DSO 530

EUROPEAN CALL OPTIONS PRICING PROJECT

GROUP 32:
CHINMAYI BENGALURU PRAKASH
FALAK JAIN
SHREE VIDYA RAVI KUMAR

1. EXECUTIVE SUMMARY

A European call option gives the investor the right to acquire an underlying asset at a particular strike price and at a particular time i.e., at expiry time. An investor can earn a profit from the call option when the stock's price at expiry is traded high above the strike price so that the option premium's cost is covered. However, valuing an option is precarious as it depends on the future value i.e., at maturity of the underlying asset. On the other hand, a European put option allows the investor to sell the underlying asset at a particular time. An investor can earn a profit from the put option when the stock's price at expiry is traded far below the strike price so that the option premium's cost is covered.

The Black-Scholes call option formula provides an approach for this and is often used to value European options. The formula is calculated by multiplying the stock price by the cumulative standard normal probability distribution function. This formula works exceptionally well in practice and is used by investors even today.

In this project, the goal was to build various machine learning models to perform the same task as the Black-Scholes call option formula. An attempt was made to predict current call option values, *Value*, using regression and whether the call option was undervalued or overvalued, *BS*, using classification techniques. Thus, the best machine learning model that produced the highest coefficient of determination and the least classification error respectively were finalized to predict these values on the test dataset which did not contain these target variables.

This report describes in detail the processes and methodologies used to predict *Value(C)* and *BS* incorporating supervised machine learning techniques. The raw dataset consists of 1,680 records and 6 columns. Two records with missing values were removed from the dataset during the data cleaning process. The remaining data was split into train and test datasets in 80:20 ratio.

As part of feature engineering and scaling, 7 and 3 additional features were created for regression and classification problems respectively and the data was scaled using a min-max scaler. Multiple Linear and Logistic Regression models were used as the baseline models respectively for regression and classification parts respectively. Above these, KNN, Decision Tree, Random Forest and Neural Networks models were executed to predict the *Value(C)* and *BS*. These models were then compared for out-of-sample R-squared values and classification errors respectively for regression and classification respectively. Additionally, cross-validation was performed for the classification part of the project to improve model's accuracy.

To conclude, for regression, the Random Forest Regressor was chosen among all the models as it produced an outstanding out-of-sample R-squared value of 99.6%, being the largest of all, to predict *Value* and, for classification, the Random Forest Classifier model was finalized as it produced the least classification error of 4.46% and the highest accuracy of 95.54%.

2. DATA CLEANING

Clean data allows one to predict call option values, *Value,* and the probability of the option being overvalued or undervalued, *BS*, using regression and classification models respectively. Therefore, the first step in model building is data cleaning.

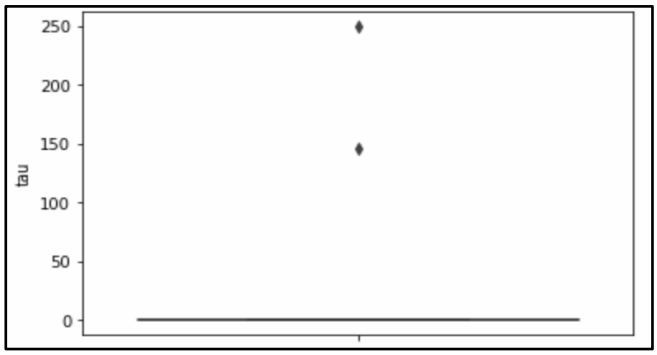


Figure 1.1: Box plot for tau

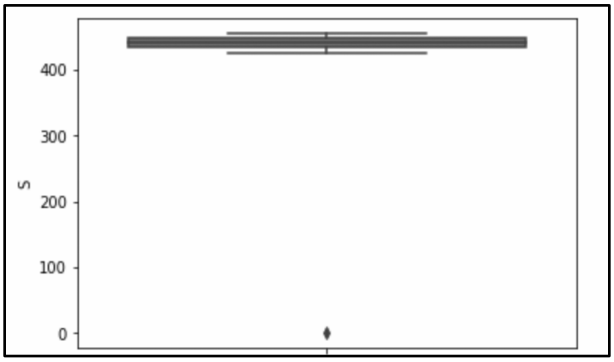


Figure 1.2: Box plot for S

From the boxplots above, it is seen that there are some options with a time to maturity of 100 years and a $0 asset value, which is highly unlikely in business sense. Therefore, these records were dropped from the data. Two additional records containing null values were also dropped. Since we have 1,680 records, these measures only get rid of 5 rows. This did not affect the effectiveness of our machine learning models adversely.

Certain outliers also existed in the interest rate field, but they fell within the limits of common business sense and were therefore retained in the data. After dropping all the records of concern, there were 1675 records with no missing or absurd values. This forms a reliable input data for feature engineering and machine learning models development.

3. REVIEW OF APPROACHES

3.1 REGRESSION ANALYSIS : Call option value can be predicted by various regression models. As part of feature engineering and feature scaling, all default features and interaction terms among *strike rate (K)*, *maturity time (tau)*, *risk free rate (r)* and *asset value (S)* were included for modeling and features were scaled using min-max scalar to have the same range of values. The features were used in multivariate regression , K-nearest neighbors regression, Decision tree regression and Random Forest with coefficient of determination as the evaluation criterion.

Model	**In-Sample r-sq**	**Out-of-sample r-sq**
Multivariate Regression	99.5%	99.5%
K-Nearest Neighbors	98.6%	98.3%
Decision Tree	99.5%	98.7%
Random Forest	99.9%	99.6%

Table 2.1: Model Exploration

R-square was the metric considered for evaluating model performance. From the above table, the model with highest r-square was the best performing model and random forest wins.

3.2 CLASSIFICATION ANALYSIS : The probability of call option to be undervalued or overvalued can be predicted using various classification models. As a part of feature engineering and feature scaling, all default features were included and scaled using min-max scalar to have the same range of values between 0 and 1. These features were used in Logistic regression, K-nearest neighbors classifier, Decision tree classifier, Random Forest classifier and Neural networks with accuracy and classification error as the evaluation criteria. To increase the accuracy, a cross-validation approach was implemented.

Model	**Accuracy**	**Classification error**
Logistic regression	93.45%	6.55%
K-Nearest Neighbors	92.26%	7.74%
Decision Tree	91.67%	8.33%
Random Forest	94.64%	5.36%
Neural Network	91.96%	8.04%

Table 2.2: Model Exploration

The model with high accuracy and low classification error rate achieves best prediction and thus, Random Forest classifier wins.

4. SUMMARY OF FINAL APPROACHES

The best models amongst the ones evaluated to determine our final predictions were utilized for further analysis. Regression was used to determine the option *Value* and classification was utilized to determine if a given option was under or overvalued. Below is a detailed summary of the final approaches for predicting *Value* and *BS*.

4.1 REGRESSION:

Using R-square as the evaluation metric, the random forest model was determined to be the best. This conclusion was reached using the in and out of sample R-square on a 80-20 train test split of the training set provided. After data cleaning, feature engineering was conducted to enrich the data beyond the four input terms provided. Every combination of two interaction terms and higher order terms up to a degree of 2 were computed. A linear regression model was used to determine the significance of each of the input variables and the insignificant terms were dropped. The following input variables apart from the default variables were included in the training of the random forest model:

- *S*S (higher order term)*
- *S*K (interaction term)*

- *S*r*
- *S*tau*
- *tau*tau*
- *K*K*
- *K*r*

The reasons for opting the random forest regressor were the following:
- Best in and out-of-sample R-square amongst the models considered
- Random forest models are robust to outliers and can therefore withstand erroneous data
- Random forest regressors can handle large models effectively
- It considers a subset of features at nodes to create splits ensuring high accuracy

In the model, a 80-20 split of training and testing has been used to train the model on 80% of the data and validate the predictions on the remaining 20%. This approach prevents overfitting of the data and ensures that a reliable balance is struck between bias and variance.

After training the random forest regressor model, an in-sample R-square of 99.9% and an out-of-sample R-square of 99.6% was obtained. The effectiveness of the machine learning model as compared to the Black-Scholes model is explained in the business understanding questions mentioned in the conclusion.

4.2 CLASSIFICATION:

Random Forest Classifier yielded the lowest classification error and the highest accuracy compared to other models. This was based on training the model on 80% of the given training set data and using the remaining 20% to test the model.

As part of feature engineering, the following additional variables were created as these variable combinations have the highest correlation with the target variable BS.
- *S*r*
- *K*tau*
- *K*r*

Further, the random forest classifier model was explored to enhance its performance by including and excluding additional variables and performing 10-fold cross validation in each case. The following results were observed,

Performance Metrics	With additional variables	Without additional variables
With cross validation	Accuracy Score = 95.24% Classification error = 4.76%	Accuracy Score = 95.54% Classification error = 4.46%
Without cross validation	Accuracy Score = 94.94% Classification error = 5.06%	Accuracy Score = 95.64% Classification error = 5.36%

Table 3.2a: Model Exploration

From the above table, it is clear that, when 10-fold cross validation is performed by considering no additional variables, the least classification error was achieved.

After performing cross validation, the best set of hyperparameters were obtained which tuned our model and resulted with the highest accuracy. The hyperparameters were:

- n_estimators = 500
- max_features = 'auto'
- max_depth = 8
- criterion = 'gini'

Thus, this random forest classifier model was finalized to predict the target variable i.e., to determine if an option value was undervalued or overvalued. As part of model interpretation, the confusion matrix was plotted as shown below and the following were calculated:

Type I error = False Positive Rate	**3%**
Type II error	**6%**
Specificity	**97%**
Sensitivity = Recall = Power = True Positive Rate	**94%**
Precision	**95.92%**

Table 3.2b: Model Interpretation

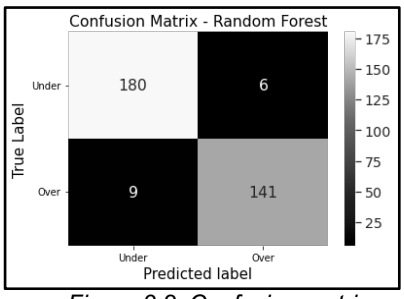


Figure 3.2: Confusion matrix

5. CONCLUSION

The entirety of the model prediction was based on the final call option value and the probability of the call option being overvalued or undervalued. Hence, the importance was given to how the model performs on the test data, based on the training data. This allowed one to prioritize the evaluation metrics such as coefficient of determination for regression problems and accuracy or classification error for classifications regardless of the type of predictors.

Apart from this, call option investment strategies based on Black-Scholes formula are highly theoretical and volatile. Over time, the dynamics of option values became complex and deeper technical knowledge was required to predict the values. In contrast to this, data-driven decisions using machine learning models provided empirical evidence for predictions which led to better investment decisions.

Machine learning models, mainly the supervised learning methods, always use historical data to predict future decisions. The call options depend on *strike rate, annual interest rate, underlying asset values* and *maturity time* to make a holistic decision. Hence, it is important to consider these predictors to understand the predicted values of a call option.

Finally, to understand how the trained model can be used on various other stocks such as Tesla, it is important to understand the data. It can only be said that the model performs well on other stock options if test data is like training data with respect to the evaluation metrics unless otherwise used by advanced techniques such as transfer learning.

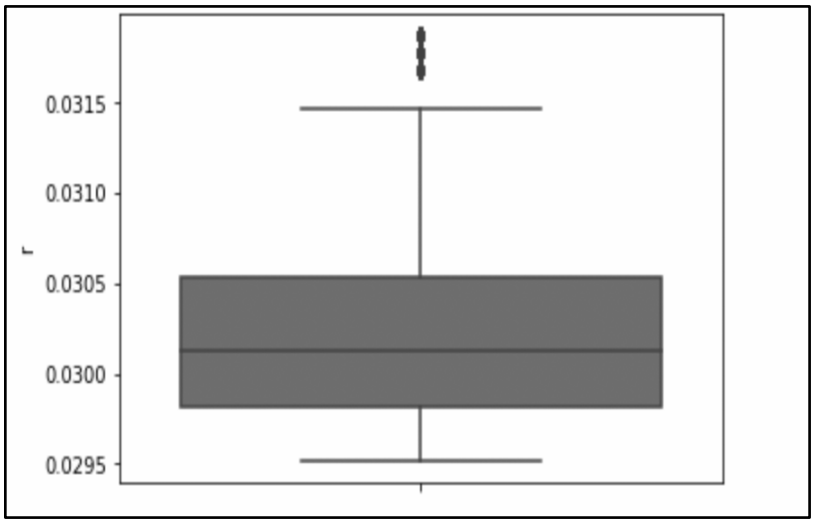
6. APPENDIX & REFERENCES:

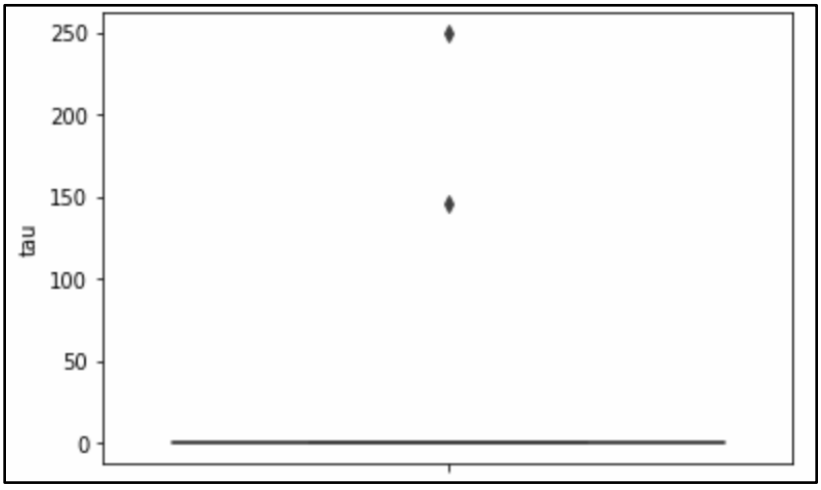
6.1: Data

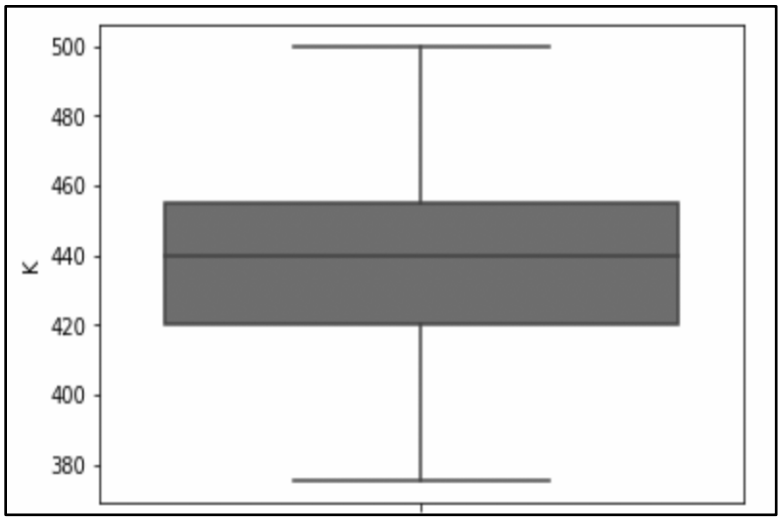
The dataset consists of the following 6 features

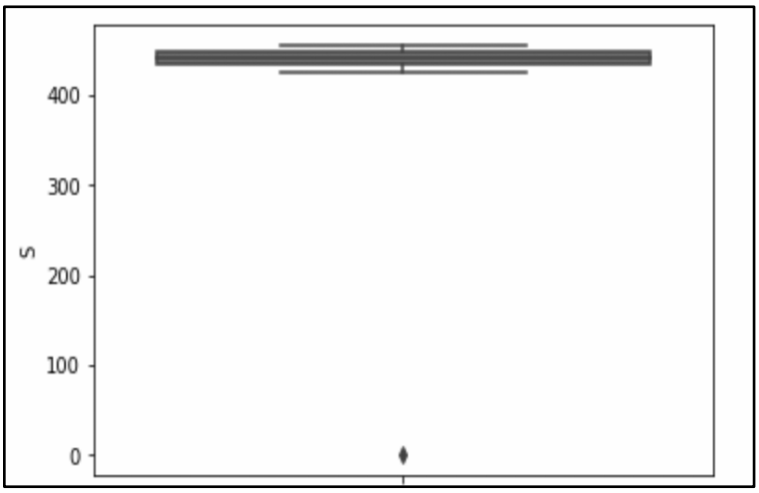
Value (C): Current option value
S: Current asset value
K: Strike price of option
r: Annual interest rate
tau: Time to maturity (in years)
BS: The Black-Scholes formula; Over - the prediction over

6.2: Distribution of input variables

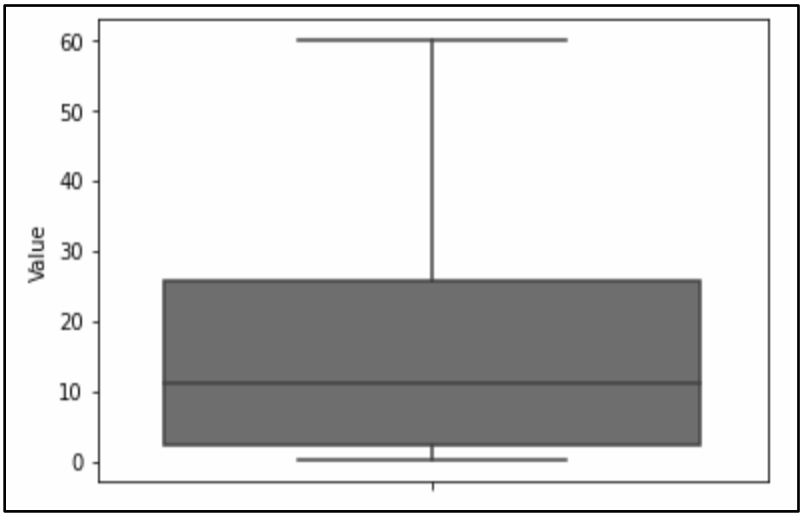


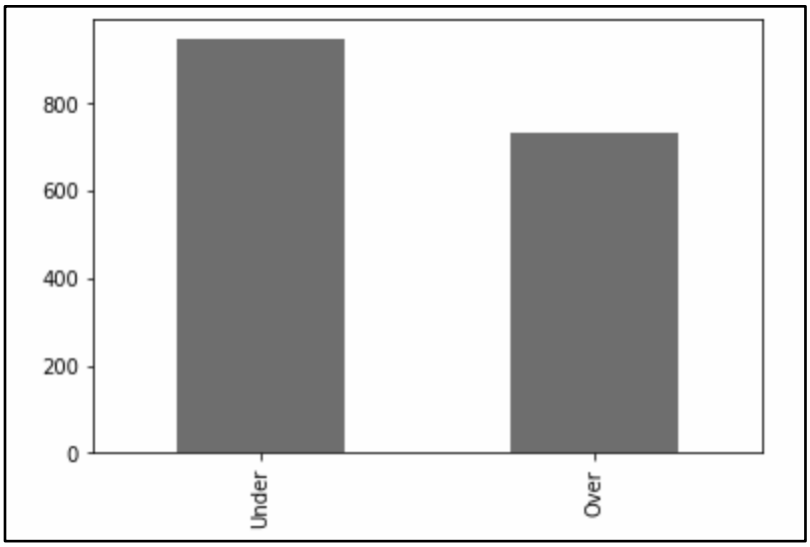






6.3: Distribution of response variables

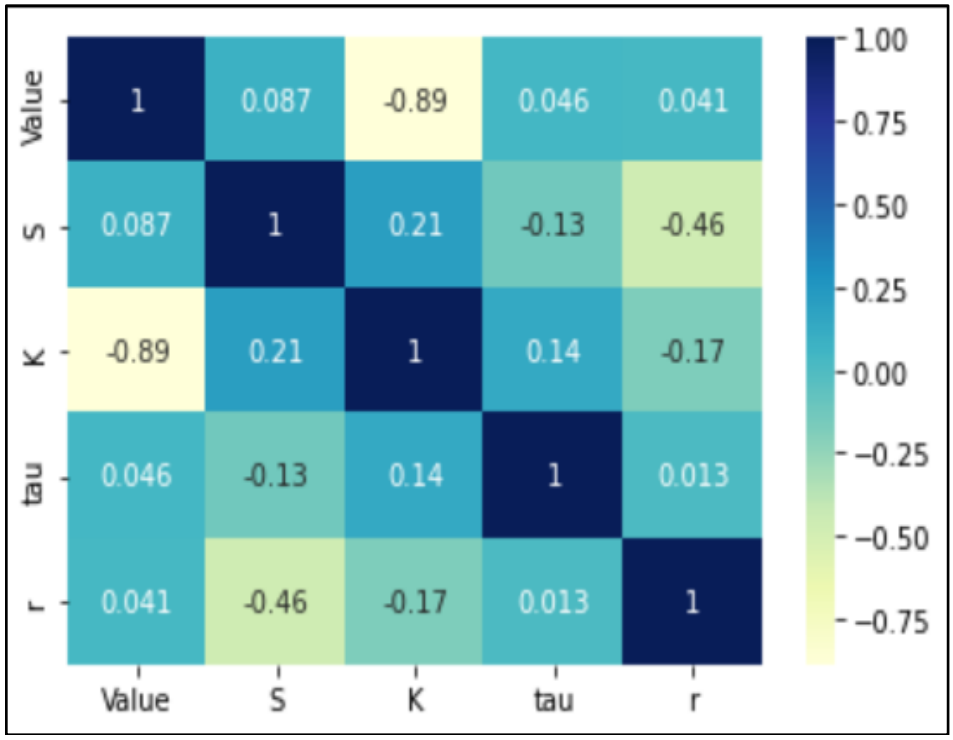




6.4: Data Description

```
RangeIndex: 1680 entries, 0 to 167
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Value   1679 non-null   float
 1   BS      1680 non-null   objec
 2   S       1679 non-null   float
 3   K       1678 non-null   float
 4   tau     1679 non-null   float
 5   r       1680 non-null   float
```

6.5: Correlation plot of input variables



6.6: Value Screening using JMP

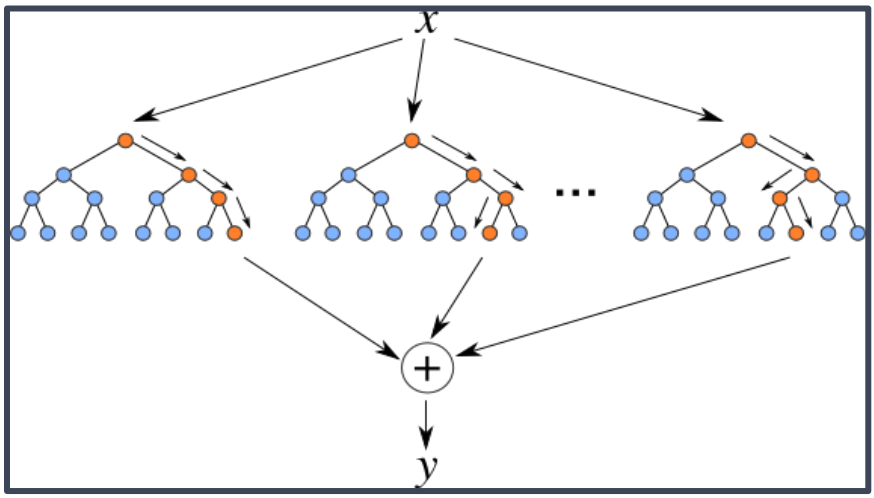
Term	Contrast		Lenth t-Ratio	Individual p-Value	Simultaneous p-Value
K	-12.4333		-507.27	<.0001*	<.0001*
S	2.2597		92.20	<.0001*	<.0001*
r	-1.0916		-44.54	<.0001*	<.0001*
tau	0.0505		2.06	0.0399*	1.0000
K*K	3.7686	*	153.76	<.0001*	<.0001*
K*S	-3.7078	*	-151.28	<.0001*	<.0001*
S*S	1.4727	*	60.08	<.0001*	<.0001*
K*r	0.2050	*	8.36	<.0001*	<.0001*
S*r	0.0632	*	2.58	0.0102*	1.0000
r*r	-0.2122	*	-8.66	<.0001*	<.0001*
K*tau	-0.0764	*	-3.12	0.0026*	0.9418
S*tau	0.5955	*	24.30	<.0001*	<.0001*
r*tau	1.0724	*	43.75	<.0001*	<.0001*
tau*tau	-0.7637	*	-31.16	<.0001*	<.0001*
K*K*K	0.3458	*	14.11	<.0001*	<.0001*
K*K*S	-0.2596	*	-10.59	<.0001*	<.0001*
K*S*S	-0.1639	*	-6.69	<.0001*	<.0001*
S*S*S	0.2672	*	10.90	<.0001*	<.0001*
K*K*r	-0.2523	*	-10.29	<.0001*	<.0001*
K*S*r	0.1161	*	4.74	<.0001*	0.0053*
S*S*r	0.1604	*	6.54	<.0001*	<.0001*
K*r*r	-0.0411	*	-1.68	0.0919	1.0000
S*r*r	0.1418	*	5.78	<.0001*	<.0001*
r*r*r	0.1210	*	4.94	<.0001*	0.0016*
K*tau*tau	1.5495	*	63.22	<.0001*	<.0001*

6.7: 80-20 Train Test Split

```
# Create test and train data
x_train, x_test, y_train, y_test = train_test_split(data_df[data_df.columns[2:]], data_df['Value'], test_size = 0.2, random_state = 123)
```

6.8: Regression

Random Forest Regressor



Random Forest Regressor Code

```
# Create a base model
rf_model = RandomForestRegressor()
# Instantiate the grid search model
#gs = GridSearchCV(estimator = rf_model,cv = 5, param_grid = {'bootstrap': [True],},)
# Fitting the model to the training data
rf_best = rf_model.fit(x_train, y_train)
# Creating predictions on test data for out-of-sample r-sq
y_pred = rf_best.predict(x_test)
print(f'Out of Sample R-square: {round(r2_score(y_test,y_pred),3)}')
# Creating predictions on training data for in-sample r-sq
y_pred = rf_best.predict(x_train)
print(f'In Sample R-square: {round(r2_score(y_train,y_pred),3)}')
```

```
Out of Sample R-square: 0.996
In Sample R-square: 0.999
```

Multiple Linear Regression with Feature Engineering (Interaction Terms), Insignificant Variables Removed and K-Fold Cross Validation

```
# Multiple linear regression with feature engineering (interaction terms with 2 variables), insignificant variables removed and cross validation
# Printing out of sample r-square
lr_model = LinearRegression()
parameters = {'normalize':[True,False]}
lm = GridSearchCV(lr_model, parameters, refit=True, cv=5)
best_model = lm.fit(x_train,y_train)
y_pred = lm.predict(x_test)
print(f'Out of Sample R-square: {round(r2_score(y_test,y_pred),3)}')
y_pred = lm.predict(x_train)
print(f'In Sample R-square: {round(r2_score(y_train,y_pred),3)}')

# r-square remains the same after using cross validation
```

```
Out of Sample R-square: 0.995
```

Decision Tree Regressor - with hyperparameter tuning and k fold validation

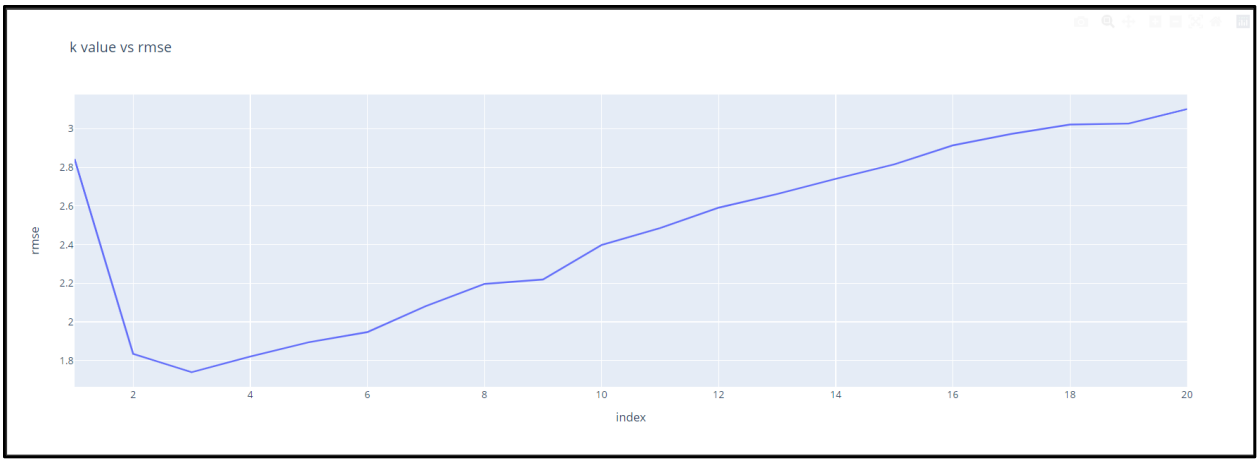
```
# Create base model
dt_model = DecisionTreeRegressor()
# Initiate grid search for hyperparameter tuning and k-fold cv
gs = GridSearchCV(dt_model,
                  param_grid = {'max_depth': range(1,11),
                                'min_samples_split':range(10,60,10)},
                  cv = 5,
                  n_jobs = 1,
                  scoring = 'neg_mean_squared_error')
# Fit model to training data
dt = gs.fit(x_train,y_train)
# Predict test data for out-of-sample r-sq
y_pred = dt.predict(x_test)
print(f'Out of Sample R-square: {round(r2_score(y_test,y_pred),3)}')
# Predict train data for in-sample r-sq
y_pred = dt.predict(x_train)
print(f'In Sample R-square: {round(r2_score(y_train,y_pred),3)}')
```

```
Out of Sample R-square: 0.987
In Sample R-square: 0.995
```

KNN Regressor

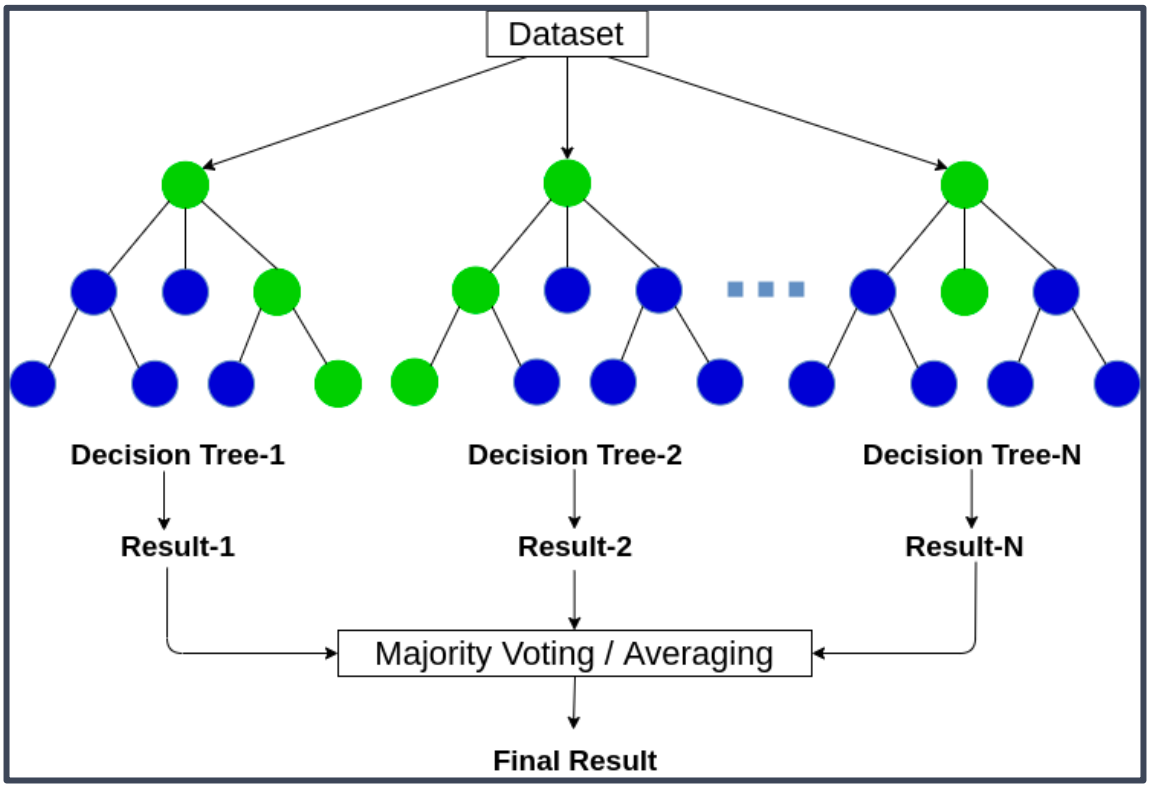
```
[ ]  rmse_val = [] #to store rmse values for different k
     for K in range(20):
         K = K+1
         model = neighbors.KNeighborsRegressor(n_neighbors = K)
         model.fit(x_train, y_train) #fit the model
         pred = model.predict(x_test) #make prediction on test set
         error = sqrt(mean_squared_error(y_test,pred)) #calculate rmse
         rmse_val.append(error) #store rmse values
         print('RMSE value for k= ' , K , 'is:', error)
```

```
RMSE value for k=  1 is: 2.842695260621061
RMSE value for k=  2 is: 1.8346159476195325
RMSE value for k=  3 is: 1.7397229306507045
RMSE value for k=  4 is: 1.8206752096832979
RMSE value for k=  5 is: 1.8945223870356516
RMSE value for k=  6 is: 1.9471601219258903
RMSE value for k=  7 is: 2.082274204332688
RMSE value for k=  8 is: 2.196810103644662
RMSE value for k=  9 is: 2.2196222240726957
RMSE value for k=  10 is: 2.3977982809284732
RMSE value for k=  11 is: 2.4858999253610268
RMSE value for k=  12 is: 2.5914879886887396
RMSE value for k=  13 is: 2.6624219157899756
RMSE value for k=  14 is: 2.740911385604616
RMSE value for k=  15 is: 2.8156429868018633
RMSE value for k=  16 is: 2.913926714263778
RMSE value for k=  17 is: 2.973212611442773
RMSE value for k=  18 is: 3.0214439016372006
```



6.9: Classification

Random Forest Classifier:



Random forest with scaled data, with Cross Validation

```
[ ] cv_model_norm = GridSearchCV(rf,param_grid=param_grid,cv=10)
    cv_model_norm.fit(X_train_norm, y_train_norm)
```

```
GridSearchCV(cv=10, estimator=RandomForestClassifier(random_state=0),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [4, 5, 6, 7, 8],
                         'max_features': ['auto', 'sqrt', 'log2'],
                         'n_estimators': [200, 500]})
```

```
[ ] cv_model_norm.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 8,
 'max_features': 'auto',
 'n_estimators': 500}
```

```
[ ] from sklearn.metrics import accuracy_score
    cv_best_model_norm = RandomForestClassifier(random_state=0, max_features='auto', n_estimators= 500, max_depth=8, criterion='gini')
    cv_best_model_norm.fit(X_train_norm, y_train_norm)
    cv_y_pred_norm = cv_best_model_norm.predict(X_test_norm)
    cv_score_norm = accuracy_score(cv_y_pred_norm, y_test_norm)

    print("AFTER CROSS VALIDATION")
    print(f'Accuracy Score: {round(cv_score,4)}')
    print(f'Classification error: {round(100 - (cv_score*100) ,2)}%')
```

```
AFTER CROSS VALIDATION
Accuracy Score: 0.9554
Classification error: 4.46%
```

Random Forest Classifier w/o scaled data, with Cross Validation (WITH ADDITIONAL FEATURES)

```
[ ] rf2 = RandomForestClassifier(random_state=0)
    param_grid = {
        'n_estimators': [200, 500],
        'max_features': ['auto', 'sqrt', 'log2'],
        'max_depth' : [4,5,6,7,8],
        'criterion' :['gini', 'entropy']
    }

    cv_model2 = GridSearchCV(rf2,param_grid=param_grid,cv=10)
    cv_model2.fit(X_train2, y_train2)

    cv_model2.best_params_
```

```
{'criterion': 'gini',
 'max_depth': 8,
 'max_features': 'auto',
 'n_estimators': 500}
```

```
[ ] cv_best_model2 = RandomForestClassifier(random_state=0, max_features='auto', n_estimators= 500, max_depth=8, criterion='gini')
    cv_best_model2.fit(X_train2, y_train2)
    cv_y_pred2 = cv_best_model2.predict(X_test2)
    cv_score2 = accuracy_score(cv_y_pred2, y_test2)
```

```
[ ] print("AFTER CROSS VALIDATION")
    print(f'Accuracy Score: {round(cv_score2,4)}')
    print(f'Classification error: {round(100 - (cv_score2*100) ,2)}%')
```

```
AFTER CROSS VALIDATION
Accuracy Score: 0.9524
Classification error: 4.76%
```

