fs_draft_070919

March 16, 2023

1 Assignment 1: Asset Price Prediction

```
In [1]: import pandas as pd
        import matplotlib.pyplot as plt
        import seaborn as sns

        from sklearn.preprocessing import MinMaxScaler
        import numpy as np
        from sklearn.decomposition import PCA
        from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
        from sklearn.feature_selection import SelectKBest
        from sklearn.feature_selection import chi2
        from sklearn.feature_selection import RFE
        from sklearn.feature_selection import f_regression
        from sklearn.linear_model import LogisticRegression
        from sklearn.ensemble import RandomForestRegressor
        from sklearn.linear_model import RandomizedLasso
        from sklearn import feature_selection
        from sklearn.model_selection import train_test_split
        from sklearn.linear_model import LinearRegression
        import statsmodels.api as sm
```

1.1 Feature Selection using market data

This notebook will provide the necessary steps at performing Feature Selection using stock market data. Feature selection uses an automated process to select the best features in a data set to be used for optimizing prediction.

```
In [2]: # Modeling
```

```
In [3]: # assign file and open csv using pandas
        File_to_process = "Sample.csv"

        # It appears that feature 1 is just all 0's, maybe they were text at one point? Feature
        columns = ['Trade Date','Close_x','Close_y','Feature 2','Feature 3','Feature 4','Feature

        # We are trying to predict the Label classifier
        # I understand that you want to remove outliers, but how do we know which one's to remov
```

```
# Removing outliers, is this just standard cutting winsorizing? Just have to be careful
remove_outliners = True
lower_outliner = 0.015
upper_outliner = 0.98

#PCA type: "By dimensions", "By variance"
pca_type = "by_variance"

#if "By_dimensions", specify the number of dimensions, if "by_variance" specify the vari
pca_value = 0.75

add_dimentions_over =0.09
```

Descriptive statistics here to understand what the data will look like then we can data cleaning properly. I will first subset the Features into groups of 3.

1.2 Global Functions

```
In [4]: def rank_to_dict(ranks, names, order=1):
            minmax = MinMaxScaler()
            ranks = minmax.fit_transform(order*np.array([ranks]).T).T[0]
            ranks = map(lambda x: round(x, 2), ranks)
            return dict(zip(names, ranks ))

        def bring_features(indices):
            features = []
            for i in range(total_columns):
                features.append(df.columns[indices[i]])
            return features
```

```
In [5]: def biplot(df, reduced_data, pca):

            fig, ax = plt.subplots(figsize = (12,8))

            # scatterplot of the reduced data
            ax.scatter(x=reduced_data.loc[:, 'PC-1'], y=reduced_data.loc[:, 'PC-2'], facecolors=

            feature_vectors = pca.components_.T

            print(feature_vectors)

            # using scaling factors to make the arrows
            arrow_size, text_pos = 7.0, 8.0,

            # projections of the original features
            for i, v in enumerate(feature_vectors):
                ax.arrow(0, 0, arrow_size*v[0], arrow_size*v[1], head_width=0.2, head_length=0.2
```

```
        ax.text(v[0]*text_pos, v[1]*text_pos, df.columns[i], color='black', ha='center',

    ax.set_xlabel("principal component 1", fontsize=14)
    ax.set_ylabel("principal component 2", fontsize=14)
    ax.set_title("PC plane with original feature projections.", fontsize=16);
    return ax
```

```
In [6]: def show_pca_chart():
    with plt.style.context('seaborn-whitegrid'):
        plt.figure(figsize=(8, 6))
        for lab, col in zip((-2, 0, 2),
                            ('blue', 'red', 'green')):
            plt.scatter(X_sklearn[y==lab, 0],
                        X_sklearn[y==lab, 1],
                        label=label_dict[lab],
                        c=col)
        plt.xlabel('Principal Component 1')
        plt.ylabel('Principal Component 2')
        plt.legend(loc='lower center')
        plt.tight_layout()

        plt.show()
```

```
In [7]: def pca_results(data, pca):

    # Dimension indexing
    dimensions = ['Dimension {}'.format(i) for i in range(1,len(pca.components_)+1)]

    # PCA components
    components = pd.DataFrame(np.round(np.abs(pca.components_), 4), columns = data.keys(
    components.index = dimensions

    # PCA explained variance
    ratios = pca.explained_variance_ratio_.reshape(len(pca.components_), 1)

    variance_ratios = pd.DataFrame(np.round(ratios, 4), columns = ['Explained Variance']
    variance_ratios.index = dimensions

    # Create a bar plot visualization
    fig, ax = plt.subplots(figsize = (14,8))

    # Plot the feature weights as a function of the components
    components.plot(ax = ax, kind = 'bar')
    ax.set_ylabel("Feature Weights")
    ax.set_xticklabels(dimensions, rotation=0)

    # Display the explained variance ratios#
    for i, ev in enumerate(pca.explained_variance_ratio_):
```

```
        ax.text(i-0.40, ax.get_ylim()[1] + 0.05, "Explained Variance\n %.4f"%(ev))

    # Return a concatenated DataFrame
    return  pd.concat([variance_ratios, components], axis = 1)
```

```
In [8]: def plot_scikit_lda(X, title):

    ax = plt.subplot(111)

    for label,marker,color in zip(
        (-2,0, 2),('^', 's', 'o'),('blue', 'red', 'green')):

        plt.scatter(X[y==label, 0],
            X[y==label, 1], # flip the figure
            marker=marker,
            color=color,
            alpha=0.5,
            label=label_dict[label])

    plt.xlabel('LD1')
    plt.ylabel('LD2')

    leg = plt.legend(loc='upper right', fancybox=True)
    leg.get_frame().set_alpha(0.5)
    plt.title(title)

    # hide axis ticks
    plt.tick_params(axis="both", which="both", bottom=False, top=False,
            labelbottom=True, left=False, right=False, labelleft=True)

    # remove axis spines
    ax.spines["top"].set_visible(False)
    ax.spines["right"].set_visible(False)
    ax.spines["bottom"].set_visible(False)
    ax.spines["left"].set_visible(False)

    plt.grid()
    plt.tight_layout
    plt.show()
```

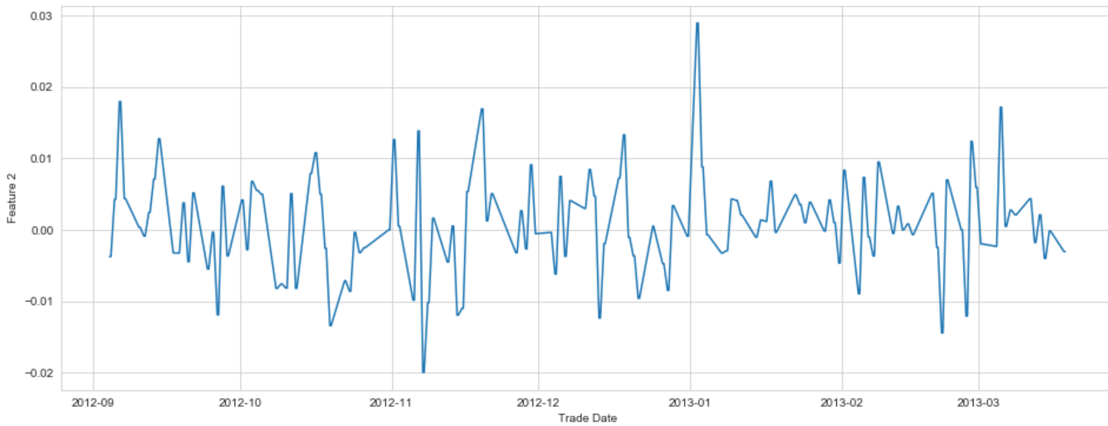
```
In [9]: def plot_selectKbest(selector):
    indices = np.argsort(selector.scores_)[::-1]

    # To get your top 10 feature names
    features = []
    for i in range(total_columns):
        features.append(df.columns[indices[i]])
```

```
# Now plot
plt.figure(figsize=(16, 6))
plt.bar(features, selector.scores_[indices[range(total_columns)]], color='r', align=
plt.xticks(rotation=90)
plt.show()
```

2 Data Reading

```
In [10]: #Global settings
         sns.set_style('whitegrid')
         np.random.seed(0)
         ranks = {}

         # load dataset into Pandas DataFrame
         df = pd.read_csv(File_to_process,parse_dates=True)


         df = df[columns]
         df.set_index(pd.DatetimeIndex(df['Trade Date']), inplace=True)
         df.sort_index()


         plt.figure(figsize=(16, 6))
         sns.lineplot(df.index, df['Feature 2'])

         df.dropna(how="all", inplace=True) # drops the empty line at file-end
         df.drop(['Trade Date'], axis=1, inplace=True)
         df.reset_index(drop=True, inplace=True)
```

```
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\pandas\plotting\_converter.py:129: Fut

To register the converters:
        >>> from pandas.plotting import register_matplotlib_converters
        >>> register_matplotlib_converters()
  warnings.warn(msg, FutureWarning)
```

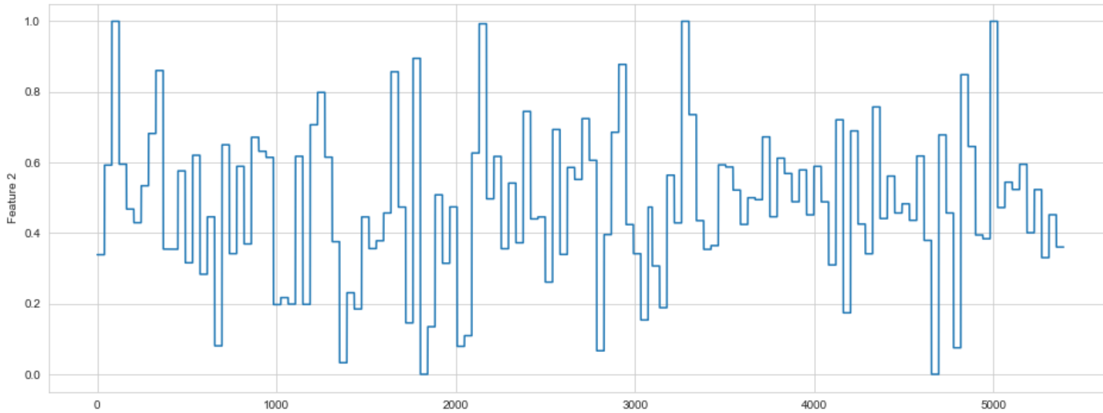


In [11]:
```
# split data table into data X and class labels y
total_columns = len(df.columns) -1

names = list(df.iloc[:,0:total_columns].columns)

x = df.iloc[:,0:total_columns].values
y = df.iloc[:,total_columns].values

label_dict = {-2: '-2', 2: '2',0: '0'}

#Using MinMaxScaler because I have negative numbers and some of the models won's accept
scaler = MinMaxScaler()

x = pd.DataFrame(scaler.fit_transform(x))

#PCA is sensitive to outliers so we winsorize the data at the 1.5% and 98% quantiles, r
#If you want to remove the outliner, comment this line.

if(remove_outliners):
    x = x.clip(lower=x.quantile(q=lower_outliner), upper=x.quantile(q=upper_outliner),

X_std = scaler.fit_transform(x)
```

