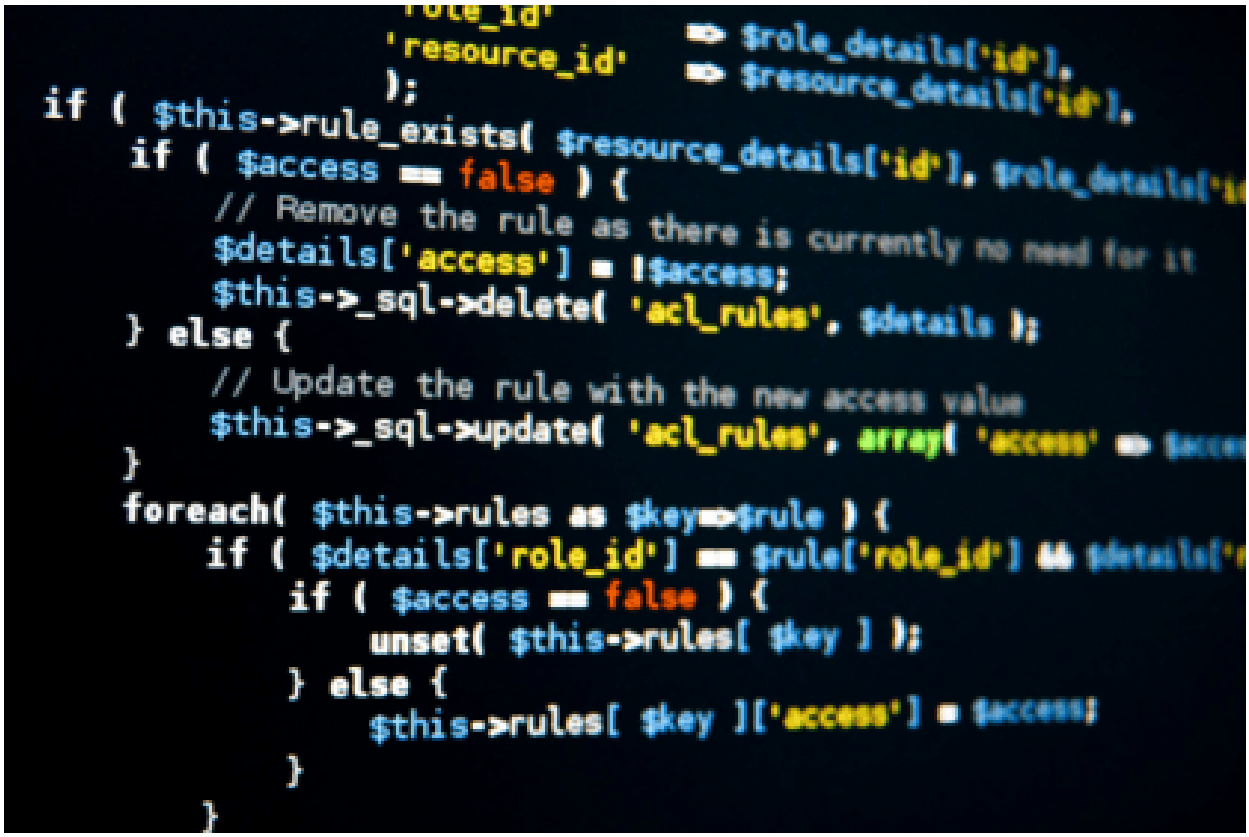


Using Machine Learning algorithms to model Apple stock prices

—

RQ: Which of the Machine Learning regression models - Linear or Polynomial - is more accurate in modelling and predicting the stock price of Apple Inc.?

Word count: 3943

Exam Session: May 2019

Table of Contents:

1 Introduction

1.1 The importance and application of my essay

Apple is the leading company in the technological frontier whose shares are openly traded on the stock market. It is one of the oldest IT companies and it's stock prices are volatile and change drastically over time. Hence, I thought that this would be the best choice for my evaluation. After observing apple stocks over time, I noticed that there was some underlying pattern which was not completely randomized. Thus, I decided to model this pattern and predict future prices using machine learning algorithms. Such applications of machine learning algorithms are already used for weather forecasting by national agencies. I wanted to see if the same concept could be useful for stock prices. Obtaining an accurate model, crucial decisions, like purchasing more stock or favorable time for selling stock, can be made using it. Although this is already implemented by investment bankers and stock agencies, the technology they use is advanced and not for individuals. Therefore, I tried using the basic framework of these algorithms to make my own prediction models for the same.

1.2 Artificial Intelligence and Machine Learning

To define AI: Artificial intelligence is an area of computer science that emphasizes the creation of intelligent machines that work and react like humans[1]. In other words, it is

[1] "What Is Artificial Intelligence (AI)? - Definition From Techopedia"

the intelligence of a machine similar to human intelligence to perform ‘cognitive’ tasks such as learning, problem solving and decision making by simulating the human brain into machines.

People often confuse Machine Learning and AI to be the same and use the terms interchangeably, however they are quite different. Machine Learning, as defined by Stanford University, is “the science of getting computers to act without explicitly programming them.” This means that it is a program that can learn and improve by looking for patterns in observations or data and hence come up with a model that will make better decisions in the future and also improve with experience without human intervention.

Machine learning is helpful in reducing the time spent behind programming. For instance, it would seem impossible to for a programmer to come up with a spell checking program that would work perfectly in a day’s time; whereas, using a machine learning tool, the programmer could create an accurate program in a fraction of that time. ML also allows you to complete seemingly ‘unprogrammable’ tasks. For example, recognising our friend’s face or voice is done subconsciously and if asked to program the same, we would be clueless; but, a machine learning algorithm can perform this task effectively and such technology is used in face recognition on smartphones.

1.3 Types of Machine Learning

Machine learning is broadly classified into two types:

1. Supervised
2. Unsupervised

Supervised machine learning is when an algorithm can learn to combine input to come up with accurate predictions on unseen data. It makes a prediction model based on labels and features. Labels here mean the thing being predicted and features are the input variables. The learning comes in when the algorithm compares the prediction with the actual output and thus improving and reducing loss which is the penalty for an inaccurate prediction.

The two major types of Supervised algorithms are:

1. Regression
2. Classification

1.4 Regression Algorithms

Regression algorithms fall under the supervised machine learning category and predict outcomes based on the input features using past examples fed to the system. Theses types of algorithms improve over time and become more accurate when more data is fed to it. It does this by making a model that establishes the relationship between two or

more variables. There are many different types of Regression algorithms but in this essay will be comparing the 2 most commonly used Regression algorithms:

1. Linear
2. Polynomial

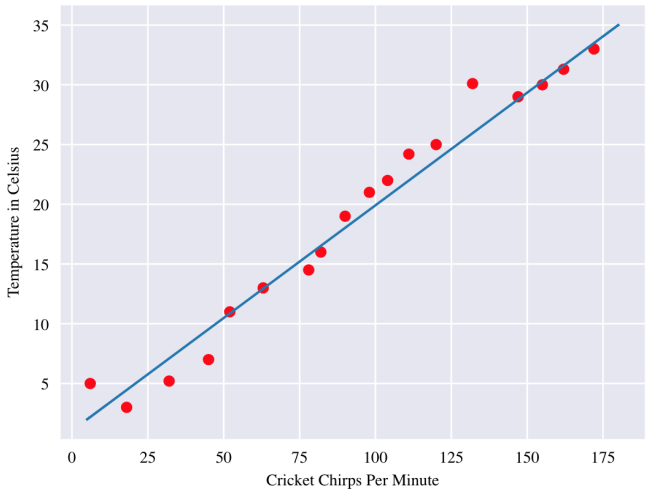
1.4.1 Linear Regression

Linear regression is probably the most famous and the simplest algorithm used in machine learning. This concept which is originally from statistic is one of the first used regression models. As the name suggests this algorithm is used to analyze and study a linear relationship between the input(x) variable and the output(y) variable to predict future outcomes on new data.

An example of linear regression using real data is as follows:

Figure 1.4.1 shows a linear regression line which in statistics is known as line of best fit. This is the prediction line and clearly shows the linear relationship between chirps per minute and rise in temperature.

Figure:1.4.1[2]



[2]"Descending Into ML: Linear Regression | Machine Learning Crash Course | Google Developers"

Here, we can represent this linear relationship using the equation:

$$Y' = w_1x_1 + b$$

Where,

Y' is the predicted label or the output

b is the bias which is the y-intercept

w_1 is the slope of the equation which is called the 'weight'

x_1 is the input feature or independent variable

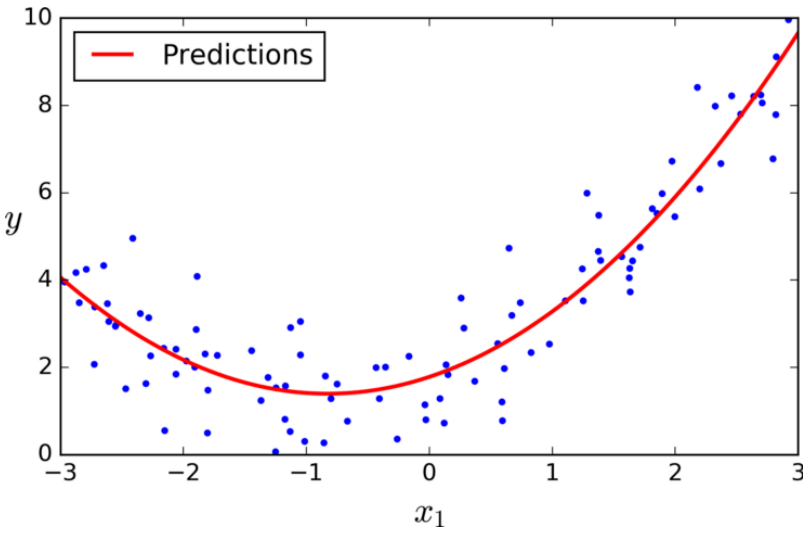
Here we have used w_1 & x_1 because a regression can have multiple features.

The model is trained using labelled examples and finds a way to minimize loss which is a number that indicates how different the prediction and the actual outcome was. A model can be said to be perfect if it has no or minimum loss and that is the goal of Machine learning. It does this by using a loss function know as **Ordinary Least Squared**. This is the sum of the squares of the distance between the regression line and each output value. Thus, it is very sensitive to outliers.

1.4.2 Polynomial Regression

Polynomial Regression is used when Linear Regression cannot accurately represent relationships between the features and the output. In this regression technique, the best fit line is not a straight line but a curve that fits into the data points. For a polynomial regression, the power of some independent variables is more than 1.

Figure:1.4.2[3]



[3]"The Most Insightful Stories About Polynomial Regression – Medium"

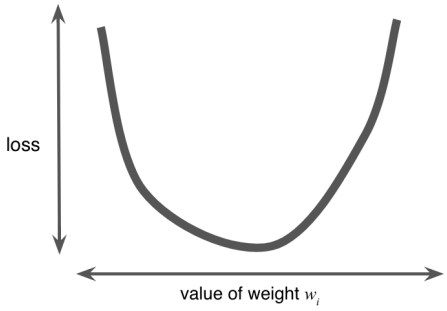
2 Algorithms

2.1 A detailed study of Linear Regression

The main aim of any linear algorithm is to reduce loss and hence provide more accurate results. In Machine Learning, iterative methods are common mainly because they scale quite well to massive datasets. Single or multiple features are taken by the model as input and one prediction(y') is returned as output. To make it easier to understand, picture a model taking one feature and returning one prediction:

$$y'=b+w_1x_1$$

Being a linear regression problem the starting values are not important. We can pick a random value and find the loss. Then using a loss function, such as least squared loss, closest simulation of the best possible model can be reached. The method involves formulating new values and then the ML system re-assesses all the features against labels, yielding new values for the function. This continues iterating till the algorithm finds the ideal parameters with the minimum potential loss. Generally, one iterates until overall loss stops altering or at the least changes extraordinarily slowly thus stating the convergence of a model.



2.1.1 Gradient Descent

For linear regression problems, the graph of loss versus weight will always have a convex shape. Simply put, the curve of the graph will always have a bowl-like shape, similar to this[4]: Figure:2.1.1.a

Convex issues have only 1 minimum point which is the only place where the gradient is precisely zero.

Calculating the loss function for each possible value of w_1 over the whole set of data would be an incompetent way of locating the point of convergence. Hence, there is a superior mechanism known as **gradient descent**.

In gradient descent, the primary step is to select an initial value for w_1. The initial point is insignificant; thus, many algorithms set w_1 to 0 or to a random value.

The gradient of the loss curve at the initial position is then computed by the gradient descent algorithm. The gradient of a function is equivalent to its slope of the graph , and indicates which direction is closer to or farther away from the minimum point.

The gradient constantly points towards steepest increment in the loss function. In order to minimize loss at the earliest, the gradient descent algorithm moves further towards the negative gradient.

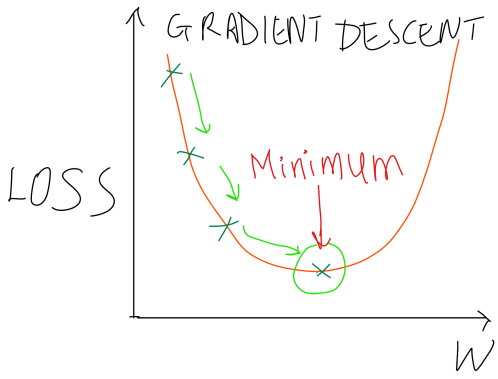
[4] "Reducing Loss: Gradient Descent | Machine Learning Crash Course | Google Developers"

Gradient descent algorithms reach the minimum by taking one step at a time moving from one point to another.The size of that step is determined by the algorithm's **learning rate** or **step size**.

In machine learning algorithms, Hyperparameters are the nodes that programmers modify. Most programmers devote a substantial amount of time calibrating the learning rate. If one selects a learning rate that is very minuscule, learning will be drawn-out and seemingly endless. Au contraire, if one chooses a learning rate that is enormous, the following point will "endlessly spring chaotically across the foot of the well like a quantum mechanics investigation that tragically misfired and might entirely overshoot the minimum."[5]

There exists a perfect learning rate that is unique to every regression problem. This is known as the Goldilocks[6] learning rate. The Goldilocks value is determined by the gradient of the loss function and the step size is directly proportional to it. Figure 2.1.1.b shows an example of this:

Figure:2.1.1.b



[5] "Reducing Loss: Gradient Descent | Machine Learning Crash Course | Google Developers"
[6] The **Goldilocks principle** is named by analogy to the children's story 'The Three Bears'. This concept is widely known as "just the right amount".

In gradient descent, a batch is the complete set of examples you use to find the gradient in a single repetition. Uptil now, we were making an assumption that the batch was the whole dataset. This becomes problematic when large firms come into the equation. When working at Apple like scales, data sets usually include billions or trillions of instances. Moreover, Apple data sets often contain an immense number of features. As a result, a batch can be humongous. In an enormous batch, even a single iteration may take forever to compute.

Another problem with large datasets containing unsystematically sampled examples is data redundancy[7]. As a matter of fact, redundancy becomes further probable as the batch size increases. Some redundancy can be helpful in evening out randomly fluctuating gradients, nonetheless very large batches tend not to bear much more predictive value than big batches.

To solve the problems mentioned above a special method called Stochastic Gradient Descent is used. However, this is not the focus of the essay.

[7] Data redundancy is a condition created within a database or data storage technology in which the same piece of data is held in two separate places.

2.2 A detailed study of Polynomial Regression

Polynomial regression is an extension of the linear regression model with additional predictors, obtained by incrementing the degree of each of the original predictors. For example, a quadratic regression uses two variables x_1, and x_2 as predictors. This method provides an easy way to produce a nonlinear fit for the data.