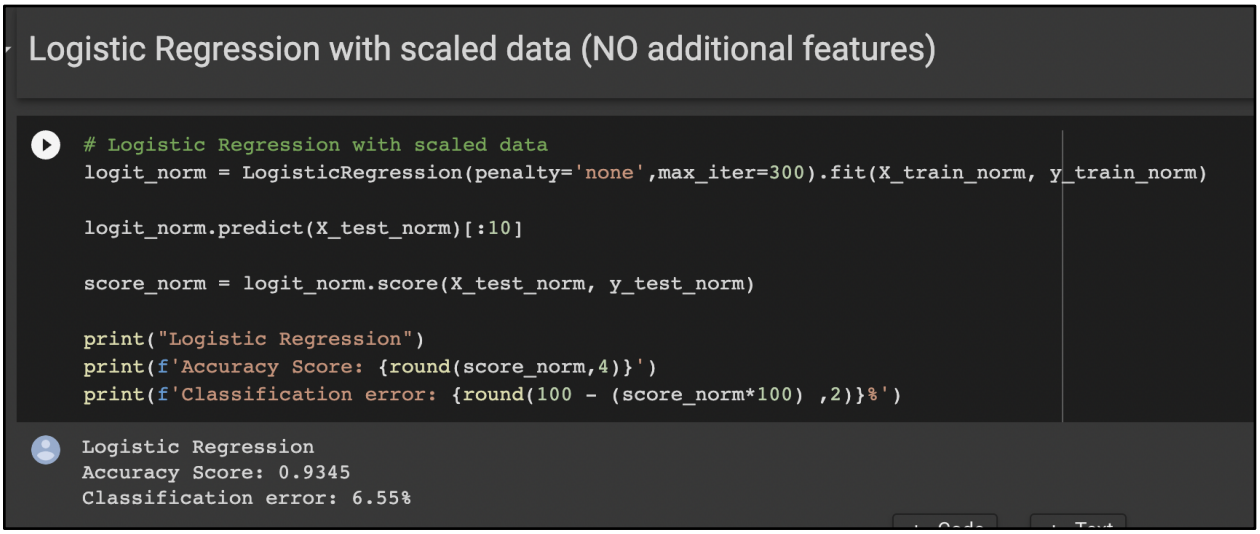
Logistic Regression:

Logistic Regression without scaled data (No additional features):

Logistic Regression w/o scaled data (NO additional features)

```
[ ] logit = LogisticRegression(penalty='none',max_iter=300).fit(X_train, y_train)

[ ] logit.predict(X_test)[:10]

    array([0, 1, 1, 1, 1, 1, 1, 1, 0, 1])

[ ] logit.predict_proba(X_test)[:10]

    array([[9.29334570e-01, 7.06654305e-02],
           [9.78300429e-02, 9.02169957e-01],
           [7.10409287e-03, 9.92895907e-01],
           [2.71294724e-02, 9.72870528e-01],
           [2.04608223e-01, 7.95391777e-01],
           [7.59952491e-02, 9.24004751e-01],
           [1.70504385e-01, 8.29495615e-01],
           [2.74735689e-04, 9.99725264e-01],
           [9.06154478e-01, 9.38455218e-02],
           [2.67710704e-02, 9.73228930e-01]])

    score = logit.score(X_test, y_test)
    print(f'Accuracy Score: {round(score,4)}')

    print(f'Classification error: {round(100 - (score*100) ,2)}%')

    Accuracy Score: 0.9345
    Classification error: 6.55%
```

Logistic Regression with scaled data (No additional features):



Logistic Regression with scaled data (NO additional features)

```
# Logistic Regression with scaled data
logit_norm = LogisticRegression(penalty='none',max_iter=300).fit(X_train_norm, y_train_norm)

logit_norm.predict(X_test_norm)[:10]

score_norm = logit_norm.score(X_test_norm, y_test_norm)

print("Logistic Regression")
print(f'Accuracy Score: {round(score_norm,4)}')
print(f'Classification error: {round(100 - (score_norm*100) ,2)}%')

Logistic Regression
Accuracy Score: 0.9345
Classification error: 6.55%
```

KNN Classifier:

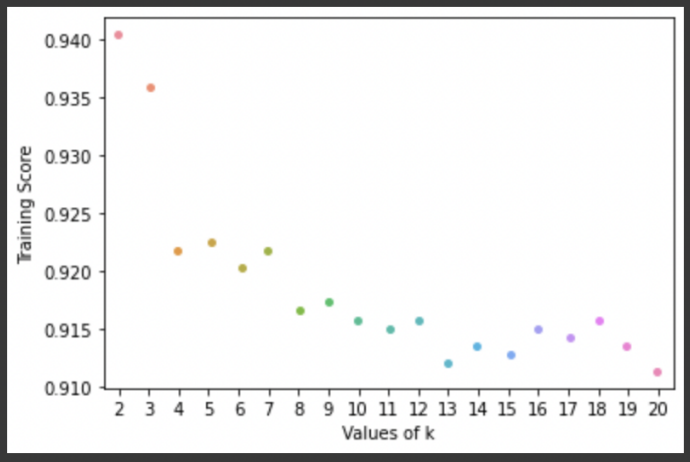
KNN without scaled data:

K Nearest Neighbours w/o scaled data

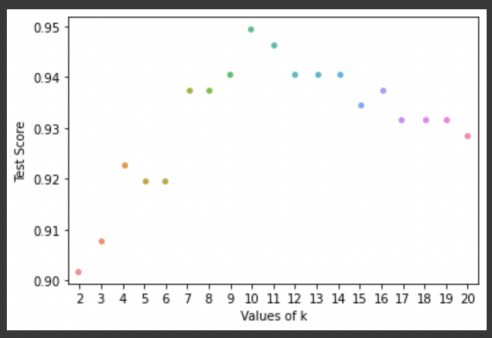
```
[ ] # Training the model for various values of k
    K = []
    training = []
    test = []
    scores = {}

    for k in range(2, 21):
        knn = KNeighborsClassifier(n_neighbors = k)
        knn.fit(X_train, y_train)

        training_score = knn.score(X_train, y_train)
        test_score = knn.score(X_test, y_test)
        K.append(k)

        training.append(training_score)
        test.append(test_score)
        scores[k] = [training_score, test_score]
```

```
    # Model Evaluation
    print("k : Training Score       Test Score")
    for keys, values in scores.items():
        print(keys, ':', values)
```

```
[ ] # k vs train scores
    ax = sns.stripplot(K, training);
    ax.set(xlabel ='Values of k', ylabel ='Training Score')
    plt.show()
```



```
[ ] # k vs test scores
    ax = sns.stripplot(K, test);
    ax.set(xlabel ='Values of k', ylabel ='Test Score')
    plt.show()
```




```
# Therefore, choosing k=10 to fit the model
knnFinal = KNeighborsClassifier(n_neighbors = 10)
knnFinal.fit(X_train, y_train)
accScore = knnFinal.score(X_test, y_test)
```

```
print(f'Accuracy Score: {round(accScore,4)}')

print(f'Classification error: {round(100 - (accScore*100) ,2)}%')
```

```
Accuracy Score: 0.9494
Classification error: 5.06%
```

