

# ISYE 7406: HW 5

Matt McDowell

March 2025

## 1 Abstract

This paper assess the effectiveness of two types different ensemble methods: *random forests* and *boosting* to create a classifier model to predict the results of professional tennis matches. These methods were compared against baseline models such as logistic regression and K-nearest-neighbors. Cross validation was used in order to tune the parameters of these models as well as evaluate the testing error and variance of their predictions. The findings demonstrate that the XGBoost model produced the best classifier and outperformed the other models. The results highlight the importance of using robust cross validation techniques to assess model performance and tune model parameters, additionally they highlight the learning capability of ensemble methods.

## 2 Introduction

Every year there are thousands of professional tennis matches played on almost every continent in the the world, generating large amounts of data on player performance. Famously, IBM was an early pioneer in using machine learning techniques as well as leveraging Watson to generate insights and make match predictions. However, compared to the early years there are now more advanced methods available such as *random forests* and *boosting* which leverage the combining many "weak" learners to create a single strong classifier. This paper explores the effectiveness of these models more traditional against baseline models to predict tennis match outcomes.

For those interested, but not familiar with tennis here is a general guide to the rules : [Tennis Basics](#).

## 3 Problem Statement/Exploratory Analysis

### 3.1 Problem Statement

The dataset used in this analysis has match ATP (men's professional tennis) data from 1968 which is considered the beginning of the "Open Era" in men's professional tennis [1].

However, this dataset only has complete in-game match statistics after 1991 therefore data from after 1991 was exclusively used. The goal of this analysis is develop a prediction/classification model that can accurately determine whether a player will win or lose a match based on their performance. This will allow coaches and players to make strategic/tactical decisions to optimize their chances at winning the match.

### 3.2 Exploratory Analysis

In total, this dataset contains 26 features from over 183,000 professional matches,. Of these 26 features only 8 pertained to in-game match performance, these along with player height, player age, and court surface were used to train the classifiers. Additionally, I created a binary variable (win) that indicates whether the player won or lost the match. In this analysis the win variable will serve as the response and the rest of the data will be used as predictors.

For this analysis I selected 10,000 samples at random and created the following histograms (Figure 1) to show the distributions of the features. A breakdown of the meaning of each of these features is as follows:

1. ht - player height (cm)
2. age - player age (yrs)
3. surface - surface type (hard court, clay, grass, carpet)
4. ace - aces hit
5. df - double faults
6. svpt - overall serve percent
7. 1stIn - 1st serve in percent
8. 1stWon - 1st serve win percent
9. 2ndWon - 2nd serve win percent
10. bpSaved - break points saved
11. bpFaced - break points faced
12. win - result of match (1-win, 0-lose)

Additionally to determine the predictive potential of this dataset I created a correlation matrix (Figure 2). Based on the correlation matrix, there are many features with "weak" correlations ( $< 0.5$ ) but there isn't a single strong predictor. Therefore this dataset appears to be a good candidate for ensemble models specifically **boosting** where multiple weak predictors are combined to make a single strong classifier.

## Distribution of Features by Surface Type

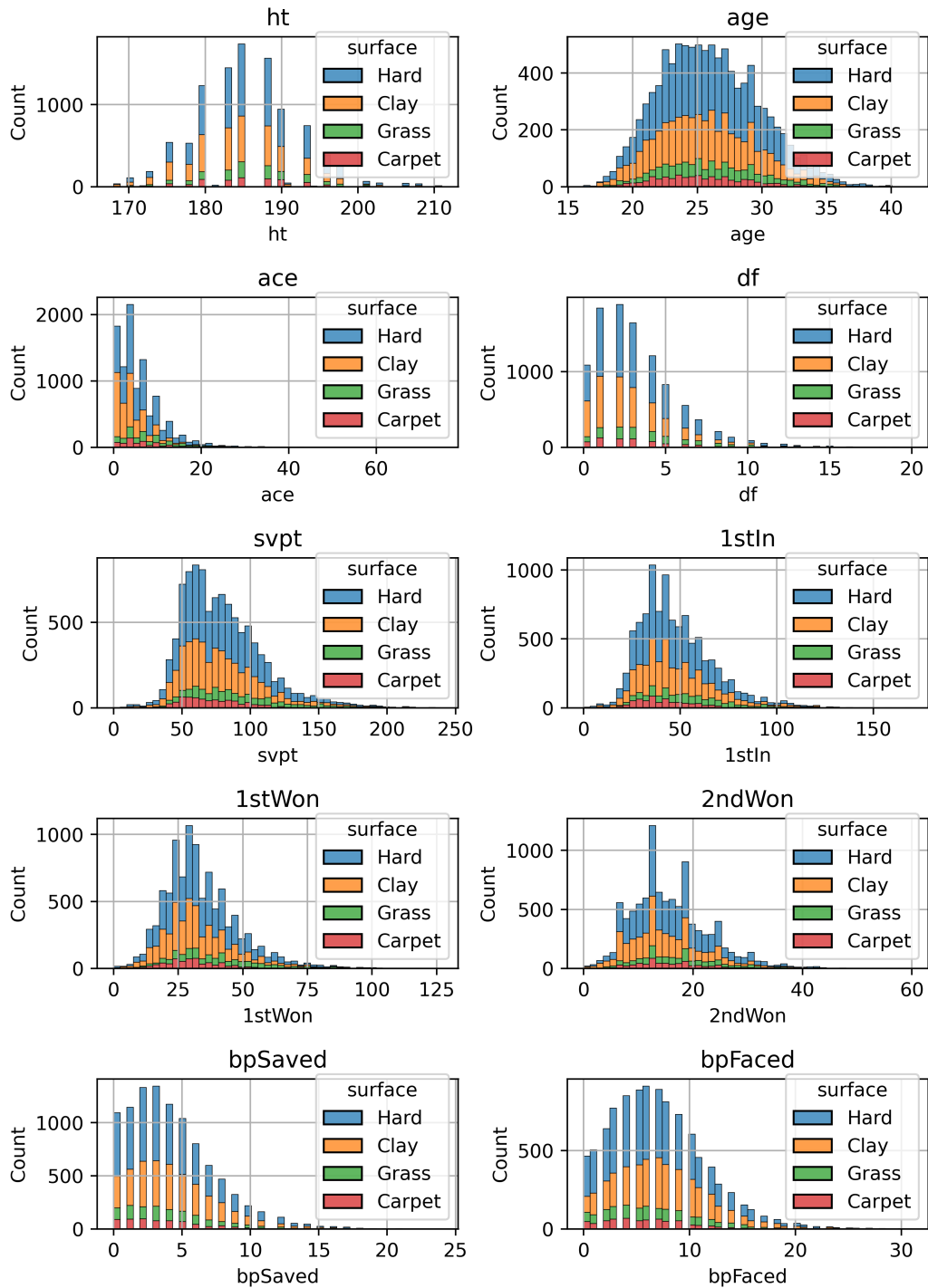


Figure 1: Distribution of Features by Surface Type

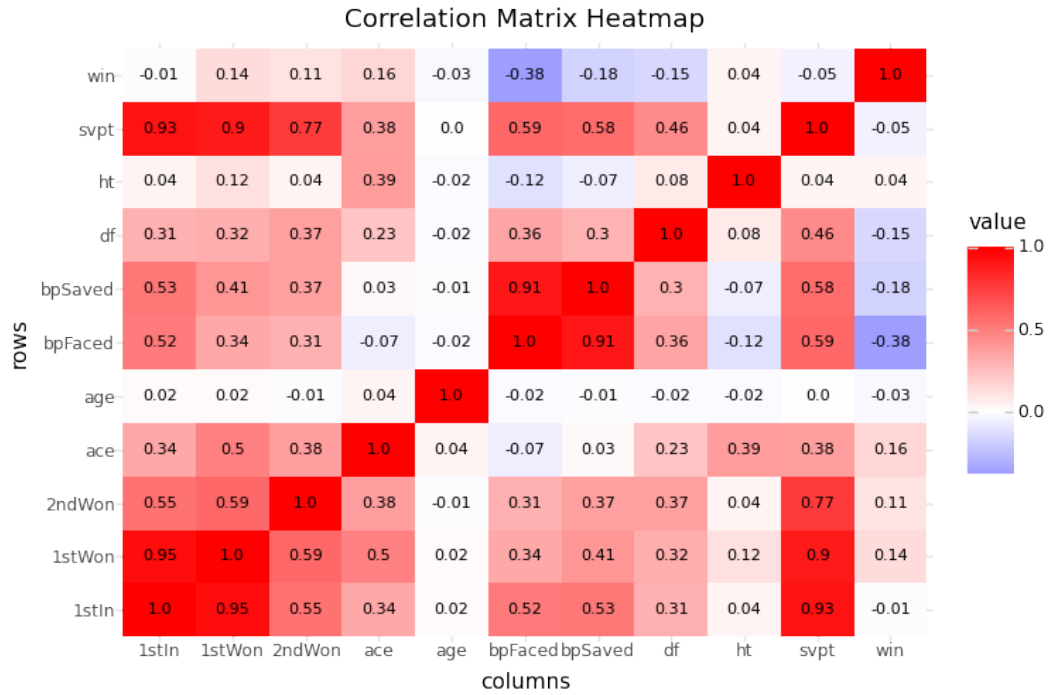


Figure 2: Correlation Matrix Heatmap

## 4 Methodology

In this section I will describe the different methods used to create/generate the ensemble classification models as well the baseline classification models. Additionally, I will briefly summarize their initial performance on the testing data and where applicable I will optimize model parameters using the testing error. Where applicable, when a decision boundary for predicting a class was needed the threshold value of 0.5 was chosen to give an equal weight to misclassifications of both wins and losses.

For this section the data was split (90/10) for training/testing and the same training and testing data was used for each model. When model parameters where optimized k-fold (k=10) cross validation was used.

### 4.1 Baseline Models

The baseline models that were chosen to test against the ensemble models are *Logistic Regression* and *K-Nearest-Neighbors*. These models were chosen because of their simplicity, interpretability, and widespread use in classification problems.

#### 4.1.1 Logistic Regression

Logistic Regression is a linear model that estimates the probability of a given class using the logistic function.

Logit Regression Results						
Dep. Variable:	win	No. Observations:	9000			
Model:	Logit	Df Residuals:	8988			
Method:	MLE	Df Model:	11			
Date:	Thu, 13 Mar 2025	Pseudo R-squ.:	0.3456			
Time:	17:43:24	Log-Likelihood:	-4082.2			
converged:	True	LL-Null:	-6238.2			
Covariance Type:	nonrobust	LLR p-value:	0.000			
	coef	std err	z	P> z	[0.025	0.975]
const	-0.0717	0.028	-2.580	0.010	-0.126	-0.017
ht	-0.2082	0.030	-6.854	0.000	-0.268	-0.149
age	-0.1853	0.028	-6.661	0.000	-0.240	-0.131
ace	-0.3076	0.039	-7.817	0.000	-0.385	-0.230
df	-0.0243	0.036	-0.676	0.499	-0.095	0.046
svpt	-3.2817	0.268	-12.260	0.000	-3.806	-2.757
1stIn	-0.1982	0.166	-1.193	0.233	-0.524	0.127
1stWon	2.9752	0.181	16.462	0.000	2.621	3.329
2ndWon	1.4224	0.096	14.856	0.000	1.235	1.610
bpSaved	1.7917	0.103	17.404	0.000	1.590	1.994
bpFaced	-2.2650	0.145	-15.580	0.000	-2.550	-1.980
surface_num	-0.1126	0.028	-4.049	0.000	-0.167	-0.058

Figure 3: Logistic Regression Model

$$\log\left(\frac{P}{1-P}\right) = B_0 + B_1x_{i,1} \dots B_{p-1}x_{i,p-1} \quad (1)$$

The basic assumption of the Logistic Regression classification model is that it models conditional probability of the response directly and makes no assumptions about the distribution of predictors. For this baseline model all predictor variables were used, the initial training error and testing errors of this model were 0.201 and 0.206 respectively. See Figure 3 for a summary of the model.

#### 4.1.2 K-Nearest-Neighbors

K-Nearest-Neighbors (KNN) is a classification model that uses distance metrics to assign data to a class based on its "k" nearest neighbors. It is another commonly used model since it is another non-parametric model, meaning that the model isn't making assumptions about the underlying data distributions. Since there was a relatively large amount of data I decided to use cross validation to optimize the k parameter by testing k-values from 1 to 100. After tuning this parameter the optimal model was KNN = 61, see Figure 4. This model's training and testing errors were 0.259 and 0.277 respectively.

## 4.2 Ensemble Models

The ensemble models that were chosen for this analysis are random forests and XGBoost. Both of these models improve classification performance by combining multiple weak learners. However, these models differ in their learning strategies: Random Forest uses on bagging, while XGBoost uses boosting.

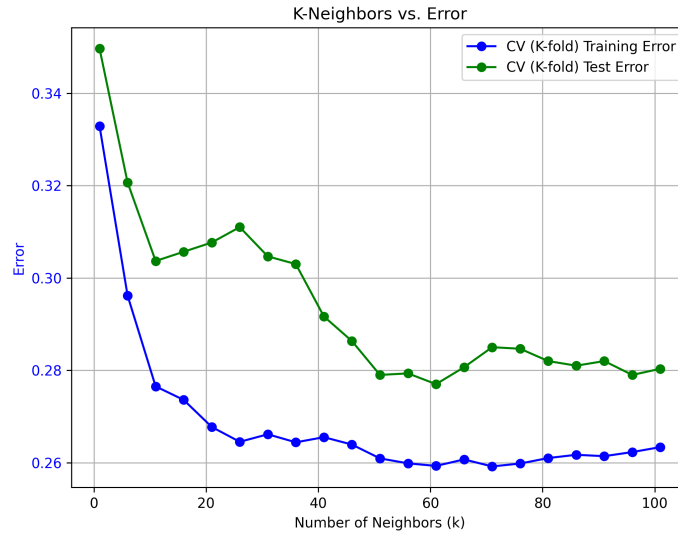


Figure 4: KNN Model

#### 4.2.1 Random Forest

Random Forest is an ensemble learning method that uses decision trees with a random subset of features and bagging to decrease the variance in classification. The key idea in bagging is to use bootstrapped (randomly sampled w/replacement) data to make each tree and use the majority vote for classification. The random subset of features introduces diversity among the trees, leading to more robust and generalizable predictions. For this analysis I used cross validation to optimize the number of estimators, see [Figure 5](#). The optimal value for number number of estimators was found to be 81 and this model's training and testing errors were 0.292 and 0.281 respectively.

#### 4.2.2 Boosting: XGBoost

XGBoost (Extreme Gradient Boosting) is a boosting ensemble method that builds decision trees sequentially, with each tree built correcting misclassifications from the previous tree. Unlike Random Forest, which grows trees independently XGBoost assigns greater weight to "weak" learners and the weighted average of the trees is used to improve the training error. For this analysis I used the number of estimators found for the random forest and used cross validation to optimize the learning rate, see [Figure 6](#). The optimal value for the learning rate was found to be 0.21 and this model's training and testing errors were 0.212 and 0.248 respectively.

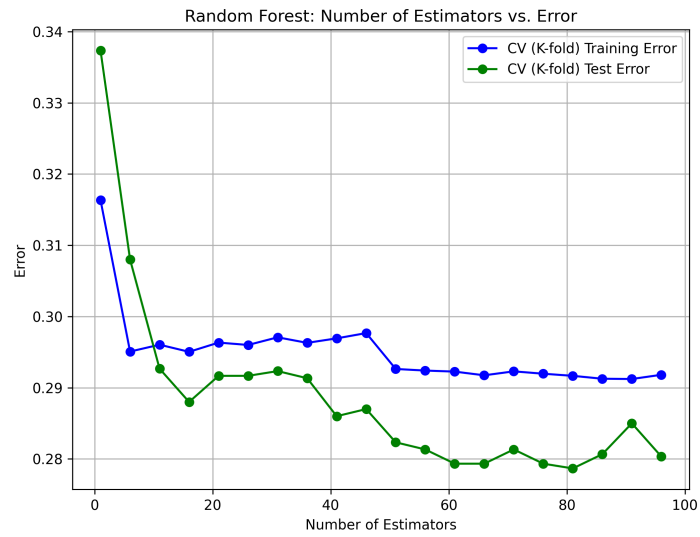


Figure 5: Random Forest Model

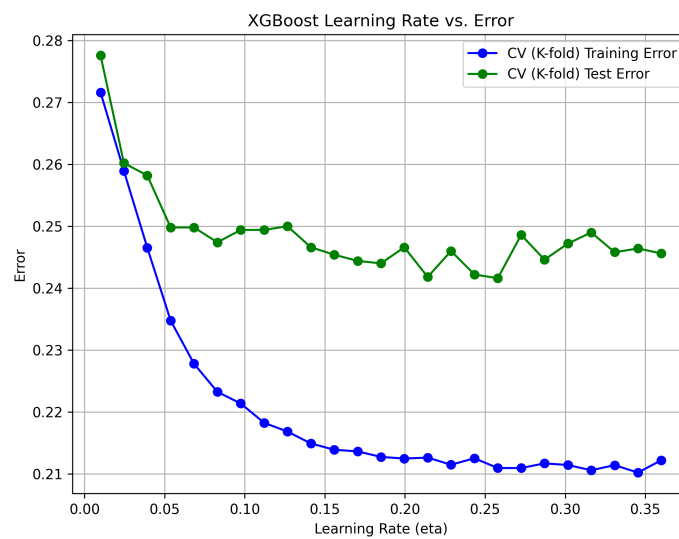


Figure 6: XGBoost Model

Model	Mean Test Error	Mean Test Variance
Logistic Regression	0.208	1.79E-4
KNN	0.250	1.12E-4
Random Forest	0.287	1.69E-4
XGBoost	0.187	1.60E-4

Table 1: Classification Model Comparison

## 5 Results/Analysis

This section contains a brief comparative analysis of the different models and their performance on dataset.

Initially, the classification models were evaluated on one split of of the data while the model parameters were selected using cross validation. In order to provide a more thorough comparison 100 Monte Carlo cross validation simulations were used to evaluate the testing errors and variances of each model, see [Table 1](#). One of the most significant findings of this analysis is how much XGBoost improved from it original performance on the testing data set. It improved by from its performance on one split of the data by 22% to become the best model. This can be attributed to the fact that allowing XGBoost to see many partitions of the data it was better able to improve upon its weak learners effectively. In contrast, **Random Forest** showed much weaker performance, and its its improvement was much more modest compared to XGBoost.

## 6 Conclusions

In summary, out of the different models the XGBoost classification model was able to significantly outperform Random Forests as well as baseline models such as Logistic Regression and KNN. The improvement of the XGBoost model after 100 Monte Carlo simulations highlights the strengths of boosting techniques to improve the classification model.

However, while the final model is strong, it is still wrong approximately 20% of the time. This shows the inherent challenges with working with the complexity of professional tennis match data and fully capturing the underlying patterns.

### 6.1 Lessons Learned

An important lesson learned is that while boosting techniques like XGBoost can significantly enhance predictive performance, there is still room for improvement. Future work could be done to further fine-tune parameters or experimenting with alternative ensemble methods to reduce the error rate. Additionally, considering the cost of misclassifications in real-world applications, techniques such as adjusting the decision threshold or using



cost-sensitive learning could be valuable in refining model performance. Finally, these results show the importance of using cross-validation techniques for both model selection and performance evaluation.

## 7 References

- [1] Sijo Manikandan. *ATP matches*. URL: <https://www.kaggle.com/datasets/sijovm/atpdata/data>.

## A Appendix A: Python Code

```
1 HW5 Ensemble Methods
2 -----
3 .. code:: ipython3
4
5     import pandas as pd
6     from datetime import datetime, timedelta
7     import numpy as np
8     import matplotlib.pyplot as plt
9     import seaborn as sns
10    from sklearn.preprocessing import LabelEncoder, StandardScaler,
11    MinMaxScaler, RobustScaler, Normalizer
12    from sklearn import model_selection
13    from sklearn.model_selection import train_test_split, RepeatedKFold,
14    cross_val_score
15    from plotnine import ggplot, aes, geom_tile, scale_fill_gradient,
16    theme_minimal, theme, geom_boxplot
17    from plotnine import ggsave, scale_y_reverse, labs,
18    scale_fill_gradient2, geom_line, geom_point, scale_color_manual,
19    geom_text
20    import statsmodels.api as sm
21    from sklearn.metrics import confusion_matrix,
22    ConfusionMatrixDisplay, accuracy_score, precision_score, roc_curve,
23    auc
24    from sklearn.neighbors import NearestNeighbors,
25    KNeighborsClassifier
26    from sklearn.ensemble import GradientBoostingClassifier,
27    RandomForestClassifier
28    import xgboost as xgb
29    from tqdm import tqdm
30
31 .. code:: ipython3
32
33     data = pd.read_csv("atp_matches_till_2022.csv")
34
35 .. code:: ipython3
36
37     data['tourney_yr'] = data['tourney_id'].apply(lambda x: x.split('-')[0]).astype(int)
38     data = data.loc[data['tourney_yr'] >= 1991]
39
40 .. code:: ipython3
41
42     data_win = data[['winner_id', 'winner_name', 'surface', 'winner_ht', 'winner_age', 'w_ace', 'w_df', 'w_svpt', 'w_1stIn', 'w_1stWon', 'w_2ndWon', 'w_bpSaved', 'w_bpFaced']].dropna()
```

```

34     data_lose = data[['loser_id', 'loser_name', 'surface', 'loser_ht', '
loser_age', 'l_ace', 'l_df', 'l_svpt', 'l_1stIn', 'l_1stWon', 'l_2ndWon
', 'l_bpSaved', 'l_bpFaced']].dropna()
35     data_win.columns = [col.split('_', 1)[1] if '_' in col else col
for col in data_win.columns]
36     data_lose.columns = [col.split('_', 1)[1] if '_' in col else col
for col in data_lose.columns]
37     data_win['win'] = 1
38     data_lose['win'] = 0
39
40 .. code:: ipython3
41
42     dataset = pd.concat([data_win, data_lose], axis=0)
43     dataset = dataset.sample(n=10000, random_state=42).reset_index(
drop=True)
44
45 .. code:: ipython3
46
47     dataset.describe()
48
49
50
51
52 .. raw:: html
53
54     <div>
55     <style scoped>
56         .dataframe tbody tr th:only-of-type {
57             vertical-align: middle;
58         }
59
60         .dataframe tbody tr th {
61             vertical-align: top;
62         }
63
64         .dataframe thead th {
65             text-align: right;
66         }
67     </style>
68     <table border="1" class="dataframe">
69         <thead>
70         <tr style="text-align: right;">
71             <th></th>
72             <th>id</th>
73             <th>ht</th>
74             <th>age</th>
75             <th>ace</th>
76             <th>df</th>
77             <th>svpt</th>

```

```

78         <th>1stIn</th>
79         <th>1stWon</th>
80         <th>2ndWon</th>
81         <th>bpSaved</th>
82         <th>bpFaced</th>
83         <th>win</th>
84     </tr>
85 </thead>
86 <tbody>
87     <tr>
88         <th>count</th>
89         <td>10000.000000</td>
90         <td>10000.000000</td>
91         <td>10000.000000</td>
92         <td>10000.000000</td>
93         <td>10000.000000</td>
94         <td>10000.000000</td>
95         <td>10000.000000</td>
96         <td>10000.000000</td>
97         <td>10000.000000</td>
98         <td>10000.000000</td>
99         <td>10000.000000</td>
100        <td>10000.000000</td>
101    </tr>
102    <tr>
103        <th>mean</th>
104        <td>106015.860700</td>
105        <td>185.425900</td>
106        <td>25.967420</td>
107        <td>5.591200</td>
108        <td>3.120200</td>
109        <td>79.364800</td>
110        <td>47.773800</td>
111        <td>33.860100</td>
112        <td>15.775600</td>
113        <td>4.147000</td>
114        <td>6.944000</td>
115        <td>0.504400</td>
116    </tr>
117    <tr>
118        <th>std</th>
119        <td>14172.725522</td>
120        <td>6.581918</td>
121        <td>3.818011</td>
122        <td>5.026869</td>
123        <td>2.534402</td>
124        <td>29.402047</td>
125        <td>19.260268</td>
126        <td>14.237858</td>

```

```

127         <td>7.057792</td>
128         <td>3.248484</td>
129         <td>4.443674</td>
130         <td>0.500006</td>
131     </tr>
132     <tr>
133         <th>min</th>
134         <td>100284.000000</td>
135         <td>168.000000</td>
136         <td>16.200000</td>
137         <td>0.000000</td>
138         <td>0.000000</td>
139         <td>0.000000</td>
140         <td>0.000000</td>
141         <td>0.000000</td>
142         <td>0.000000</td>
143         <td>0.000000</td>
144         <td>0.000000</td>
145         <td>0.000000</td>
146     </tr>
147     <tr>
148         <th>25%</th>
149         <td>102110.000000</td>
150         <td>180.000000</td>
151         <td>23.200000</td>
152         <td>2.000000</td>
153         <td>1.000000</td>
154         <td>58.000000</td>
155         <td>34.000000</td>
156         <td>24.000000</td>
157         <td>11.000000</td>
158         <td>2.000000</td>
159         <td>4.000000</td>
160         <td>0.000000</td>
161     </tr>
162     <tr>
163         <th>50%</th>
164         <td>103507.000000</td>
165         <td>185.000000</td>
166         <td>25.700000</td>
167         <td>4.000000</td>
168         <td>3.000000</td>
169         <td>74.000000</td>
170         <td>44.000000</td>
171         <td>31.000000</td>
172         <td>15.000000</td>
173         <td>4.000000</td>
174         <td>6.000000</td>
175         <td>1.000000</td>

