

1.0 INTRODUCTION

Every year there are over 2,000+ matches played in Europe. Many factors determine the outcome of these matches. Some of these factors include players, teams, infrastructure, strategy etc. Making a prediction on a match outcome is quite difficult due to many factors to be put into consideration.

The project is developed to analyze previous matches played and predict future matches between two teams. Since the teams are the target, we will build our predictors based on the teams statistics and features. We will design a GUI that will enable a user select both the home and away team and generate the predictors based on the selected teams to predict the match outcome.

2.0 DATASET

The dataset is an sqlite database file with 7 tables. The tables are listed as follows:

1. Country: This table has 11 records and 2 columns that contains the list of countries in europe soccer league.
2. League : This table has 11 records and 3 columns that contains the list of soccer leagues in a selected country. To be able to fetch the leagues for a country, there has to be a merge with the league and match table as there is no direct reference of the league to the country.
3. Match : This table has 25,979 records and 114 columns. It contains all match history information like the players that played for both the home and away teams, the goals scored, bookkeeper odds, fouls, set kicks, cards.
4. Player : This table has 11,060 records and 7 columns. It contains the list of players under a team.
5. Player_Attributes : This table has 183,978 records and 42 columns. It contains all players attributes such as speed, accuracy, defense skills, dribbling, height, weight, header, shot etc.It has a player_id foreign key to reference the player table.
6. Team : This table has 299 records and 5 columns. It contains the list of teams in a selected league.
7. Team_Attributes : This table has 1458 records and 25 columns. It contains detailed information about the teams such as defence levels,attack levels, build up play speeds, possessions.It has a team_id foreign key to reference the team table.

3.0 ALGORITHMS (DECISION TREE)

Decision Tree Mining is a type of data mining technique that is used to build Classification Models. It builds classification models in the form of a tree-like structure, just like its name. This type of mining belongs to supervised class learning.

Gini Index: It is calculated by subtracting the sum of squared probabilities of each class from one. It favors larger partitions and is easy to implement whereas information gain favors smaller partitions with distinct values.

The mathematical formula for the Decision tree gini index is:

$$\text{Gini} = 1 - \sum_{i=1}^n (p_i)^2$$

The degree of gini index varies from 0 to 1,

Where 0 depicts that all the elements be allied to a certain class, or only one class exists there.

The gini index of value as 1 signifies that all the elements are randomly distributed across various classes, and A value of 0.5 denotes the elements are uniformly distributed into some classes.

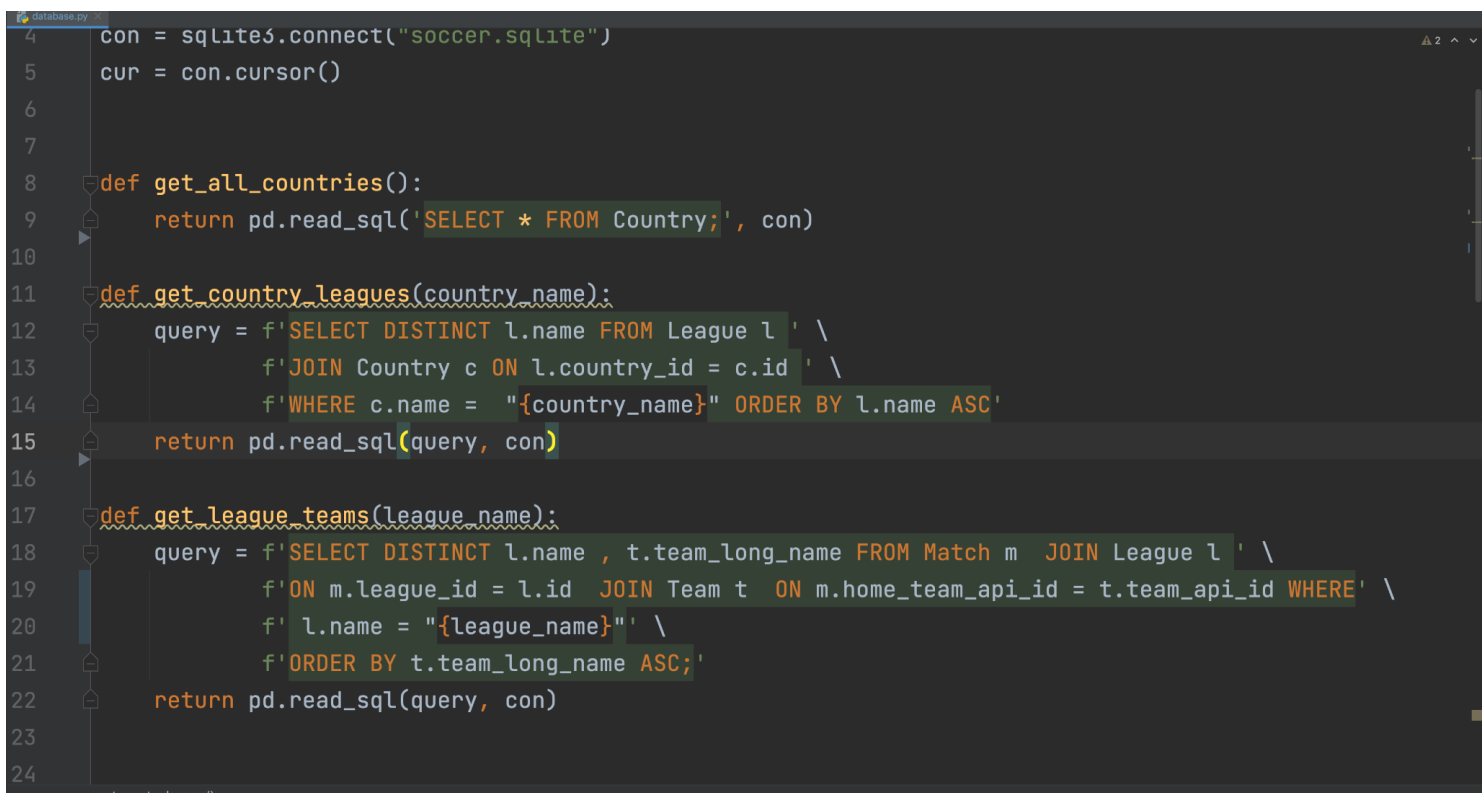
4.0 EXPERIMENTAL SETUP

This section discusses the codes we wrote in order to execute the program. It covers both the GUI and the model prediction.

4.1 Data Preprocessing

This has a list of methods that manage the access of data from the dataset. Below are the methods:

4.1.1 : GUI Data

A screenshot of a code editor with a dark background and light-colored text. The code is written in Python and uses the pandas library to interact with a SQLite database. The code is organized into four lines of code, each starting with a line number from 4 to 24. The code defines three methods: get_all_countries(), get_country_leagues(), and get_league_teams(). The first method connects to a SQLite database named 'soccer.sqlite' and returns a DataFrame of all countries. The second method returns a DataFrame of all leagues in a specific country. The third method returns a DataFrame of all teams in a specific league. The code uses f-strings to format SQL queries and the read_sql() method to execute the queries and return the results as DataFrames.

```
4 con = sqlite3.connect("soccer.sqlite")
5 cur = con.cursor()
6
7
8 def get_all_countries():
9     return pd.read_sql('SELECT * FROM Country;', con)
10
11 def get_country_leagues(country_name):
12     query = f'SELECT DISTINCT l.name FROM League l ' \
13             f'JOIN Country c ON l.country_id = c.id ' \
14             f'WHERE c.name = "{country_name}" ORDER BY l.name ASC'
15     return pd.read_sql(query, con)
16
17 def get_league_teams(league_name):
18     query = f'SELECT DISTINCT l.name , t.team_long_name FROM Match m JOIN League l ' \
19             f'ON m.league_id = l.id JOIN Team t ON m.home_team_api_id = t.team_api_id WHERE ' \
20             f'l.name = "{league_name}" ' \
21             f'ORDER BY t.team_long_name ASC;'
22     return pd.read_sql(query, con)
23
24
```

1. get_all_countries() : This fetches the list of countries from the sqlite dataset in a dataframe. No parameter is passed to this method.
2. get_country_leagues() : This fetches all the leagues in a country. The country name parameter is passed.

3. `get_league_teams()` : This fetches all the teams in a league by joining with the match table and removing duplicates.

4.1.2: Match Predictors

```
24
25 def get_match_predictors():
26     query = f'SELECT CASE WHEN home_team_goal > away_team_goal THEN 1 WHEN home_team_goal < away_team_goal THEN 2 ' \
27             f'ELSE 0 END AS Match_Outcome, ht_buildUpPlaySpeed, ht_buildUpPlayDribbling, ht_buildUpPlayPassing, ht_chanceCreationPassing, ht_chanceCreationCrossing, ' \
28             f'ht_chanceCreationShooting, ht_defencePressure, ht_defenceAggression, ht_defenceTeamWidth, at_buildUpPlaySpeed, at_buildUpPlayDribbling, ' \
29             f'at_buildUpPlayPassing, ' \
30             f'at_chanceCreationPassing, at_chanceCreationCrossing, at_chanceCreationShooting, at_defencePressure, at_defenceAggression, at_defenceTeamWidth ' \
31             f'FROM Match m JOIN ( SELECT team_api_id, AVG(buildUpPlaySpeed) AS ht_buildUpPlaySpeed, ' \
32             f'AVG(buildUpPlayDribbling) AS ht_buildUpPlayDribbling, AVG(buildUpPlayPassing) AS ht_buildUpPlayPassing, ' \
33             f'AVG(chanceCreationPassing) AS ht_chanceCreationPassing, AVG(chanceCreationCrossing) AS ht_chanceCreationCrossing, ' \
34             f'AVG(chanceCreationShooting) AS ht_chanceCreationShooting, ' \
35             f'AVG(defencePressure) AS ht_defencePressure, ' \
36             f'AVG(defenceAggression) AS ht_defenceAggression, AVG(defenceTeamWidth) AS ht_defenceTeamWidth FROM Team_Attributes ' \
37             f'GROUP BY team_api_id ) ht_attr ON ht_attr.team_api_id = home_team_api_id JOIN ' \
38             f'(SELECT team_api_id, AVG(buildUpPlaySpeed) AS at_buildUpPlaySpeed, AVG(buildUpPlayDribbling) ' \
39             f'AS at_buildUpPlayDribbling, AVG(buildUpPlayPassing) AS at_buildUpPlayPassing, ' \
40             f'AVG(chanceCreationPassing) AS at_chanceCreationPassing, AVG(chanceCreationCrossing) ' \
41             f'AS at_chanceCreationCrossing, AVG(chanceCreationShooting) AS at_chanceCreationShooting, ' \
42             f'AVG(defencePressure) AS at_defencePressure, AVG(defenceAggression) AS at_defenceAggression, ' \
43             f'AVG(defenceTeamWidth) AS at_defenceTeamWidth FROM Team_Attributes GROUP BY team_api_id ) ' \
44             f'at_attr ON at_attr.team_api_id = away_team_api_id;'
45     return pd.read_sql(query, con)
46
```

The figure 4.1.2 shows the `get_match_predictors()` method that creates the match predictors which the decision tree classifier trains. The method returns the team statistics and ratings.

4.1.3 Match Target

```
47 def get_team_predictors(home_team_name, away_team_name):
48     query = f'SELECT ht_buildUpPlaySpeed, ht_buildUpPlayDribbling, ht_buildUpPlayPassing, ht_chanceCreationPassing, ' \
49             f' ht_chanceCreationCrossing, ht_chanceCreationShooting, ht_defencePressure, ht_defenceAggression, ht_defenceTeamWidth, ' \
50             f' at_buildUpPlaySpeed, at_buildUpPlayDribbling, at_buildUpPlayPassing, at_chanceCreationPassing, at_chanceCreationCrossing, at_chanceCreationShooting, ' \
51             f' at_defencePressure, at_defenceAggression, at_defenceTeamWidth ' \
52             f' FROM (SELECT AVG(buildUpPlaySpeed) AS ht_buildUpPlaySpeed, AVG(buildUpPlayDribbling) AS ht_buildUpPlayDribbling, AVG(buildUpPlayPassing) ' \
53             f' AS ht_buildUpPlayPassing, AVG(chanceCreationPassing) AS ht_chanceCreationPassing, ' \
54             f' AVG(chanceCreationCrossing) AS ht_chanceCreationCrossing, AVG(chanceCreationShooting) AS ht_chanceCreationShooting, ' \
55             f' AVG(defencePressure) AS ht_defencePressure, AVG(defenceAggression) AS ht_defenceAggression, AVG(defenceTeamWidth) AS ht_defenceTeamWidth ' \
56             f' FROM Team_Attributes home_attr JOIN Team home_team ' \
57             f' ON home_attr.team_api_id = home_team.team_api_id ' \
58             f' WHERE team_long_name = "{home_team_name}" ) ht_attr JOIN ( ' \
59             f' SELECT AVG(buildUpPlaySpeed) AS at_buildUpPlaySpeed, AVG(buildUpPlayDribbling) AS at_buildUpPlayDribbling, AVG(buildUpPlayPassing) ' \
60             f' AS at_buildUpPlayPassing, AVG(chanceCreationPassing) AS ' \
61             f' AS at_chanceCreationPassing, AVG(chanceCreationCrossing) AS at_chanceCreationCrossing, ' \
62             f' AVG(chanceCreationShooting) AS at_chanceCreationShooting, AVG(defencePressure) AS at_defencePressure, AVG(defenceAggression) ' \
63             f' AS at_defenceAggression, AVG(defenceTeamWidth) AS at_defenceTeamWidth ' \
64             f' FROM Team_Attributes away_attr JOIN Team away_team ON away_attr.team_api_id = away_team.team_api_id ' \
65             f' WHERE team_long_name = "{away_team_name}" ) at_attr ON 1=1 ;'
66     return pd.read_sql(query, con)
67
68
69
70
```

get_team_predictors()

The figure 4.1.3 shows the `get_team_predictors()` method that creates the match target for the decision tree classifier. The method returns the team statistics and ratings of the selected home and away team from the GUI.

4.2 GUI Code 1:

```
16 class play_match_window(QDialog):
17     def __init__(self):
18         super(play_match_window, self).__init__()
19         self.setWindowTitle("DATS6103 GROUP PROJECT - TEAM 3 (SOCCER MATCH PREDICTION)")
20         self.setFixedWidth(800)
21         self.setFixedHeight(1000)
22         qtRectangle = self.frameGeometry()
23         centerPoint = QDesktopWidget().availableGeometry().center()
24         qtRectangle.moveCenter(centerPoint)
25         self.move(qtRectangle.topLeft())
26         self.formGroupBox = QGroupBox("PREDICT SOCCER MATCH")
27         self.homeTeamCountryComboBox = QComboBox()
28         self.awayTeamCountryComboBox = QComboBox()
29         self.homeTeamLeagueComboBox = QComboBox()
30         self.awayTeamLeagueComboBox = QComboBox()
31         self.homeTeamComboBox = QComboBox()
32         self.awayTeamComboBox = QComboBox()
33         self.result = QPlainTextEdit()
34         self.result.setFixedWidth(500)
35         self.result.setFixedHeight(200)
36         self.result.setDisabled(True)
37         self.homeTeamCountryComboBox.addItem(database.get_all_countries().name)
38         self.homeTeamCountryComboBox.currentIndexChanged.connect(self.getHomeCountryLeagues)
39         self.homeTeamLeagueComboBox.addItem(
40             database.get_country_leagues(self.homeTeamCountryComboBox.currentText()).name)
41         self.homeTeamLeagueComboBox.currentIndexChanged.connect(self.getHomeLeagueTeams)
42         self.homeTeamComboBox.clear()
43         self.homeTeamComboBox.addItem(
44             database.get_league_teams(self.homeTeamLeagueComboBox.currentText()).team_long_name)
45         self.awayTeamCountryComboBox.addItem(database.get_all_countries().name)
46         self.awayTeamCountryComboBox.currentIndexChanged.connect(self.getAwayCountryLeagues)
47         self.awayTeamLeagueComboBox.addItem(
48             database.get_country_leagues(self.awayTeamCountryComboBox.currentText()).name)
49         self.awayTeamLeagueComboBox.currentIndexChanged.connect(self.getAwayLeagueTeams)
50         self.awayTeamComboBox.clear()
51         self.awayTeamComboBox.addItem(
52             database.get_league_teams(self.awayTeamLeagueComboBox.currentText()).team_long_name)
53         self.platMatchForm()
```

1. We first declared the class and its constructor. I also defined the window title, fixed height and fixed width. We used the frame geometry to centralize the form when executed on any computer. I declared a group box control to embed all other controls inside it.
2. we used the combo box control to create the following dropdowns
 - a. Home Team Country Combo
 - b. Home Team League Combo

- c. Home Team Combo
- d. Away Team Country Combo
- e. Away Team League Combo
- f. Away Team Combo

We also created a plan edit field to display the results of the predicted model and I also sized it appropriately and disabled it to make it readonly. We added a frame control to display the team's formation plot.

3. After creating all the controls, populated all the country, league and team combo with data from the dataset using sql queries in the database.py file. When a country is selected it triggers and populates the league combo and when a league is selected it triggers and populates the team combo
4. we added a button group of default pyqt5 buttons which consists of the cancel and ok buttons. The cancel button closes the form and the ok button submits the form to perform the model analysis and prediction.

4.3 GUI Code 2

```
51         self.buttonBox = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel)
52         self.buttonBox.accepted.connect(self.analyzeMatch)
53         self.buttonBox.rejected.connect(self.reject)
54         self.figure = plt.figure()
55         self.canvas = FigureCanvas(self.figure)
56         mainLayout = QVBoxLayout()
57         mainLayout.addWidget(self.formGroupBox)
58         mainLayout.addWidget(self.buttonBox)
59         mainLayout.addWidget(self.canvas)
60         self.setLayout(mainLayout)
61
62     def getHomeCountryLeagues(self):
63         self.homeTeamLeagueComboBox.clear()
64         self.homeTeamComboBox.clear()
65         self.homeTeamLeagueComboBox.addItem(
66             database.get_country_leagues(self.homeTeamCountryComboBox.currentText()).name)
67
68     def getAwayCountryLeagues(self):
69         self.awayTeamLeagueComboBox.clear()
70         self.awayTeamComboBox.clear()
71         self.awayTeamLeagueComboBox.addItem(
72             database.get_country_leagues(self.awayTeamCountryComboBox.currentText()).name)
73
74     def getHomeLeagueTeams(self):
75         self.homeTeamComboBox.clear()
76         self.homeTeamComboBox.addItem(
77             database.get_league_teams(self.homeTeamLeagueComboBox.currentText()).team_long_name)
78
79     def getAwayLeagueTeams(self):
80         self.awayTeamComboBox.clear()
81         self.awayTeamComboBox.addItem(
82             database.get_league_teams(self.awayTeamLeagueComboBox.currentText()).team_long_name)
83
```

1. We created box layout as the main form layout and i added the form group containing the controls, the button box containing the ok and cancel buttons and the canvas to display the team formation as widgets to the main layout
2. getHomeCountryLeagues: This method is triggered whenever a home team country is changed. It returns a dataframe of the list of leagues for the home team.
3. getAwayCountryLeagues: This method is triggered whenever a away team country is changed. It returns a dataframe of the list of leagues for the away team.
4. getHomeLeagueTeams: This method is triggered whenever a home team league is changed. It returns a dataframe of the list of teams in that league for the home team.

5. `getAwayLeagueTeams`: This method is triggered whenever a away team league is changed. It returns a dataframe of the list of teams in that league for the away team.

4.3 GUI Code 3

```
110
111     def platMatchForm(self):
112         layout = QFormLayout()
113         layout.addRow(QLabel("Home Team Country"), self.homeTeamCountryComboBox)
114         layout.addRow(QLabel("Home Team League"), self.homeTeamLeagueComboBox)
115         layout.addRow(QLabel("Home Team"), self.homeTeamComboBox)
116         layout.addRow(QLabel("VS"))
117         layout.addRow(QLabel("Away Team Country"), self.awayTeamCountryComboBox)
118         layout.addRow(QLabel("Away Team League"), self.awayTeamLeagueComboBox)
119         layout.addRow(QLabel("Away Team"), self.awayTeamComboBox)
120         layout.addRow(QLabel("Prediction Results"), self.result)
121         self.formGroupBox.setLayout(layout)
122
```

The `play_match_form()` is the method that finally displays the form(box layout).To make the GUI more readable we added labels for each control. We also used the `add row` also to make the controls uniform and easy to navigate. When the labels are called, it also calls its respective text edit or combo defined earlier.

4.4 Match Formation

```
def team_formation(home_team_name, away_team_name):  
    ~~~~~  
    con.row_factory = sqlite3.Row  
    cur = con.cursor()  
    ~~~~~  
    sql = f'SELECT h.home_player_1, h.home_player_2, h.home_player_3, h.home_player_4, h.home_player_5, h.home_player_6, ' \  
          f'h.home_player_7, h.home_player_8, h.home_player_9, h.home_player_10, h.home_player_11, ' \  
          f'a.away_player_1, a.away_player_2, a.away_player_3, a.away_player_4, a.away_player_5, a.away_player_6, ' \  
          f'a.away_player_7, a.away_player_8, a.away_player_9, a.away_player_10, a.away_player_11, ' \  
          f'h.home_player_X1, h.home_player_X2, h.home_player_X3, h.home_player_X4, h.home_player_X5, h.home_player_X6, ' \  
          f'h.home_player_X7, h.home_player_X8, h.home_player_X9, h.home_player_X10, h.home_player_X11, ' \  
          f'h.home_player_Y1, h.home_player_Y2, h.home_player_Y3, h.home_player_Y4, h.home_player_Y5, h.home_player_Y6, ' \  
          f'h.home_player_Y7, h.home_player_Y8, h.home_player_Y9, h.home_player_Y10, h.home_player_Y11, ' \  
          f'a.away_player_X1, a.away_player_X2, a.away_player_X3, a.away_player_X4, a.away_player_X5, a.away_player_X6, ' \  
          f'a.away_player_X7, a.away_player_X8, a.away_player_X9, a.away_player_X10, a.away_player_X11, ' \  
          f'a.away_player_Y1, a.away_player_Y2, a.away_player_Y3, a.away_player_Y4, a.away_player_Y5, a.away_player_Y6, ' \  
          f'a.away_player_Y7, a.away_player_Y8, a.away_player_Y9, a.away_player_Y10, a.away_player_Y11 ' \  
          f'FROM ' \  
          f'(' SELECT * FROM Match WHERE id = (SELECT MAX(m.id) FROM Match m JOIN Team ht ON m.home_team_api_id = ht.team_api_id AND  
          f'JOIN ' \  
          f'(' SELECT * FROM Match WHERE id = (SELECT MAX(m.id) FROM Match m JOIN Team at ON m.away_team_api_id = at.team_api_id AND  
          f'ON 1=1:' \  
    cur.execute(sql)  
    match = cur.fetchone()
```

The `team_formation(home_team_name, away_team_name)` method is used for plotting the home team vs away team formation in the application on the PyQt5 Canvas. This method is invoked when the ok button is clicked. It takes in 2 inputs: `home_team_name` and `away_team_name` which the user selects on the GUI and plots their respective formations. The player name and position data is queried from the “Team” table for the most recent match played. It is preprocessed and then plotted onto 2 subplots. We use the `scatter()` function for displaying the player position markers and the `annotate()` function for displaying the player names. We also add the field markings using the `Arc()`, `Rectangle()` methods from Matplotlib onto the subplots to make the visualization better.

4.5 Match Prediction

```
84     def analyzeMatch(self):
85         home_team = str(self.homeTeamComboBox.currentText())
86         away_team = str(self.awayTeamComboBox.currentText())
87         self.figure.clear()
88         team_formation.team_formation(home_team, away_team)
89         self.canvas.draw()
90         match_predictors_variables = database.get_match_predictors()
91         match_predictors_variables = match_predictors_variables.dropna()
92         teams_predictor_variables = database.get_team_predictors(home_team, away_team).fillna(0)
93         X = match_predictors_variables.values[:, 1:19]
94         y = match_predictors_variables.values[:, 0]
95         p = teams_predictor_variables.values[:, 0:18]
96         clf = DecisionTreeClassifier(criterion="gini")
97         clf.fit(X, y)
98         y_pred = clf.predict(p)
99         report = classification_report(y_pred, y)
100         if y_pred == 1:
101             outcome = home_team + " (Home team) wins"
102         elif y_pred == 2:
103             outcome = away_team + " (Away team) wins"
104         else:
105             outcome = "Draw"
106         self.result.setPlainText(outcome)
107         self.result.appendPlainText(report)
```

This method is triggered when the ok button is clicked. In this method the home and away team selected are collected from the form and passed to the `get_home_away_team_matches` which is also a method that gets the last n number of matches of both the home and away team played in their respective leagues. Next, the team formations plot is displayed in the canvas created earlier. And based on the teams selected the target variables and predictor variables are generated and passed to the prediction model.

The x variable is the predictor variables of all the team ratings and statistics from each match played.

The y is the target variable that has the match outcome of win, draw, loss.

The predictor variables are trained using the decision tree classifier and the prediction is obtained using the decision tree predictor.

5.0 RESULTS

5. 1 Play Match Window

PREDICT SOCCER MATCH

Home Team Country

Belgium

Home Team League

Belgium Jupiler League

Home Team

Beerschot AC

VS

Away Team Country

Belgium

Away Team League

Belgium Jupiler League

Away Team

Beerschot AC

Prediction Results

Cancel

OK

The figure 5.1 shows the main GUI of the project. This is the first page that pops up when the program is executed. The view is a window where all controls are defined. The window has a title to make it more descriptive.

Since we are playing a soccer match we decided to have both team selections for the home and away team in different combo boxes. When a country is selected, triggers a method to populate the leagues for that country. When a league is selected it also triggers a method to populate the teams under the selected league. The data used to populate the country, league and teams are called from the methods in the database.py file as described in the code subroutines above.

A text field was added below to also help in specifying the amount of matches to analyze. This is because we have a very robust database with over 300,000+ records and for the purpose of the presentation it makes sense to be able to control the range of data requested.

Lastly we have two action buttons, which are the cancel button and the Ok button. The cancel button is used to quit the whole form and the ok button is used to proceed with the model training and generating the results. Based on the team selection, the matches for both teams are generated while limiting the records to the number of matches specified,

5.2 Match Played Results

PREDICT SOCCER MATCH

Home Team Country

England

Home Team League

England Premier League

Home Team

Arsenal

VS

Away Team Country

Germany

Away Team League

Germany 1.Bundesliga

Away Team

1.FC Kaiserslautern

Prediction Results

Draw

	precision	recall	f1-score	support
0.0	1.00	0.25	0.41	23382
1.0	0.00	0.00	0.00	0
2.0	0.00	0.00	0.00	0
accuracy			0.25	23382
macro avg	0.33	0.08	0.14	23382
weighted avg	1.00	0.25	0.41	23382

Cancel

OK

From the figure 5.2, when the home and away team is selected , the generated predictors are passed to the decision tree classifier for prediction. When the result is returned, we first added the match outcome label to the plain text edit using the set text function and we further appended the results from the classification report to the plain text edit using the append function so that it will not overwrite the match outcome displayed. The match outcome can be one of the following prediction

- Home Team Wins
- Both Teams Draw
- Away Team Wins

```
if y_pred == 1:
    outcome = home_team + " (Home team) wins"
elif y_pred == 2:
    outcome = away_team + " (Away team) wins"
else:
    outcome = "Draw"
self.result.setPlainText(outcome)
self.result.appendPlainText(report)
```

play_match_window > platMatchForm()

After prediction, if the predicted value is equals to 1, then the home team wins, if the predicted value is equals to 2, then the away team wins, else it is an obvious draw.

```
/usr/local/bin/python3.9 /Users/babayegar/PycharmProjects/DATS6103/main.p
The accuracy of this prediction is 0.6383970575656488
```

	precision	recall	f1-score	support
0.0	0.56	0.53	0.54	6207
1.0	0.77	0.68	0.72	12216
2.0	0.49	0.67	0.57	4959

```
Run: main x
The accuracy of this prediction is 0.4591566162004961
```

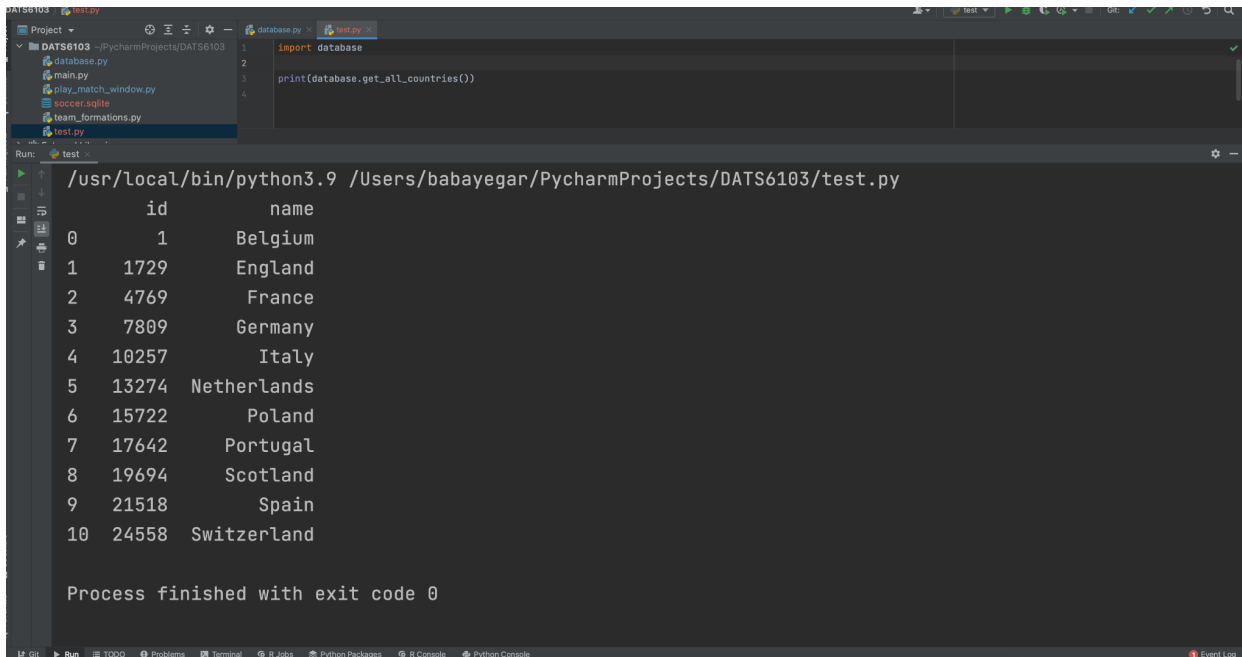
	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	0
1.0	1.00	0.46	0.63	23382
2.0	0.00	0.00	0.00	0

On the classification report, we also noticed that the accuracy will always change after each match played. This is because our dataset changes based on the selected home and away team. From a test of 20 different matches we noticed that tougher teams playing against each other always returned an accuracy of 45 to 55 %. This makes it more realistic because they are both strong teams. On the other hand, a stronger team playing a small team offered a very high accuracy of most times 64% to 89%.

We also went ahead to display the team formation returned as an axis on the canvas we created earlier. It has to be an axis rather than a plot.

5.3 Data Preprocessing Results

5.3.1 Country Data



The screenshot shows a PyCharm IDE with a project named 'DATS6103'. The file explorer on the left shows several files: 'database.py', 'main.py', 'play_match_window.py', 'soccer.sqlite', 'team_formation.py', and 'test.py'. The 'test.py' file is open in the editor, showing the following code:

```
1 import database
2
3 print(database.get_all_countries())
4
```

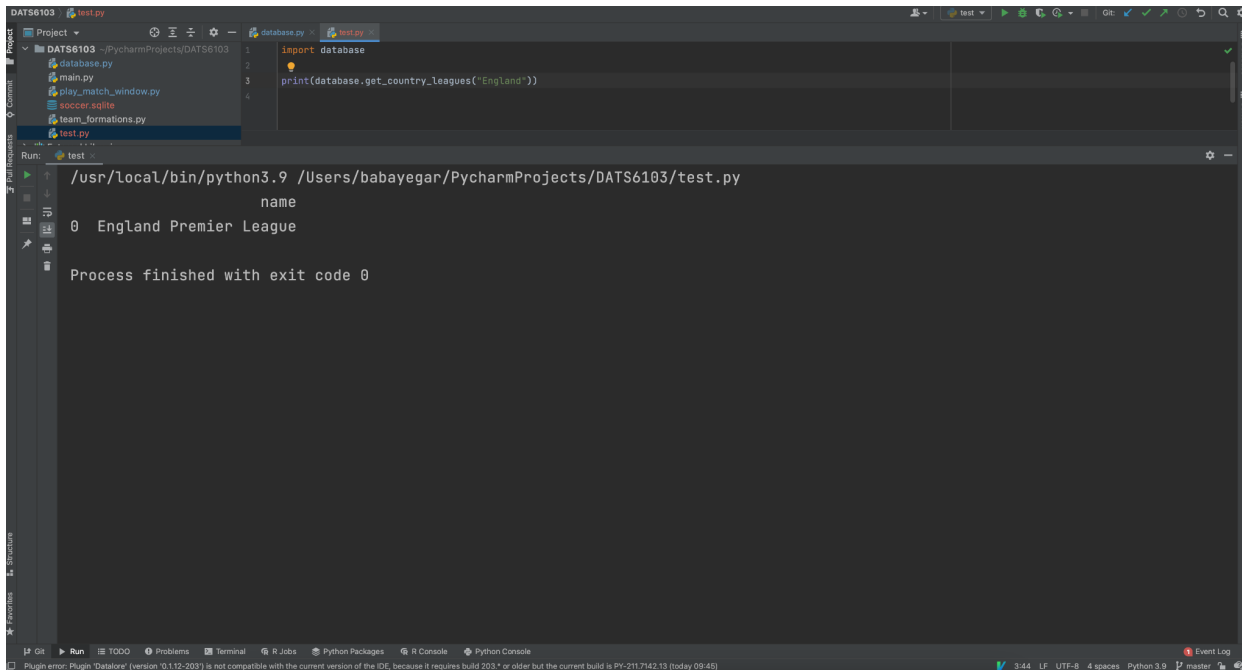
The Run console at the bottom shows the output of the script, which is a pandas DataFrame with 11 rows and 2 columns: 'id' and 'name'. The data is as follows:

	id	name
0	1	Belgium
1	1729	England
2	4769	France
3	7809	Germany
4	10257	Italy
5	13274	Netherlands
6	15722	Poland
7	17642	Portugal
8	19694	Scotland
9	21518	Spain
10	24558	Switzerland

Below the DataFrame, the console shows the message: 'Process finished with exit code 0'.

Figure 3.3.1 shows the list of countries returned from the `get_all_counries()` method is the database file. It returns both the country's id and name. When the results are obtained, it is returned in the form of a panda dataframe. This requires no parameter to be passed.

5.3.2 League Data



The screenshot shows a PyCharm IDE window for a project named 'DATS6103'. The file explorer on the left lists several files: 'database.py', 'main.py', 'play_match_window.py', 'soccer.sqlite', 'team_formation.py', and 'test.py'. The 'test.py' file is open in the editor, containing the following code:

```
1 import database
2
3 print(database.get_country_leagues("England"))
4
```

The 'Run' window at the bottom shows the execution of 'test.py' using the interpreter '/usr/local/bin/python3.9'. The output is as follows:

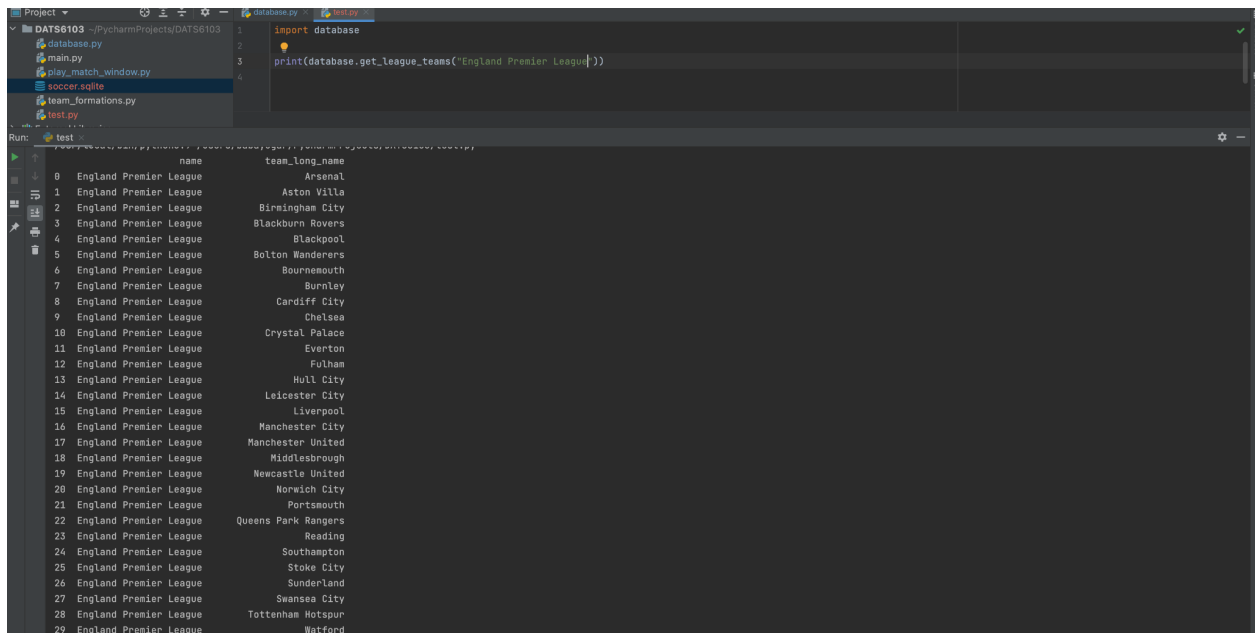
```
name
0 England Premier League

Process finished with exit code 0
```

The status bar at the bottom indicates the current file is 'test.py', line 3, column 44, with 1 line feed, 4 spaces, and Python 3.9. A small error message is visible in the bottom left corner: 'Plugin "DataGrip" (version "0.112-2021") is not compatible with the current version of the IDE, because it requires build 203.* or older but the current build is PY-211.7442.13 (today 09:45)'.

Figure 3.3.2 shows the list of leagues in a country returned from the `get_country_leagues()` method is the database file. It returns both the league name. When the results are obtained, it is returned in the form of a panda dataframe. This requires the country name parameter to be passed.

5.3.3 Team Data



The screenshot shows a PyCharm IDE with a project named 'DATS6103'. The file explorer on the left lists several files: 'database.py', 'main.py', 'play_match_window.py', 'soccer.sqlite', 'team_formation.py', and 'test.py'. The 'test.py' file is open in the editor, showing the following code:

```
1 import database
2
3 print(database.get_league_teams('England Premier League'))
```

The 'Run' window at the bottom displays the output of the script as a pandas DataFrame. The DataFrame has two columns: 'name' and 'team_long_name'. It contains 20 rows of data, all for the 'England Premier League'.

	name	team_long_name
0	England Premier League	Arsenal
1	England Premier League	Aston Villa
2	England Premier League	Birmingham City
3	England Premier League	Blackburn Rovers
4	England Premier League	Blackpool
5	England Premier League	Bolton Wanderers
6	England Premier League	Bournemouth
7	England Premier League	Burnley
8	England Premier League	Cardiff City
9	England Premier League	Chelsea
10	England Premier League	Crystal Palace
11	England Premier League	Everton
12	England Premier League	Fulham
13	England Premier League	Hull City
14	England Premier League	Leicester City
15	England Premier League	Liverpool
16	England Premier League	Manchester City
17	England Premier League	Manchester United
18	England Premier League	Middlesbrough
19	England Premier League	Newcastle United
20	England Premier League	Norwich City
21	England Premier League	Portsmouth
22	England Premier League	Queens Park Rangers
23	England Premier League	Reading
24	England Premier League	Southampton
25	England Premier League	Stoke City
26	England Premier League	Sunderland
27	England Premier League	Swansea City
28	England Premier League	Tottenham Hotspur
29	England Premier League	Watford

Figure 3.3.3 shows the list of leagues in a country returned from the `get_league_teams()` method is the database file. It returns both the league name and the team name. As said earlier, because there was no direct relationship on the team and league table, a join with the match table had to be used and duplicates dropped. When the results are obtained, it is returned in the form of a pandas dataframe. This requires the league name parameter to be passed.

6.0 LIMITATION

1. The model works in a manner using a team's performance within its own league or with teams in other European leagues to predict match outcome. So the model's prediction would not be as convincing when working on matches across leagues with significant performance difference. (EPL teams versus CSL teams)
2. The model is trained on using a full dataset containing matches from season 2008 to 2016. This means the model captures the "average" performance of a team instead of its most recent, relevant formation when making predictions. It makes sense since we want to feed the machine with as much information as possible to make statistical significant predictions, making it a trade-off between prediction consistency vs. prediction significance.
3. One potential way to address this is to assign weights to more recent matches, those in the 15-16 season for example. How to properly design weights however, becomes another issue.
4. Our current application implements only the Decision Tree algorithm for a faster execution. Being a linear classifier, it suffers limitations embedded in its family (poor performance dealing with non-Gaussian features) and its own (overfitting). We attempted to include multiple classifiers (Random Forest, Naive Bayes, SVM) in earlier versions of the application, but decided to drop them due to the random RAM overflow bug incurred.
5. Soccer is a science that we are only beginning to quantitatively measure, and there are a lot of fields that were not considered in the scope of this dataset. For instance, the condition of the grass and weather were not taken into account. Other factors such as a team's matching strategy (preserving strength to stay at particular placement in league or tournament to avoid/target a particular opponent) could be even harder to simulate.

7.0 SUMMARY & CONCLUSION

This project works on the European Soccer dataset and implements various methods and techniques in Python coding, model building and Decision Tree algorithm we learnt in Data Mining Class, to make predictions on team encounter results of win, lose or draw. Through the process we managed to learn new packages(PyQT5 for constructing GUI) and incorporated skills from other classes (Data slicing through SQL queries using sqlite3 package) and share the work and learning together as a team.

The prediction engine was trained with Decision Tree and it could yield prediction with a certain level of confidence. But the model was using the entirety of the dataset from season 08 to 16 and recent changes in players, coaches, tactics and formations could be attached with more significance. Due to other limitations including those embedded in linear classifiers as well as unaccounted external features like weather, pitch conditions and match strategies, this model is not designed to tackle “close calls” between competitive teams.

8.0 REFERENCES

1. <https://www.kaggle.com/hugomathien/soccer>
2. <https://www.geeksforgeeks.org/pyqt5-set-fix-window-size-for-height-or-width/>
3. <https://pythonprogramminglanguage.com/pyqt5-center-window/#:~:text=To%20center%20a%20Python%20PyQt,the%20center%20of%20the%20screen.>
4. https://www.tutorialspoint.com/pyqt5/pyqt5_tutorial.pdf
5. <https://datacarpentry.org/python-ecology-lesson/09-working-with-sql/index.html>
6. <https://github.com/BABAYEGAR/Data-Mining/blob/master/Demo/PyQt5/Demo/Main.py>
7. <https://blog.quantinsti.com/gini-index/>