Final V2

February 9, 2022

1 Final Project

1.1 May 3rd, 2021

1.1.1 STAT4050

1.1.2 Using Machine Learning to Predict Stock Prices

1.1.3 Jonathan Miranda

1.1.4 Professor Shepanik

How can we use Linear Regression and Machine Learning to predict stocks? Predicting how the stock market will perform is one of the most difficult things to do. There are so many factors involved in the prediction – physical factors vs. physhological, rational and irrational behaviour, etc. All these aspects combine to make share prices volatile and very difficult to predict with a high degree of accuracy.

Can we use machine learning as a game changer in this domain? Using features like the latest announcements about an organization, their quarterly revenue results, etc., machine learning techniques have the potential to unearth patterns and insights we didn't see before, and these can be used to make unerringly accurate predictions.

First, we have to talk about confidence intervals when we use linear regression because it isn't always correct. It's a guess based on an algorithm that reads in copious amounts of data. But our confidence intervals are calculated ranges/boundaries that are applied to a certain parameter. This parameter is usually supported heavily with mathematics behind it. When we look at using it with predicting stock prices, we can expect a certain range of values to fall within our confidence interval. If it doesn't fall within that interval we can reject that value as null. Any null hypothesis that we fail to reject would be a reasonable value based on our data, these values will appear inside our confidence interval. So when it comes to stocks we can't fully predict the future of what a value will be, but we can get a good estimate.

Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things: (1) does a set of predictor variables do a good job in predicting an outcome (dependent) variable? (2) Which variables in particular are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable? These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. The simplest form of the regression equation with one dependent and one independent variable is defined by the formula y = c + b*x, where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable.

When selecting the model for the analysis, an important consideration is model fitting. Adding independent variables to a linear regression model will always increase the explained variance of the model (typically expressed as R^2). However, overfitting can occur by adding too many variables to the model, which reduces model generalizability. A simple model is usually preferable to a more complex model. Statistically, if a model includes a large number of variables, some of the variables will be statistically significant due to chance alone.

1.2 Confidence Interval

1.2.1 Sample Error

Lets set our population to be the daily return of the S&P 500 from January 1st, 2010 to the present day of writing this which is May 3rd, 2021. We can take the recent 10 daily returns to calculate the mean, but will this value be the same as the population mean? What if we increase the sample size to 100 or even 1000?

YOU WILL NEED TO ADD THE pandas_datareader PACKAGE IF YOU WANT THIS CODE TO WORK FOR YOU!

[49]:
```
### First we import our needed libraries
import datetime as dt
import matplotlib.pyplot as plt
from matplotlib import style
import pandas as pd
import pandas_datareader.data as web
import numpy as np
import statsmodels.formula.api as sm

### Make Our graphs look good
style.use('ggplot')

#get SPY data from August 2010 to the present
start = dt.datetime(2010, 1, 1)
end = dt.datetime.now()
df = web.DataReader("SPY", 'yahoo', start, end)
df.reset_index(inplace=True)
df.set_index("Date", inplace=True)
### We want only the open and close columns
spy_total = df[['Open','Close']]

print(spy_total.head())

#calculate log returns
spy_log_return = np.log(spy_total.Close).diff().dropna()
print('Population mean:', np.mean(spy_log_return))
print('Population standard deviation:',np.std(spy_log_return))
```

```
                 Open       Close
Date
```

```
2010-01-04  112.370003  113.330002
2010-01-05  113.260002  113.629997
2010-01-06  113.519997  113.709999
2010-01-07  113.500000  114.190002
2010-01-08  113.889999  114.570000
Population mean: 0.0004555836611185909
Population standard deviation: 0.010894589213800876
```

Next lets look at the last 10 days sample and the last 1000 days sample

[50]:
```
print('10 days sample returns:', np.mean(spy_log_return.tail(10)))
print('10 days sample standard deviation:', np.std(spy_log_return.tail(10)))
print('1000 days sample returns:', np.mean(spy_log_return.tail(1000)))
print('1000 days sample standard deviation:', np.std(spy_log_return.tail(1000)))
```

```
10 days sample returns: -7.694148666494982e-05
10 days sample standard deviation: 0.005759735672514859
1000 days sample returns: 0.0005481959626308309
1000 days sample standard deviation: 0.013002562157467945
```

The two samples has different means and variances as we expected.