```

```

176         </tr>
177         <tr>
178             <th>75%</th>
179             <td>104745.000000</td>
180             <td>190.000000</td>
181             <td>28.600000</td>
182             <td>8.000000</td>
183             <td>4.000000</td>
184             <td>95.000000</td>
185             <td>58.000000</td>
186             <td>41.000000</td>
187             <td>20.000000</td>
188             <td>6.000000</td>
189             <td>10.000000</td>
190             <td>1.000000</td>
191         </tr>
192         <tr>
193             <th>max</th>
194             <td>210013.000000</td>
195             <td>211.000000</td>
196             <td>41.600000</td>
197             <td>75.000000</td>
198             <td>20.000000</td>
199             <td>240.000000</td>
200             <td>170.000000</td>
201             <td>127.000000</td>
202             <td>60.000000</td>
203             <td>24.000000</td>
204             <td>31.000000</td>
205             <td>1.000000</td>
206         </tr>
207     </tbody>
208 </table>
209 </div>
210
211
212
213 .. code:: ipython3
214
215     dataset.info()
216
217
218 .. parsed-literal::
219
220     <class 'pandas.core.frame.DataFrame'>
221     RangeIndex: 10000 entries, 0 to 9999
222     Data columns (total 14 columns):
223     #   Column      Non-Null Count  Dtype
224     ---  ---

```

```

225     0    id      10000 non-null  int64
226     1   name      10000 non-null  object
227     2  surface  10000 non-null  object
228     3    ht      10000 non-null  float64
229     4   age      10000 non-null  float64
230     5    ace      10000 non-null  float64
231     6    df      10000 non-null  float64
232     7   svpt      10000 non-null  float64
233     8  1stIn      10000 non-null  float64
234     9  1stWon      10000 non-null  float64
235    10  2ndWon      10000 non-null  float64
236    11 bpSaved      10000 non-null  float64
237    12 bpFaced      10000 non-null  float64
238    13   win      10000 non-null  int64
239  dtypes: float64(10), int64(2), object(2)
240  memory usage: 1.1+ MB
241
242
243  .. code:: ipython3
244
245
246      numeric_cols = dataset.drop(columns=['id', 'name', 'win', 'surface
247      ']).columns
248
249      # Create a 2x5 grid
250      fig, axes = plt.subplots(nrows=5, ncols=2, figsize=(7, 10))
251      axes = axes.flatten() # Flatten the 2D array for easy iteration
252
253      # Plot each histogram
254      for i, col in enumerate(numeric_cols):
255          sns.histplot(data=dataset, x=col, bins=50, hue="surface",
256          hue_order = ['Hard', 'Clay', 'Grass', 'Carpet'], ax=axes[i], multiple=
257          "stack")
258
259          axes[i].set_title(col)
260          axes[i].grid(True)
261
262
263      fig.suptitle("Distribution of Features by Surface Type", fontsize
264      =16, fontweight='bold')
265      plt.tight_layout()
266
267
268      plt.savefig("Distribution of Features by Surface Type.png", format
269      ="png", dpi=500)
270      plt.show()
271
272  .. image:: output_7_0.png

```

```

269
270
271 .. code:: ipython3
272
273     le = LabelEncoder()
274     dataset['surface_num'] = le.fit_transform(dataset['surface'])
275
276 .. code:: ipython3
277
278     corr = dataset.drop(columns=['id', 'name', 'surface']).corr().
round(2).reset_index().melt(id_vars="index", var_name="columns",
value_name="value")
279
280     corr = corr.rename(columns={"index": "rows"})
281
282     plot = (ggplot(corr, aes(x='columns', y='rows', fill='value'))
+ geom_tile()
283     + scale_fill_gradient2(low="blue", high="red", mid="white"
, midpoint=0)
284     + geom_text(aes(label='value'), color='black', size=8)
285     + theme_minimal()
286     + labs(title="Correlation Matrix Heatmap")
287     )
288
289
290     #plot.tight_layout()
291     plot.show()
292     ggsave(plot, filename='correlation_matrix.png', width=7.5, height
=5)
293
294
295
296 .. image:: output_9_0.png
297     :width: 640px
298     :height: 480px
299
300
301 .. parsed-literal::
302
303     C:\Users\mmcd\miniconda3\Lib\site-packages\plotnine\ggplot.py
:615: PlotnineWarning: Saving 7.5 x 5 in image.
304     C:\Users\mmcd\miniconda3\Lib\site-packages\plotnine\ggplot.py
:616: PlotnineWarning: Filename: correlation_matrix.png
305
306
307 .. code:: ipython3
308
309     data = dataset.drop(columns=['id', 'name', 'surface'])
310     response = data[['win']]
311     features = data.drop(columns=['win'])

```



```

312     train_size = round(len(data)*0.9)
313     test_size = round(len(data)*0.1)
314
315     train_features, test_features, train_response, test_response =
train_test_split(
316         features, response, train_size = train_size , test_size =
test_size, random_state=123)
317
318
319     train_data = pd.concat([train_response, train_features], axis=1)
320     test_data = pd.concat([test_response, test_features], axis=1)
321
322 .. code:: ipython3
323
324     def standardize_data(features,response):
325
326
327         #scale dataset
328         feautre_scaler = StandardScaler()
329         response_scaler = StandardScaler()
330         feautre_scaler.fit(features)
331         response_scaler.fit(response)
332
333         features_scaled = feautre_scaler.transform(features) #scaled
Features
334         response_scaled = response_scaler.transform(response) # scaled
Response
335
336
337
338         features_scaled = pd.DataFrame(features_scaled, columns=features
.columns)
339         response_scaled = pd.DataFrame(response_scaled, columns=response
.columns)
340         return features_scaled,response_scaled,feautre_scaler,
response_scaler
341
342 .. code:: ipython3
343
344     X_train = train_features
345     Y_train = train_response
346     X_test = test_features
347     Y_test = test_response
348
349
350     X_train_stand, Y_train_stand,X_train_stand_scaler,
Y_train_stand_scaler = standardize_data(features=X_train, response
=Y_train)

```

```

351 X_test_stand, Y_test_stand, X_test_stand_scaler,
Y_test_stand_scaler = standardize_data(features=X_test, response=
Y_test)

352
353 X_train_stand = sm.add_constant(X_train_stand)
354 X_test_stand = sm.add_constant(X_test_stand)
355
356
357 logit = sm.Logit(Y_train.reset_index(drop=True), X_train_stand)
358 logit = logit.fit()
359
360 # Print summary
361 print(logit.summary())
362
363 train_predictions = logit.predict(X_train_stand)
364 test_predictions = logit.predict(X_test_stand)
365
366
367 train_predictions = (train_predictions > 0.5).astype(int)
368 test_predictions = (test_predictions > 0.5).astype(int)
369
370 # lr = LogisticRegression()
371 # lr.fit(X_train_scaled, y_train)
372 # y_pred = lr.predict(X_test_scaled)
373
374 # nb = GaussianNB(priors = [0.5,0.5])
375
376 # nb.fit(X_train, Y_train.values.ravel())
377
378 # train_predictions = nb.predict(X_train)
379 # test_predictions = nb.predict(X_test)
380
381 lr_train_err = 1 - accuracy_score(y_true=Y_train.values, y_pred=
train_predictions)
382 lr_test_err = 1 - accuracy_score(y_true=Y_test.values, y_pred=
test_predictions)
383 print(f"Train Error: {round(lr_train_err, 4)}")
384 print(f"Test Error: {round(lr_test_err, 4)}")
385
386
387 .. parsed-literal::
388
389 Optimization terminated successfully.
390 Current function value: 0.453582
391 Iterations 6
392
393 Logit Regression Results

```

=====

```

394 Dep. Variable:          win    No. Observations:
      9000
395 Model:              Logit    Df Residuals:
      8988
396 Method:            MLE      Df Model:
      11
397 Date:              Thu, 13 Mar 2025    Pseudo R-squ.:
      0.3456
398 Time:              17:43:24    Log-Likelihood:
      -4082.2
399 converged:          True      LL-Null:
      -6238.2
400 Covariance Type:    nonrobust    LLR p-value:
      0.000
401
=====
402               coef      std err          z      P>|z|
403 [0.025      0.975]
-----
404 const          -0.0717      0.028      -2.580      0.010
      -0.126      -0.017
405 ht             -0.2082      0.030      -6.854      0.000
      -0.268      -0.149
406 age            -0.1853      0.028      -6.661      0.000
      -0.240      -0.131
407 ace            -0.3076      0.039      -7.817      0.000
      -0.385      -0.230
408 df             -0.0243      0.036      -0.676      0.499
      -0.095      0.046
409 svpt           -3.2817      0.268     -12.260      0.000
      -3.806      -2.757
410 1stIn           -0.1982      0.166      -1.193      0.233
      -0.524      0.127
411 1stWon          2.9752      0.181     16.462      0.000
      2.621      3.329
412 2ndWon          1.4224      0.096     14.856      0.000
      1.235      1.610
413 bpSaved         1.7917      0.103     17.404      0.000
      1.590      1.994
414 bpFaced        -2.2650      0.145     -15.580      0.000
      -2.550      -1.980
415 surface_num     -0.1126      0.028      -4.049      0.000
      -0.167      -0.058
416
=====

```

```

417     Train Error: 0.2012
418     Test Error: 0.206
419
420
421 .. code:: ipython3
422
423     X_train = train_features
424     Y_train = train_response
425     X_test = test_features
426     Y_test = test_response
427
428     X_train_stand, Y_train_stand, X_train_stand_scaler,
Y_train_stand_scaler = standardize_data(features=X_train, response
=Y_train)
429     X_test_stand, Y_test_stand, X_test_stand_scaler,
Y_test_stand_scaler = standardize_data(features=X_test, response=
Y_test)
430
431     X_train_stand = sm.add_constant(X_train_stand)
432     X_test_stand = sm.add_constant(X_test_stand)
433
434     n1 = round(len(data)*0.9)
435     n2 = round(len(data)*0.1)
436     b = [10]
437     # Initialize the result DataFrame
438     result_df = pd.DataFrame({
439         "B Cross Validations": [],
440         "Average Error Rate": [],
441         "Average Error Variance": []
442     })
443
444     for B in b:
445         # Reinitialize the lists for storing metrics for each B
446
447         test_error_rates = []
448         test accuracies = []
449
450         for i in range(B):
451
452             X_combined = pd.concat([X_train_stand, X_test_stand], axis
=0)
453             y_combined = pd.concat([Y_train, Y_test], axis=0).values.
flatten()
454
455
456             X_train_sample, X_test_sample, Y_train_sample,
Y_test_sample = train_test_split(
457                 X_combined, y_combined, train_size=n1, test_size=n2,
random_state=i

```

```

458         )
459
460         # Train the model on the training data
461         logit = sm.Logit(Y_train_sample, X_train_sample)
462         logit = logit.fit(dis=0)
463
464         test_predictions = logit.predict(X_test_sample)
465
466
467
468         test_predictions = (test_predictions > 0.5).astype(int)
469         #lr.fit(X_train, Y_train)
470
471
472         # Make predictions on the testing data
473
474
475
476
477         # Calculate error rate (MSE)
478         nb_test_error_rate = 1 - accuracy_score(y_true=
479 Y_test_sample, y_pred=test_predictions)
480
481         # Append to the lists
482
483         test_error_rates.append(nb_test_error_rate)
484
485         # Calculate the variance of the testing error rates after all
486         iterations for this B
487         test_error_variance = np.var(test_error_rates)
488
489         # Calculate the averages for the current B value
490
491         average_test_error_rate = np.mean(test_error_rates)
492
493         metrics_df = pd.DataFrame({
494             "B Cross Validations": [B],
495             "Average Error Rate": [average_test_error_rate],
496             "Average Error Variance": [test_error_variance]
497         })
498
499         result_df = pd.concat([result_df, metrics_df], axis=0)
500
501
502         result_df
503
504

```

```

505
506
507 .. raw:: html
508
509     <div>
510     <style scoped>
511         .dataframe tbody tr th:only-of-type {
512             vertical-align: middle;
513         }
514
515         .dataframe tbody tr th {
516             vertical-align: top;
517         }
518
519         .dataframe thead th {
520             text-align: right;
521         }
522     </style>
523     <table border="1" class="dataframe">
524     <thead>
525         <tr style="text-align: right;">
526             <th></th>
527             <th>B Cross Validations</th>
528             <th>Average Error Rate</th>
529             <th>Average Error Variance</th>
530         </tr>
531     </thead>
532     <tbody>
533         <tr>
534             <th>0</th>
535             <td>10.0</td>
536             <td>0.2083</td>
537             <td>0.000179</td>
538         </tr>
539     </tbody>
540 </table>
541 </div>
542
543
544
545 .. code:: ipython3
546
547     final_cv_result = pd.concat([pd.DataFrame({
548         'Model': ['Logistic Regression'],
549         'Mean Test Error': [result_df["Average Error Rate"].iloc[0]],
550         'Test Error Var': [result_df["Average Error Variance"].iloc
551 [0]]
552     })])
553     #model_errors['Logistic Regression'] = test_error_rates

```

```

553
554 .. code:: ipython3
555
556     X_train = train_features
557     Y_train = train_response
558     X_test = test_features
559     Y_test = test_response
560
561
562     X_train_stand, Y_train_stand, X_train_stand_scaler,
Y_train_stand_scaler = standardize_data(features=X_train, response
=Y_train)
563     X_test_stand, Y_test_stand, X_test_stand_scaler,
Y_test_stand_scaler = standardize_data(features=X_test, response=
Y_test)
564
565
566     K = 101
567     train_error = []
568     test_error = []
569     for k in range(1, K+1, 5):
570         knn = KNeighborsClassifier(n_neighbors=k)
571         knn.fit(X_train_stand, Y_train.values.ravel())
572         train_predictions = knn.predict(X_train_stand)
573         test_predictions = knn.predict(X_test_stand)
574
575         cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
576         train_score = 1 - model_selection.cross_val_score(knn,
X_train_stand, Y_train.astype(int).values.ravel(), cv=cv, scoring=
'accuracy').mean()
577         test_score = 1 - model_selection.cross_val_score(knn,
X_test_stand, Y_test.astype(int).values.ravel(), cv=cv, scoring=
'accuracy').mean()
578
579         # train_error.append(1 - accuracy_score(y_true=Y_train.values.
ravel(), y_pred=train_predictions))
580         train_error.append(train_score)
581         # test_error.append(1 - accuracy_score(y_true=Y_test.values.
ravel(), y_pred=test_predictions))
582         test_error.append(test_score)
583
584
585         # # Create a plot with dual y-axes
586         plt.close('all')
587         fig, ax1 = plt.subplots(figsize=(8, 6))
588
589         # Plot MSE on the first y-axis
590         ax1.plot(range(1, K+1, 5), train_error, 'b-', marker='o', label='CV
(K-fold) Training Error')

```

```

591     ax1.set_xlabel('Number of Neighbors (k)')
592     ax1.set_ylabel('Error', color='blue')
593     ax1.tick_params(axis='y', labelcolor='blue')
594
595
596     ax1.plot(range(1,K+1,5), test_error, 'g-', marker='o', label='CV (
K-fold) Test Error')
597     ax1.set_xlabel('Number of Neighbors (k)')
598     #ax1.set_ylabel('Error', color='blue')
599     ax1.tick_params(axis='y', labelcolor='blue')
600     plt.legend()
601     plt.grid()
602     plt.title("K-Neighbors vs. Error")
603     plt.savefig("K-Neighbors vs. Error.png", format="png", dpi=300)
604
605     # knn_train_err = (train_error[12])
606     # knn_test_err = (test_error[12])
607     # print(f"Train Error: {round(knn_train_err, 4) }")
608     # print(f"Test Error: {round(knn_test_err, 4) }")
609
610
611
612 .. image:: output_15_0.png
613
614
615 .. code:: ipython3
616
617     knn_train_err = (train_error[12])
618     knn_test_err = (test_error[12])
619     print(f"Train Error: {round(knn_train_err, 4) }")
620     print(f"Test Error: {round(knn_test_err, 4) }")
621
622
623 .. parsed-literal::
624
625     Train Error: 0.2593
626     Test Error: 0.277
627
628
629 .. code:: ipython3
630
631     X_train = train_features
632     Y_train = train_response
633     X_test = test_features
634     Y_test = test_response
635
636
637     X_train_stand, Y_train_stand,X_train_stand_scaler,
Y_train_stand_scaler = standardize_data(features=X_train, response
=Y_train)

```



```

637     X_test_stand, Y_test_stand, X_test_stand_scaler,
        Y_test_stand_scaler = standardize_data(features=X_test, response=
        Y_test)
638
639
640     n1 = round(len(data)*0.9)
641     n2 = round(len(data)*0.1)
642     b = [10]
643     # Initialize the result DataFrame
644     result_df = pd.DataFrame({
645         "B Cross Validations": [],
646         "Average Error Rate": [],
647         "Average Error Variance": []
648     })
649
650     for B in b:
651         # Reinitialize the lists for storing metrics for each B
652
653         test_error_rates = []
654         test_accuracies = []
655
656         for i in range(B):
657
658             X_combined = pd.concat([X_train_stand, X_test_stand], axis
        =0)
659             y_combined = pd.concat([Y_train, Y_test], axis=0).values.
        flatten()
660
661
662             X_train_sample, X_test_sample, Y_train_sample,
        Y_test_sample = train_test_split(
663                 X_combined, y_combined, train_size=n1, test_size=n2,
        random_state=i
664             )
665
666             # Train the model on the training data
667             knn = KNeighborsClassifier(n_neighbors=60)
668             knn.fit(X_train_stand, Y_train.values.ravel())
669
670             test_predictions = knn.predict(X_test_sample)
671
672
673
674             test_predictions = (test_predictions > 0.5).astype(int)
675             #lr.fit(X_train, Y_train)
676
677
678             # Make predictions on the testing data
679

```

```

680
681
682
683     # Calculate error rate (MSE)
684     nb_test_error_rate = 1 - accuracy_score(y_true=
Y_test_sample, y_pred=test_predictions)
685
686     # Append to the lists
687
688     test_error_rates.append(nb_test_error_rate)
689
690     # Calculate the variance of the testing error rates after all
iterations for this B
691     test_error_variance = np.var(test_error_rates)
692
693     # Calculate the averages for the current B value
694
695     average_test_error_rate = np.mean(test_error_rates)
696
697
698     metrics_df = pd.DataFrame({
699         "B Cross Validations": [B],
700         "Average Error Rate": [average_test_error_rate],
701         "Average Error Variance": [test_error_variance]
702     })
703
704
705     result_df = pd.concat([result_df, metrics_df], axis=0)
706
707
708     result_df
709
710
711
712
713 .. raw:: html
714
715     <div>
716     <style scoped>
717         .dataframe tbody tr th:only-of-type {
718             vertical-align: middle;
719         }
720
721         .dataframe tbody tr th {
722             vertical-align: top;
723         }
724
725         .dataframe thead th {
726             text-align: right;

```

```

727     }
728 </style>
729 <table border="1" class="dataframe">
730   <thead>
731     <tr style="text-align: right;">
732       <th></th>
733       <th>B Cross Validations</th>
734       <th>Average Error Rate</th>
735       <th>Average Error Variance</th>
736     </tr>
737   </thead>
738   <tbody>
739     <tr>
740       <th>0</th>
741       <td>10.0</td>
742       <td>0.2497</td>
743       <td>0.000112</td>
744     </tr>
745   </tbody>
746 </table>
747 </div>
748
749
750
751 .. code:: ipython3
752
753     final_cv_result = pd.concat([final_cv_result,pd.DataFrame({
754         'Model': ['KNN'],
755         'Mean Test Error': [result_df["Average Error Rate"].iloc[0]],
756         'Test Error Var': [result_df["Average Error Variance"].iloc
757         [0]]
758     }))]
759
760 .. code:: ipython3
761
762     X_train = train_features
763     Y_train = train_response
764     X_test = test_features
765     Y_test = test_response
766
767     # Standardize Data (Assuming standardize_data() function works as
768     expected)
769     X_train_stand, Y_train_stand, X_train_stand_scaler,
770     Y_train_stand_scaler = standardize_data(features=X_train, response
771     =Y_train)
772     X_test_stand, Y_test_stand, X_test_stand_scaler,
773     Y_test_stand_scaler = standardize_data(features=X_test, response=
774     Y_test)

```

```

770     Y_train_stand = Y_train.values.ravel() # Ensure Y values remain
binary (0/1)
771     Y_test_stand = Y_test.values.ravel()
772
773     # Define range of estimators (1 to 101 in steps of 5)
774     n_estimators_values = list(range(1, 101, 5))
775
776     # Lists to store errors
777     train_error = []
778     test_error = []
779
780     # Cross-validation setup
781     cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
782
783     # Loop through different n_estimators values
784     for n_estimators in tqdm(n_estimators_values):
785         rf_model = RandomForestClassifier(n_estimators=n_estimators,
max_depth=3, random_state=42)
786
787         # Fit the model
788         rf_model.fit(X_train_stand, Y_train_stand)
789
790         # Cross-validation scores
791         train_score = 1 - cross_val_score(rf_model, X_train_stand,
Y_train_stand, cv=cv, scoring='accuracy').mean()
792         test_score = 1 - cross_val_score(rf_model, X_test_stand,
Y_test_stand, cv=cv, scoring='accuracy').mean()
793
794         # Append errors
795         train_error.append(train_score)
796         test_error.append(test_score)
797
798     # Plot results
799     plt.figure(figsize=(8, 6))
800     plt.plot(n_estimators_values, train_error, 'b-', marker='o', label=
'CV (K-fold) Training Error')
801     plt.plot(n_estimators_values, test_error, 'g-', marker='o', label=
'CV (K-fold) Test Error')
802     plt.xlabel('Number of Estimators')
803     plt.ylabel('Error')
804     plt.title("Random Forest: Number of Estimators vs. Error")
805     plt.legend()
806     plt.grid()
807     plt.savefig("RandomForest_NumEstimators_vs_Error.png", format="png
", dpi=300)
808     plt.show()
809
810
811 .. parsed-literal::

```

```

812
813
814
815 .. image:: output_19_1.png
816
817
818 .. code:: ipython3
819
820     X_train = train_features
821     Y_train = train_response
822     X_test = test_features
823     Y_test = test_response
824
825     X_train_stand, Y_train_stand, X_train_stand_scaler,
826     Y_train_stand_scaler = standardize_data(features=X_train, response
827     =Y_train)
828
829     X_test_stand, Y_test_stand, X_test_stand_scaler,
830     Y_test_stand_scaler = standardize_data(features=X_test, response=
831     Y_test)
832
833
834     Y_train_stand = Y_train.values.ravel() # Ensure Y values remain
835     binary (0/1)
836     Y_test_stand = Y_test.values.ravel()
837
838
839     n1 = round(len(data)*0.9)
840     n2 = round(len(data)*0.1)
841     b = [10]
842     # Initialize the result DataFrame
843     result_df = pd.DataFrame({
844         "B Cross Validations": [],
845         "Average Error Rate": [],
846         "Average Error Variance": []
847     })
848
849     for B in b:
850         # Reinitialize the lists for storing metrics for each B
851
852         test_error_rates = []
853         test_accuracies = []
854
855         for i in range(B):
856
857             X_combined = pd.concat([X_train_stand, X_test_stand], axis
858             =0)
859             y_combined = pd.concat([Y_train, Y_test], axis=0).values.
860             flatten()
861
862
863

```

```

854     X_train_sample, X_test_sample, Y_train_sample,
Y_test_sample = train_test_split(
855         X_combined, y_combined, train_size=n1, test_size=n2,
random_state=i
856     )
857
858     # Train the model on the training data
859     rf_model = RandomForestClassifier(n_estimators=80,
max_depth=3, random_state=42)
860
861     # Fit the model
862     rf_model.fit(X_train_stand, Y_train_stand)
863
864     test_predictions = rf_model.predict(X_test_sample)
865
866
867
868     test_predictions = (test_predictions > 0.5).astype(int)
869     #lr.fit(X_train, Y_train)
870
871
872     # Make predictions on the testing data
873
874
875
876
877     # Calculate error rate (MSE)
878     nb_test_error_rate = 1 - accuracy_score(y_true=
Y_test_sample, y_pred=test_predictions)
879
880     # Append to the lists
881
882     test_error_rates.append(nb_test_error_rate)
883
884     # Calculate the variance of the testing error rates after all
iterations for this B
885     test_error_variance = np.var(test_error_rates)
886
887     # Calculate the averages for the current B value
888
889     average_test_error_rate = np.mean(test_error_rates)
890
891
892     metrics_df = pd.DataFrame({
893         "B Cross Validations": [B],
894         "Average Error Rate": [average_test_error_rate],
895         "Average Error Variance": [test_error_variance]
896     })
897

```

```

898
899     result_df = pd.concat([result_df, metrics_df], axis=0)
900
901
902     result_df
903
904
905
906
907 .. raw:: html
908
909     <div>
910     <style scoped>
911         .dataframe tbody tr th:only-of-type {
912             vertical-align: middle;
913         }
914
915         .dataframe tbody tr th {
916             vertical-align: top;
917         }
918
919         .dataframe thead th {
920             text-align: right;
921         }
922     </style>
923     <table border="1" class="dataframe">
924         <thead>
925             <tr style="text-align: right;">
926                 <th></th>
927                 <th>B Cross Validations</th>
928                 <th>Average Error Rate</th>
929                 <th>Average Error Variance</th>
930             </tr>
931         </thead>
932         <tbody>
933             <tr>
934                 <th>0</th>
935                 <td>10.0</td>
936                 <td>0.2869</td>
937                 <td>0.000169</td>
938             </tr>
939         </tbody>
940     </table>
941     </div>
942
943
944
945 .. code:: ipython3
946

```

```

947     final_cv_result = pd.concat([final_cv_result, pd.DataFrame({
948         'Model': ['Random Forest'],
949         'Mean Test Error': [result_df["Average Error Rate"].iloc[0]],
950         'Test Error Var': [result_df["Average Error Variance"].iloc
951         [0]]
952     })])
953 .. code:: ipython3
954
955     X_train = train_features
956     Y_train = train_response
957     X_test = test_features
958     Y_test = test_response
959
960     # Standardize Data (Assuming standardize_data() function works as
961     # expected)
962     X_train_stand, Y_train_stand, X_train_stand_scaler,
963     Y_train_stand_scaler = standardize_data(features=X_train, response
964     =Y_train)
965     X_test_stand, Y_test_stand, X_test_stand_scaler,
966     Y_test_stand_scaler = standardize_data(features=X_test, response=
967     Y_test)
968
969     Y_train_stand = Y_train
970     Y_test_stand = Y_test
971
972     # Define range of learning rates (eta)
973     eta_values = np.linspace(0.01, 0.36, 25) # From 0.01 to 0.3 in 10
974     steps
975
976     # Lists to store errors
977     train_error = []
978     test_error = []
979
980     # Cross-validation setup
981     cv = RepeatedKFold(n_splits=10, n_repeats=5, random_state=42)
982
983     # Loop through different eta values
984     for eta in tqdm(eta_values):
985         xgb_model = xgb.XGBClassifier(learning_rate=eta, n_estimators
986         =80, max_depth=3, random_state=42, eval_metric="error")
987
988         # Fit the model
989         xgb_model.fit(X_train_stand, Y_train_stand)
990
991         # Cross-validation scores
992         train_score = 1 - cross_val_score(xgb_model, X_train_stand,
993         Y_train_stand, cv=cv, scoring='accuracy').mean()

```



```

986     test_score = 1 - cross_val_score(xgb_model, X_test_stand,
Y_test_stand, cv=cv, scoring='accuracy').mean()
987
988     # Append errors
989     train_error.append(train_score)
990     test_error.append(test_score)
991
992     # Plot results
993     plt.figure(figsize=(8, 6))
994     plt.plot(eta_values, train_error, 'b-', marker='o', label='CV (K-
fold) Training Error')
995     plt.plot(eta_values, test_error, 'g-', marker='o', label='CV (K-
fold) Test Error')
996     plt.xlabel('Learning Rate (eta)')
997     plt.ylabel('Error')
998     plt.title("XGBoost Learning Rate vs. Error")
999     plt.legend()
1000     plt.grid()
1001     plt.savefig("XGBoost_LearningRate_vs_Error.png", format="png", dpi
=300)
1002     plt.show()
1003
1004
1005 .. parsed-literal::
1006
1007
1008
1009
1010
1011 .. image:: output_22_1.png
1012
1013
1014 .. code:: ipython3
1015
1016     xgb_model = xgb.XGBClassifier(learning_rate=.21, n_estimators=80,
max_depth=3, random_state=42, eval_metric="error")
1017
1018     # Fit the model
1019     xgb_model.fit(X_train_stand, Y_train_stand)
1020
1021     # Cross-validation scores
1022     train_score = 1 - cross_val_score(xgb_model, X_train_stand,
Y_train_stand, cv=cv, scoring='accuracy').mean()
1023     test_score = 1 - cross_val_score(xgb_model, X_test_stand,
Y_test_stand, cv=cv, scoring='accuracy').mean()
1024
1025     print("train: : ", train_score)
1026     print("test: : ", test_score)
1027

```

```

1028
1029 .. parsed-literal::
1030
1031     train: : 0.21211111111111114
1032     test: : 0.24760000000000015
1033
1034
1035 .. code:: ipython3
1036
1037     X_train = train_features
1038     Y_train = train_response
1039     X_test = test_features
1040     Y_test = test_response
1041
1042     X_train_stand, Y_train_stand, X_train_stand_scaler,
1043     Y_train_stand_scaler = standardize_data(features=X_train, response
1044     =Y_train)
1045
1046     X_test_stand, Y_test_stand, X_test_stand_scaler,
1047     Y_test_stand_scaler = standardize_data(features=X_test, response=
1048     Y_test)
1049
1050
1051     Y_train_stand = Y_train# Ensure Y values remain binary (0/1)
1052     Y_test_stand = Y_test
1053
1054
1055     n1 = round(len(data)*0.9)
1056     n2 = round(len(data)*0.1)
1057     b = [100]
1058     # Initialize the result DataFrame
1059     result_df = pd.DataFrame({
1060         "B Cross Validations": [],
1061         "Average Error Rate": [],
1062         "Average Error Variance": []
1063     })
1064
1065     for B in tqdm(b):
1066         # Reinitialize the lists for storing metrics for each B
1067
1068         test_error_rates = []
1069         test_accuracies = []
1070
1071         for i in range(B):
1072
1073             X_combined = pd.concat([X_train_stand, X_test_stand], axis
1074             =0)
1075
1076             y_combined = pd.concat([Y_train, Y_test], axis=0).values.
1077             flatten()

```

```

1071     X_train_sample, X_test_sample, Y_train_sample,
Y_test_sample = train_test_split(
1072         X_combined, y_combined, train_size=n1, test_size=n2,
random_state=i
1073     )
1074
1075     # Train the model on the training data
1076     xgb_model = xgb.XGBClassifier(learning_rate=.21,
n_estimators=80, max_depth=3, random_state=42, eval_metric="error"
)
1077
1078     # Fit the model
1079     xgb_model.fit(X_train_stand, Y_train_stand)
1080
1081     test_predictions = xgb_model.predict(X_test_sample)
1082
1083
1084
1085     test_predictions = (test_predictions > 0.5).astype(int)
1086     #lr.fit(X_train, Y_train)
1087
1088
1089     # Make predictions on the testing data
1090
1091
1092
1093
1094     # Calculate error rate (MSE)
1095     nb_test_error_rate = 1 - accuracy_score(y_true=
Y_test_sample, y_pred=test_predictions)
1096
1097     # Append to the lists
1098
1099     test_error_rates.append(nb_test_error_rate)
1100
1101     # Calculate the variance of the testing error rates after all
iterations for this B
1102     test_error_variance = np.var(test_error_rates)
1103
1104     # Calculate the averages for the current B value
1105
1106     average_test_error_rate = np.mean(test_error_rates)
1107
1108
1109     metrics_df = pd.DataFrame({
1110         "B Cross Validations": [B],
1111         "Average Error Rate": [average_test_error_rate],
1112         "Average Error Variance": [test_error_variance]
1113     })

```

```

1114
1115
1116     result_df = pd.concat([result_df, metrics_df], axis=0)
1117
1118
1119     result_df
1120
1121
1122 .. parsed-literal::
1123
1124
1125
1126
1127
1128
1129 .. raw:: html
1130
1131     <div>
1132     <style scoped>
1133         .dataframe tbody tr th:only-of-type {
1134             vertical-align: middle;
1135         }
1136
1137         .dataframe tbody tr th {
1138             vertical-align: top;
1139         }
1140
1141         .dataframe thead th {
1142             text-align: right;
1143         }
1144     </style>
1145     <table border="1" class="dataframe">
1146     <thead>
1147         <tr style="text-align: right;">
1148             <th></th>
1149             <th>B Cross Validations</th>
1150             <th>Average Error Rate</th>
1151             <th>Average Error Variance</th>
1152         </tr>
1153     </thead>
1154     <tbody>
1155         <tr>
1156             <th>0</th>
1157             <td>100.0</td>
1158             <td>0.18749</td>
1159             <td>0.00016</td>
1160         </tr>
1161     </tbody>
1162 </table>

```

```

1163     </div>
1164
1165
1166
1167 .. code:: ipython3
1168
1169     final_cv_result = pd.concat([final_cv_result, pd.DataFrame({
1170         'Model': ['XGBoost'],
1171         'Mean Test Error': [result_df["Average Error Rate"].iloc[0]],
1172         'Test Error Var': [result_df["Average Error Variance"].iloc
1173     [0]]
1174     }))]
1175
1176 .. code:: ipython3
1177
1178     final_cv_result
1179
1180
1181
1182 .. raw:: html
1183
1184     <div>
1185     <style scoped>
1186         .dataframe tbody tr th:only-of-type {
1187             vertical-align: middle;
1188         }
1189
1190         .dataframe tbody tr th {
1191             vertical-align: top;
1192         }
1193
1194         .dataframe thead th {
1195             text-align: right;
1196         }
1197     </style>
1198     <table border="1" class="dataframe">
1199         <thead>
1200             <tr style="text-align: right;">
1201                 <th></th>
1202                 <th>Model</th>
1203                 <th>Mean Test Error</th>
1204                 <th>Test Error Var</th>
1205             </tr>
1206         </thead>
1207         <tbody>
1208             <tr>
1209                 <th>0</th>
1210                 <td>Logistic Regression</td>

