DSO 530 Group Project

Options Pricing Prediction

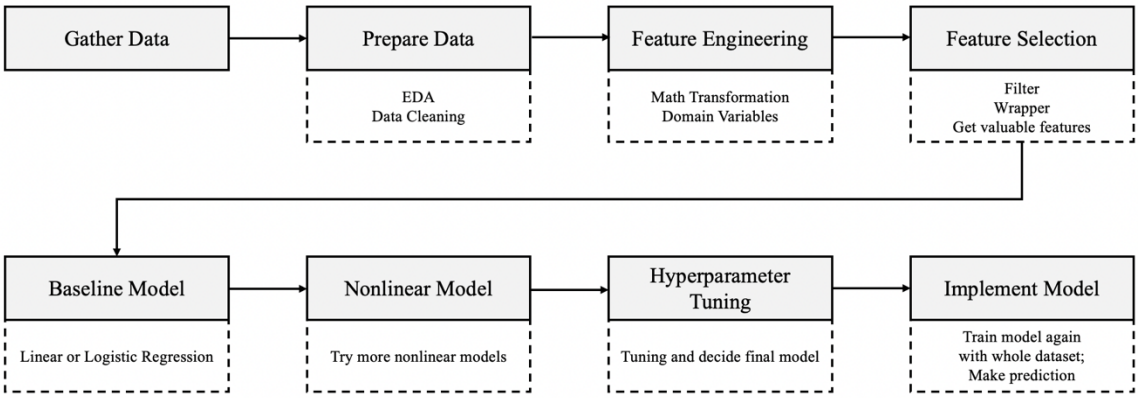
Ge Zeng, Teresa Cao, Xinyue Yu,
Xiaoying Zou, Yufeng Xie, Zirui Zhou

1. Executive Summary

This project focuses on predicting European call option prices by building and comparing regression and classification machine learning models.

The original training dataset contains 1,680 records and 6 columns. For each record, we have 4 features, 1 numerical and 1 categorical dependent variables. We started with exploratory data analysis to understand data patterns and preprocess data. After data cleaning for regression and classification, we generated 1,678 new features through feature engineering, and selected around 30 features through feature selection. By randomly splitting the data into training and testing dataset in the proportion of 4:1, we constructed regression and classification models including decision tree, neural network, LGBM and other models with the selected features, performed hyperparameter tuning, and found out our best performers for regression and classification. We chose the gradient boosting tree as our final regression model and LGBM as our final classification model. The gradient boosting tree achieves the out-of-sample R^2 of 99.89% in testing, and the LGBM model achieves an accuracy score of 93.40% in testing.

Figure 1. *Flowchart of Project*



2. Data Description

2.1 Summary Statistics Table

Table 1. *Numeric Fields Summary*

	min	max	mean	std	count	% Populated	# Zero	% Zero
Value	0.13	60.15	15.07	14.04	1679	99.94	0	0.00
S	0.00	455.88	440.64	13.13	1679	99.94	1	0.06
K	375.00	500.00	438.24	23.41	1678	99.88	0	0.00
tau	0.00	250.00	0.44	7.06	1679	99.94	0	0.00
r	0.03	0.03	0.03	0.00	1680	100.00	0	0.00

Table 2. *Categorical Fields Summary*

	Data type	# Nonnull records	# Non-zero records	% Populated	# Unique values
BS	object	1680	1680	100	2

2.2 Data Distribution Graph

See appendix.

3. Data Cleaning

3.1 Regression

We dropped the record that has 3 missing values, S, K and tau. Then, we dropped the record that does not contain the dependent variable, Value. After dropping two records, we replaced 2 outliers in tau with the mean to correct outliers and filled in the missing S with the mean.

3.2 Classification

We did the same data cleaning procedure except that we didn't drop the record that does not contain Value as it still contains the dependent variable BS.

4. Candidate Variables

4.1 Regression

We implemented feature engineering and created 5 types of variables including basic transformation variables, exponential variables, ratio variables, domain and deviation variables and polynomial variables. Finally we created 1678 candidate variables.

We performed 3 basic transformations on the original variables S, K, tau and r. We also added one variable, S-K. We performed Log Transformation and Square Root Transformation on S, K, tau, and r. Then we used tau as the index to create 6 Exponential Variables, r^tau, exp(r)^tau, S^tau, log(S)^tau, K^tau and log(K)^tau. Next, We used any two of the variables generated above to create Ratio Variables. In addition, we created one Domain Variable and Deviation Variable, which we defined as:

- Domain = S*(1+r)^tau
- Deviation = (S-mean(S))^2

In the end, we used any two of the variables generated above to create Quadratic Polynomial Variables with the degree of 2.

4.2 Classification

At first, we used the same variables which we created above for regression. However, the modeling results are dissatisfactory, and we found out that simpler variables would give better model performance. Therefore, we decided to remove some complicated variables. As a result, we removed Log Variables, Nature Exponential Variables and Quadratic Polynomial Variables.

5. Feature Selection Process

After feature engineering, we conducted a feature selection process including filter and wrapper to select the most significant variables. We used mutual information metric as our filter to select the top 80 variables with univariate significance, then used linear regression model as our regression wrapper to select the top 31 variables, and boosted tree classification model as our classification wrapper to select the top 30 variables. The following table shows the methods we used in regression and classification parts.

Table 3. *Filters and Wrappers for Feature Selection Process*

Attributes	Filter		Wrapper	
	Methods	Number of Selected Features Subset-1	Methods	Number of Selected Features Subset-2*
Regression	Mutual Information Regression	80	Linear Regression Model	31
Classification	Mutual Information Classification	80	Boosted Tree Classification Model	30

* See Selected Feature Subset-2 in Appendix

6. Model Algorithms

6.1 Regression Model Tuning

After feature selection, we used our final 31 candidate variables for model building and hyperparameter tuning. We used the ouf-of-sample R square from 10-fold cross validation as our evaluation metric. We tried four baseline models with default parameters including linear regression (99.72%), decision tree (99.72%), random forest (99.84%), and neural network (99.89%), all of which achieved up to 99% out-of-sample R square. indicating that our feature engineering and selection have been paid off.

Then we continued with 5 different models including Xgboost tree, LGBM tree, Gradient Boosting tree, Support Vector Machine, and K-Nearest Neighbors for regression prediction. For each model, we tuned hyperparameters to achieve best prediction performance. Then, we compared performances among different models with best choices of hyperparameters to finalize our model decision.

After model tuning, we summarized the models with best choices of hyperparameters. The table below shows our models with the optimal set of hyperparameters. According to the result, gradient boosting tree is our best model. With n_estimators=50, max_depth=30, mon_sample_leaf=20, min_sample_split=50, the gradient boosting tree can achieve 99.98% out-of-sample R square. We enclosed our complete hyperparameter tuning process in the Appendix

Figure 2. *Regression Best Tuning Results*

Model	Parameters								Performance	
Linear Regression	**#Variables**		**fit_intercept**		**positive**		**copy_X**	**normalize**	**Train**	**Test**
	31		TRUE		TRUE		TRUE	TRUE	97.29%	97.24%
Neural Network	**learning_rate**	**nodes**	**max_iter**	**layer**	**momentum**	**alpha**	**learning_rate_init**	**nesterovs_momentum**	**Train**	**Test**
	constant	100+100	200	2	0.9	0.0005	0.001	TRUE	99.82%	99.82%
Decision Tree	**criterion**		**max_depth**		**min_samples_leaf**		**min_samples_split**	**max_features**	**Train**	**Test**
	gini		None		1		10	None	99.93%	99.72%
Xgboost	**max_depth**				**eta**		**lambda**		**Train**	**Test**
	9				0.05		0.3		99.98%	99.82%
SVM	**kernel**				**C**		**tol**		**Train**	**Test**
	default				9		0.00001		99.83%	99.82%
LGBM	**n_estimator**		**learning_rate**				**max_depth**	**min_child_samples**	**Train**	**Test**
	default		0.1				20	2	99.96%	99.83%

Model	Parameters					Performance	
Gradient Boosting Tree	n_estimators	max_depth	min_sample_leaf	min_sample_split		Train	Test
	50	30	20	50		99.89%	**99.89%**
Random Forest	bootstrap	n_estimators	min_samples_leaf	min_samples_split		Train	Test
	TRUE	100	1	2		99.98%	99.84%
KNN	# neighbors		p	weights	metric	Train	Test
	10		1	uniform	minkowski	99.98%	99.84%

6.2 Classification Model Tuning

To select the most effective classification model, we tried 6 classification models including logistic regression, LDA, decision tree, neural network, random forest and boosted tree models. At first, we tried to used variables created from regression, but we got relatively high classification errors. So we deleted some complicated variables, simplified our feature engineer process and kept 30 features for model fitting.

We experimented with hyperparameter tuning for different models and compared their prediction accuracy under 10-fold cross validation. The table below listed the best performers for different classification models. We attach the complete hyperparameter tuning table in the appendix.

Our optimal performer is the LGBM classifier of 500 trees, maximum depth of 5 and learning rate of 0.01. With 10-fold cross validation, we achieved the average accuracy score of 97.1% for the training set and 93.4% for the test set. Equivalently, we achieved an average classification error of 2.9% for training set and 6.6% for the test set.

Table 4. *Classification Tuning Results*

Model	Parameters				Accuracy	
Logistic Regression	penalty		solver		trn	tst
	l2		lbfgs		0.918	0.915
Single Decision Tree	max_depth		splitter		trn	tst
	20		random		1.000	0.915
Random Forest	n_estimators	max_depth	min_samples_leaf	min_samples_split	trn	tst
	10	10	1	2	0.992	0.925
Boosted Tree (LGBM)	n_estimators	max_depth	learning_rate		trn	tst
	500	**5**	**0.01**		**0.971**	**0.934**
Neural Network	hidden_layer_sizes	layer	learning_rate	activation	trn	tst
	20	2	constant	logistic	0.656	0.656
LDA	n_components		solver	shrinkage	trn	tst
	None		svd	None	0.926	0.921

7. Conclusion and Business Insight

Our group walked through a complete machine learning process with the given dataset to make regression predictions for C (Option Value) and classification regression for BS (Black-Scholes estimation). For data preparation, we did exploratory data analysis and data cleaning. We performed feature engineering, feature selection, modeling and hyperparameter tuning. Finally, we selected Gradient Boosting Tree (with n_estimators=50, max_features=None, max_depth=30, min_samples_leaf=20, min_samples_split=50) as our best regression model as it achieved 99.98% out-of-sample R square. For the best classification model, we selected LGBM (with n_estimators= 500, max_depth=5, learning_rate=0.01) as it achieved the average accuracy score of 93.40% for the test set.

When building our models, we weighed the tradeoff between prediction accuracy and interpretation. Although we believe both accuracy and interpretation are important, prediction accuracy is our key focus in this project as achieving high prediction accuracy is our main goal for this project. A more accurate model is viewed as a more valuable model. Interpretability seems to be a subjective model assessment, but it's critical when we want to achieve a real business objective. We included all four predictor variables (S, K, r, t) in our prediction because they are primary factors that determine the value of an option from a business perspective.

After finishing the project, we found that machine learning models may outperform Black-Scholes in prediction accuracy. As we know, Black-Scholes model is limited to European options and restricted to many assumptions, such as lognormal distribution and constant risk-free interest rate. These assumptions may lower the accuracy of model predictions. Thus, the machine learning model outperforms with greater flexibility in working with different data and higher prediction accuracy.

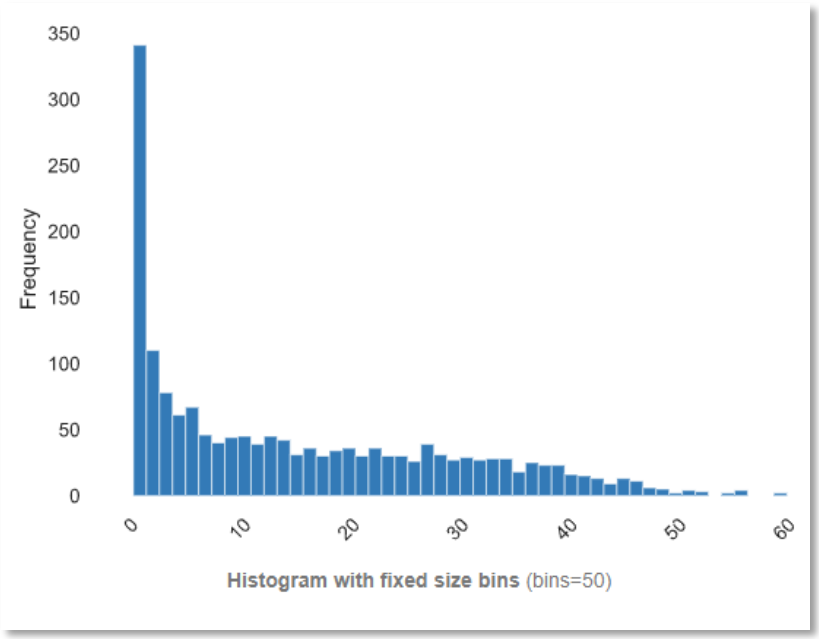
Although we achieved high prediction accuracy with our models, we would not apply it to predict option values for Tesla stock. First, valuing an option is tricky because it depends on the future value of the underlying asset. We have no information about underlying assets and we are not sure if the assets are similar to Tesla stock. Second, machine learning methods make predictions only based on past price movements and do not take external uncertainties into consideration. Also, our model does not factor in volatility, which is critical in predicting the option value for high-volatile stocks like Tesla. However, even if we can obtain the data of historical or implied volatility, it is still difficult to predict its option value since its high volatility not only comes from stock markets but also comes from Tesla's strong connection with cryptocurrency and the influence of their CEO Elon Musk. Thus, predicting option values for Tesla stock with our model is not feasible.

Our group also brainstormed and discussed how to improve our future model-building process. In our model, the filter we used may lower the importance of S and K, which may potentially decreased the accuracy score for the test set of our classification model. We also realized that the filter itself could be problematic in nature, potentially resulting in losing significant variables. Moreover, we learned to avoid over-complicated variables and models, which would lower both accuracy and interpretability. In the future, we will start with the simple variables and basic models and apply advanced models for better prediction and interpretability.

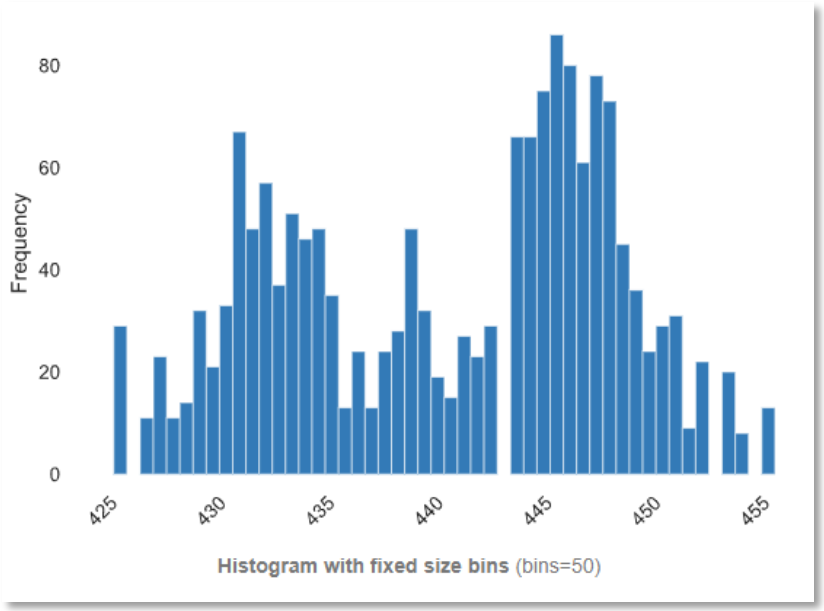
8. Appendix

8.1 Field Distribution

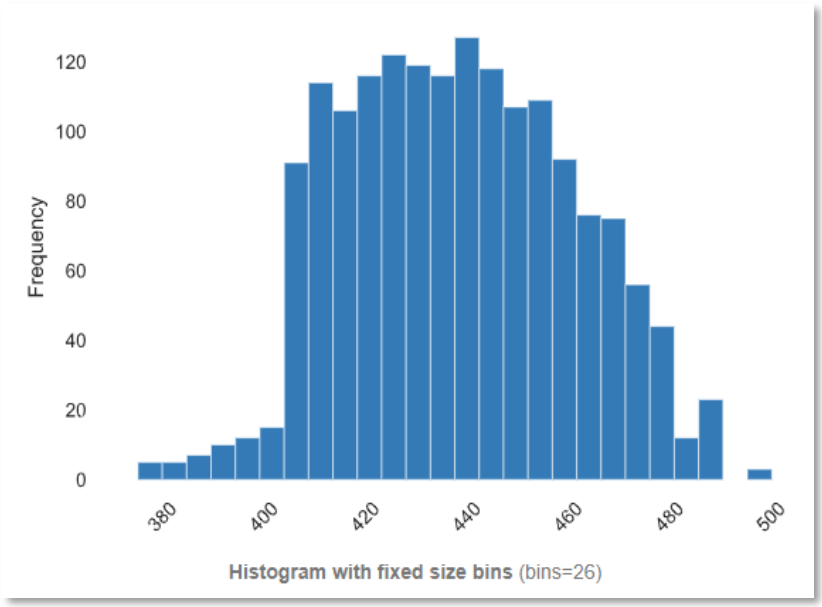
8.1.1 Value



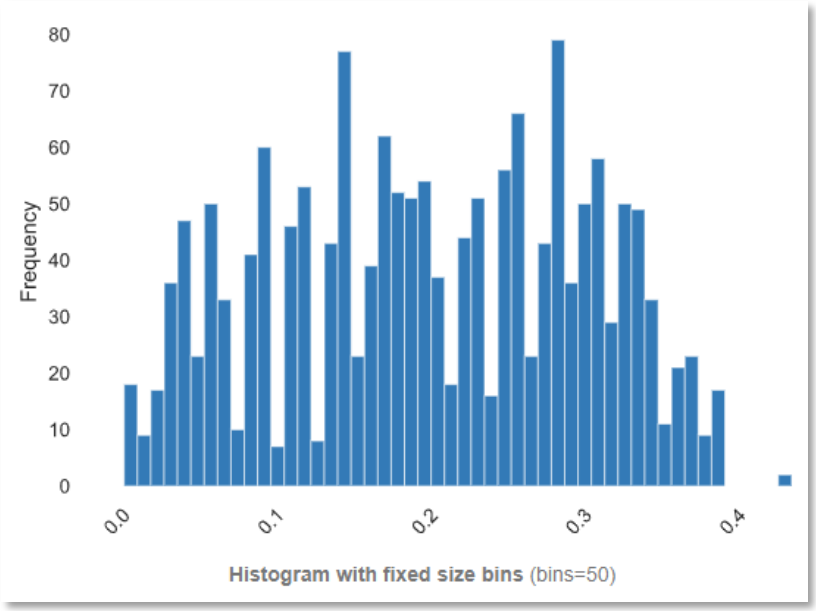
8.1.2 S



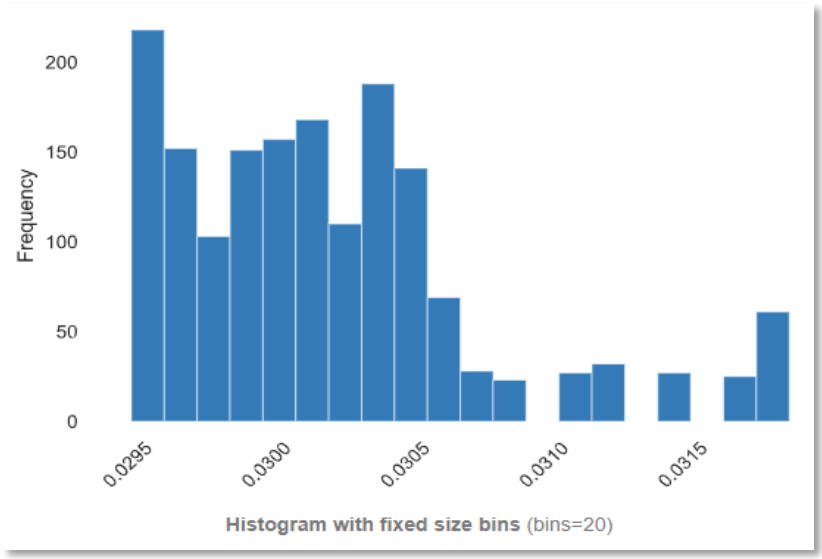
8.1.3 K


Frequency
120
100
80
60
40
20
0
380
400
420
440
460
480
500
Histogram with fixed size bins (bins=26)

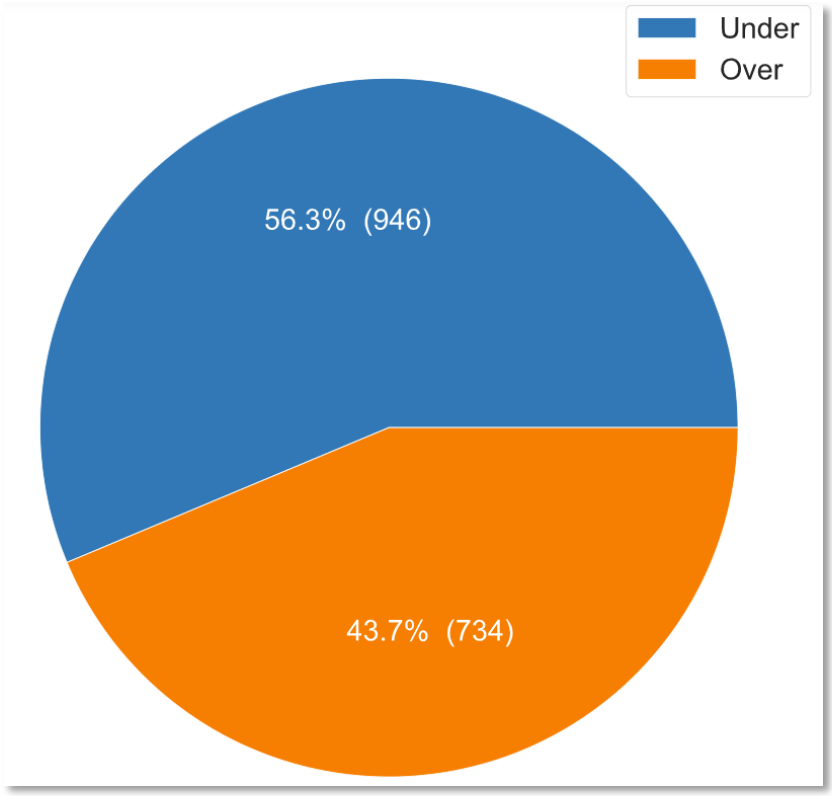
8.1.4 tau


Frequency
80
70
60
50
40
30
20
10
0
0.0
0.1
0.2
0.3
0.4
Histogram with fixed size bins (bins=50)

8.1.5 r


200
150
100
50
0
Frequency
0.0295
0.0300
0.0305
0.0310
0.0315
Histogram with fixed size bins (bins=20)

8.1.6 BS


Under
Over
56.3% (946)
43.7% (734)

8.2 Result of Hyperparameter Tuning

8.2.1 Regression

Model	Parameters								Performance	
Linear Regression	Iteration	#Variables		fit_intercept	positive	copy_X	normalize		Train	Test
	Default	31		TRUE	FALSE	FALSE	FALSE		99.743%	99.723%
	2	31		TRUE	FALSE	TRUE	FALSE		99.750%	99.723%
	3	31		TRUE	FALSE	TRUE	TRUE		99.750%	99.723%
	4	31		TRUE	TRUE	TRUE	TRUE		97.287%	97.239%
Neural Network	Iteration	layer	nodes	activation	alpha	learning_rate			Train	Test
	Default	1	100	relu	0.0001	constant			99.403%	99.391%
	2	1	10	relu	0.0001	adaptive			97.782%	97.573%
	3	1	200	relu	0.0001	adaptive			99.612%	99.602%
	4	2	100+100	relu	0.0001	adaptive			99.814%	99.805%
	5	2	100+100	relu	0.0001	constant			99.811%	99.801%
	6	2	100+100	tanh	0.0001	constant			99.636%	99.629%
	7	1	100	tanh	0.0001	constant			98.644%	98.617%
	8	2	100+100	relu	0.0005	constant			99.824%	99.815%
Decision Tree	Iteration	max_depth		min_samples_leaf	min_samples_split	max_features			Train	Test
	Default	None		1	2	None			100.000%	99.726%
	2	5		35	60	auto			99.295%	99.143%
	3	10		35	60	auto			99.320%	99.170%
	4	None		10	2	None			99.797%	99.645%
	5	None		1	10	None			99.932%	99.724%
Xgboost	Iteration	max_depth		eta		lambda			Train	Test
	Default	Default		Default		Default			99.990%	99.790%
	2	9		0.05		0.3			99.980%	99.820%
SVM	Iteration	kernel		C		tol			Train	Test
	Default	Default		Default		Default			97.700%	96.880%
	2	linear		Default		Default			96.960%	96.910%
	3	default		9		0.00001			99.830%	99.820%
LGBM	Iteration	n_estimator	learning_rate	max_depth		min_child_samples			Train	Test
	Default	Default	Default	Default		Default			99.870%	99.740%
	2	1000	0.1	Default		Default			99.920%	99.730%
	3	default	0.1	8		50			99.650%	99.570%
	4	default	0.1	8		10			99.930%	99.820%
	5	default	0.1	20		2			99.960%	99.830%
Gradient Boosting Tree	Iteration	learning_rate	n_estimators	max_depth	max_features	min_sample_leaf	min_sample_split	subsample	Train	Test
	Default	0.1	100	3	None	1	2	1	99.894%	99.793%
	2	0.1	50	30	None	20	50	1	99.894%	99.894%
	3	0.2	100	3	None	10	2	1	99.894%	99.796%
	4	0.1	100	20	None	10	2	1	99.894%	99.797%
	5	0.3	50	10	None	20	2	1	99.894%	99.796%
	6	0.1	200	3	None	1	50	1	99.894%	99.797%
Random Forest	Iteration	bootstrap	n_estimators	max_depth	max_features	min_samples_leaf	min_samples_split	criterion	Train	Test
	Default	TRUE	100	None	auto	1	2	squared_error	99.978%	99.840%
	2	TRUE	100	100	auto	20	2	squared_error	99.978%	99.840%
	3	TRUE	200	100	auto	10	2	squared_error	99.978%	99.838%
	4	TRUE	50	None	auto	1	2	squared_error	99.978%	99.839%
KNN	Iteration	# neighbors		p		weights	metric		Train	Test
	Default	5		2		uniform	minkowski		99.978%	99.838%
	2	10		2		uniform	minkowski		99.978%	99.839%
	3	3		2		uniform	minkowski		99.978%	99.839%
	4	5		1		uniform	minkowski		99.978%	99.839%
	5	3		1		uniform	minkowski		99.978%	99.837%
	6	10		1		uniform	minkowski		99.978%	99.840%

8.2.1 Classification

Model	Parameters				Accuracy	
Logistic Regression	penalty		solver		trn	tst
	none		saga		0.892	0.891
	l2		saga		0.892	0.892
	none		lbfgs		0.917	0.915
	l2		lbfgs		0.918	0.915
Single Decision Tree	max_depth		splitter		trn	tst
	None		random		1.000	0.912
	5		best		0.946	0.909
	10		random		0.991	0.904
	10		best		0.975	0.912
	20		random		1.000	0.915
	15		best		1.000	0.906
Random Forest	n_estimators	max_depth	min_samples_leaf	min_samples_split	trn	tst
	10	10	1	2	0.992	0.925
	20	10	10	10	0.945	0.916
	20	20	20	20	0.930	0.915
	50	30	20	20	0.932	0.911
	50	50	10	10	0.946	0.921
	50	50	20	20	0.931	0.915
	50	50	30	50	0.923	0.911
	50	50	15	20	0.936	0.918
	100	50	10	10	0.947	0.918
	300	50	5	10	0.967	0.923
Boosted Tree	n_estimators	max_depth	learning_rate		trn	tst
	20	3	0.1		0.934	0.918
	100	5	0.01		0.943	0.921
	200	5	0.1		1.000	0.936
	200	5	0.01		0.953	0.930
	500	5	0.1		1.000	0.938
	500	**5**	**0.01**		**0.971**	**0.934**
	1000	50	0.003		0.975	0.933
	1000	50	0.005		0.995	0.934
	1000	100	0.003		0.975	0.933
Neural Network	hidden_layer_sizes	layer	learning_rate	activation	trn	tst
	5	1	constant	relu	0.527	0.528
	5	1	adaptive	logistic	0.563	0.563
	10	1	constant	relu	0.555	0.555
	10	2	adaptive	logistic	0.607	0.606
	20	2	adaptive	relu	0.568	0.567
	20	2	constant	logistic	0.656	0.656
LDA	n_components		solver	shrinkage	trn	tst
	None		svd	None	0.926	0.921
	1		lsqr	auto	0.915	0.912
	1		eigen	auto	0.915	0.912

8.3 Code Reference

8.3.1 Load & EDA

```
import pandas as pd
from datetime import datetime
start_time = datetime.now()

# %pip install plotly
# %pip install playsound
# %pip install xgboost
# %pip install lightgbm
# %pip install mlxtend

from sklearn.decomposition import PCA
from sklearn.cross_decomposition import PLSRegression

import matplotlib.cbook as cbook
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR

from sklearn.metrics import r2_score
from sklearn.model_selection import cross_validate
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
import numpy as np
import xgboost as xgb
import lightgbm as lgb
import matplotlib.pyplot as plt
from IPython.display import display as d
from IPython.display import Audio

print('LOAD DURATION: ', datetime.now() - start_time) # about a minute
```

8.3.2 Imputation

```
data.head()
```

	Value	S	K	tau	r	BS
0	21.670404	431.623898	420.0	0.341270	0.03013	Under
1	0.125000	427.015526	465.0	0.166667	0.03126	Over
2	20.691244	427.762336	415.0	0.265873	0.03116	Under
3	1.035002	451.711658	460.0	0.063492	0.02972	Over
4	39.553020	446.718974	410.0	0.166667	0.02962	Under

```
# clean data
data=data.drop('BS', axis=1)
data=data.drop(axis=0, index=292)
data=data.drop(axis=0, index=818)

data.loc[data['S']==0,'S'] = data['S'].mean()
data.loc[data['tau']==250,'tau'] = data['tau'].mean()
data.loc[data['tau']==146,'tau'] = data['tau'].mean()
```

```
data.shape
```

```
# clean data
data=data.drop('Value', axis=1)
data['BS']=[1 if i=='Over' else 0 for i in data['BS']]
first_column = data.pop('BS')
data.insert(0, 'BS', first_column)
data=data.drop(axis=0, index=292)

data['K']=data['K'].fillna(data['K'].mean())
data.loc[data['S']==0,'S'] = data['S'].mean()
data.loc[data['tau']==250,'tau'] = data['tau'].mean()
data.loc[data['tau']==146,'tau'] = data['tau'].mean()
data
```

8.3.3 Feature Engineering

```
# add basic variable
import math
data['S-K']=[data.loc[i,'S']-data.loc[i,'K'] for i in data.index]
var=data.columns[1:5].to_list()
for i in var:
    data[i+'_'+'log'] = [math.log(n) for n in data[i]]
    data[i+'_'+'sqrt'] = [math.sqrt(n) for n in data[i]]
for i in data.columns[3:5].to_list():
    data[i+'_'+'exp'] = [math.exp(n) for n in data[i]]
```

```
X_trntst = data.iloc[:,1:17]
Y_trntst = data['Value']

data_cols1=X_trntst.columns.tolist()
data_cols1
```

...

```
r_l = ['r','r_exp']
S_l = ['S', 'S_log']
K_l = ['K', 'K_log']

# add power variables
for i in ['tau']:
    for j in r_l+S_l+K_l:
        if 0 in X_trntst[j].unique():
            continue
        else:
            tmp_name = f'{j}^{i}'
            print(tmp_name)

            X_trntst[tmp_name] = X_trntst[j]**X_trntst[i]
```

...

```
data_cols2=X_trntst.columns.tolist()

for i in data_cols2:
    for j in data_cols2:
        if i < j:

            if 0 in X_trntst[j].unique():
                continue


            else:
                # Add ratio variables
                tmp_name = f'{i}/{j}'
                print(tmp_name)

                X_trntst[tmp_name] = X_trntst[i]/X_trntst[j]


X_trntst.shape
```

...

```
# domain
X_trntst['domain']  = [(data.loc[i,'S']*(1+data.loc[i,'r'])**data.loc[i,'tau']) for i in data.index]
X_trntst['deviation'] = [(n-data['S'].mean())**2 for n in data['S']]
```

```
data_cols3=X_trntst.columns.tolist()
len(data_cols3)
```

...

```
import re
data_cols3_str=','.join(data_cols3)
remove_sqrt_1 = re.findall(r'(\w+/\w+_sqrt|\w+_sqrt/\w+|\w+_sqrt)',data_cols3_str)
data_cols3 = [i for i in data_cols3 if i not in remove_sqrt_1]
len(data_cols3)
```

...

```
X_trntst.head()
```

```
from sklearn import preprocessing
def polynomial_features(dataframe, num_cols):
    df = dataframe.loc[:, num_cols].copy()
    normal_num = df.shape[1]
    pf = preprocessing.PolynomialFeatures(
        degree=2,
        interaction_only=False,
        include_bias=False
    )
    pf.fit(df)
    poly_feats = pf.transform(df)
    poly_feats = poly_feats[:, normal_num:]
    poly_columns = pf.get_feature_names(df.columns)[normal_num:]
    poly_columns = [(col.replace(' ', '--*--').replace('^2', '--^2')+'_poly') for col in poly_columns]
    df_poly = pd.DataFrame(poly_feats, columns=poly_columns)
    return df_poly
```

```
poly_df = polynomial_features(X_trntst, data_cols3)
for col in poly_df.columns:
    X_trntst[col] = poly_df[col].tolist()
```

8.3.4 Feature Selection

```
from sklearn.model_selection import train_test_split
# train/test split
X_train, X_test, y_train, y_test = train_test_split(X_trntst, Y_trntst, test_size=0.2, random_state=1)
```

```
import scipy.stats as sps
from sklearn.feature_selection import mutual_info_classif as MIC

result = MIC(X_train,y_train)

k = result.shape[0] - sum(result <= 0)
```

```
from sklearn.model_selection import train_test_split
# train/test split
X_train, X_test, y_train, y_test = train_test_split(X_trntst, Y_trntst, test_size=0.2, random_state=1)
```

```
from sklearn.feature_selection import mutual_info_regression as MIR

result = MIR(X_train,y_train)
```

```
dd=pd.Series(data=result,index=X_train.columns.tolist())
```

```
dd=dd.sort_values(ascending=False)
dd[0:80]
```

```
filter_columns=list(dd[0:80].index)
```

SequentialFeatureSelector

```
[34]:
# Z-scale all features before SFS
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_trn_scaled = scaler.fit_transform(X_train)
X_tst_scaled = scaler.transform(X_test)
X_trn_scaled = pd.DataFrame(X_trn_scaled, index = X_train.index, columns = X_train.columns)
X_tst_scaled = pd.DataFrame(X_tst_scaled, index = X_test.index, columns = X_test.columns)
```

```
[35]:
%%time
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
# model=DecisionTreeClassifier(criterion='gini',max_depth=10)
model=LinearRegression()
# for SFS forward, use k_feature=Number of variables minus 1; for backward use 2
sfs=SFS(model,k_features=79  ,forward=True,verbose=2,scoring='r2',cv=0,n_jobs=-1)
sfs.fit(X_trn_scaled[filter_columns],y_train)
```

```
%%time
import warnings
from lightgbm import LGBMClassifier
warnings.filterwarnings('ignore', category=np.VisibleDeprecationWarning)

num_wrapper=30
clf=LGBMClassifier()
sfs=SFS(clf, k_features=num_wrapper, forward=True, verbose=2, scoring='accuracy', n_jobs=-1)
sfs.fit(X_trn_scaled[filter_columns],y_train)
```

...

```
sfs.k_feature_names_
```

...

```
pd.DataFrame.from_dict(sfs.get_metric_dict()).T
```

...

```
from mlxtend.plotting import plot_sequential_feature_selection as plot_sfs
fig1=plot_sfs(sfs.get_metric_dict(),
              kind='std_dev',
              figsize=(30,4))

plt.ylim([0,1])
plt.xlim([0,30])
plt.title('SFS (w. std_dev)')
plt.grid()
plt.show()
```

...

```
vars_SBS=pd.DataFrame.from_dict(sfs.get_metric_dict()).T
vars_SBS.to_csv('varsFS.csv',index=False)
```

```
selected = list(vars_SBS.iloc[29,3])
print('# features selected:', len(selected))
selected
```

8.3.5 Regression Model Tuning

Build best models

[34]:
```
labels = []
train_scores = []
test_scores = []
```

[35]:
```
num_cross_val = 10
nvars = len(X_trntst[1])
```

[36]:
```
def clf_score(clf, x_train, y_train, label, train_scores, test_scores, cv=num_cross_val, n_jobs=-1):
    score = cross_validate(clf, x_train, y_train, scoring=None, cv=cv, n_jobs=n_jobs,
                           return_train_score=True, return_estimator=True)
    train_scores.append(score['train_score'])
    test_scores.append(score['test_score'])
    labels.append(label)

    mean_score=pd.DataFrame()
    mean_score.loc[0,'mean_train_score']=np.mean(score['train_score'])
    mean_score.loc[0,'mean_test_score']=np.mean(score['test_score'])
    print(mean_score)
    result=pd.DataFrame()
    result['Train']=score['train_score']
    result['Test']=score['test_score']
    print(result)
```

[ ]:
```
%%time
# Linear regression
# params={'copy_X':False,'fit_intercept':False,'normalize':False}
lr = LinearRegression(fit_intercept=True,copy_X=True,normalize=True,positive=True)
clf_score(lr, X_trntst, Y_trntst, 'LR', train_scores, test_scores, n_jobs=-1)
```

...

[ ]:
```
%%time
# Single decision tree
params={ 'max_depth':None, 'min_samples_leaf':1,'min_samples_split':10,'max_features':None}
dt = DecisionTreeRegressor(**params)
clf_score(dt, X_trntst, Y_trntst, 'DT', train_scores, test_scores, n_jobs=-1)
```

...

[ ]:
```
%%time
# Neural Network
params={'hidden_layer_sizes':(100,100),'alpha':0.001}
nn = MLPRegressor(**params)
clf_score(nn, X_trntst, Y_trntst, 'NN', train_scores, test_scores, n_jobs=-1)
```

...

[ ]:
```
# Boosted tree
params={'n_estimators': 200, 'max_depth': 3, 'min_samples_split': \
        50, 'learning_rate': 0.1, 'loss': 'ls', 'min_samples_leaf': 1,'alpha':0.5}
bt = GradientBoostingRegressor()
clf_score(bt, X_trntst, Y_trntst, 'BT', train_scores, test_scores, n_jobs=-1)
```

...

```
# Random forest
#params={'n_estimators':100, 'max_depth':100, 'max_features':'auto','min_samples_leaf':20}
params={'n_estimators':50, 'max_depth':None, 'max_features':'auto','min_samples_leaf':1}
rf = RandomForestRegressor()
clf_score(rf, X_trntst, Y_trntst, 'RF', train_scores, test_scores, n_jobs=-1)
```

```
# KNN
# params={}
params={'n_neighbors':5}
knn = KNeighborsRegressor(n_neighbors=10,p=1)
clf_score(rf, X_trntst, Y_trntst, 'KNN', train_scores, test_scores, n_jobs=-1)
```

```
%%time
# SVR
# for i in range(1,150,10):
params={"C":9 ,"tol":0.00001}
svr = SVR(**params)
clf_score(svr, X_trntst, Y_trntst, 'SVR', train_scores, test_scores, n_jobs=-1)
# print(f'max_depth :{i}')
```

```
%%time
# I usually comment out this model since it takes a long time to train and doesn't work that well
# xgboost 'eta': 0.05, 'lambda': 0.3, 'max_depth': 9
params={}  # 确定 eta:0.05 "lambda":0.5
xg = xgb.XGBRegressor(**params)
clf_score(xg, X_trntst, Y_trntst, 'XGB', train_scores, test_scores, n_jobs=-1)
```

```
%%time
# default
# I usually comment out this model since it takes a long time to train and doesn't work that well
# xgboost
params={}
xg = xgb.XGBRegressor(**params)
clf_score(xg, X_trntst, Y_trntst, 'XGB', train_scores, test_scores, n_jobs=-1)
```

```
%%time
# lightgbm
params={'learning_rate':0.1, 'max_depth':20,'min_child_samples':2}
lg = lgb.LGBMRegressor(**params)
clf_score(lg, X_trntst, Y_trntst, 'LGB', train_scores, test_scores, n_jobs=-1)
```

8.3.6 Classification Model Tuning

Build best models

In [66]:
```
labels = []
train_scores = []
test_scores = []
```

In [67]:
```
num_cross_val = 15
nvars = len(X_trntst[1])
```

In [84]:
```
def clf_score(clf, x_train, y_train, label, train_scores, test_scores, cv=num_cross_val, n_jobs=-1):
    from sklearn.model_selection import StratifiedKFold
    kfolds = StratifiedKFold(n_splits = 15, shuffle = True)
    score = cross_validate(clf, x_train, y_train, scoring=None, cv=kfolds, n_jobs=n_jobs,
                           return_train_score=True, return_estimator=True)
    train_scores.append(score['train_score'])
    test_scores.append(score['test_score'])
    labels.append(label)
    print(np.mean(score['train_score']), np.mean(score['test_score']))
    print(pd.DataFrame(score['test_score'],score['train_score']))
```

In [96]:
```
%%time
# Logistic regression
# params={'copy_X':False,'fit_intercept':False,'normalize':False}
lr = LogisticRegression(penalty='l2', solver='lbfgs')
clf_score(lr, X_trntst, Y_trntst, 'LR', train_scores, test_scores, n_jobs=-1)
```

...

In [102]:
```
%%time
# Single decision tree
# for i in range(1,150,10):
params={'max_depth':50, 'min_samples_leaf':20,'min_samples_split':30,'max_features':'auto'}
dt = DecisionTreeClassifier(max_depth=15, splitter='best')
clf_score(dt, X_trntst, Y_trntst, 'DT', train_scores, test_scores, n_jobs=-1)
# print(f'max_depth :{i}')
```

...

In [104]:
```
%%time
# Neural Network
params={}
nn = MLPClassifier(hidden_layer_sizes=(20,2), learning_rate='constant', activation='logistic')
clf_score(nn, X_trntst, Y_trntst, 'NN', train_scores, test_scores, n_jobs=-1)
```

...

In [112]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda=LDA(n_components=1, solver='svd', shrinkage=None)
clf_score(lda, X_trntst, Y_trntst, 'NN', train_scores, test_scores, n_jobs=-1)
```

...

In [114]:
```
%%time
# Random forest
# params={'n_estimators':100, 'max_depth':100, 'max_features':'auto','min_samples_leaf':20}
params={}
rf = RandomForestClassifier(n_estimators=1000,max_depth=50,min_samples_leaf=5,min_samples_split=10)
clf_score(rf, X_trntst, Y_trntst, 'RF', train_scores, test_scores, n_jobs=-1)
```

...

In [180]:
```
##### %%time
# Boosted tree
params={}
bt = LGBMClassifier(n_estimators=600, max_depth=50, learning_rate=0.01)
clf_score(bt, X_trntst, Y_trntst, 'BT', train_scores, test_scores, n_jobs=-1)
```

...

8.4 Selected Feature Subset-2

8.4.1 Classification

```
selected=['S-K',
 'S-K/r_sqrt',
 'S-K/S_sqrt',
 'K/S',
 'S-K/tau_sqrt',
 'S-K/r_sqrt^tau',
 'K/K_sqrt',
 'K',
 'K_sqrt',
 'K/r',
 'K_sqrt/r_sqrt',
 'K_sqrt^tau/tau',
 'K_sqrt^tau/r',
 'K_sqrt^tau/S',
 'K_sqrt/tau_sqrt',
 'K/tau',
 'K/K_sqrt^tau',
 'K^tau/S',
 'K_sqrt^tau/S_sqrt',
 'K_sqrt/tau',
 'K^tau/K_sqrt',
 'tau_sqrt',
 'K_sqrt/S_sqrt^tau',
 'K/S^tau',
 'r_sqrt/tau_sqrt',
 'K_sqrt/S^tau',
 'S_sqrt^tau/r^tau',
 'r_sqrt^tau/tau',
 'r_sqrt^tau/tau_sqrt',
 'domain']
```

8.4.2 Regression

```
selected=['K/S--*--r_exp/r_exp^tau_poly',
 'S-K/tau_exp--*--K_log^tau/r_log_poly',
 'S-K/tau_exp--*--S_log^tau/r_log_poly',
 'S-K/tau_exp--*--K_log^tau/K_sqrt_poly',
 'S-K/S_sqrt--*--K_log^tau/tau_exp_poly',
 'S-K/tau_exp--*--S_log^tau/S_sqrt_poly',
 'tau_exp--*--S-K/r_log_poly',
 'tau_exp--*--S-K/S_sqrt_poly',
 'K/S^tau--*--K^tau/S_poly',
 'K/S--*--K^tau/S^tau_poly',
 'S-K/tau_exp--*--K_log^tau/S_poly',
 'K_log/S_log--*--K_log^tau/S_log^tau_poly',
 'S-K/tau_exp--*--K_log^tau/r_poly',
 'S-K/r--*--S_log^tau/tau_exp_poly',
 'tau_exp--*--S-K/r_poly',
 'K_sqrt/S^tau--*--K^tau/S_sqrt_poly',
 'K_log/S_log--*--r_exp/r_exp^tau_poly',
 'S-K/r_exp^tau--*--K_log^tau/r_exp^tau_poly',
 'S-K/r_exp^tau--*--K_log^tau/K_sqrt_poly',
 'S-K/r_exp^tau--*--S_log^tau/r_exp^tau_poly',
 'S-K/S_log^tau--*--r_exp/tau_log_poly',
 'S-K/tau_log--*--S_log/S_log^tau_poly',
 'S-K/tau_log--*--K_log/S_log^tau_poly',
 'S-K/S_log^tau--*--S_sqrt/r^tau_poly',
 'S-K/tau_log--*--K_log/K_log^tau_poly',
 'S-K--*--S_log^tau/r_exp^tau_poly',
 'K_log^tau--*--S-K/r_exp^tau_poly',
 'S-K/r_exp^tau--*--K_log^tau/S_log_poly',
 'S-K/S_log--*--S_log^tau/r_exp^tau_poly',
 'S-K/S_log^tau--*--S_log/r^tau_poly',
 'S-K/S_log^tau--*--r_exp^tau/tau_log_poly']
```