To estimate the entire population mean we need to define our standard error of the mean. Which is:

$$SE = \frac{\sigma}{\sqrt{n}}$$

Where σ is the standard deviation and n as the sample size. If we want to estimate the population of a certain interval so that 95% of the time it will contain the population mean, the interval is calculated as:

$$(\mu - 1.96 * SE, \mu + 1.96 * SE)$$

Where μ is the sample mean and SE is the standard error. This interval is the confidence interval. The 1.96 comes from a 95% confidence interval when we assume the sample mean is normally distributed.

[51]:
```
### Apply the formula above to calculate confidence interval
bottom_1 = np.mean(spy_log_return.tail(10))-1.96*np.std(spy_log_return.
 ↪tail(10))/(np.sqrt(len((spy_log_return.tail(10)))))
upper_1 = np.mean(spy_log_return.tail(10))+1.96*np.std(spy_log_return.tail(10))/
 ↪(np.sqrt(len((spy_log_return.tail(10)))))
bottom_2 = np.mean(spy_log_return.tail(1000))-1.96*np.std(spy_log_return.
 ↪tail(1000))/(np.sqrt(len((spy_log_return.tail(1000)))))
upper_2 = np.mean(spy_log_return.tail(1000))+1.96*np.std(spy_log_return.
 ↪tail(1000))/(np.sqrt(len((spy_log_return.tail(1000)))))

###Print the Outcomes
```

```
print('10 days 95% confidence interval:', (bottom_1,upper_1))
print('1000 days 95% confidence interval:', (bottom_2,upper_2))
```

```
10 days 95% confidence interval: (-0.0036468626420160023, 0.003492979668686103)
1000 days 95% confidence interval: (-0.0002577111893451975,
0.0013541031146068593)
```

Our 95% confidence interval became much smaller when we made the sample size much larger with regards going from 10 to 1000 days.

Normal Distribution inside of our Confidence Interval Normal distribution is commonly used that we have tables with some of the most common critical values of it. The most common values we look for are 90%, 95%, and 99% for a confidence level. The critical values for those confidence levels are as follows: 1.64, 1.96, and 2.32 respectively. For these values we can figure out our upper and lower bounds as follows:

$$UpperBound = \mu + 1.64 * SE$$
$$LowerBound = \mu + 1.64 * SE$$

Where the 1.64 can be replaced for the critical value that you are looking for. This was used above when comparing the 10 and 1000 day confidence intervals.

When we are using the 1.96 critical value for the 95% confidence interval, can we assume that this mean follows a normal distribution? Yes, we can. The is supported by the **central limit theorem**. This theorem tells us that a very large sample size from a given population will be almost equal to the mean of said population. Also that the means of the samples will also be approximately normally distributed.

1.2.2 Simple Linear Regression

In the finance and economics field, most of the models are linear ones. We can see linear regression everywhere, from the foundation of the model portfolio theory to the nowadays popular Fama-French asset pricing model. It's very important to understand how linear regression works in order to have a comprehensive understanding of those theories.

If we are holding a stock, we must be curious about the relationship between our stock return and the market return. Let's say we hold Tesla stock on the first day of last year. In order to see the relation directly, we plot the daily return of our stock on the y-axis and plot the S&P 500 index daily return on the x-axis. Then we will plot it.

[52]:
```
### Get our Tesla and S&P500 data
spy_table = web.DataReader("SPY", 'yahoo', start, end)
spy_table.reset_index(inplace=True)
spy_table.set_index("Date", inplace=True)

tesla_table = web.DataReader("TSLA", 'yahoo', start, end)
tesla_table.reset_index(inplace=True)
tesla_table.set_index("Date", inplace=True)
```

