PSX MARKET VOLUME ANALYIS & PREDICTIONS

JUNE 14, 2021

MUHAMMAD AMIN GHIAS

Table of Contents

ABSTRACT:

Stock Exchange is an important aspect in the financial growth of a company. Many people invest their money in stocks. This project will try to predict the volume of the stock market using our dataset

INTRODUCTION

People invest their time and money in the stock market it is important to know about the stability of the stock market. This project will predict the market volume using the different features in the data set

BUSINESS PROBLEM & MAIN OBJECTIVE

As the stock market can vary so people like to know whether the market is stable and their investment is safe. For this purpose we will study the market volume and predict the market volume using Machine Learning Regression techniques.

DATA ACQUISITION AND WRANGLING

DATA SOURCE

For this project we will use online data the source is from kaggle

CSV file with url:" https://www.kaggle.com/zusmani/pakistan-stock-exchange-kse-100?select=Stock+Exchange+KSE+100%28Pakistan%29.csv"

The Dataset is of Karachi Stock Exchange (KSE) 100 Index from 2008 to 2021, following is the dataframe which consists of 7 columns and 3221 rows

```
In [83]: data.shape
Out[83]: (3221, 7)
```

DATA DESCRIPTION

The description of columns are:

- Date: Calendar date
- Open: KSE 100 opening index points on that date
- High: KSE 100 highest index points on that date
- Lowe: KSE 100 lowest index points on that date
- Close: KSE 100 closing index points on that date
- Change: KSE 100 change in index points on that date
- Volume: Total Market volume

In [3]:
```
data.head(20)
```
Out[3]:

	Date	Open	High	Low	Close	Change	Volume
0	23-Feb-21	31,722.16	31,800.90	31,597.31	31,626.19	-21.38	718,191,025
1	22-Feb-21	31,874.78	31,958.58	31,612.55	31,647.57	-203.61	721,952,658
2	19-Feb-21	31,748.75	31,904.30	31,749.43	31,851.18	91.36	694,795,084
3	18-Feb-21	32,049.85	32,104.67	31,745.72	31,759.82	-288.86	577,837,595
4	17-Feb-21	32,166.21	32,390.77	32,044.01	32,048.68	-93.15	701,658,181
5	16-Feb-21	31,898.51	32,155.42	31,891.80	32,141.83	250.03	514,044,525
6	15-Feb-21	31,695.01	31,959.00	31,552.69	31,891.80	339.11	486,340,423
7	12-Feb-21	31,703.25	31,819.14	31,517.21	31,552.69	-151.98	442,547,019
8	11-Feb-21	32,121.31	32,255.15	31,653.40	31,704.67	-410.7	1,124,724,205
9	10-Feb-21	32,186.99	32,234.98	31,949.20	32,115.37	-51.32	1,011,825,950
10	9-Feb-21	32,275.59	32,365.63	32,149.63	32,166.69	-80.25	663,958,819
11	8-Feb-21	32,393.94	32,478.42	32,185.62	32,246.94	-139.15	428,332,082
12	4-Feb-21	32,358.58	32,602.03	32,302.37	32,386.09	27.13	440,244,554
13	3-Feb-21	32,187.71	32,451.23	32,167.97	32,358.96	190.99	616,200,447

In [8]:
```
data.count()
```
Out[8]:
```
Date      3221
Open      3221
High      3221
Low       3221
Close     3221
Change    3221
Volume    3221
dtype: int64
```

In [9]:
```
data.describe()
```
Out[9]:

	Date	Open	High	Low	Close	Change	Volume
count	3221	3221	3221	3221	3221	3221	3221
unique	3221	3213	3215	3211	3209	3104	3220
top	21-Apr-11	29269.00	6641.78	6639.00	6641.77	0	302831072
freq	1	2	2	3	3	5	2

```
In [10]: data.dtypes
Out[10]: Date      object
         Open      object
         High      object
         Low       object
         Close     object
         Change    object
         Volume    object
         dtype: object
```

DATA CLEANING

The CSV dataset contains " , " and " " which have been removed

```
In [6]: # Removing the symbols in the Data

        df.replace(',','', regex=True, inplace=True)
```

```
In [7]: df.head()
```

Out[7]:

	Date	Open	High	Low	Close	Change	Volume
0	23-Feb-21	31722.16	31800.90	31597.31	31626.19	-21.38	718191025
1	22-Feb-21	31874.78	31958.58	31612.55	31647.57	-203.61	721952658
2	19-Feb-21	31748.75	31904.30	31749.43	31851.18	91.36	694795084
3	18-Feb-21	32049.85	32104.67	31745.72	31759.82	-288.86	577837595
4	17-Feb-21	32166.21	32390.77	32044.01	32048.68	-93.15	701658181

The X and Y data set has been made from dataframe also the date column has been removed, furthermore the datatype has been changed to float of X and Y dataframe.

The target variable is the market volume which we will be predicting in our Analysis

```
In [11]: #  Formimng the X and Y dataframes

         feature_cols = [x for x in df.columns if x != 'Volume']
         X = df[feature_cols]
         y = df['Volume']



         # Dropping the date coulum from dataframe
         X=X.drop('Date', axis=1)
```

```
In [12]: #  Converting Datatype into float

         X = X.astype(float)
         y = y.astype(float)
```

```
In [13]: X.head()
```

Out[13]:

	Open	High	Low	Close	Change
0	31722.16	31800.90	31597.31	31626.19	-21.38
1	31874.78	31958.58	31612.55	31647.57	-203.61
2	31748.75	31904.30	31749.43	31851.18	91.36
3	32049.85	32104.67	31745.72	31759.82	-288.86

DATA ANALYSIS & USE OF REGRESSION MODELS:

Initially simple Linear Regression is used and data is split into test and train sets for the analysis,

The predicted Y (Market Volume) is checked for errors with train and test sets both

In [14]:
```
# Making test and train sets of data

X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.3, random_state=42)
```

In [15]:
```
# Applying Linear Regaression

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

LR = LinearRegression()

# Storage for error values
error_df = list()

# Data that have not been one-hot encoded
LR = LR.fit(X_train, y_train)
y_train_pred = LR.predict(X_train)
y_test_pred = LR.predict(X_test)

error_df.append(pd.Series({'train': mean_squared_error(y_train, y_train_pred),
                           'test' : mean_squared_error(y_test,  y_test_pred)},
                           name='Mean Square Error'))


error_df = pd.concat(error_df, axis=1)
error_df
```

Out[15]:

	Mean Square Error
train	1.417859e+16
test	1.368677e+16

For Linear Regression the MSE is nearly same for training and testing set

SCALING EFFECT ON DATA

Similarly the StandardScaler, MinMaxScaler, MaxAbsScaler was applied on data and then linear regression applied, the results are

```
- standardscaling    1.368677e+16
- minmaxscaling      1.368677e+16
- maxabsscaling      1.368677e+16
```

No big difference by using transformation so it has no significant effect.

K-FOLDS TECHNIQUE:

The K-fold method is used to split data in 3 parts 2 for training and 1 for test also random samples of test and train sets are taken

```
In [21]: kf = KFold(shuffle=True, random_state=72018, n_splits=3)
```

```
In [22]:
for train_index, test_index in kf.split(X):
    print("Train index:", train_index[:10], len(train_index))
    print("Test index:",test_index[:10], len(test_index))
    print('')
```

```
Train index: [ 0  2  3  4  5  7  8 10 11 12] 2147
Test index: [ 1  6  9 15 17 18 21 22 23 30] 1074

Train index: [ 0  1  4  6  7  8  9 10 11 12] 2147
Test index: [ 2  3  5 14 16 19 20 24 26 27] 1074

Train index: [ 1  2  3  5  6  9 14 15 16 17] 2148
Test index: [ 0  4  7  8 10 11 12 13 25 28] 1073
```

Linear Regression on different combination of the kfolds done to see it effects

```
In [23]: #from sklearn.metrics import r2_score, mean_squared_error

         scores = []
         lr = LinearRegression()

         for train_index, test_index in kf.split(X):
             X_train, X_test, y_train, y_test = (X.iloc[train_index, :],
                                                 X.iloc[test_index, :],
                                                 y[train_index],
                                                 y[test_index])

             lr.fit(X_train, y_train)

             y_pred = lr.predict(X_test)

             score = r2_score(y_test.values, y_pred)

             scores.append(score)

         scores

Out[23]: [0.24403273215319998, 0.20678344963496753, 0.21933033348969955]
```

Using Kfold random sampling shows that mean square error is same so we can use any train and test set from dataframe

Now scalar transformation was applied on the same above code the following result came

```
[0.24403273215319998, 0.20678344963496842, 0.21933033348969977]
```

Applying statndar scalar transform on Linear regression on the random train test sets shows that mean square error is same

Using Combine multiple processing steps into a Pipeline and using cross_val_predict

In [25]:
```
s = StandardScaler()
lr = LinearRegression()
```

In [26]:
```
estimator = Pipeline([("scaler", s),
                      ("regression", lr)])
```

In [27]:
```
kf
```

Out[27]:
```
KFold(n_splits=3, random_state=72018, shuffle=True)
```

In [28]:
```
predictions = cross_val_predict(estimator, X, y, cv=kf)
```

In [29]:
```
r2_score(y, predictions)
```

Out[29]:
```
0.22443001302912113
```

In [30]:
```
np.mean(scores) # almost identical!
```

Out[30]:
```
0.2233821717592894
```

Hyper parameter tuning applied

LASSO REGRESSION

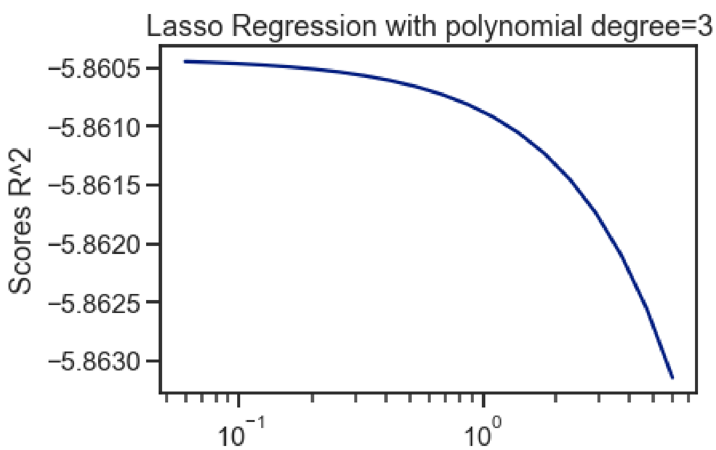
In [32]:
```
# Using Lasso Regression

scores = []
coefs = []
for alpha in alphas:
    las = Lasso(alpha=alpha, max_iter=100000)

    estimator = Pipeline([
        ("scaler", s),
        ("lasso_regression", las)])

    predictions = cross_val_predict(estimator, X, y, cv = kf)

    score = r2_score(y, predictions)

    scores.append(score)
```

Its result was

In [33]:
```
list(zip(alphas,scores))
```

Out[33]:
```
[(1e-09, 0.2244655197390277З),
 (1e-08, 0.22446551973902695),
 (1e-07, 0.22446551973902806),
 (1e-06, 0.22446551973909867),
 (1e-05, 0.22446551973980367),
 (0.0001, 0.22446551974677964),
 (0.001, 0.224465519816429),
 (0.01, 0.22446552051332114),
 (0.1, 0.22446552748201365),
 (1.0, 0.22446559715736347)]
```

Also further analysis done:



RIDGE REGRESSION:

In [43]:
```
pf = PolynomialFeatures(degree=2)
alphas = np.geomspace(4, 20, 20)
scores=[]
for alpha in alphas:
    ridge = Ridge(alpha=alpha, max_iter=100000)

    estimator = Pipeline([
        ("scaler", s),
        ("polynomial_features", pf),
        ("ridge_regression", ridge)])

    predictions = cross_val_predict(estimator, X, y, cv = kf)
    score = r2_score(y, predictions)
    scores.append(score)

plt.plot(alphas, scores)
plt.xlabel('Alpha')
plt.ylabel('Scores    R^2');
plt.title("Ridge Regression")
```

Out[43]:
```
Text(0.5, 1.0, 'Ridge Regression')
```

Finding which features have the most impact on the market volume by knowing the values of coefficients of

For Lasso Regression:

```
In [52]: df_importances.sort_values(by=1)
Out[52]:
```

	0	1
6	Open^2	-3.532522e+08
3	Low	-1.288844e+08
19	Close Change	-1.114819e+08
18	Close^2	-6.743895e+07
16	Low Close	-5.280444e+07
13	High Close	-5.010651e+07
9	Open Close	-4.969601e+07
1	Open	-3.538156e+06
20	Change^2	-2.472802e+04
0	1	0.000000e+00
4	Close	1.247580e+07
14	High Change	1.417423e+07
5	Change	2.461047e+07
10	Open Change	3.867063e+07
17	Low Change	5.060702e+07
11	High^2	6.493485e+07
12	High Low	1.094239e+08
8	Open Low	1.108216e+08
7	Open High	1.244567e+08
15	Low^2	1.692748e+08
2	High	1.786853e+08

For Ridge regression:

```
In [56]: df_importances_r.sort_values(by=1)
Out[56]:
```

	0	1
1	Open	-1.619891e+09
4	Close	-5.347810e+08
6	Open^2	-3.473548e+08
9	Open Close	-2.920707e+08
18	Close^2	-2.405514e+08
19	Close Change	-2.380289e+08
13	High Close	-1.601867e+08
14	High Change	-1.547836e+08
3	Low	-1.430218e+08
7	Open High	-9.016008e+07
10	Open Change	-7.440822e+07
11	High^2	-4.801754e+07
0	1	0.000000e+00
20	Change^2	2.165977e+04
5	Change	2.557228e+07
8	Open Low	6.482534e+07
16	Low Close	1.244641e+08
12	High Low	3.446025e+08
17	Low Change	4.554139e+08
15	Low^2	6.494193e+08
2	High	2.355813e+09

Both of these results show that **high (highest index points)** has the most effect on market volume as their coefficients are high

USE OF GRIDSEARCH:

Ridge regression:

In [57]:
```
from sklearn.model_selection import GridSearchCV

# Same estimator as before
estimator = Pipeline([("scaler", StandardScaler()),
        ("polynomial_features", PolynomialFeatures()),
        ("ridge_regression", Ridge())])

params = {
    'polynomial_features__degree': [1, 2, 3, 4, 5],
    'ridge_regression__alpha': np.geomspace(4, 20, 30)
}

grid = GridSearchCV(estimator, params, cv=kf)
```

In [58]:
```
grid.fit(X, y)
```

Best value of alpha and polynomial degree found for ridge regression

In [59]:
```
grid.best_score_, grid.best_params_
```

Out[59]:
```
(0.19489532377404703,
 {'polynomial_features__degree': 1, 'ridge_regression__alpha': 4.0})
```

In [60]:
```
y_predict = grid.predict(X)
```

In [61]:
```
r2_score(y, y_predict)
```

Out[61]:
```
0.20135460445629172
```

In [62]:
```
# Notice that "grid" is a fit object!
# We can use grid.predict(X_test) to get brand new predictions!
grid.best_estimator_.named_steps['ridge_regression'].coef_
```

Out[62]:
```
array([       0.        , -63510858.63213889,  90954319.01469189,
         6888468.9437722 ,  23508791.91170569,  15987416.79210745])
```

Lasso Regression:

```
In [64]: from sklearn.model_selection import GridSearchCV

         # Same estimator as before
         estimator = Pipeline([("scaler", StandardScaler()),
                 ("polynomial_features", PolynomialFeatures()),
                 ("lasso_regression", Lasso())])

         params = {
             'polynomial_features__degree': [1, 2, 3, 4, 5],
             'lasso_regression__alpha': np.geomspace(0.06, 6.0, 20)
         }

         grid = GridSearchCV(estimator, params, cv=kf)
```

```
In [65]: grid.fit(X, y)
```

```
In [66]: grid.best_score_, grid.best_params_

Out[66]: (0.19611916692073283,
          {'lasso_regression__alpha': 0.06, 'polynomial_features__degree': 1})

In [67]: y_predict = grid.predict(X)

In [68]: r2_score(y, y_predict)

Out[68]: 0.2023489672011689

In [69]: # Notice that "grid" is a fit object!
         # We can use grid.predict(X_test) to get brand new predictions!
         grid.best_estimator_.named_steps['lasso_regression'].coef_

Out[69]: array([ 0.00000000e+00, -5.86823822e+07,  1.93432295e+08, -9.59130367e+07,
                 1.90308222e+07,  1.63233941e+07])
```

Best value of alpha and polynomial degree found for Lasso regression

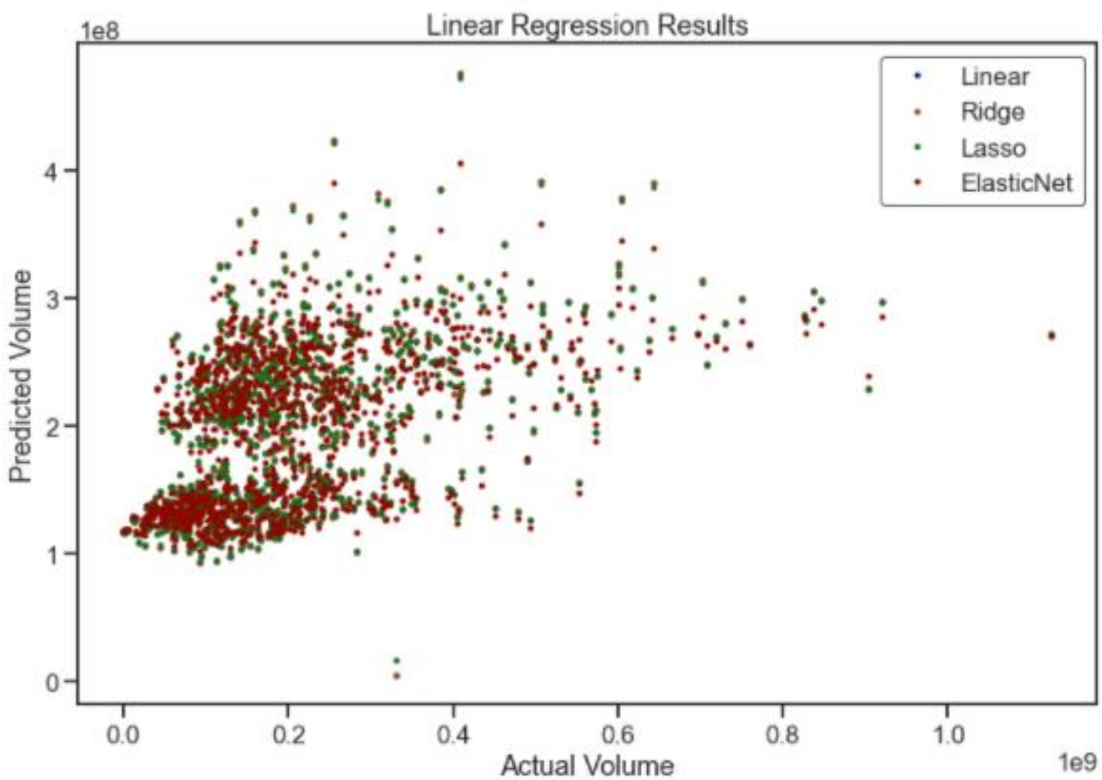
REGULARIZATION REGRESSION:

In [80]:

```
rmse_vals = [linearRegression_rmse, ridgeCV_rmse, lassoCV_rmse, elasticNetCV_rmse]

labels = ['Linear', 'Ridge', 'Lasso', 'ElasticNet']

rmse_df = pd.Series(rmse_vals, index=labels).to_frame()
rmse_df.rename(columns={0: 'RMSE'}, inplace=1)
rmse_df
```

Out[80]:

	RMSE
Linear	1.252523e+08
Ridge	1.252523e+08
Lasso	1.252488e+08
ElasticNet	1.261176e+08



Again normal linear Regression , ridge and lasso regression done following are the results:

```
r2 score of Lasso for alpha = 0.001: -2.9047859025462057
r2 score of Ridge for alpha = 0.001: -2.920382127071007
r2 score for Linear Regression: -2.904176597551378
Magnitude of Lasso coefficients: 7746492830.174958
Number of coeffients not equal to 0 for Lasso: 5
Magnitude of Ridge coefficients: 7610526671.31369
Number of coeffients not equal to 0 for Ridge: 5
Magnitude of Linear Regression coefficients: 7751405540.6519375
Number of coeffients not equal to 0 for Linear Regression: 5
```

RESULTS:

DISCUSSION:

We used normal linear regression, utilized the scaling transformation to see what effect it had on our result, we further used K-Fold, Cross validation, then for regularization Lasso and Ridge Regression was also used, we analyzed and studied all the results to come up with a decision

CONCLUSION

After applying differed regression methods and doing many analysis we found out that for our dataset **normal linear regression with polynomial degree 1 is best for our data analysis and predicting the KSE 100 index market volume**

The feature which had the most effect on market volume prediction is the highest index of the day as we found out in our analysis.

SUGGESTIONS

The dataset has few features and Ridge, Lasso Regression and scaling had nearly no effect on our analysis. More features will help to make the analysis better add a bit more complexity and reduce the error also