KNN with scaled data:

KNN with scaled data

```
# KNN
knn_norm = KNeighborsClassifier(n_neighbors = 10)
knn_norm.fit(X_train_norm, y_train_norm)
accScore_norm = knn_norm.score(X_test_norm, y_test_norm)

print("KNN")
print(f'Accuracy Score: {round(accScore_norm,4)}')
print(f'Classification error: {round(100 - (accScore_norm*100) ,2)}%')
```

```
KNN
Accuracy Score: 0.9226
Classification error: 7.74%
```

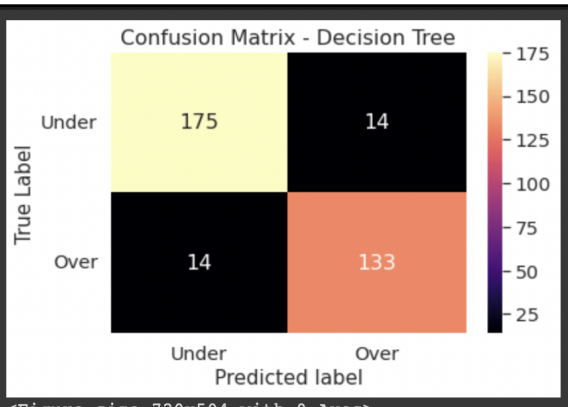
Decision Tree Classifier:

Decision Tree Classifier with scaled data:

Decision Tree with scaled data

```
[ ]  clf = DecisionTreeClassifier(max_depth =3, random_state = 42)
     clf.fit(X_train_norm, y_train_norm)

     test_pred_decision_tree_norm = clf.predict(X_test_norm)
```

```
[ ]  labels = data_copy['BS'].unique()

     #import the relevant packages
     from sklearn import metrics
     import seaborn as sns
     import matplotlib.pyplot as plt
     #get the confusion matrix
     confusion_matrix = metrics.confusion_matrix(y_test_norm,
                                                 test_pred_decision_tree_norm)
     #turn this into a dataframe
     matrix_df_norm = pd.DataFrame(confusion_matrix)
     #plot the result
     ax = plt.axes()
     sns.set(font_scale=1.3)
     plt.figure(figsize=(10,7))
     sns.heatmap(matrix_df_norm, annot=True, fmt="g", ax=ax, cmap="magma")
     #set axis titles
     ax.set_title('Confusion Matrix - Decision Tree')
     ax.set_xlabel("Predicted label", fontsize =15)
     ax.set_xticklabels(labels)
     ax.set_ylabel("True Label", fontsize=15)
     ax.set_yticklabels(list(labels), rotation = 0)
     plt.show()
```




```
<Figure size 720x504 with 0 Axes>
```

```
[ ]  score_dt_norm = metrics.accuracy_score(y_test_norm, test_pred_decision_tree_norm)

     print(f'Accuracy Score: {round(score_dt_norm,4)}')

     print(f'Classification error: {round(100 - (score_dt_norm*100) ,2)}%')
```

```
Accuracy Score: 0.9167
Classification error: 8.33%
```

Cost Complexity Pruning for Decision Trees:

Using Cost Complexity Pruning for Decision Trees (scaled data)

PS: Gives the same result for both original and scaled data

```
from sklearn.metrics import accuracy_score
import sklearn
print('The scikit-learn version is {}.'.format(sklearn.__version__))
```

```
The scikit-learn version is 1.0.2.
```

```
clf_tree = DecisionTreeClassifier(random_state=0)
```

```
path = clf_tree.cost_complexity_pruning_path(X_train_norm, y_train_norm)
ccp_alphas = path.ccp_alphas
```

```
from sklearn.model_selection import StratifiedKFold
kfolds = StratifiedKFold(n_splits = 10, shuffle = True, random_state = 1)

accuracies = []
for ccp_alpha in ccp_alphas:
  score_for_alpha = []
  for train_index, test_index in kfolds.split(X_train_norm, y_train_norm):
    clf = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf.fit(X_train_norm[train_index], y_train_norm[train_index])
    y_pred = clf.predict(X_train_norm[test_index])
    score = accuracy_score(y_pred, y_train_norm[test_index])
    score_for_alpha.append(score)
  accuracies.append(sum(score_for_alpha)/len(score_for_alpha))
```

```
print("The accuracies: ",accuracies)
print("\nThe index corresponding to the maximum of the accuracies: ",np.argmax(accuracies))
```

```
The accuracies:  [0.9053731343283582, 0.9053731343283582, 0.9061194029850747, 0.90611940298507

The index corresponding to the maximum of the accuracies:  18
```

```
 max_index = np.argmax(accuracies)
 # Use the selected alpha to retrain a tree on the entire (X_train,y_train)
alpha_cv = ccp_alphas[max_index]
clf_tree_final = DecisionTreeClassifier(random_state=0, ccp_alpha=alpha_cv)
clf_tree_final.fit(X_train_norm, y_train_norm)
# Evaluate on the (X_test,y_test)
y_pred_test = clf_tree_final.predict(X_test_norm)
score_test = accuracy_score(y_test_norm, y_pred_test)
print(f'Accuracy Score: {round(score_test,4)}')

print(f'Classification error: {round(100 - (score_test*100) ,2)}%')
```

```
max_index = np.argmax(accuracies)
# Use the selected alpha to retrain a tree on the entire (X_train,y_train)
alpha_cv = ccp_alphas[max_index]
clf_tree_final = DecisionTreeClassifier(random_state=0, ccp_alpha=alpha_cv)
clf_tree_final.fit(X_train_norm, y_train_norm)
# Evaluate on the (X_test,y_test)
y_pred_test = clf_tree_final.predict(X_test_norm)
score_test = accuracy_score(y_test_norm, y_pred_test)
print(f'Accuracy Score: {round(score_test,4)}')

print(f'Classification error: {round(100 - (score_test*100) ,2)}%')
```

```
Accuracy Score: 0.9018
Classification error: 9.82%
```

```
fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
from sklearn import tree
tree.plot_tree(clf_tree_final, feature_names = ['S','K','tau','r'], class_names=['0','1'], filled = True);
```

Neural Networks:

Neural Networks without scaled data:

Neural Networks w/o scaled data

```
from sklearn.neural_network import MLPClassifier

nn = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
```

```
#nn.predict_proba(X_test)
```

```
nn.predict(X_test)
```

```
array([0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0,
       0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0,
       0, 1, 0, 1, 1, 0])
```

```
[ ]  score_nn = nn.score(X_test, y_test)

     print("Neural Network")
     print(f'Accuracy Score: {round(score_nn,4)}')

     print(f'Classification error: {round(100 - (score_nn*100) ,2)}%')
```

```
Neural Network
Accuracy Score: 0.9286
Classification error: 7.14%
```

Neural Networks without scaled data:

Neural Networks with scaled data

```
[ ]  nn_norm = MLPClassifier(random_state=1, max_iter=300).fit(X_train_norm, y_train_norm)
```

```
[ ]  #nn_norm.predict_proba(X_test_norm)
```

