# Neural Networks & Deep Learning - ICP-4

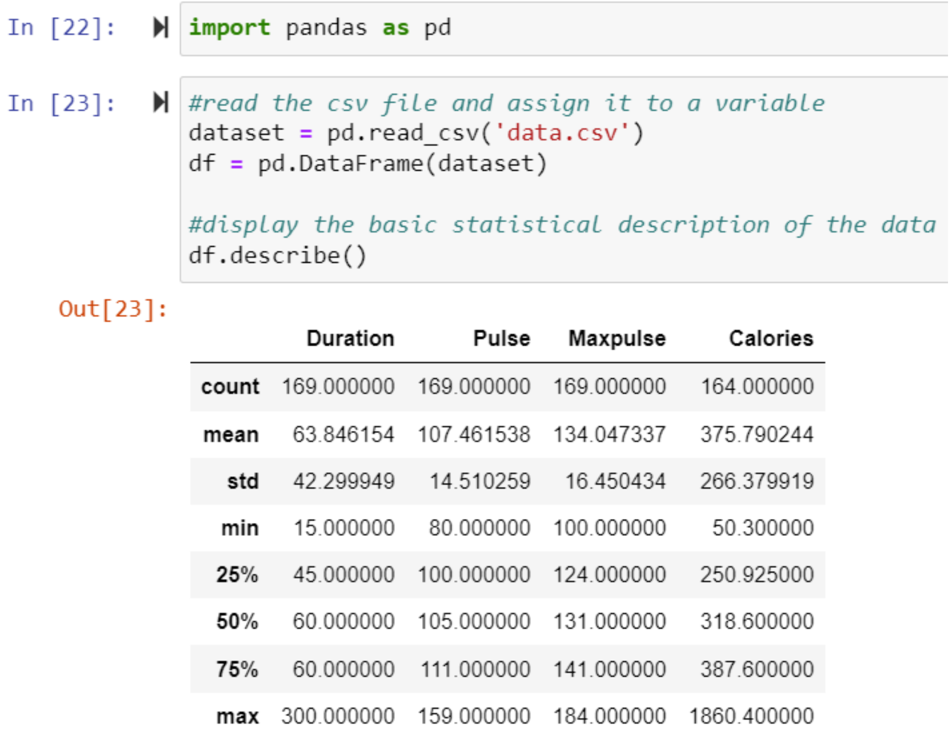
CS 5720 (CRN 23216)

Student ID: 700745451

Student Name: Kamala Ramesh

## Question 1:

- Import the pandas libraries. Read the csv file containing the data sets and display the basic statistical descriptions of the dataset.



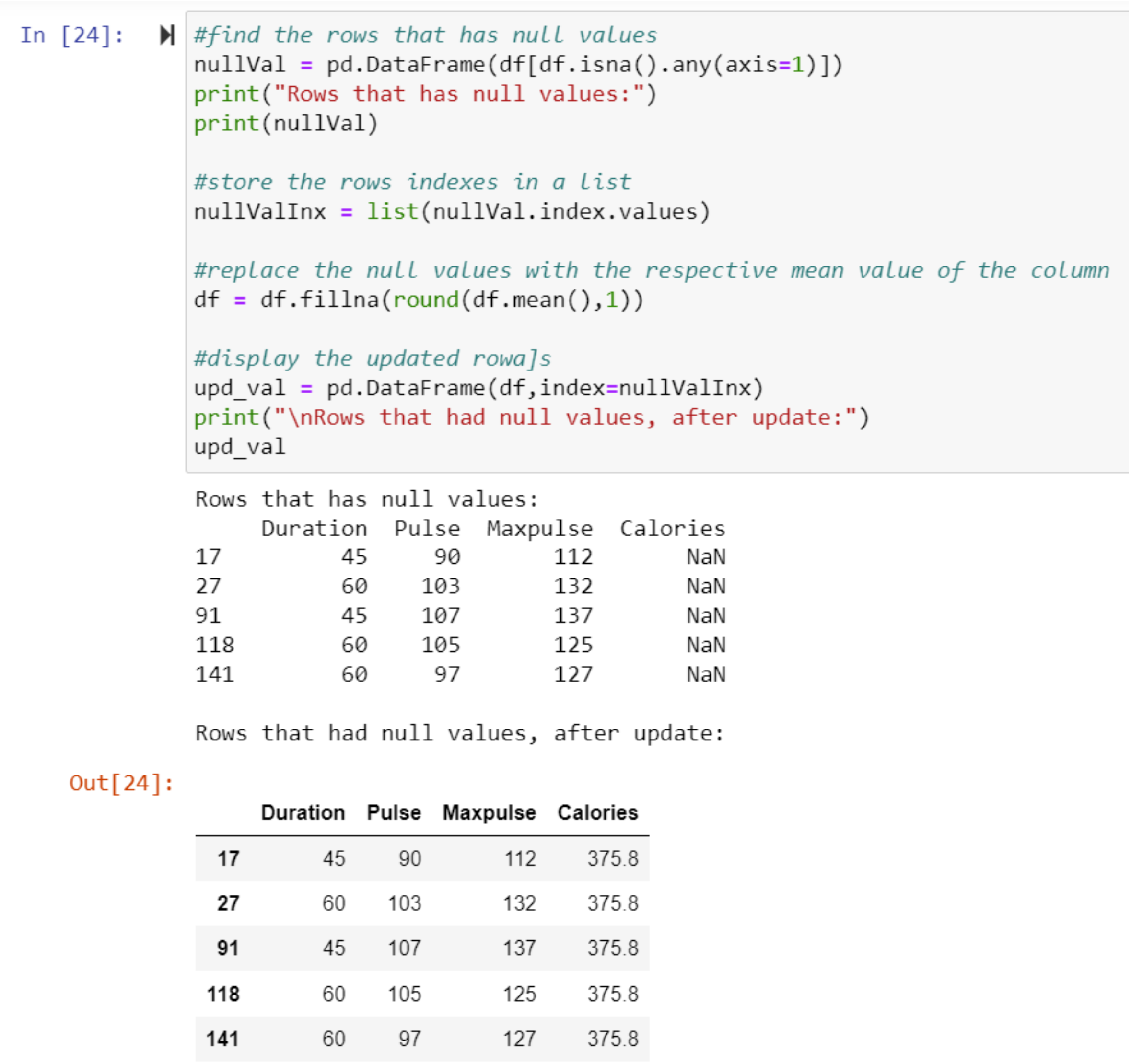
```
In [22]:  import pandas as pd
```

```
In [23]:  #read the csv file and assign it to a variable
          dataset = pd.read_csv('data.csv')
          df = pd.DataFrame(dataset)

          #display the basic statistical description of the data
          df.describe()
```