The standard approach for the extension of a linear regression to a nonlinear relation between the dependent and independent variables, has been to provide a polynomial function in the place of a linear one.

$$Y = \beta_0 + \beta_1 X_i + \beta_2 X_i^2 + \beta_3 X_i^3 + \ldots\ldots + \beta_p X_i^P$$

As we increment the degree of the polynomial, the resultant graph contains high fluctuations which will lead to curves that are over-flexible. These shapes lead to overfitting.

3 Code

3.1 Basic Linear Regression Model: Program A[8]

Firstly, using the Dataset and the source code obtained from Github, I made a basic linear regression model to show the linear shape of the graph and calculate mean squared error.

I used the Spyder IDE in the Anaconda Navigator to run the python code.

On evaluating the output, we can see the shape of the graph obtained is almost linear and this graph shows that the predicted values are very close to the real values and that is what we aim for in a Linear Regression graph. This is great for understanding the nature and shape of a basic linear regression graph.

Code Exploration:

Figure:3.1.a

```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 import sklearn
4
5 import seaborn as sns
6 sns.set_style("whitegrid")
7 sns.set_context("poster")
8
```

Importations of various libraries. Included libraries are *matplotlib*, *pandas*, *sklearn, and seaborn*.

[8] Complete Code and Output in Appendix 5.1.1.

Figure:3.1.b

```
from sklearn.datasets import load_boston
boston = load_boston()
bos = pd.DataFrame(boston.data)
bos.columns = boston.feature_names
bos['PRICE'] = boston.target
print(bos.head())

X = bos.drop('PRICE', axis = 1)
Y = bos['PRICE']

```

Then, I imported the pre-existing dataset *Boston* for my Linear Regression model. I convert the data and represent it in a table format to make it easier observation and understanding. I also add an extra column named ‘PRICE’ used in the plotting of the graph.

Then, I split the data into ‘X’ & ‘Y’. ‘Y’ are the target values and ‘X’ are the predictor values. Thus, to simplify I set Y as Price and X as all the other features.

Figure:3.1.c

```

X_train, X_test, Y_train, Y_test = (sklearn.model_selection.
                                    train_test_split(X, Y, test_size = 0.33, random_state = 5))
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)

```

I split the data into training and test set. In general, the greater the training data, the more accurate the model will be. Hence, I split the boston dataset in 2/3rds training data and 1/3rd test data.

Figure:3.1.d:

```
25
26 from sklearn.linear_model import LinearRegression
27
28 lm = LinearRegression()
29 lm.fit(X_train, Y_train)
30
31 Y_pred = lm.predict(X_test)
32
33 plt.scatter(Y_test, Y_pred)
34 plt.xlabel("Prices: $Y_i$")
35 plt.ylabel("Predicted prices: $\hat{Y}_i$")
36 plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
37
```

This is the main code block. Here, I write the code for the Linear Regression model. I import it from the sklearn library and train and test it. I use a scatter plot to show the result of my model.

Figure:3.1.e

```
37
38 mse = sklearn.metrics.mean_squared_error(Y_test, Y_pred)
39 print(mse)
40
```

This is the final and smallest but most crucial block of code, because a Linear Regression model is only as good as its predictions. Meaning, a better Linear Regression model that will make accurate predictions is only obtained by reducing the Mean Squared Error(MSE). As explained in the Body of my essay, MSE is the Square of the difference between the actual value and the prediction made. Thus, the lower the MSE the better the Model. This is essential when trying to increase efficiency of models.

Output Evaluation:

Figure:3.1.f

```
In [15]: runfile('/Users/twisha_shah17/.spyder-py3/temp.py', wdir='/
Users/twisha_shah17/.spyder-py3')
      CRIM    ZN  INDUS  CHAS  ...    PTRATIO       B  LSTAT  PRICE
0  0.00632  18.0   2.31   0.0  ...       15.3  396.90   4.98   24.0
1  0.02731   0.0   7.07   0.0  ...       17.8  396.90   9.14   21.6
2  0.02729   0.0   7.07   0.0  ...       17.8  392.83   4.03   34.7
3  0.03237   0.0   2.18   0.0  ...       18.7  394.63   2.94   33.4
4  0.06905   0.0   2.18   0.0  ...       18.7  396.90   5.33   36.2
```

This table is the representation of data with the target column ‘PRICE’. By observing the table we can see that there is some connection between the five examples however we cannot see a clear correlation between them. Figure:3.1.g

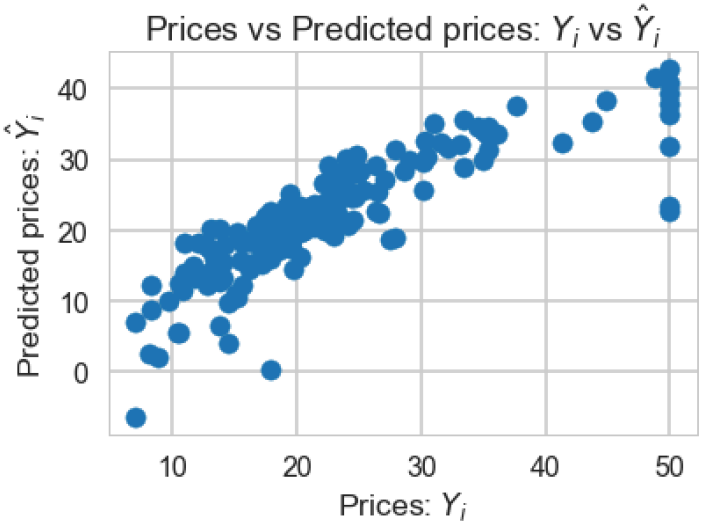
```
[5 rows x 14 columns]
(339, 13)
(167, 13)
(339,)
(167,)
```

These are the shapes of the training and test datasets. Figure:3.1.h

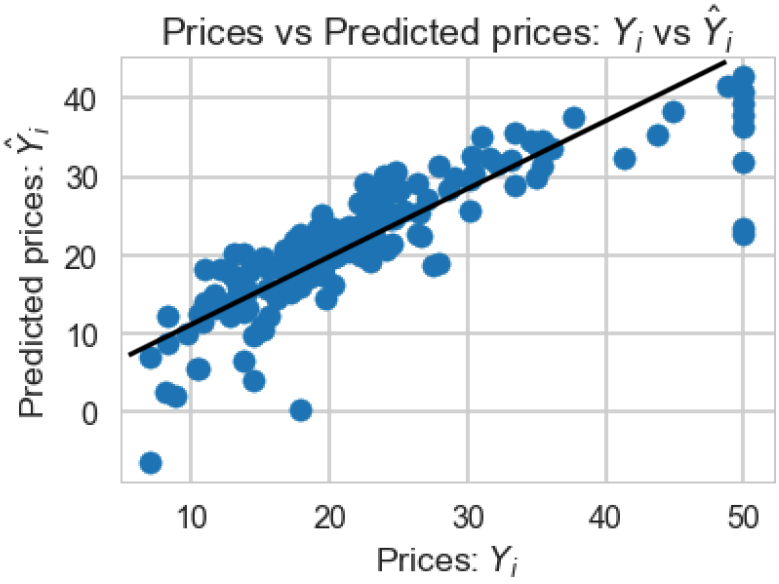
```
28.54136727561835
```

This is the MSE of my Linear Regression model which is a very crucial part of the output. This number is quite large but doesn’t make the model redundant. A better model can be obtained by further lowering the MSE.

