# linear_regression

April 16, 2019

Gahan Saraiya (18MCEC10)
AIM: Implementation of Linear Regression
Linear Regression

- Implementation with statastical Method
- Implementation with Gradient Descent Method

```
In [1]: import pandas as pd
        # Setting up inline plotting using jupyter notebook "magic"
        # %matplotlib notebook

        import matplotlib.pyplot as plt
        import numpy as np
        # Read below thread for plotting guides
        # https://www.oreilly.com/library/view/python-data-science/9781491912126/ch04.html
```

## 1 Statastical Regression

```
In [2]: class StatasticalRegression(object):
            def __init__(self, dataframe_path="data.txt"):
                self.dataframe = pd.read_csv(dataframe_path, header=None)
                self.alpha, self.beta = None, None

            def compute_constant(self):
                x_mean, y_mean = self.dataframe.mean()
        #         print("Mean X: {}\nMean Y: {}".format(x_mean, y_mean))
                self.dataframe['input_diff'] = self.dataframe[0] - x_mean  # mean value differ
                self.dataframe['output_diff'] = self.dataframe[1] - y_mean  # mean value diffe
                self.dataframe['sqrd_input_diff'] = self.dataframe.input_diff ** 2  # square o
                self.dataframe['mul_input_output_diff'] = self.dataframe.input_diff * self.data
        #         print("Result: \n{}".format(self.dataframe))
                self.beta = sum(self.dataframe.mul_input_output_diff)/ sum(self.dataframe.sqrd_
                self.alpha = y_mean - self.beta*x_mean
                return self.alpha, self.beta

            def hypothesis(self, feature, alpha=None, beta=None):
                if not (alpha and beta):
```

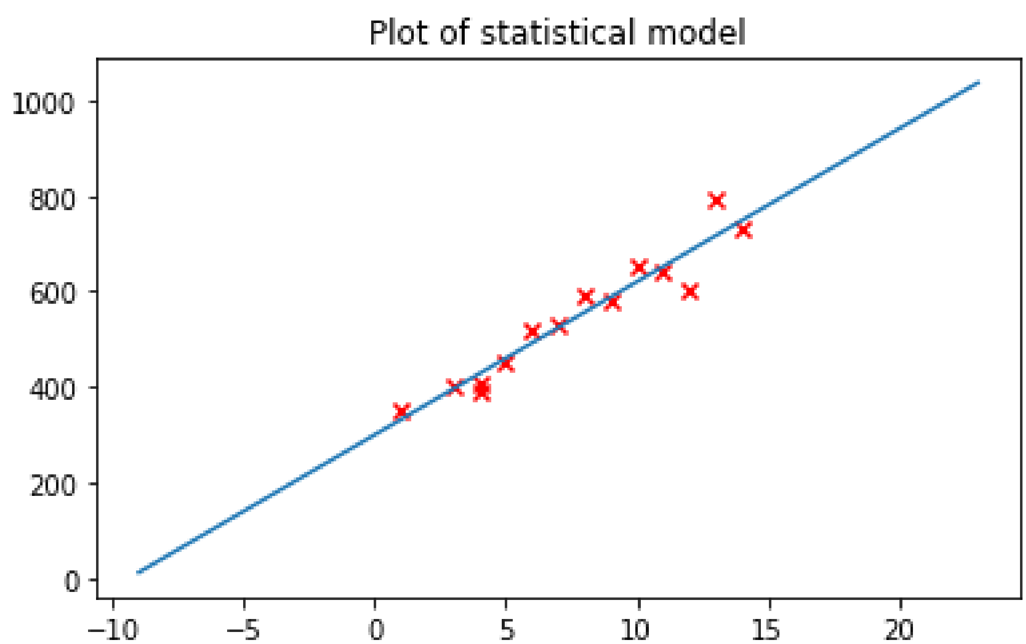
```
            alpha, beta = self.compute_constant()
        return (alpha + beta*feature)

    def add_predicted_values(self):
        self.dataframe['prediction'] = self.hypothesis(self.dataframe[0], alpha, beta)

    def draw_accuracy_plot(self):
        fig = plt.figure().add_axes((0.1, 0.2, 0.8, 0.7))
        x = range(min(self.dataframe[0]) - 10, max(self.dataframe[0]) + 10)
        fig.plot(x, [self.hypothesis(i, alpha, beta) for i in x])
        fig.scatter(self.dataframe[0], self.dataframe[1], marker='x', c="red")
        fig.set_title("Plot of statistical model")

s = StatasticalRegression()
s.dataframe.head()
```

Out[2]:

| | 0 | 1 |
|---|---|---|
| 0 | 4 | 390 |
| 1 | 9 | 580 |
| 2 | 10 | 650 |
| 3 | 14 | 730 |
| 4 | 4 | 410 |

### 1.0.1 Plot Prediction Plot for Statastical Regression

In [3]:
```
alpha, beta = s.compute_constant()
s.add_predicted_values()
s.draw_accuracy_plot()
```



## 2 Linear Regression (Gradient Descent Method)

In [4]:
```
class LinearRegression(object):
    def __init__(self, dataframe_path="data.csv", initial_theta=np.array([1,1]), learni
        self.df = pd.read_csv(dataframe_path)
        self.dataframe = pd.read_csv(dataframe_path)  # import directory
        self.actual_output = self.dataframe['y']
        self.dataframe.insert(0, 'bias', 1)  # add bias value x_0
        self.dataframe = self.dataframe.drop(['y'], axis=1)
        self.theta = initial_theta
        self.learning_rate = learning_rate
        self.hypothesis = None
        self.traing_set_length = len(self.actual_output)

    def hypothesis(self, feature_matrix, theta):
        return feature_matrix.dot(theta)

    def compute_cost(self, theta):
        sqrErrors = (self.dataframe.dot(theta) - self.actual_output) ** 2
        cost = (1/2*self.traing_set_length) * sum(sqrErrors)
        return cost

    def gradient_descent(self, theta, no_of_iterations=1000, log=False):
        cost_history = np.zeros(no_of_iterations)
        theta_history = np.zeros((no_of_iterations, 2))
        prediction_history = []

        for i in range(no_of_iterations):
            # prediction = self.hypothesis(self.dataframe, theta)
            prediction = self.dataframe.dot(theta)
            error = self.dataframe.T.dot((prediction - self.actual_output))
            theta -= (1/self.traing_set_length) * self.learning_rate * error
            theta_history[i,:] = theta.T
            cost_history[i]  = self.compute_cost(theta)
            prediction_history.append(prediction)
            if log:
                print("{}. theta -> {}\tcost -> {}".format(i, theta_history[i], cost_hi

        return theta, cost_history, theta_history, prediction_history

    def draw_accuracy_plot(self):
        fig = plt.figure().add_axes((0.1, 0.2, 0.8, 0.7))
        x = range(min(self.dataframe[0]) - 10, max(self.dataframe[0]) + 10)
#         fig.plot(x, [self.hypothesis(i, alpha, beta) for i in x])
```

```
            fig.scatter(self.dataframe[0], self.actual_output['y'], marker='x', c="red")
            fig.set_title("Plot of statistical model")

        def hypothesis_plot(self, iteration_number, predictions):
            plt.scatter(self.dataframe.feature1, self.df.y)
            plt.xlabel('feature1')
            plt.ylabel('actual output')
            plt.plot(self.dataframe.feature1, predictions[iteration_number], color='r')
            plt.show()

    l = LinearRegression(learning_rate=0.001)
    no_of_iterations = 10**2
    l.df.head()
```

Out[4]:

| | feature1 | y |
|---|---|---|
| 0 | 4 | 390 |
| 1 | 9 | 580 |
| 2 | 10 | 650 |
| 3 | 14 | 730 |
| 4 | 4 | 410 |

```
In [5]: l.dataframe.head()
```

Out[5]:

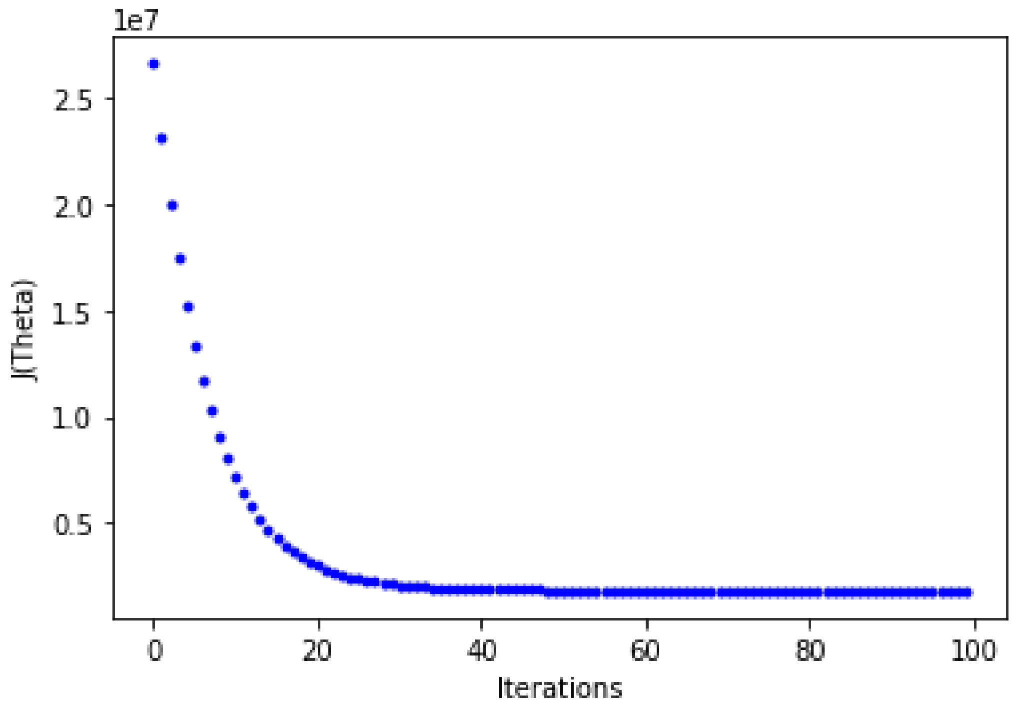
| | bias | feature1 |
|---|---|---|
| 0 | 1 | 4 |
| 1 | 1 | 9 |
| 2 | 1 | 10 |
| 3 | 1 | 14 |
| 4 | 1 | 4 |

# 3 Calculate and plot Cost Function J(theta)

```
In [6]: theta, costs, thetas, predictions = l.gradient_descent(theta=np.array((0, 0)), no_of_i
```

## 3.1 Plot cost function values over iteration

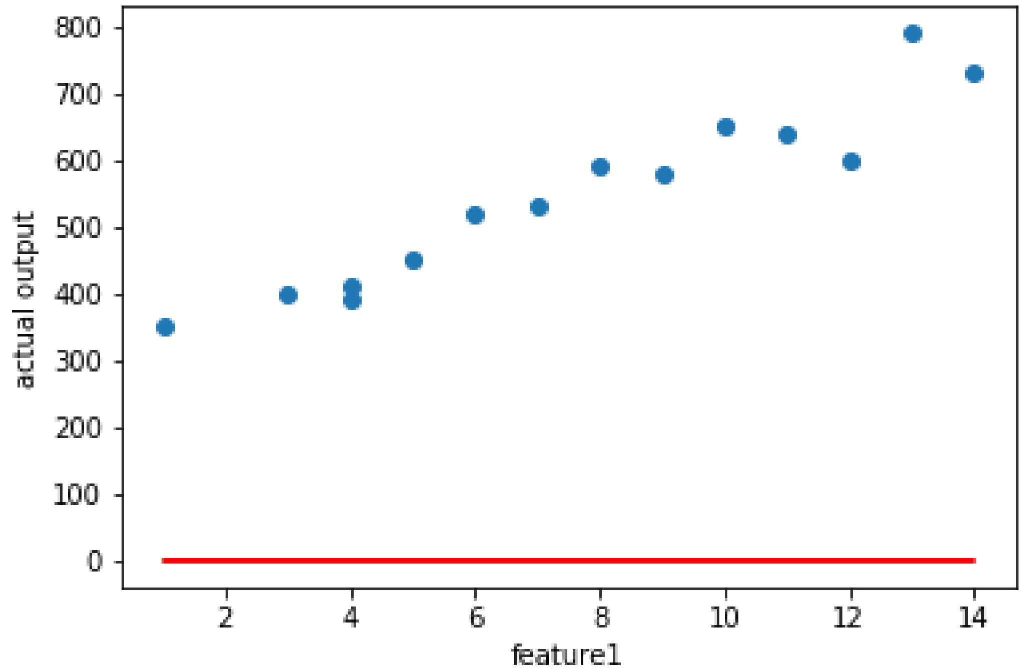
```
In [7]: fig,ax = plt.subplots()  # optional parameter figsize=(12,8)
        ax.set_ylabel('J(Theta)')
        ax.set_xlabel('Iterations')
        jThetaPlot = ax.plot(range(no_of_iterations), costs[:no_of_iterations], 'b.')

        # thetas = pd.DataFrame(thetas)
        # features = l.df.feature1
        # features.head()
```




# 4 Plotting hypothesis

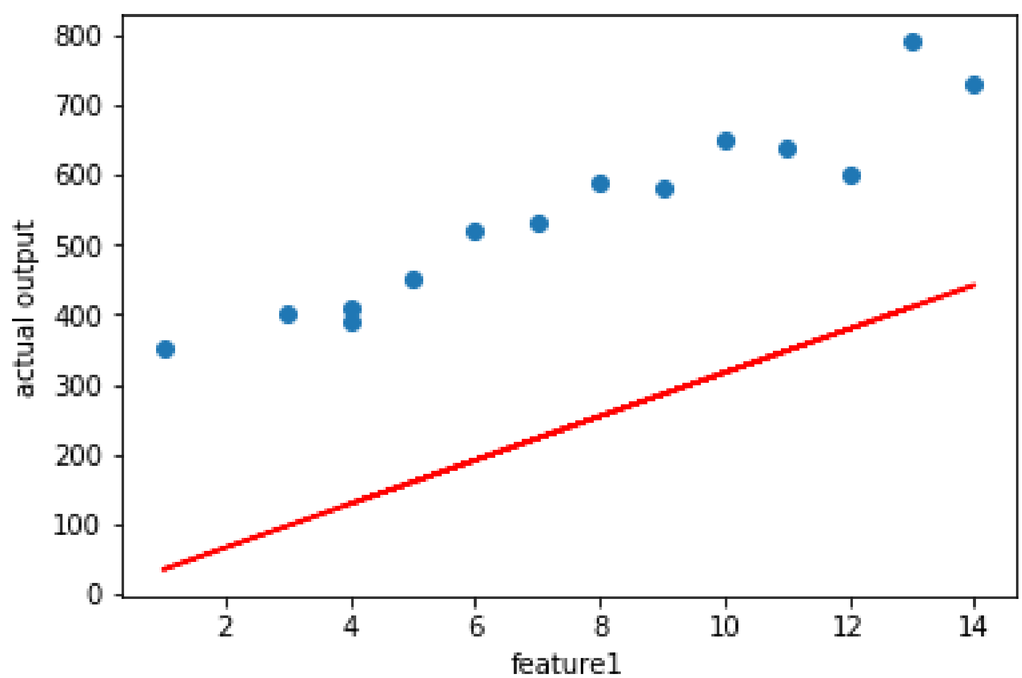
### 4.0.1 Initial Plot

```
In [8]: l.hypothesis_plot(0, predictions)
```



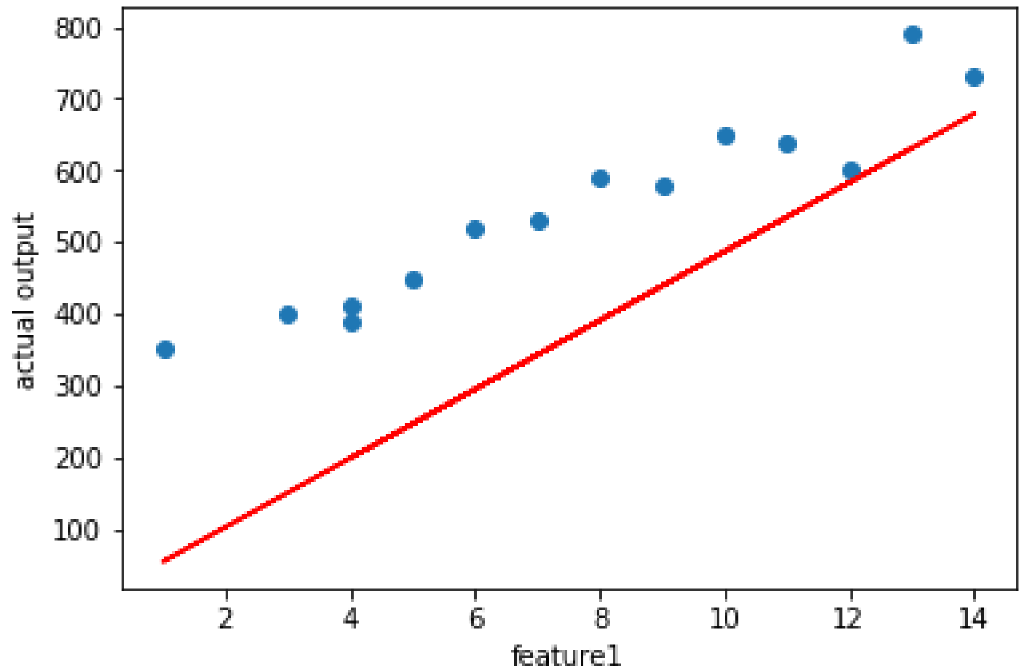

#### 4.0.2 Hypothesis after 10 iteration

```
In [9]: l.hypothesis_plot(9, predictions)
```



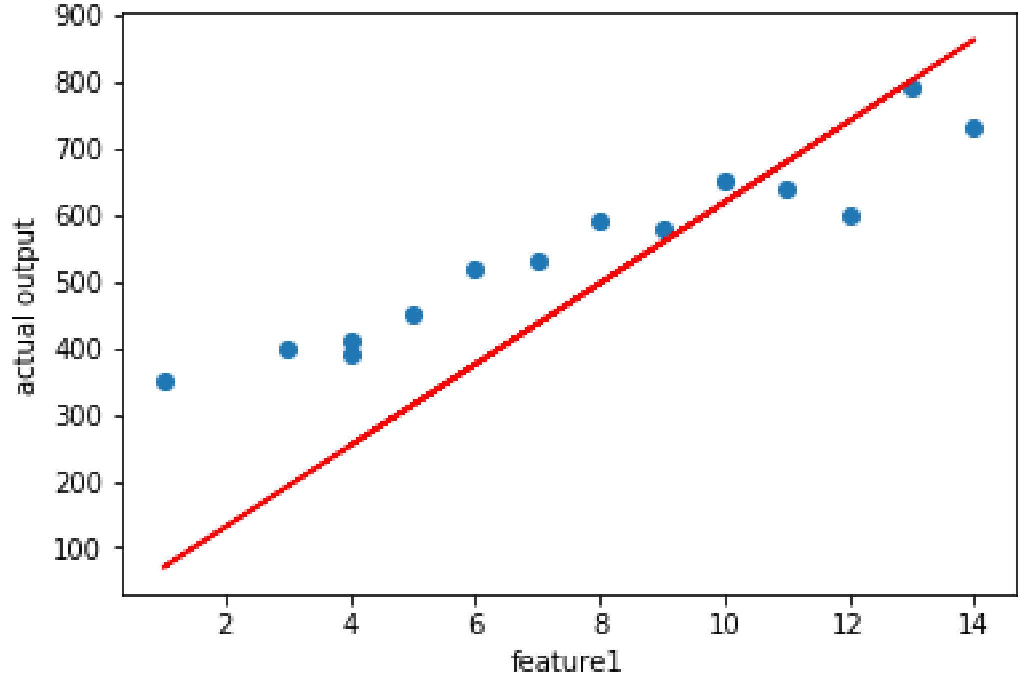
#### 4.0.3 Hypothesis after 20 iteration

```
In [10]: l.hypothesis_plot(19, predictions)
```



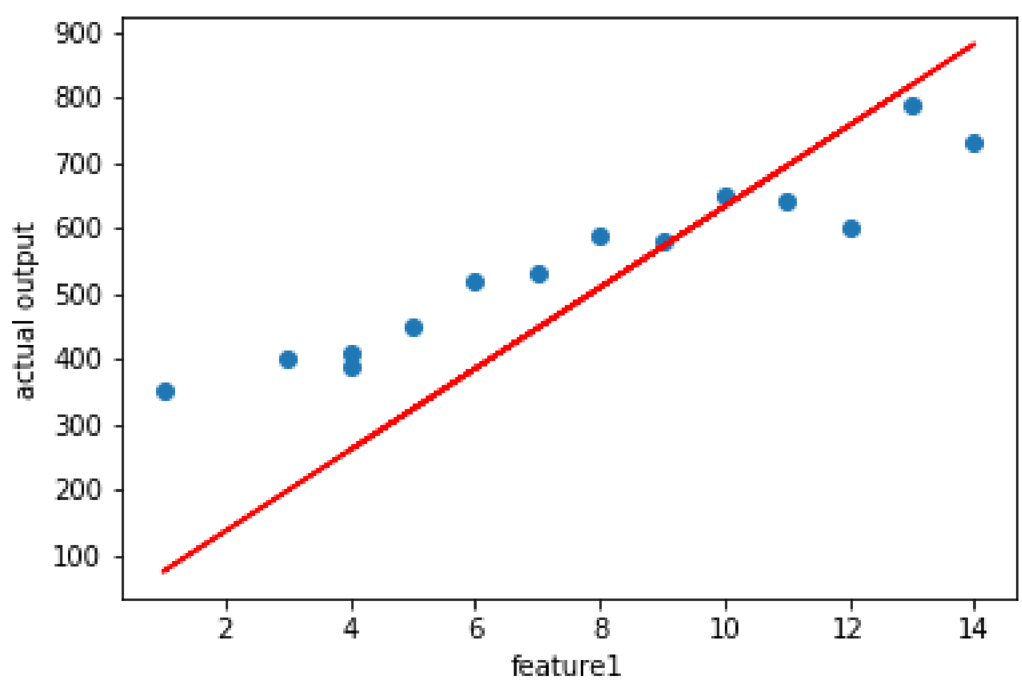

#### 4.0.4 Hypothesis after 50 iteration

```
In [11]: ### Hypothesis after 10 iteration

In [12]: l.hypothesis_plot(49, predictions)
```




#### 4.0.5 Hypothesis after 100 iteration

```
In [13]: l.hypothesis_plot(99, predictions)
```



```
In [ ]:
```