Name: Pagadala Sahil Naidu

Subject: Neural Networks & Deep Learning

Course ID: CS-5720

Email: sxp86940@ucmo.edu

CRN: 22317

Click the ICP3 named link to access the assignment in GitHub.

https://github.com/Sahilnaidupagadala03/Neural_Networks_Deeplearning

Below is Voice over video.

https://youtu.be/Nb10efzPFhM

Question 1

a:

c:

```
In [29]: import pandas as pan
         # Q1: a
         x = pan.read_csv('data.csv')

In [30]: # c
         x.describe()
```

Out[30]:

	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

a: Importing pandas as pan. x = pd.read_csv('data.csv') line reads csv file named data.csv and stores data in ‘x’.

c: output a summary of the statistics for each numeric column in dataframe.

d: -i

e:

```
In [31]: # d
         x.isnull().any()
```

```
Out[31]: Duration    False
         Pulse       False
         Maxpulse    False
         Calories     True
         dtype: bool
```

```
In [32]: # d - i

         x.fillna(x.mean(),inplace=True)
         x.isnull().any()
```

```
Out[32]: Duration    False
         Pulse       False
         Maxpulse    False
         Calories    False
         dtype: bool
```

```
In [33]: # e
         x.agg({'Pulse':['min','max','count','mean'],'Maxpulse':['min','max','count','mean']})
```

Out[33]:

	Pulse	Maxpulse
min	80.000000	100.000000
max	159.000000	184.000000
count	169.000000	169.000000
mean	107.461538	134.047337

d: x.isnull().any() expression is used to check whether there are any Null values in each column and .any() extension to it checks if there is at least one missing value in each column.

i) fillna method to replace missing values in x with the mean of each column. Mean() calculates the mean separately and inpace = true modifies the original data frame x.

e: This code computes the following aggregations for the 'Pulse' and 'Maxpulse' columns in the DataFrame x.

f:

In [34]:
```
# f
#filter the dataframe to select the rows btwn 500 and 1000
x.loc[(x['Calories']>500)&(x['Calories']<1000)]
```

Out[34]:

	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
90	180	101	127	600.1
99	90	93	124	604.1
103	90	90	100	500.4
106	180	90	120	800.3
108	90	90	120	500.3

.loc locates the values of the calories between 500 and 1000.

g:

h:

In [38]:
```
# g
# filter the dataframe to select the rows with calorie values > 500 and pulse < 100
x.loc[(x['Calories']>500)&(x['Pulse']<100)]
```

Out[38]:

	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

In [39]:
```
# h
#create a new dataframe that contains all columns except Maxpulse
df_modified=x[['Duration','Pulse','Calories']]
df_modified.head()
```

Out[39]:

	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

g: .loc [….] locates the data values for calories greater than 500 and pulse less than 100 and filters the table and store in x. Output is printed.

h: df_modified stores the newer data frame with Duration, Pulse and Calories in it.

i:

```
In [40]: # i
         del x['Maxpulse']
```

```
In [41]: x.head()
```

Out[41]:

	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

Del deletes the Maxpulse coloumn from the x data frame.

j:

```
In [42]: # j
         # Convert the datatype of Calories column to int datatype.
         x.dtypes
```

```
Out[42]: Duration      int64
         Pulse         int64
         Calories    float64
         dtype: object
```

```
In [43]: x = x.astype({'Calories':int})
         x.dtypes
```

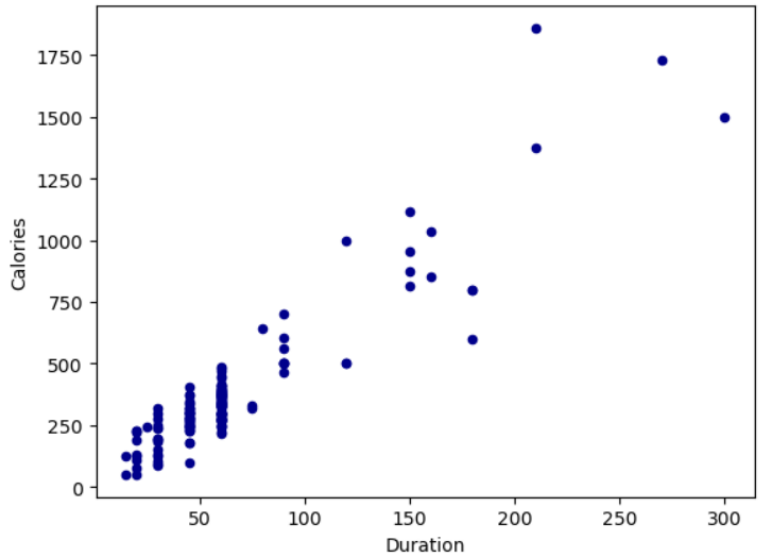
```
Out[43]: Duration    int64
         Pulse       int64
         Calories    int32
         dtype: object
```

.datatypes prints the data type of the respective column and .astype() converts the datatype of the required column and calorie in this case as int.

k:

In [44]:
```
# k: create a scatter plot for the 2 columns (Duration and Calories)
x.plot.scatter(x='Duration',y='Calories',c='DarkBlue')
```

Out[44]: <AxesSubplot:xlabel='Duration', ylabel='Calories'>



.plot.scatter() will create a scatter plot graph in the output. Here with Duration on x axis, calories o the y axis and the plot colour will be Dark blue.

2) Linear Regression:

a:

In [46]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
from sklearn import preprocessing
from sklearn.metrics import mean_squared_error

import seaborn as sns
sns.set(style="white", color_codes=True)
import warnings
warnings.filterwarnings("ignore")
```

In [48]:
```
# 2 - Linear Regression
# a
df =pd.read_csv("Salary_Data (2).csv")
```

All the imports namely seaborn, numpy, pandas, matplotlib.pyplot, train_test_split from sklearn.model_selection, sklearn.learn_model that has LinearRegression lib init, metrics and preprocessing from sklearn and mean_squared_error from sklearn.metrics.

a: .read_csv() to read the salary_Data dataset csv file.

b:

c:

d:

In [50]:
```
# b
# split the data in train_test partitions,such that 1/3 of the data is reserved as test subset
X = df.iloc[:,:-1].values
Y = df.iloc[:,1].values
X_Train,X_Test,Y_Train,Y_Test = train_test_split(X,Y,test_size=1/3,random_state=0)
```

In [51]:
```
# c
# train and predict the model
regressor=LinearRegression()
regressor.fit(X_Train,Y_Train)

Y_Pred=regressor.predict(X_Test)
```

In [52]:
```
# d
# calculate the mean squared error
mean_squared_error(Y_Test,Y_Pred)
```

Out[52]: 21026037.329511296

b: X: This line extracts the features from your Data Frame. It selects all rows and all columns except the last one (iloc[:, :-1]). The result is assigned to the variable X.

Y: This line extracts the target variable from your Data Frame. It selects all rows and only the last column (iloc[:, 1]). The result is assigned to the variable Y.

train_test_split() function is used to split your dataset into training and testing sets. It takes the features (X) and the target variable (Y) and test_size=1/3, random_state=0 parameters and returns four sets of data for train and splits fr X and Y.

c:

LinearRegression is a simple linear regression model used for predicting a target variable based on one or more independent variables. regressor.fit(X_Train, Y_Train): This line trains (fits) the linear regression model using the training data. It takes the training features X_Train and the corresponding target variable Y_Train to learn the coefficients of the linear regression equation. Y_Pred line predicts the target variable (Y_Pred) for the testing set X_Test using the trained linear regression model.

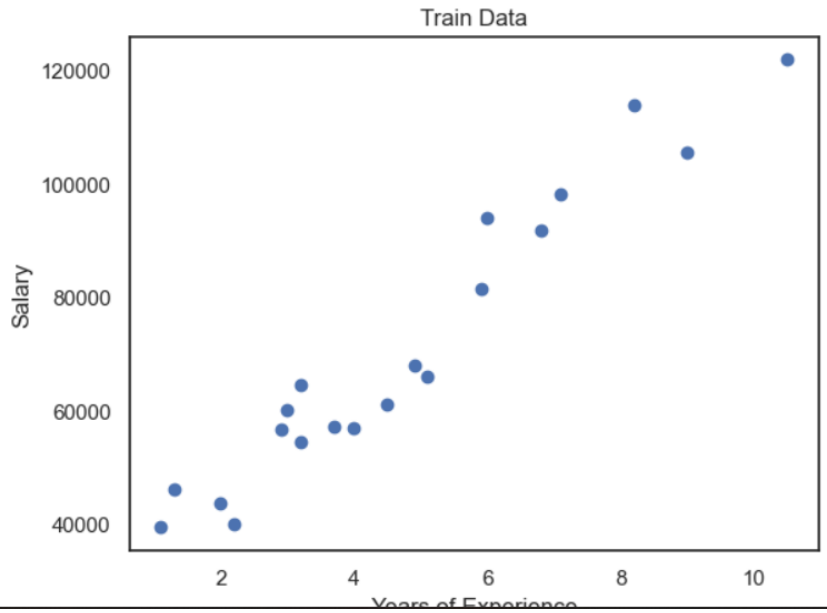
d:

mean_squared_error() function sklearn.metrics will give the mean swaure value which will be the error.

e:

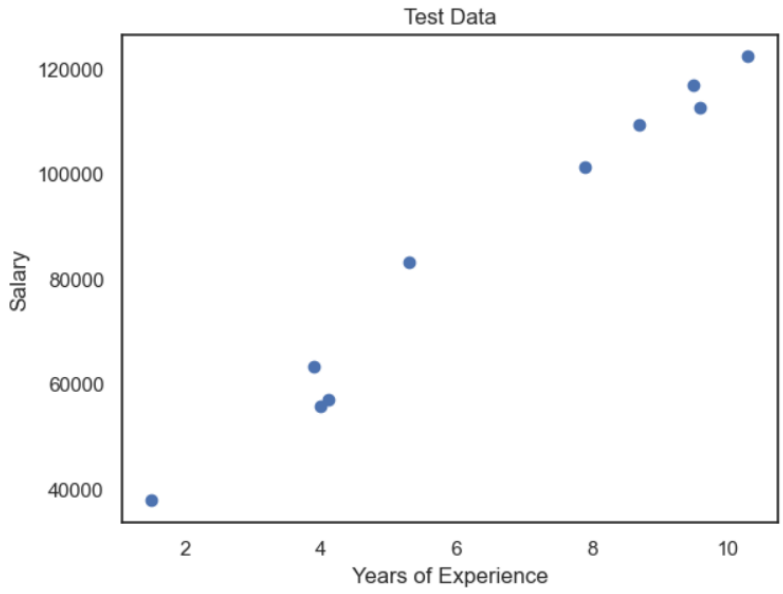
In [53]:
```
# e
#visualize both train and test data using scatter plot
plt.title("Train Data")
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.scatter(X_Train,Y_Train)
plt.show
```

Out[53]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [55]:
```
plt.title("Test Data")
plt.xlabel('Years of Experience')
plt.ylabel('Salary')
plt.scatter(X_Test,Y_Test)
plt.show
```

Out[55]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



Xlabel and Y label are the titles for the x and y axis and title() is the method used for naming the entire plot. Plt.scatter() will prepare a scatter plot for the values inserted and .show() will give the plot output.