Figure:3.1.i

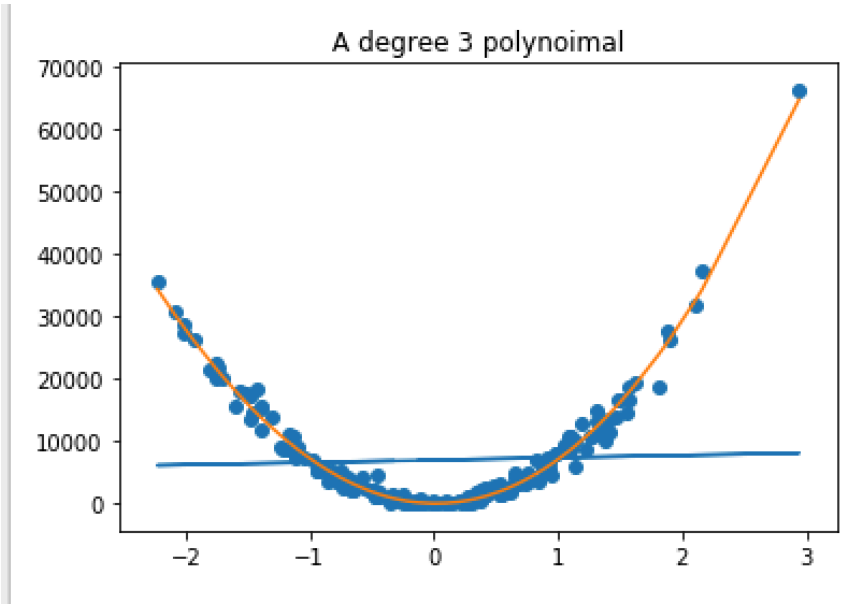


This is the pictorial representation of my Linear Regression model, which is of most concern in this essay. As seen, there is a clear linear correlation between the features of the graph. Hence, a best fit line can easily be drawn for this graph as below. Figure:3.1.j



3.2 Basic Polynomial Regression Model: Program B[9]

This is an example of a simple polynomial regression graph: Figure:3.2



In the above image, we can see how much better the polynomial regression model is in depicting the relationship and the trend of the scatterplot compared to the linear regression model.

I have mainly used *sklearn* and *matplotlib.pyplot* for the arrangement of data points and the creation of the regression model. This is a very basic model and is a basic framework for the more complex polynomial regression model I make later on.

[9] Complete code and Output in Appendix 5.1.2

3.3 Modeling Apple Stocks

Before beginning the program for creating the ML model for Apple's Stock Prices, a dataset is required.

The dataset I used for this program was obtained from the official NASDAQ[10] website. The dataset contains all stock prices of Apple Inc. for a duration of 3 months starting 22 May to 22 August. The database was in a '.csv' format.[11]

3.3.1 Modeling Apple Stocks : Program C[12]

First I made a program that represented the data extracted from the dataset in the form of a table and a graph and using a linear prediction model and saw a strong linear correlation between stock prices and time.

The code for the same is as follows: Figure:3.3.1.a

```
1 import pandas as pd
2 import sklearn
3 from sklearn.model_selection import train_test_split
4 mydata = pd.read_csv('HistoricalQuotes.csv')
5 dataset = pd.DataFrame(mydata)
6 print(dataset)
7
8 x = dataset.drop('close', axis = 1)
9 y = dataset['close']
10
```

In this code block i've declared the libraries and imported my dataset from *HistoricalQuote.csv* which is the datasheet using the pre-defined function *read_csv*. Then I've used the print function to display the data in a list like manner. Later, I've

[10] National Association of Securities Dealers Automated Quotations, Official Stock Exchange of the USA
[11] Complete Datasheet in Appendix 5.1.3
[12] Complete code and Output in Appendix 5.1.4

selected the feature *close* which is the closing stock price as the target feature(y-variable) and all the other features are assigned to the x-variable.

Figure:3.3.1.b

```
10
11 x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state = 123)
12
13 from sklearn.linear_model import LinearRegression
14 lm = LinearRegression()
15 lm.fit(x_train,y_train)
16
17 y_pred = lm.predict(x_test)
18
```

Here, I have split the data into training and test data by a 4:1 ratio respectively. I wanted the model to have a better predictions with minimum loss thus I did not go for 3:1 ratio. I have done this using the *train_test_split* methods imported from *sklearn* library. Then I have used the *LinearRegression()* function to make my model, fit my data in the model and predict the values.

Figure:3.3.1.c

```
18
19 import matplotlib.pyplot as plt
20
21 plt.scatter(y_pred, y_test)
22 plt.xlabel("Actual Stocks")
23 plt.ylabel("Predicted Price")
24 plt.title("Stock of Apple Inc. over the past 3 months")
25
```

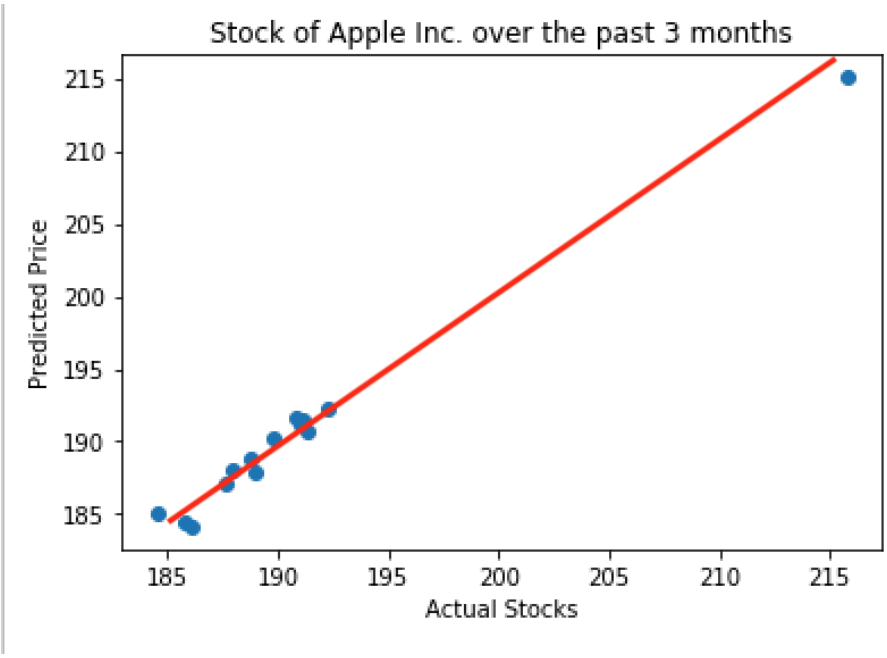
I have formed a scatter plot graph and marked all the data points on it. I have allocated certain labels to the axes and given a title to the scatter plot for knowing its purpose.

Figure:3.3.1.d

```
25
26 from sklearn.linear_model import LinearRegression
27 model = LinearRegression()
28 model.fit(x_test,y_test)
29 plt.plot(y_pred, y_test)
30
31 MeanSqEr = sklearn.metrics.mean_squared_error(y_test, y_pred)
32 print(MeanSqEr)
```

Finally, I again used the *LinearRegression()* function and plotted the best fit line. The last two lines of code that give the MSE for the created linear regression model are extremely important.

Figure:3.3.1.e



As seen in the output above, the linear regression model does an excellent job in predicting the value of the stock price compared to the actual values. This can be further seen when we look at the MSE.

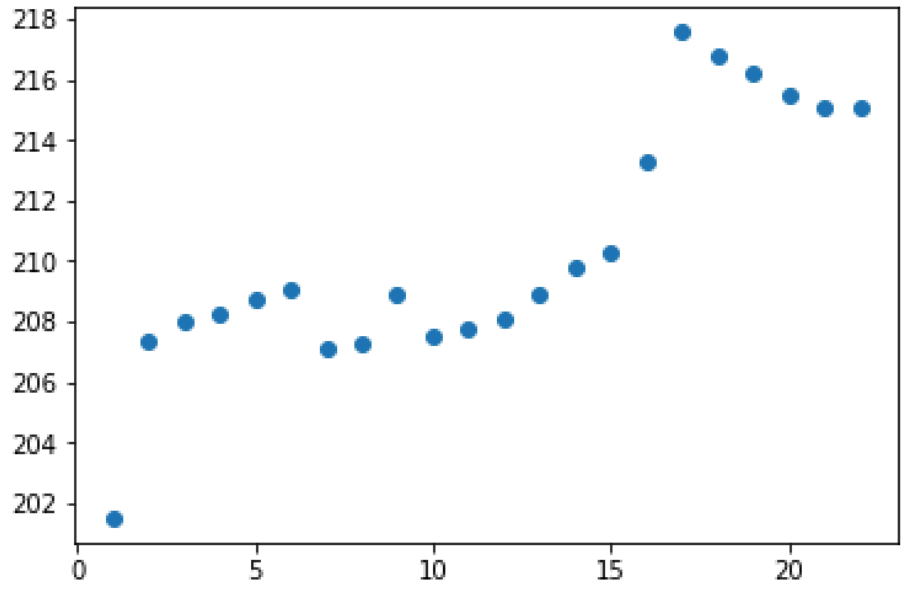
Figure: 3.3.1.f

```
0.6665730755952063
```

The MSE is only 0.66 which is very low and hence we conclude that this is an accurate linear regression model. This is fairly good for longer time durations but will not be as efficient in predicting stock prices over a shorter period of time.

3.3.2 Modeling Apple Stocks : Program D[13]

For obtaining a model that can accurately predict stock prices over a shorter period of time rather than a greater trend, we have extracted the data for one month starting 22nd July to 22nd August from *HistoricalQuotes.csv* and created a scatter plot for the same. The graph obtained: Figure: 3.3.2.a



As assumable from above, linear regression might not be able to predict as efficiently as a polynomial regression model. Hence, I fit this data into multiple models - 1 linear and 3 polynomial regression models.

But I will be evaluating the code in blocks.

[13] Complete Code and Output in Appendix 5.1.5

Figure:3.3.2.b

```
from numpy import *

x = array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22])
y = array([201.5,207.39,207.99,208.23,208.74,209.07,207.11,207.25,208.88,207.53,207.79,208.05,
           208.87,209.75,210.24,213.32,217.58,216.79,216.25,215.46,215.04,215.05])

```

The above code block is just assigning the data I have extracted from the dataset to the x and y variables.

Figure:3.3.2.c

```
from scipy.interpolate import *

lin = polyfit(x,y,1)
quad = polyfit(x,y,2)
cubic = polyfit(x,y,3)
high = polyfit(x,y,4)
```

In this code block I have made multiple linear and polynomial functions and fit the data into the models assigning them a degree from 1 to 4 respectively.

Figure:3.3.2.d

```
from matplotlib.pyplot import *

plot(x,y,'o')
bf = linspace(1,25,500)
plot(bf,polyval(lin,bf),'b-')
plot(bf,polyval(quad,bf),'r--' )
plot(bf,polyval(cubic,bf),'b:')
plot(bf,polyval(high,bf),'g:')

```

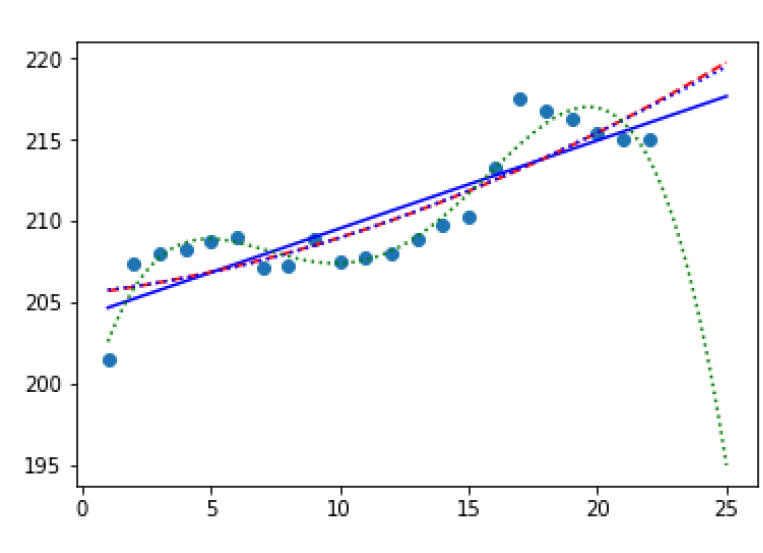
In this code block I have made a scatter plot and then have plotted all the different models on the graph. I have given all the model different colors and assigned them all different colors and patterns to make them easily differentiable.

Figure: 3.3.2.e

```
yfit = lin[0] * x + lin[1]
yresi = y-yfit
SqSumResi = sum(pow(yresi,2))
SqSumTotal = len(y) * var(y)
rsquared = 1 - SqSumResi/SqSumTotal
print(rsquared)
```

In this final part of code, I have calculated the Mean Squared Error for the linear regression model using multiple mathematical operations.

The graph obtained from the following code is as below: Figure:3.3.2.f



The MSE was calculated to be: Figure: 3.3.2.g

```
0.7372596630250858
```

4 Evaluation and Conclusion

After all those long lines of code and the final graph obtained from the output of the final program, there are many observations and evaluations.

4.1 General Observations and Evaluations

Firstly, the general trend of the Stock prices over a longer period of time can be modelled accurately using a linear regression graph that minimizes loss.

However, when we break down the graph and look at a certain segment i.e. certain shorter durations of time, Linear regression although efficient is still not the most accurate.

Here, we can see that the lower degree polynomials are slightly better compared to the linear regression line in modelling and prediction.

We can also compare the Mean Squared Error between the 1st and the 2nd. The Mean Squared Error for the 1st program is 0.666 which is significantly less than the 0.737 of the 2nd program. One might argue that the difference between the two is only 0.071 which is a very small number; however, when creating and training regression models to predict such numerical data, this small difference is also significant.

PROGRAM	ALGORITHM USED	TIME PERIOD	MEAN SQUARED ERROR
Program A	Linear Regression	1 year	28.54
Program B	Polynomial Regression	Not Applicable	1.73
Program C	Linear Regression	3 Months	0.666
Program D	Linear and Polynomial Regression	1 Month	0.737

4.2 Conclusion and Limitations

Pros and Cons of Linear and Polynomial Regression models

4.2.1 Linear Regression

After completing and training the model, I found that there are many pros and benefits of using a linear regression model. Firstly, it is the simplest model both to understand and create. Secondly, it is very efficient in modelling and predicting features that have a clear linear relation. It can be trained easily in a short period of time and works well with unseen data. Calculating loss takes only a few steps and generally loss calculated is quite low on average. Furthermore,if loss is greater, the model can be easily tweaked to perform better with new data.

Although there are numerous benefits of using linear models, there are some cons and limitations as well. The most basic being that it doesn't work well with data that has non-linear relations as seen in Figure: 3.2. Another limitations is that it is not efficient in modelling and providing a best fit line for data with multiple features, especially not with features of different data(numerical) types. Lastly, it is not very realistic and is prone to greta increase in loss from extreme values and unpredictable data.

4.2.2 Polynomial Regression

Similar to the linear model, the polynomial model also has its own pros. It is far better than Linear regression models in modelling non-linear data. It can work with multiple features and different numerical data types. It can be of varying degrees from quadratic all the way to an infinite degree polynomial. It is flexible to extreme and unpredictable values and can easily adapt to them. Lastly and most importantly, it is more realistic than the linear model.

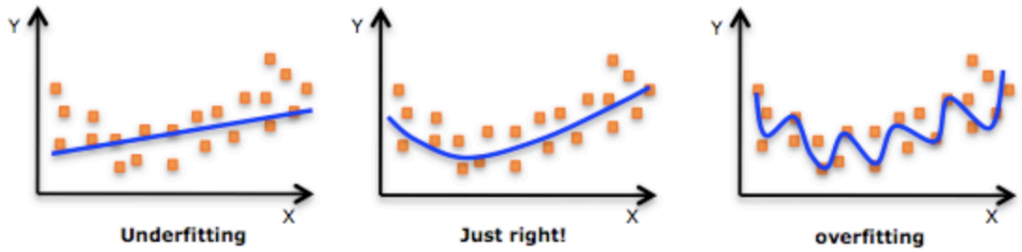
There are quite a few limitations and cons to the Polynomial model as well. Firstly, it is not as easy as linear regression model to create and train and sometimes tend to become very complex as we work with higher degree polynomials. It is not as easy to calculate loss of polynomial curves. As the degree of the polynomial increases, the model becomes more and more unpredictable and complicated and takes unnecessary points in the curve as well. It can sometimes be over-flexible and this leads to

overfitting. This can also be seen in the output of higher degree polynomials in Figure:3.3.2.f.

4.3 Regression Spline

There is a certain regression technique that solved the aforementioned problem of overfitting. This technique involves combining linear regression and polynomial regression to make a model that fits the points just flexibly enough. Such a technique is known as Regression spline.[14]

Figure:4



Due to its ability to simply overfitting polynomial functions, Regression Spline is one amongst the most crucial non linear regressions. In polynomial regression, we produced new features by utilizing different polynomial functions on the current features which forced a global structure on the on the dataset. To avoid this, we can partition the distribution of data into discrete segments and fit low degree polynomials or straight line

[14] (Singh)

functions on every one of these segments. This is the method used in most common regression models.

4.4 Final Conclusion

Thus, to finally conclude and answer the research question “which is better: linear or polynomial?” , linear regression models are better in representing and modelling trends over longer periods of time but lower degree polynomials more accurately represent relations over shorter time durations. This is supported by the observations made in the outputs for Programs C and D (Figures: 3.3.1.e & 3.3.2.f) and is also supporting my hypothesis.

5 Bibliographies and Appendices

1. "Descending Into ML: Linear Regression | Machine Learning Crash Course | Google Developers". *Google Developers*, 2018, https://developers.google.com/machine-learning/crash-course/descending-into-ml/linear-regression. Accessed 24 Aug 2018.
2. "Gradient Descent — ML Cheatsheet Documentation". *Ml-Cheatsheet.Readthedocs.Io*, 2018, http://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html. Accessed 24 Aug 2018.
3. "Reducing Loss: Gradient Descent | Machine Learning Crash Course | Google Developers". *Google Developers*, 2018, https://developers.google.com/machine-learning/crash-course/reducing-loss/gradient-descent. Accessed 24 Aug 2018.
4. "The Most Insightful Stories About Polynomial Regression – Medium". *Medium.Com*, 2018, https://medium.com/tag/polynomial-regression. Accessed 24 Aug 2018.
5. "What Is Artificial Intelligence (AI)? - Definition From Techopedia". *Techopedia.Com*, 2018, https://www.techopedia.com/definition/190/artificial-intelligence-ai. Accessed 24 Aug 2018.
6. Grobler, Jaques. "Linear Regression Example — Scikit-Learn 0.19.2 Documentation". *Scikit-Learn.Org*, 2018, http://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html. Accessed 24 Aug 2018.
7. Singh, Gurchetan. "Introduction To Regression Splines (With Python Codes)". *Analytics Vidhya*, 2018, https://www.analyticsvidhya.com/blog/2018/03/introduction-regression-splines-python-codes/. Accessed 24 Aug 2018.
8. Brownlee, Jason. "Supervised And Unsupervised Machine Learning Algorithms". *Machine Learning Mastery*, 2018, https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/. Accessed 24 Aug 2018.
9. "Python Machine Learning Tutorial, Scikit-Learn: Wine Snob Edition". *Elitedatascience*, 2018, https://elitedatascience.com/python-machine-learning-tutorial-scikit-learn. Accessed 24 Aug 2018.
10. "Import And Plot Stock Price Data With Python, Pandas And Seaborn". *August Kleimo*, 2018, http://www.augustkleimo.com/import-and-plot-stock-price-data-with-python-pandas-and-seaborn/. Accessed 24 Aug 2018.
11. "Apple Inc. Common Stock Historical Stock Prices". *NASDAQ*, 2018, https://www.nasdaq.com/symbol/aapl/historical. Accessed 24 Aug 2018.

5.1 Appendix

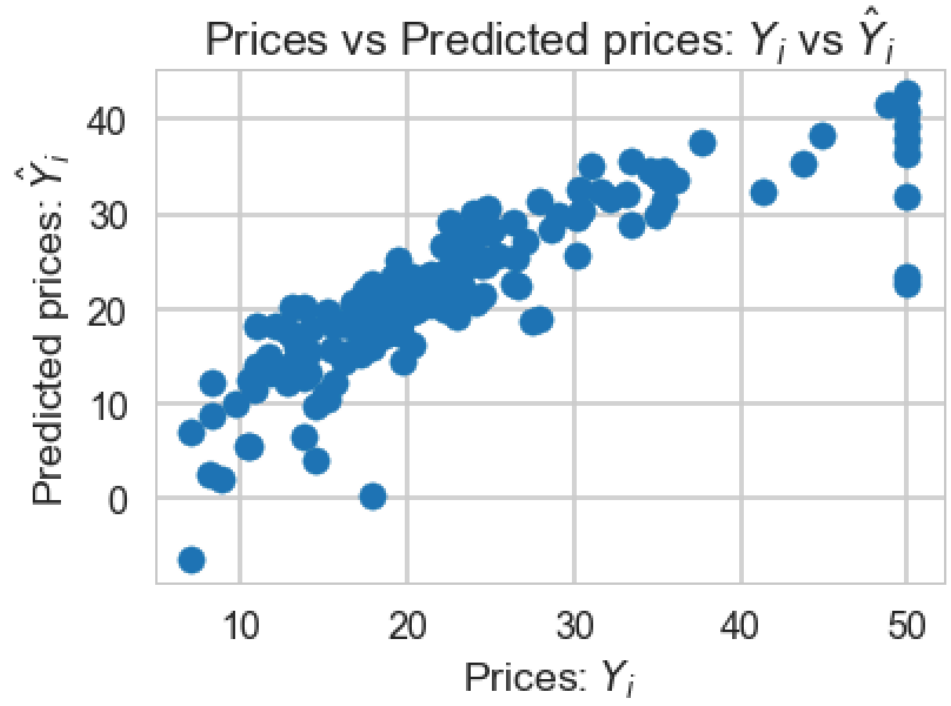
5.1.1 Program A

Python Code:

```
import pandas as pd
import matplotlib.pyplot as plt
import sklearn

import seaborn as sns
sns.set_style("whitegrid")
sns.set_context("poster")

from sklearn.datasets import load_boston
boston = load_boston()
bos = pd.DataFrame(boston.data)
bos.columns = boston.feature_names
bos['PRICE'] = boston.target
print(bos.head())

X = bos.drop('PRICE', axis = 1)
Y = bos['PRICE']

X_train, X_test, Y_train, Y_test = (sklearn.model_selection.
                                    train_test_split(X, Y, test_size = 0.33, random_state = 5))
print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)

from sklearn.linear_model import LinearRegression

lm = LinearRegression()
lm.fit(X_train, Y_train)

Y_pred = lm.predict(X_test)

plt.scatter(Y_test, Y_pred)
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")

mse = sklearn.metrics.mean_squared_error(Y_test, Y_pred)
print(mse)
```

Output:

```
      CRIM    ZN  INDUS  CHAS  ...    PTRATIO       B  LSTAT  PRICE
0  0.00632  18.0   2.31   0.0  ...       15.3  396.90   4.98   24.0
1  0.02731   0.0   7.07   0.0  ...       17.8  396.90   9.14   21.6
2  0.02729   0.0   7.07   0.0  ...       17.8  392.83   4.03   34.7
3  0.03237   0.0   2.18   0.0  ...       18.7  394.63   2.94   33.4
4  0.06905   0.0   2.18   0.0  ...       18.7  396.90   5.33   36.2

[5 rows x 14 columns]
(339, 13)
(167, 13)
(339,)
(167,)
28.54136727561835
```

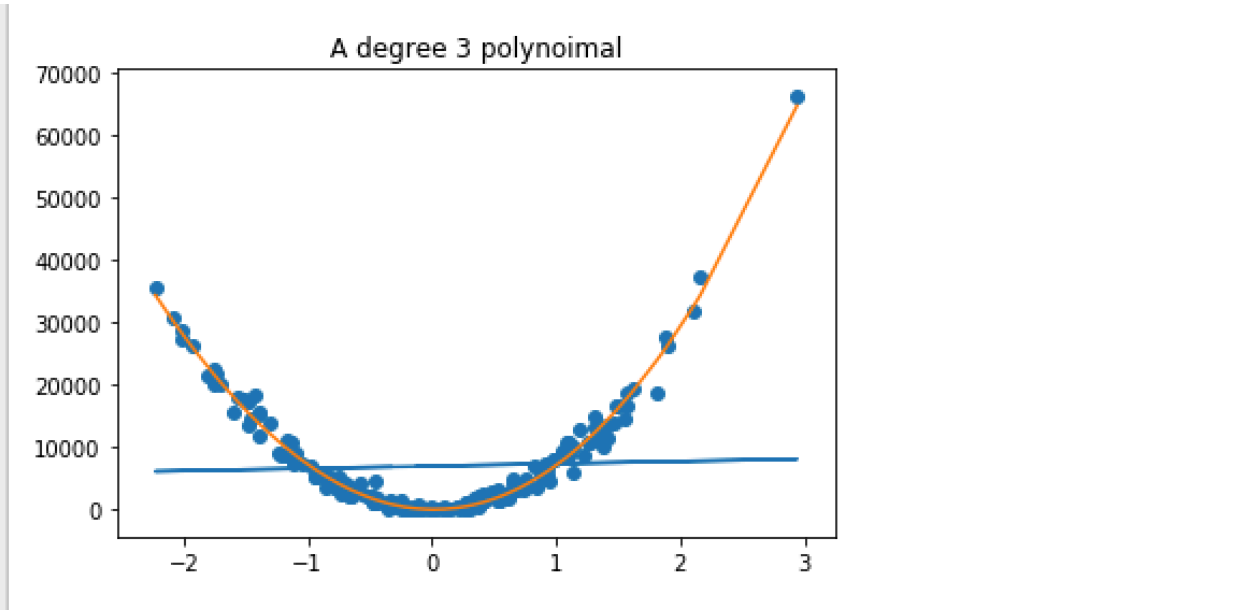


5.1.2 Program B

Python Code:

```
from sklearn.datasets import make_regression
import matplotlib.pyplot as plt
X, y = make_regression(n_samples = 200, n_features=1, noise=8, bias=2)
y2 = y**2
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X, y2)
plt.plot(X, model.predict(X))
import numpy as np
from sklearn.preprocessing import PolynomialFeatures
poly_features = PolynomialFeatures(degree = 3)
X_poly = poly_features.fit_transform(X)
poly_model.fit(X_poly, y2)

pred = poly_model.predict(X_poly)
new_X, new_y = zip(*sorted(zip(X, pred)))
plt.plot(new_X, new_y)
plt.scatter(X,y2)
plt.title("A degree 3 polynoimal")
```

Output:



5.1.3 Dataset

HistoricalQuotes.csv:

```
"date","close","volume","open","high","low"
"15:48","215.645","14,808,142","214.60","217.05","214.60"
"2018/08/22","215.0500","18965930.0000","214.1000","216.3600","213.8400"
"2018/08/21","215.0400","25682700.0000","216.8000","217.1900","214.0250"
"2018/08/20","215.4600","30149410.0000","218.1000","219.1800","215.1100"
"2018/08/17","217.5800","35034410.0000","213.4400","217.9500","213.1600"
"2018/08/16","213.3200","27595310.0000","211.7500","213.8121","211.4700"
"2018/08/15","210.2400","28595230.0000","209.2200","210.7400","208.3300"
"2018/08/14","209.7500","20679270.0000","210.1550","210.5600","208.2600"
"2018/08/13","208.8700","25864510.0000","207.7000","210.9520","207.7000"
"2018/08/10","207.5300","24592460.0000","207.3600","209.1000","206.6700"
"2018/08/09","208.8800","23389530.0000","207.2800","209.7800","207.2000"
"2018/08/08","207.2500","22493750.0000","206.0500","207.8100","204.5200"
"2018/08/07","207.1100","25525650.0000","209.3200","209.5000","206.7600"
"2018/08/06","209.0700","25396100.0000","208.0000","209.2500","207.0700"
"2018/08/03","207.9900","33408050.0000","207.0300","208.7400","205.4803"
"2018/08/02","207.3900","62317430.0000","200.5800","208.3800","200.3500"
"2018/08/01","201.5000","67812280.0000","199.1300","201.7600","197.3100"
"2018/07/31","190.2900","37847260.0000","190.3000","192.1400","189.3400"
"2018/07/30","189.9100","20912890.0000","191.9000","192.2000","189.0700"
"2018/07/27","190.9800","23975350.0000","194.9900","195.1900","190.1000"
"2018/07/26","194.2100","19005150.0000","194.6100","195.9600","193.6100"
"2018/07/25","194.8200","16553500.0000","193.0600","194.8500","192.4300"
"2018/07/24","193.0000","18680930.0000","192.4500","193.6601","192.0500"
"2018/07/23","191.6100","15955820.0000","190.6800","191.9600","189.5600"
"2018/07/20","191.4400","20670830.0000","191.7800","192.4300","190.1700"
"2018/07/19","191.8800","20182050.0000","189.6900","192.5500","189.6900"
"2018/07/18","190.4000","16365720.0000","191.7800","191.8000","189.9300"
"2018/07/17","191.4500","15515000.0000","189.7500","191.8700","189.2000"
"2018/07/16","190.9100","15009800.0000","191.5200","192.6500","190.4150"
"2018/07/13","191.3300","12506830.0000","191.0800","191.8400","190.9000"
"2018/07/12","191.0300","18000180.0000","189.5300","191.4100","189.3100"
"2018/07/11","187.8800","18776390.0000","188.5000","189.7799","187.6100"
"2018/07/10","190.3500","15801370.0000","190.7100","191.2800","190.1801"
"2018/07/09","190.5800","19636390.0000","189.5000","190.6800","189.3000"
```

```
"2018/07/06","187.9700","17426160.0000","185.4200","188.4340","185.2000"
"2018/07/05","185.4000","16543880.0000","185.2600","186.4100","184.2800"
"2018/07/03","183.9200","13954810.0000","187.7900","187.9500","183.5400"
"2018/07/02","187.1800","17675210.0000","183.8200","187.3000","183.4200"
"2018/06/29","185.1100","22551660.0000","186.2900","187.1900","182.9100"
"2018/06/28","185.5000","17352380.0000","184.1000","186.2100","183.8000"
"2018/06/27","184.1600","24987260.0000","185.2280","187.2800","184.0300"
"2018/06/26","184.4300","24378480.0000","182.9900","186.5300","182.5400"
"2018/06/25","182.1700","31632320.0000","183.4000","184.9200","180.7300"
"2018/06/22","184.9200","27185130.0000","186.1200","186.1500","184.7000"
"2018/06/21","185.4600","25649240.0000","187.2500","188.3500","184.9400"
"2018/06/20","186.5000","20490090.0000","186.3500","187.2000","185.7300"
"2018/06/19","185.6900","33538550.0000","185.1400","186.3300","183.4500"
"2018/06/18","188.7400","18458410.0000","187.8800","189.2200","187.2000"
"2018/06/15","188.8400","61438290.0000","190.0300","190.1600","188.2600"
"2018/06/14","190.8000","21584720.0000","191.5500","191.5700","190.2200"
"2018/06/13","190.7000","21480120.0000","192.4200","192.8800","190.4400"
"2018/06/12","192.2800","16859020.0000","191.3850","192.6110","191.1500"
"2018/06/11","191.2300","18069930.0000","191.3500","191.9700","190.2100"
"2018/06/08","191.7000","26580710.0000","191.1700","192.0000","189.7700"
"2018/06/07","193.4600","21329170.0000","194.1400","194.2000","192.3350"
"2018/06/06","193.9800","20912820.0000","193.6300","194.0800","191.9200"
"2018/06/05","193.3100","21535570.0000","193.0650","193.9400","192.3600"
"2018/06/04","191.8300","26190620.0000","191.6350","193.4200","191.3500"
"2018/06/01","190.2400","23377690.0000","187.9910","190.2600","187.7500"
"2018/05/31","186.8700","27468700.0000","187.2200","188.2300","186.1400"
"2018/05/30","187.5000","18503410.0000","187.7200","188.0000","186.7800"
"2018/05/29","187.9000","22409110.0000","187.6000","188.7500","186.8700"
"2018/05/25","188.5800","17186760.0000","188.2300","189.6500","187.6500"
"2018/05/24","188.1500","23129400.0000","188.7700","188.8400","186.2100"
"2018/05/23","188.3600","19966590.0000","186.3500","188.5000","185.7600"
"2018/05/22","187.1600","15228270.0000","188.3750","188.8800","186.7800"
```

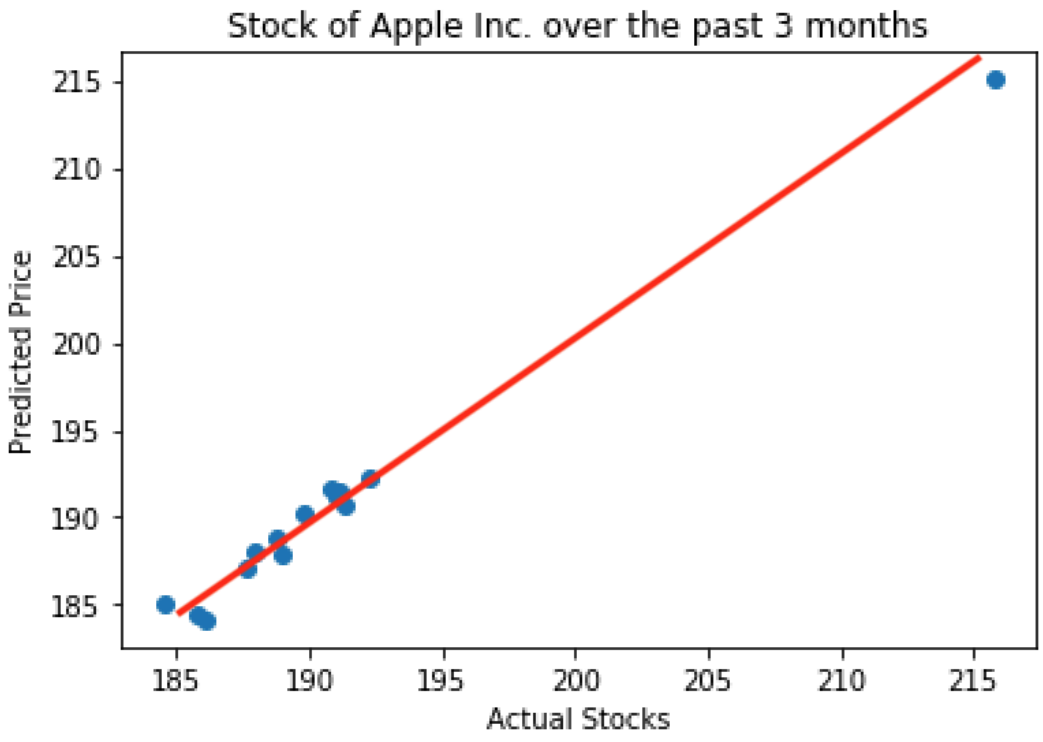
5.1.4 Program C

Python Code:

```
import pandas as pd
import sklearn
from sklearn.model_selection import train_test_split
mydata = pd.read_csv('HistoricalQuotes.csv')
dataset = pd.DataFrame(mydata)
print(dataset)

x = dataset.drop('close', axis = 1)
y = dataset['close']

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state = 123)

from sklearn.linear_model import LinearRegression
lm = LinearRegression()
lm.fit(x_train,y_train)

y_pred = lm.predict(x_test)

import matplotlib.pyplot as plt

plt.scatter(y_pred, y_test)
plt.xlabel("Actual Stocks")
plt.ylabel("Predicted Price")
plt.title("Stock of Apple Inc. over the past 3 months")

from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_test,y_test)
plt.plot(y_pred, y_test)

MeanSqEr = sklearn.metrics.mean_squared_error(y_test, y_pred)
print(MeanSqEr)
```

Output:

```
     close     open      high       low
0   216.44  214.600  217.0500  214.6000
1   215.05  214.100  216.3600  213.8400
2   215.04  216.800  217.1900  214.0250
3   215.46  218.100  219.1800  215.1100
4   217.58  213.440  217.9500  213.1600
5   213.32  211.750  213.8121  211.4700
6   210.24  209.220  210.7400  208.3300
7   209.75  210.155  210.5600  208.2600
8   208.87  207.700  210.9520  207.7000
9   207.53  207.360  209.1000  206.6700
10  208.88  207.280  209.7800  207.2000
11  207.25  206.050  207.8100  204.5200
12  207.11  209.320  209.5000  206.7600
13  209.07  208.000  209.2500  207.0700
14  207.99  207.030  208.7400  205.4803
15  207.39  200.580  208.3800  200.3500
16  201.50  199.130  201.7600  197.3100
17  190.29  190.300  192.1400  189.3400
18  189.91  191.900  192.2000  189.0700
19  190.98  194.990  195.1900  190.1000
20  194.21  194.610  195.9600  193.6100
21  194.82  193.060  194.8500  192.4300
22  193.00  192.450  193.6601  192.0500
23  191.61  190.680  191.9600  189.5600
24  191.44  191.780  192.4300  190.1700
25  191.88  189.690  192.5500  189.6900
26  190.40  191.780  191.8000  189.9300
27  191.45  189.750  191.8700  189.2000
28  190.91  191.520  192.6500  190.4150
29  191.33  191.080  191.8400  190.9000
..     ...      ...       ...       ...


36  183.92  187.790  187.9500  183.5400
37  187.18  183.820  187.3000  183.4200
38  185.11  186.290  187.1900  182.9100
39  185.50  184.100  186.2100  183.8000
40  184.16  185.228  187.2800  184.0300
41  184.43  182.990  186.5300  182.5400
42  182.17  183.400  184.9200  180.7300
43  184.92  186.120  186.1500  184.7000
44  185.46  187.250  188.3500  184.9400
45  186.50  186.350  187.2000  185.7300
46  185.69  185.140  186.3300  183.4500
47  188.74  187.880  189.2200  187.2000
48  188.84  190.030  190.1600  188.2600
49  190.80  191.550  191.5700  190.2200
50  190.70  192.420  192.8800  190.4400
51  192.28  191.385  192.6110  191.1500
52  191.23  191.350  191.9700  190.2100
53  191.70  191.170  192.0000  189.7700
54  193.46  194.140  194.2000  192.3350
55  193.98  193.630  194.0800  191.9200
56  193.31  193.065  193.9400  192.3600
57  191.83  191.635  193.4200  191.3500
58  190.24  187.991  190.2600  187.7500
59  186.87  187.220  188.2300  186.1400
60  187.50  187.720  188.0000  186.7800
61  187.90  187.600  188.7500  186.8700
62  188.58  188.230  189.6500  187.6500
63  188.15  188.770  188.8400  186.2100
64  188.36  186.350  188.5000  185.7600
65  187.16  188.375  188.8800  186.7800

[66 rows x 4 columns]
0.6665730755952063
```


Stock of Apple Inc. over the past 3 months
Predicted Price
Actual Stocks
215
210
205
200
195
190
185
185
190
195
200
205
210
215

5.1.5 Program D

Python Code:

```
from numpy import *

x = array([1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22])
y = array([201.5,207.39,207.99,208.23,208.74,209.07,207.11,207.25,208.88,207.53,207.79,208.05,
           208.87,209.75,210.24,213.32,217.58,216.79,216.25,215.46,215.04,215.05])

from scipy.interpolate import *

lin = polyfit(x,y,1)
quad = polyfit(x,y,2)
cubic = polyfit(x,y,3)
high = polyfit(x,y,4)

from matplotlib.pyplot import *

plot(x,y,'o')
bf = linspace(1,25,500)
plot(bf,polyval(lin,bf),'b-')
plot(bf,polyval(quad,bf),'r--' )
plot(bf,polyval(cubic,bf),'b:')
plot(bf,polyval(high,bf),'g:')

yfit = lin[0] * x + lin[1]
yresi = y-yfit
SqSumResi = sum(pow(yresi,2))
SqSumTotal = len(y) * var(y)
rsquared = 1 - SqSumResi/SqSumTotal
print(rsquared)

```

Output: