Aalto University

CS-C3240 – Machine Learning

Maximizing UFC betting profits with regression models.

Contents

1. Introduction

Predicting results has been part of sporting events as long as they've existed. Often those predictions are based on human emotions rather than logical reasoning, which have led to the rise of a profitable multi-billion-dollar betting industry. Since its high variance characteristics, predicting outcomes has become huge part of Mixed Martial Arts, known as MMA. In this article we'll only make predictions of UFC fights, since the organizations data is most accurate and easily available.

1.1 Basics of betting

Sports betting industry uses both purely statistical methods on historic data and sport analysis to predict the result of an upcoming event. Implied probability (denoted IP) of the event is often described with decimal odds which is the inverse of the IP. Decimal odds make counting potential winnings over a bet simpler than it would be from a percentage probability. For example, if there is an upcoming MMA fight between fighter 1 and fighter 2 the odds for certain events can be listed as follows:

$$A = \text{"fighter 1 wins"} \ \ and \ \ P(A) = 60\ \% = 0.6$$

Event	A	A* = "fighter 2 wins"
Decimal odds	1/0.6 = 1.67	1/(1-0.6) = 2.5

with assumption that draw isn't possible outcome. That means, if you bet 10 \$ on event A and event A happens, you'll be paid 1.67*10 \$ = 16.7 \$ back. On the other hand, if A* happens, you'll lose your bet.

1.2 Basics of profitability

One way to analyze if a bet is profitable in the long run is to calculate its expected value (EV). Definition of EV is following:

$$EV = (Probability\ of\ winning * Potential\ payout)$$
$$-(Probability\ of\ losing * Amount\ Wagered)$$

A negative EV (denoted EV_-) means that the bet, even though possibly profitable one time, isn't profitable in the long run. For example, if we expect the $P_e(A) = 0.7$ then the EV with 10 \$ bet would be:

$$EV = P_e(A) * \left(\frac{1}{P(A)} * 10\ \$ - 10\ \$\right) - P_e(A^*) * 10\ \$ = -1{,}3\ \$ < 0$$

which means that our bet is EV_+ and therefore profitable in the long run.

Being profitable in sports betting is extremely rare, because you are constantly competing against big corporations full of data and sports analysts trying to win your money. Even if placing only EV_+ bets you'll probably run out of money before reaching the vast number of bets required to end up winning. But could machine learning models predict the outcomes well enough to make profitable betting easier to reach? To that question we will try to get answers by creating different regressions models and testing them against different betting strategies.

2. Problem formulation

To predict IP of the fight outcome, we will use feature vector

$$\boldsymbol{x} = \begin{pmatrix} \Delta Total\ rounds\ fought \\ \Delta Wins \\ \Delta Losses \\ \Delta Win\ streak \\ \Delta Ranking \end{pmatrix},$$

where Δ (denoted as ‘d’ in code and data) indicates difference between red and blue corner fighters.

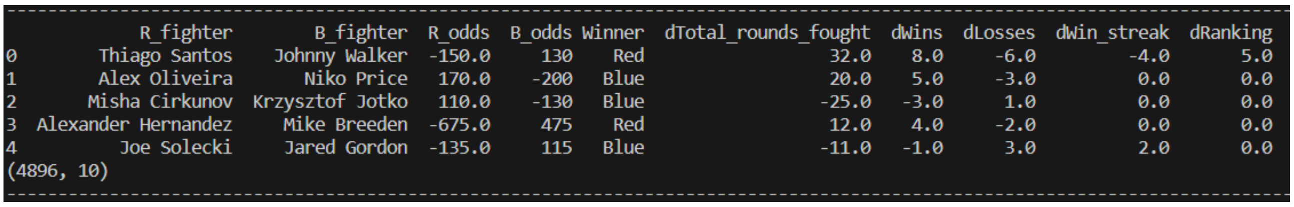
Our label in this project will be the odds for the red corner fighter to win.

Data used in the project is a dataset[1] acquired from Kaggle constructed from UFC’s official fight records[2] and betting odds[3].

3. Methods

3.1 Data

The dataset makes up a (4896, 119) sized matrix, where every row contains 119 datapoints from a single fight. From those columns we derive our feature columns by calculating all differences in favor of red corner (Exact formulas explained in code). Therefore, if a difference value is positive, it favors the red corner, and if the value is negative, it favors the blue corner. After calculating feature columns and filtering out the columns we don’t need anymore, we are left with a following (4896, 10) sized pandas data frame:



```
-------------------------------------------------------------------------------------------------------------------
           R_fighter         B_fighter  R_odds  B_odds Winner  dTotal_rounds_fought  dWins  dLosses  dWin_streak  dRanking
0      Thiago Santos     Johnny Walker  -150.0     130    Red                  32.0    8.0     -6.0         -4.0       5.0
1      Alex Oliveira        Niko Price   170.0    -200   Blue                  20.0    5.0     -3.0          0.0       0.0
2      Misha Cirkunov  Krzysztof Jotko   110.0    -130   Blue                 -25.0   -3.0      1.0          0.0       0.0
3  Alexander Hernandez    Mike Breeden  -675.0     475    Red                  12.0    4.0     -2.0          0.0       0.0
4         Joe Solecki     Jared Gordon  -135.0     115   Blue                 -11.0   -1.0      3.0          2.0       0.0
(4896, 10)
-------------------------------------------------------------------------------------------------------------------
```

3.2 Feature selection

Our feature selection is based on the premise that winning fighters tend to be more experienced and/or are in hype based in their recent performances. With columns ‘dTotal_rounds_fought’ and ‘dWins’ we try to measure the experience difference and with ‘dWin_streak’ the difference in recent performances. For example, in row one the red corner fighter Thiago Santos is way older than his opponent, so the experience statistics are on his side, but he has lost many of his recent matches so that should affect negatively to the IP for his win.

3.3 ML model

To predict red corner odds, we’ll create both polynomial regression model and linear regression model with regularization. Models fit well for our problem, because all our features are numerical values and label, we try to predict, is continuous numerical value. We chose polynomial regression model over regular linear one, because it is not clear whether the relationships between label and all the features is linear or to some higher degree. If all relationships between label and features turn out to be linear, linear regression model with regularization should give better predictions. Regularized model should work better than non-regularized one since the vast number of feature datapoints.

3.4 Loss function

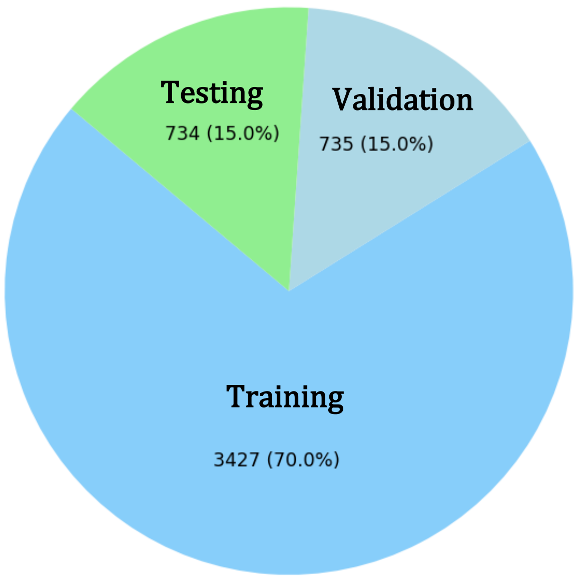
For polynomial regression model and linear regression model with regularization we'll use mean squared error (denoted MSE) as our loss function. MSE is defined as follows:

$$MSE = \frac{1}{n}\sum_{k=1}^{n}(y_k - \overline{y_k})^2 ,$$

where n is the number of predictions, y_k is the label and $\overline{y_k}$ is the corresponding prediction. MSE is generally recognized as the best loss function for regression models since it penalizes errors increasingly as the size of the error increases.

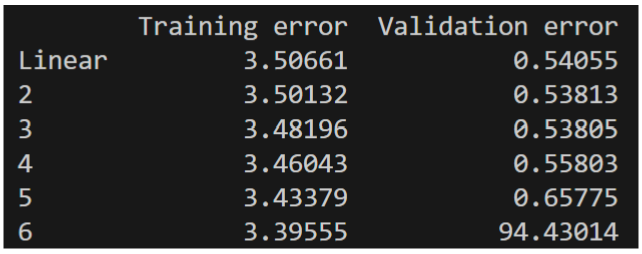
3.5 Training, validation and testing sets

Based on quick research from different sources it's common to allocate around 70-80 % of the data to training, 10-15 % into validation and the rest 10-15 % into testing. In this article we'll allocate 70 % of the data into training, 15 % into validation and 15 % into testing. This will be done by first shuffling our data, then splitting it into temporary and training sets and finally splitting the temporary set into testing and validation sets.



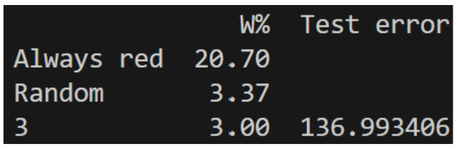
4. Results

After training and validating the models, the obtained errors were following:



```
        Training error  Validation error
Linear         3.50661           0.54055
2              3.50132           0.53813
3              3.48196           0.53805
4              3.46043           0.55803
5              3.43379           0.65775
6              3.39555          94.43014
```

Since the validation error is clearly smaller than the training error for all the models, it suggests that the models are overfitted. As we get the lowest validation error with 3rd degree model, we'll test the model by calculating the test error and betting performance for that model.



```
               W%  Test error
Always red  20.70
Random       3.37
3            3.00  136.993406
```

Meanings of the betting strategies:

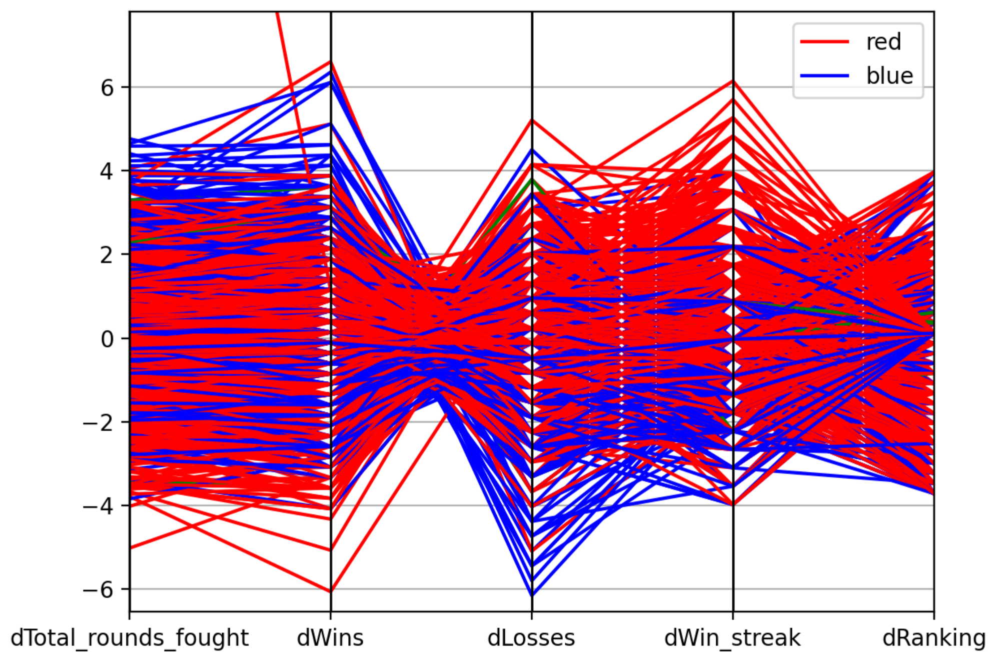
Always red	Always placing bet on red corner's win
Random	Placing bet on red corner's wins when the bet is EV_+. Uses the random probabilities generated to calculate EVs. Number shown is the average of all winning percentages after 10 runs.
3	Placing bet on red corner's wins when the bet is EV_+. Uses the predicted probabilities of the model to calculate EVs.

Test error of the 3rd degree model goes through the roof, which implies that our model is clearly overfitted. Also, the betting performance against two 'non-intelligent' strategies were worse.

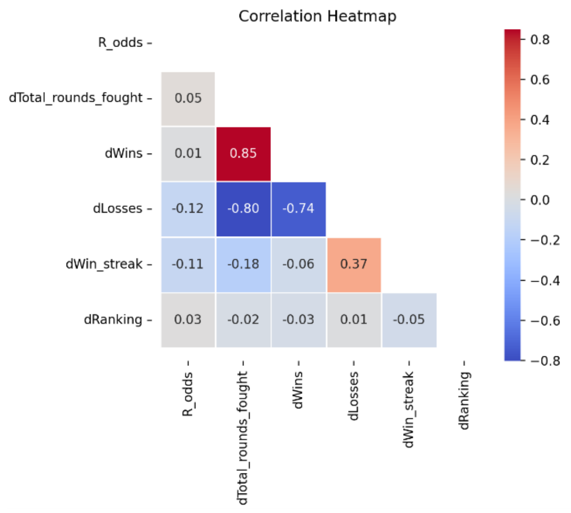
5. Conclusions

Test results show us that we couldn't create a satisfactorily performing model. As was known beforehand, predicting the IPs of fight outcomes is hard and our model couldn't do any better job than computer randomly guessing the IPs for red corners win.

If we visualize our data with parallel coordinates chart and pairwise correlation heatmap we can see that our data doesn't have any clear and learnable patterns for our model to pick up.



In the above chart the line color shows the favorite to win the match. Therefore, if the difference columns would correlate with favorite status, roughly all the blue lines should be on the negative side and all red lines on the positive side of the chart.



From the correlation heatmap we see that the only significant correlations can be found between the values that are strongly dependent on the number of matches fought. Clearly none of the features correlate with our label, which helps to explain why our model failed to predict the label based on the given data.

To make our model better performing, we should've searched for patterns and correlations in data beforehand and base our data selection on that information rather than a guess of good features.

6. References

[1]: https://www.kaggle.com/datasets/mdabbert/ultimate-ufc-dataset, 17.9.2023

[2]: http://ufcstats.com/statistics/events/completed, 17.9.2023

[3]: https://www.bestfightodds.com, 17.9.2023

7. Appendix

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.preprocessing import PolynomialFeatures, StandardScaler
from sklearn.metrics import mean_squared_error
from sklearn.utils import shuffle
import matplotlib.pyplot as plt
import seaborn as sns
import random

class UfcBettingPredictor:
  def __init__(self):
    self.dataframe = pd.read_csv('./data/ufcData.csv')
    self.errorsDf = pd.DataFrame()
    self.models = {}
    self.moneyWon = 0
    self.moneyBetted = 0

  def printDf(self):
    print('-----------------------------------'*4)
    print(self.dataframe.head())
    print(self.dataframe.shape)
    print('-----------------------------------'*4)

  def printHead(self, df):
    print('-----------------------------------'*3)
    print(df.head())
    print('-----------------------------------'*3)

  def filterDf(self, columns):
    self.dataframe = self.dataframe[columns]

  def fillNa(self, value):
    self.value = value
    self.dataframe = self.dataframe.fillna(self.value)

  def createDiffColumns(self):
    for index, row in self.dataframe.iterrows():
      # Calculation are written so that favorable outcome for red fighter
      # is positive and for blue fighter is negative

      # Assuming that having more rounds(=experience) is better
      self.dataframe.loc[
        index,
        'dTotal_rounds_fought'
      ] = row['R_total_rounds_fought'] - row['B_total_rounds_fought']

      # Assuming that having more wins is better
      self.dataframe.loc[
        index,
        'dWins'
      ] = row['R_wins'] - row['B_wins']

      # Assuming that having more losses is worse
      self.dataframe.loc[
        index,
        'dLosses'
      ] = row['B_losses'] - row['R_losses']

      # If both have win streak we calculate the difference between win
streaks
      R_win_streak = self.dataframe['R_current_win_streak']
      B_win_streak = self.dataframe['B_current_win_streak']

      if B_win_streak[index] and R_win_streak[index]:
        self.dataframe.loc[
          index,
          'dWin_streak'
        ] = row['R_current_win_streak'] - row['B_current_win_streak']

      # If red corner doesn't have a win streak we'll count it's
      # lose streak in favor of the blue corner
      elif B_win_streak[index] and not R_win_streak[index]:
        self.dataframe.loc[
          index,
          'dWin_streak'
        ] = -row['R_current_lose_streak'] - row['B_current_win_streak']

      # If blue corner doesn't have a win streak we'll count it's lose
streak
      # in favor of the red corner
      elif not B_win_streak[index] and R_win_streak[index]:
        self.dataframe.loc[
          index,
          'dWin_streak'
        ] = row['R_current_win_streak']+row['B_current_lose_streak']

      # If both have lose streaks we'll subtract the red corner's lose
streak
      # from the blue corner's lose streak
      # -> we end up >0 difference if red corner has shorter lose streak
      # and <0 if blue corner has shorter lose streak
      else:
        self.dataframe.loc[
          index,
          'dWin_streak'
        ] = row['B_current_lose_streak'] - row['R_current_lose_streak']

      # If both have ranking we calculate the difference between
      # rankings (0 is the best rank, 15 worst ->  diff maps from -15 to
15)
      R_weightclass_rank = self.dataframe['R_match_weightclass_rank']
      B_weightclass_rank = self.dataframe['B_match_weightclass_rank']

      if B_weightclass_rank[index] and R_weightclass_rank[index]:
        self.dataframe.loc[
          index,
          'dRanking'
        ] = row['B_match_weightclass_rank'] -
row['R_match_weightclass_rank']

      # For non ranked fighters we'll assume the difference to 0
      else:
        self.dataframe.loc[index, 'dRanking'] = 0

  def moneylineToDecimal(self, column):
    self.dataframe[column] = self.dataframe[column].apply(
      lambda x: x/100 + 1 if x > 0 else 100/abs(x) + 1
    )

  def calculateExpectedValue(self, predictedOdds, givenOdds, betSize):
    return predictedOdds/givenOdds*betSize-betSize - (1-
predictedOdds)*betSize

  def calculateResult(self, givenOdds, rowNumber, betSize):
    if self.dataframe['Winner'][rowNumber] == 'Red':
      self.moneyWon += betSize * (givenOdds - 1)
    else:
      self.moneyWon -= betSize

  def splitSets(self, features, labels, setSize, random_state=16):
    # First we shuffle the sets
    features_shuffled, labels_shuffled = shuffle(
      features,
      labels,
      random_state=random_state
    )

    # 1-setSize of the data is used for training, setSize for temporary
sets
    features_train,features_temp, labels_train, labels_temp =
train_test_split(
      features_shuffled,
      labels_shuffled,
      test_size=setSize,
      random_state=random_state
    )

    # Temporary sets are split in half for validation and test sets
    # both 15 % of initial data
    features_test, features_val, labels_test, labels_val =
train_test_split(
      features_temp,
      labels_temp,
      test_size=0.5,
      random_state=random_state
    )

    self.printHead(features_train)
    self.printHead(labels_train)
    self.printHead(features_val)
    self.printHead(labels_val)
    self.printHead(features_test)
    self.printHead(labels_test)

    return (features_train, labels_train),\
           (features_val, labels_val),\
           (features_test, labels_test)

  def drawPieChart(self, labels, sizes, title):
    # Colors for each slice
    colors = ['lightskyblue', 'lightblue', 'lightgreen']

    # Create a pie chart
    plt.figure(figsize=(6, 6))
    plt.pie(
      sizes,
      labels=labels,
      colors=colors,
      autopct=lambda p: f'{int(p * sum(sizes) / 100)} ({p:.1f}%)',
startangle=140
    )
    plt.axis('equal')  # Equal aspect ratio ensures that pie is drawn as
a circle.
    plt.title(title)
    # Display the pie chart
    plt.show()

  def drawParallelCoordinates(self, X):
    ss = StandardScaler()

    scaledDf = ss.fit_transform(X)
    scaledDf = pd.DataFrame(scaledDf, columns=X.columns)
```

```python
    scaledDf['categorized_odds'] = pd.cut(
      self.dataframe['R_odds'],
      bins=[1.00, 2.00, 2.01, 100],
      labels=['red', 'even', 'blue'],
      right=False
    )

    self.printHead(scaledDf)

    plot = pd.plotting.parallel_coordinates(
      scaledDf,
      'categorized_odds',
      color=('r', 'b' , 'c', 'g')
    )

    #hide 'nan' from legend
    handles, labels = plot.get_legend_handles_labels()
    plot.legend(handles[:2], labels[:2])
    plt.show()

  def drawPairWiseHeatmap(self):

    numberData = self.dataframe[[
      'R_odds',
      'dTotal_rounds_fought',
      'dWins',
      'dLosses',
      'dWin_streak',
      'dRanking'
    ]]
    numberDataCorr = numberData.corr()

    plt.figure(figsize=(8, 6))
    # Hides the upper triangle of the heatmap
    # lower triangle is enough since matrix is symmetric
    hideUpperTriangle = np.triu(numberDataCorr)
    sns.heatmap(
      numberDataCorr,
      annot=True,
      cmap="coolwarm",
      fmt=".2f",
      linewidths=0.5,
      annot_kws={"size": 10},
      mask=hideUpperTriangle
    )

    # Add labels and a title
    plt.title("Correlation Heatmap")
    plt.xlabel("Variables")
    plt.ylabel("Variables")

    # Show the heatmap
    plt.show()

  def polynomialModel(self, X_train, y_train, X_val, y_val, degreeList):

    for degree in degreeList:
      poly = PolynomialFeatures(degree=degree)
      X_train_poly = poly.fit_transform(X_train)

      model = LinearRegression(fit_intercept=False)
      model.fit(X_train_poly, y_train)

      y_train_pred = model.predict(X_train_poly)
      tr_error = mean_squared_error(y_train, y_train_pred)
      self.errorsDf.loc[degree, 'Training error'] = round(tr_error, 5)

      X_val_poly = poly.transform(X_val)
      y_val_pred = model.predict(X_val_poly)
      val_error = mean_squared_error(y_val, y_val_pred)
      self.errorsDf.loc[degree, 'Validation error'] = round(val_error, 5)

      self.models[degree] = model

    # Checking degree corresponding the smallest validation error
    self.minErrorDegree = self.errorsDf['Validation error'].idxmin()

  def linearModel(self, X_train, y_train, X_val, y_val):
    model = Ridge(alpha=0.5)
    model.fit(X_train, y_train)

    y_train_pred = model.predict(X_train)
    tr_error = mean_squared_error(y_train, y_train_pred)
    self.errorsDf.loc['Linear', 'Training error'] = round(tr_error, 5)

    y_val_pred = model.predict(X_val)
    val_error = mean_squared_error(y_val, y_val_pred)
    self.errorsDf.loc['Linear', 'Validation error'] = round(val_error, 5)

    self.models['Linear'] = model


  def testBetting(self, X_test, y_test):

    self.moneyWon = 0
    betSize = 10
    results = pd.DataFrame()

    for row in self.dataframe.iterrows():
      self.calculateResult(row[1]['R_odds'], row[0], betSize)
    results.loc[
      'Always red',
      'W%'
    ] = 100*round(self.moneyWon/(len(y_test)*10), 3)
    self.moneyWon = 0
    self.randomList = []
    for i in range(10):
      for row in self.dataframe.iterrows():
        if self.calculateExpectedValue(
          random.randrange(1, 10),
          row[1]['R_odds'], betSize
        ) > 0:
          self.moneyBetted += betSize
          self.calculateResult(row[1]['R_odds'], row[0], betSize)
      self.randomList.append(100*round(self.moneyWon/self.moneyBetted,
3))
    results.loc[
      'Random',
      'W%'
    ] = round(sum(self.randomList)/len(self.randomList), 3)

    self.moneyWon = 0
    self.moneyBetted = 0

    model = self.models[self.minErrorDegree]

    poly = PolynomialFeatures(degree=self.minErrorDegree)
    X_test_poly = poly.fit_transform(X_test)
    y_test_pred = model.predict(X_test_poly)

    testError = mean_squared_error(y_test, y_test_pred)
    results.loc[self.minErrorDegree, 'Test error'] = testError

    for predictedOdds,\
        givenOdds,\
        rowNumber in zip(y_test_pred, y_test, y_test.index):
      if self.calculateExpectedValue(predictedOdds, givenOdds, betSize) >
0:
        self.moneyBetted += betSize
        self.calculateResult(givenOdds, rowNumber, betSize)
    results.loc[
      self.minErrorDegree,
      'W%'
    ] = 100*round(self.moneyWon/self.moneyBetted, 3)
    results.fillna('', inplace=True)
    print('-----------------------------------'*4)
    print(results)


  def main(self):
    self.printDf()

    self.filterDf([
      'R_fighter',
      'B_fighter',
      'R_odds',
      'B_odds',
      'Winner',
      'B_current_lose_streak',
      'B_current_win_streak',
      'B_losses',
      'B_total_rounds_fought',
      'B_wins',
      'R_current_lose_streak',
      'R_current_win_streak',
      'R_losses',
      'R_total_rounds_fought',
      'R_wins',
      'B_match_weightclass_rank',
      'R_match_weightclass_rank',
    ])
    self.printDf()

    self.fillNa(-1)

    self.moneylineToDecimal('R_odds')
    self.moneylineToDecimal('B_odds')
    self.createDiffColumns()
    self.printDf()

    self.filterDf([
      'R_fighter',
      'B_fighter',
      'R_odds',
      'B_odds',
      'Winner',
      'dTotal_rounds_fought',
      'dWins',
      'dLosses',
      'dWin_streak',
      'dRanking'
    ])
    self.printDf()

    features = self.dataframe[[
      'dTotal_rounds_fought',
      'dWins',
      'dLosses',
      'dWin_streak',
      'dRanking'
    ]]
    labels = self.dataframe['R_odds']

    datasetArray = self.splitSets(features, labels, 0.3, 16)

    self.drawPieChart(
      ['Training', 'Validation', 'Testing'],
      [len(datasetArray[0][0]),
       len(datasetArray[1][0]),
       len(datasetArray[2][0])],
       'Data points count in each set'
    )

    degrees = [2, 3, 4, 5, 6]

    self.drawParallelCoordinates(features)
    self.drawPairWiseHeatmap()

    self.linearModel(
      datasetArray[0][0],
      datasetArray[0][1],
      datasetArray[1][0],
      datasetArray[1][1]
    )

    self.polynomialModel(
      datasetArray[0][0],
      datasetArray[0][1],
      datasetArray[1][0],
      datasetArray[1][1],
      degrees
    )

    print(self.errorsDf)

    self.testBetting(
      datasetArray[2][0],
      datasetArray[2][1]
    )

predictor = UfcBettingPredictor()
predictor.main()
```