CS306: DATA ANALYSIS AND VISUALIZATION

LAB 4: Airline Data (Big) Regression Analysis

STUDENT ID: 201801407

NAME: PRATVI SHAH

In [68]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.lines as mlines
%matplotlib inline
from scipy import stats
import math
import seaborn as sns
from sklearn import preprocessing
from scipy import stats
from scipy.stats import norm
from scipy.stats import iqr
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
```

In [48]:
```
#download = drive.CreateFile({'id': '1vawuuoBZitiCrLjOaLAqYY5K5j4Tjk6n'})
#download.GetContentFile('2008.csv.bz2')
df=pd.read_csv('2008.csv.bz2')
```

In [49]:
```
data=df[['AirTime','Distance']]
```

Removing null values

In [50]:
```
data=data.dropna()

for i in data.columns:
    data=data[data['{}'.format(i)].notnull()]
row_count=data.shape[0]
print(data.head())
```

```
   AirTime  Distance
0    116.0       810
1    113.0       810
2     76.0       515
3     78.0       515
4     77.0       515
```

Taking Distance as X and Airtime as Y

In [51]:
```
X=data['Distance']
Y=data['AirTime']
```

Q1

Pearson Correlation Coefficient

In [52]:
```
def pearson_coeff(x,y):
    N=(np.mean(x*y) - np.mean(x)*np.mean(y))
    D1=np.sqrt(np.mean(x*x) - np.mean(x)*np.mean(x))
    D2=np.sqrt(np.mean(y*y) - np.mean(y)*np.mean(y))
    D=D1*D2
    r=N/D
    return r;
```

In [53]:
```
pearson_corr_coeff=pearson_coeff(np.array(X),np.array(Y))
print('Calculated Pearson Correlation coefficient: ',pearson_corr_coeff)
r=stats.pearsonr(X,Y)
print('Inbuilt: ',r[0])
```

```
Calculated Pearson Correlation coefficient:  0.9828758232165179
Inbuilt:  0.9828758232165375
```

Finding intercept c=($\beta 1$) and slope m=($\beta 2$)

In [54]:
```
def get_constants(x,y):
    m = (np.mean(x) * np.mean(y) - np.mean(x*y) ) / (np.mean(x) * np.mean(x) - np.mean(x*x))
    b = np.mean(y) - (np.mean(x) * m)
    return b,m
```

In [55]:
```
x=np.array(X).reshape([-1,1]).transpose()[0]
y=np.array(Y)
c,m=get_constants(x,y)
print('Intercept of regression line: ',c)
print('Slope of regression line: ',m)
ypred= m*x + c
```

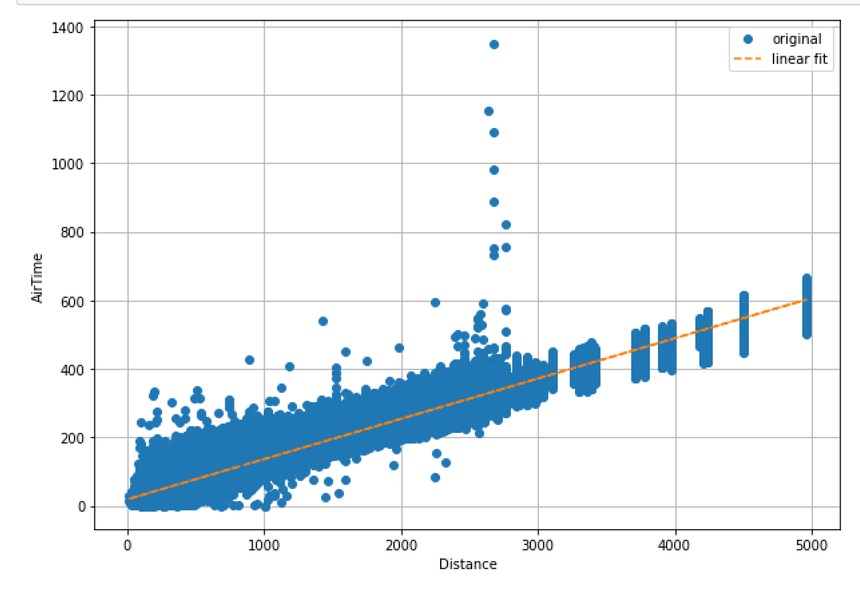
```
Intercept of regression line:  18.257032395292754
Slope of regression line:  0.1176840924861962
```

Calculating RMSE of y and y_predicted

In [56]:
```
rmse= np.sqrt(mean_squared_error(y, ypred))
print('RMSE :', rmse)
```

```
RMSE : 12.427072327524876
```

In [57]:
```
#initial predicted fit
plt.figure(figsize=[10,7])
plt.plot(x,y,'o',label='original')
plt.plot(x,ypred,'--',label='linear fit')
plt.xlabel('Distance')
plt.ylabel('AirTime')
plt.grid(True)
plt.legend()
plt.show()
```



Q2

a) 95% confidence interval for slope using t-test method:

$$\hat{X} = \bar{X} \pm t_{\alpha/2,n-2} SE$$

SE= Standard Error = $\frac{S_y}{\sqrt{n} S_x}$

$\alpha = 0.05$

n = length of X

S_i = Standard Deviation of i

As n is very large we can approximate $t_{\alpha/2,n-2} \approx z$ of $95\% = 1.96$

In [58]:
```
sx=np.sqrt(np.sum((x-np.mean(x))**2))
l_interval=m - 1.96*np.std(y)/sx
r_interval=m + 1.96*np.std(y)/sx
print('\n95% confidence interval for slope: ['+str(l_interval)+','+str(r_interval)+']')
slope, intercept, r_value, p_value, std_err = stats.linregress(np.array(X),np.array(Y))
l_interval=m - 1.96*std_err
r_interval=m + 1.96*std_err
print('\nUsing inbuilt function 95% confidence interval for slope: ['+str(l_interval)+','+st
r(r_interval)+']')
```

```
95% confidence interval for slope: [0.11759445894938748,0.11777372602300491]

Using inbuilt function 95% confidence interval for slope: [0.11766757578900006,0.117700609183
39234]
```

b) Mean of y_0 when x_0=1200

$$\hat{Y} = \tilde{Y} \pm t_{\alpha/2,n-2} SE$$

$\alpha = 0.05$

$x_h = 1200$

$\tilde{Y}$ = predicted value using slope and interval from previous question

$$SE = StandardError = \sqrt{MSE \times (\frac{1}{n} + \frac{(x_h - \bar{x})^2}{\sum(x_i - \bar{x})^2})}$$

As n is very large we approximate $t_{\alpha/2,n-2} \approx z$ of $95\% = 1.96$

In [59]:
```
Xo=1200
yl=l_interval*Xo + c
yr=r_interval*Xo + c
n = len(x)
print('Mean of y0 calculated using the confidence interval obtained previously : ',(yl+yr)/2
)

s = np.sqrt(np.sum((y - ypred)**2)/(n - 2))
k = np.sqrt((1/n) + (Xo - np.mean(x))/(np.sum((x - np.mean(x))**2)))
y0=m*Xo + c
y_l=y0-1.96*s*k
y_r=y0+1.96*s*k
print('\n95% confidence interval of y: ['+str(y_l)+','+str(y_r)+']')
print('\nMean of y0 calculated using the confidence interval of 95% on y : ',(y_l+y_r)/2)
```

```
Mean of y0 calculated using the confidence interval obtained previously :  159.47794337872818

95% confidence interval of y: [159.46863352970556,159.4872532277508]

Mean of y0 calculated using the confidence interval of 95% on y :  159.47794337872818
```

Q3

In [61]:
```
def my_wls(x,y,ypred,lim,R,m):
    err=1
    runs=0
    m_old=0
    rmse2=0

    while (err>lim and runs<R):
        m_old=m             #initial slope
        d=y-ypred           #di = difference between original and predicted
        ui=d/(iqr(d)*3)     #IQR of di as mentioned in pdf
        w=np.zeros(len(y))  #weights
        for i in range(len(ui)):
            if ui[i]<1:
                w[i]=(1-ui[i]*ui[i])**2
            else:
                w[i]=0
        x_wls=[]
        y_wls=[]
        for i in range(len(x)):
            x_wls.append(x[i]*np.sqrt(w[i]))
            y_wls.append(y[i]*np.sqrt(w[i]))

        #getting parameter for new weights
        c,m=get_constants(np.array(x_wls),np.array(y_wls))

        #new predicted values for original x
        ypred=m*x + c

        #new predicted y for weighted x
        y_wsl_pred=m*np.array(x_wls) + c

        #RMSE with respect to weighted y
        rmse2=np.sqrt(mean_squared_error(y_wls, y_wsl_pred))

        #Measuring error wrt the slope
        err=np.abs(m_old-m)
        print(m,c,rmse2)
        runs+=1
    return c,m,x_wls,y_wls
```

In [62]:
```
#value printed will be slope , intercept  , RMSE for weighted case


c_wls,m_wls,x_wls,y_wls=my_wls(x,y,ypred,1e-6,50,m)
```

```
0.11528177441627069 17.078704784602294 9.058507826844439
0.11428223657056451 17.16162999294626 8.168294223394552
0.11376797345757363 17.333371749454543 7.995930072290792
0.11352397236016248 17.438702406891437 7.942130506596651
0.11341199315370075 17.498382637886266 7.922912380658683
0.1133640513772611 17.52332920662188 7.916644905156864
0.11334302788527148 17.53588732557482 7.913899914007562
0.11333453070116262 17.539685454717855 7.913020764218985
0.11333058797818968 17.54191275150066 7.912423624896749
0.11332888475468635 17.543204449221534 7.912234756843423
0.11332824649040706 17.543873649658707 7.912215709180831
```

WLS intercept and slope

In [63]:
```
print('WLS intercept: ',c_wls)
print('WLS slope: ', m_wls)
```

```
WLS intercept:  17.543873649658707
WLS slope:  0.11332824649040706
```

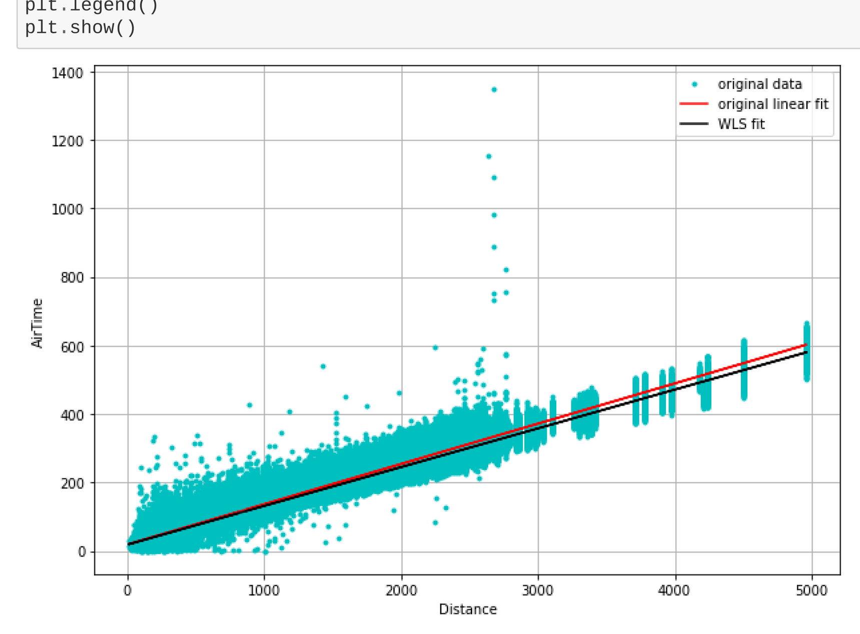
RMSE using new values

In [65]:
```
ypred2=c_wls + m_wls*np.array(x_wls)
rmse2=np.sqrt(mean_squared_error(y_wls, ypred2))
print('\nRMSE with weighted data:', rmse2)
ypred3=c_wls + x*m_wls
rmse3=np.sqrt(mean_squared_error(y, ypred3))
print('\nRMSE with original data:', rmse3)
```

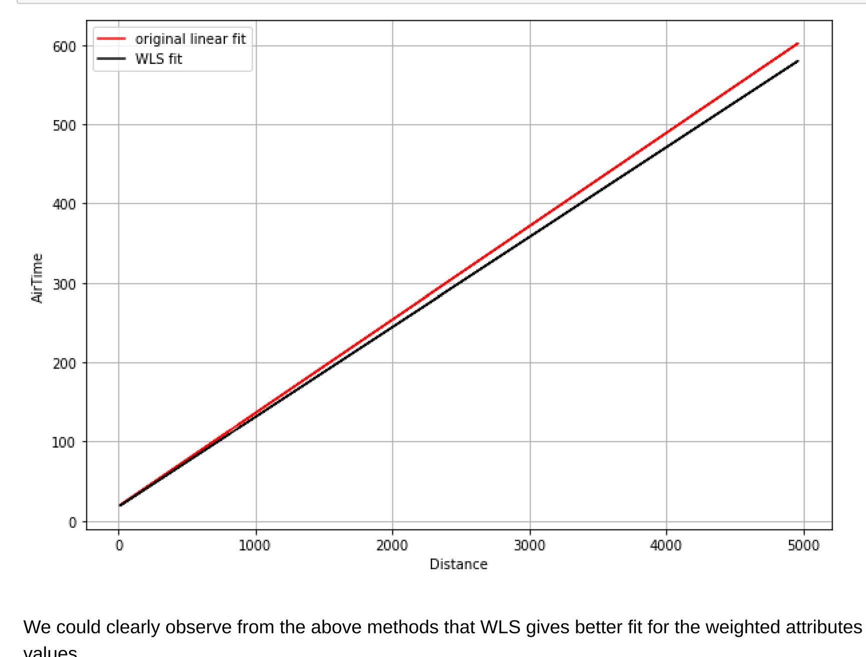
```
RMSE with weighted data: 7.912215709180831

RMSE with original data: 13.250042900185754
```

Comparing OLS fit with WLS fit

In [66]:
```
ypred_wls=c_wls + x*m_wls
plt.figure(figsize=[10,7])
plt.plot(x,y,'c.',label='original data')
plt.plot(x,ypred,'r-',label='original linear fit')
plt.plot(x,ypred_wls,'k-',label='WLS fit')
plt.xlabel('Distance')
plt.ylabel('AirTime')
plt.grid(True)
plt.legend()
plt.show()
```



In [67]:
```
plt.figure(figsize=[10,7])
plt.plot(x,ypred,'r-',label='original linear fit')
plt.plot(x,ypred_wls,'k-',label='WLS fit')
plt.xlabel('Distance')
plt.ylabel('AirTime')
plt.grid(True)
plt.legend()
plt.show()
```



We could clearly observe from the above methods that WLS gives better fit for the weighted attributes and not the original values.

RMSE with weighted data: 7.912215709180831

RMSE with original data: 13.250042900185754

The fit obtained in Q1 is the best fit with all the outliers while the effect of outliers is nullified in the WLS method. This nullifying effect is what the weighted attributes represent.

We can also verify that any WLS fit other than the one obtained in Q1 will give worse fit to the original data as the Q1 fit has slope and intercept values which were obtained by minimizing the MSE of the predicted linear fit and the original data.

From this we can conclude that WLS gives us a better fit for data in which outliers beyond 3IQR are removed. In WLS even when the outliers were present in the data they did not contribute to the fit suggesting the robustness of the WLS model.

In [ ]: