Predicting Stock Market Prices Using Linear & Polynomial Regression

Author: Luke Liu, **Course:** EECS 16A, **Term:** Fall 2019

Introduction

The stock market is an extremely complex system that is difficult to model due to a large number of factors that determine the day-to-day price changes. One of the most exciting and useful applications of machine learning, is to predict these price changes days, months, or perhaps years into the future. Although this application was briefly mentioned by Professor Ranade in lecture, it is unfortunately not further explored in the context of EECS 16A. In this demo, we will build our own linear regression and polynomial regression models based on historical stock prices from top companies, examine their effectiveness, and use them to predict price changes in the stock market. In the end, we will also explore higher-level machine learning techniques that can be used more effectively to achieve our goal.

Wait... What even is linear regression and polynomial regression??

Linear Regression

Linear regression models the relationship between two variables through fitting a linear line of best fit to the observed data. It is the most basic and one of the most widely used predictive analysis techniques in machine learning. The general linear regression formula is:

$$y_i = \alpha_0 + \alpha_1 x_i + \epsilon_i$$

To revisit, Homework 13 Question 2 is an example of building a linear regression model: [http://www.eecs16a.org/homework/prob13.pdf (http://www.eecs16a.org/homework/prob13.pdf)](http://www.eecs16a.org/homework/prob13.pdf)

Polynomial Regression

Polynomial regression is a special case of linear regression. It models the relationship between two variables through fitting an n-th degree polynomial to the observed data. The general polynomial regression formula is:

$$y_i = \alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 + \ldots + \alpha_n x_i^n + \epsilon_i$$

To revisit, Discussion 13A Question 3 is an example of building a polynomial regression model of degree 4: http://www.eecs16a.org/discussion/dis13A.pdf#page=3 (http://www.eecs16a.org/discussion/dis13A.pdf#page=3)

Setup Code

You do not have to worry about understanding the code here. Source: Homework 14 Problem 1.

In [1]:
```
import numpy as np
import numpy.matlib
import matplotlib.pyplot as plt

%matplotlib inline

"""Function that constructs a polynomial curve for a set of coefficien
ts
   that multiply the polynomial terms and the x range."""
def poly_curve(params,x_input):
    # params contains the coefficients that multiply the polynomial te
rms, in degree of lowest degree to highest degree
    degree=len(params)-1
    x_range=[x_input[1], x_input[-1]]
    x=np.linspace(x_range[0],x_range[1],1000)
    y=x*0

    for k in range(0,degree+1):
        coeff=params[k]
        y=y+list(map(lambda z:coeff*z**k,x))
    return x,y

"""Function that defines a data matrix for some input data."""
def data_matrix(input_data,degree):
    # degree is the degree of the polynomial you plan to fit the data
with
    Data=np.zeros((len(input_data),degree+1))

    for k in range(0,degree+1):
        Data[:,k]=(list(map(lambda x:x**k ,input_data)))

    return Data

"""Given a set of x and y points, and a range of polynomial degrees to
try, this function calculates polynomial
   fits to the data for polynomials of different degrees. It returns t
he "cost", i.e. the magnitude of the error
   vector for each fit. The output is an array of the cost correspondi
ng to each degree. """
def cost(x, y, start_deg, end_deg):
    c = []
    for degree in range(start_deg, end_deg):
        D = data_matrix(x, degree)
        params = leastSquares(D, y)
        error = np.linalg.norm(y-np.dot(D, params))
        c.append(error)
    return c
```

Time to Import Real Data!

Historical stock data for top tech companies (Facebook, Apple, Amazon) have already been compiled from Yahoo Finance. It consists of all their stock prices from the last year **(Dec. 10, 2018 to Dec. 5, 2019)**. Fun (and important) fact: Stock market only runs on **business days**! As a result, there would only be a total of 250 days in the data. Feel free to check out their CSV spreadsheets contained in the demo folder; their data contain each day's opening, highest, lowest, closing prices, and more. However, we will only use the **closing price of stocks** at the end of each business day. Although day trading is not modeled, it is an active area of research that you are encouraged to look into!

Now it's time to pick your favourite company!

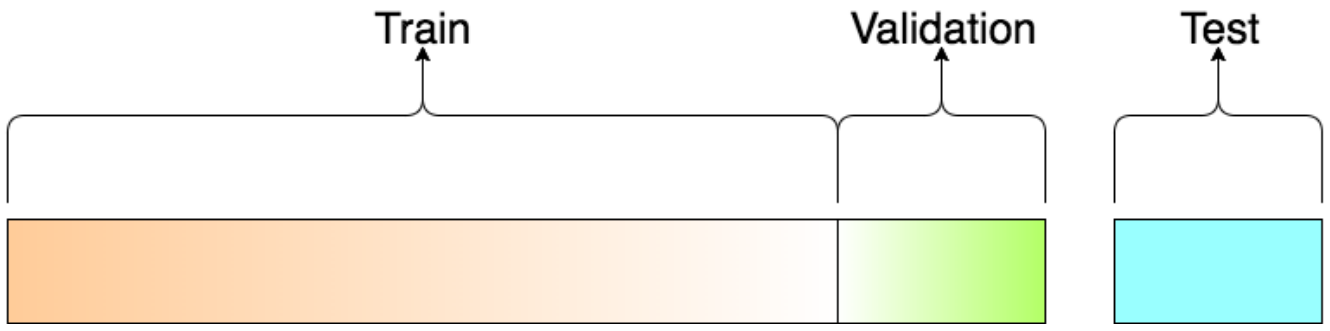
In [2]:
```
apple = np.loadtxt('data/apple.csv', delimiter=",", skiprows=1, usecol
s=(1, 4))
facebook = np.loadtxt('data/facebook.csv', delimiter=",", skiprows=1,
usecols=(1, 4))
amazon = np.loadtxt('data/amazon.csv', delimiter=",", skiprows=1, usec
ols=(1, 4))

stock_data = apple ### Your choice here
```

In real-world machine learning, the observed data is typically divided into the following three sets. In the simplest words,

Training Set: This set is used to fit the model.

Validation Set: This set evaluates the effectiveness of the model and adjusts it when necessary.

Testing Set: This set evaluates the effectiveness of the final model.



A visualization of the split

Credit/I recommend further reading here: https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7 (https://towardsdatascience.com/train-validation-and-test-sets-72cb40cba9e7)

For simplicity purposes, we will only be dealing with training set and testing set. Validation set is beyond the scope of EECS 16A.

We will be dedicating 80% of the data (200 days) towards the training set and 20% of the data (50 days) towards the testing set. Since we do do not know whether our model would accurately fit the testing data, it is safer that the testing set contains days that come before the days in the training set, so we can more effectively predict future prices.

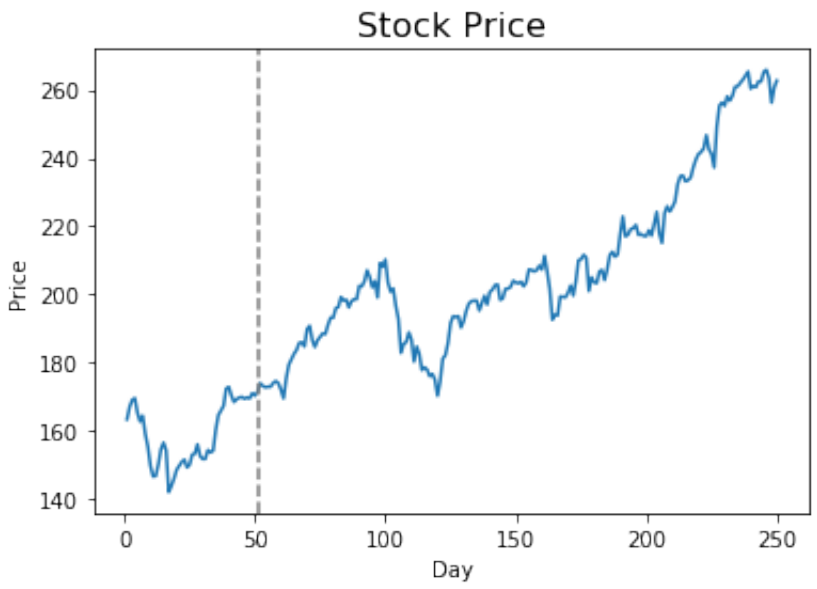
Our testing set contains the stock market prices from day 1 to 50, while the training set contains the stock market prices from day 51 to 250.

In [3]:
```
day_change = 51
training_data = stock_data[day_change:]
testing_data = stock_data[:day_change]

dates, prices = stock_data[:, 0], stock_data[:, 1]
dates_t, prices_t = training_data[:, 0], training_data[:, 1]
dates_v, prices_v = testing_data[:, 0], testing_data[:, 1]
dates_p = np.arange(1, 300)
```

Run the block below to see the company's stock trend last year. Keep in mind we will use the days 51-250 to fit the data and days 1-50 to see how effective the fit is.

In [4]:
```
fig = plt.figure()
ax = fig.add_subplot()
ax.plot(dates, prices)
plt.title('Stock Price', fontsize=16)
plt.axvline(day_change, color='gray', linestyle='--')
plt.xlabel('Day')
plt.ylabel('Price')
```

Out[4]:
```
Text(0, 0.5, 'Price')
```



Linear Regression

Stage 1: Training/Modeling

First, we will apply the linear regression technique to model the stock price. Recall the equation

$$y_i = \alpha_0 + \alpha_1 x_i$$

What are the unknowns in this case? What is given? What technique can we use to solve this system?

Answer: The unknowns are α_0 and α_1. There are 250 sets of x_i (date) and y_i (stock price) that are given (there are 250 days!) When there are more equations than variables (aka. when a system is overdetermined), we use least squares and model the problem as following:

$$\begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_{250} \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{250} \end{bmatrix}$$

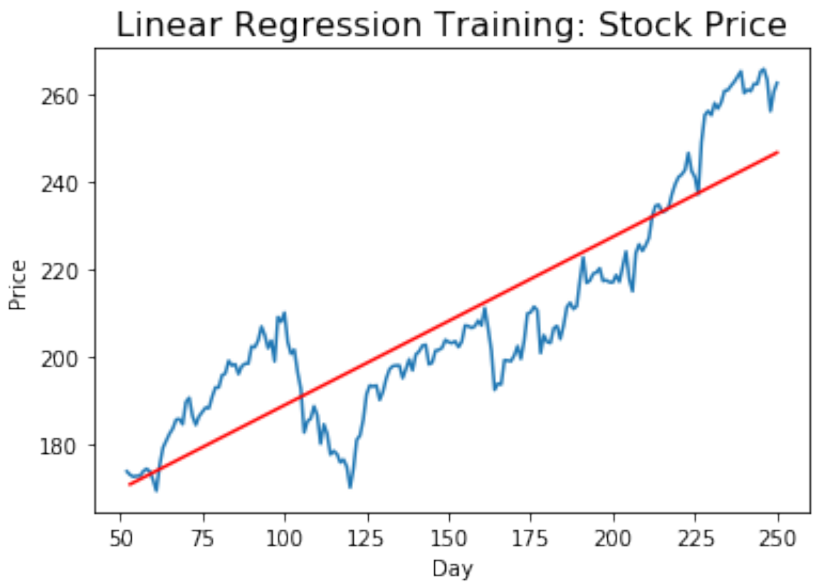
Now write the function that computes the least squares solution and solve for $\vec{\alpha}$!

```
In [5]: """Function that computes the Least Squares Approximation"""
        def leastSquares(A, y):
            return np.dot(np.linalg.inv(np.dot(A.T, A)), np.dot(A.T, y)) ### Y
        our answer here

        D_a = data_matrix(dates_t, 1) # Turn the dates into the matrix form
        sol_a = leastSquares(D_a, prices_t) # Computes least squares
```

Let's see how you did! Run the block below to plot your line of best fit as determined through linear regression.

In [6]:
```
fig = plt.figure()
ax = fig.add_subplot()
x_line, y_line = poly_curve(sol_a, dates_t)
ax.plot(dates_t, prices_t)
ax.plot(x_line, y_line, 'r')
plt.title('Linear Regression Training: Stock Price', fontsize=16)
plt.xlabel('Day')
plt.ylabel('Price')
```

Out[6]:
```
Text(0, 0.5, 'Price')
```

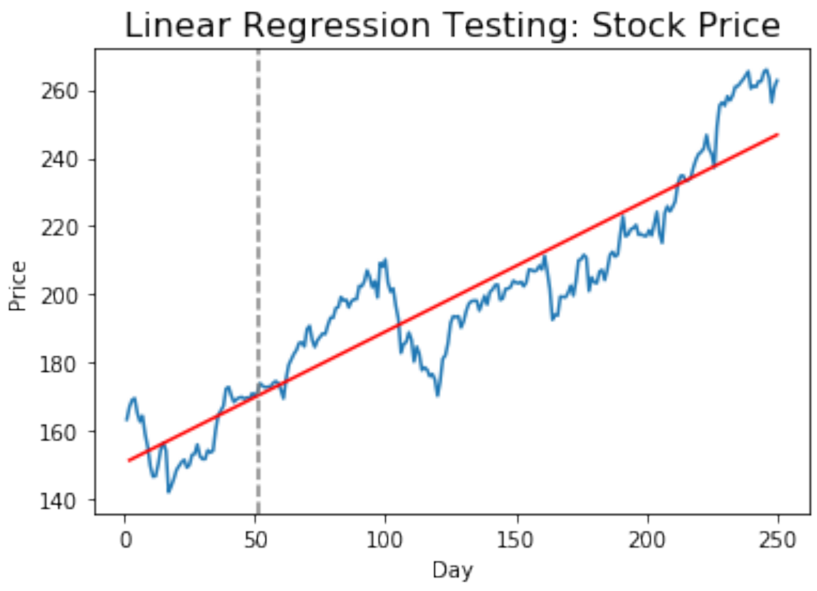


Stage 2: Testing

In the testing phase, we determine how effective our model is by looking at days 0-50. Run the block below to see the result. Would you say that this is an accurate model?

In [7]:
```
fig = plt.figure()
ax = fig.add_subplot()
x_line, y_line = poly_curve(sol_a, dates)
ax.plot(dates, prices)
ax.plot(x_line, y_line, 'r')
plt.axvline(day_change, color='gray', linestyle='--')
plt.title('Linear Regression Testing: Stock Price', fontsize=16)
plt.xlabel('Day')
plt.ylabel('Price')
```

Out[7]:
```
Text(0, 0.5, 'Price')
```



We can determine the magnitude of the error vector for the testing set by running the code block below.

In [8]:
```
D = data_matrix(dates_v, 1)
error = np.linalg.norm(prices_v-np.dot(D, sol_a))
print("Magnitude of error vector: " + str(error))
```

```
Magnitude of error vector: 59.711948207889755
```

Step 3: Predicting

Using the linear regression model that you have built, what would be the price of the stock for the company you have chosen in **N** market days?
(Treat today as December 5, 2019)

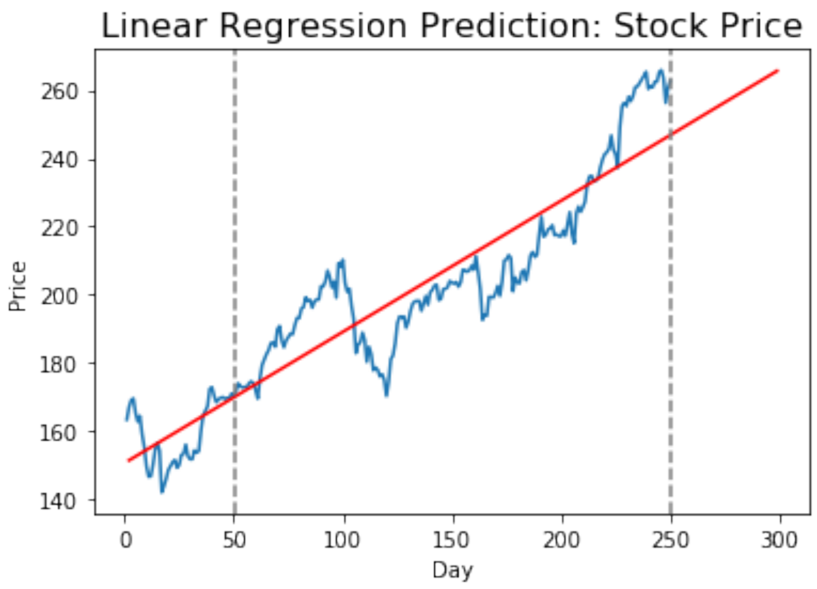
In [9]:
```
N = 5  ### Input your value

D = data_matrix([250+N], 1)
new_price = np.dot(D, sol_a)[0]
print("The stock price for the company you have chosen in " + str(N) +
" market days is: $" + str(round(new_price, 2)))
```

```
The stock price for the company you have chosen in 5 market days is:
$248.73
```

Let's extend our plot to predict the price for the upcoming 50 days!

In [10]:
```
fig = plt.figure()
ax = fig.add_subplot()
x_line, y_line = poly_curve(sol_a, dates_p)
ax.plot(dates, prices)
ax.plot(x_line, y_line, 'r')
plt.axvline(day_change, color='gray', linestyle='--')
plt.axvline(250, color='gray', linestyle='--')
plt.title('Linear Regression Prediction: Stock Price', fontsize=16)
plt.xlabel('Day')
plt.ylabel('Price')
```

Out[10]:
```
Text(0, 0.5, 'Price')
```



Analysis

What can you conclude about the advantages/disadvantages of a linear regression model? Do you think that it is effective?

Advantages of using Linear Regression:

- High level of transparency on how prediction is produced
- Universally-accepted and most used/easiest predictive analytics method

Disadvantages of using Linear Regression:

- Can only be used to produce a linear curve
- Not that good in predictive analysis as it is highly restricted and and oversimply the complex situations

Polynomial Regression

Stage 1: Training/Modeling

First, we have to determine the degree that we want for the polynomial regression model. With the help of the cost function, we can determine the cost of the error vector for each degree. Intuitively, it may seem like we should pick the degree with the lowest cost because it supposedly fits the data the best. However, the degree with the lowest cost may not represent the best model since it may be due to the presence of outliers - refer to Homework 14 Problem 1 (http://www.eecs16a.org/homework/prob14.pdf (http://www.eecs16a.org/homework/prob14.pdf)). We will tackle this problem by making a decision on which degree to use on our own, by examining the values returned by the cost function.

In [11]:
```
cost_list = cost(dates_t, prices_t, 1, 11)
for i in range(10):
    print("Degree " + str(i+1) + ": " + str(cost_list[i]))
```

```
Degree 1: 169.0603032354343
Degree 2: 122.17403565488395
Degree 3: 104.75020554892417
Degree 4: 96.89341330704795
Degree 5: 96.29033720037826
Degree 6: 91.35633472951811
Degree 7: 82.2124029583442
Degree 8: 74.74564551630871
Degree 9: 84.14478474576997
Degree 10: 6194.943863630576
```

Which degree would you pick and why?

Answer: We know that the higher the degree we choose, the more inaccurate our model would become since higher-order polynomials tend to "overfit." It seems like that choosing degree 3 may be the best option since it does not deviate so much from the lowest cost and it is a relatively low-degree polynomial. I encourage you to come back and try different degrees later!

Once again, we use least squares to compute the solution to

$$y_i = \alpha_0 + \alpha_1 x_i + \alpha_2 x_i^2 + \ldots + \alpha_n x_i^n$$

where n is the degree that we have chosen.

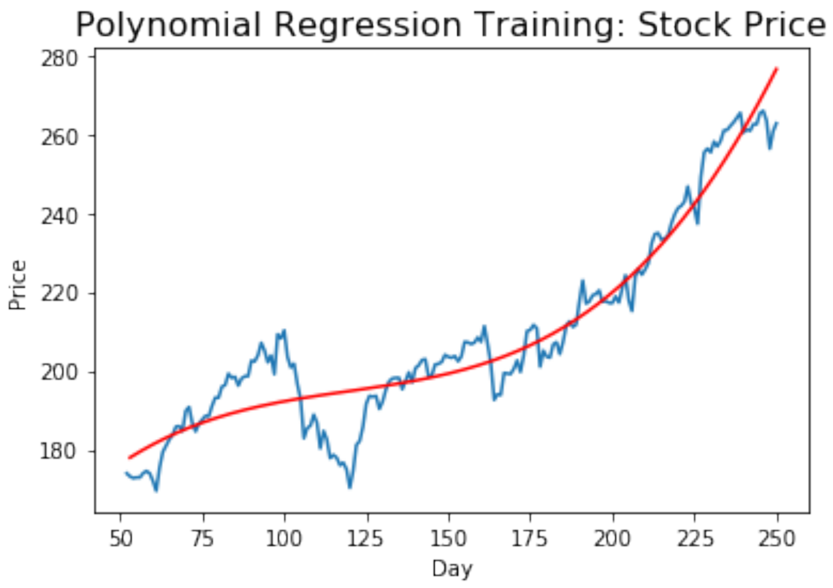
$$\begin{bmatrix} 1 & x_1 & x_1^2 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & \ldots & x_2^n \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{250} & x_{250}^2 & \ldots & x_{250}^n \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{250} \end{bmatrix}$$

In [12]:
```
degree = 3 ### Feel free to adjust this value to see different results
D_b = data_matrix(dates_t, degree)
sol_b = leastSquares(D_b, prices_t)
```

Let's see how you did! Run the code block below to display your curve of best fit.

```
In [13]: fig = plt.figure()
         ax = fig.add_subplot()
         x_curve, y_curve = poly_curve(sol_b, dates_t)
         ax.plot(dates_t, prices_t)
         ax.plot(x_curve, y_curve, 'r')
         plt.title('Polynomial Regression Training: Stock Price', fontsize=16)
         plt.xlabel('Day')
         plt.ylabel('Price')
```

```
Out[13]: Text(0, 0.5, 'Price')
```

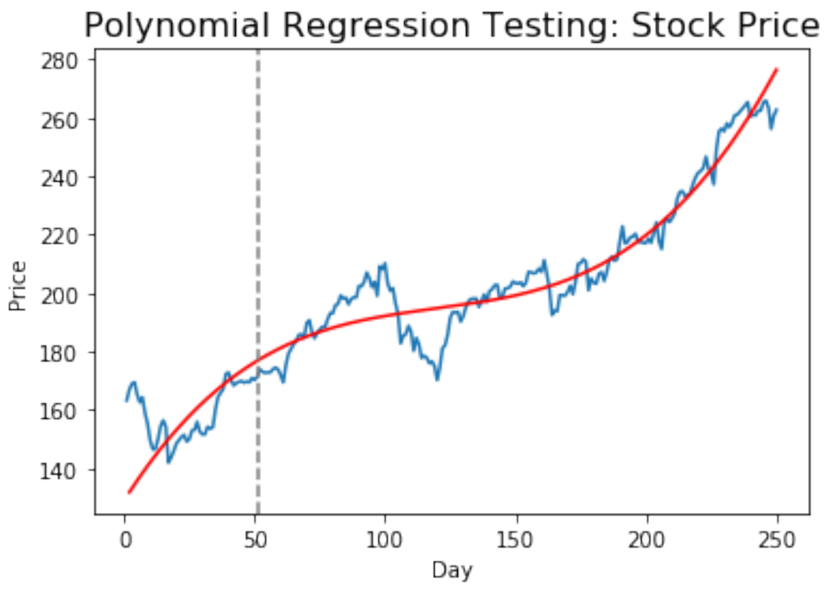


Stage 2: Testing

In the testing phase, we determine how effective our model is by looking at days 0-50. Run the block below to see the result. Would you say that this is an accurate model?

In [14]:
```
fig = plt.figure()
ax = fig.add_subplot()
x_curve, y_curve = poly_curve(sol_b, dates)
ax.plot(dates, prices)
ax.plot(x_curve, y_curve, 'r')
plt.axvline(day_change, color='gray', linestyle='--')
plt.title('Polynomial Regression Testing: Stock Price', fontsize=16)
plt.xlabel('Day')
plt.ylabel('Price')
```

Out[14]:
```
Text(0, 0.5, 'Price')
```



We can determine the magnitude of the error vector for the testing set by running the code block below. Compare it to the magnitude of error vector you calculated for linear regression. Which one is better?

In [15]:
```
D = data_matrix(dates_v, degree)
error = np.linalg.norm(prices_v-np.dot(D, sol_b))
print("Magnitude of error vector: " + str(error))
```

```
Magnitude of error vector: 96.26097572747166
```

Stage 3: Predicting

Using the polynomial regression you have built, what would be the price of the stock for the company you have chosen in **N** market days?
(Treat today as December 5, 2019)

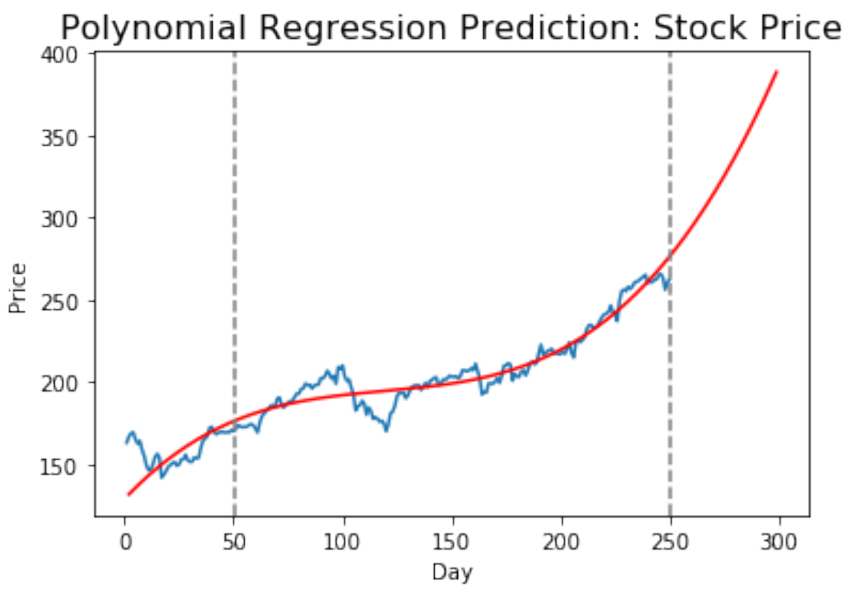
In [16]:
```
N = 5  ### Input your value

D = data_matrix([250+N], degree)
new_price = np.dot(D, sol_b)[0]
print("The stock price for the company you have chosen in " + str(N) +
" market days is: $" + str(round(new_price, 2)))
```

```
The stock price for the company you have chosen in 5 market days is:
$285.01
```

Let's extend our plot to predict the price for the upcoming 50 days!

In [17]:
```
fig = plt.figure()
ax = fig.add_subplot()
x_curve, y_curve = poly_curve(sol_b, dates_p)
ax.plot(dates, prices)
ax.plot(x_curve, y_curve, 'r')
plt.axvline(day_change, color='gray', linestyle='--')
plt.axvline(250, color='gray', linestyle='--')
plt.title('Polynomial Regression Prediction: Stock Price', fontsize=16
)
plt.xlabel('Day')
plt.ylabel('Price')
```

Out[17]:
```
Text(0, 0.5, 'Price')
```



Feel free to go back to stage 1 to test out models of different degrees to see their effectiveness!

Analysis

What can you conclude about the advantages/disadvantages of a polynomial regression model? Do you think that it is effective?

Advantages of using Polynomial Regression:

- Can fit many type of curves
- Provide the best approximation of the relationship between the dependent and independent variables

Disadvantages of using Polynomial Regression:

- Too sensitive to outliers (presence of outliers seriously distort the results of a nonlinear analysis)
- Tend to diverge very quickly, especially in higher-degree polynomials

Conclusion

As we have seen, linear regression and polynomial regression produced drastically different results towards predicting stock market prices. Although polynomial regression fits the data better and may perform better in the short run, linear regression generally performs better in the long run because it is less likely to overfit the training data. While each has their own advantages and disadvantages, it turns out that neither of these models are good enough to model the price change. Stock market is influenced by many, many factors, including economic growth, interest rates, confidence/expectations, and more.

As you embark on the journey of taking EECS 16B and upper division machine learning courses (e.g. CS 189, 182), you will learn more and more effective techniques, and gradually build more robust models towards elusive goals such as predicting the stock market. Hope this demo inspired you and gave you some insight of machine learning in the real world!

Extension: Other Machine Learning Techniques

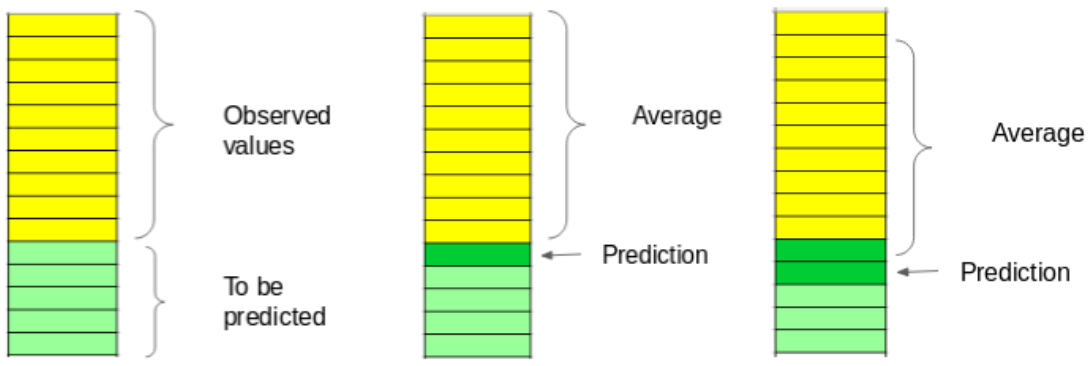
What are some other machine learning techniques that can model the stock market price change (effectively or ineffectively)?

NOTE: They are out of scope for EECS 16A and are shown below purely for personal interest!

Moving Average:

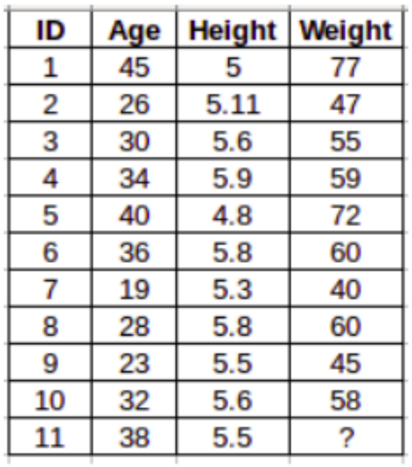
"Average" is commonly used in our daily lives, whether for calculating the average marks to determine the overall performance in a course or finding the average temperature in the past few days to get an idea of today's temperature. This method, even simpler than linear regression, is a great starting point for making predictions.

In the stock market model, the predicted closing price would just be the average of previously observed values. The "moving" average would use the latest set of values for prediction; for each subsequent step, the predicted values are taken into consideration while removing the oldest observed value from the set.

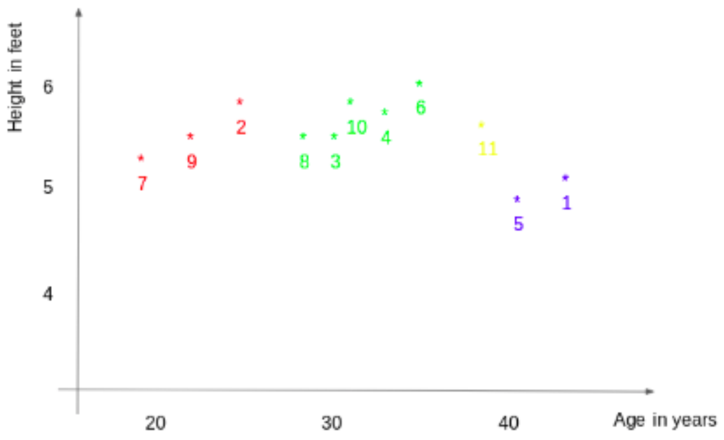


k-Nearest Neighbours

Based on independent variables, kNN finds the similarity between new data points and old data points. In the below example, there are sets of age and height for 10 people. To determine the weight for #11, we consider the weight of the nearest neighbours of this ID; the weight of #11 is predicted to be the average of its three nearest neighbours (ID 1, 5, 6): (77+72+60)/3 = 69.66 kg.



ID	Age	Height	Weight
1	45	5	77
2	26	5.11	47
3	30	5.6	55
4	34	5.9	59
5	40	4.8	72
6	36	5.8	60
7	19	5.3	40
8	28	5.8	60
9	23	5.5	45
10	32	5.6	58
11	38	5.5	?



The Prophet Forecasting Model

A decomposable time series model with three main model components: trend, seasonality, and holidays. They are combined in the following equation:

$$y(t) = g(t) + s(t) + h(t) + \epsilon_t$$

g(t): piecewise linear or logistic growth curve for modelling non-periodic changes in time series
s(t): periodic changes (e.g. weekly/yearly seasonality)
h(t): effects of holidays (user provided) with irregular schedules
ϵ_t : error term accounts for any unusual changes not accommodated by the model

Long Term Short Memory (LTSM)

A widely used sequence prediction method that is proven to be extremely effective. It is able to store past information that is important, and forget the information that is not. LSTM has three gates:

The input gate: The input gate adds information to the cell state
The forget gate: It removes the information that is no longer required by the model
The output gate: Output Gate at LSTM selects the information to be shown as output

Credit/I recommend further reading here: https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/ (https://www.analyticsvidhya.com/blog/2018/10/predicting-stock-price-machine-learningnd-deep-learning-techniques-python/)

Credit

Special thanks to Ricky Liou, Phoebe Li and Grace Chen for proofreading this demo!