Gauri Thakur | CY23C021 | Methodology Report | *Fantasy IPL Prediction Contest by AI Club IITM*

(Methods and code are highlighted with grey colour)

Analysis Inferences:

- Using shape I found that train and test sets are of the same dimension for both classification and prediction tasks.
- This urged me to check for similarity in data and I found the train sets were exactly the same except for the target, and both test sets were also identical. I used iloc and equals methods for this.
- Got the overview of the datasets using head and info, the descriptive statistics using describe, null values using dft.isnull().sum().to_list(), and count of unique object values in each column using dft.select_dtypes(include=['object']).nunique().
- Checked for new entries of player names in test set and found 3 new player names which were not in train set using new_players = set(dftt['Player_name']).difference(set(dft['Player_name'])). Same method for venue gave no new entries.
- Checked correlation between batting and bowling innings using corr and found direct negative correlation.

Data Preprocessing:

- Dropped unnecessary columns ['match_id','Starting_11','match_name','bowling_team_bowl', 'batting_team_bat', 'prev_wickets','prev_conceded','prev_catches','prev_Overs_Bowled','prev_fielding_heroics','prev_overs','bowling_innings','batting_innings'] which seemed to have no correlation with target and which had very high NaN values with respect to total data points from both train and test for Prediction data using
- df.drop(axis=1, inplace=True)
- Filled NaN values in 'prev_runs', 'prev_sixes', 'prev_fours', 'prev_balls' by their average values (using both train and test set) grouped by season using:

```
dft_concat = pd.concat([dft, dftt], ignore_index=True)
for column in ['prev_runs', 'prev_sixes', 'prev_fours','prev_balls']:
    avg_values = dft_concat.groupby('season')[column].transform('mean')
  dft[column] = dft[column].fillna(avg_values)  #also for test set
```

- Filled NaN values of prev_Dream Team with the mode of value for each player in combined train and test set. Before that I filled 0 in NaN places in the combined set to calculate average.

```
dft_concat['prev_Dream Team'] = dft_concat['prev_Dream Team'].fillna(0)
mode_dt_dft = dft_concat.groupby('Player_name')['prev_Dream Team'].apply(lambda x: x.mode().iloc[0])
dft['prev_Dream Team'] = dft['prev_Dream Team'].fillna(dft['Player_name'].map(mode_dt_dft)) #also for test
```

- Filled NaN values in prev_Total_FP with average of values for each unique player take from the combined dataset and put it as 0 where a player didn't have any history of prev_Total_FP.

```
fp_mean_player_dft = dft_concat.groupby('Player_name')['prev_Total_FP'].mean()
dft['prev_Total_FP'] = dft['prev_Total_FP'].fillna(dft['Player_name'].map(fp_mean_player_dft))
dft['prev_Total_FP'] = dft['prev_Total_FP'].fillna(0)   #also for test set
```

- Encoded Player names using Label Encoder (due to high number of unique values) for combined datastet. Dropped the initial column too.

```
combined_players = pd.concat([dft['Player_name'], dftt['Player_name']], axis=0)
label_encoder = LabelEncoder()
label_encoder.fit(combined_players)
dft['Player_name_en'] = label_encoder.transform(dft['Player_name'])  #same for test set
```

- One Hot Encoded 'venue','home_team','away_team' due to small number of unique values

```
dft = pd.get_dummies(dft, columns=['venue','home_team','away_team'], drop_first=True)  #also for test set
```

- Same methods were used for Classification set but some features were directly dropped as they didn't affect the target. The target was encoded using Label Encoding as well.
- Columns dropped from Classification dataset: 'match_id','venue','prev_runs', 'prev_balls', 'prev_sixes', 'prev_fours','luck','Starting_11','match_name','bowling_team_bowl', 'batting_team_bat', 'prev_wickets', 'prev_conceded', 'prev_catches', 'prev_Overs_Bowled','prev_fielding_heroics', 'prev_duck', 'prev_overs', 'batting_innings','bowling_innings','home_team','away_team'

Feature Engineering:

- Made two new features for Prediction dataset as: 'strike_rate' which is prev_runs per prev_balls and 'boundary_percentage' which is ((prev_sixes + prev_fours) per prev_runs)*100. Filled NaN values with 0 and erroneous high values with 1.

```
dft['strike_rate'] = dft['prev_runs'] / dft['prev_balls'] #also for test set
dft['strike_rate'] = dft['strike_rate'].fillna(0) #also for test set
dft['boundary_percentage'] = ((dft['prev_sixes'] + dft['prev_fours']) / dft['prev_runs']) * 100 #also for test set
dft['boundary_percentage'] = dft['boundary_percentage'].fillna(0) #also for test set
large_value_threshold = 1e10
dftt.replace([np.inf, -np.inf], 1, inplace=True)
dftt[dftt > large_value_threshold] = 1
```

Modelling:

- Split dataset into X (all features except target) and y (target).
- Split the train set into another train and test (80:20 resp.). Data points Shuffle parameter=42

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- **Prediction model:** Used **Random forest regressor** as this data can't be used to fit a curve due to its complex nature and no solid relation trends. Splitted Test set RMSE in range around 31-32 depending on preprocessing. Predicted given test set using the same.

```
rf_model = RandomForestRegressor(n_estimators=100, random_state=42) #initialize
rf_model.fit(X_train, y_train) #fit data
y_pred = rf_model.predict(X_test) #predict on test set (splitted from train set only)
rmse = np.sqrt(mean_squared_error(y_test, y_pred)) #rmse metric calculation
fp_pred = rf_model.predict(dftt) #prediction on actual test set
FP = pd.DataFrame({'Total_FP': fp_pred})
```

- **Classification model:** Used **XGBoost Classifier** due to complex relations and no direct relation of a player for being classified as a Captain, Vice Captain or player. Splitted Test set accuracy around 91-92 depending on preprocessing. Decoded Target for submission

```
xgb_clf = XGBClassifier() #initializing
xgb_clf.fit(X_train, y_train) #fit on train data
y_pred = xgb_clf.predict(X_test) #predict on test set (splitted from train set only)
accuracy = accuracy_score(y_test, y_pred) #accuracy metric
cvc_pred = xgb_clf.predict(dfct) #predictions of Captain/Vice Captain on actual test set
CVC = pd.DataFrame({'Captain/Vice Captain': cvc_pred})
original_labels = label_encoder.inverse_transform(cvc_pred) #decoding Captain/Vice Captain
CVC['Captain/Vice Captain'] = original_labels
```

- **File submission:** Joined match id from sample submission file with prediction and classification dataframes.

```
submission_df = pd.concat([sub['match_id'],FP, CVC], axis=1)
submission_df.to_csv('submission.csv', index=False) #convert submission df to csv
```

Discussion:

- I initially wanted to use Target Encoder for player names as it would have the relationship of a player with his Fantasy Points but couldn't do that due to 3 new players being in test set
- I tried different approaches for preprocessing and some of them (even after being wrong such as setting prev_Total_FP as 0 for NaN values) gave good result for test set but i chose to stick with the approaches which made sense to me.
- I was unable to relate data with a player being C, VC or NC. I tried predicting with and without certain columns but yielded the same result so decided to just go and drop the unrelated columns.
- I tried simple (Linear) to complex (Neural Networks) algorithms but the ones I used gave best results. SVR and SVC gave almost similar results.
- I used many visualisation techniques (they're not in submitted code though) like pair plots, heatmaps etc and couldn't find solid relations so decided to choose the algorithms that I finally used.
- I used Google and ChatGPT to understand IPL and FPL as I have no idea about cricket. I got help with some syntax from online resources.