1-Introduction:

The ultimate objective of this project is to implement a complete trading strategy using machine learning. In the first submission, raw tick data, containing 20 days of S&P500 E-mini features were downloaded and used to create financial data structures in the form of volume and dollar bars; leading to data that has better statistical properties than traditional fixed time interval sampling.

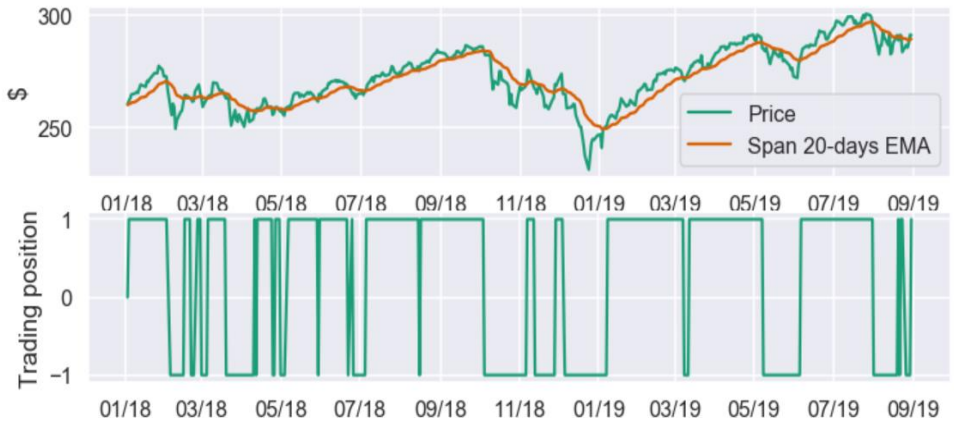
In this second submission, additional features are calculated, and some are graphed mainly, simple moving average (SMA), exponential moving average (EMA), moving average convergence divergence (MACD), the relative strength index (RSI). Then components (from PCA) are to be extracted to decide which features are helpful in predicting the target variables. Feature engineering and cross validation are then explained with relevant referencing in preparation for submission 3, where forecasting process is involved.

The developed code is based on tick data downloaded using yahoo finance (yfinance as yf), for SPY for the period from "SPY", start="2018-01-01", end="2019-09-01". Other implemented libraries are: numpy, pandas, seaborn, statsmodels.graphics.tsaplots (plot_acf), matplotlib.pyplot and scipy(stats), pandas_datareader, matplotlib.dates, %matplotlib inline, pandas plotting and converters.

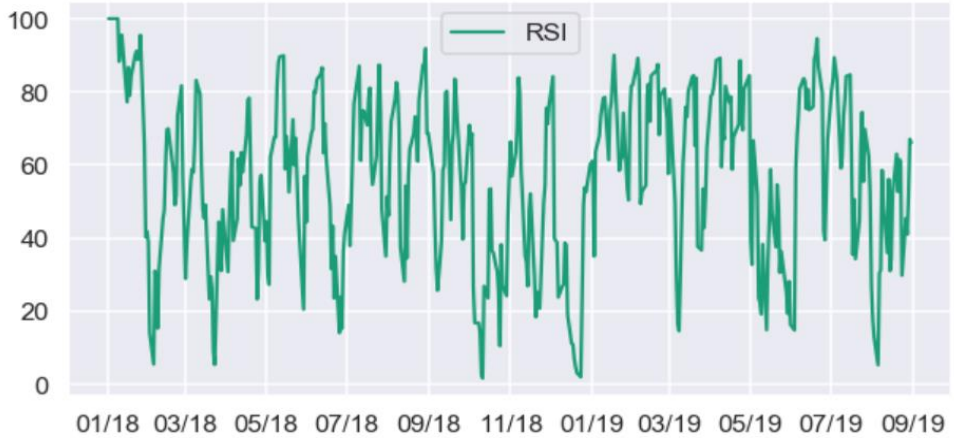
2-Activities:

2.1- Select at least four explanatory variables and perform the necessary transformations so that they are useful in the model phase. You are encouraged to use more than four variables.

Additional features were generated, plotted and added to the data frame. Calculation was done based on Adjusted close price. Selection was based on the recommended features for trading strategies as referenced below in Appendix-I. However, OBV was not successfully generated using codes from different sources and shall be further studies prior to submission 3. On the other hand, included in the code calculations of OBV using "back trader" module. Charts below represent the different features in comparison.


300
$
250
Price
Span 20-days EMA
01/18
03/18
05/18
07/18
09/18
11/18
01/19
03/19
05/19
07/19
09/19
Trading position
1
0
−1


8
6
4
2
0
−2
−4
MACD
01/18
03/18
05/18
07/18
09/18
11/18
01/19
03/19
05/19
07/19
09/19

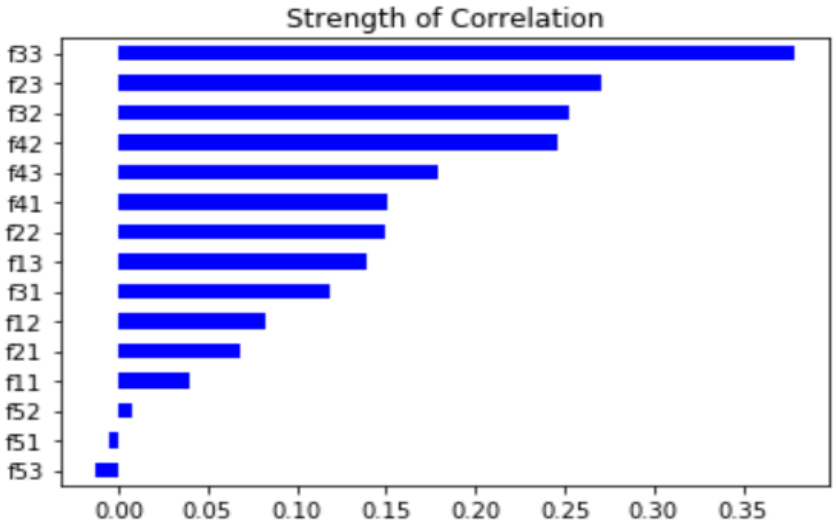

100
80
60
40
20
0
RSI
01/18
03/18
05/18
07/18
09/18
11/18
01/19
03/19
05/19
07/19
09/19

```
data.tail() # To check whether data is in format or not.
#Result shows it is still in format, now we will feature selection process
```

Date	Open	High	Low	Close	Adj Close	Volume	MACD	RSI	EMA
2019-08-26	287.27	288.00	285.58	288.00	286.68	72423800	1.656848	45.382895	289.416142
2019-08-27	289.54	289.95	286.03	286.87	285.55	66668900	1.754724	41.000226	289.047938
2019-08-28	286.14	289.07	285.25	288.89	287.56	59696700	1.651069	51.995062	288.906229
2019-08-29	291.72	293.16	290.61	292.58	291.24	57899400	1.257481	67.004956	289.128493
2019-08-30	294.22	294.24	291.42	292.45	291.11	62901200	0.945154	66.032469	289.317208

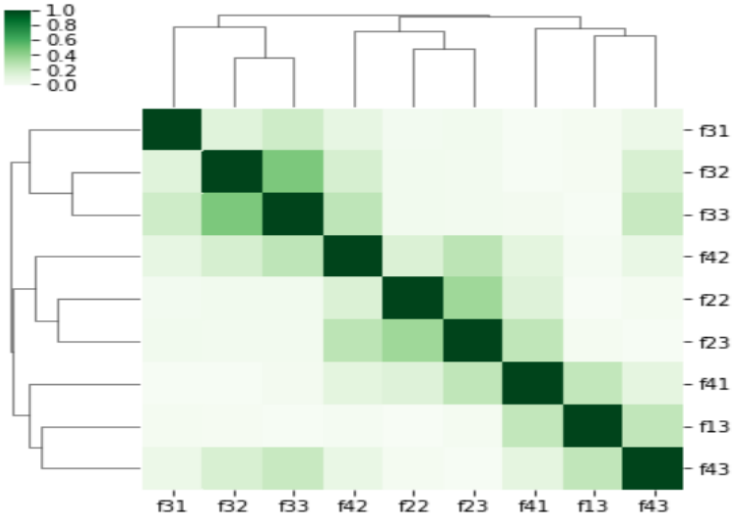
Calculated features are then saved in another dataframe called features inclusive of all downloaded features and those added as briefed above. Target variables (outcome) can be (partially) predicted with three factors, f1, f2 and f3. Several variations on features which each contain some information about the three factors, plus a few which contain some interaction effects, and some which do not contain any useful data. Additional features can be randomly created for illustrative purposes.

Before evaluating the features for predictive strength quick **data preparation stage is essential to "standardize" or "normalize" data** so that fair comparisons between features of differing scale can be visualized. Also, it is good practice since not all the data follow random distribution. The scikit-learn StandardScaler() method and some pandas are implemented to transform the data, (Gray, 2018). Finding the correlation of the target values provides a graph as below



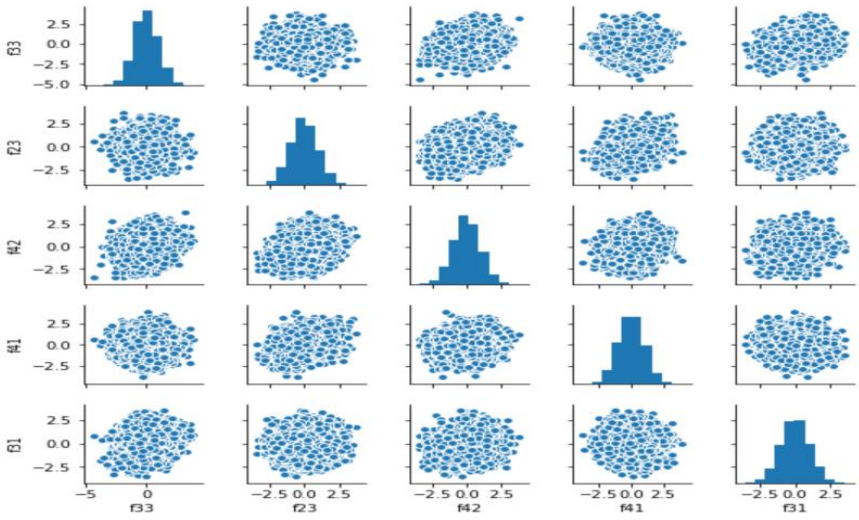
Source:https://github.com/convergenceIM/alpha-scientist/blob/master/content/03_Feature_Selection.ipynb

While correlation is not the perfect metric, it gives a reasonable sense of strength of each feature's historical relationship to the outcome variable. However, for orthogonality, seaborn clustermap chart type which plots a heatmap representation of a covariance matrix and runs a hierarchical clustering algorithm to group together the most closely related features, can be used.



Source:https://github.com/convergenceIM/alpha-scientist/blob/master/content/03_Feature_Selection.ipynb

The diagonal of dark green represents each feature being perfectly correlated with itself, but certain clusters of features are similar to one another. For features with correlations of greater than 0.1 shall be selected and others are considered noise and weak features. This can be manually selected as being close to the diagonal. Graph below shows the plot of each pair.



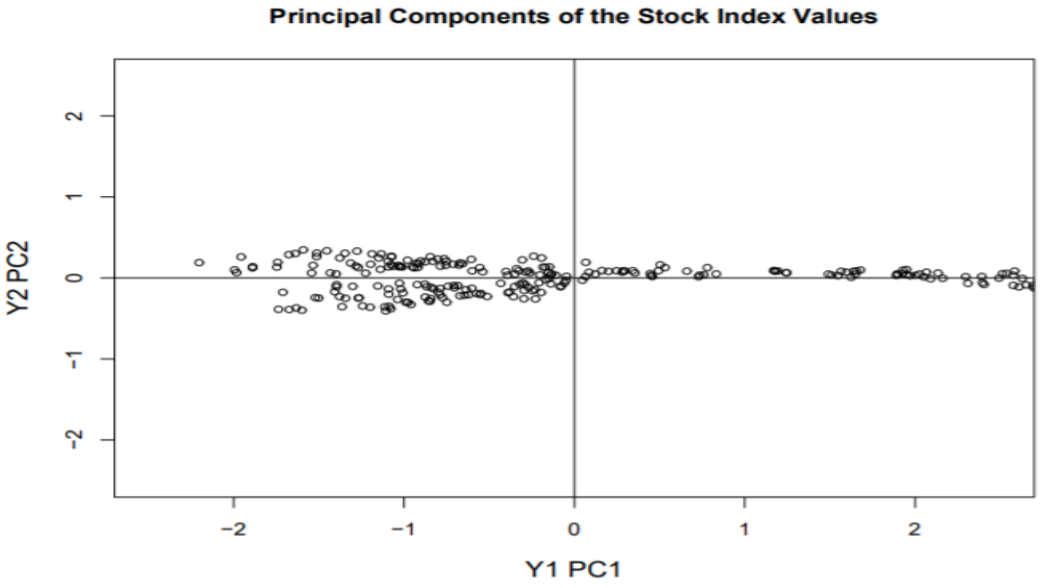
Source:https://github.com/convergenceIM/alpha-scientist/blob/master/content/03_Feature_Selection.ipynb

2.2- Investigate feature engineering techniques such as PCA and encoding target variables using one-hot encoding. Write a short paragraph about each technique investigated and show an implementation of it in a Jupiter Notebook. Make sure to include references that indicate where the ideas were sourced.

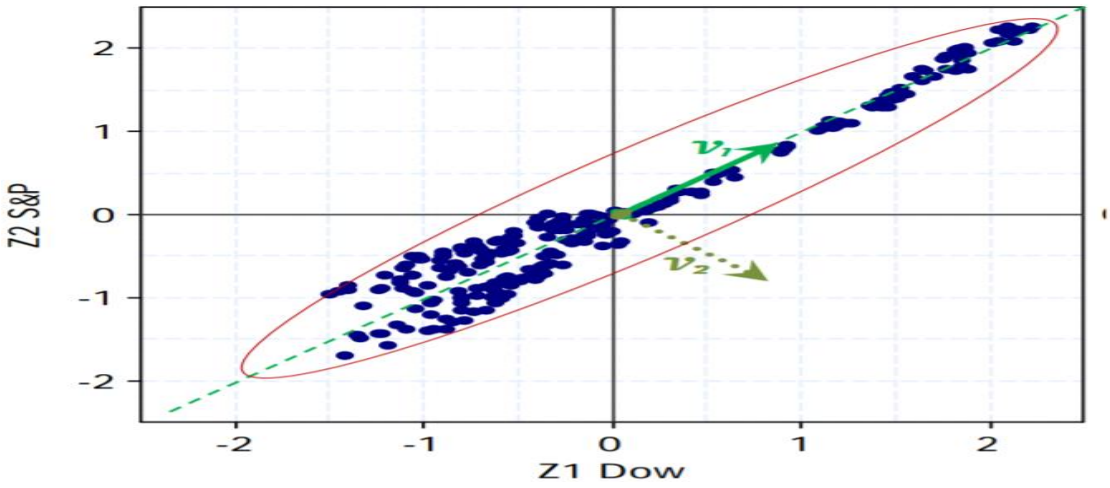
Different techniques for feature engineering and feature selection are discussed in Appendix-II and III in addition to correlation explained above and other various applications. **Python codes are implemented for PCA in this submission.** Examples python codes are provided for each selection method, (Brownlee, Feature Selection For Machine Learning in Python, 2016). (Shetye, 2019) broadly categorized feature selection into 3 categories :1. Filter Method, 2. Wrapper Method, 3. Embedded Method. She highlighted the following issues with

respect to each of the techniques; “Filter method is less accurate; it can also be used for checking multi co-linearity in data”. Other methods “give more accurate results but as they are computationally expensive, these methods are suited when you have lesser features (~20)”.

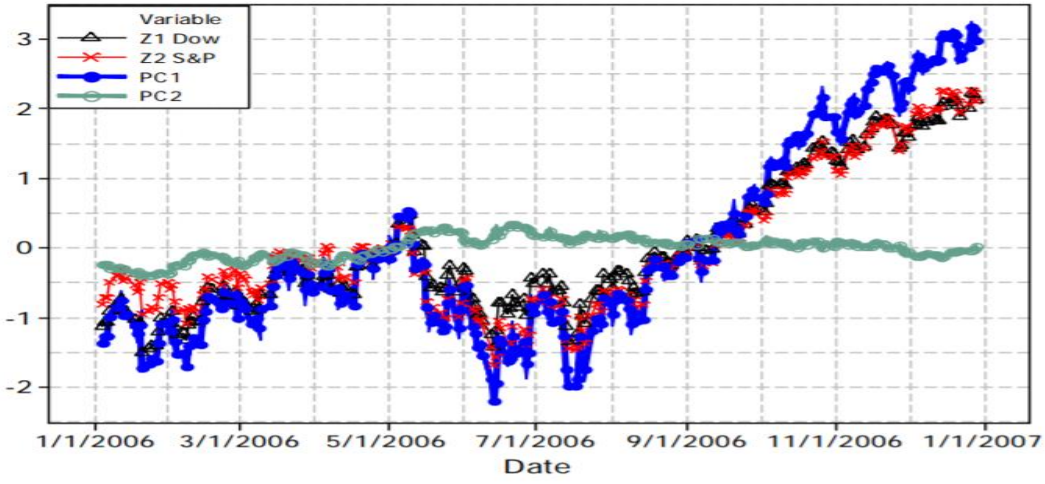
With respect to PCA, it uses linear algebra to transform the dataset into a compressed or reduced form. The number of dimensions or principal component in the transformed result can be selected, (Brownlee, 2016). The figure below represents Principal components of the daily closing values for the Dow and S&P 500 indices, (Twain, FM, 2015). To calculate principal components in such application, standardized data are multiplied by each eigenvector. For example, stock price data for two tickers, will create two new variables, the first principle component, PC1, and the second principle component PC2. The first plot below is a scatterplot of PC1 and PC2. In essence, principle components provide weights that rotate the data in 2nd figure onto a new axis. Now the spread of the data is represented along PC1, the new horizontal axis. The second component, PC2, is along the vertical axis explains the rest of the variability in the data.



1st figure: Principal components of the daily closing values for the Dow and S&P 500
Source: (Twain, FM, 2015, p. 12)



2nd figure: Standardized daily closing values for the Dow and S&P 500 indices. The first eigenvector v1, is drawn from the origin (0, 0) to the point (0.707, 0.707) and represents the direction of the most variation in the data.
Source: (Twain, FM, 2015, p. 9)



3rd figure: Principal components of the daily closing values for the Dow and S&P 500 indices.Source: (Twain, FM, 2015, p. 13)

in the 3rd figure above, it is clear that PC1 follows Z1, Z2 while PC2 is of another trend. Also PC1 seems to increase when both variables are increasing. This mens that the two dimensions can be reduced to only one, that is PC1. The code for this specific example on stock data is not available. Typically, however, as mentioned earlier, PCA performs best with a normalized feature set. So standard scalar normalization can be implemented to normalize the feature set. Performing PCA using Scikit-Learn is a two-step process:1) Initialize the PCA class by passing the number of components to the constructor. 2) Call the fit and then transform methods by passing the feature set to these methods. The transform method returns the specified number of principal components. when the number of components in the constructor is not specified, all features in the feature set will be returned for both the training and test sets. The PCA class contains "explained_variance_ratio_" which returns the variance caused by each of the principal components. Typical code is as follows:

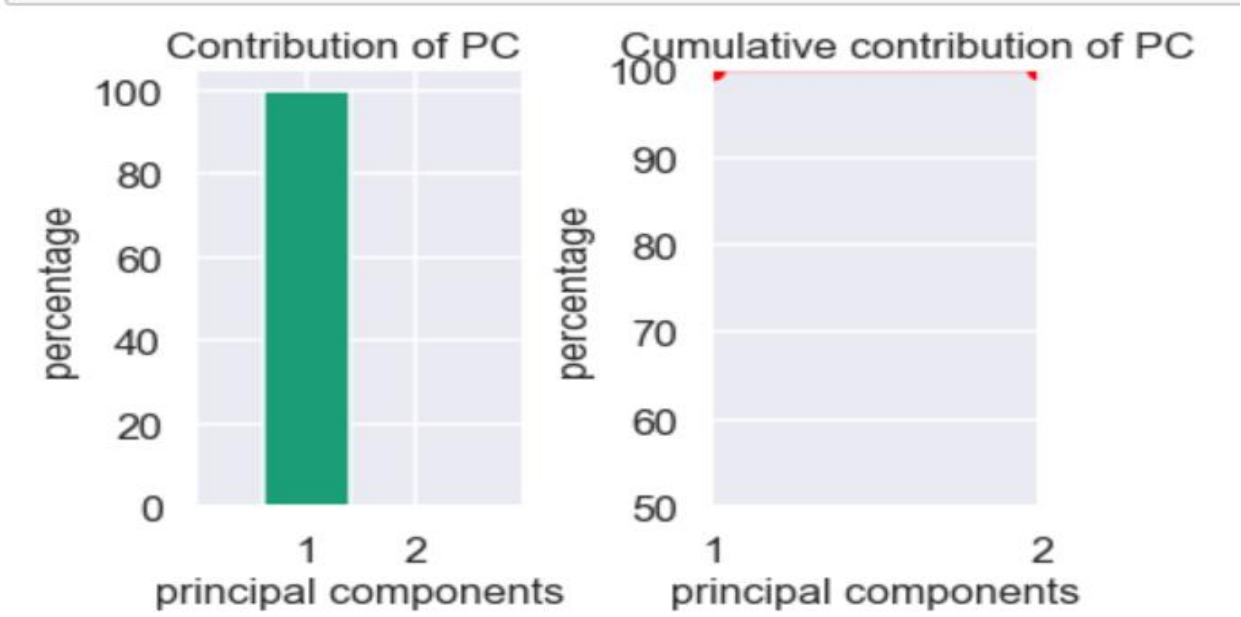
```
#https://stackabuse.com/implementing-pca-in-python-with-scikit-learn/
from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

from sklearn.decomposition import PCA

pca = PCA()
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)

explained_variance = pca.explained_variance_ratio_
```

With respect to encoding target variables using one-hot encoding, description of one-hot-encoding is provided in Appendix-II feature engineering. A one hot encoding is a representation of categorical variables as binary vectors. This first requires that the categorical values be mapped to integer values. Then, each integer value is represented as a binary vector that is all zero values

except the index of the integer, which is marked with a 1. Typically, a one hot encoding allows the representation of categorical data to be more expressive as many machine learning algorithms cannot work with categorical data directly. The categories must be converted into numbers. This is required for both input and output variables that are categorical. An integer encoding can be used either directly, rescaled where needed. There may be problems when there is no ordinal relationship and allowing the representation to lean on any such relationship might be damaging to learning to solve the problem. In these cases, the network would be given more expressive power to learn a probability-like number for each possible label value. When a one hot encoding is used for the output variable, it may offer a more nuanced set of predictions than a single label, (Brownlee, How to One Hot Encode Sequence Data in Python, 2017). Codes and output from submission are as follows:

```
# Let's implement PCA algorithm on our variables data calculated above

num_pc = 2

X = np.asarray(data)
[n,m] = X.shape
print ('The number of timestamps are '+ format(n))
print ('The number of explanatory variables are ' + format(m))

pca = PCA(n_components=num_pc) # number of principal components
pca.fit(X)

percentage =  pca.explained_variance_ratio_
percentage_cum = np.cumsum(percentage)
print (format(percentage_cum[-1]*100) + ' of the variance is explained by the first 2 PCs')

pca_components = pca.components_
```

```
The number of timestamps are 419
The number of explanatory variables are 9
99.99999999997804 of the variance is explained by the first 2 PCs
```

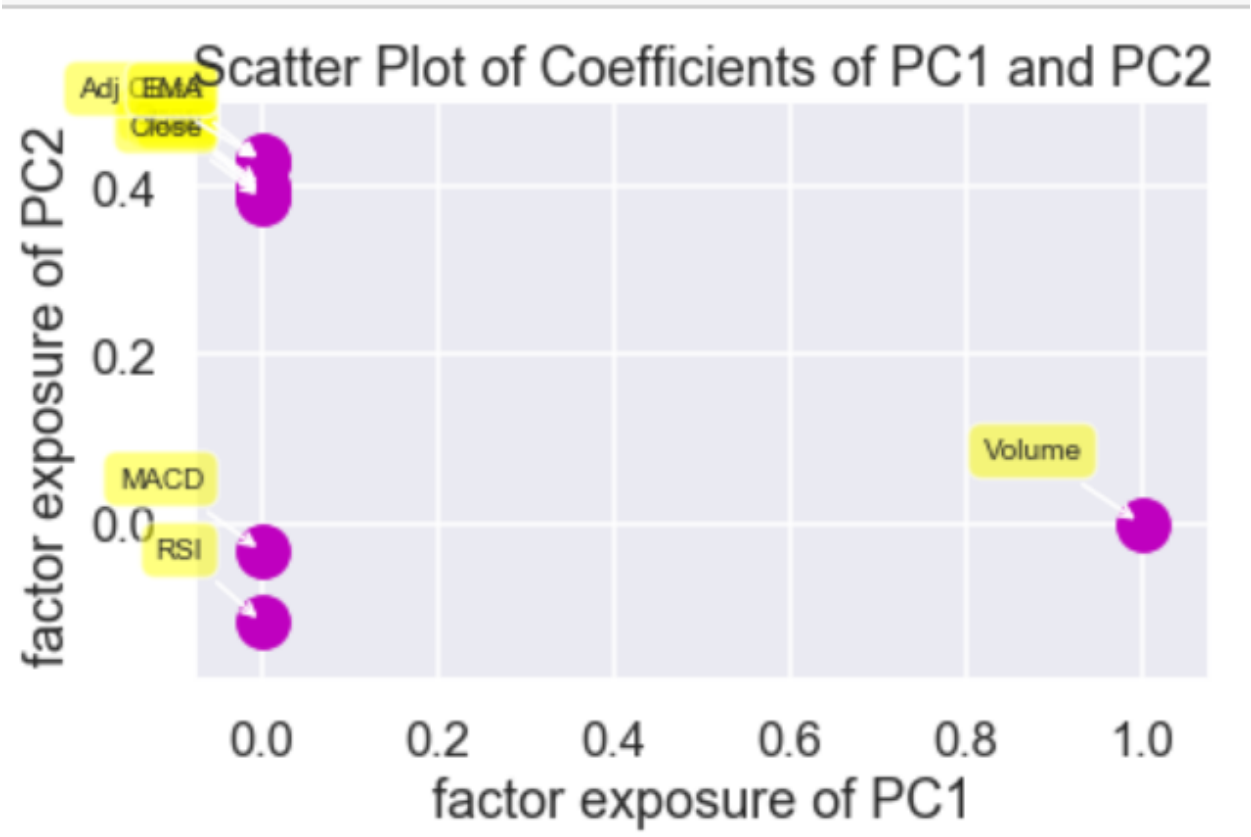



```
#constructing "statistical risk factors" from principal component calculated above
factor_returns = X.dot(pca_components.T)
factor_returns = pd.DataFrame(columns=["factor 1", "factor 2"], index=data.index, data=factor_returns)
factor_returns.head()
```

Date	factor 1	factor 2
2018-01-02	8.665570e+07	657.229347
2018-01-03	9.007040e+07	661.410188
2018-01-04	8.063640e+07	661.873259
2018-01-05	8.352400e+07	665.943853
2018-01-08	5.731920e+07	660.039017

```
#calculatinmg factor returns which are are an analogue to the principal component matrix
factor_exposures = pd.DataFrame(index=["factor 1", "factor 2"], columns=data.columns, data = pca.components_).T
factor_exposures
```

	factor 1	factor 2
Open	-1.309558e-07	3.999357e-01
High	-1.183555e-07	3.926041e-01
Low	-1.589775e-07	3.909061e-01
Close	-1.435452e-07	3.857324e-01
Adj Close	-1.528739e-07	4.301545e-01
Volume	1.000000e+00	2.832728e-07
EMA	-7.501454e-08	4.293610e-01
MACD	2.842110e-08	-3.321139e-02
RSI	-2.720672e-07	-1.175992e-01



2.3- Familiarization with the cross-validation techniques for forecasting financial time series. For example, traditional k-fold cross-validation versus walk forward analysis, and Purged K-Fold CV. Write a short paragraph explaining each technique researched. Research at least three (they don't have to be the 3 mentioned here).

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model. It is a popular method because it is simple to understand and because it

generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split. The general procedure is as follows:

- Shuffle the dataset randomly.
- Split the dataset into k groups
- For each unique group:
- Take the group as a hold out or test data set
- Take the remaining groups as a training data set
- Fit a model on the training set and evaluate it on the test set
- Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times. Appendix-IV presents details of cross validation sourced from Wikipedia, (wikipedia, n.d.). In this section additionally 3 approaches shall be re-iterated with the associated python code as sourced form (Rai, 2018) and associated code also except for the LOOCV.

The validation set approach: In this approach, 50% of the dataset is reserved for validation and the remaining 50% for model training. However, a major disadvantage of this approach is that since only 50% of the dataset is trained, there is a huge possibility that some interesting information about the data will be missed which will lead to a higher bias.

Leave one out cross validation (LOOCV): In this approach, only one data point from the available dataset is reserved and train the model on the rest of the data. This process iterates for each data point hence all data points will be used, and bias will be low. On the other hand, the cross-validation process is repeated n times (where n is number of data points) which results in a higher execution time. This approach leads to higher variation in testing model effectiveness

because as being tested against one data point. On the other hand, estimation gets highly influenced by the data point. So, if the data point turns out to be an outlier, it can lead to a higher variation.

Stratified k-fold cross validation: Stratification is the process of rearranging the data to ensure that each fold is a good representative of the whole. Since the model must be trained on a large portion of the dataset, to avoid bias in trend identification, a good ratio of testing data points will be required. The training and testing process multiple times should be iterated the train and test dataset distribution should be changed. This helps in validating the model effectiveness properly. The "k-fold cross validation" meets these requirements. Choosing the value of k is challenging as a lower value of k is more biased, and hence undesirable. On the other hand, a higher value of K is less biased, but can suffer from large variability. It is important to know that a smaller value of k always takes us towards validation set approach, whereas a higher value of k leads to LOOCV approach. Precisely, LOOCV is equivalent to n-fold cross validation where n is the number of training examples.

In [2]:
```
#https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/
#The validation set approach
train, validation = train_test_split(data, test_size=0.50, random_state = 5)

#https://datascience.stackexchange.com/questions/30607/leave-one-out-cross-validation-using-sklearn-multiple-csv
#leave-one-out-cross-validation-using-sklearn-multiple-csv
import glob
import numpy as np
import pandas as pd
from sklearn.cross_validation import LeaveOneOut
path=r'.....................\Data\New
    design process data'
filelist=glob.glob(path + "/*.csv")
loo=LeaveOneOut(n=52)
for train,test in loo.split(filelist):
    print("%s %s" % (train, test))

#https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/
#k-fold cross validation with repition
from sklearn.model_selection import RepeatedKFold
rkf = RepeatedKFold(n_splits=5, n_repeats=10, random_state=None)
# X is the feature set and y is the target
for train_index, test_index in rkf.split(X):
     print("Train:", train_index, "Validation:", val_index)
     X_train, X_test = X[train_index], X[val_index]
     y_train, y_test = y[train_index], y[val_index]
```

3. References

Brownlee, J. (2016, 20 May). *Machine learning Mastery*. Retrieved from https://machinelearningmastery.com/feature-selection-machine-learning-python/

Brownlee, J. (2017). *machine learning mastry*. Retrieved from machinelearningmastery.com: https://machinelearningmastery.com/how-to-one-hot-encode-sequence-data-in-python/

Gray, C. (2018, July). *The Alpha Scientist*. Retrieved from The Alpha Scientist: https://alphascientist.com/feature_engineering.html

Hudson & Thames. (2019). mlfinlab, Release 0.4.1. Retrieved 9 9, 2019, from https://buildmedia.readthedocs.org/media/pdf/mlfinlab/latest/mlfinlab.pdf

Joubert, J. F. (2018, May 7). Create Financial Data Structures: Time, Tick, Volume, and Dollar Bars. (Version of Pycharm Professional Edition: 2017.3). Retrieved from https://github.com/Jackal08/financial-data-structures

Martinez, G. (2019, April 24). Advanced candlesticks for machine learning. Retrieved from https://towardsdatascience.com/advanced-candlesticks-for-machine-learning-i-tick-bars-a8b93728b4c5

Pardo, M. L. (2018). Advances in financial Machine learning. John Wiley and Sons.

Rai, S. (2018, May). *analytics vidhya*. Retrieved from www.analyticsvidhya.com: https://www.analyticsvidhya.com/blog/2018/05/improve-model-performance-cross-validation-in-python-r/

Rencberoglu, E. (2019, April 1). *Towards data science*. Retrieved from https://towardsdatascience.com/feature-engineering-for-machine-learning-3a5e293a5114

Sean. (2018, February 5). *Worrier Trading Blog*. Retrieved from Worrier Trading: https://www.warriortrading.com/riding-the-trend-top-indicators-for-trend-trading/

Shetye, A. (2019, February 11). *Towards data Science*. Retrieved from Towards data Science: https://towardsdatascience.com/feature-selection-with-pandas-e3690ad8504b

Twain, FM. (2015). *semantic scholar org*. Retrieved from https://pdfs.semanticscholar.org/db59/f01a2e8c0aadb95488fbbe4ea4590790de42.pdf

wikipedia. (n.d.). *Cross-validation (statistics)*. Retrieved from wikipedia: https://en.wikipedia.org/wiki/Cross-validation_(statistics)

Williams, R. (2015, January 22). Serial Correlation [Optional; Very brief overview]. University of Notre Dame. Retrieved from https://www3.nd.edu/~rwilliam/

Appendix-I

Top Four Trend Trading Strategy

Trend trading is one of the popular strategies in the current investing world where investors are amplifying price movements by trading with the momentum of the market. Trend trading relies on key technical indicators to gauge the strength, persistence and likely continuation of any trend that an investor intends to trade on. Out of the entire technical analysis toolkit, these are the top 4 indicators for trend trading, these are:

Moving Averages (MA), where a progressive average price for a set number of past day (or hours, months, years, etc) is used. Every point on a moving average line is the average for that day, which makes for a smooth representation of a price's movement. There are a number of popular configurations for moving averages, but they can be created for any time frame and for any price (closing, high, low, etc). Traders use moving averages to identify trends, points of resistance and crossovers between different moving average lines, among many other techniques. When calculating M days simple moving average (SMA), the first M−1 are not valid, as M prices are required for the first moving average data point.

Moving Average Convergence Divergence (MACD), is an oscillating indicator that fluctuates around zero and is a measure of both trend and momentum. The calculation of the MACD follows the same logic as a simple moving average but incorporates additional features to give a better picture of a more recent moving average compared to an older one. When the MACD crosses over into positive territory it is seen as a buy signal, and the opposite holds for negative territory. The MACD is usually used as a complement for other technical indicators, and not as a stand-alone indicator in trend trading. It is calculated as MACD= (12-day EMA - 26-day EMA)

The relative strength index (RSI), is an oscillator that attempts to measure excessive sentiment in a trending stock. RSI values range from 0 to 100. RSI values of 70 or above indicate that a security is becoming overbought or overvalued, and therefore may be primed for a trend reversal or corrective pullback in price. On the other side, an RSI reading of 30 or below is commonly interpreted as indicating an oversold or undervalued condition that may signal a trend change or corrective price reversal to the upside. For example, a stock with a strong trend and an RSI of 60 likely has a little more way to go before stopping or correcting downward. The RSI is one of the best complimentary indicators available for trend trading. The RSI is calculated as follows:
RSI = 100 - 100 / (1 + RS)RS = Average gain of last 14 trading days / Average loss of last 14 trading days

The on-balance volume (OBV), is an indicator that measures the volume trend for a security. Volume is an important complimentary measure that is used to confirm price trends by determining whether they are occurring on a high or low number of trades. Generally, a high number of trades accompanying an upward trend is a supporting signal for that trend, and the same for a low number of trades with a downward trend, (Sean, 2018).Investopedia provides a summary and historical background to the indicator, "On-balance volume (OBV) is a momentum indicator that uses volume flow to predict changes in stock price. Joseph Granville first developed the OBV metric in the 1960s. He believed that when volume increases sharply without a significant change in the stock's price, the price will eventually jump upward, and vice versa". On balance volume always builds on its own previous value. In other words, it is a cumulative indicator. If the close price increases, then volume is added to the previous indicator value. If the close price decreases volume is subtracted from the previous indicator value. It

should be noted that it compares the current close value with the close [-1] (previous close) and not the current open value vs current close value. This means that a green candle day can exist and still deduct volume. An example of this might be caused when price gaps down and closes up.

Appendix II

Feature Engineering

All machine learning algorithms use some input data to create outputs. This input data comprises features, which are usually in the form of structured columns. Algorithms require features with some specific characteristic to work properly. Here, the need for feature engineering arises with mainly two goals:

- Preparing the proper input dataset, compatible with the machine learning algorithm requirements.
- Improving the performance of machine learning models.

The features used influence more than everything else the result. No algorithm alone can supplement the information gain given by correct feature engineering, (Rencberoglu, 2019). Some feature engineering techniques might work better with some algorithms or datasets, while some of them might be beneficial in all cases. The recommended way to achieve expertise in feature engineering is practicing different techniques on various datasets and observing their effect on model performances. Feature engineering techniques are:

1.Imputation: missing values are one of the most common problems you can encounter when you try to prepare your data for machine learning. The reason for the missing values might be human errors, interruptions in the data flow, privacy concerns, and so on. Whatever is the reason, missing values affect the performance of the machine learning models. Some machine learning platforms automatically drop the rows which include missing values in the model training phase, and it decreases the model performance because of the reduced training size. On the other hand, most of the algorithms do not accept datasets with missing values and gives an error. The simplest solution to the missing values is to drop the rows or the entire column. There is not an optimum threshold for dropping but you can use 70% as an example value and try to drop the rows and columns which have missing values with higher than this threshold.

Numerical Imputation: Imputation is a preferable option rather than dropping because it preserves the data size. However, there is an important selection of what you impute to the missing values. I suggest beginning with considering a possible default value of missing values in the column. For example, if you have a column that only has 1 and NA, then it is likely that the NA rows correspond to 0. For another example, if you have a column that shows the "customer visit count in last month", the missing values might be replaced with 0 as long as you think it is a sensible solution. Another reason for the missing values is joining tables with different sizes and in this case, imputing 0 might be reasonable as well. Except for the case of having a default value for missing values, I think the best imputation way is to use the medians of the columns. As the averages of the columns are sensitive to the outlier values, while medians are more solid in this respect.

Categorical Imputation: replacing the missing values with the maximum occurred value in a column is a good option for handling categorical columns. But if you think the values in the column are distributed uniformly and there is not a dominant value, imputing a category like "Other" might be more sensible, because in such a case, your imputation is likely to converge a random selection.

2.Handling Outliers: before mentioning how outliers can be handled, the best way to detect the outliers is to demonstrate the data visually. Statistical methodologies are less precise but on the other hand, they have a superiority, being fast. Two of the different ways of outliers detection; standard deviation, and percentiles.

Outlier Detection with Standard Deviation: if a value has a distance to the average higher than x * standard deviation, it can be assumed as an outlier. There is no trivial solution for x, but usually, a value between 2 and 4 seems practical. In addition, z-score can be used instead of the formula above. Z-score (or standard score) standardizes the distance between a value and the mean using the standard deviation.

Outlier Detection with Percentiles: a certain percent of the value from the top or the bottom can be assumed as an outlier. The key point is here to set the percentage value once again, and this depends on the distribution of your data as mentioned earlier. Additionally, a common mistake is using the percentiles according to the range of the data. In other words, if data ranges from 0 to 100, your top 5% is not the values between 96 and 100. Top 5% means here the values that are out of the 95th percentile of data.

Another option for handling outliers is to cap them instead of dropping. So data size cab ne maintained if better for the final model performance. On the other hand, capping can affect the distribution of the data, thus it better not to exaggerate it.

3.Binning: Binning can be applied on both categorical and numerical data. The main motivation of binning is to make the model more robust and prevent overfitting, however, it has a cost to the performance. Every time you bin something, you sacrifice information and make your data more regularized. The trade-off between performance and overfitting is the key point of the binning process. For numerical columns, except for some obvious overfitting cases, binning might be redundant for algorithms, due to its effect on model performance. However, for categorical columns, the labels with low frequencies probably affect the robustness of statistical models negatively. Thus, assigning a general category to these less frequent values helps to keep the robustness of the model.

4.Log Transform: is one of the most commonly used mathematical transformations in feature engineering. It helps to handle skewed data and after transformation, the distribution becomes more approximate to normal. In most of the cases the magnitude order of the data changes within the range of the data.

5.One-hot encoding: is one of the most common encoding methods in machine learning. This method spreads the values in a column to multiple flag columns and assigns 0 or 1 to them. These binary values express the relationship between grouped and encoded column. This method changes your categorical data, which is challenging to understand for algorithms, to a numerical format and enables you to group your categorical data without losing any information. If you have N distinct values in the column, it is enough to map them to N-1 binary columns, because the missing value can be deducted from other columns. If all the columns in our hand are equal to 0, the missing value must be equal to 1.

6.Grouping Operations: in most machine learning algorithms, every instance is represented by a row in the training dataset, where every column shows a different feature of the instance. This kind of data called “Tidy”. Datasets such as transactions rarely fit the definition of tidy data above, because of the multiple rows of an instance. In such a case, we group the data by the instances and then every instance is represented by only one row. The key point of group by operations is to decide the aggregation functions of the features. For numerical features, average and sum functions are usually convenient options, whereas for categorical features it more complicated.

Categorical Column Grouping: three different ways are suggested for aggregating categorical columns: The first option is to select the label with the highest frequency. In other words, this is

the max operation for categorical columns, but ordinary max functions generally do not return this value, a lambda function is to be used for this purpose.

Pivot table: this approach resembles the encoding method in the preceding step with a difference. Instead of binary notation, it can be defined as aggregated functions for the values between grouped and encoded columns.

Numerical Column Grouping: these are grouped using sum and mean functions in most of the cases. Both can be preferable according to the meaning of the feature.

7.Feature Split: splitting features is a good way to make them useful in terms of machine learning. Most of the time the dataset contains string columns that violates tidy data principles. By extracting the utilizable parts of a column into new features:

- We enable machine learning algorithms to comprehend them.
- Make possible to bin and group them.
- Improve model performance by uncovering potential information.

Split function is a good option, however, there is no one way of splitting features. It depends on the characteristics of the column, how to split it.

The example above handles the names longer than two words by taking only the first and last elements and it makes the function robust for corner cases, which should be regarded when manipulating strings like that.

8.Scaling: in most cases, the numerical features of the dataset do not have a certain range and they differ from each other. In real life, it is nonsense to expect age and income columns to have the same range. But from the machine learning point of view, scaling solves the problem of how these two columns can be compared. The continuous features become identical in terms of the range, after a scaling process. This process is not mandatory for many algorithms, but it might be still nice to apply. However, the algorithms based on distance calculations such as k-NN or k-Means need to have scaled continuous features as model input. Basically, there are two common ways of scaling: **Normalization**: (or min-max normalization) scale all values in a fixed range between 0 and 1. This transformation does not change the distribution of the feature and due to the decreased standard deviations, the effects of the outliers increases. Therefore, before normalization, it is recommended to handle the outliers. **Standardization** (or z-score normalization) scales the values while considering standard deviation. If the standard deviation of features is different, their range also would differ from each other. This reduces the effect of the outliers in the features.

9.Extracting Date: though date columns usually provide valuable information about the model target, they are neglected as an input or used nonsensically for the machine learning algorithms. It might be the reason for this, that dates can be present in numerous formats, which make it hard to understand by algorithms, even they are simplified to a format like "01–01–2017". Building an ordinal relationship between the values is very challenging for a machine learning algorithm if you leave the date columns without manipulation. There are three recommended types of preprocessing for dates:

- **Extracting the parts of the date into different columns**: Year, month, day, etc.
- **Extracting the time period between the current date and columns in terms of years, months, days, etc.**
- **Extracting some specific features from the date: Name of the weekday, Weekend or not, holiday or not, etc**.

If the date column is transformed into the extracted columns like above, the information of them become disclosed and machine learning algorithms can easily understand them.

Appendix-III

Feature selection

Feature selection is a process where you automatically select those features in the data that contribute most to the prediction variable or output in which you are interested. Having irrelevant features in your data can decrease the accuracy of many models, especially linear algorithms like linear and logistic regression. Three benefits of performing feature selection before modeling your data are: Reduces Overfitting: Less redundant data means less opportunity to make decisions based on noise, Improves Accuracy: Less misleading data means modeling accuracy improves, Reduces Training Time: Less data means that algorithms train faster. (Brownlee, Feature Selection For Machine Learning in Python, 2016) highlights that there are mainly 4 different automatic feature selection techniques for which example python codes are provided below:

1. Univariate Selection.

In [1]:
```
# Feature Extraction with Univariate Statistical Tests (Chi-squared for classification)
import pandas
import numpy
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
# Load data
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = pandas.read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
test = SelectKBest(score_func=chi2, k=4)
fit = test.fit(X, Y)
# summarize scores
numpy.set_printoptions(precision=3)
print(fit.scores_)
features = fit.transform(X)
# summarize selected features
print(features[0:5,:])
```

```
[ 111.52  1411.887   17.605   53.108 2175.565  127.669    5.393  181.304]
[[148.    0.   33.6  50. ]
 [ 85.    0.   26.6  31. ]
 [183.    0.   23.3  32. ]
 [ 89.   94.   28.1  21. ]
 [137.  168.   43.1  33. ]]
```

You can see the scores for each attribute and the 4 attributes chosen (those with the highest scores): plas, test, mass and age.

2. Recursive Feature Elimination.

In [2]:
```
# Feature Extraction with RFE(recursive feature elimination)
from pandas import read_csv
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load data
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = LogisticRegression(solver='lbfgs')
rfe = RFE(model, 3)
fit = rfe.fit(X, Y)
print("Num Features: %d" % fit.n_features_)
print("Selected Features: %s" % fit.support_)
print("Feature Ranking: %s" % fit.ranking_)
```

```
Num Features: 3
Selected Features: [ True False False False False  True  True False]
Feature Ranking: [1 2 4 5 6 1 1 3]
```

```
C:\Users\Hanaa\Desktop\python\lib\site-packages\sklearn\linear_model\logistic.py:947: ConvergenceWarning: lbfgs failed to co
nverge. Increase the number of iterations.
  "of iterations.", ConvergenceWarning)
```

In [5]:
```
# Recursive Feature Elimination
from sklearn import datasets
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression
# load the iris datasets
dataset = datasets.load_iris()
# create a base classifier used to evaluate a subset of attributes
model = LogisticRegression()
# create the RFE model and select 3 attributes
rfe = RFE(model, 3)
rfe = rfe.fit(dataset.data, dataset.target)
# summarize the selection of the attributes
print(rfe.support_)
print(rfe.ranking_)
```

```
[False  True  True  True]
[2 1 1 1]
```

3. Principle Component Analysis.

In [3]:
```
# Feature Extraction with PCA
import numpy
from pandas import read_csv
from sklearn.decomposition import PCA
# load data
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
pca = PCA(n_components=3)
fit = pca.fit(X)
# summarize components
print("Explained Variance: %s" % fit.explained_variance_ratio_)
print(fit.components_)
```

```
Explained Variance: [0.889 0.062 0.026]
[[-2.022e-03  9.781e-02  1.609e-02  6.076e-02  9.931e-01  1.401e-02
   5.372e-04 -3.565e-03]
 [-2.265e-02 -9.722e-01 -1.419e-01  5.786e-02  9.463e-02 -4.697e-02
  -8.168e-04 -1.402e-01]
 [-2.246e-02  1.434e-01 -9.225e-01 -3.070e-01  2.098e-02 -1.324e-01
  -6.400e-04 -1.255e-01]]
```

You can see that the transformed dataset (3 principal components) bare little resemblance to the source data.

4. Feature Importance, (Brownlee, Feature Selection For Machine Learning in Python, 2016).

In [4]:
```
# Feature Importance with Extra Trees Classifier
from pandas import read_csv
from sklearn.ensemble import ExtraTreesClassifier
# load data
url = "https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv"
names = ['preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age', 'class']
dataframe = read_csv(url, names=names)
array = dataframe.values
X = array[:,0:8]
Y = array[:,8]
# feature extraction
model = ExtraTreesClassifier(n_estimators=10)
model.fit(X, Y)
print(model.feature_importances_)
```

```
[0.109 0.237 0.1   0.079 0.072 0.13  0.117 0.155]
```

```
You can see that we are given an importance score for each attribute where the larger score the more important the attribute.
The scores suggest at the importance of plas, age and mass.
```

In [12]:
```
# Feature Importance
from sklearn import datasets
from sklearn import metrics
from sklearn.ensemble import ExtraTreesClassifier
# load the iris datasets
dataset = datasets.load_iris()
# fit an Extra Trees model to the data
model = ExtraTreesClassifier()
model.fit(dataset.data, dataset.target)
# display the relative importance of each attribute
print(model.feature_importances_)
```

```
C:\Users\Hanaa\Desktop\python\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_estima
tors will change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

```
[0.025 0.066 0.399 0.511]
```

Above codes are either sourced as indicated or developed by self-based on previous assignments

Appendix-IV

Cross validation

There are two types of cross-validation: exhaustive and non-exhaustive cross-validation.

Exhaustive cross-validation methods are those which learn and test on all possible ways to divide the original sample into a training and a validation set. They include: 1)**Leave-p-out cross-validation** which involves using p observations as the validation set and the remaining observations as the training set. 2)**Leave-one-out cross-validation (LOOCV)** is a case of leave-p-out cross-validation with p = 1. The process looks like jackknife; however, with cross-validation one computes a statistic on the left-out sample(s), while with jackknifing one computes a statistic from the kept samples only.

Non-exhaustive cross-validation methods do not compute all ways of splitting the original sample. Those methods are approximations of leave-p-out cross-validation. They include; 1) **k-Fold Cross-Validation** is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining $k - 1$ subsamples are used as training data. The cross-validation process is then repeated k times, with each of the k subsamples used exactly once as the validation data. The k results can then be averaged to produce a single estimation. The advantage of this method over repeated random sub-sampling (see below) is that all observations are used for both training and validation, and each observation is used for validation exactly once. When k = n (the number of observations), the k-fold cross-validation is exactly the leave-one-out cross-validation. In stratified k-fold cross-validation, the folds are selected so that the mean response value is approximately equal in all the folds. In the case of binary classification, this means that each fold contains roughly the same proportions of the two types of class labels. 2)**Holdout method**, data points are randomly assigned to two sets d0 and d1, usually called the training set and the test set, respectively. The size of each of the sets is arbitrary although typically the test set is smaller than the training set. We then train (build a model) on d0 and test (evaluate its performance) on d1. In typical cross-validation, results of multiple runs of model-testing are averaged together; in contrast, the holdout method, in isolation, involves a single run. It should be used with caution because without such averaging of multiple runs, one may achieve highly misleading results. One's indicator of predictive accuracy (F*) will tend to be unstable since it will not be smoothed out by multiple iterations. Similarly, indicators of the specific role played by various predictor variables (e.g., values of regression coefficients) will tend to be unstable. While the holdout method can be framed as "the simplest kind of cross-validation", many sources instead classify holdout as a type of simple validation, rather than a simple or degenerate form of cross-validation. 3) **Repeated random sub-sampling validation method**, also known as Monte Carlo cross-validation, creates multiple random splits of the dataset into training and validation data. For each such split, the model is fit to the training data, and predictive accuracy is assessed using the validation data. The results are then averaged over the splits. The advantage of this method (over k-fold cross validation) is that the proportion of the training/validation split is not dependent on the number of iterations (folds). The disadvantage of this method is that some observations may never be selected in the validation

subsample, whereas others may be selected more than once. In other words, validation subsets may overlap. This method also exhibits Monte Carlo variation, meaning that the results will vary if the analysis is repeated with different random splits. As the number of random splits approaches infinity, the result of repeated random sub-sampling validation tends towards that of leave-p-out cross-validation. In a stratified variant of this approach, the random samples are generated in such a way that the mean response value (i.e. the dependent variable in the regression) is equal in the training and testing sets. This is particularly useful if the responses are dichotomous with an unbalanced representation of the two response values in the data.

Nested cross-validation method is used simultaneously for selection of the best set of hyperparameters and for error estimation (and assessment of generalization capacity). At least two variants can be distinguished: **k*l-fold cross-validation**. This is a truly nested variant, which contains an outer loop of k folds and an inner loop of l folds. The total data set is split in k sets. One by one, a set is selected as (outer) test set and the k-1 other sets are combined into the corresponding outer training set. This is repeated for each of the k sets. Each outer training set is further sub-divided into l sets. One by one, a set is selected as inner test (validation) set and the l-1 other sets are combined into the corresponding inner training set. This is repeated for each of the l sets. The inner training sets are used to fit model parameters, while the outer test set is used as a validation set to provides an unbiased evaluation of the model fit. Typically, this is repeated for many different hyperparameters (or even different model types) and the validation set is used to determine the best hyperparameter set (and model type) for this inner training set. After this, a new model is fit on the entire outer training set, using the best set of hyperparameters from the inner cross-validation. The performance of this model is then evaluated using the outer test set. **k-fold cross-validation with validation and test set**, is a type of k*l-fold cross-validation when l=k-1. A single k-fold cross-validation is used with both a validation and test set. The total data set is split in k sets. One by one, a set is selected as test set. Then, one by one, one of the remaining sets is used as a validation sets and the other k-2 sets are used as training sets until all possible combinations have been evaluated. Like the k*l-fold cross validation, the training set is used for model fitting and the validation set is used for model evaluation for each of the hyperparameter sets. Finally, for the selected parameter set, the test set is used to evaluate the model with the best parameter set. Here, two variants are possible: either evaluating the model that was trained on the training set or evaluating a new model that was fit on the combination of the train and the validation set, (wikipedia, n.d.).