Out[23]:

| | Duration | Pulse | Maxpulse | Calories |
|---|---|---|---|---|
| **count** | 169.000000 | 169.000000 | 169.000000 | 164.000000 |
| **mean** | 63.846154 | 107.461538 | 134.047337 | 375.790244 |
| **std** | 42.299949 | 14.510259 | 16.450434 | 266.379919 |
| **min** | 15.000000 | 80.000000 | 100.000000 | 50.300000 |
| **25%** | 45.000000 | 100.000000 | 124.000000 | 250.925000 |
| **50%** | 60.000000 | 105.000000 | 131.000000 | 318.600000 |
| **75%** | 60.000000 | 111.000000 | 141.000000 | 387.600000 |
| **max** | 300.000000 | 159.000000 | 184.000000 | 1860.400000 |

- Get the rows that has null values in any of their column values, copy their indexes into a list. Here we are storing the indexes just to see how our data looks after updating the null values. Update all the null values with the respective mean value. Now using the row indexes we can see how the rows look after updating.



In [24]:

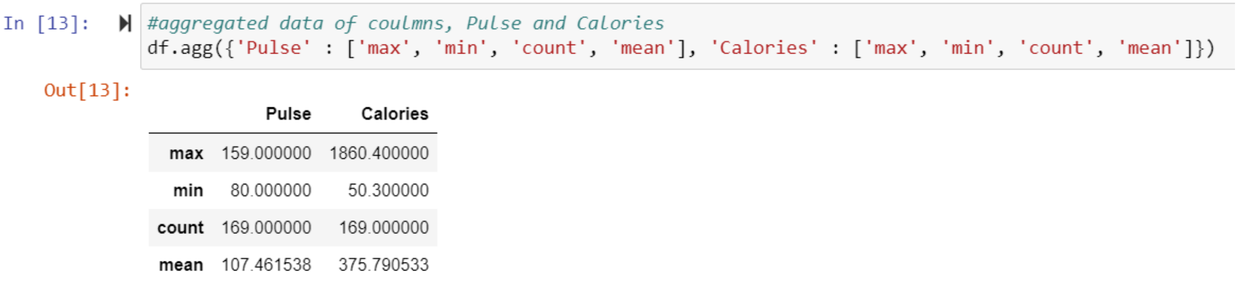
```
#find the rows that has null values
nullVal = pd.DataFrame(df[df.isna().any(axis=1)])
print("Rows that has null values:")
print(nullVal)

#store the rows indexes in a list
nullValInx = list(nullVal.index.values)

#replace the null values with the respective mean value of the column
df = df.fillna(round(df.mean(),1))

#display the updated rowa]s
upd_val = pd.DataFrame(df,index=nullValInx)
print("\nRows that had null values, after update:")
upd_val
```

```
Rows that has null values:
     Duration  Pulse  Maxpulse  Calories
17         45     90       112       NaN
27         60    103       132       NaN
91         45    107       137       NaN
118        60    105       125       NaN
141        60     97       127       NaN

Rows that had null values, after update:
```

Out[24]:

| | Duration | Pulse | Maxpulse | Calories |
|---|---|---|---|---|
| **17** | 45 | 90 | 112 | 375.8 |
| **27** | 60 | 103 | 132 | 375.8 |
| **91** | 45 | 107 | 137 | 375.8 |
| **118** | 60 | 105 | 125 | 375.8 |
| **141** | 60 | 97 | 127 | 375.8 |

