DSO530: Applied Modern Statistical Learning Methods

Options Pricing Group Project

May 2024

by

Group 45

Amrusha Buddhiraju - 8152387596

Chinmay Atul Sashittal - 1262643453

Evan Benham - 6591536764

Muskan Aggarwal - 7683221886

Muhammad Murtadha Ramadhan - 4034705476

Vineetha Jinan - 2295890291 (vjinan@usc.edu)

Executive Summary

In the world of call option pricing, precision stands as a cornerstone for informed decision-making, underscoring the vital importance of prediction accuracy for investors seeking to navigate complex financial markets. However, alongside precision, we also recognize the essential role of interpretability, especially in unraveling the intricate web of factors that influence option prices. In the dynamic landscape of financial markets, where everything hinges on multifaceted variables and complex relationships, understanding these underlying drivers is imperative to sound decision-making. With this intuition in mind, we embarked on our analysis of predicting European call option pricing based on current asset value, strike price, annual interest rate and time to maturity, utilizing a number of different approaches in both regression and classification, such as support vector regression, KNN classifier, and others. Our champion machine learning models, random forest regression and random forest classifier, showcased superior performance compared to the Black-Scholes model, yielding a 99.66% mean r-squared and 6.36% classification error rate, respectively. This validation not only underscores the relevance of machine learning in financial analysis, but also highlights its potential to provide deeper insights for investors. By leveraging these advanced techniques, investors may be able to gain a more comprehensive understanding of option pricing dynamics, enabling them to make more informed decisions within the ever-evolving landscape of financial markets.

From a business perspective, we advocate for the inclusion of all relevant predictor variables in our prediction models. Variable selection, while tempting for simplification, risks overlooking critical factors that could significantly influence option prices. By encompassing all relevant variables, our models provide a comprehensive understanding of the market dynamics, facilitating more informed decision-making processes. However, with regard to the applicability of our trained model to predict option values for stocks, such as Tesla, we are cautious. While our model demonstrates proficiency in predicting option values based on S&P 500 data, extending its application to Tesla stock requires thorough validation and consideration of unique market dynamics specific to Tesla. However, with proper validation and calibration, as well as sufficient data and computing power, our model could serve as a valuable tool for option pricing for Tesla stock, offering insights to guide strategic, data-driven decisions for investors.

In the following report, we will provide further details of our analysis. We will first give an overview of our team's objective, with important background into call option pricing, its significance, and the existing Black-Scholes model for prediction. We will then delve into our methodology, giving a brief summary of our datasets and data preprocessing, before detailing our regression and classification analyses. Finally, we will conclude our report with a more comprehensive exploration of the potential business impact of our findings.

Background

This project revolves around the valuation of European call options on the S&P 500 index. A European call option grants the holder the right, but not the obligation, to purchase an asset at a predetermined price, called the strike price, on a specific expiration date. These options are pivotal financial instruments used for hedging risk, speculative trading, and enhancing portfolio returns due to their leverage effect.

Traditionally, the Black-Scholes model has been a foundational tool for pricing European call options. Developed by Fischer Black and Myron Scholes in 1973, this model calculates the theoretical value of options using parameters such as the strike price, underlying asset price, volatility of the asset, time to expiration, and the risk-free rate of return. The model's formulation is grounded in the assumption that the asset prices follow a geometric Brownian motion with constant drift and volatility. In 1997, the importance of the Black-Scholes model was recognized with the award of the Nobel Prize in Economics.

However, despite its widespread use and theoretical appeal, the Black-Scholes model has limitations, particularly in high-frequency trading environments where market conditions can change rapidly. It assumes volatility is constant and markets are static without accounting for more complex factors like varying interest rates, transaction costs, or divergences from the log-normal distribution of asset returns. These simplifications can lead to pricing errors, particularly for options that are far from expiration or deep in/out of the money.

Given these limitations, there is a growing interest in exploring alternative models and methodologies that incorporate more dynamic variables and better handle real-world market complexities. Machine learning approaches, for example, offer the potential to model option prices based on non-linear relationships and complex market dynamics that traditional models might not capture effectively. This project aims to explore such alternatives, comparing their performance against the Black-Scholes formula to potentially uncover more robust methods of option valuation.

Methodology

Data Sets

The dataset focuses on European call option pricing data on the S&P 500, incorporating variables like current asset value (S), strike price (K), annual interest rate (r), and time to maturity (tau). For regression analysis, the dependent variable is the current option value (Value), and for classification analysis, it is the binary indicator (BS), denoting whether the option's value prediction is over or under the actual value. The dataset includes a training set with detailed records for each option, and a testing set without labels for Value and BS, intended to validate the performance of machine learning models developed from the training data.

Data Preprocessing

During the data processing phase, multiple steps were undertaken to prepare the data for analysis. These included exploratory data analysis (See Appendix), checking for missing values to ensure data completeness and standardizing data to maintain consistency across the dataset.

- **Missing value check**: Upon examination of both the train and test datasets, it was found that there was no missing data in any of the columns. As a result, there was no need for imputation to fill in missing values.. (See Figure 1)
- **Data Standardization:** Before proceeding with further analysis, data standardization was also carried out. This step ensured that the dataset had a uniform scale, enhancing the reliability of any analytical methods applied later. Standardizing the data helped in comparing features that had different units or scales effectively.

Regression analysis

After preprocessing, the dataset was ready for the data modeling process. To have a robust modeling process, 10-fold cross-validation (10-fold CV) was also implemented. Then, initial data modeling was performed for several algorithms which were Linear Regression, Random Forest, KNN Regression, and Support Vector Regression with default hyperparameters used in each algorithm. To test model performance we used the validation set approach by splitting the available training data into test and train datasets (20% test and 80% train).

Linear Regression: Linear regression is a simple yet powerful algorithm that assumes a linear relationship between the input features and the target variable.

- Performance: The initial modeling using linear regression yielded a mean R-squared score of 92.44% from cross-validation. While linear regression is straightforward and easy to interpret, its performance may be limited due to potential multicollinearity caused by the correlated variables.

Random Forest Regression: Random forest regression is an ensemble learning method that builds multiple decision trees during training and outputs the mean prediction of the individual trees. Its ability to handle non-linear relationships and feature interactions made it an ideal choice.

- Performance: Random forest regression demonstrated superior performance compared to other algorithms, achieving an impressive mean R-squared score of 99.66%.

KNN Regression: K-nearest neighbors (KNN) regression is a non-parametric algorithm that predicts the target variable by averaging the values of its k-nearest neighbors in the feature space.

- Performance: KNN regression performed reasonably well, yielding a mean R-squared score of 94.82% from cross-validation.

Support Vector Regression (SVR): Support vector regression is a supervised learning algorithm that finds the optimal hyperplane in a high-dimensional space to minimize the error between the predicted and actual values. SVR can handle non-linear relationships through the use of kernel functions

- Performance: SVR exhibited the lowest performance among the algorithms considered, with a mean R-squared score of 59.15% from cross-validation.

From the initial modeling result, it can be seen that Random Forest Regression outperformed all other algorithms with an R-squared score of 99.66%. Then, hyperparameter tuning for each algorithm was performed to fine-tune the model as well as to find the best hyperparameter for prediction. The method used for hyperparameter tuning in this case was GridSearch Cross Validation.

Modeling Technique	**Best Hyperparameters**	**Mean R-Squared Score from CV**
Linear Regression	{}	92.44%
Random Forest Regression	{'max_features': None, 'n_estimators': 200}	99.67%
KNN Regression	{'n_neighbors': 10, 'weights': 'distance'}	96.14%
Support Vector Regression	{'gamma': 'scale'}	59.15%

From the hyperparameter tuning process, it can be observed that Random Forest Regression still outperformed the other algorithms in terms of R-squared score performance with the optimal hyperparameters making it our final choice for Value prediction.

Classification Analysis

For predicting the binary status (BS) of over or underestimation in European call option pricing, we evaluated Logistic Regression, Random Forest Classifier, Support Vector Machine (SVM), and K-Nearest Neighbors (KNN) Classifier methods. We tested the performance of these approaches, by comparing their classification errors. To do this we used the validation set approach by splitting the available training data into test and train datasets (20% test and 80% train).

Logistic Regression: Logistic regression is a linear model used for binary classification tasks, estimating the probability of a binary outcome based on one or more predictor variables. It was considered due to its straightforward application in binary classification tasks and its ability to provide probabilities, which are useful for threshold tuning.

- Performance: Logistic regression achieved a classification error rate of 12.26%.

Random Forest Classifier: Random forest classifier is an ensemble learning method that constructs multiple decision trees during training and outputs the mode of the classes (for classification tasks) of the individual trees. Its ability to handle complex relationships and reduce overfitting through ensemble learning made it highly suitable for this task.

- Performance: Random forest classifier demonstrated the best performance, with a classification error rate of 6.36%.

Support Vector Machine (SVM): SVM is a supervised learning algorithm that finds the optimal hyperplane to separate data points into different classes in a high-dimensional space. It was chosen for its robustness in high-dimensional spaces.

- Performance: SVM exhibited a classification error rate of 15.8%.

K-Nearest Neighbors (KNN) Classifier: KNN classifier is a simple and interpretable non-parametric algorithm that assigns a class label to an instance based on the majority class of its k nearest neighbors in the feature space.

- Performance: KNN classifier achieved a classification error rate of 14.76%.

This selection process was crucial in ensuring we chose the most suitable model for our classification needs, with the Random Forest ultimately standing out due to its low error rate and robust performance across various metrics.

Modeling Technique	**Hyperparameters Used**	**Classification Error Rate**
Logistic Regression	Max_iter = 200	0.1226
Random Forest Classifier	N_estimators = 200	0.0636
Support Vector Machine	Kernel = ‘RBF’	0.158
KNN Classifier	K = 5	0.1476

We see that the Random Forest Error classifier with 200 estimators gave us the lowest classification error of 6.36% meaning that only 6.36% of options would be misclassified. When it comes to predicting European call option pricing, accuracy is the highest priority, leading us to random forest classifier as our final choice for predicting BS over/under prediction. Beyond classification error, Random forest has the following advantages over other models:

- **Robustness to Overfitting and Reduced Variance**: By aggregating the predictions of multiple decision trees, Random Forests are less prone to overfitting and have lower variance.
- **Feature Importance**: By evaluating feature importance based on how much each feature decreases impurity across all trees, Random Forests provide insights into which features are critical for classification. (See Figure 3)

The classification prediction can further be improved by using ensemble methods that combine multiple Random Forest models or other types of models to boost performance further.

Business Impact

Leveraging machine learning models for option pricing compared to the traditional Black-Scholes formula has the following business implications and considerations for real-world application :

- **Including All Predictor Variables**: Initially, including all four predictor variables (current asset value, strike price, interest rate, and time to maturity) can be advantageous. This allows the model to identify which factors have the most significant influence on option pricing. Subsequently, feature selection techniques can be employed to pinpoint the most impactful variables, potentially enhancing model efficiency.
- **Machine learning vs. Black-Scholes**: ML models can outperform the Black-Scholes formula in predicting option values due to their ability to capture complex relationships between multiple variables and the option price that may be neglected otherwise. The Black-Scholes formula relies on assumptions that may not always reflect real-world market conditions. While machine learning models can achieve higher accuracy by learning complex patterns from historical data, this complexity can lead to models that are difficult to interpret and computationally expensive to run. Additionally, models trained on historical data may not generalize well to unseen market conditions.
- **Model applicability to new stocks**: Directly applying the trained model to new stocks is not ideal. The model is trained on historical data, and market dynamics can evolve considerably. Before applying the model to a specific stock like Tesla, it's advisable to first assess its performance on unseen data from the same asset class (S&P 500). Additionally, incorporating domain knowledge about the stock market and Tesla specifically could further refine the model's accuracy.
- **Prediction Accuracy vs. Interpretability**: In the context of option pricing, prediction accuracy holds greater significance than interpretability. Options are financial instruments used for managing risk and speculation. Even minor pricing errors can result in substantial financial losses. While understanding how the model arrives at a prediction can be valuable (interpretability), it's less critical than the model's ability to deliver a correct option value.

Conclusion

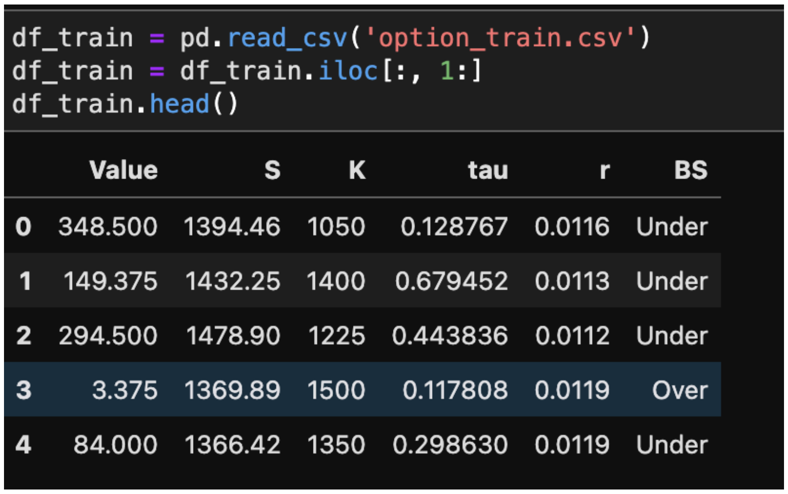
The analysis revealed that machine learning models are a viable alternative to the traditional Black-Scholes formula, particularly when dealing with complex market dynamics. However, there's room for further exploration & optimization:

- **Segmentation and Specialized Models**: Stock options exhibit diverse behaviors based on underlying assets, maturities, and market conditions. Future work could involve segmenting options based on similar characteristics and building specialized predictor models for each segment. This approach could potentially improve overall accuracy by introducing a balance of specificity and generalizability within the segment.
- **Ensemble Models**: Combining the strengths of traditional models, like Black-Scholes, and modern machine learning models could lead to more accurate predictions.
- **Incorporating Market Sentiment**: Option pricing is influenced not only by fundamental factors but also by market sentiment and broader economic trends. Integrating advanced language models capable of analyzing news articles, social media data, and other textual sources could enable more holistic prediction models that account for these intangible factors.

By pursuing these avenues for improvement, we can further refine option pricing models, leading to more informed financial decisions and potentially reducing risk in the options market.

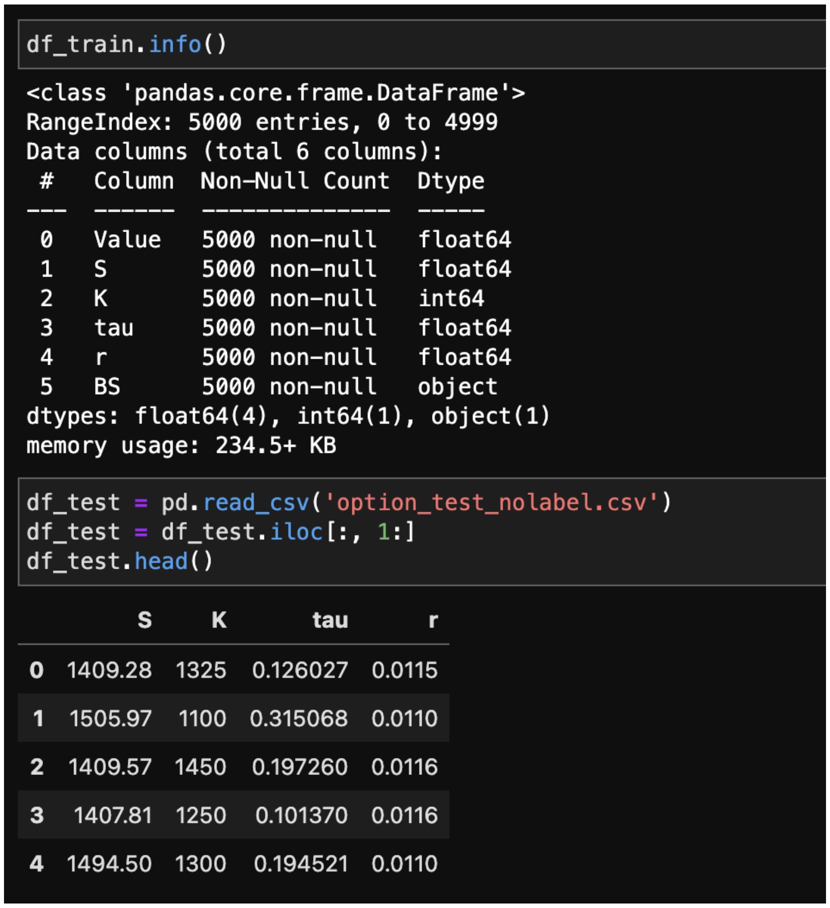
Appendix

Data Snapshot



```
df_train = pd.read_csv('option_train.csv')
df_train = df_train.iloc[:, 1:]
df_train.head()
```

	Value	S	K	tau	r	BS
0	348.500	1394.46	1050	0.128767	0.0116	Under
1	149.375	1432.25	1400	0.679452	0.0113	Under
2	294.500	1478.90	1225	0.443836	0.0112	Under
3	3.375	1369.89	1500	0.117808	0.0119	Over
4	84.000	1366.42	1350	0.298630	0.0119	Under

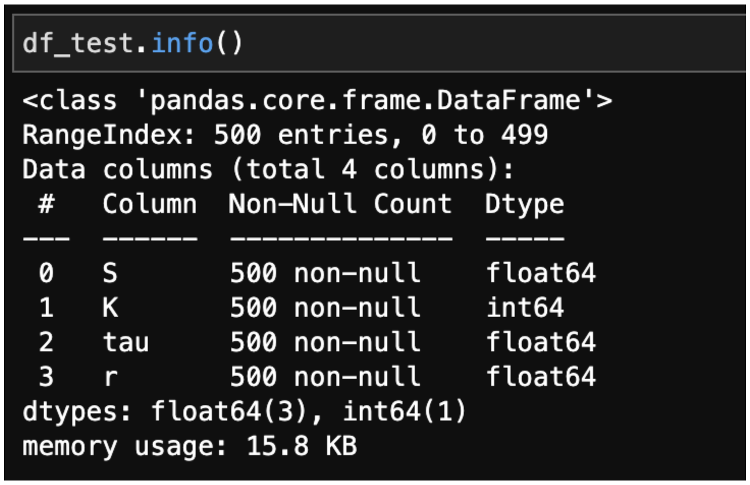


```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5000 entries, 0 to 4999
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Value   5000 non-null   float64
 1   S       5000 non-null   float64
 2   K       5000 non-null   int64  
 3   tau     5000 non-null   float64
 4   r       5000 non-null   float64
 5   BS      5000 non-null   object 
dtypes: float64(4), int64(1), object(1)
memory usage: 234.5+ KB
```

```
df_test = pd.read_csv('option_test_nolabel.csv')
df_test = df_test.iloc[:, 1:]
df_test.head()
```

	S	K	tau	r
0	1409.28	1325	0.126027	0.0115
1	1505.97	1100	0.315068	0.0110
2	1409.57	1450	0.197260	0.0116
3	1407.81	1250	0.101370	0.0116
4	1494.50	1300	0.194521	0.0110



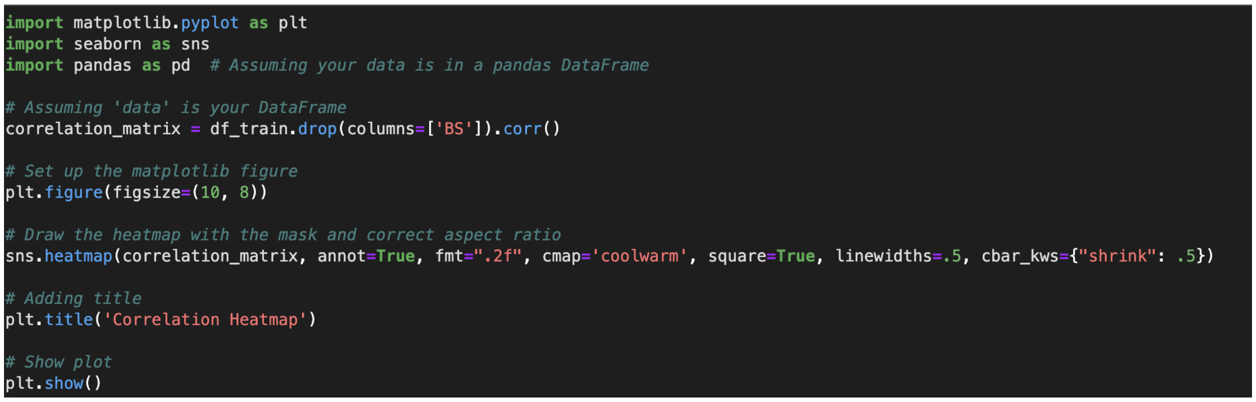
```
df_test.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   S       500 non-null    float64
 1   K       500 non-null    int64  
 2   tau     500 non-null    float64
 3   r       500 non-null    float64
dtypes: float64(3), int64(1)
memory usage: 15.8 KB
```

Exploratory data analysis

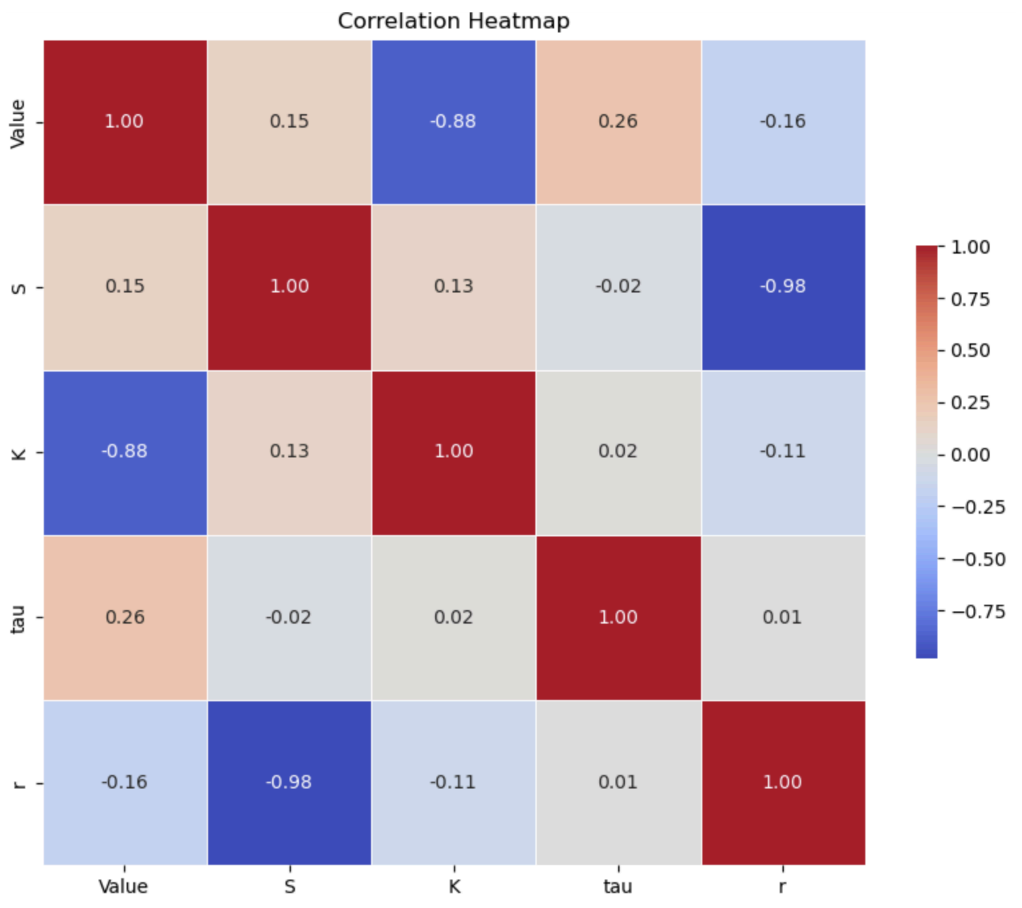
Correlation

From the correlation heatmap, we see that:

- There is high correlation between strike price K and Value, indicating that strike price could be a strong predictor of Value which is our response variable for regression
- There is high correlation between predictor variables annual interest rate r and current asset value S which may cause multicollinearity which could cause bias in our modeling

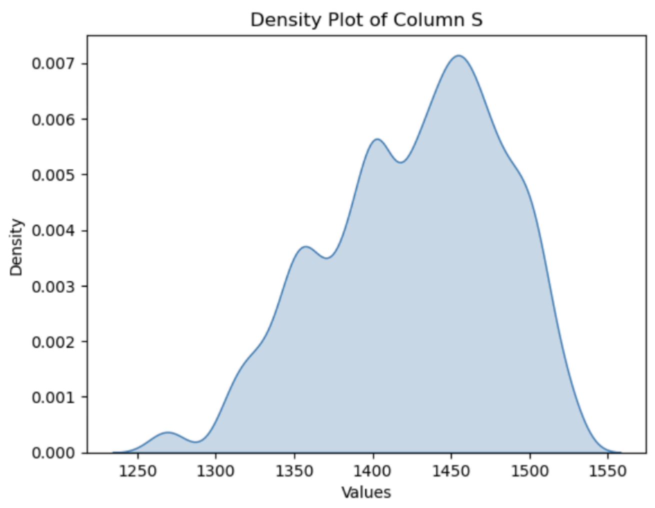


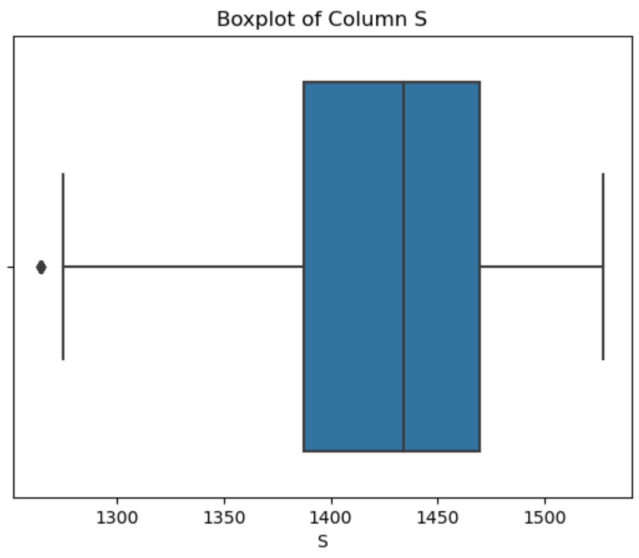
```python
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd  # Assuming your data is in a pandas DataFrame

# Assuming 'data' is your DataFrame
correlation_matrix = df_train.drop(columns=['BS']).corr()

# Set up the matplotlib figure
plt.figure(figsize=(10, 8))

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap='coolwarm', square=True, linewidths=.5, cbar_kws={"shrink": .5})

# Adding title
plt.title('Correlation Heatmap')

# Show plot
plt.show()
```



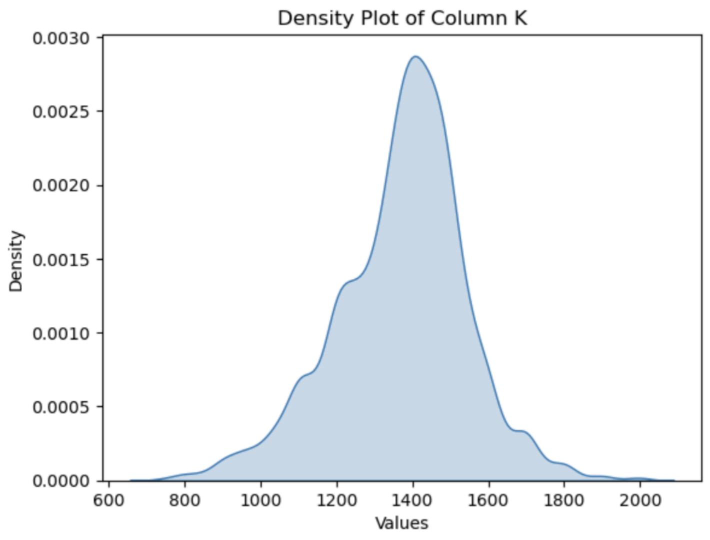
Variable distribution

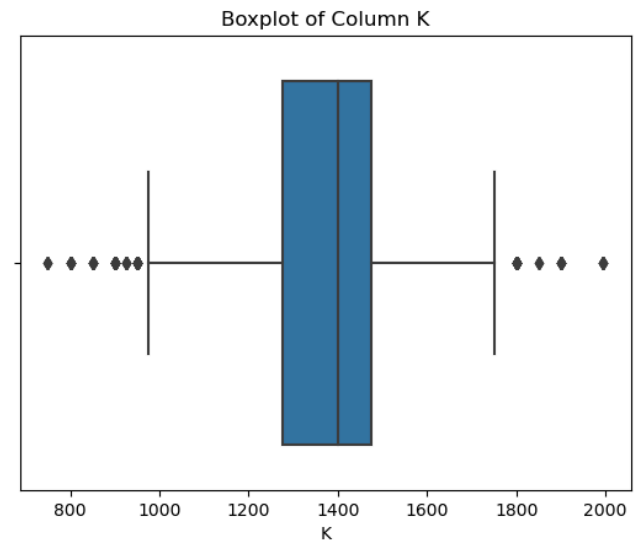
Current Asset Value (S): Current asset value has minimal outliers



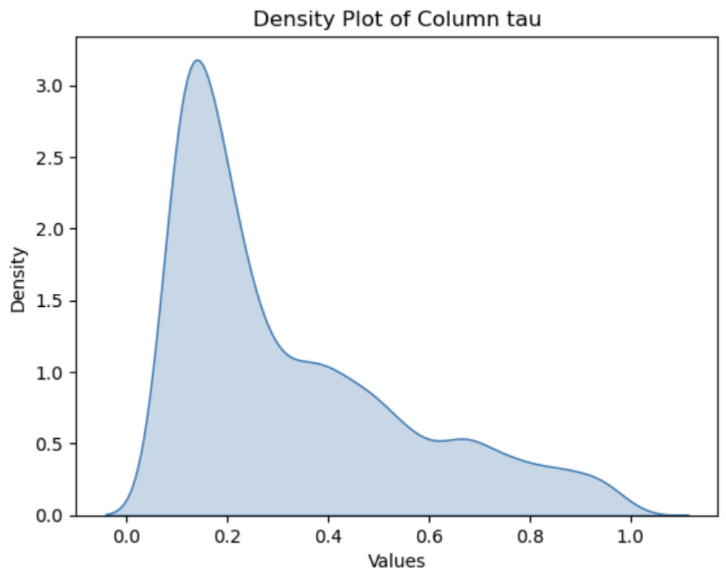


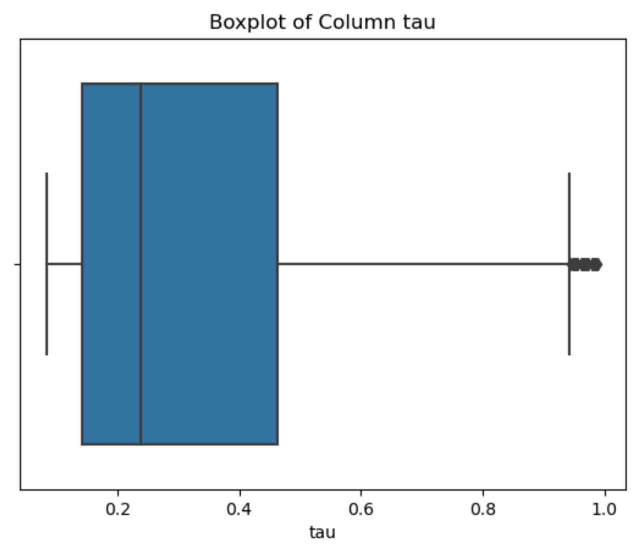
Strike price (K): Strike price is slightly left skewed with a few outliers on both ends



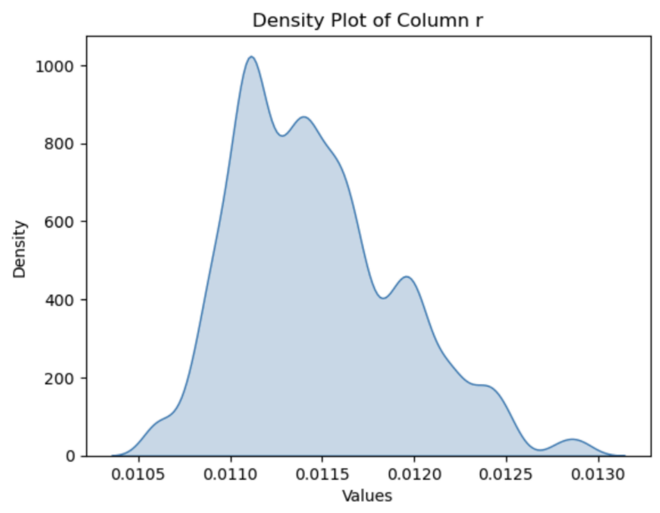


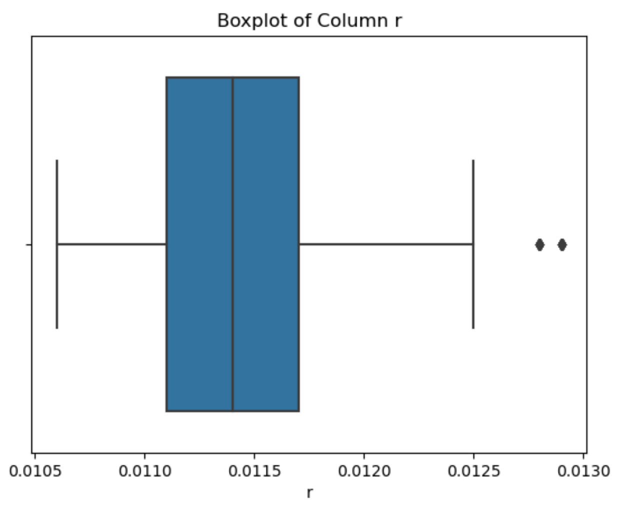
Time to maturity (tau): Time to maturity is right-skewed with a few outliers on the upper end



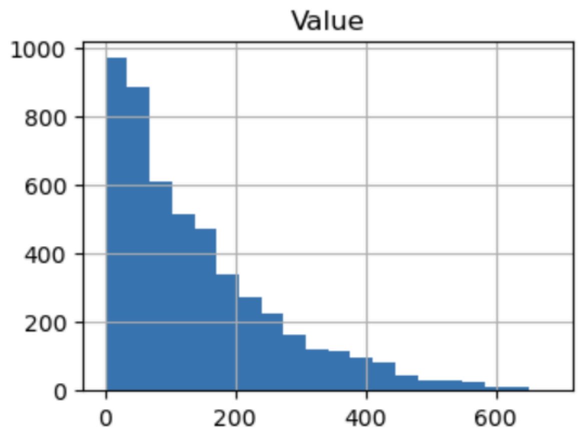


Annual Interest Rate (r): Annual interest rate has no skew with a few outliers on the upper end

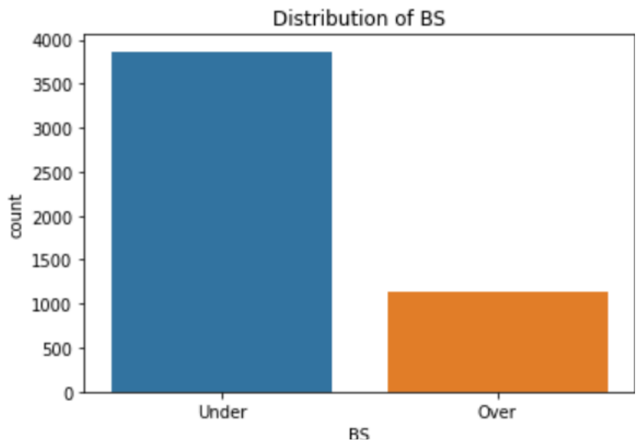




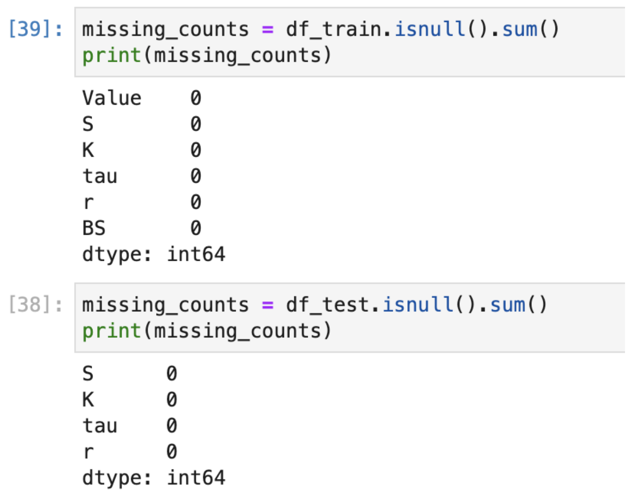
Current option value (Value): The response variable for our regression analysis, current option value is right-skewed



BS: The response variable for our classification analysis has some class imbalance with 77% of options in training data categorized as Under and the remaining 23% categorized as Over.



Data-Preprocessing



```
[39]: missing_counts = df_train.isnull().sum()
      print(missing_counts)

Value    0
S        0
K        0
tau      0
r        0
BS       0
dtype: int64
```

```
[38]: missing_counts = df_test.isnull().sum()
      print(missing_counts)

S      0
K      0
tau    0
r      0
dtype: int64
```

```
[41]: from sklearn.preprocessing import StandardScaler

      scaler = StandardScaler(with_mean=True,
                              with_std=True,
                              copy=True)

      scaler.fit(X)

      X_std = scaler.transform(X)
```

```
[42]: feature_std = pd.DataFrame(X_std, columns=X.columns);

      feature_std.std()
```

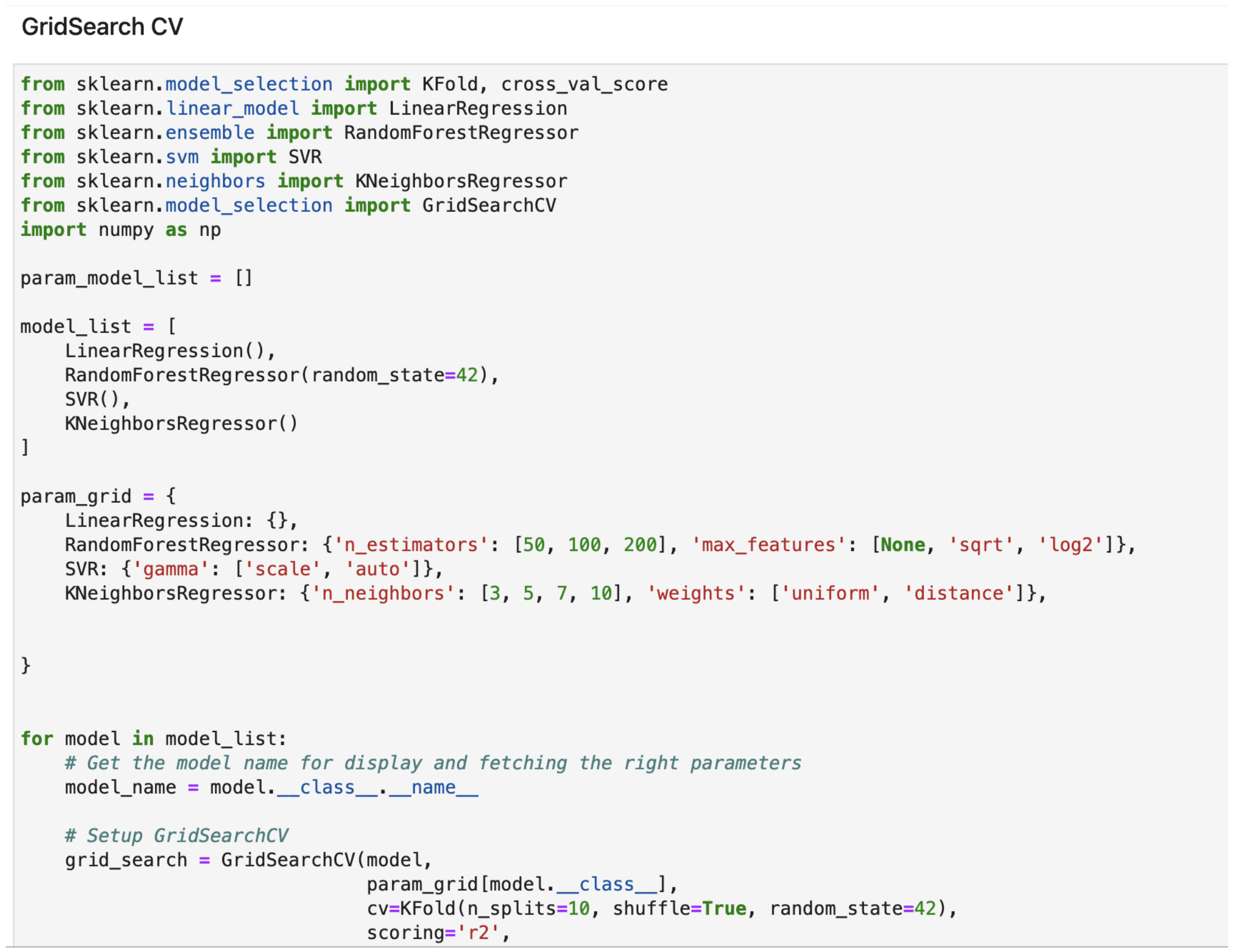
```
[42]: S      1.0001
      K      1.0001
      tau    1.0001
      r      1.0001
      dtype: float64
```

Regression Models

```
: from sklearn.model_selection import KFold, cross_val_score
  from sklearn.linear_model import LinearRegression
  from sklearn.ensemble import RandomForestRegressor
  from sklearn.svm import SVR
  from sklearn.neighbors import KNeighborsRegressor

  model_list = [LinearRegression(),
                RandomForestRegressor(random_state=42),
                SVR(),
                KNeighborsRegressor()
               ]

  for model in model_list:

      # Initialize the linear regression model
      model = model

      # Setup cross-validation (using 5 folds here)
      kf = KFold(n_splits=10, shuffle=True, random_state=42)

      # Perform cross-validation using R-squared as the metric
      r_squared_scores = cross_val_score(model, X, y, cv=kf, scoring='r2')
      mse_scores = cross_val_score(model, X, y, cv=kf, scoring='neg_mean_squared_error')
      rmse_scores = np.sqrt(-mse_scores)

      # Display the R-squared scores for each fold and the mean R-squared
      print(f"Model: {model}")
      print(f"Mean R-squared: {np.mean(r_squared_scores)}")
```

```
Model: LinearRegression()
Mean R-squared: 0.9244138358730758
Model: RandomForestRegressor(random_state=42)
Mean R-squared: 0.9966829364412189
Model: SVR()
Mean R-squared: 0.5915042929412553
Model: KNeighborsRegressor()
Mean R-squared: 0.9482982706182936
```

Regression Models with GridSearch CV for Hyperparameter Tuning



GridSearch CV

```python
from sklearn.model_selection import KFold, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor
from sklearn.svm import SVR
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV
import numpy as np

param_model_list = []

model_list = [
    LinearRegression(),
    RandomForestRegressor(random_state=42),
    SVR(),
    KNeighborsRegressor()
]

param_grid = {
    LinearRegression: {},
    RandomForestRegressor: {'n_estimators': [50, 100, 200], 'max_features': [None, 'sqrt', 'log2']},
    SVR: {'gamma': ['scale', 'auto']},
    KNeighborsRegressor: {'n_neighbors': [3, 5, 7, 10], 'weights': ['uniform', 'distance']},


}


for model in model_list:
    # Get the model name for display and fetching the right parameters
    model_name = model.__class__.__name__

    # Setup GridSearchCV
    grid_search = GridSearchCV(model,
                               param_grid[model.__class__],
                               cv=KFold(n_splits=10, shuffle=True, random_state=42),
                               scoring='r2',
                              verbose=2
                             )

    # Perform grid search
    grid_search.fit(X, y)

    # Display the best parameters and the mean R-squared of the best estimator
    print(f"Model: {model_name}")
    print(f"Best parameters: {grid_search.best_params_}")
    print(f"Best mean R-squared: {grid_search.best_score_}\n")
    param_model_list.append([model_name, grid_search.best_params_, grid_search.best_score_])
```

```
Fitting 10 folds for each of 1 candidates, totalling 10 fits
[CV] END .................................................... total time=   0.0s
[CV] END .................................................... total time=   0.0s
[CV] END .................................................... total time=   0.0s
[CV] END .................................................... total time=   0.0s
[CV] END .................................................... total time=   0.0s
[CV] END .................................................... total time=   0.0s
[CV] END .................................................... total time=   0.0s
[CV] END .................................................... total time=   0.0s
[CV] END .................................................... total time=   0.0s
[CV] END .................................................... total time=   0.0s
Model: LinearRegression
Best parameters: {}
Best mean R-squared: 0.9244138358730758
```

```
Fitting 10 folds for each of 9 candidates, totalling 90 fits
[CV] END .................max_features=None, n_estimators=50; total time=   0.3s
[CV] END .................max_features=None, n_estimators=50; total time=   0.3s
[CV] END .................max_features=None, n_estimators=50; total time=   0.3s
[CV] END .................max_features=None, n_estimators=50; total time=   0.3s
[CV] END .................max_features=None, n_estimators=50; total time=   0.3s
[CV] END .................max_features=None, n_estimators=50; total time=   0.3s
[CV] END .................max_features=None, n_estimators=50; total time=   0.3s
[CV] END .................max_features=None, n_estimators=50; total time=   0.3s
[CV] END .................max_features=None, n_estimators=50; total time=   0.3s
[CV] END .................max_features=None, n_estimators=50; total time=   0.3s
[CV] END ................max_features=None, n_estimators=100; total time=   0.5s
[CV] END ................max_features=None, n_estimators=100; total time=   0.5s
[CV] END ................max_features=None, n_estimators=100; total time=   0.5s
[CV] END ................max_features=None, n_estimators=100; total time=   0.5s
[CV] END ................max_features=None, n_estimators=100; total time=   0.5s
[CV] END ................max_features=None, n_estimators=100; total time=   0.5s
[CV] END ................max_features=None, n_estimators=100; total time=   0.5s
[CV] END ................max_features=None, n_estimators=100; total time=   0.5s
[CV] END ................max_features=None, n_estimators=100; total time=   0.5s
[CV] END ................max_features=None, n_estimators=100; total time=   0.5s
[CV] END ................max_features=None, n_estimators=200; total time=   1.2s
[CV] END ................max_features=None, n_estimators=200; total time=   1.1s
[CV] END ................max_features=None, n_estimators=200; total time=   1.2s
[CV] END ................max_features=None, n_estimators=200; total time=   1.1s
[CV] END ................max_features=None, n_estimators=200; total time=   1.1s
[CV] END ................max_features=None, n_estimators=200; total time=   1.0s
[CV] END ................max_features=None, n_estimators=200; total time=   1.0s
[CV] END ................max_features=None, n_estimators=200; total time=   1.0s
[CV] END ................max_features=None, n_estimators=200; total time=   1.1s
[CV] END ................max_features=None, n_estimators=200; total time=   1.0s
[CV] END .................max_features=sqrt, n_estimators=50; total time=   0.2s
[CV] END .................max_features=sqrt, n_estimators=50; total time=   0.2s
[CV] END .................max_features=sqrt, n_estimators=50; total time=   0.2s
[CV] END .................max_features=sqrt, n_estimators=50; total time=   0.2s
[CV] END .................max_features=sqrt, n_estimators=50; total time=   0.2s
[CV] END .................max_features=sqrt, n_estimators=50; total time=   0.2s
[CV] END .................max_features=sqrt, n_estimators=50; total time=   0.2s
```

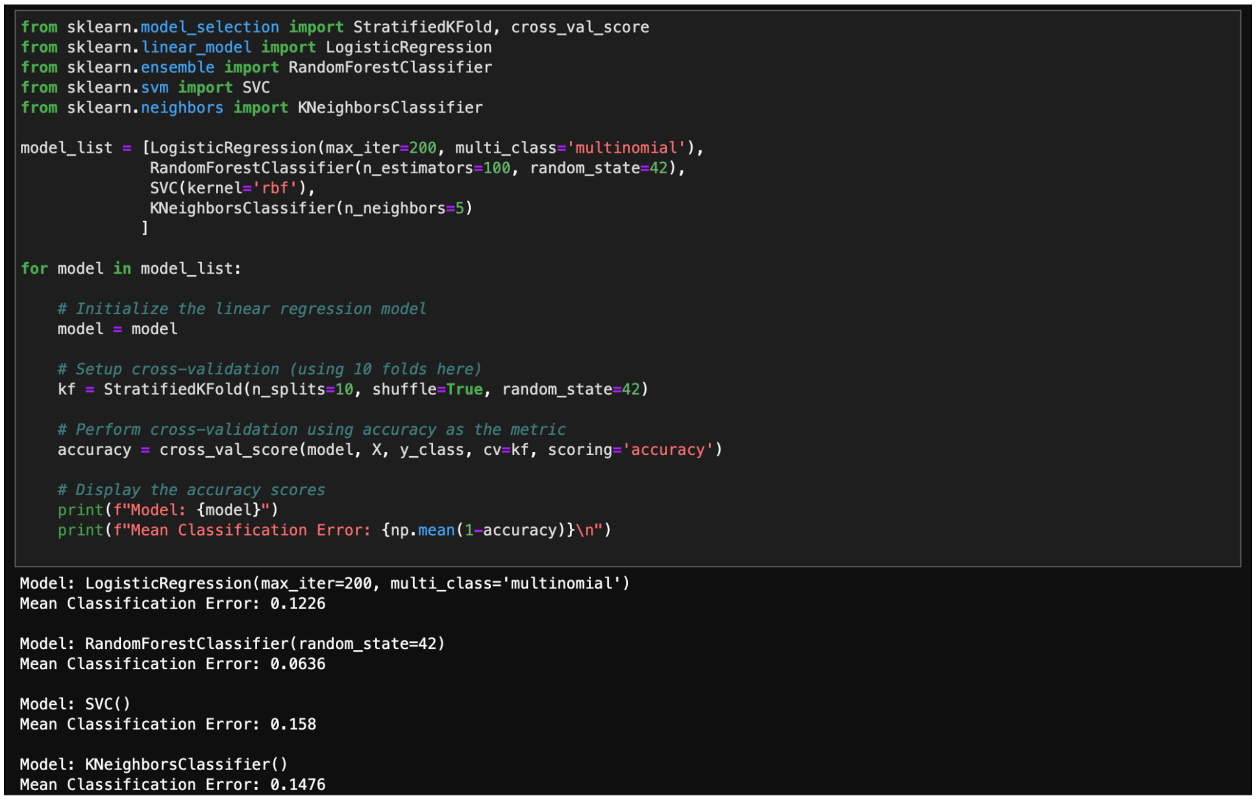
```
[CV] END ................max_features=sqrt, n_estimators=200; total time=   0.6s
[CV] END ................max_features=sqrt, n_estimators=200; total time=   0.6s
[CV] END ................max_features=sqrt, n_estimators=200; total time=   0.6s
[CV] END ................max_features=sqrt, n_estimators=200; total time=   0.6s
[CV] END .................max_features=log2, n_estimators=50; total time=   0.2s
[CV] END .................max_features=log2, n_estimators=50; total time=   0.2s
[CV] END .................max_features=log2, n_estimators=50; total time=   0.2s
[CV] END .................max_features=log2, n_estimators=50; total time=   0.2s
[CV] END .................max_features=log2, n_estimators=50; total time=   0.2s
[CV] END .................max_features=log2, n_estimators=50; total time=   0.2s
[CV] END .................max_features=log2, n_estimators=50; total time=   0.2s
[CV] END .................max_features=log2, n_estimators=50; total time=   0.2s
[CV] END .................max_features=log2, n_estimators=50; total time=   0.2s
[CV] END .................max_features=log2, n_estimators=50; total time=   0.2s
[CV] END ................max_features=log2, n_estimators=100; total time=   0.3s
[CV] END ................max_features=log2, n_estimators=100; total time=   0.3s
[CV] END ................max_features=log2, n_estimators=100; total time=   0.3s
[CV] END ................max_features=log2, n_estimators=100; total time=   0.3s
[CV] END ................max_features=log2, n_estimators=100; total time=   0.3s
[CV] END ................max_features=log2, n_estimators=100; total time=   0.3s
[CV] END ................max_features=log2, n_estimators=100; total time=   0.3s
[CV] END ................max_features=log2, n_estimators=100; total time=   0.3s
[CV] END ................max_features=log2, n_estimators=100; total time=   0.3s
[CV] END ................max_features=log2, n_estimators=100; total time=   0.4s
[CV] END ................max_features=log2, n_estimators=200; total time=   0.6s
[CV] END ................max_features=log2, n_estimators=200; total time=   0.7s
[CV] END ................max_features=log2, n_estimators=200; total time=   0.7s
[CV] END ................max_features=log2, n_estimators=200; total time=   0.6s
[CV] END ................max_features=log2, n_estimators=200; total time=   0.6s
[CV] END ................max_features=log2, n_estimators=200; total time=   0.6s
[CV] END ................max_features=log2, n_estimators=200; total time=   0.6s
[CV] END ................max_features=log2, n_estimators=200; total time=   0.6s
[CV] END ................max_features=log2, n_estimators=200; total time=   0.6s
[CV] END ................max_features=log2, n_estimators=200; total time=   0.6s
Model: RandomForestRegressor
Best parameters: {'max_features': None, 'n_estimators': 200}
Best mean R-squared: 0.9967113888381011
```

```
Fitting 10 folds for each of 2 candidates, totalling 20 fits
[CV] END ........................................gamma=scale; total time=   0.4s
[CV] END ........................................gamma=scale; total time=   0.4s
[CV] END ........................................gamma=scale; total time=   0.4s
[CV] END ........................................gamma=scale; total time=   0.4s
[CV] END ........................................gamma=scale; total time=   0.4s
[CV] END ........................................gamma=scale; total time=   0.4s
[CV] END ........................................gamma=scale; total time=   0.6s
[CV] END ........................................gamma=scale; total time=   0.4s
[CV] END ........................................gamma=scale; total time=   0.5s
[CV] END ........................................gamma=scale; total time=   0.4s
[CV] END .........................................gamma=auto; total time=   0.4s
[CV] END .........................................gamma=auto; total time=   0.3s
[CV] END .........................................gamma=auto; total time=   0.3s
[CV] END .........................................gamma=auto; total time=   0.3s
[CV] END .........................................gamma=auto; total time=   0.3s
[CV] END .........................................gamma=auto; total time=   0.3s
[CV] END .........................................gamma=auto; total time=   0.3s
[CV] END .........................................gamma=auto; total time=   0.3s
[CV] END .........................................gamma=auto; total time=   0.3s
[CV] END .........................................gamma=auto; total time=   0.3s
Model: SVR
Best parameters: {'gamma': 'scale'}
Best mean R-squared: 0.5915042929412553
```

```
Fitting 10 folds for each of 8 candidates, totalling 80 fits
[CV] END .....................n_neighbors=3, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=3, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=3, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=3, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=3, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=3, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=3, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=3, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=3, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=3, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=3, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=3, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=3, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=3, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=3, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=3, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=3, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=3, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=3, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=3, weights=distance; total time=   0.0s
[CV] END .....................n_neighbors=5, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=5, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=5, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=5, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=5, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=5, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=5, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=5, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=5, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=5, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=5, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=5, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=5, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=5, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=5, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=5, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=5, weights=distance; total time=   0.0s
```

```
[CV] END .....................n_neighbors=7, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=7, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=7, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=7, weights=uniform; total time=   0.0s
[CV] END .....................n_neighbors=7, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=7, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=7, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=7, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=7, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=7, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=7, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=7, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=7, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=7, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=7, weights=distance; total time=   0.0s
[CV] END ....................n_neighbors=10, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=10, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=10, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=10, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=10, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=10, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=10, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=10, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=10, weights=uniform; total time=   0.0s
[CV] END ....................n_neighbors=10, weights=uniform; total time=   0.0s
[CV] END ...................n_neighbors=10, weights=distance; total time=   0.0s
[CV] END ...................n_neighbors=10, weights=distance; total time=   0.0s
[CV] END ...................n_neighbors=10, weights=distance; total time=   0.0s
[CV] END ...................n_neighbors=10, weights=distance; total time=   0.0s
[CV] END ...................n_neighbors=10, weights=distance; total time=   0.0s
[CV] END ...................n_neighbors=10, weights=distance; total time=   0.0s
[CV] END ...................n_neighbors=10, weights=distance; total time=   0.0s
[CV] END ...................n_neighbors=10, weights=distance; total time=   0.0s
[CV] END ...................n_neighbors=10, weights=distance; total time=   0.0s
[CV] END ...................n_neighbors=10, weights=distance; total time=   0.0s
Model: KNeighborsRegressor
Best parameters: {'n_neighbors': 10, 'weights': 'distance'}
Best mean R-squared: 0.9614580739723302
```

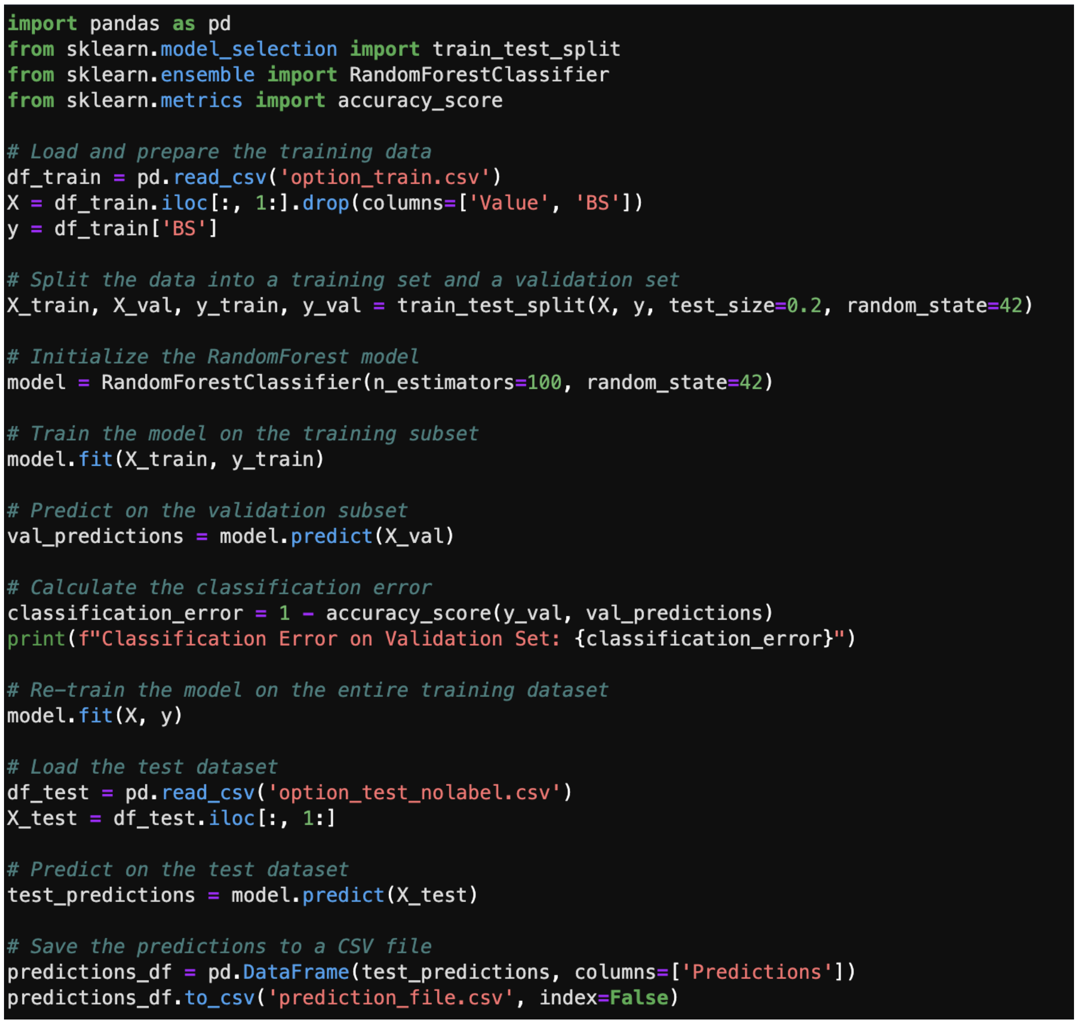
Classification Models and Results



```
from sklearn.model_selection import StratifiedKFold, cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier

model_list = [LogisticRegression(max_iter=200, multi_class='multinomial'),
              RandomForestClassifier(n_estimators=100, random_state=42),
              SVC(kernel='rbf'),
              KNeighborsClassifier(n_neighbors=5)
             ]

for model in model_list:

    # Initialize the linear regression model
    model = model

    # Setup cross-validation (using 10 folds here)
    kf = StratifiedKFold(n_splits=10, shuffle=True, random_state=42)

    # Perform cross-validation using accuracy as the metric
    accuracy = cross_val_score(model, X, y_class, cv=kf, scoring='accuracy')

    # Display the accuracy scores
    print(f"Model: {model}")
    print(f"Mean Classification Error: {np.mean(1-accuracy)}\n")
```

```
Model: LogisticRegression(max_iter=200, multi_class='multinomial')
Mean Classification Error: 0.1226

Model: RandomForestClassifier(random_state=42)
Mean Classification Error: 0.0636

Model: SVC()
Mean Classification Error: 0.158

Model: KNeighborsClassifier()
Mean Classification Error: 0.1476
```

Final Model - Random Forest Classifier



```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Load and prepare the training data
df_train = pd.read_csv('option_train.csv')
X = df_train.iloc[:, 1:].drop(columns=['Value', 'BS'])
y = df_train['BS']

# Split the data into a training set and a validation set
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Initialize the RandomForest model
model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model on the training subset
model.fit(X_train, y_train)

# Predict on the validation subset
val_predictions = model.predict(X_val)

# Calculate the classification error
classification_error = 1 - accuracy_score(y_val, val_predictions)
print(f"Classification Error on Validation Set: {classification_error}")

# Re-train the model on the entire training dataset
model.fit(X, y)

# Load the test dataset
df_test = pd.read_csv('option_test_nolabel.csv')
X_test = df_test.iloc[:, 1:]

# Predict on the test dataset
test_predictions = model.predict(X_test)

# Save the predictions to a CSV file
predictions_df = pd.DataFrame(test_predictions, columns=['Predictions'])
predictions_df.to_csv('prediction_file.csv', index=False)
```

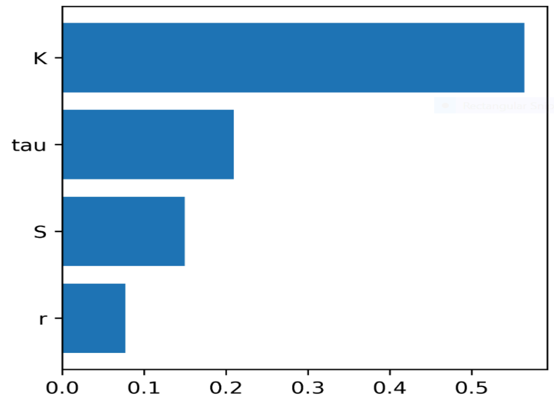


Figure 3. Random Forest Classifier - Feature Importance