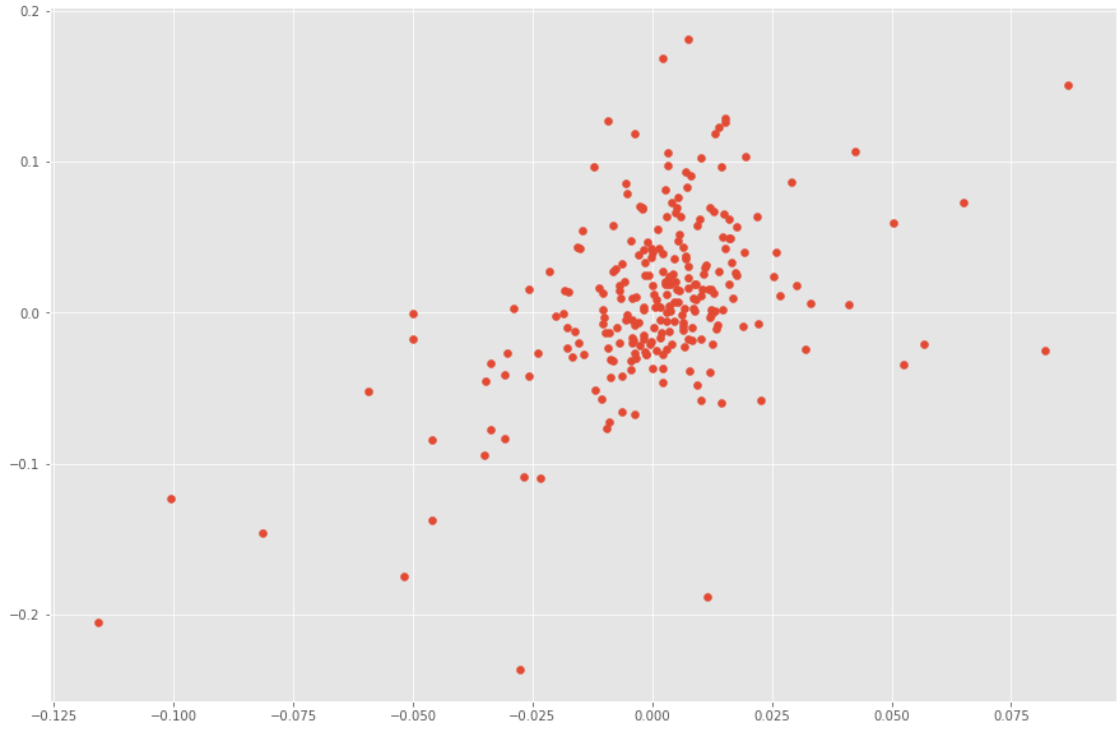
```
#fetch data from Jan 1st 2020 to December 31st 2020
spy = spy_table.loc['2020-1-1':'2021-1-1',['Close']]
tesla = tesla_table.loc['2020-1-1':'2021-1-1',['Close']]

spy_total = spy_table[['Open','Close']]
tesla_total = tesla_table[['Open','Close']]

#calculate log return
spy_log = np.log(spy.Close).diff().dropna()
tesla_log = np.log(tesla.Close).diff().dropna()
df = pd.concat([spy_log,tesla_log],axis = 1).dropna()
df.columns = ['spy','tesla']
print(df.head())
print(df.tail())
```

```
                spy     tesla
Date
2020-01-03 -0.007601  0.029203
2020-01-06  0.003808  0.019072
2020-01-07 -0.002816  0.038067
2020-01-08  0.005315  0.048033
2020-01-09  0.006758 -0.022189
                spy     tesla
Date
2020-12-24  0.003883  0.024150
2020-12-28  0.008554  0.002897
2020-12-29 -0.001910  0.003459
2020-12-30  0.001426  0.042321
2020-12-31  0.005068  0.015552
```

[53]:
```
plt.figure(figsize = (15,10))
plt.scatter(df.spy,df.tesla)
plt.show()
```



We see that the plot is scattered, but there is a general correlation between the two. The higher that the S&P 500's daily return is, the higher Tesla's daily return is. This shows that there is positive correlation between the two.