- Here two columns, Pulse and Calories are selected and their respective max value, min value, count and mean are aggregated and displayed



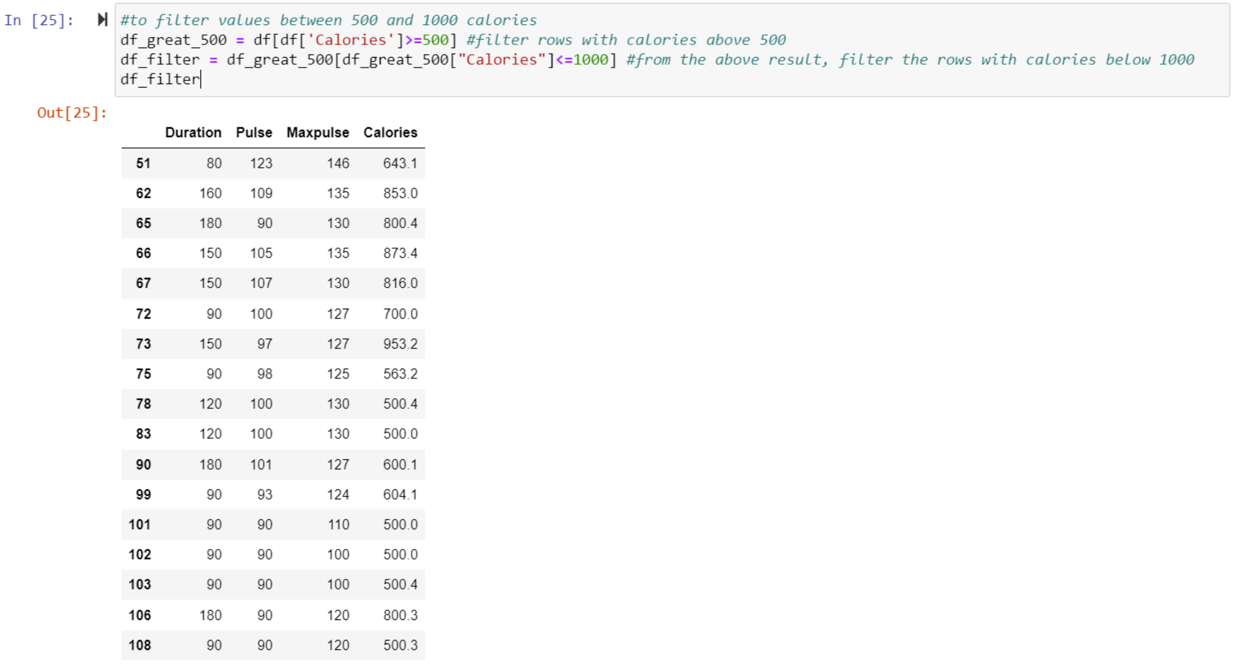
In [13]:

```
#aggregated data of coulmns, Pulse and Calories
df.agg({'Pulse' : ['max', 'min', 'count', 'mean'], 'Calories' : ['max', 'min', 'count', 'mean']})
```

Out[13]:

| | Pulse | Calories |
|---|---|---|
| **max** | 159.000000 | 1860.400000 |
| **min** | 80.000000 | 50.300000 |
| **count** | 169.000000 | 169.000000 |
| **mean** | 107.461538 | 375.790533 |

- Here we are displaying the rows whose Calories column values are between 500 and 1000. This is done in two steps. First we are filtering values greater than 500 and store it. Then filter the resulted data with values less than 1000.



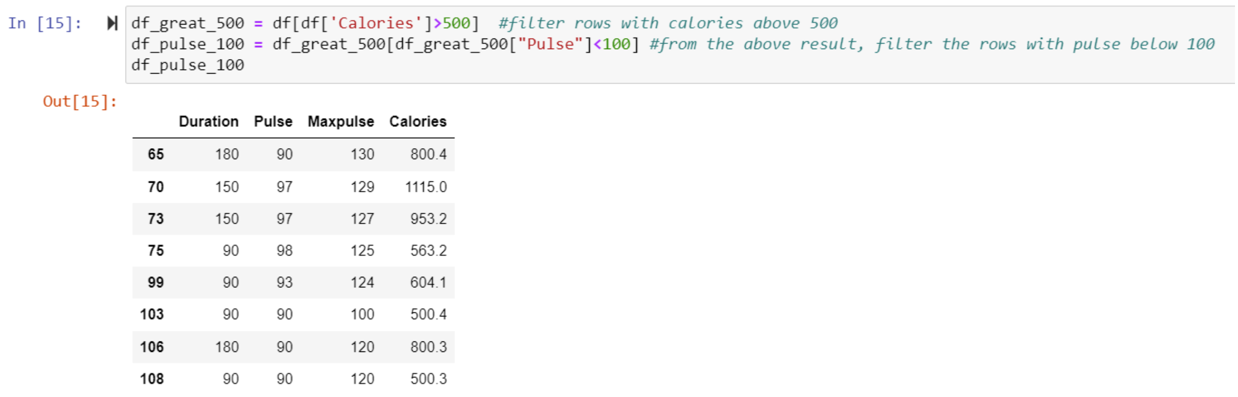
In [25]:

```
#to filter values between 500 and 1000 calories
df_great_500 = df[df['Calories']>=500] #filter rows with calories above 500
df_filter = df_great_500[df_great_500["Calories"]<=1000] #from the above result, filter the rows with calories below 1000
df_filter
```

Out[25]:

| | Duration | Pulse | Maxpulse | Calories |
|---|---|---|---|---|
| **51** | 80 | 123 | 146 | 643.1 |
| **62** | 160 | 109 | 135 | 853.0 |
| **65** | 180 | 90 | 130 | 800.4 |
| **66** | 150 | 105 | 135 | 873.4 |
| **67** | 150 | 107 | 130 | 816.0 |
| **72** | 90 | 100 | 127 | 700.0 |
| **73** | 150 | 97 | 127 | 953.2 |
| **75** | 90 | 98 | 125 | 563.2 |
| **78** | 120 | 100 | 130 | 500.4 |
| **83** | 120 | 100 | 130 | 500.0 |
| **90** | 180 | 101 | 127 | 600.1 |
| **99** | 90 | 93 | 124 | 604.1 |
| **101** | 90 | 90 | 110 | 500.0 |
| **102** | 90 | 90 | 100 | 500.0 |
| **103** | 90 | 90 | 100 | 500.4 |
| **106** | 180 | 90 | 120 | 800.3 |
| **108** | 90 | 90 | 120 | 500.3 |

- Here we are displaying the rows whose Calories column values are greater than 500 and Pulse column values are less than 100. This is done in two steps. First, we are filtering Calories values greater than 500 and store it. Then filter the resulted data with Pulse values less than 1000.

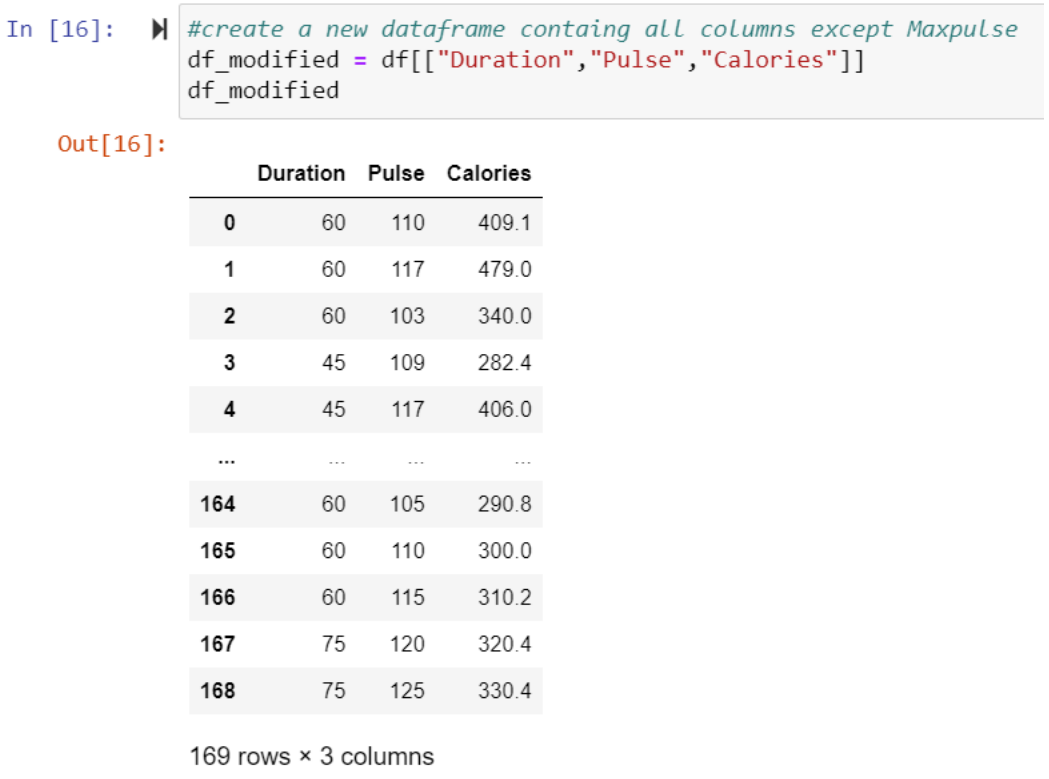


```
In [15]:  df_great_500 = df[df['Calories']>500]  #filter rows with calories above 500
          df_pulse_100 = df_great_500[df_great_500["Pulse"]<100] #from the above result, filter the rows with pulse below 100
          df_pulse_100
```

Out[15]:

| | Duration | Pulse | Maxpulse | Calories |
|---|---|---|---|---|
| **65** | 180 | 90 | 130 | 800.4 |
| **70** | 150 | 97 | 129 | 1115.0 |
| **73** | 150 | 97 | 127 | 953.2 |
| **75** | 90 | 98 | 125 | 563.2 |
| **99** | 90 | 93 | 124 | 604.1 |
| **103** | 90 | 90 | 100 | 500.4 |
| **106** | 180 | 90 | 120 | 800.3 |
| **108** | 90 | 90 | 120 | 500.3 |

- Creating a new data frame containing the all the columns except Maxpulse



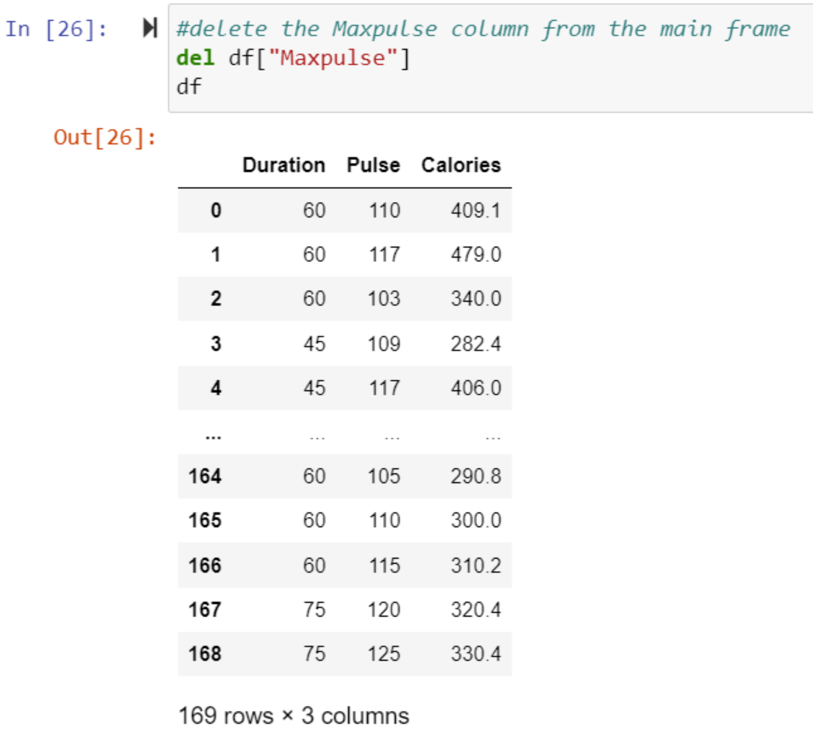
```
In [16]:  #create a new dataframe containg all columns except Maxpulse
          df_modified = df[["Duration","Pulse","Calories"]]
          df_modified
```

Out[16]:

| | Duration | Pulse | Calories |
|---|---|---|---|
| **0** | 60 | 110 | 409.1 |
| **1** | 60 | 117 | 479.0 |
| **2** | 60 | 103 | 340.0 |
| **3** | 45 | 109 | 282.4 |
| **4** | 45 | 117 | 406.0 |
| **...** | ... | ... | ... |
| **164** | 60 | 105 | 290.8 |
| **165** | 60 | 110 | 300.0 |
| **166** | 60 | 115 | 310.2 |
| **167** | 75 | 120 | 320.4 |
| **168** | 75 | 125 | 330.4 |

169 rows × 3 columns

- Delete the Maxpulse column from the main data frame.



In [26]:

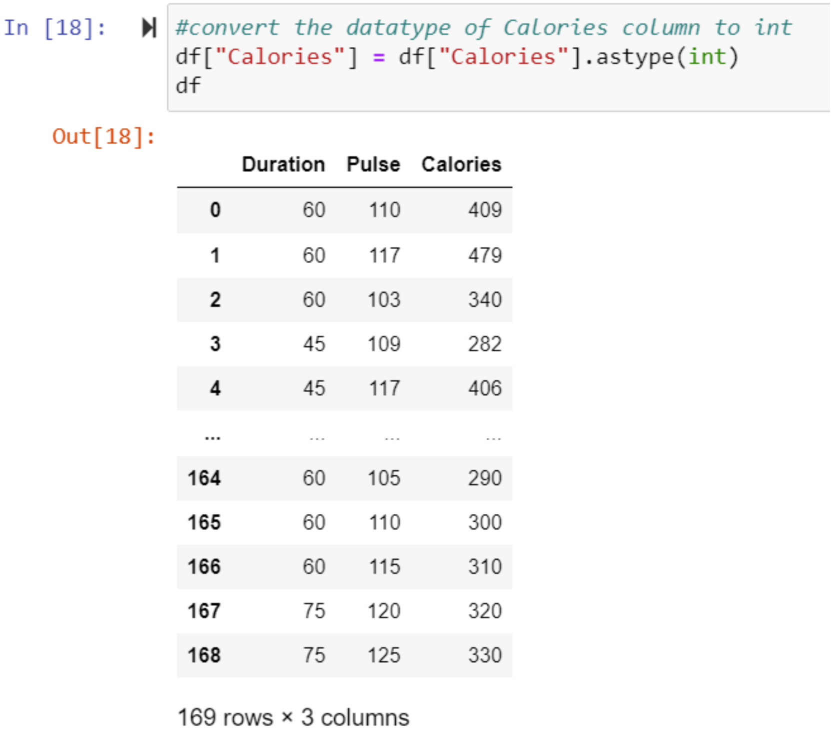
```
#delete the Maxpulse column from the main frame
del df["Maxpulse"]
df
```

Out[26]:

| | Duration | Pulse | Calories |
|---|---|---|---|
| **0** | 60 | 110 | 409.1 |
| **1** | 60 | 117 | 479.0 |
| **2** | 60 | 103 | 340.0 |
| **3** | 45 | 109 | 282.4 |
| **4** | 45 | 117 | 406.0 |
| **...** | ... | ... | ... |
| **164** | 60 | 105 | 290.8 |
| **165** | 60 | 110 | 300.0 |
| **166** | 60 | 115 | 310.2 |
| **167** | 75 | 120 | 320.4 |
| **168** | 75 | 125 | 330.4 |

169 rows × 3 columns

- Converting the datatype of Calories column from float to int.



In [18]:

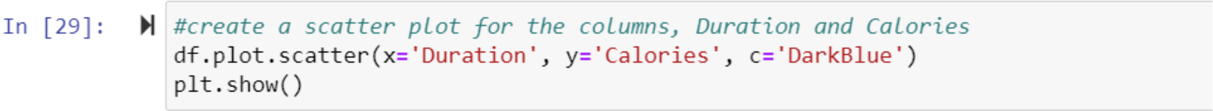
```
#convert the datatype of Calories column to int
df["Calories"] = df["Calories"].astype(int)
df
```

Out[18]:

| | Duration | Pulse | Calories |
|---|---|---|---|
| **0** | 60 | 110 | 409 |
| **1** | 60 | 117 | 479 |
| **2** | 60 | 103 | 340 |
| **3** | 45 | 109 | 282 |
| **4** | 45 | 117 | 406 |
| **...** | ... | ... | ... |
| **164** | 60 | 105 | 290 |
| **165** | 60 | 110 | 300 |
| **166** | 60 | 115 | 310 |
| **167** | 75 | 120 | 320 |
| **168** | 75 | 125 | 330 |

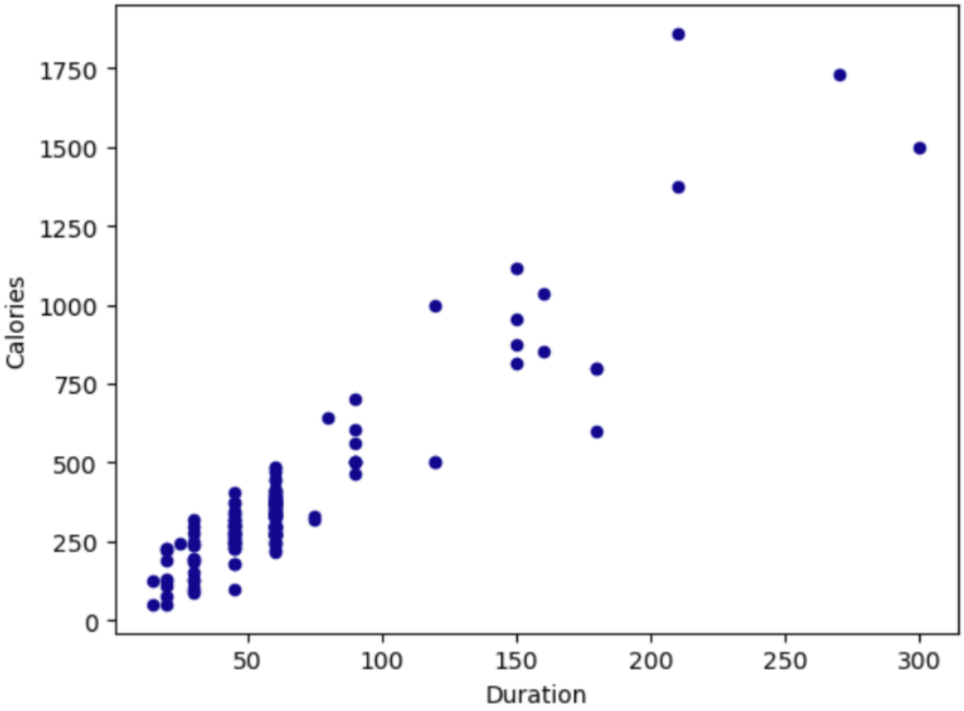
169 rows × 3 columns

- Creating a scatter plot for Duration and Calories.



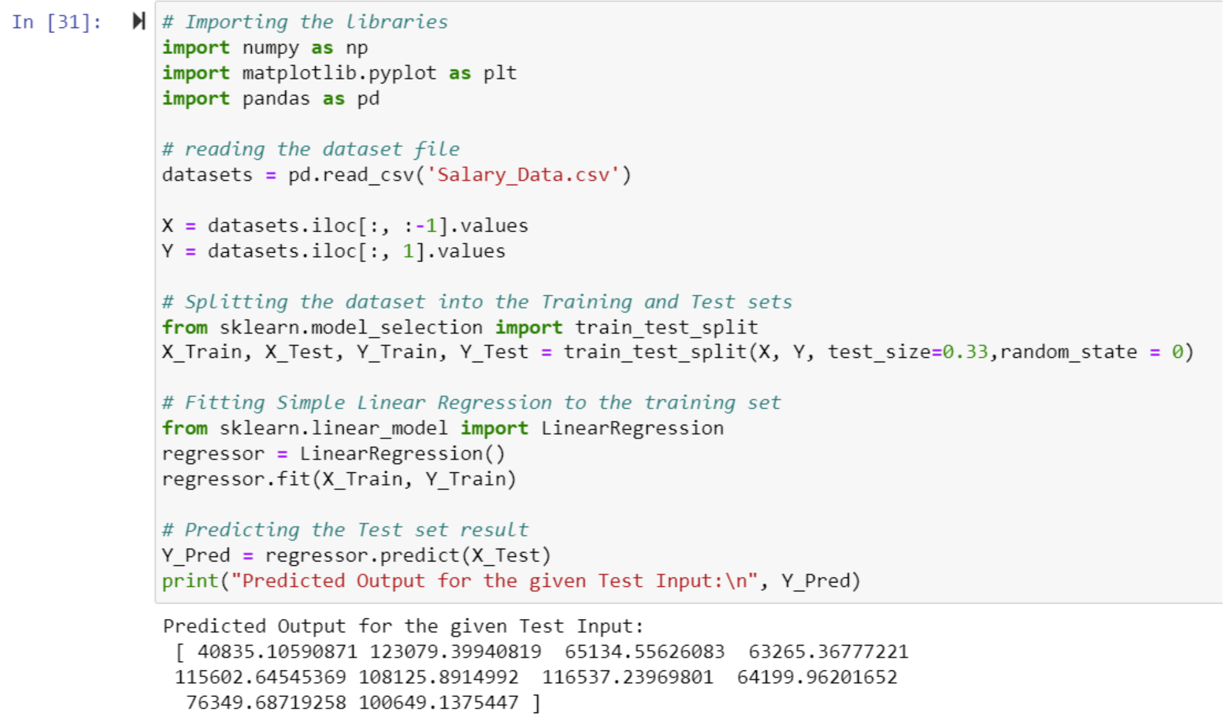
In [29]:

```
#create a scatter plot for the columns, Duration and Calories
df.plot.scatter(x='Duration', y='Calories', c='DarkBlue')
plt.show()
```



## Question 2:

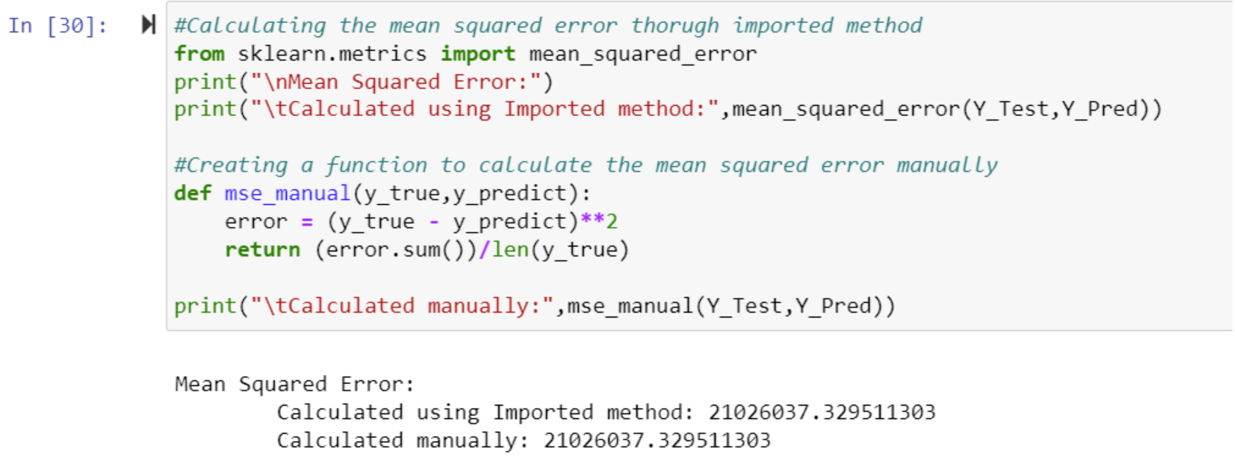
- Import the required libraries. Read the csv file containing the data set. Split the data into Training sets and Test sets such that 1/3rd of the data is reserved for test set. Train the model and get the predicted results, by feeding the test inputs to the model.



```
In [31]:
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# reading the dataset file
datasets = pd.read_csv('Salary_Data.csv')

X = datasets.iloc[:, :-1].values
Y = datasets.iloc[:, 1].values

# Splitting the dataset into the Training and Test sets
from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size=0.33,random_state = 0)

# Fitting Simple Linear Regression to the training set
from sklearn.linear_model import LinearRegression
regressor = LinearRegression()
regressor.fit(X_Train, Y_Train)

# Predicting the Test set result
Y_Pred = regressor.predict(X_Test)
print("Predicted Output for the given Test Input:\n", Y_Pred)
```

```
Predicted Output for the given Test Input:
 [ 40835.10590871 123079.39940819  65134.55626083  63265.36777221
 115602.64545369 108125.8914992  116537.23969801  64199.96201652
  76349.68719258 100649.1375447 ]
```

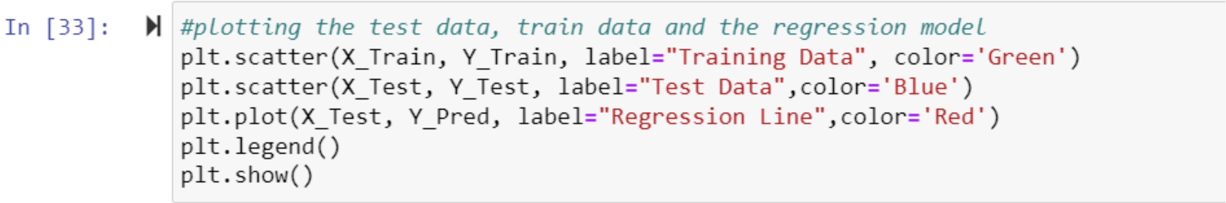
- Calculating the mean squared error. Here we calculated the mean squared error using both inbuilt function and user defined function.



```
In [30]:
#Calculating the mean squared error thorugh imported method
from sklearn.metrics import mean_squared_error
print("\nMean Squared Error:")
print("\tCalculated using Imported method:",mean_squared_error(Y_Test,Y_Pred))

#Creating a function to calculate the mean squared error manually
def mse_manual(y_true,y_predict):
    error = (y_true - y_predict)**2
    return (error.sum())/len(y_true)

print("\tCalculated manually:",mse_manual(Y_Test,Y_Pred))
```

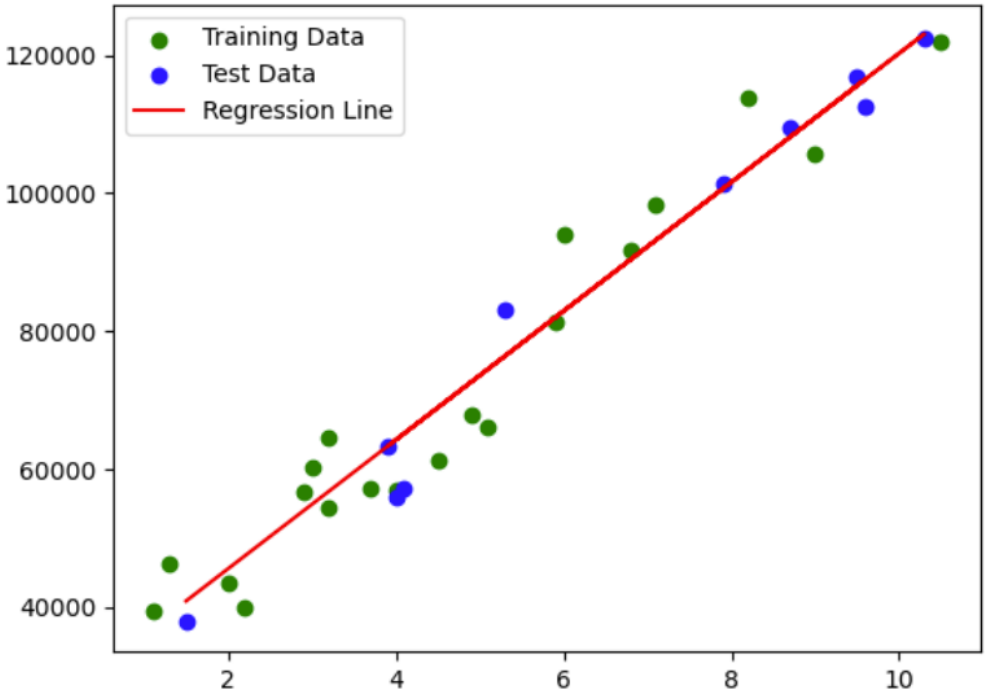
```
Mean Squared Error:
        Calculated using Imported method: 21026037.329511303
        Calculated manually: 21026037.329511303
```

- Visualizing the train data, test data using scatter plot and the regression line as well.



In [33]:

```
#plotting the test data, train data and the regression model
plt.scatter(X_Train, Y_Train, label="Training Data", color='Green')
plt.scatter(X_Test, Y_Test, label="Test Data",color='Blue')
plt.plot(X_Test, Y_Pred, label="Regression Line",color='Red')
plt.legend()
plt.show()
```


● Training Data
● Test Data
Regression Line