In [12]:
```
df_c = pd.DataFrame(X_std, columns=names)
```

In [13]:
```
plt.figure(figsize=(16, 6))
sns.lineplot(df_c.index, df_c['Feature 2'] )
#plt.title("Plotting close_x after scaling")
```

Out[13]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x28804f492b0>
```



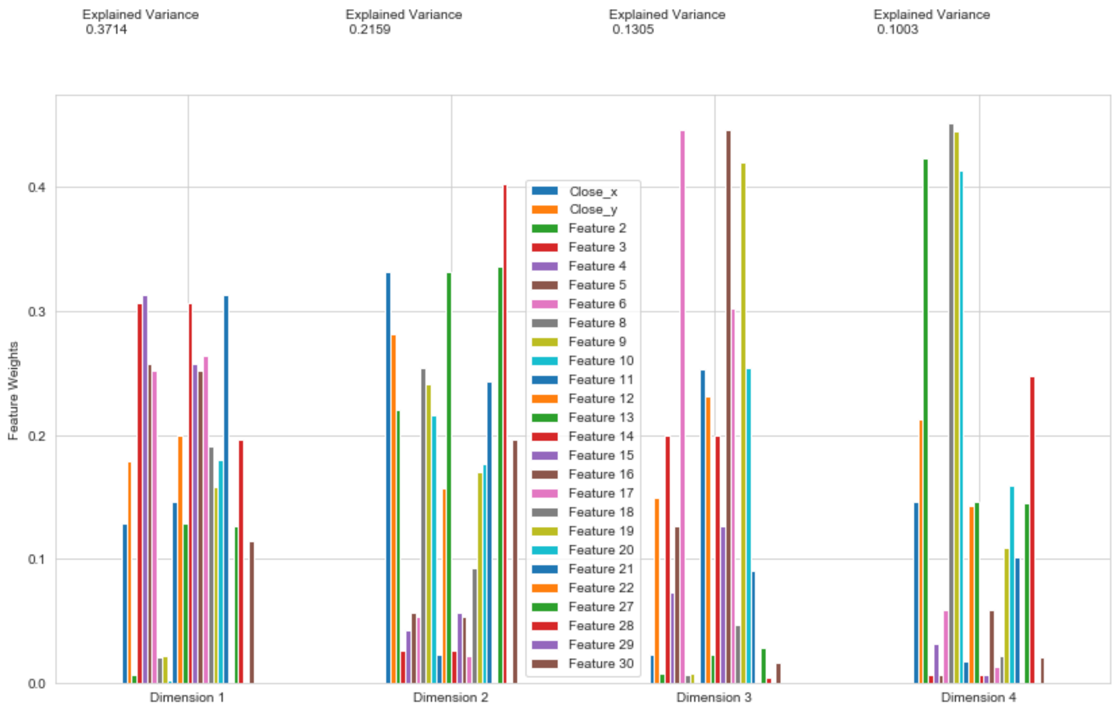
2.1 Principle Component Analysis (PCA)

- PCA is an unsupervised learning algorithm, that ignores labels, and minimizes the variance in the dataset with the best components "dependent variable"-- would this be the best method if we are trying to predict label?

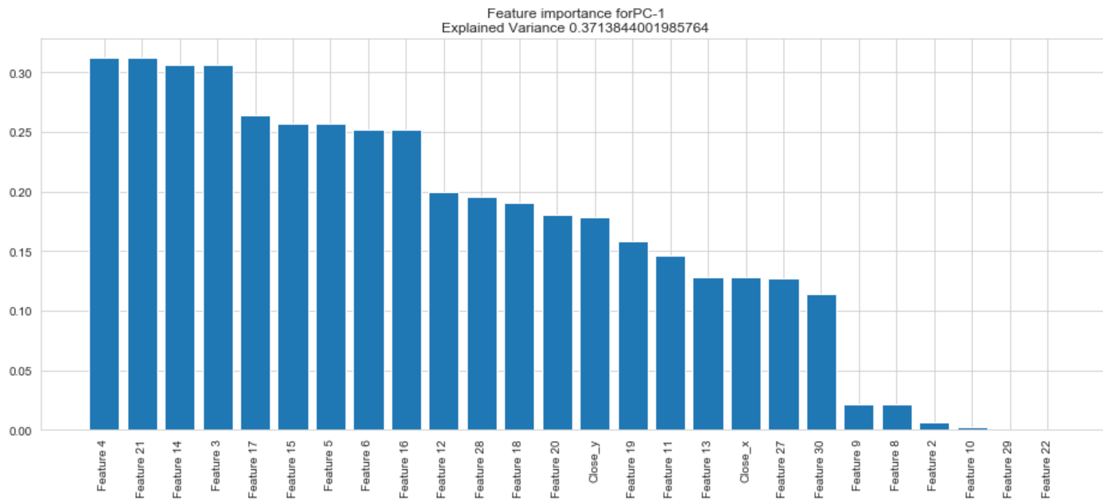
```
In [14]: if(pca_type == "by_dimention"):
             pca = PCA(n_components= pca_value, whiten=True)
         else:
             pca = PCA(pca_value)

         X_pca = pca.fit_transform(X_std)

         # Dimension indexing
         dimensions = ['PC-{}'.format(i) for i in range(1,len(pca.components_)+1)]

         principalDf = pd.DataFrame(data = X_pca, columns = dimensions)

         df_pca = pd.DataFrame(pca.components_, columns=names, index = dimensions)

         print("Explained Variance Ratio: ")
         print(pca.explained_variance_ratio_)
```

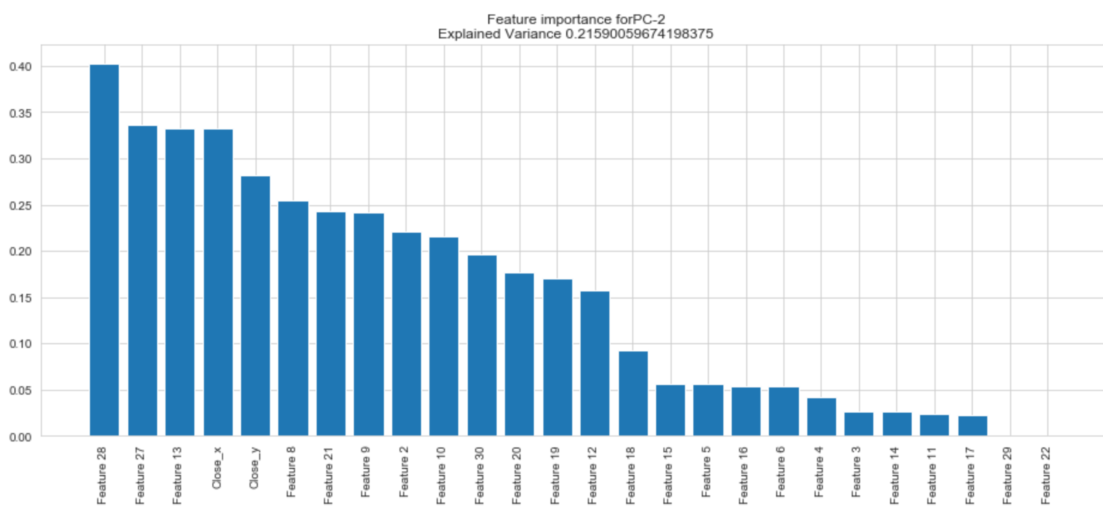
```
Explained Variance Ratio:
[0.3713844  0.2159006  0.13052009 0.10034216]
```

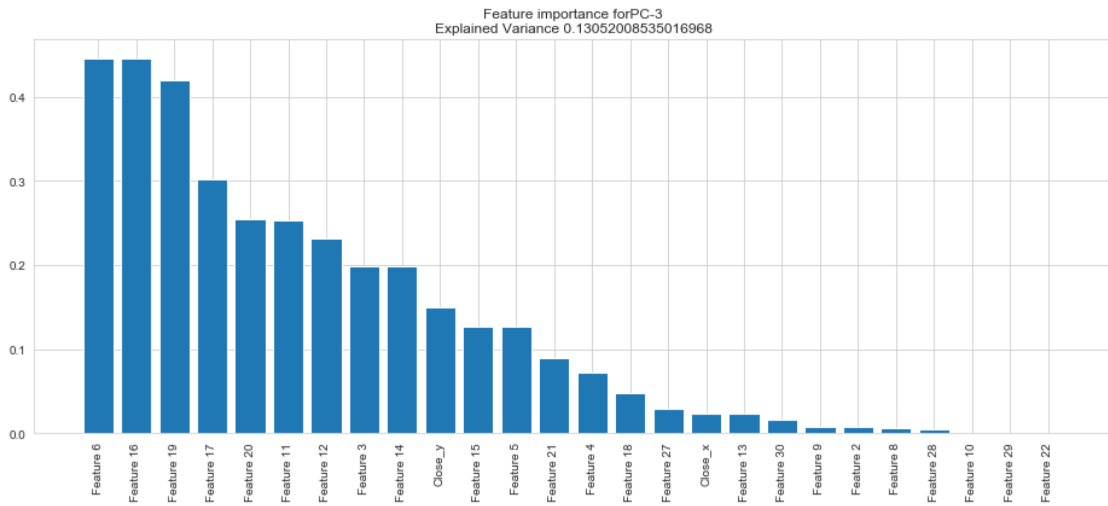
```
In [15]: pca_results = pca_results(df_pca, pca)
```

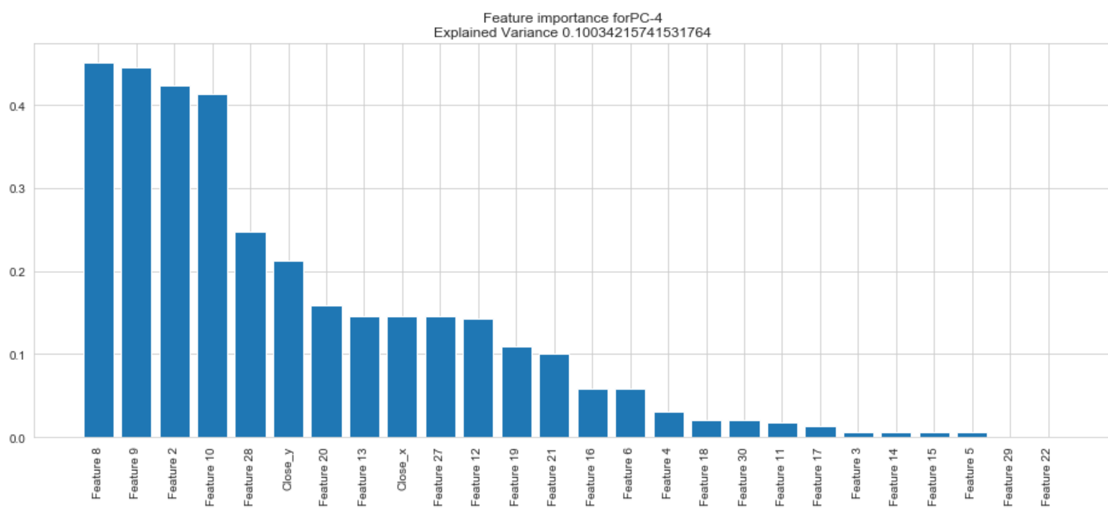