To truly model the relation between the rates of return we need a slope and an intercept to model it with a straight line, that is called **Linear Regression**. In order to model this we need the best straight line possible for the data. The objective is to make the sum of the squared residuals as small as possible. We can do this with the OLD method. X and Y are used to represent the two variables. The linear relation between them is:

$$Y = \alpha + \beta * x + \epsilon$$

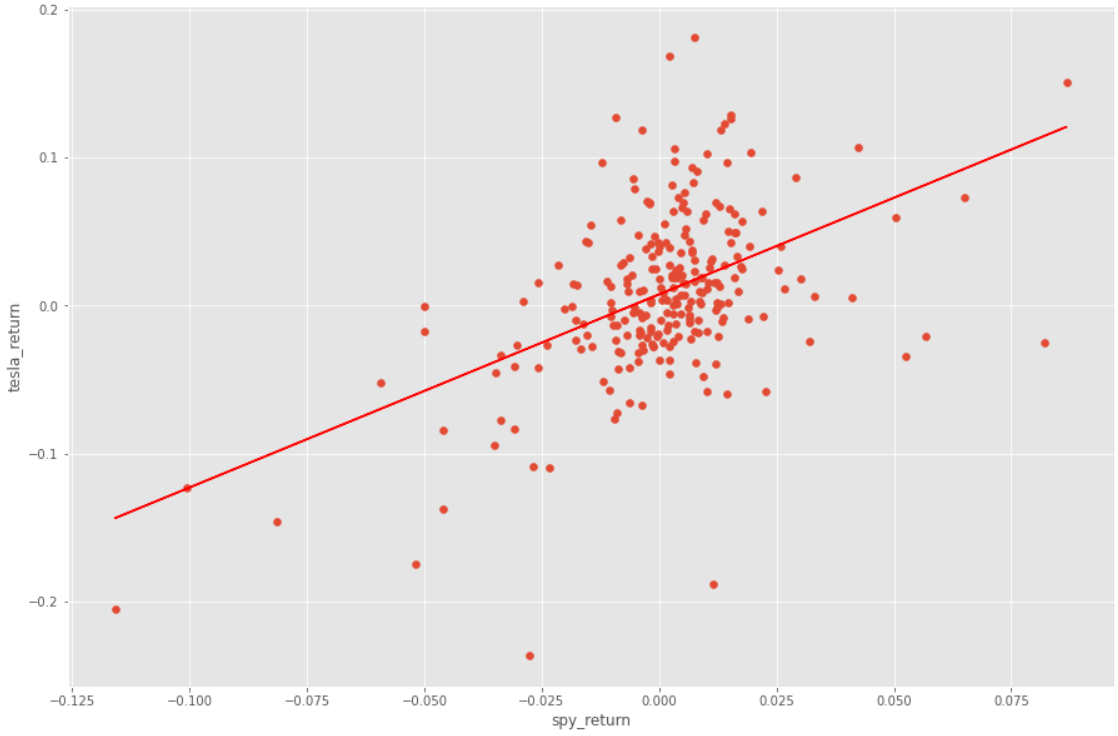
Where α is called the intercept and β is called the slope. If the scatter points can be represented by $ (x_1,y_1),(x_2, y_2),(x_3,y_3)…(x_n,y_n)$. Then the intercept and slope are as follows:

$$\beta = \frac{\sum_{i=1}^{n}(x-\bar{x})(y-\bar{y})}{\sum_{i=1}^{n}(x-\bar{x})^2}$$

$$\alpha = \bar{y} - \hat{\beta}\bar{x}$$

Where $\bar{x}$ is the mean of X and $\bar{y}$ is the mean of Y. In python, it isnt needed to hard code the equation above. We have a package that handles that for us.

[54]:
```
plt.figure(figsize = (15,10))
plt.scatter(df.spy,df.tesla)
plt.xlabel('spy_return')
plt.ylabel('tesla_return')
plt.plot(df.spy,model.predict(),color = 'red')
plt.show()
```



The red line is the fitted linear regression straight line. As we can see there are lot of statistical results in the summary table. Now let's talk about some important statistical parameters. We built a simple linear regression model above by using the ols() function in statsmodels. The 'model' instance has lots of properties. The most commonly used one is parameters, or slope and intercept. We can access to them by:

[55]:
```
print ('parameters: ',model.params)
print ('residual: ', model.resid.tail())
print ('fitted values: ',model.predict())
```

```
parameters:  Intercept    0.007622
spy          1.305339
dtype: float64
residual:  Date
2020-12-24    0.011459
2020-12-28   -0.015891
2020-12-29   -0.001670
```

```
2020-12-30    0.032837
2020-12-31    0.001315
dtype: float64
fitted values:  [-0.00229979  0.01259261  0.00394666  0.01456053  0.01644332
0.00386036
  0.01656869  0.00563049  0.01056871  0.01843555  0.01167875  0.00506354
  0.00777978  0.00911835 -0.00403825 -0.01347116  0.02122986  0.00654347
  0.01185158 -0.01629789  0.01728307  0.02736709  0.02260984  0.01200706
  0.00064654  0.01733073  0.00988241  0.01600518  0.00622866  0.00971178
  0.00425396  0.01384841  0.00224846 -0.00589027 -0.0364041  -0.03254409
  0.00281199 -0.05236007  0.00212617  0.06296199 -0.0302984   0.06136769
 -0.0365071  -0.01413689 -0.09851821  0.07347725 -0.05761441 -0.12365425
  0.11469658 -0.14364901  0.07626328 -0.06020291  0.01039303 -0.05753169
 -0.0261871   0.12083546  0.02701848  0.08169842 -0.03184901  0.04934015
 -0.01198088 -0.05248743  0.03740108 -0.01138315  0.09247795  0.00895224
  0.0507209   0.02733634 -0.00435052  0.04556355 -0.02041272  0.01390435
  0.04241851 -0.01558027 -0.03262634  0.03627672  0.00752852  0.02569125
  0.02630871  0.00160581  0.04135432 -0.00458835 -0.02740029  0.01121762
  0.01962723 -0.0012565   0.02327916  0.0290438   0.00788993 -0.01865808
 -0.01567047  0.02315081  0.013609    0.04678885 -0.00585454  0.02961102
 -0.00142108  0.01009873  0.02360638  0.02690122  0.00521165  0.0134257
  0.01288735  0.01838605  0.02487935  0.00418889  0.04065584  0.02330581
 -0.00214783  0.00031797 -0.06988546  0.0231616   0.0197518   0.03250817
  0.00218869  0.00812464 -0.00559076  0.01596939  0.01361772 -0.02610739
  0.02154161 -0.0237557   0.02666783  0.02423676  0.01673393  0.01479088
  0.02761905 -0.00591075  0.01756825  0.00016973  0.02088279 -0.00372994
  0.02442939  0.01955985  0.00331599  0.01140095  0.01812895  0.01039636
  0.01503115 -0.00804641 -0.00081192  0.01710671 -0.00068318  0.02357823
  0.00295648  0.01789666  0.01666568  0.01265446  0.01570448  0.0163187
  0.00855894  0.01151788 -0.0031977   0.02569594  0.00526026  0.00766095
  0.01176213  0.01043913  0.00217574  0.0116801   0.01224451  0.02078278
  0.01218196  0.0206438   0.01047323  0.01602673  0.00288493  0.01985907
  0.02636733 -0.03809135 -0.00307929 -0.02853982  0.03314793 -0.01524237
  0.00828653  0.02470288  0.01420059  0.00243145 -0.00390943 -0.01270777
 -0.00698715  0.02084898 -0.02300644  0.01109686  0.02855674  0.02912489
  0.00049382  0.01748186  0.01597564 -0.00483054  0.03056699 -0.01106766
  0.03014827  0.01913977  0.01922728  0.0284548  -0.00092451 -0.00060571
  0.00600794  0.00683311 -0.01237587  0.01284053  0.0051489   0.01476275
  0.01204651 -0.01672606  0.00311436 -0.0377732   0.02082199 -0.00605727
  0.02217157  0.03046802  0.03647423  0.0328348   0.00732401  0.0239224
  0.00570637  0.01728309 -0.00510263  0.02556996  0.02381525  0.0005827
 -0.00818265  0.01310632 -0.00134731  0.01542357  0.02848834  0.00560808
  0.01125248  0.00183042  0.02182164  0.01036539  0.0072662   0.0188229
  0.00493704  0.01143621 -0.00413808  0.00719511  0.00609063  0.00176482
  0.02515135  0.00966909  0.01490123 -0.00315272  0.00294654  0.00542027
  0.00879466  0.01269059  0.01878819  0.00512947  0.00948328  0.01423755]
```

[56]:
```
model = sm.ols(formula = 'tesla~spy',data = df).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                  tesla   R-squared:                       0.242
Model:                            OLS   Adj. R-squared:                  0.239
Method:                 Least Squares   F-statistic:                     79.99
Date:                Wed, 05 May 2021   Prob (F-statistic):           8.60e-17
Time:                        21:02:58   Log-Likelihood:                402.08
No. Observations:                 252   AIC:                            -800.2
Df Residuals:                     250   BIC:                            -793.1
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept      0.0076      0.003      2.455      0.015       0.002       0.014
spy            1.3053      0.146      8.944      0.000       1.018       1.593
==============================================================================
Omnibus:                       23.072   Durbin-Watson:                   1.989
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               77.683
Skew:                          -0.245   Prob(JB):                     1.35e-17
Kurtosis:                       5.676   Cond. No.                         47.0
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
```

From the OLS Regression Results, we can see *std err*. This is the standard error of the intercept and slope. The null hypothesis is $H_0 : \beta = 0$ and the alternative is $H_1 : \beta \neq 0$. This can be calculated by: $t = \frac{\beta - 0}{SE}$. Where SE is:

$$SE = \sqrt{\frac{\frac{1}{n-2}\sum_{i=1}^{n} \hat{\epsilon}^2}{\sum_{i=1}^{n} (x_i - \bar{x})^2}}$$

The distribution that is used here is different from a normal distribution. The one listed above is *Student's t-distribution*. The column t in the table above is the test score, and $P>|t|$ is the p-value. When observing the p-value, we can see that the significance level of spy which is also the slope, is very high due to the p-score being so close to 0. In other words, we have a near 100% confidence to claim that the slope is not 0. With that also, there exists a linear relation between X and Y. The following 2 columns are the lower band and upper band of the parameters at 95% confidence interval. At 95% confidence level, we can claim that the true value of the parameter is within this range.

1.2.3 Model Significance

Sum of Squared Errors or SSE, is used to measure the difference of the fitted valus and the actual values as follows:

$$SSE = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \sum_{i=1}^{n} \hat{\epsilon_i}^2$$

If the SSE perfectly fitted the sample, then the SSE would be 0. The reason that we use the squared error is that the positive and negative errors would offset each other if we summed them up. Theres another measurement of the dispersion of a sample is **total sum of squares** or SS and it is as follows:

$$SS = \sum_{i=1}^{n} (y_i - \bar{y_i})^2$$

We can see that the SS is divided by the number of sample size n. From SSE and SS we can calculate the **Coefficient of Determination** or r-square. R-square is the proportion of variation that is explained by the linear relationship between X and Y. It is calculated with:

$$r^2 = 1 = \frac{SSE}{SS} = 1 - \frac{\sum_{i=1}^{n} (y_i - \hat{y}_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y_i})^2}$$

We can assume that the model perfectly fits the sample, which means all of the sample points lie on the straight line. Then the SSE would be 0 and the r-square would be 1. That would equate to perfect fitness. The higher your r-squared value is the more variation can be explained by the linear relation.

1.2.4 Using Machine Learning

It is extremely hard to try and predict the direction of the stock market. I will be doing some very basic Machine Learning Practices to read in some stock data from a certain time frame and then try to predict it's value. I will be using a **Support Vector Regression or SVR**, it is a type of support vector machine and is a type of supervised learning algorithm that analyzes data for regression analysis. Our small program will predict the stock prices for 30 days in the future based off of the currect adjusted close price. We will be looking and predicting the Amazon stock. Machine Learning is classified under many categories. We will be working with the supervised learning which is where machine experiences the examples along with the labels or targets for each example. The labels in the data help the algorithm to correlate the features. There is also Unsupervised Learning and Reinforcement Learning. Regression is a technique used to predict the value of a response (dependent) variables, from one or more predictor (independent) variables. Most commonly used regressions techniques are: Linear Regression and Logistic Regression. We will focus on the Linear Regression. With machine learning we have to seperate our data into training and testing data. Separating data into training and testing sets is an important part of evaluating data mining models. Typically, when you separate a data set into a training set and testing set, most of the data is used for training, and a smaller portion of the data is used for testing. Analysis Services randomly samples the data to help ensure that the testing and training

sets are similar. By using similar data for training and testing, you can minimize the effects of data discrepancies and better understand the characteristics of the model. After a model has been processed by using the training set, you test the model by making predictions against the test set. Because the data in the testing set already contains known values for the attribute that you want to predict, it is easy to determine whether the model's guesses are correct.

[57]:
```
### Import all our dependencies
import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.svm import SVR
from sklearn.model_selection import train_test_split

### Import our Amazon Stock
amzn_table = web.DataReader("AMZN", 'yahoo', start, end)
amzn_table.reset_index(inplace=True)
amzn_table.set_index("Date", inplace=True)

### We only want the Adj Close of the stock
df = amzn_table[['Adj Close']]

### Creating a variable called forecast_out to store the number of days into␣
 ↪the future that we want to predict which is 30.
forecast_out = 30 #this variable can be changed depending how many days you␣
 ↪want to predict out

### Creating a column that is shifted n days up. This will hold the predicted␣
 ↪values.
df['Prediction'] = df[['Adj Close']].shift(-forecast_out)

### Create an independent dataset X. This dataset will be used to train the␣
 ↪machine learning model.
X = np.array(df.drop(['Prediction'],1))

### Remove the last n days
X = X[:-forecast_out]

### Create the dependent dataset y.
y = np.array(df['Prediction'])

### Get all of the y values except the last n rows
y = y[:-forecast_out]
```

```
<ipython-input-57-98880d975fed>:19: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
df['Prediction'] = df[['Adj Close']].shift(-forecast_out)
```

With all of this information coded we can start to create or training models. We will be looking at 50% training and 50% testing. We will be comparing these values against the same data but under a linear regression model to find which one has the best confidence value!

[58]:
```
### Splitting the data into 50% training and 50% testing
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.5)

### Creating and training the SVR
svr_rbf = SVR(kernel = 'rbf', C=1e3, gamma = 0.1)
svr_rbf.fit(x_train, y_train)

### Testing Model: Score returns the coefficient of determination R^2 of the␣
 ↪prediction.
### The best possible score is 1.0
svm_confidence = svr_rbf.score(x_test, y_test)
print("svm confidence: ", svm_confidence)
```

```
svm confidence:  0.8481554859541901
```

[59]:
```
### Creating and training the Linear Regression Model that we will test against.
lr = LinearRegression()

### Training the model
lr.fit(x_train, y_train)

### Testing Model: Score returns the coefficient of determination R^2 of the␣
 ↪prediction.
### The best possible score is 1.0
lr_confidence = lr.score(x_test, y_test)
print("lr confidence: ", lr_confidence)
```

```
lr confidence:  0.9845872839828046
```

In this case here of using 50% of data for training and 50% for testing, we see that the linear regression model has a higher confidence level if we go off of the R^2 score. Lets start to explore some other possible options for our training and testing percentages. We will be test 60/40, 70/30, 80/20, 90/10, and 95/5 respectively. We want to find the one that has the closest R^2 value to 1.0. With that data set we will use it to make our predictions!

[60]:
```
### 60% traiing 40% testing
# SVR
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.4)
svr_rbf = SVR(kernel = 'rbf', C=1e3, gamma = 0.1)
svr_rbf.fit(x_train, y_train)
svm_confidence = svr_rbf.score(x_test, y_test)
print("svm confidence: ", svm_confidence)
# LR
```

```
lr = LinearRegression()
lr.fit(x_train, y_train)
lr_confidence = lr.score(x_test, y_test)
print("lr confidence: ", lr_confidence)
###Linear Regression wins this round
```

```
svm confidence:  0.870329089705198
lr confidence:  0.9817222491485447
```

[61]:
```
### 70% traiing 30% testing
# SVR
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.3)
svr_rbf = SVR(kernel = 'rbf', C=1e3, gamma = 0.1)
svr_rbf.fit(x_train, y_train)
svm_confidence = svr_rbf.score(x_test, y_test)
print("svm confidence: ", svm_confidence)
# LR
lr = LinearRegression()
lr.fit(x_train, y_train)
lr_confidence = lr.score(x_test, y_test)
print("lr confidence: ", lr_confidence)
###Linear Regression wins this round
```

```
svm confidence:  0.8988899481375056
lr confidence:  0.9863371058776976
```

[62]:
```
### 80% traiing 20% testing
# SVR
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
svr_rbf = SVR(kernel = 'rbf', C=1e3, gamma = 0.1)
svr_rbf.fit(x_train, y_train)
svm_confidence = svr_rbf.score(x_test, y_test)
print("svm confidence: ", svm_confidence)
# LR
lr = LinearRegression()
lr.fit(x_train, y_train)
lr_confidence = lr.score(x_test, y_test)
print("lr confidence: ", lr_confidence)
###Linear Regression Wins this round!
```

```
svm confidence:  0.8850106797787987
lr confidence:  0.9862758645305443
```

[63]:
```
### 90% traiing 10% testing
# SVR
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
svr_rbf = SVR(kernel = 'rbf', C=1e3, gamma = 0.1)
svr_rbf.fit(x_train, y_train)
svm_confidence = svr_rbf.score(x_test, y_test)
```

```
print("svm confidence: ", svm_confidence)
# LR
lr = LinearRegression()
lr.fit(x_train, y_train)
lr_confidence = lr.score(x_test, y_test)
print("lr confidence: ", lr_confidence)
###Linear Regression Wins this round!
```

```
svm confidence:  0.9243275677323396
lr confidence:  0.9869883001475364
```

[64]:
```
### 95% traiing 5% testing
# SVR
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.05)
svr_rbf = SVR(kernel = 'rbf', C=1e3, gamma = 0.1)
svr_rbf.fit(x_train, y_train)
svm_confidence = svr_rbf.score(x_test, y_test)
print("svm confidence: ", svm_confidence)
# LR
lr = LinearRegression()
lr.fit(x_train, y_train)
lr_confidence = lr.score(x_test, y_test)
print("lr confidence: ", lr_confidence)
###Linear Regression Wins this round!
```

```
svm confidence:  0.8936633008357618
lr confidence:  0.9862391185431866
```

With this data we can see that our closest R^2 values came with the 90% training and 10% testing split. We will use these values for our predictions. For this we will take the last 30 rows of the data from the dataframe of the Adj Close price and store it in a variable. We will then transform it into a numpy array. Then we will arrive at our moment of truth, printing out the predicted future prices! But first we have to get our best train/test split setup which was the 90/10.

[66]:
```
### Setting our 90/10 split back up for the predictions
# SVR
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.1)
svr_rbf = SVR(kernel = 'rbf', C=1e3, gamma = 0.1)
svr_rbf.fit(x_train, y_train)
svm_confidence = svr_rbf.score(x_test, y_test)
# LR
lr = LinearRegression()
lr.fit(x_train, y_train)
lr_confidence = lr.score(x_test, y_test)

### Set x_forecast equal to the last n rows of the original dataset from Adj␣
 ↪Close
```

```
x_forecast = np.array(df.drop(['Prediction'],1))[-forecast_out:]

### Print the SVR model predictions for the next n days.

svm_prediction = svr_rbf.predict(x_forecast)
print(svm_prediction)

### Pring the Linear Regression model predictions for the next n days.

lr_prediction = lr.predict(x_forecast)
print(lr_prediction)
```

```
[2208.86191957 2210.88200822 3074.69103091 2986.77450887 3156.71046773
 3346.9828885  3095.54023359 2359.85397537 2934.30250548 2427.29799058
 2355.75936301 1541.27990836 2501.72155473 2261.49792684 2315.80486902
 2458.80115274 2111.17952924 1540.96761305 2012.24868338 2289.88971517
 3107.07302277 2189.46971061 1543.86633141 1538.26039793 1541.65618792
 1538.2603979  1538.26039791 1553.07714289 3159.44423728 3050.44578822]
[3177.08951581 3135.22275919 3141.14218229 3165.45579814 3144.48660374
 3184.28103631 3252.93372094 3320.36565291 3317.38039422 3374.38909223
 3394.81478371 3469.60228891 3476.97840003 3498.12216657 3429.38733034
 3476.67083247 3497.54760633 3469.40742933 3431.12102954 3459.1587675
 3404.80697227 3437.47124758 3507.35520427 3516.00341279 3558.13691163
 3571.27866207 3567.28779773 3484.26234107 3407.71032983 3365.31008877]
```

Now that we have our predictions, dont actually use these for the real stock market! There are so many other factors that have to get applied that we just cant fully apply to a machine learning model. Just take this information witha grain of salt, it is not financial advice. We could definetly take these prices and compare them for the next 30 days to see which one was the closest and from there make a final decision on which of the two models is a better fit!

[ ]: