172
           1       0.68      0.78      0.72       150
           2       0.89      0.85      0.87       286

    accuracy                           0.82       608
   macro avg       0.81      0.81      0.81       608
weighted avg       0.83      0.82      0.82       608
```

In [120…

```
# pipeline with recursive feature elimination and classification
pipeline_SVM =  ImblearnPipeline([('preprocessor', preprocessor),
                               ('feature_elimination', RFE(estimator=SVC(ke
                              ('smote', smote),
                           ('svc', SVC(C=0.1, kernel='linear'))])

# Fit the pipeline
pipeline_SVM.fit(X_train, y_train)

# transformed feature names
transformed_feature_names = get_feature_names(preprocessor)

# mask of selected features from RFE
selected_mask_SVM = pipeline_SVM.named_steps['feature_elimination'].support_
selected_features_SVM = transformed_feature_names[selected_mask]

print("Selected features:", selected_features_SVM)
```

```
Selected features: ['avg_home_player_rating' 'avg_away_player_rating'
 'home_avg_goals_scored_5' 'away_avg_goals_scored_5'
 'home_avg_goals_conceded_5' 'away_avg_goals_conceded_5'
 'home_goal_count_x_buildUpPlaySpeed' 'away_goal_count_x_buildUpPlaySpeed'
 'rating_difference' 'buildUpPlaySpeed_ratio']
```

In [119…

```
# Predict on the validation set (assuming X_val is preprocessed accordingly,
y_pred_SVM_RFE = pipeline_SVM.predict(X_test)

# Evaluate the model
print("Accuracy:", accuracy_score(y_test, y_pred_SVM_RFE))
print(classification_report(y_test, y_pred_SVM_RFE))
```

```
Accuracy: 0.8009868421052632
              precision    recall  f1-score   support

           0       0.89      0.77      0.83       172
           1       0.57      0.91      0.70       150
           2       0.99      0.76      0.86       286

    accuracy                           0.80       608
   macro avg       0.82      0.81      0.80       608
weighted avg       0.86      0.80      0.81       608
```

In [123…

```
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

# parameter grid
param_grid_SVM = {
    'feature_elimination__n_features_to_select': [5, 10, 15, 20],
    'svc__C': [0.1, 1, 10]}

# GridSearchCV object
grid_search_SVM = GridSearchCV(pipeline_SVM, param_grid_SVM, cv=cv, scoring=
grid_search_SVM.fit(X_train, y_train)
```

```
# Best model
print("Best parameters:", grid_search_SVM.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search_SVM.best_sco
```

```
Best parameters: {'feature_elimination__n_features_to_select': 20, 'svc__
C': 1}
Best cross-validation score: 0.82
```

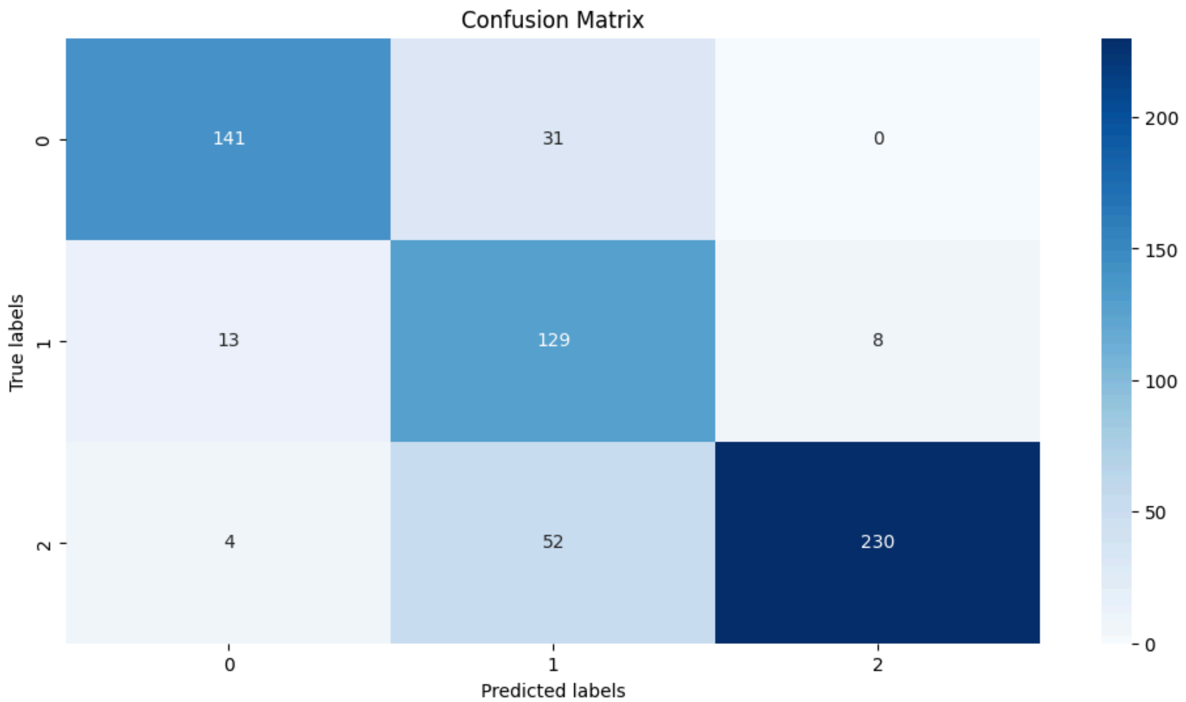
In [124…

```
# best estimator from the grid search
best_model_ = grid_search_SVM.best_estimator_

# Predict on the validation set
y_pred_test_RF = best_model_.predict(X_test)

# Evaluate the model
print("Validation Accuracy:", accuracy_score(y_test, y_pred_test_RF))
print(classification_report(y_test, y_pred_test_RF))
```

```
Validation Accuracy: 0.8223684210526315
              precision    recall  f1-score   support

           0       0.89      0.82      0.85       172
           1       0.61      0.86      0.71       150
           2       0.97      0.80      0.88       286

    accuracy                           0.82       608
   macro avg       0.82      0.83      0.82       608
weighted avg       0.86      0.82      0.83       608
```

In [125…

```
# Generating the confusion matrix
cm = confusion_matrix(y_test, y_pred_test_RF)
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

# Printing the classification report
print(classification_report(y_test, y_pred_test_RF))
```



```
              precision    recall  f1-score   support

           0       0.89      0.82      0.85       172
           1       0.61      0.86      0.71       150
           2       0.97      0.80      0.88       286

    accuracy                           0.82       608
   macro avg       0.82      0.83      0.82       608
weighted avg       0.86      0.82      0.83       608
```

Bagging with Random Forest

In [127…

```
print("Best parameters:", grid_search.best_params_)
```

```
Best parameters: {'classifier__max_depth': 20, 'classifier__n_estimators':
100, 'feature_elimination__n_features_to_select': 10}
```

In [128…

```
best_randomforest = RandomForestClassifier(
    n_estimators=100,
    max_depth=20
)
```

In [130…

```
# Initialize the BaggingClassifier using the RandomForestClassifier instance
clf = BaggingClassifier(estimator=best_randomforest, n_estimators=100, rando
```

In [131…

```
# Create a SMOTE object to oversample the minority class
smote = SMOTE(random_state=42)
```

In [154…

```
pipeline_ =  ImblearnPipeline([('preprocessor', preprocessor),
                               ('feature_selection', RFE(estimator=RandomFo
                              ('smote', smote),
                           ('bagging', clf)])
```

In [155…

```
param_grid = {
    'bagging__estimator__max_depth': [3, 5, 10],
    'bagging__n_estimators': [10, 50, 100]
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)

grid_search = GridSearchCV(pipeline_, param_grid, cv=cv)
grid_search.fit(X_train, y_train)

print("Best parameters:", grid_search.best_params_)
print("Best cross-validation score: {:.2f}".format(grid_search.best_score_))
```

```
Best parameters: {'bagging__estimator__max_depth': 10, 'bagging__n_estimato
rs': 10}
Best cross-validation score: 0.85
```

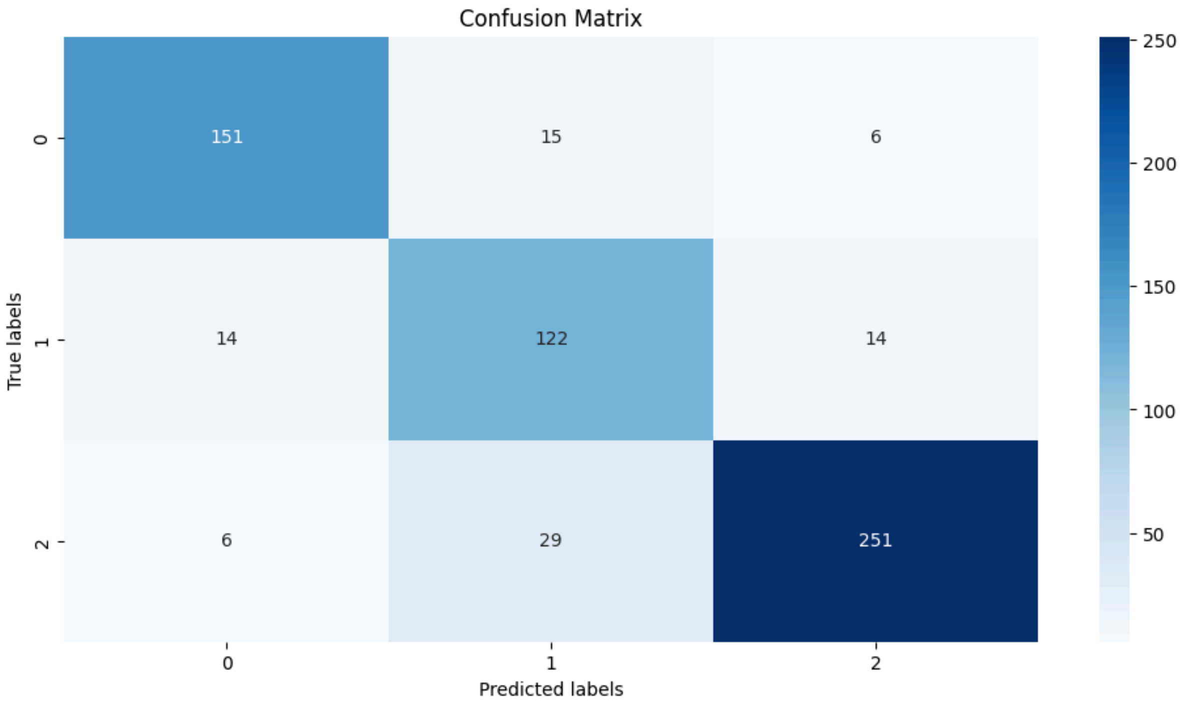
In [156…

```
# best estimator from the grid search
best_model_BG = grid_search.best_estimator_

# Predict on the validation set
y_pred_test_BG = best_model_BG.predict(X_test)

# Evaluate the model
print("Validation Accuracy:", accuracy_score(y_test, y_pred_test_BG))
print(classification_report(y_test, y_pred_test_BG))
```

```
Validation Accuracy: 0.8618421052631579
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       172
           1       0.73      0.81      0.77       150
           2       0.93      0.88      0.90       286

    accuracy                           0.86       608
   macro avg       0.85      0.86      0.85       608
weighted avg       0.87      0.86      0.86       608
```

In [157…

```
# Generating the confusion matrix
cm = confusion_matrix(y_test, y_pred_test_BG)
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

# Printing the classification report
print(classification_report(y_test, y_pred_test_BG))
```




```
              precision    recall  f1-score   support

           0       0.88      0.88      0.88       172
           1       0.73      0.81      0.77       150
           2       0.93      0.88      0.90       286

    accuracy                           0.86       608
   macro avg       0.85      0.86      0.85       608
weighted avg       0.87      0.86      0.86       608
```

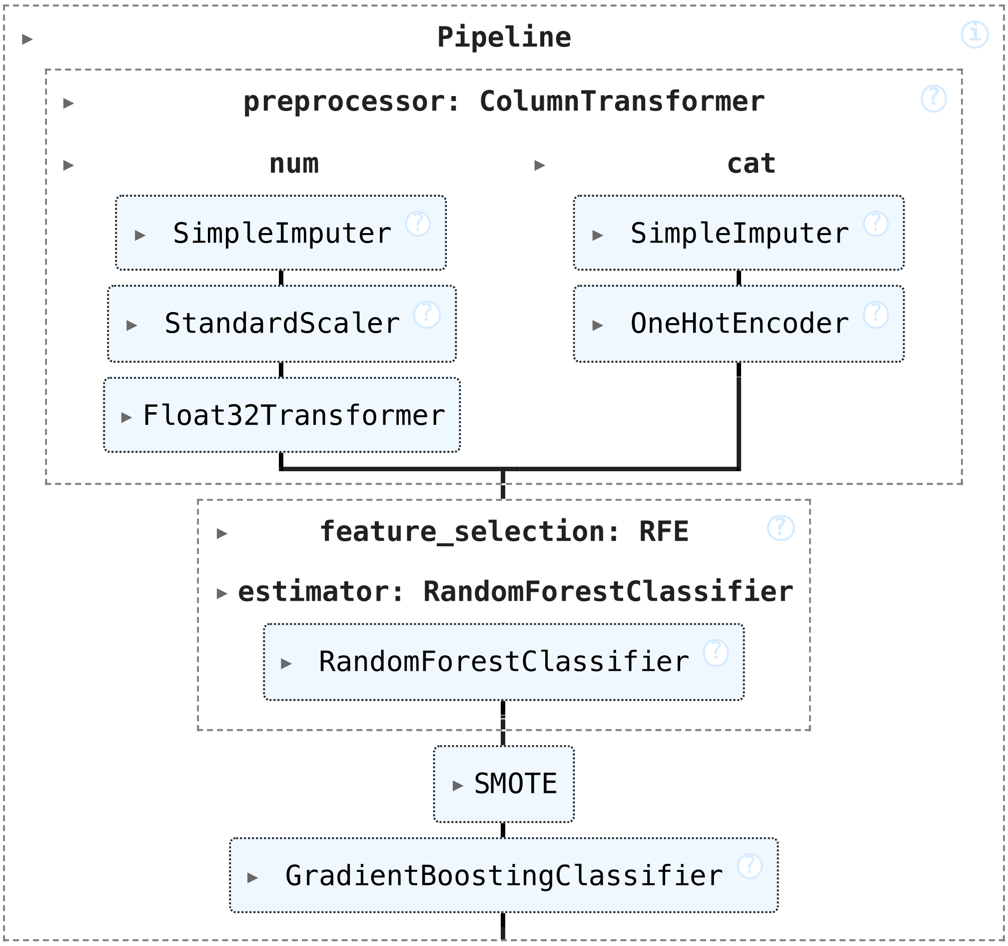
Gradient Boosting

In [158…

```
# GradientBoostingClassifier
gradient_boosting_classifier = GradientBoostingClassifier(n_estimators=100,

# SMOTE object to oversample the minority class
```

```
smote = SMOTE(random_state=42)

# preprocessing and classification pipeline
pipeline_GB = ImblearnPipeline([('preprocessor', preprocessor),
                               ('feature_selection', RFE(estimator=RandomFo
                              ('smote', smote),
                           ('gb_RF', gradient_boosting_classifier)])
# Fit the model
pipeline_GB.fit(X_train, y_train)
```

Out[158]:



In [159…

```
# Predict with the model
predictions = pipeline_GB.predict(X_test)
```

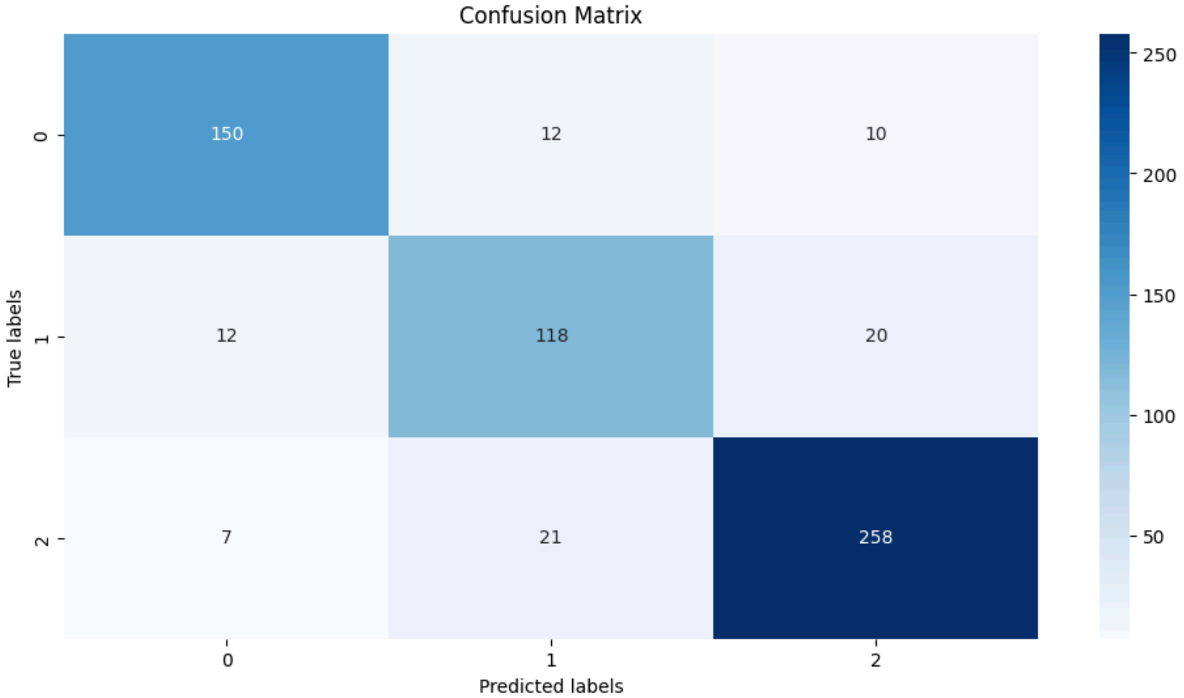
In [160…

```
print(f"Accuracy: {accuracy_score(y_test,predictions)}")
```

```
Accuracy: 0.8651315789473685
```

In [161…

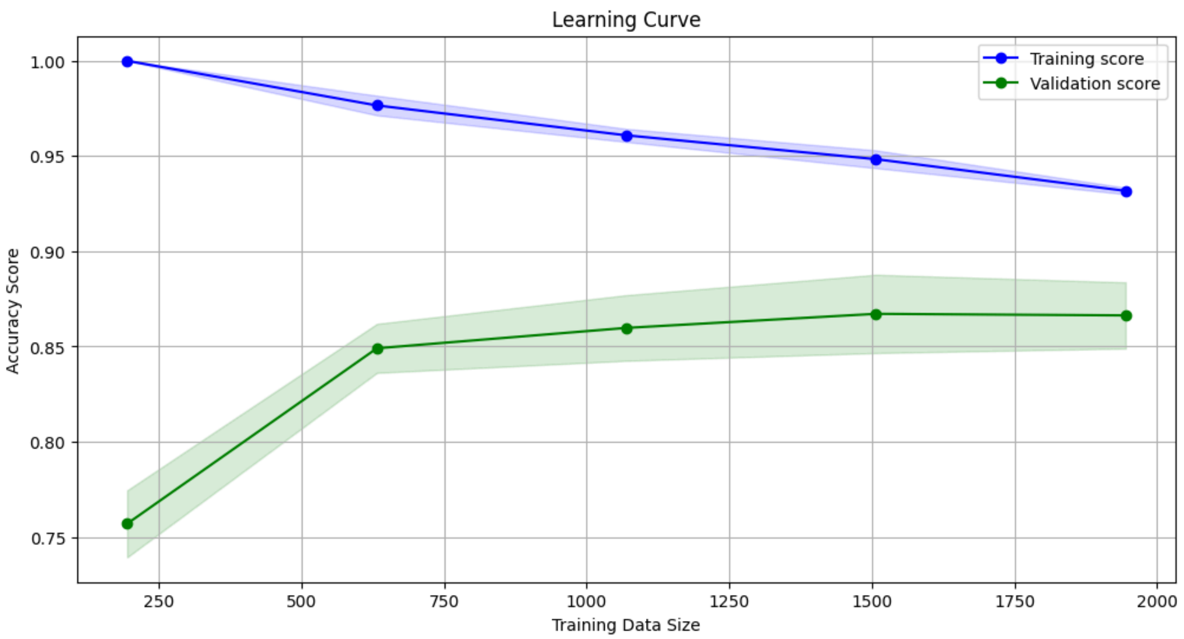
```
# Generating the confusion matrix
cm = confusion_matrix(y_test, predictions)
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

# Printing the classification report
print(classification_report(y_test, predictions))
```




```
              precision    recall  f1-score   support

           0       0.89      0.87      0.88       172
           1       0.78      0.79      0.78       150
           2       0.90      0.90      0.90       286

    accuracy                           0.87       608
   macro avg       0.85      0.85      0.85       608
weighted avg       0.87      0.87      0.87       608
```

In [222…

```
plot_learning_curves(pipeline_GB, X_train, y_train, cv=cv)
```



Bagging with SVM

In [163…

```
bagging_SV = BaggingClassifier(estimator=SVC(C=1), n_estimators=10, random_s
```

In [171…

```
#  pipeline
pipeline_SVM = ImblearnPipeline([('preprocessor', preprocessor),
```

```
                              ('feature_selection', RFE(estimator=SVC(kerne
                             ('smote', smote),
                           ('bagging_SVM', bagging_SV)]).fit(X_train, y_tra
```

In [172…

```
predictions_SVM = pipeline_SVM.predict(X_test)
```

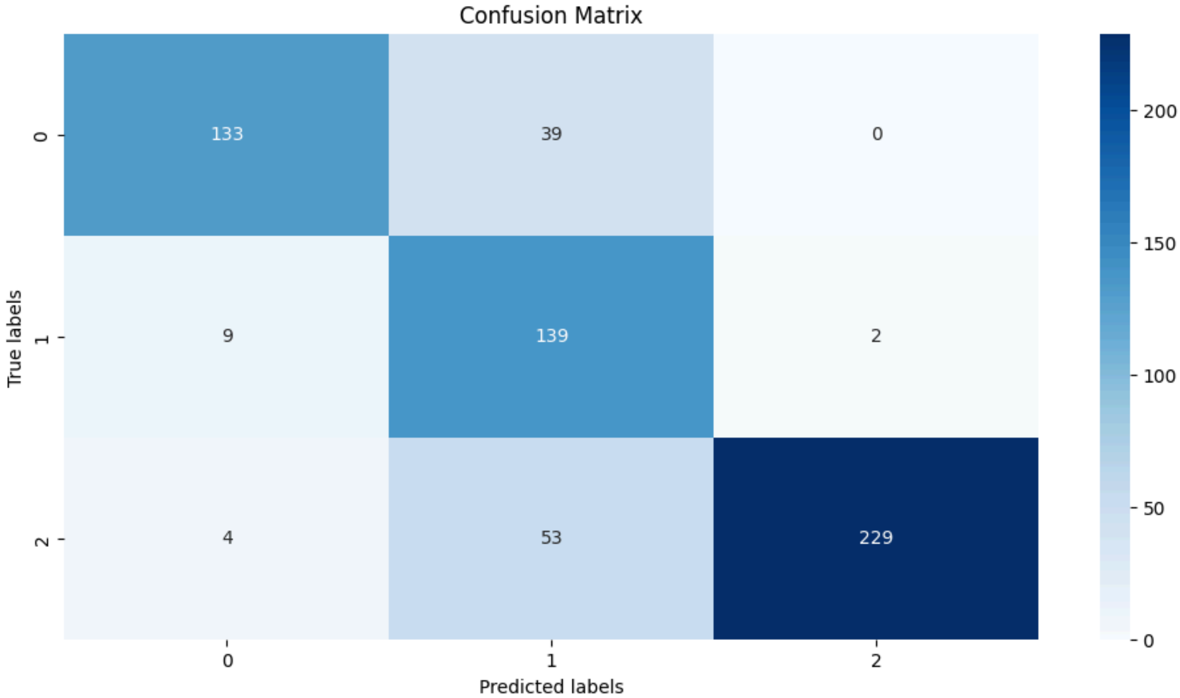
In [173…

```
print(f"Accuracy: {accuracy_score(y_test, predictions_SVM)}")
```

```
Accuracy: 0.8240131578947368
```

In [174…

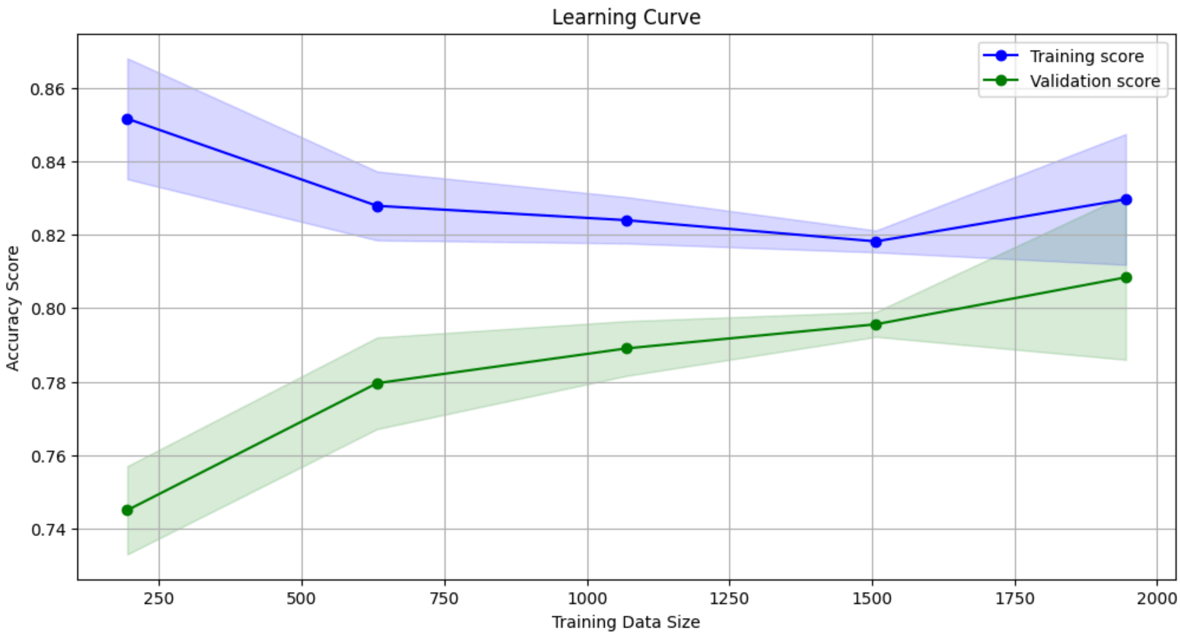
```
# Generating the confusion matrix
cm = confusion_matrix(y_test, predictions_SVM)
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

# Printing the classification report
print(classification_report(y_test, predictions_SVM))
```




```
              precision    recall  f1-score   support

           0       0.91      0.77      0.84       172
           1       0.60      0.93      0.73       150
           2       0.99      0.80      0.89       286

    accuracy                           0.82       608
   macro avg       0.83      0.83      0.82       608
weighted avg       0.87      0.82      0.83       608
```

In [221…

```
plot_learning_curves(pipeline_SVM, X_train, y_train, cv=cv)
```



AdaBoost with SVM

In [382…
```
ada_boost_clf = AdaBoostClassifier(estimator=SVC(), algorithm='SAMME', learn
```

In [389…
```
pipeline_SVM_boosting = ImblearnPipeline([
    ('preprocessor', preprocessor),
    ('feature_selection', RFE(estimator=SVC(kernel='linear'))),
    ('smote', smote),
    ('gb_SVM', ada_boost_clf)
])
```

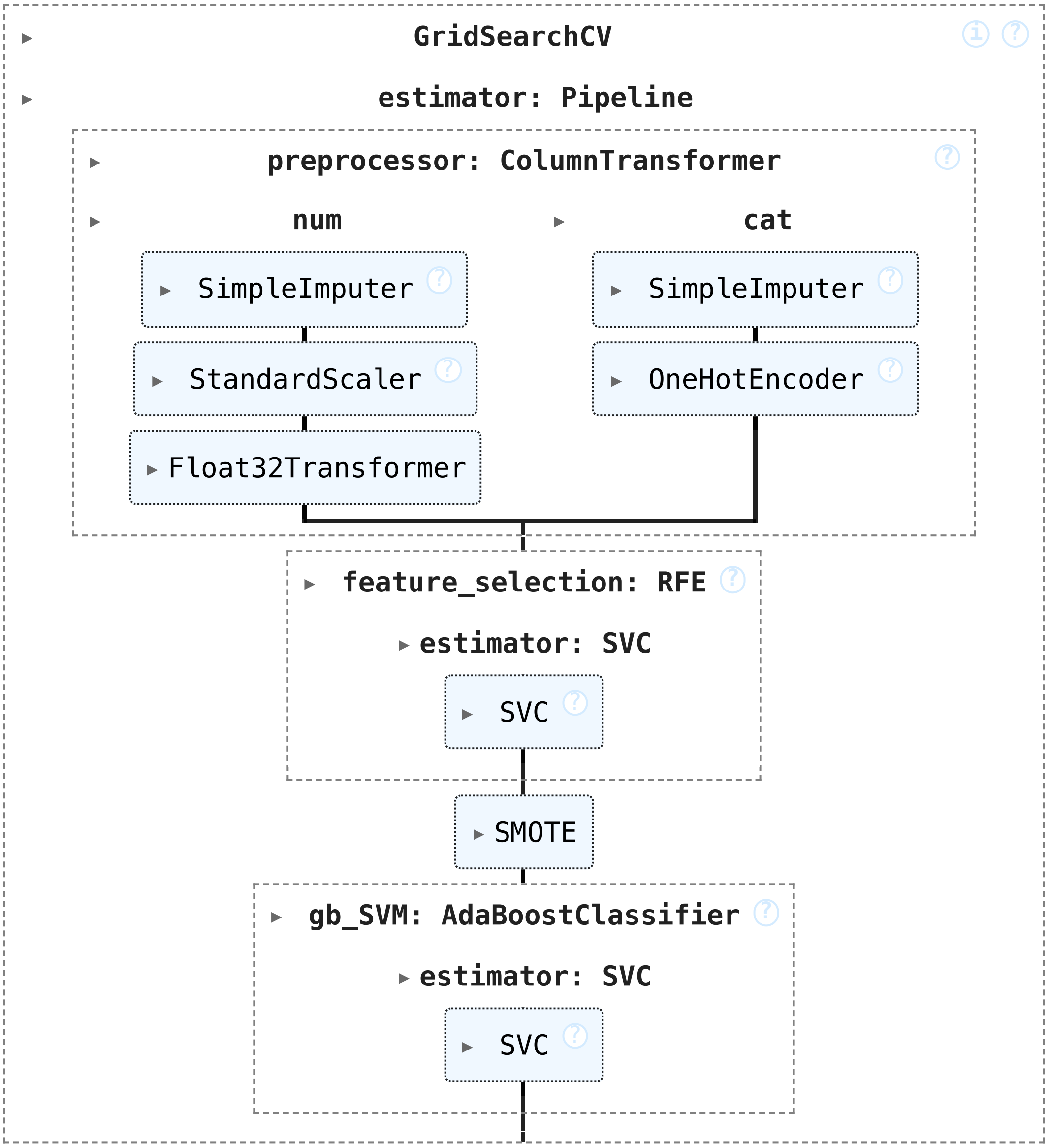
In [390…
```
param_grid = {
    'feature_selection__n_features_to_select': [5, 10, 15],
    'gb_SVM__estimator__C': [0.1, 1, 10],
    'gb_SVM__n_estimators': [50, 100]
}
```

In [391…
```
grid_search_Bst = GridSearchCV(pipeline_SVM_boosting, param_grid, cv=cv, sco
```

In [392…
```
grid_search_Bst.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

Out[392]:

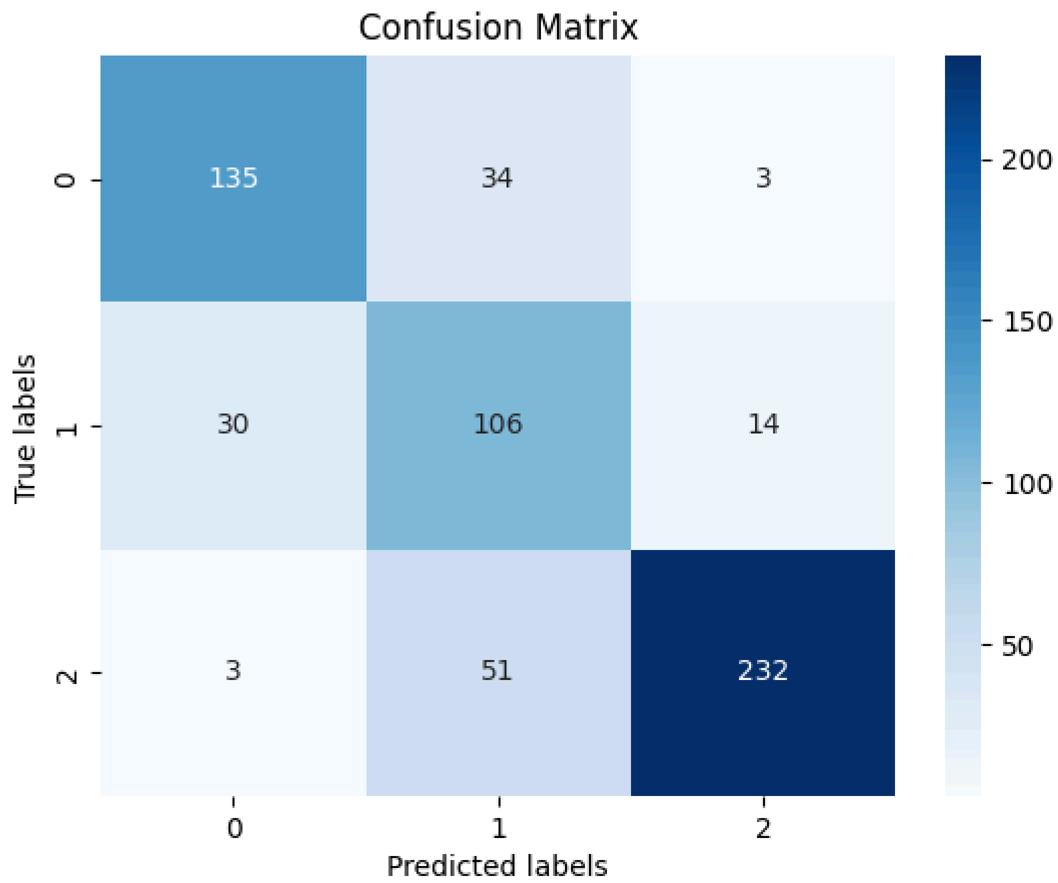


In [393…

```
y_pred_test_Bst = grid_search_Bst.predict(X_test)
```

In [394…

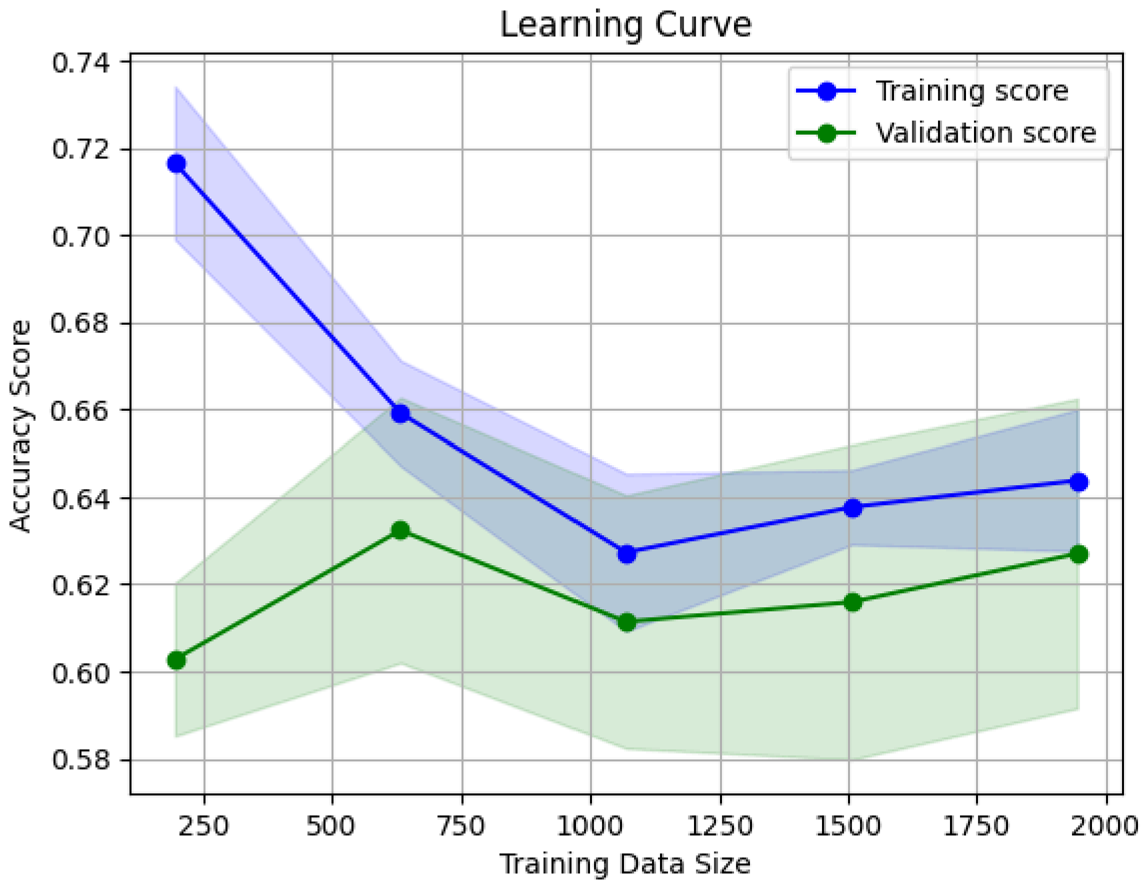
```
# Generating the confusion matrix
cm = confusion_matrix(y_test, y_pred_test_Bst)
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

# Printing the classification report
print(classification_report(y_test, y_pred_test_Bst))
```




```
              precision    recall  f1-score   support

           0       0.80      0.78      0.79       172
           1       0.55      0.71      0.62       150
           2       0.93      0.81      0.87       286

    accuracy                           0.78       608
   macro avg       0.76      0.77      0.76       608
weighted avg       0.80      0.78      0.79       608
```

In [395…

```
plot_learning_curves(pipeline_SVM_boosting, X_train, y_train, cv=cv)
```



Stacking

In [396…

```
logreg = LogisticRegression(max_iter=1000)
```

In [77]:

```
# Define base learners
base_learners = [
    ('svc', SVC(probability=True)),
    ('rf', RandomForestClassifier()),
    ('dt', DecisionTreeClassifier())
]


# Build the stacking classifier
stacking_clf = StackingClassifier(estimators=base_learners, final_estimator=
```

In [78]:

```
pipeline_stacking = ImblearnPipeline([('preprocessor', preprocessor),
                                  ('smote', smote),
                           ('stacking', stacking_clf)])
```

In [79]:

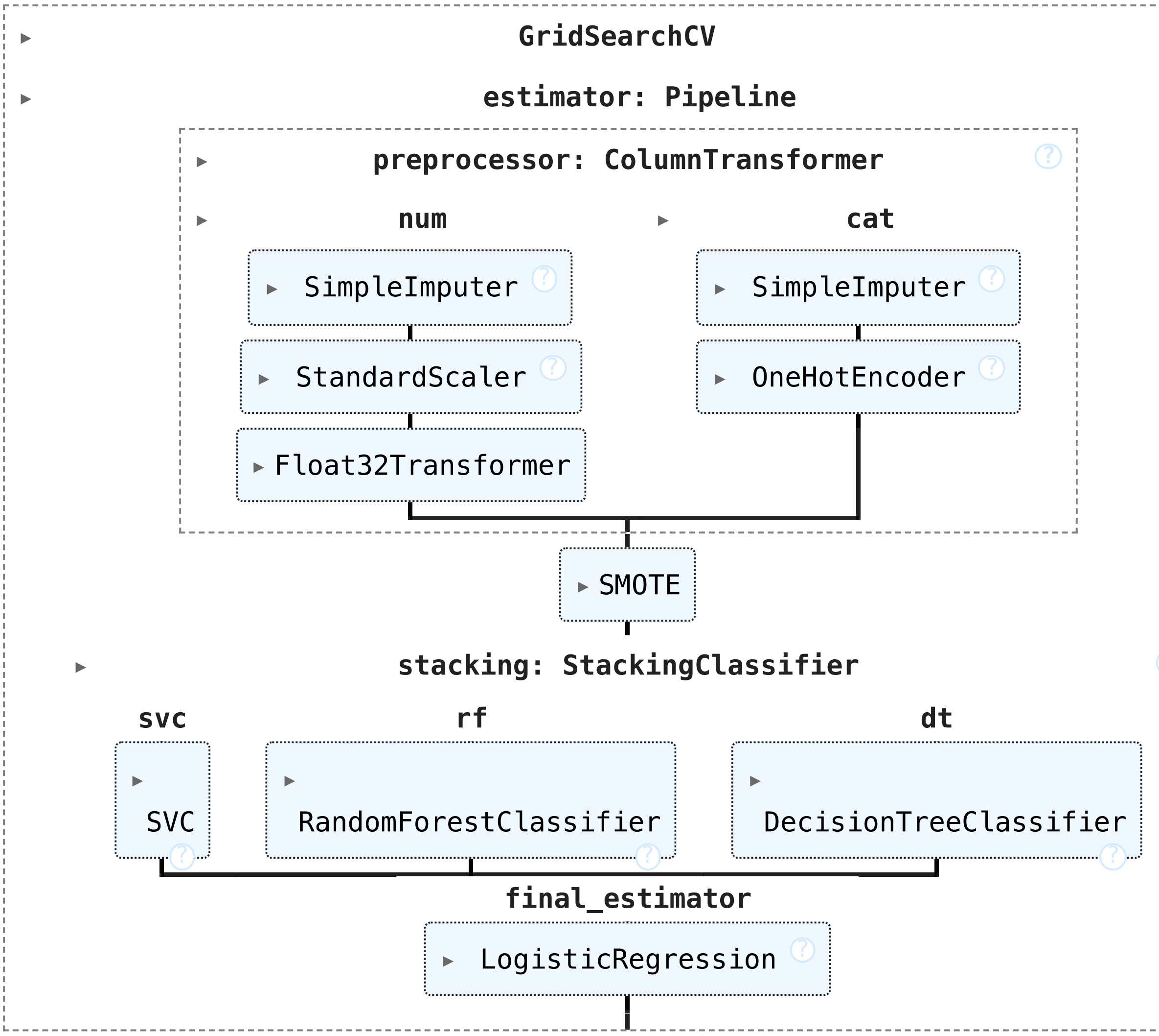
```
param_grid_stack = {
    'stacking__svc__C': [0.1, 1, 10],
    'stacking__rf__n_estimators': [10, 50, 100],
    'stacking__rf__max_depth': [None, 10, 20],
    'stacking__final_estimator__C': [0.1, 1, 10]
}
```

In [80]:

```
grid_search_stacking = GridSearchCV(pipeline_stacking, param_grid_stack, cv=
grid_search_stacking.fit(X_train, y_train)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
```

Out[80]:



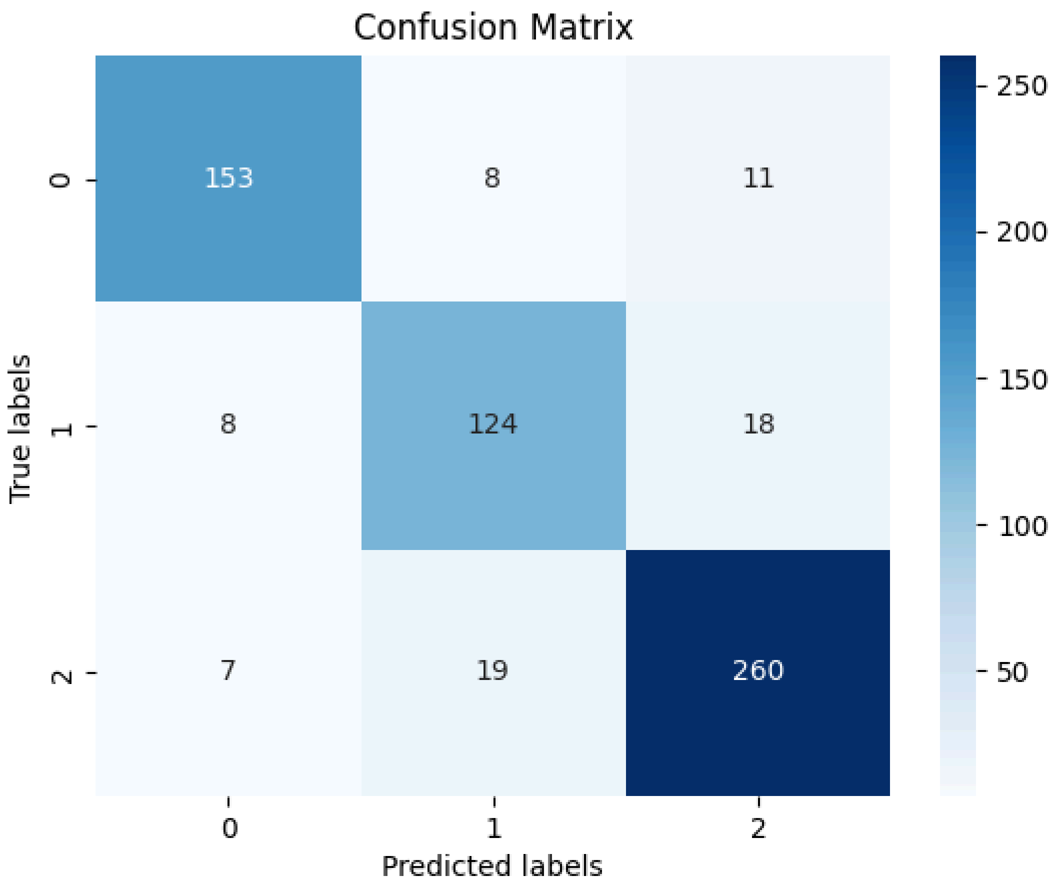
In [81]:

```
print("Best parameters:", grid_search_stacking.best_params_)
print("Best cross-validation accuracy: {:.2f}".format(grid_search_stacking.b

# Evaluate on the test set
test_score = grid_search_stacking.score(X_test, y_test)
print("Test set score: {:.2f}".format(test_score))
```

```
Best parameters: {'stacking__final_estimator__C': 0.1, 'stacking__rf__max_d
epth': None, 'stacking__rf__n_estimators': 10, 'stacking__svc__C': 1}
Best cross-validation accuracy: 0.86
Test set score: 0.88
```

In [82]:

```
# Predict on the validation set
y_pred_test_stack = grid_search_stacking.predict(X_test)

# Generating the confusion matrix
cm = confusion_matrix(y_test, y_pred_test_stack)
sns.heatmap(cm, annot=True, fmt="d", cmap='Blues')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')
plt.title('Confusion Matrix')
plt.show()

# Printing the classification report
print(classification_report(y_test, y_pred_test_stack))
```




```
              precision    recall  f1-score   support

           0       0.91      0.89      0.90       172
           1       0.82      0.83      0.82       150
           2       0.90      0.91      0.90       286

    accuracy                           0.88       608
   macro avg       0.88      0.88      0.88       608
weighted avg       0.88      0.88      0.88       608
```

Neural Networks

In [438…
```
# `y_train` initially contains integers from 0 to 2
y_train = to_categorical(y_train, num_classes=3)
y_test = to_categorical(y_test, num_classes=3)
```

In [439…
```
# Apply transformations
X_train_preprocessed = preprocessor.fit_transform(X_train)
X_test_preprocessed = preprocessor.transform(X_test)
```

In [440…
```
# determine the number of input features
n_features = X_train_preprocessed.shape[1]
n_features
```

Out[440]:
```
86
```

In [441…
```
# define model
model = Sequential()
model.add(Dense(5, activation='relu',  kernel_initializer='he_normal', kerne
model.add(Dense(86,activation='relu',  kernel_initializer='he_normal', kerne
model.add(Dense(3, activation='softmax'))
# compile the model
```

```
model.compile( optimizer='adam', loss='categorical_crossentropy', metrics=[
# fit the model
model.summary()
```

```
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-pack
ages/keras/src/layers/core/dense.py:88: UserWarning: Do not pass an `input_
shape`/`input_dim` argument to a layer. When using Sequential models, prefe
r using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential_54"

Layer (type)	Output Shape	Para
dense_171 (Dense)	(None, 5)	
dense_172 (Dense)	(None, 86)	
dense_173 (Dense)	(None, 3)	

Total params: 1,212 (4.73 KB)
Trainable params: 1,212 (4.73 KB)
Non-trainable params: 0 (0.00 B)

In [442…

```
history = model.fit(X_train_preprocessed, y_train, validation_split=0.2, epo
```

```
Epoch 1/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 2s 7ms/step - accuracy: 0.4446 - loss: 2.8567 -
val_accuracy: 0.4353 - val_loss: 2.6006
Epoch 2/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4746 - loss: 2.5323 -
val_accuracy: 0.4969 - val_loss: 2.3014
Epoch 3/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.5388 - loss: 2.2493 -
val_accuracy: 0.5893 - val_loss: 2.0037
Epoch 4/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6086 - loss: 1.9687 -
val_accuracy: 0.6612 - val_loss: 1.7295
Epoch 5/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.6788 - loss: 1.6771 -
val_accuracy: 0.7228 - val_loss: 1.4889
Epoch 6/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7461 - loss: 1.4395 -
val_accuracy: 0.7618 - val_loss: 1.3019
Epoch 7/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.7795 - loss: 1.2482 -
val_accuracy: 0.8111 - val_loss: 1.1541
Epoch 8/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8233 - loss: 1.1036 -
val_accuracy: 0.8090 - val_loss: 1.0437
Epoch 9/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8368 - loss: 0.9830 -
val_accuracy: 0.8398 - val_loss: 0.9422
Epoch 10/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8473 - loss: 0.8837 -
val_accuracy: 0.8501 - val_loss: 0.8631
Epoch 11/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8556 - loss: 0.8189 -
val_accuracy: 0.8501 - val_loss: 0.7955
Epoch 12/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8525 - loss: 0.7753 -
val_accuracy: 0.8542 - val_loss: 0.7387
Epoch 13/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8477 - loss: 0.7030 -
val_accuracy: 0.8480 - val_loss: 0.6910
Epoch 14/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8694 - loss: 0.6478 -
val_accuracy: 0.8563 - val_loss: 0.6524
Epoch 15/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8734 - loss: 0.6080 -
val_accuracy: 0.8542 - val_loss: 0.6161
Epoch 16/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8644 - loss: 0.5818 -
val_accuracy: 0.8624 - val_loss: 0.5925
Epoch 17/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8670 - loss: 0.5662 -
val_accuracy: 0.8604 - val_loss: 0.5647
Epoch 18/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8782 - loss: 0.5275 -
val_accuracy: 0.8542 - val_loss: 0.5512
Epoch 19/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8738 - loss: 0.5102 -
val_accuracy: 0.8583 - val_loss: 0.5249
Epoch 20/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8800 - loss: 0.4830 -
val_accuracy: 0.8604 - val_loss: 0.5082
Epoch 21/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8893 - loss: 0.4481 -
val_accuracy: 0.8665 - val_loss: 0.4934
Epoch 22/100
```

```
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8950 - loss: 0.4337 -
val_accuracy: 0.8645 - val_loss: 0.4839
Epoch 23/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8938 - loss: 0.4166 -
val_accuracy: 0.8645 - val_loss: 0.4716
Epoch 24/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8917 - loss: 0.4241 -
val_accuracy: 0.8747 - val_loss: 0.4599
Epoch 25/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8890 - loss: 0.4042 -
val_accuracy: 0.8686 - val_loss: 0.4559
Epoch 26/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8947 - loss: 0.4051 -
val_accuracy: 0.8665 - val_loss: 0.4483
Epoch 27/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8991 - loss: 0.3893 -
val_accuracy: 0.8624 - val_loss: 0.4403
Epoch 28/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.8935 - loss: 0.3830 -
val_accuracy: 0.8583 - val_loss: 0.4446
Epoch 29/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8976 - loss: 0.3887 -
val_accuracy: 0.8645 - val_loss: 0.4437
Epoch 30/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9148 - loss: 0.3577 -
val_accuracy: 0.8604 - val_loss: 0.4255
Epoch 31/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9010 - loss: 0.3799 -
val_accuracy: 0.8665 - val_loss: 0.4289
Epoch 32/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8935 - loss: 0.3687 -
val_accuracy: 0.8583 - val_loss: 0.4174
Epoch 33/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9006 - loss: 0.3546 -
val_accuracy: 0.8665 - val_loss: 0.4291
Epoch 34/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9005 - loss: 0.3653 -
val_accuracy: 0.8665 - val_loss: 0.4227
Epoch 35/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.8998 - loss: 0.3560 -
val_accuracy: 0.8727 - val_loss: 0.4126
Epoch 36/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9191 - loss: 0.3393 -
val_accuracy: 0.8583 - val_loss: 0.4091
Epoch 37/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9168 - loss: 0.3348 -
val_accuracy: 0.8583 - val_loss: 0.4084
Epoch 38/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9249 - loss: 0.3287 -
val_accuracy: 0.8645 - val_loss: 0.4094
Epoch 39/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9128 - loss: 0.3366 -
val_accuracy: 0.8624 - val_loss: 0.4062
Epoch 40/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9044 - loss: 0.3537 -
val_accuracy: 0.8645 - val_loss: 0.4013
Epoch 41/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9152 - loss: 0.3352 -
val_accuracy: 0.8480 - val_loss: 0.4037
Epoch 42/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9102 - loss: 0.3404 -
val_accuracy: 0.8624 - val_loss: 0.4006
Epoch 43/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9063 - loss: 0.3393 -
```

```
val_accuracy: 0.8563 - val_loss: 0.4028
Epoch 44/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9267 - loss: 0.3130 -
val_accuracy: 0.8686 - val_loss: 0.3955
Epoch 45/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9140 - loss: 0.3329 -
val_accuracy: 0.8542 - val_loss: 0.4007
Epoch 46/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9247 - loss: 0.3069 -
val_accuracy: 0.8686 - val_loss: 0.4059
Epoch 47/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9181 - loss: 0.3271 -
val_accuracy: 0.8563 - val_loss: 0.3967
Epoch 48/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9229 - loss: 0.3129 -
val_accuracy: 0.8604 - val_loss: 0.3955
Epoch 49/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9239 - loss: 0.3189 -
val_accuracy: 0.8604 - val_loss: 0.4092
Epoch 50/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9205 - loss: 0.3102 -
val_accuracy: 0.8604 - val_loss: 0.3932
Epoch 51/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9267 - loss: 0.3008 -
val_accuracy: 0.8665 - val_loss: 0.4072
Epoch 52/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9292 - loss: 0.3158 -
val_accuracy: 0.8624 - val_loss: 0.3908
Epoch 53/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9171 - loss: 0.3094 -
val_accuracy: 0.8583 - val_loss: 0.3927
Epoch 54/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9273 - loss: 0.3047 -
val_accuracy: 0.8665 - val_loss: 0.3951
Epoch 55/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9341 - loss: 0.2959 -
val_accuracy: 0.8563 - val_loss: 0.3877
Epoch 56/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9266 - loss: 0.2937 -
val_accuracy: 0.8624 - val_loss: 0.3995
Epoch 57/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9165 - loss: 0.3058 -
val_accuracy: 0.8645 - val_loss: 0.3981
Epoch 58/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9355 - loss: 0.2963 -
val_accuracy: 0.8583 - val_loss: 0.3847
Epoch 59/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9277 - loss: 0.2995 -
val_accuracy: 0.8665 - val_loss: 0.3944
Epoch 60/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9321 - loss: 0.3008 -
val_accuracy: 0.8665 - val_loss: 0.3887
Epoch 61/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9199 - loss: 0.3087 -
val_accuracy: 0.8624 - val_loss: 0.3865
Epoch 62/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9273 - loss: 0.3021 -
val_accuracy: 0.8624 - val_loss: 0.3864
Epoch 63/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9219 - loss: 0.3025 -
val_accuracy: 0.8563 - val_loss: 0.3800
Epoch 64/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9326 - loss: 0.2836 -
val_accuracy: 0.8624 - val_loss: 0.3810
```

```
Epoch 65/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9251 - loss: 0.2890 -
val_accuracy: 0.8706 - val_loss: 0.3944
Epoch 66/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9345 - loss: 0.2816 -
val_accuracy: 0.8583 - val_loss: 0.3783
Epoch 67/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9294 - loss: 0.2864 -
val_accuracy: 0.8624 - val_loss: 0.3836
Epoch 68/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9341 - loss: 0.2816 -
val_accuracy: 0.8624 - val_loss: 0.3725
Epoch 69/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9370 - loss: 0.2785 -
val_accuracy: 0.8542 - val_loss: 0.3794
Epoch 70/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9342 - loss: 0.2789 -
val_accuracy: 0.8706 - val_loss: 0.3858
Epoch 71/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9355 - loss: 0.2910 -
val_accuracy: 0.8645 - val_loss: 0.3753
Epoch 72/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9286 - loss: 0.2749 -
val_accuracy: 0.8563 - val_loss: 0.3857
Epoch 73/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9159 - loss: 0.3042 -
val_accuracy: 0.8645 - val_loss: 0.3824
Epoch 74/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9270 - loss: 0.2793 -
val_accuracy: 0.8686 - val_loss: 0.3801
Epoch 75/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9314 - loss: 0.2786 -
val_accuracy: 0.8624 - val_loss: 0.3848
Epoch 76/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9207 - loss: 0.3003 -
val_accuracy: 0.8542 - val_loss: 0.3759
Epoch 77/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9313 - loss: 0.2823 -
val_accuracy: 0.8665 - val_loss: 0.3691
Epoch 78/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9252 - loss: 0.2923 -
val_accuracy: 0.8542 - val_loss: 0.3730
Epoch 79/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9276 - loss: 0.2856 -
val_accuracy: 0.8706 - val_loss: 0.3811
Epoch 80/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9414 - loss: 0.2685 -
val_accuracy: 0.8645 - val_loss: 0.3893
Epoch 81/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9379 - loss: 0.2706 -
val_accuracy: 0.8583 - val_loss: 0.3738
Epoch 82/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9364 - loss: 0.2671 -
val_accuracy: 0.8624 - val_loss: 0.3802
Epoch 83/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9242 - loss: 0.2861 -
val_accuracy: 0.8665 - val_loss: 0.3896
Epoch 84/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9400 - loss: 0.2654 -
val_accuracy: 0.8645 - val_loss: 0.3703
Epoch 85/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9356 - loss: 0.2674 -
val_accuracy: 0.8665 - val_loss: 0.3764
Epoch 86/100
```

```
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9415 - loss: 0.2618 -
val_accuracy: 0.8624 - val_loss: 0.3743
Epoch 87/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9337 - loss: 0.2659 -
val_accuracy: 0.8645 - val_loss: 0.3675
Epoch 88/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9444 - loss: 0.2534 -
val_accuracy: 0.8624 - val_loss: 0.3701
Epoch 89/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9315 - loss: 0.2656 -
val_accuracy: 0.8665 - val_loss: 0.3748
Epoch 90/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9310 - loss: 0.2852 -
val_accuracy: 0.8686 - val_loss: 0.3848
Epoch 91/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9413 - loss: 0.2519 -
val_accuracy: 0.8604 - val_loss: 0.3679
Epoch 92/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.9268 - loss: 0.2700 -
val_accuracy: 0.8686 - val_loss: 0.3751
Epoch 93/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9469 - loss: 0.2515 -
val_accuracy: 0.8563 - val_loss: 0.3667
Epoch 94/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9393 - loss: 0.2598 -
val_accuracy: 0.8665 - val_loss: 0.3687
Epoch 95/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9314 - loss: 0.2714 -
val_accuracy: 0.8604 - val_loss: 0.3701
Epoch 96/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9449 - loss: 0.2523 -
val_accuracy: 0.8563 - val_loss: 0.3678
Epoch 97/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9294 - loss: 0.2690 -
val_accuracy: 0.8583 - val_loss: 0.3629
Epoch 98/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9372 - loss: 0.2505 -
val_accuracy: 0.8686 - val_loss: 0.3695
Epoch 99/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 4ms/step - accuracy: 0.9304 - loss: 0.2719 -
val_accuracy: 0.8624 - val_loss: 0.3741
Epoch 100/100
61/61 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step - accuracy: 0.9331 - loss: 0.2610 -
val_accuracy: 0.8604 - val_loss: 0.3834
```

In [443…

```
# evaluate the model
loss, acc = model.evaluate(X_test_preprocessed, y_test, verbose=0)
print('Test Accuracy: %.3f' % acc)
```

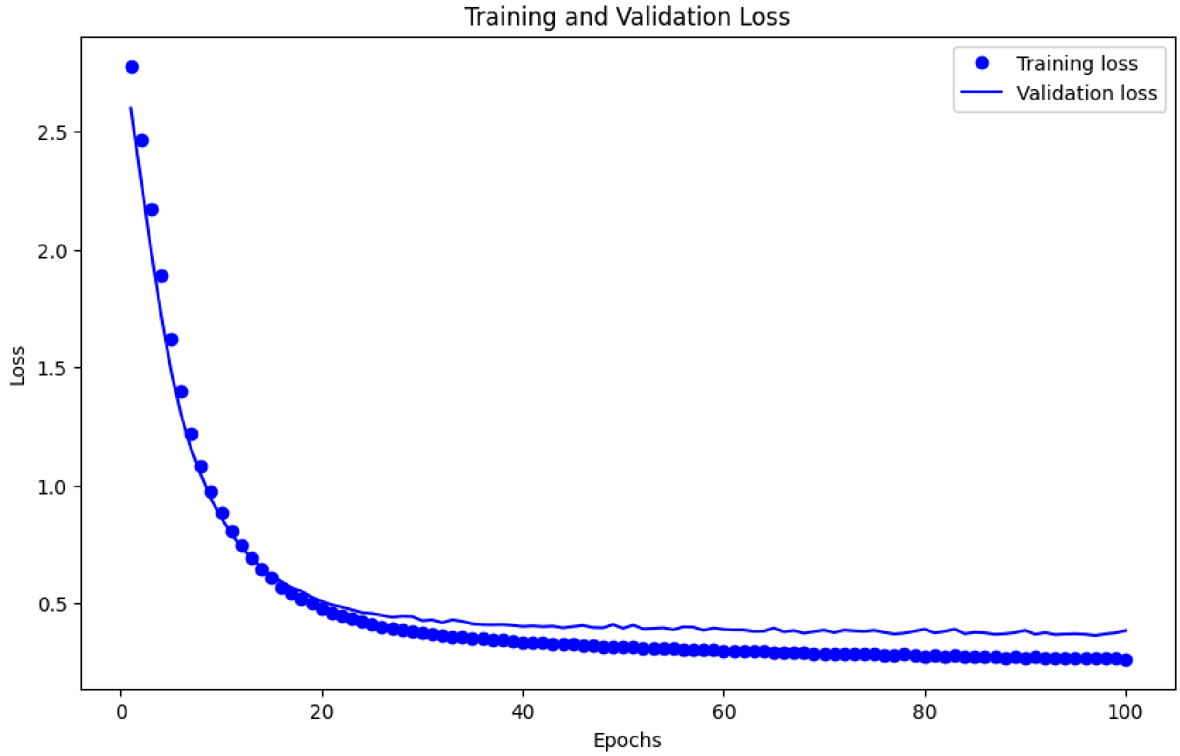
```
Test Accuracy: 0.877
```

In [444…

```
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(loss) + 1)
```

In [445…

```
plt.figure(figsize=(10, 6))
plt.plot(epochs, loss, 'bo', label='Training loss')  # 'bo' gives us blue do
plt.plot(epochs, val_loss, 'b', label='Validation loss')  # 'b' gives us a s
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```



In [448…
```
predictions_nn = model.predict(X_test_preprocessed)
predictions_nn = np.argmax(predictions_nn, axis=1)
y_test_class = np.argmax(y_test, axis=1)
```

```
19/19 ━━━━━━━━━━━━━━━━━━━━ 0s 2ms/step
```

In [451…
```
print(classification_report(y_test_class, predictions_nn, target_names=['Cla
```

```
              precision    recall  f1-score   support

     Class 1       0.94      0.87      0.90       172
     Class 2       0.86      0.76      0.81       150
     Class 3       0.85      0.94      0.90       286

    accuracy                           0.88       608
   macro avg       0.88      0.86      0.87       608
weighted avg       0.88      0.88      0.88       608
```

Results

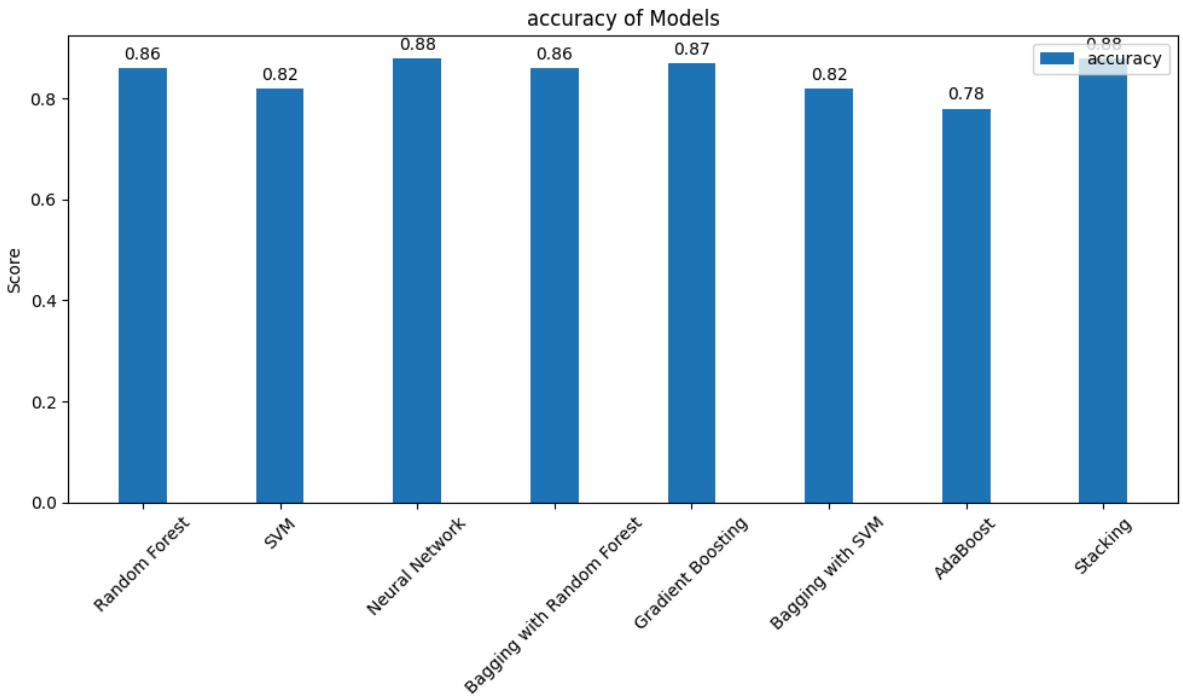
In [455…
```
model_metrics = {
    'Random Forest': {'accuracy': 0.86, 'precision_macro': 0.85, 'recall_mac
    'SVM': {'accuracy': 0.82, 'precision_macro': 0.82, 'recall_macro': 0.83,
    'Neural Network': {'accuracy': 0.88, 'precision_macro': 0.88, 'recall_ma
    'Bagging with Random Forest': {'accuracy': 0.86, 'precision_macro': 0.85
    'Gradient Boosting': {'accuracy': 0.87, 'precision_macro': 0.85, 'recall
    'Bagging with SVM': {'accuracy': 0.82, 'precision_macro': 0.83, 'recall_
    'AdaBoost': {'accuracy': 0.78, 'precision_macro': 0.76, 'recall_macro':
    'Stacking': {'accuracy': 0.88, 'precision_macro': 0.88, 'recall_macro':
}
```

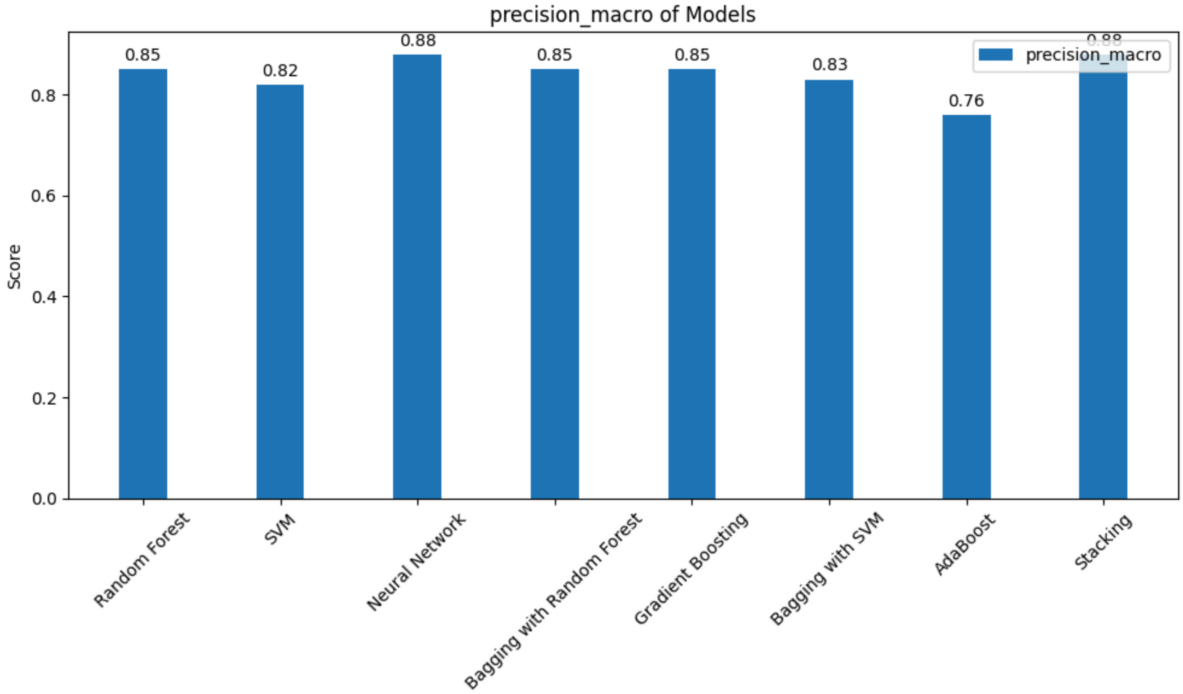
In [456…
```
models_ = list(model_metrics.keys())
accuracies = [model_metrics[model_]['accuracy'] for model_ in models_]
precision_macros = [model_metrics[model_]['precision_macro'] for model_ in m
recall_macros = [model_metrics[model_]['recall_macro'] for model_ in models_
f1_macros = [model_metrics[model_]['f1_macro'] for model_ in models_]
precision_weighteds = [model_metrics[model_]['precision_weighted'] for mode
```

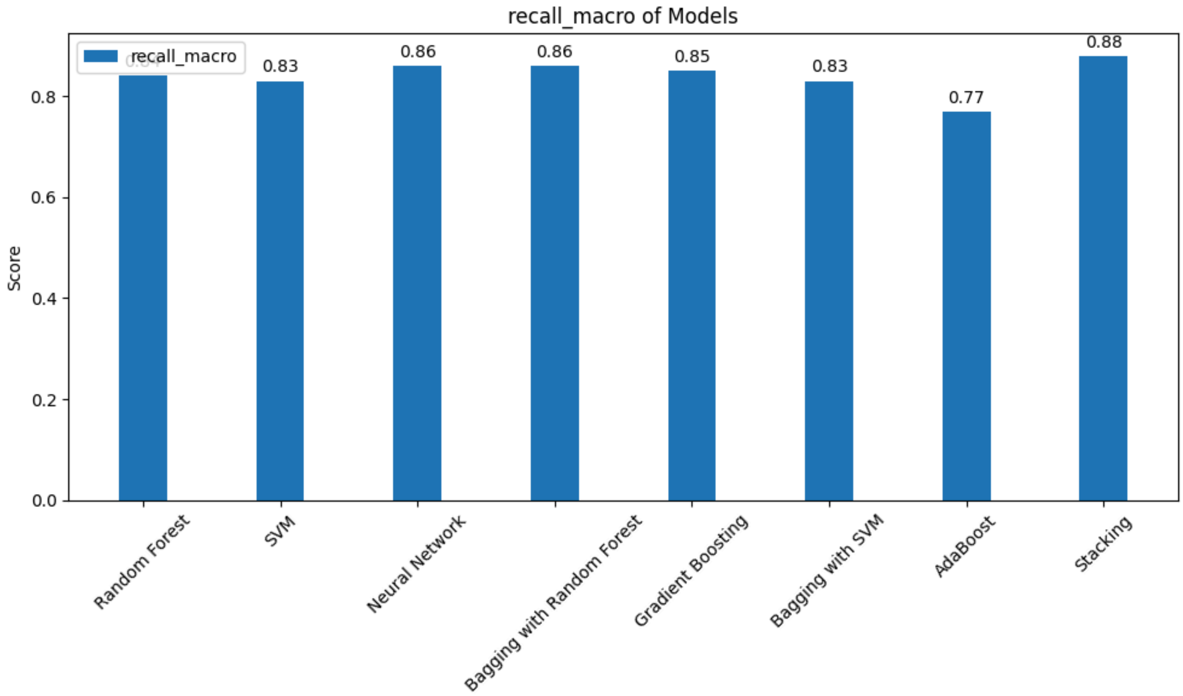
```
recall_weighteds = [model_metrics[model_]['recall_weighted'] for model_ in r
f1_weighteds = [model_metrics[model_]['f1_weighted'] for model_ in models_]
```

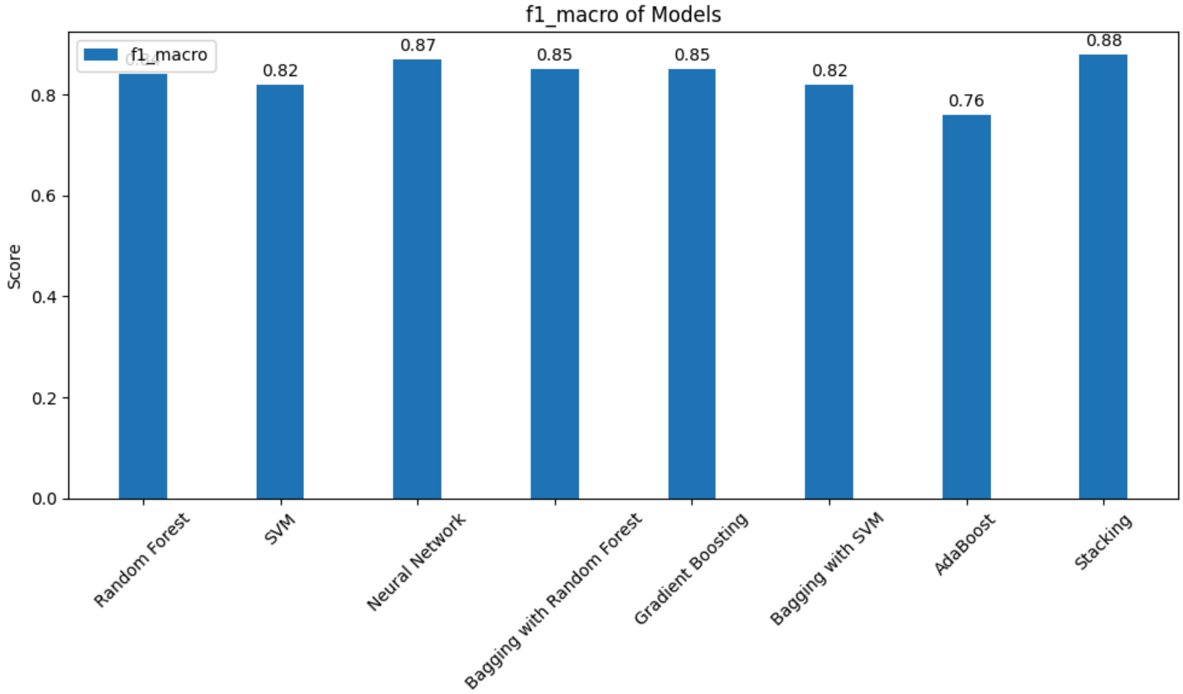
In [457…

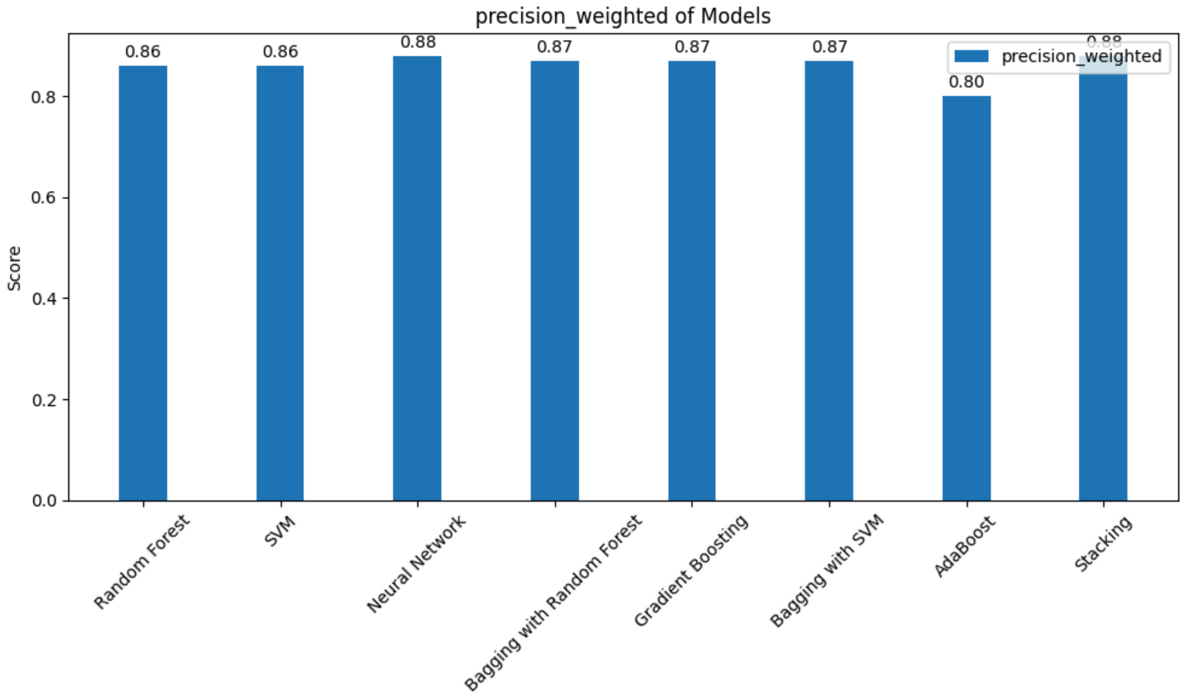
```
models_ = list(model_metrics.keys())
metrics = ['accuracy', 'precision_macro', 'recall_macro', 'f1_macro', 'preci
data = {metric: [model_metrics[model_][metric] for model_ in models_] for me

x = np.arange(len(models_))  # the label locations
width = 0.35  # the width of the bars

# Loop through each metric and create an individual plot
for metric, values in data.items():
    plt.figure(figsize=(10, 6))  # set the size of the figure
    rects = plt.bar(x, values, width, label=metric)
    plt.ylabel('Score')
    plt.title(f'{metric} of Models')
    plt.xticks(x, models_, rotation=45)
    plt.legend()

    # Attach a text label above each bar in rects, displaying its height
    for rect in rects:
        height = rect.get_height()
        plt.annotate(f'{height:.2f}',
                     xy=(rect.get_x() + rect.get_width() / 2, height),
                     xytext=(0, 3),  # 3 points vertical offset
                     textcoords="offset points",
                     ha='center', va='bottom')

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

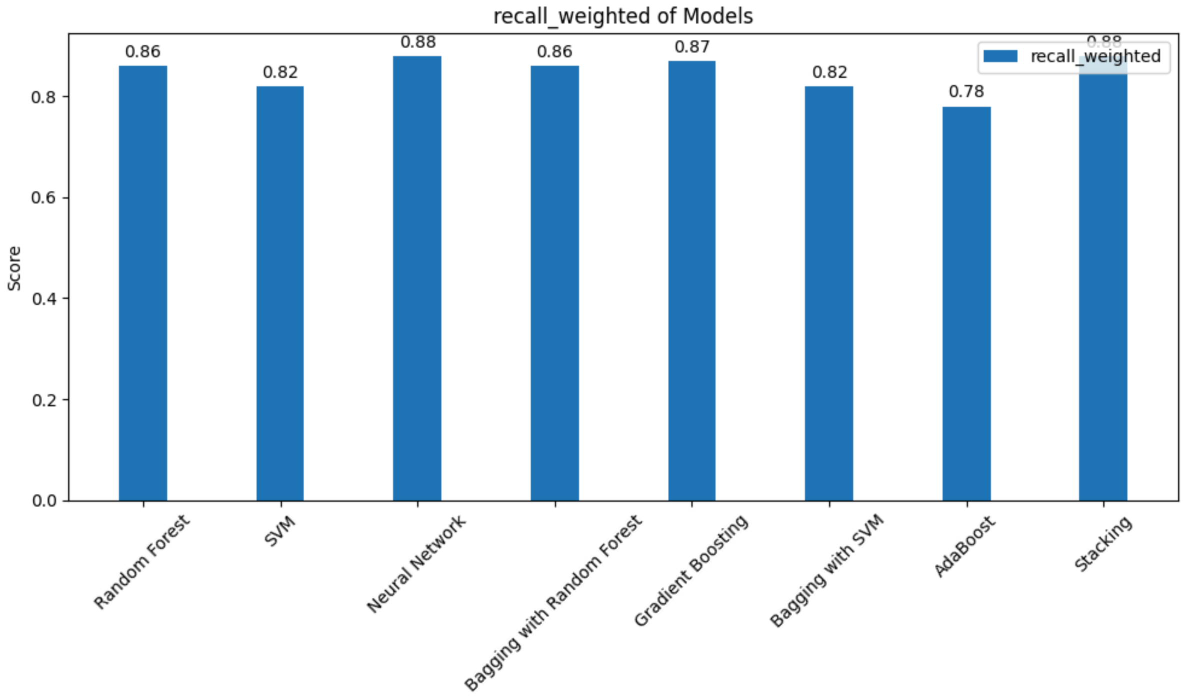


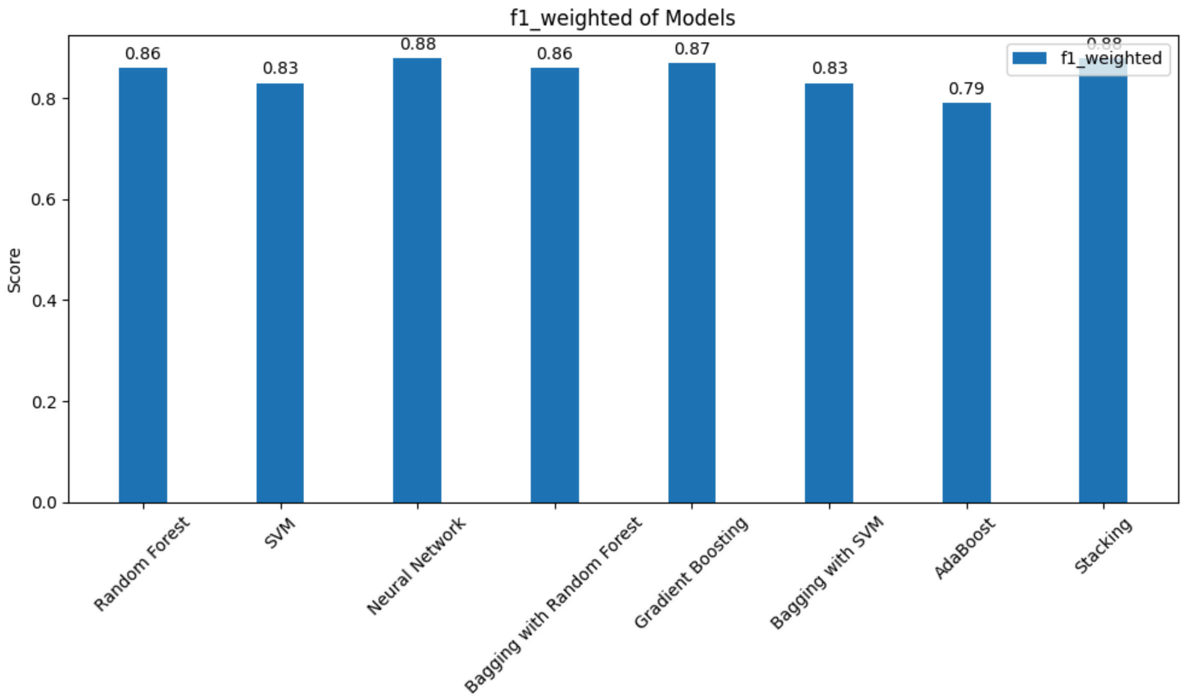

precision_macro of Models
precision_macro
Score
0.0
0.2
0.4
0.6
0.8
0.85
0.82
0.88
0.85
0.85
0.83
0.76
0.88
Random Forest
SVM
Neural Network
Bagging with Random Forest
Gradient Boosting
Bagging with SVM
AdaBoost
Stacking


recall_macro of Models
recall_macro
Score
0.0
0.2
0.4
0.6
0.8
0.83
0.86
0.86
0.85
0.83
0.77
0.88
Random Forest
SVM
Neural Network
Bagging with Random Forest
Gradient Boosting
Bagging with SVM
AdaBoost
Stacking


f1_macro of Models
f1_macro
Score
0.0
0.2
0.4
0.6
0.8
0.82
0.87
0.85
0.85
0.82
0.76
0.88
Random Forest
SVM
Neural Network
Bagging with Random Forest
Gradient Boosting
Bagging with SVM
AdaBoost
Stacking


precision_weighted of Models
precision_weighted
Score
0.0
0.2
0.4
0.6
0.8
0.86
0.86
0.88
0.87
0.87
0.87
0.80
Random Forest
SVM
Neural Network
Bagging with Random Forest
Gradient Boosting
Bagging with SVM
AdaBoost
Stacking


recall_weighted of Models
recall_weighted
Score
0.0
0.2
0.4
0.6
0.8
0.86
0.82
0.88
0.86
0.87
0.82
0.78
0.88
Random Forest
SVM
Neural Network
Bagging with Random Forest
Gradient Boosting
Bagging with SVM
AdaBoost
Stacking


f1_weighted of Models
f1_weighted
Score
0.0
0.2
0.4
0.6
0.8
0.86
0.83
0.88
0.86
0.87
0.83
0.79
0.88
Random Forest
SVM
Neural Network
Bagging with Random Forest
Gradient Boosting
Bagging with SVM
AdaBoost
Stacking

Conclusion

This code demonstrated the use of machine learning techniques to predict football match outcomes based on team attributes, average player ratings, historical win rates, and goal statistics from recent matches. Among the models tested, the Stacking classifier and the Neural Network were the most effective, achieving high accuracy and robustness in prediction. Simpler models like Random Forest and Gradient Boosting also performed better, suggesting a trade-off between model complexity and interpretability. Future work could explore getting more data sources such as player-level data, match conditions to enhance the predictive power of the models. However, while this predictive modelling presents a power tool for forecasting football match outcomes, it requires careful implementation, continuous refinement and a balanced consideration of risk and reward to support decision making in the dynamic field of sports.