```
In [16]: for j in range(0,len(pca.components_)):
             ranking = np.abs(pca.components_[j])
             indices = np.argsort(ranking)[::-1]
             features = bring_features(indices)

             if(pca.explained_variance_ratio_[j] > add_dimentions_over):
                 ranks['PC-'+str(j)] = rank_to_dict(ranking[indices[range(total_columns)]], feat

             plt.figure(figsize=(16,6))
             plt.bar(features, ranking[indices[range(total_columns)]])
             plt.xticks(rotation=90)

             plt.title("Feature importance for" + dimensions[j] + "\nExplained Variance " + str(
             plt.show()
```


Feature importance forPC-1
Explained Variance 0.3713844001985764
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Feature 4
Feature 21
Feature 14
Feature 3
Feature 17
Feature 15
Feature 5
Feature 6
Feature 16
Feature 12
Feature 28
Feature 18
Feature 20
Close_y
Feature 19
Feature 11
Feature 13
Close_x
Feature 27
Feature 30
Feature 9
Feature 8
Feature 2
Feature 10
Feature 29
Feature 22


Feature importance forPC-2
Explained Variance 0.21590059674198375
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Feature 28
Feature 27
Feature 13
Close_x
Close_y
Feature 8
Feature 21
Feature 9
Feature 2
Feature 10
Feature 30
Feature 20
Feature 19
Feature 12
Feature 18
Feature 15
Feature 5
Feature 16
Feature 6
Feature 4
Feature 3
Feature 14
Feature 11
Feature 17
Feature 29
Feature 22






```
In [17]: print("Variance by feature in each dimension")
         print(pca_results.head)
```

```
Variance by feature in each dimension
<bound method NDFrame.head of              Explained Variance  Close_x  Close_y  Feature 2  Feat
Dimension 1              0.3714   0.1281   0.1786     0.0066     0.3063
Dimension 2              0.2159   0.3318   0.2819     0.2205     0.0262
Dimension 3              0.1305   0.0231   0.1495     0.0070     0.1992
Dimension 4              0.1003   0.1458   0.2128     0.4234     0.0060

             Feature 4  Feature 5  Feature 6  Feature 8  Feature 9  ...  \
Dimension 1     0.3127     0.2572     0.2522     0.0208     0.0214  ...
```

```
Dimension 2     0.0423     0.0562     0.0533     0.2544     0.2411  ...
Dimension 3     0.0724     0.1262     0.4462     0.0062     0.0075  ...
Dimension 4     0.0313     0.0060     0.0588     0.4521     0.4451  ...

             Feature 17  Feature 18  Feature 19  Feature 20  Feature 21  \
Dimension 1      0.2635      0.1904      0.1581      0.1800      0.3126
Dimension 2      0.0217      0.0926      0.1699      0.1762      0.2430
Dimension 3      0.3022      0.0472      0.4202      0.2541      0.0900
Dimension 4      0.0130      0.0211      0.1094      0.1589      0.1011

             Feature 22  Feature 27  Feature 28  Feature 29  Feature 30
Dimension 1         0.0      0.1267      0.1960         0.0      0.1143
Dimension 2         0.0      0.3359      0.4031         0.0      0.1968
Dimension 3         0.0      0.0286      0.0040         0.0      0.0165
Dimension 4         0.0      0.1454      0.2479         0.0      0.0204

[4 rows x 27 columns]>
```

3 Linear Discriminant Analysis (LDA)

LDA is a "supervised" method and computes the directions ("linear discriminants") that represents the axes that maximize the seperation between multiple classes.

Using LDA for the classification of label as an indicator in predicting the preformance of the asset from time to time.

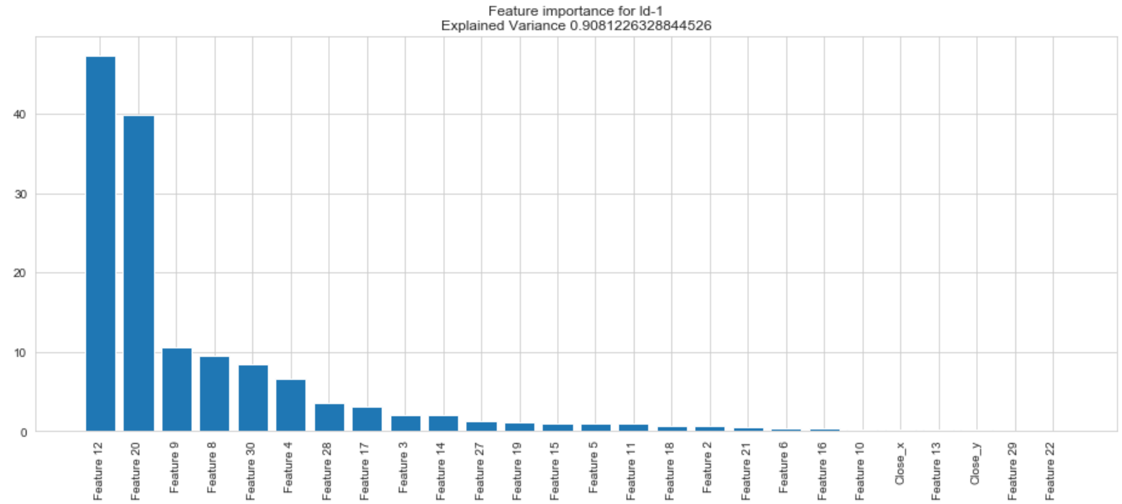
Ex: comparisons between classification accuracies for image recognition after using PCA or LDA show that PCA tends to outperform LDA if the number of samples per class is relatively small (PCA vs. LDA A.M Martinez et al. 2001). In practice it is not uncommon to use bothe LDA and PCA in combination e.g. PCA for dimensionality reduction folloed by an LDA.

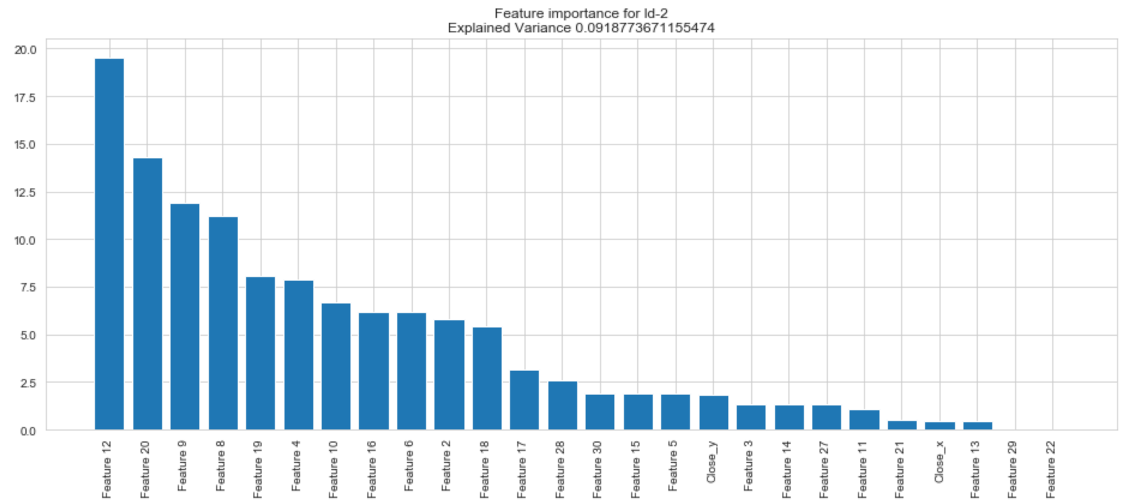
```
In [18]: lda = LDA(n_components=2, solver="svd")
         x_lda = lda.fit(X_std, y).transform(X_std)
```

```
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\discriminant_analysis.py:388:
  warnings.warn("Variables are collinear.")
```

```
In [19]: # Dimension indexing

         dimensions = ['ld-{}'.format(i) for i in range(1, len(lda.classes_))]

         for j in range(0,len(dimensions)):
             sc_vector = np.array(np.abs(lda.scalings_.T[j]))
             indices = np.argsort(sc_vector)[::-1]
             features = bring_features(indices)
             ranks[dimensions[j]] = rank_to_dict(sc_vector[indices[range(total_columns)]], featu
             plt.figure(figsize=(16,6))
             plt.bar(features, sc_vector[indices[range(total_columns)]])
```

```
plt.xticks(rotation=90)

plt.title("Feature importance for " + dimensions[j] + "\nExplained Variance " + str
plt.show()
```

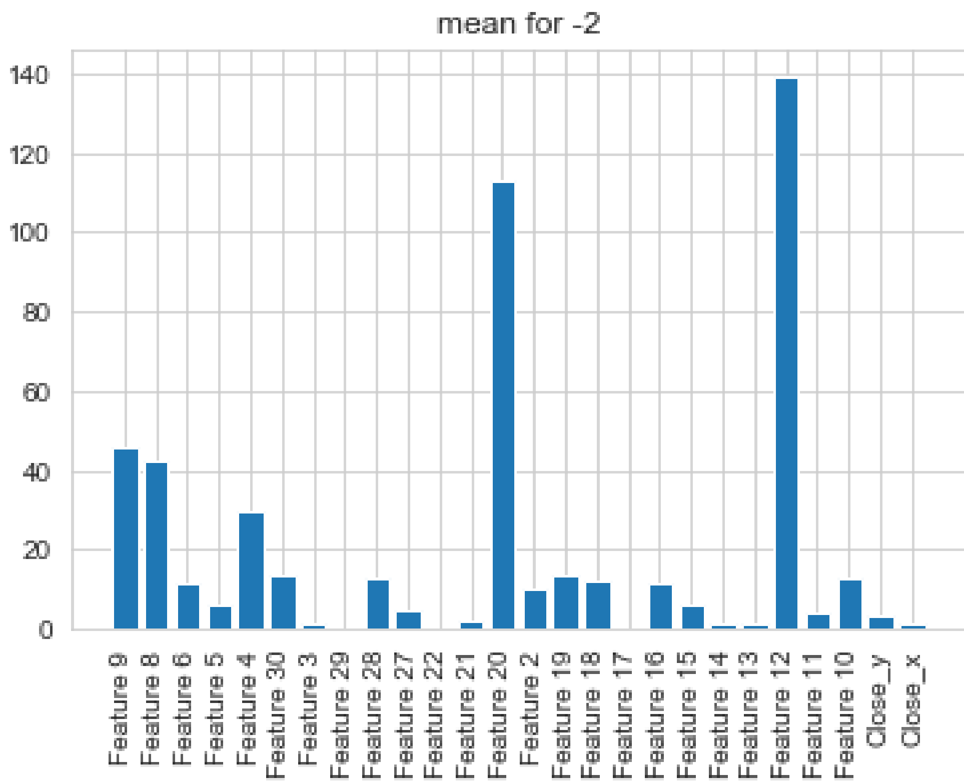


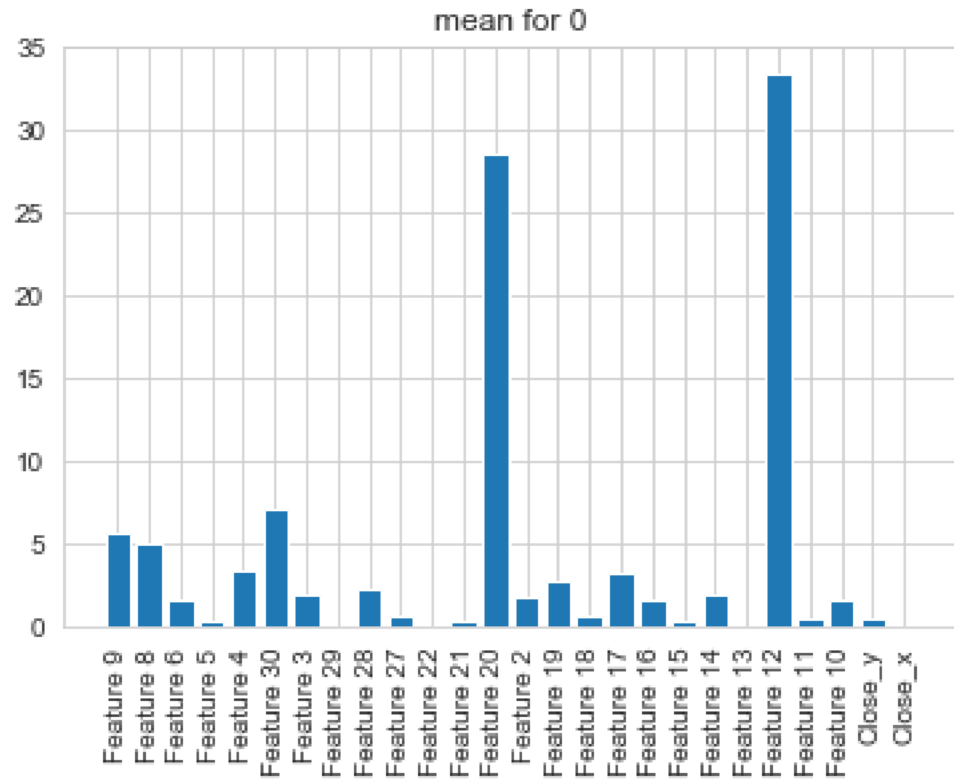


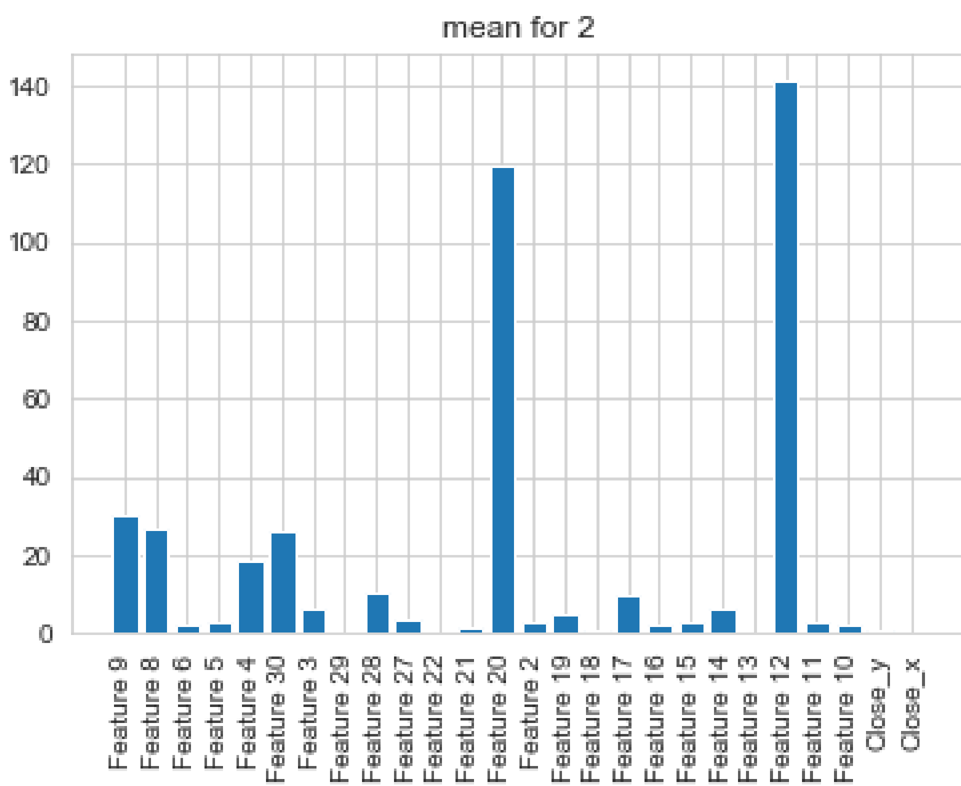

```
In [20]: print("Sorted results")

         print(sorted(zip(map(lambda x: round(x, 4),lda.xbar_[indices[range(total_columns)]]),fe
                  reverse=True))
```

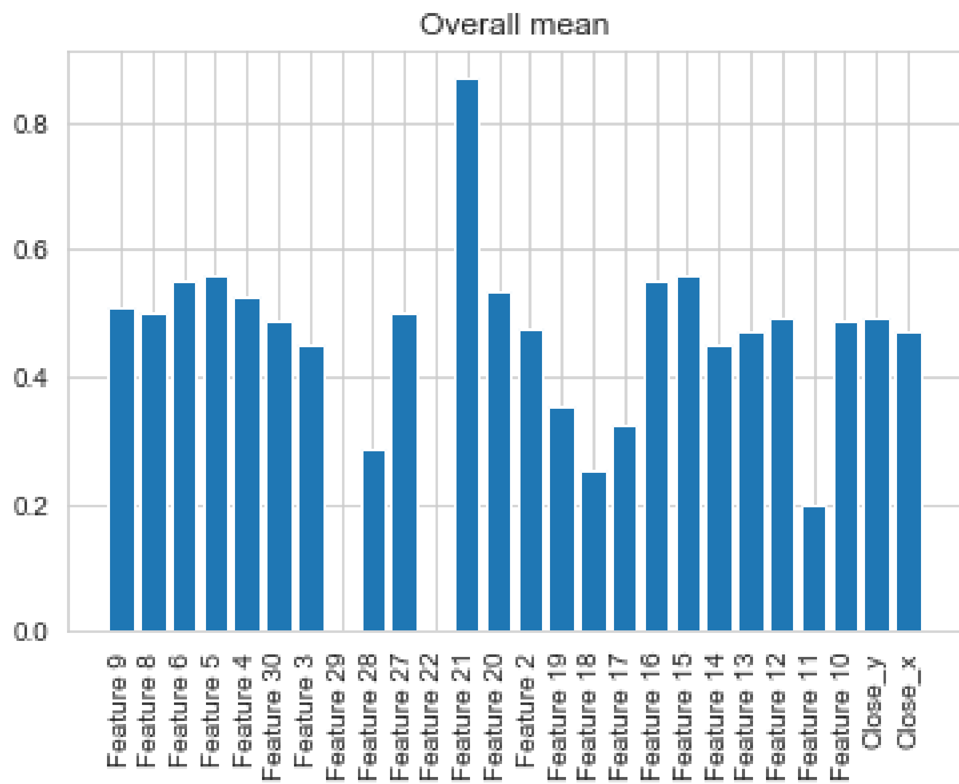
```
Sorted results
[(0.8708, 'Feature 21'), (0.5584, 'Feature 5'), (0.5584, 'Feature 15'), (0.5512, 'Feature 6'), (
```

```
In [21]: for j in range(0,len(label_dict)):
             plt.title("mean for " + label_dict[lda.classes_[j]])
             means = sorted(zip(names,np.abs(lda.coef_[j])), reverse=True)
             plt.bar(*zip(*means))
             plt.xticks(rotation=90)
             plt.show()

         plt.title("Overall mean" )
         means = sorted(zip(names,lda.xbar_), reverse=True)
         plt.bar(*zip(*means))
         plt.xticks(rotation=90)
         plt.show()
```




mean for 0
35
30
25
20
15
10
5
0
Feature 9
Feature 8
Feature 6
Feature 5
Feature 4
Feature 30
Feature 3
Feature 29
Feature 28
Feature 27
Feature 22
Feature 21
Feature 20
Feature 2
Feature 19
Feature 18
Feature 17
Feature 16
Feature 15
Feature 14
Feature 13
Feature 12
Feature 11
Feature 10
Close_y
Close_x


mean for 2
140
120
100
80
60
40
20
0
Feature 9
Feature 8
Feature 6
Feature 5
Feature 4
Feature 30
Feature 3
Feature 29
Feature 28
Feature 27
Feature 22
Feature 21
Feature 20
Feature 2
Feature 19
Feature 18
Feature 17
Feature 16
Feature 15
Feature 14
Feature 13
Feature 12
Feature 11
Feature 10
Close_y
Close_x



3.1 Recursive Feature Elimination (RFE)

Selects most important features. Using this to weight each feature by level of importance.

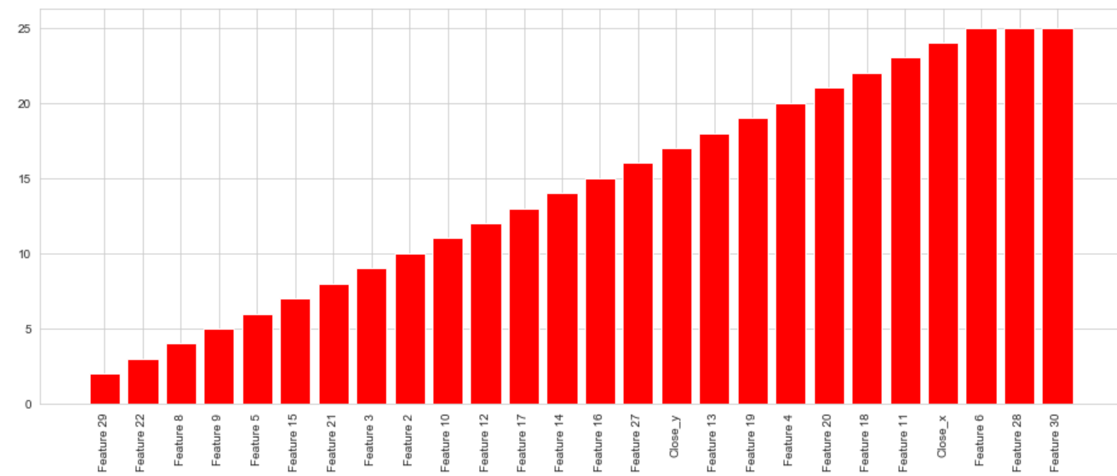
The Recursive Elimination method works by recursively removing attributes and building a model on those attributes that remain. It uses accuracy to rank each feature according to their importance. The RFE method takes the model to be used and the number of required features as input. It then gives the ranking of all variables, 1 being the most important. It also gives its support, True being relevant feature and False being irrelevant feature.

```
In [22]: # Feture Extraction
         model = LogisticRegression(random_state=0, solver='newton-cg', multi_class='auto')
         rfe = RFE(model, 3)
         fit = rfe.fit(X_std, y)
```

```
In [23]: indices = np.argsort(fit.ranking_)[::-1]
         features = bring_features(indices)

         Reversed_ranking=list(map(lambda y:abs(total_columns -y),fit.ranking_[indices[range(tot
         ranks["RFE"] = rank_to_dict(Reversed_ranking, features)
```

```
#Now plot
plt.figure(figsize=(16,6))
plt.bar(features, Reversed_ranking, color='r', align='center')
plt.xticks(rotation=90)
plt.show()
```

```
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\utils\validation.py:595: DataC
  warnings.warn(msg, DataConversionWarning)
```




```
In [24]: #no of features
         nof_list=np.arange(1,total_columns)
         high_score=0
         # Variable to store in the optimum features
         nof=0
         score_list=[]
         for n in range(len(nof_list)):
             X_train, X_test, y_train, y_test = train_test_split(X_std,y, test_size = 0.3, rando
             model = LinearRegression()
             rfe = RFE(model,nof_list[n])
             X_train_rfe = rfe.fit_transform(X_train,y_train)
             X_test_rfe = rfe.transform(X_test)
             model.fit(X_train_rfe,y_train)
             score = model.score(X_test_rfe,y_test)
             score_list.append(score)
             if(score>high_score):
                 high_score = score
                 nof = nof_list[n]
         print("Optimum number of features: %d" %nof)
         print("Score with %d features: %f" % (nof, high_score))
```

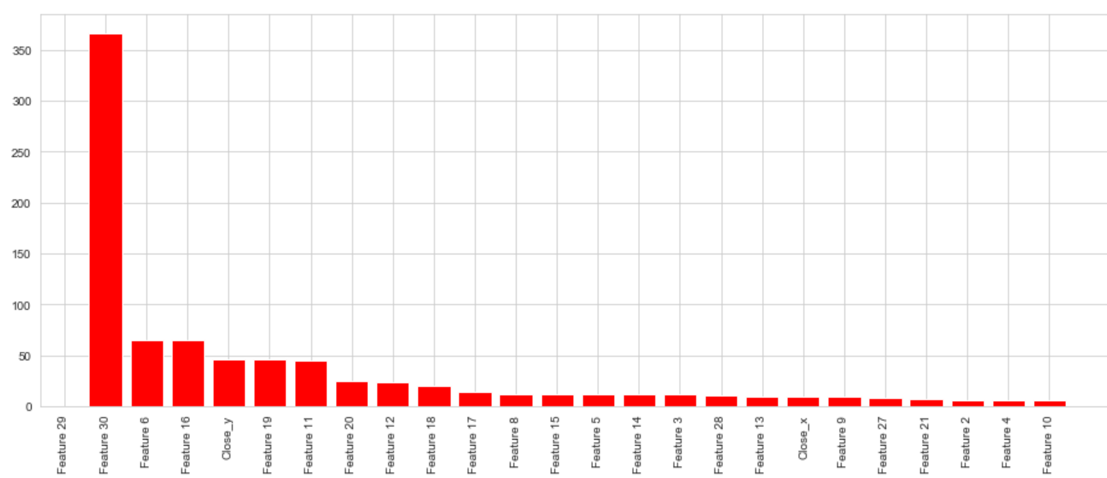
```
Optimum number of features: 23
Score with 23 features: 0.739411
```

4 Select K Best

- Select features according to the k highest scores.

For example, if you pass chiˆ2 as a scor function, SelectKBest will compute the chi2 statistic between each feature of X and y (assumed to be class labels). A small value will mean the feature is independent of y. A large value will mean the feature is non-randomly related to y, and so likely to provide important information. Only k features will be retained.

```
In [25]: selector = SelectKBest(score_func=chi2, k='all')
         selector = selector.fit(X_std, y)

         # summarize scores
         np.set_printoptions(precision=3)

         #features = selector.transform(X_std)
         # summarize selected features
         #print(features[0:5,:])
         indices = np.argsort(selector.scores_)[::-1]
         features = bring_features(indices)

         plot_selectKbest(selector)
         ranks["selectbest"] = rank_to_dict(selector.scores_[indices[range(total_columns)]],feat
```

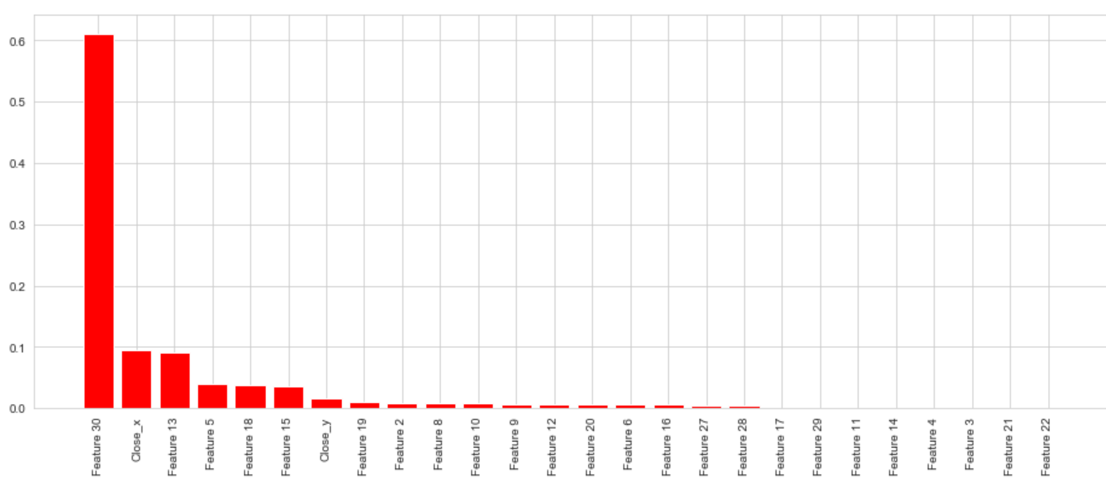


It appears that Feature 30 has a lot of importance in out preliminary analysis

5 Random Forest Regressor

A Random Forest is an ensemble technique capable of performing both regression and classification tasks with the use of multiple decision trees and a method called BootStrap Aggregation, commonly known as bagging. Bagging in the Random Forest method, involves training each decision tree on a different data sample here sampling is done with replacement.

```
In [26]: rf = RandomForestRegressor(n_estimators=500) #Number of tree's, we should bump this up
         rf.fit(X_std, y)

         indices = np.argsort(rf.feature_importances_)[::-1]
         features = bring_features(indices)

         ranks["RF"] = rank_to_dict(rf.feature_importances_, names)

         plt.figure(figsize=(16,6))
         plt.bar(features, rf.feature_importances_[indices[range(total_columns)]], color='r', al
         plt.xticks(rotation=90)
         plt.show()
```



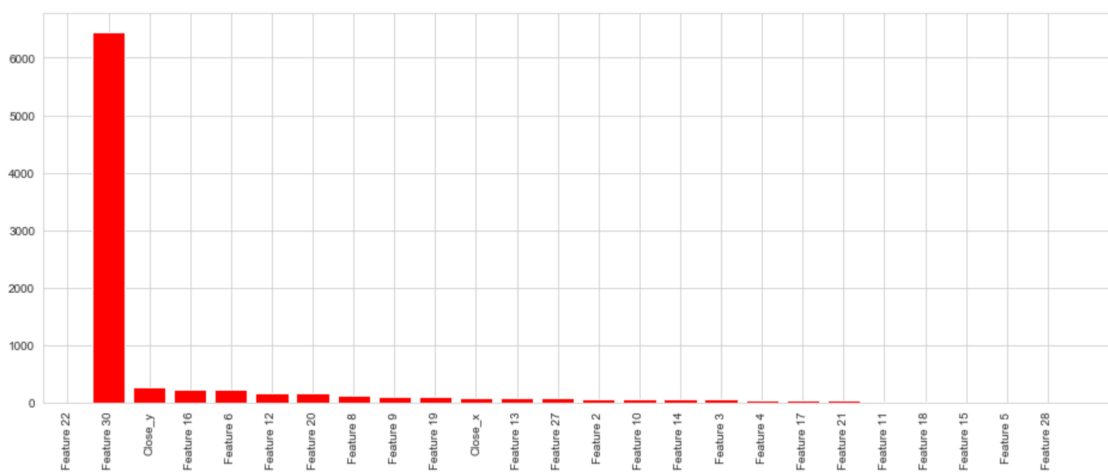

```
In [27]: print(sorted(zip(map(lambda x: round(x, 4), rf.feature_importances_[indices[range(total
                          reverse=True))

[(0.6115, 'Feature 30'), (0.0942, 'Close_x'), (0.0903, 'Feature 13'), (0.0398, 'Feature 5'), (0.
```

6 f_regression

Used only for numeric targets and based on linear regression performance.

```
In [28]: f, pval = feature_selection.f_regression(X_std, y, center=True)

         indices = np.argsort(f)[::-1]
         features = bring_features(indices)

         plt.figure(figsize=(16,6))
         plt.bar(features, f[indices[range(total_columns)]], color='r', align='center')
         plt.xticks(rotation=90)
         plt.show()

         ranks["Corr"] = rank_to_dict(f, features)
```

```
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\feature_selection\univariate_s
  corr /= X_norms
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\scipy\stats\_distn_infrastructure.py:8
  return (self.a < x) & (x < self.b)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\scipy\stats\_distn_infrastructure.py:8
  return (self.a < x) & (x < self.b)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\scipy\stats\_distn_infrastructure.py:1
  cond2 = cond0 & (x <= self.a)
```




```
In [29]: print(features)
         print(f)
```

```
['Feature 29', 'Feature 22', 'Feature 30', 'Close_y', 'Feature 16', 'Feature 6', 'Feature 12', '
[9.006e+01 2.708e+02 6.845e+01 6.059e+01 5.525e+01 3.888e+00 2.409e+02
 1.191e+02 1.053e+02 6.516e+01 3.043e+01 1.784e+02 9.006e+01 6.059e+01
 3.888e+00 2.409e+02 5.000e+01 1.783e+01 1.053e+02 1.760e+02 3.628e+01
       nan 7.645e+01 5.064e-01       nan 6.455e+03]
```

7 LASSO

Embedded methds are iterative in a sense that takes care of each iteration of the model training process and carefully extract those features which contribute the most to the training for a particular iteratio. Regularization methods are the most commonly used embedded methods which penalize a featue given a coefficient threshold.

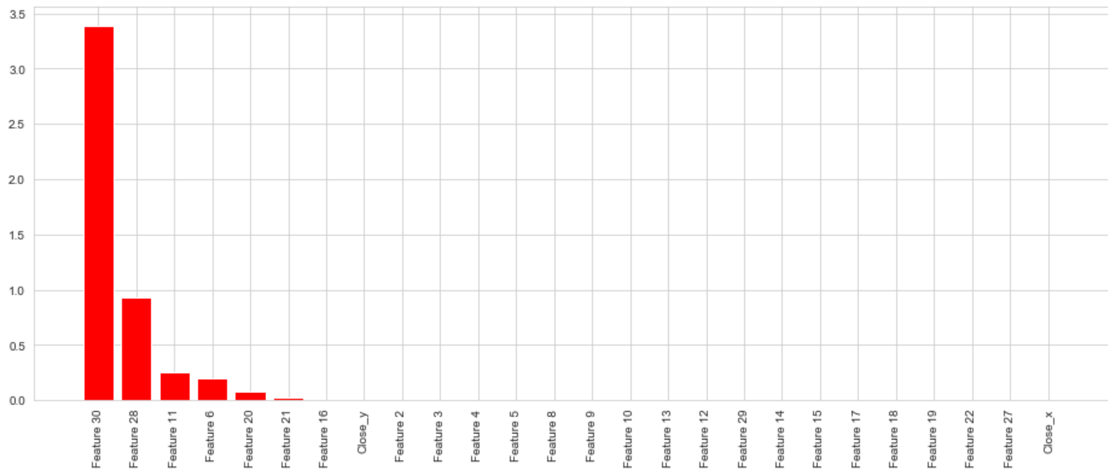
Here we will do feautre selection using Lasso regularization. If the feature is irrelevant, lasso penalizes it's coefficient and make it 0. Hence, the features with the coefficient = 0 are removed and the rest are taken.

```
In [30]: from sklearn.linear_model import LassoCV
         reg = LassoCV(cv=15, random_state=0).fit(X_std, y)

         reg.fit(X_std, y)
         print("Best alpha using built-in LassoCV: %f" %reg.alpha_)
         print("Best alpha using built-in LassoCV: %f" %reg.score(X_std,y))
         coef = pd.Series(reg.coef_, index = names)

         coeficients = np.abs((reg.coef_))

         indices = np.argsort(coeficients)[::-1]
         features = bring_features(indices)

         plt.figure(figsize=(16,6))
         plt.bar(features, coeficients[indices[range(total_columns)]], color = 'r', align = 'cen
         plt.xticks(rotation=90)
         plt.show()

         ranks["Lazzo"] = rank_to_dict(coeficients[indices[range(total_columns)]], features)

         print(names)
         print(reg.coef_)
```

```
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
```

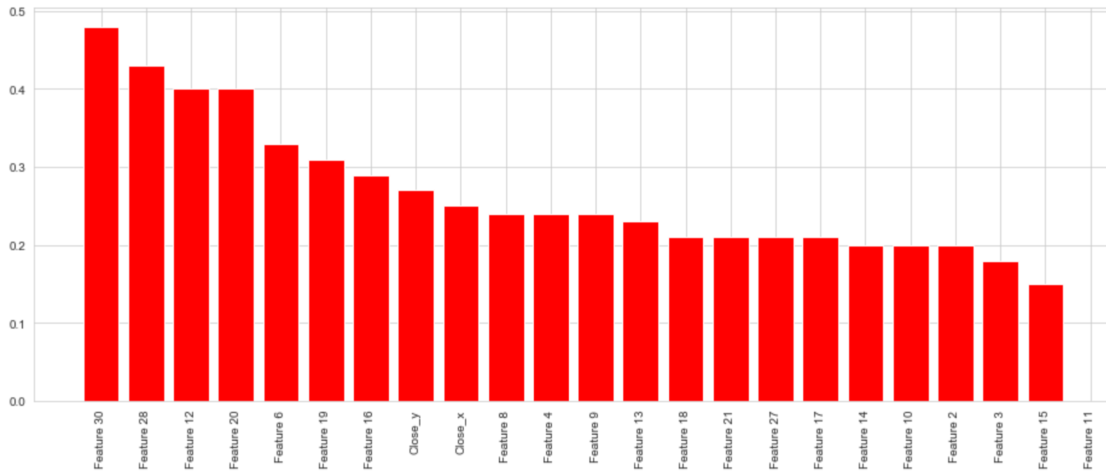
```
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)
C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\linear_model\coordinate_descen
  ConvergenceWarning)


Best alpha using built-in LassoCV: 0.011118
```

```
Best alpha using built-in LassoCV: 0.654006
```




```
['Close_x', 'Close_y', 'Feature 2', 'Feature 3', 'Feature 4', 'Feature 5', 'Feature 6', 'Feature
[ 0.000e+00  0.000e+00  0.000e+00 -0.000e+00  0.000e+00  0.000e+00
 -1.943e-01 -0.000e+00  0.000e+00  0.000e+00 -2.558e-01 -0.000e+00
  0.000e+00 -0.000e+00  0.000e+00 -7.098e-06 -0.000e+00 -0.000e+00
  0.000e+00  7.852e-02  2.536e-02  0.000e+00  0.000e+00 -9.293e-01
  0.000e+00  3.392e+00]
```

In [31]:
```
r = {}
for name in features:
    r[name] = round(np.mean([ranks[method][name]
                             for method in ranks.keys()]), 2)

methods = sorted(ranks.keys())

ranks["Mean"] = r
methods.append("Mean")


df2 = pd.DataFrame(ranks, columns = methods)
df2.index.rename('feature Name', inplace=True)
df2 = df2.sort_values(by='Mean', ascending =False)
plt.figure(figsize=(16, 6))
plt.bar(df2.index, df2['Mean'], color='r', align='center')
plt.xticks(rotation=90)
plt.show()
```




```
In [32]: print(df2)
```

```
              Corr  Lazzo  PC-0  PC-1  PC-2  PC-3    RF   RFE  ld-1  ld-2  \
feature Name
Feature 30    0.01   1.00  0.37  0.49  0.04  0.05  1.00  1.00  0.18  0.10
Feature 28    1.00   0.27  0.63  1.00  0.01  0.55  0.01  1.00  0.07  0.13
Feature 12    0.04   0.00  0.64  0.39  0.52  0.32  0.01  0.43  1.00  1.00
Feature 20    0.02   0.02  0.58  0.44  0.57  0.35  0.01  0.83  0.84  0.73
Feature 6     0.00   0.06  0.81  0.13  1.00  0.13  0.01  1.00  0.01  0.31
Feature 19    0.00   0.00  0.51  0.42  0.94  0.24  0.02  0.74  0.02  0.41
Feature 16    0.01   0.00  0.81  0.13  1.00  0.13  0.01  0.57  0.01  0.31
Close_y       0.01   0.00  0.57  0.70  0.34  0.47  0.03  0.65  0.00  0.09
Close_x       0.03   0.00  0.41  0.82  0.05  0.32  0.15  0.96  0.00  0.02
Feature 8     0.02   0.00  0.07  0.63  0.01  1.00  0.01  0.09  0.20  0.57
Feature 4     0.02   0.00  1.00  0.10  0.16  0.07  0.00  0.78  0.14  0.40
Feature 9     0.01   0.00  0.07  0.60  0.02  0.98  0.01  0.13  0.22  0.61
Feature 13    0.01   0.00  0.41  0.82  0.05  0.32  0.15  0.70  0.00  0.02
Feature 18    0.01   0.00  0.61  0.23  0.11  0.05  0.06  0.87  0.01  0.28
Feature 21    0.01   0.01  1.00  0.60  0.20  0.22  0.00  0.26  0.01  0.03
Feature 27    0.01   0.00  0.41  0.83  0.06  0.32  0.01  0.61  0.03  0.07
Feature 17    0.03   0.00  0.84  0.05  0.68  0.03  0.00  0.48  0.06  0.16
Feature 14    0.01   0.00  0.98  0.07  0.45  0.01  0.00  0.52  0.04  0.07
Feature 10    0.04   0.00  0.01  0.54  0.00  0.91  0.01  0.39  0.00  0.34
Feature 2     0.00   0.00  0.02  0.55  0.02  0.94  0.01  0.35  0.01  0.30
Feature 3     0.00   0.00  0.98  0.07  0.45  0.01  0.00  0.30  0.04  0.07
Feature 15    0.00   0.00  0.82  0.14  0.28  0.01  0.06  0.22  0.02  0.10
Feature 11     NaN   0.08  0.47  0.06  0.57  0.04  0.00  0.91  0.02  0.06
Feature 22    0.04   0.00  0.00  0.00  0.00  0.00  0.00  0.04  0.00  0.00
Feature 29    0.01   0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00  0.00
Feature 5      NaN   0.00  0.82  0.14  0.28  0.01  0.07  0.17  0.02  0.10
```

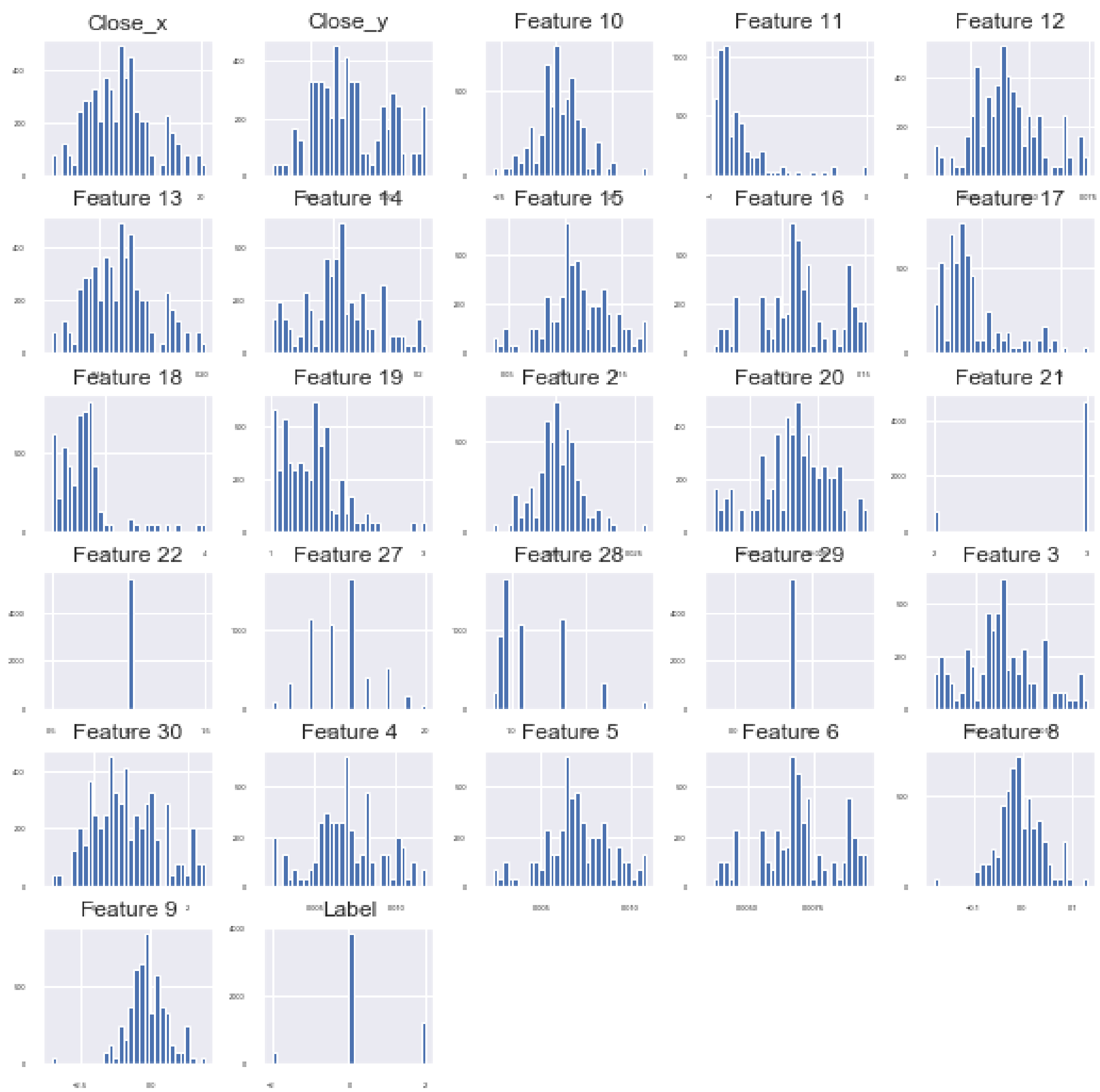
```
             selectbest  Mean
feature Name
Feature 30         1.00  0.48
Feature 28         0.02  0.43
Feature 12         0.05  0.40
Feature 20         0.05  0.40
Feature 6          0.17  0.33
Feature 19         0.11  0.31
Feature 16         0.17  0.29
Close_y            0.11  0.27
Close_x            0.01  0.25
Feature 8          0.02  0.24
Feature 4          0.00  0.24
Feature 9          0.01  0.24
Feature 13         0.01  0.23
Feature 18         0.04  0.21
Feature 21         0.00  0.21
Feature 27         0.01  0.21
Feature 17         0.02  0.21
Feature 14         0.02  0.20
Feature 10         0.00  0.20
Feature 2          0.00  0.20
Feature 3          0.02  0.18
Feature 15         0.02  0.15
Feature 11         0.11   NaN
Feature 22          NaN   NaN
Feature 29          NaN   NaN
Feature 5          0.02   NaN
```

```
In [33]: print(df.describe())
```

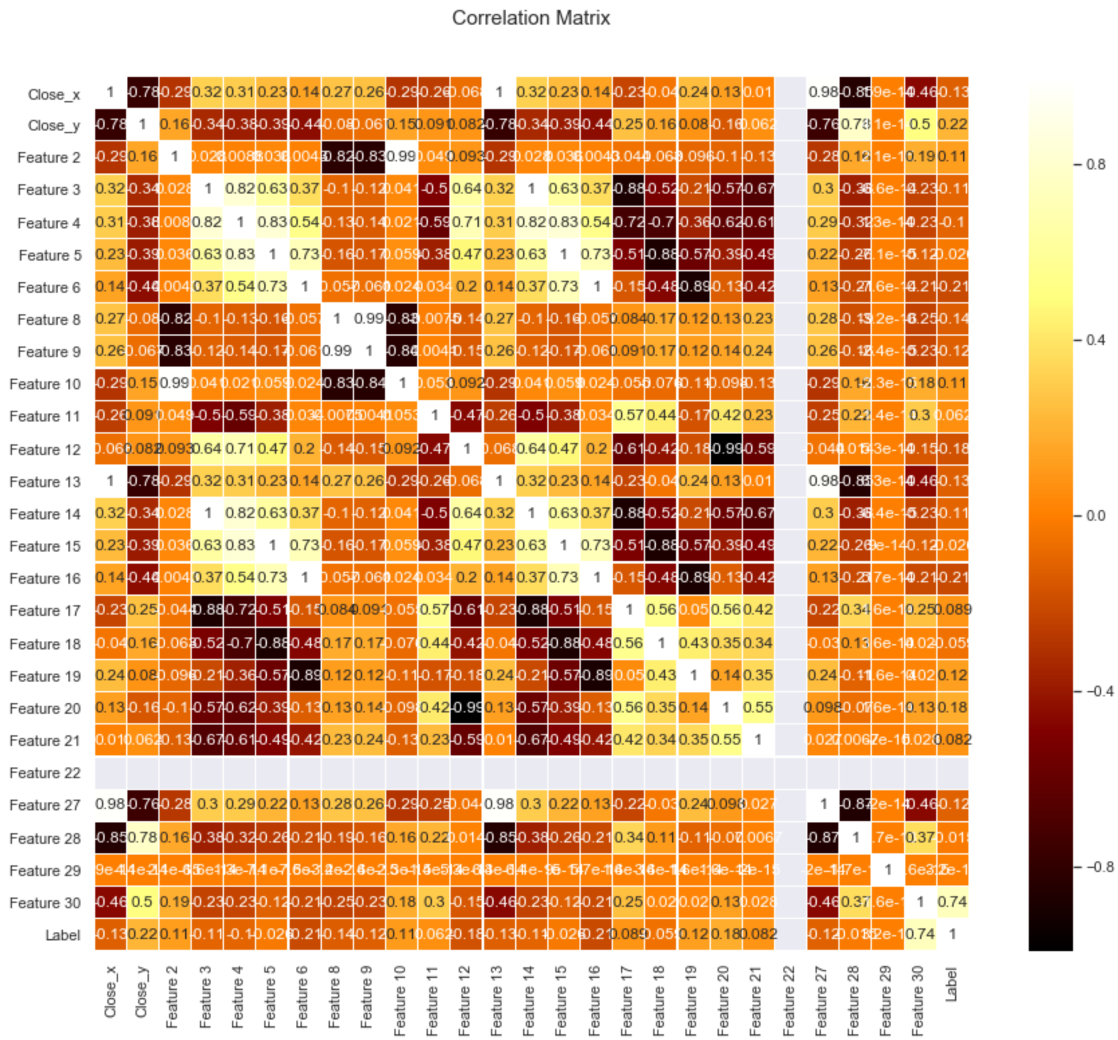
```
           Close_x      Close_y    Feature 2    Feature 3    Feature 4  \
count  5394.000000  5394.000000  5394.000000  5394.000000  5394.000000
mean     15.991915  1453.712736     0.000720     0.006794     0.006985
std       1.556545    48.670473     0.007154     0.002591     0.002141
min      12.640000  1348.750000    -0.019982     0.001961     0.002367
25%      14.850000  1414.000000    -0.003248     0.004935     0.005729
50%      16.000000  1448.500000     0.000348     0.006667     0.006881
75%      16.800000  1495.500000     0.004985     0.008328     0.008233
max      20.200000  1555.000000     0.028998     0.013090     0.011842

         Feature 5    Feature 6    Feature 8    Feature 9   Feature 10  ...  \
count  5394.000000  5394.000000  5394.000000  5394.000000  5394.000000  ...
mean      0.007004     0.006975    -0.005644    -0.104607     1.156656  ...
std       0.001820     0.001463     0.044140     0.737417    10.314946  ...
min       0.002282     0.003639    -0.179747    -3.550000   -28.500000  ...
25%       0.006079     0.006146    -0.032452    -0.500000    -4.750000  ...
```

```
50%       0.006890     0.006888    -0.009524    -0.150000     0.750000  ...
75%       0.008246     0.007985     0.020626     0.300000     7.250000  ...
max       0.010895     0.009589     0.130263     1.980000    40.750000  ...

        Feature 18   Feature 19   Feature 20   Feature 21  Feature 22  \
count  5394.000000  5394.000000  5394.000000  5394.000000      5394.0
mean      1.566502     1.518631    -0.033701     2.870782         1.0
std       0.536260     0.368077     0.012703     0.335472         0.0
min       0.909813     1.015965    -0.063076     2.000000         1.0
25%       1.220607     1.216113    -0.041119     3.000000         1.0
50%       1.516625     1.512051    -0.032678     3.000000         1.0
75%       1.694564     1.699343    -0.025225     3.000000         1.0
max       4.018696     3.030833    -0.006699     3.000000         1.0

        Feature 27   Feature 28    Feature 29   Feature 30        Label
count  5394.000000  5394.000000  5.394000e+03  5394.000000  5394.000000
mean     15.522062     1.107019  3.600000e-01     0.706737     0.334446
std       1.569921     0.233929  5.551630e-17     0.711442     1.021615
min      12.000000     0.870000  3.600000e-01    -0.890000    -2.000000
25%      14.000000     0.950000  3.600000e-01     0.190000     0.000000
50%      16.000000     0.950000  3.600000e-01     0.630000     0.000000
75%      16.000000     1.350000  3.600000e-01     1.170000     0.000000
max      20.000000     1.910000  3.600000e-01     2.360000     2.000000

[8 rows x 27 columns]
```

In [34]:
```python
from matplotlib import pyplot

sns.set()
df.hist(sharex=False, sharey=False, xlabelsize = 4, ylabelsize=4, bins=30, figsize=(10,
pyplot.show()
```



In [35]:
```
colormap = pyplot.cm.afmhot
pyplot.figure(figsize=(16,12))
pyplot.title('Correlation Matrix', y=1.05, size=15)
sns.heatmap(df.corr(),linewidths=0.1,vmax=1.0, square=True,
            cmap=colormap, linecolor='white', annot=True)
pyplot.show()
```

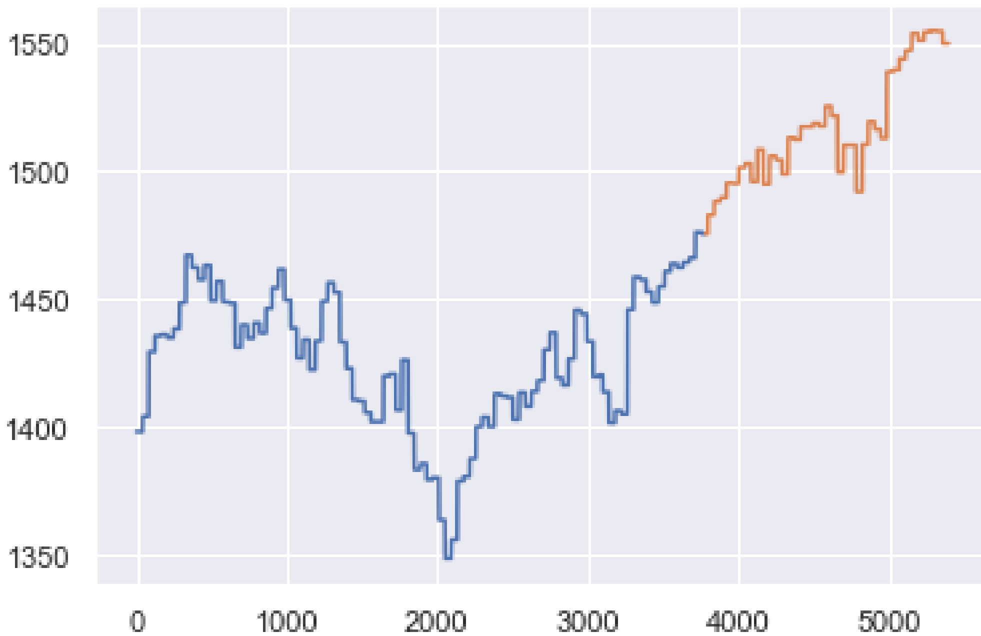


In [46]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.ensemble import AdaBoostClassifier, GradientBoostingClassifier, RandomFore
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, me
print(df.count())

X = df.loc[:,'Close_x':'Feature 30']
Y = df.loc[:,'Label']

# Hold out data split for testing 20%
validation= 0.20

# 80/20 train_test remaining split
train_size = int(len(X.index) * 0.7)
```

```python
print(len(y))
print(train_size)

X_train, X_test = X.loc[0:train_size, :], X.loc[train_size: len(X.index), :]
Y_train, Y_test = Y[0:train_size+1], Y.loc[train_size: len(X.index)]

print('Observations: %d' % (len(X.index)))
print('X Training Observations: %d' % (len(X_train.index)))
print('X Testing Observations: %d' % (len(X_test.index)))
print('y Training Observations: %d' % (len(Y_train)))
print('y Testing Observations: %d' % (len(Y_test)))

pyplot.plot(X_train['Close_y'])
pyplot.plot([None for i in X_train['Close_y']] + [x for x in X_test['Close_y']])
pyplot.show()

num_folds = 10
scoring = 'accuracy'

models = []
models.append(('KNN' , KNeighborsClassifier()))
models.append(('SVM' , SVC()))
models.append(('RF' , RandomForestClassifier(n_estimators=50)))
models.append(('XGBoost', XGBClassifier()))
```

```
Close_x       5394
Close_y       5394
Feature 2     5394
Feature 3     5394
Feature 4     5394
Feature 5     5394
Feature 6     5394
Feature 8     5394
Feature 9     5394
Feature 10    5394
Feature 11    5394
Feature 12    5394
Feature 13    5394
Feature 14    5394
Feature 15    5394
Feature 16    5394
Feature 17    5394
Feature 18    5394
Feature 19    5394
Feature 20    5394
Feature 21    5394
Feature 22    5394
Feature 27    5394
```

```
Feature 28    5394
Feature 29    5394
Feature 30    5394
Label         5394
dtype: int64
5394
3775
Observations: 5394
X Training Observations: 3776
X Testing Observations: 1619
y Training Observations: 3776
y Testing Observations: 1619
```



In [47]:
```
# Checking accuracy of selected models

results = []
names = []
'''
for name, model in models:
    kfold = KFold(n_splits=num_folds, random_state=42)
    cv_results = cross_val_score(model, X_train, y_train, cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
```

```
        print(msg) '''


    for name, model in models:
        clf = model
        clf.fit(X_train, Y_train)
        Y_pred = clf.predict(X_test)
        accu_score = accuracy_score(Y_test, Y_pred)
        print(name + ": " + str(accu_score))
```

```
KNN: 0.18962322421247685
SVM: 0.7974058060531192


C:\Users\ANONYMOUS\Anaconda3\envs\env01\lib\site-packages\sklearn\svm\base.py:196: FutureWarning
  "avoid this warning.", FutureWarning)


RF: 0.721432983323039
XGBoost: 0.6454601605929586
```

8 Recommendations

1. I think Feature selection or eliminating some features does not have many changes
2. Taking the closing price difference of each day to have stationary data
3. Look into minute timeperiods for better prediction