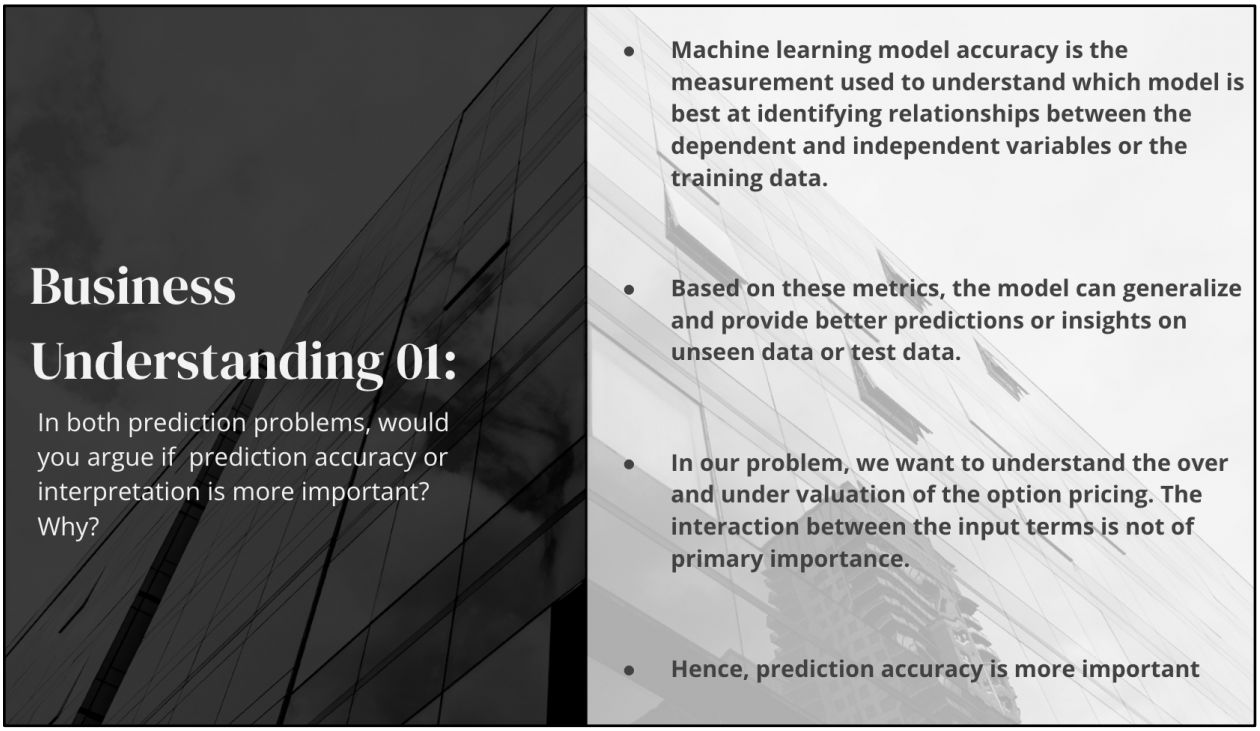
```
nn_norm.predict(X_test_norm)
```

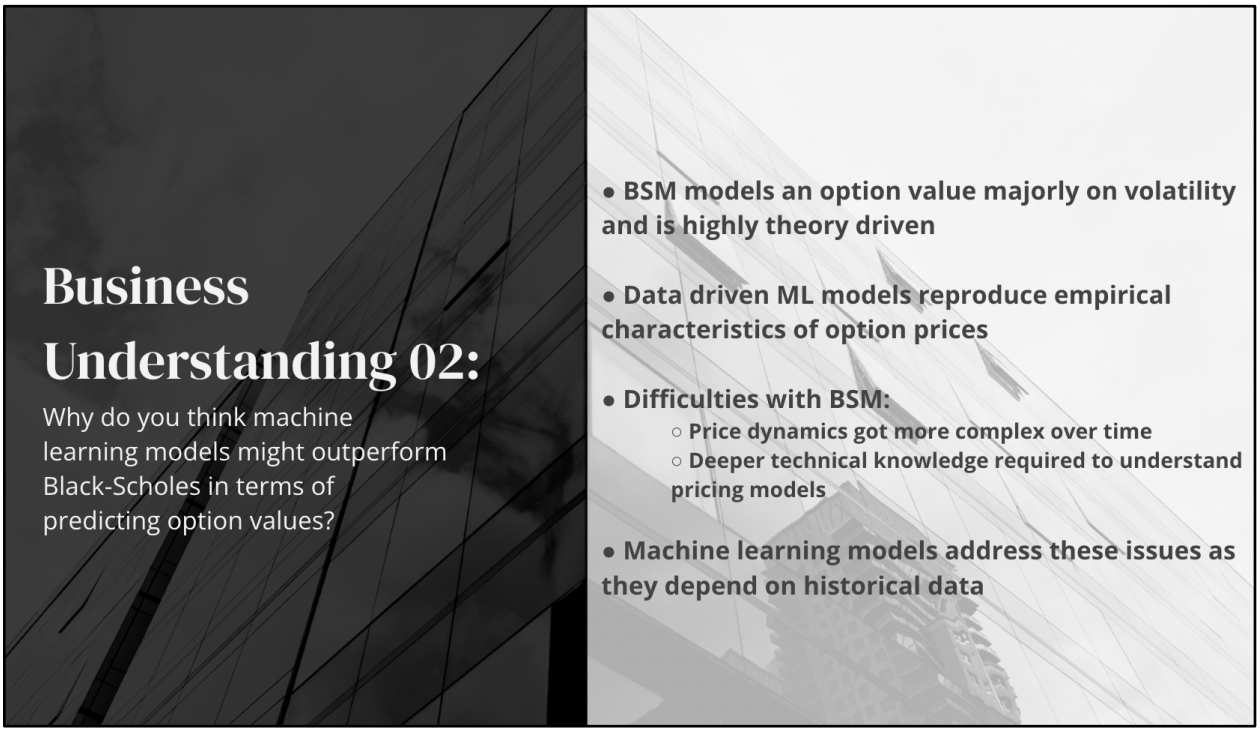
```
array([0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 1,
       0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1,
       1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 0,
       1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0,
       0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0,
       0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0,
       0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 0,
       0, 1, 0, 1, 1, 0])
```

```
[ ]  score_nn_norm = nn_norm.score(X_test_norm, y_test_norm)

     print("Neural Network")
     print(f'Accuracy Score: {round(score_nn_norm,4)}')

     print(f'Classification error: {round(100 - (score_nn_norm*100) ,2)}%')
```

```
Neural Network
Accuracy Score: 0.9196
Classification error: 8.04%
```

6.10: Business Understandings





Business Understanding 03:

Can you argue from a business perspective that all four predictor variables should be included in the prediction?

YES!!

- **S: Current asset value**
  - **price that provides value to the underlying asset**
  - **without it, the option has no hold intrinsic value**
- **K: Strike price of option**
  - **price at which a derivative contract is exercised**
  - **without it, there would be no indication of the ability to exercise the call option**
- **r: Annual interest rate**
  - **interest rate determines the return over principle**
  - **it is an indication of the value that can be earnt by the ownership and sale of stock**
- **tau: Time to maturity (in years)**
  - **interest rate is of little value without tau**
  - **tau complements the rate of return**

Business Understanding 04:

Are you comfortable about directly using your trained model to predict option values for Tesla stocks? Why?

NO!

- **No! Our trained model is Random Forest Regressor which is used to predict option values (C)**
- **Model's r-sq on train data = 99.9%**

  Model's r-sq on test data = 99.6%
- **When we are trying to predict option values for other stocks, we have to first consider our data. If the test data is similar to the training data then we can comfortably say that the trained model would perform well on Tesla stocks.**
- **However, if the test data is not similar to the training data then the model will not perform well.**