```

```

1211         <td>0.20830</td>
1212         <td>0.000179</td>
1213     </tr>
1214     <tr>
1215         <th>0</th>
1216         <td>KNN</td>
1217         <td>0.24970</td>
1218         <td>0.000112</td>
1219     </tr>
1220     <tr>
1221         <th>0</th>
1222         <td>Random Forest</td>
1223         <td>0.28690</td>
1224         <td>0.000169</td>
1225     </tr>
1226     <tr>
1227         <th>0</th>
1228         <td>XGBoost</td>
1229         <td>0.18749</td>
1230         <td>0.000160</td>
1231     </tr>
1232 </tbody>
1233 </table>
1234 </div>
1235
1236
1237
1238 .. code:: ipython3
1239
1240     # X_train = train_features
1241     # Y_train = train_response
1242     # X_test = test_features
1243     # Y_test = test_response
1244
1245     # #model = LinearRegression()
1246
1247     # sfs = SequentialFeatureSelector(estimator = linear_model.
1248     LinearRegression(),
1249     #
1250     k_features='best',
1251     #
1252     forward=True,
1253     #
1254     floating = True,
1255     #
1256     scoring=aic_score,
1257     #
1258     cv=None)
1259     # selected_features = sfs.fit(X_train, Y_train)
1260     # selected_features = list(sfs.k_feature_names_)
1261
1262     # X_train_stepwise = X_train[selected_features]
1263     # X_test_stepwise = X_test[selected_features]
1264
1265

```

```

1259 # X_train_stepwise = sm.add_constant(X_train_stepwise)
1260 # X_test_stepwise = sm.add_constant(X_test_stepwise)
1261
1262 # lr_stepwise = sm.OLS(Y_train, X_train_stepwise).fit()
1263
1264 # train_predictions = lr_stepwise.predict(X_train_stepwise)
1265 # test_predictions = lr_stepwise.predict(X_test_stepwise)
1266
1267
1268 # lr_stepwise_train_mse = mean_squared_error(y_true=Y_train.values
, y_pred=train_predictions)
1269 # lr_stepwise_test_mse = mean_squared_error(y_true=Y_test.values,
y_pred=test_predictions)
1270
1271 # print(lr_stepwise.summary())
1272
1273 # print(f"Train MSE: {round(lr_stepwise_train_mse, 4) }")
1274 # print(f"Test MSE: {round(lr_stepwise_test_mse, 4) }")
1275
1276 .. code:: ipython3
1277
1278 Y_test_probs = logit.predict(X_test_stand) # Use probability
scores
1279
1280 # Compute ROC curve and AUC
1281 fpr, tpr, thresholds = roc_curve(Y_test, Y_test_probs)
1282 roc_auc = auc(fpr, tpr)
1283
1284 # Plot ROC curve
1285 plt.figure(figsize=(8, 6))
1286 plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (
AUC = {roc_auc:.2f})')
1287 plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label
='Chance (AUC = 0.50)')
1288 plt.xlabel('False Positive Rate')
1289 plt.ylabel('True Positive Rate')
1290 plt.title('Logistic Regression ROC Curve')
1291 plt.legend(loc="lower right")
1292
1293 # Save and show plot
1294 plt.grid(True)
1295 plt.savefig("roc_curve.png", format="png", dpi=300)
1296 plt.show()
1297
1298 knn = KNeighborsClassifier(n_neighbors=60)
1299 knn.fit(X_train_stand, Y_train.values.ravel())
1300 Y_test_probs = knn.predict(X_test_stand)
1301
1302

```

```

1303     # Compute ROC curve and AUC
1304     fpr, tpr, thresholds = roc_curve(Y_test, Y_test_probs)
1305     roc_auc = auc(fpr, tpr)
1306
1307     # Plot ROC curve
1308     plt.figure(figsize=(8, 6))
1309     plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (
1310     AUC = {roc_auc:.2f})')
1311     plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label
1312     = 'Chance (AUC = 0.50)')
1313     plt.xlabel('False Positive Rate')
1314     plt.ylabel('True Positive Rate')
1315     plt.title('KNN ROC Curve')
1316     plt.legend(loc="lower right")
1317
1318     # Save and show plot
1319     plt.grid(True)
1320     plt.savefig("roc_curve.png", format="png", dpi=300)
1321     plt.show()
1322
1323     rf_model = RandomForestClassifier(n_estimators=80, max_depth=3,
1324     random_state=42)
1325
1326     # Fit the model
1327     rf_model.fit(X_train_stand, Y_train_stand)
1328     Y_test_probs = rf_model.predict(X_test_stand)
1329
1330     # Compute ROC curve and AUC
1331     fpr, tpr, thresholds = roc_curve(Y_test, Y_test_probs)
1332     roc_auc = auc(fpr, tpr)
1333
1334     # Plot ROC curve
1335     plt.figure(figsize=(8, 6))
1336     plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (
1337     AUC = {roc_auc:.2f})')
1338     plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label
1339     = 'Chance (AUC = 0.50)')
1340     plt.xlabel('False Positive Rate')
1341     plt.ylabel('True Positive Rate')
1342     plt.title('Random Forrest ROC Curve')
1343     plt.legend(loc="lower right")
1344
1345     # Save and show plot
1346     plt.grid(True)
1347     plt.savefig("roc_curve.png", format="png", dpi=300)
1348     plt.show()
1349
1350     xgb_model = xgb.XGBClassifier(learning_rate=.21, n_estimators=80,
1351     max_depth=3, random_state=42, eval_metric="error")

```



```

1346 # Fit the model
1347 xgb_model.fit(X_train_stand, Y_train_stand)
1348 Y_test_probs = xgb_model.predict(X_test_stand)
1349
1350
1351 # Compute ROC curve and AUC
1352 fpr, tpr, thresholds = roc_curve(Y_test, Y_test_probs)
1353 roc_auc = auc(fpr, tpr)
1354
1355 # Plot ROC curve
1356 plt.figure(figsize=(8, 6))
1357 plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (
1358 AUC = {roc_auc:.2f})')
1359 plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--', label
1360 = 'Chance (AUC = 0.50)')
1361 plt.xlabel('False Positive Rate')
1362 plt.ylabel('True Positive Rate')
1363 plt.title('XGBoost ROC Curve')
1364 plt.legend(loc="lower right")
1365
1366 # Save and show plot
1367 plt.grid(True)
1368 plt.savefig("roc_curve.png", format="png", dpi=300)
1369 plt.show()
1370 ::
1371
1372
1373

```

---

```

1374
1375 ValueError                                Traceback (most recent
1376 call last)
1377
1378 Cell In[28], line 1
1379 ----> 1 Y_test_probs = logit.predict(X_test_stand) # Use
1380 probability scores
1381         3 # Compute ROC curve and AUC
1382         4 fpr, tpr, thresholds = roc_curve(Y_test, Y_test_probs)
1383
1384 File ~\miniconda3\Lib\site-packages\statsmodels\base\model.py
1385 :1174, in Results.predict(self, exog, transform, *args, **kwargs)
1386     1127 """
1387     1128 Call self.model.predict with self.params as the first
1388     argument.
1389     1129

```

```

1387         (...)
1388         1169 returned prediction.
1389         1170 """
1390         1171 exog, exog_index = self._transform_predict_exog(exog,
1391         1172                                                         transform=
transform)
1392     -> 1174 predict_results = self.model.predict(self.params, exog, *
args,
1393         1175                                                         **kwargs)
1394         1177 if exog_index is not None and not hasattr(predict_results,
1395         1178                                                         ,
predicted_values'):
1396         1179         if predict_results.ndim == 1:
1397
1398
1399     File ~\miniconda3\Lib\site-packages\statsmodels\discrete\
discrete_model.py:543, in BinaryModel.predict(self, params, exog,
which, linear, offset)
1400         540 if exog is None:
1401         541         exog = self.exog
1402     --> 543 linpred = np.dot(exog, params) + offset
1403         545 if which == "mean":
1404         546         return self.cdf(linpred)
1405
1406
1407     ValueError: shapes (1000,11) and (12,) not aligned: 11 (dim 1) !=
12 (dim 0)

```

Listing 1: Code