Neural Networks and Deep Learning – Assignment 4

GitHub Link: https://github.com/srija1609/NNDL-ICP-4
Name: Baby Srija Bitra
Student ID: 700755908

1. **Data Manipulation:**
   a) Read the provided CSV file 'data.csv'
   b) https://drive.google.com/drive/folders/1h8C3mLsso-R-sIOLsvoYwPLzy2fJ4IOF?usp=sharing
   c) Show the basic statistical description about the data.

In this program, I have first imported the libraries required which are pandas, numpy and matplotlib. To read the given csv file I have used pandas pd.read_csv method into Data Frame. This supports optionally iterating or breaking of the file into chunks too.

For question c. I have used the describe () method which returns description of the data in the Data Frame. As our Data Frame contains numerical data the description contains for each column as count, mean, standard deviation, min, max values as statistical description.

```
In [1]: #1.a) #Importing basic packages for creating arrays & plotting graph

        import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt

        #Loading the Data Set
        #b) Read the provided CSV file 'data.csv'.
        df = pd.read_csv("data.csv")

        #c) Show the basic statistical description about the data.
        print(df.describe())
```

```
         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
```

   d) Check if the data has null values.
   i. Replace the null values with the mean

To check if the given data contains null values I have used df.isna().sum() method. This helps in returning the number of Null values in all columns of a pandas DataFrame. It shows the total Null values in a particular colum. As per our data file, the output shows that the Calories column has 5 Null values. Now these must be replaced with the mean values.

To calculate the mean() we use the mean function of the particular column. With the help of fillna() function we will change all 'Null' of that particular column for which we have its mean. The output below checks the null values first and prints then the null values are replaced with mean as shown. We then again verify if columns still have the null values. As shown in the output the Null values are 0 after replacing.

```
In [2]: #d) Check if the data has null values.
        df.isna().sum()

Out[2]: Duration    0
        Pulse       0
        Maxpulse    0
        Calories    5
        dtype: int64
```

```
In [3]: #d)i) Replacing Null values with the mean for Calories Column
        df["Calories"].fillna(df["Calories"].mean(),inplace=True)
        print(df)

             Duration  Pulse  Maxpulse  Calories
        0          60    110       130     409.1
        1          60    117       145     479.0
        2          60    103       135     340.0
        3          45    109       175     282.4
        4          45    117       148     406.0
        ..        ...    ...       ...       ...
        164        60    105       140     290.8
        165        60    110       145     300.0
        166        60    115       145     310.2
        167        75    120       150     320.4
        168        75    125       150     330.4

        [169 rows x 4 columns]
```

```
In [4]: #verifying any columns still with null values
        df.isna().sum()

Out[4]: Duration    0
        Pulse       0
        Maxpulse    0
        Calories    0
        dtype: int64
```

e) Select at least two columns and aggregate the data using: min, max, count, mean
To do this I have used groupby to group all the data in the dataframe and then by using the aggregate () method which allows to apply a function or a list of function names to be executed along one of the axis of the DataFrame. By using this we aggregate the min, max, mean and count of the given data in dataframe. The output is as shown below:

```
In [5]: #e) Select at least two columns and aggregate the data using: min, max, count, mean
        df.groupby('Duration').aggregate(['min', max, 'count',np.mean,])
```

Out[5]:

	Pulse				Maxpulse				Calories			
	min	max	count	mean	min	max	count	mean	min	max	count	mean
Duration												
15	80	124	2	102.000000	100	139	2	119.500000	50.5	124.2	2	87.350000
20	83	153	9	125.000000	107	172	9	146.000000	50.3	229.4	9	151.600000
25	152	152	1	152.000000	168	168	1	168.000000	244.2	244.2	1	244.200000
30	80	159	16	109.812500	107	182	16	137.000000	86.2	319.2	16	192.125000
45	90	149	35	107.485714	103	175	35	133.228571	100.7	406.0	35	279.096585
60	92	136	79	106.126582	101	170	79	132.860759	215.2	486.0	79	341.046465
75	120	125	2	122.500000	150	150	2	150.000000	320.4	330.4	2	325.400000
80	123	123	1	123.000000	146	146	1	146.000000	643.1	643.1	1	643.100000
90	90	100	8	93.750000	100	127	8	116.375000	466.4	700.0	8	541.800000
120	100	100	3	100.000000	130	157	3	139.000000	500.0	1000.1	3	666.833333
150	97	107	4	101.500000	127	135	4	130.250000	816.0	1115.0	4	939.400000
160	109	110	2	109.500000	135	137	2	136.000000	853.0	1034.4	2	943.700000
180	90	101	3	93.666667	120	130	3	125.666667	600.1	800.4	3	733.600000
210	108	137	2	122.500000	160	184	2	172.000000	1376.0	1860.4	2	1618.200000
270	100	100	1	100.000000	131	131	1	131.000000	1729.0	1729.0	1	1729.000000
300	108	108	1	108.000000	143	143	1	143.000000	1500.2	1500.2	1	1500.200000

f) Filter the dataframe to select the rows with calories values between 500 and 1000

The df['calories'] > 500 expression creates a boolean array that is True for the rows where the calories value is greater than 500. The df['calories'] < 1000 expression creates a boolean array that is True for the rows where the calories value is less than 1000. The & operator is used to combine these two arrays into a single boolean array that is True only for the rows where both conditions are True.

g) The df['calories'] > 500 expression creates a boolean array that is True for the rows where the calories value is greater than 500. The df[pulse] < 100 expression creates a boolean array that is True for the rows where the calories value is less than 100. The & operator is used to combine these two arrays into a single boolean array that is True only for the rows where both conditions are True.

```
In [6]: #f) Filter the dataframe to select the rows with calories values between 500 and 1000.
        df = df[(df['Calories'] > 500) & (df['Calories'] < 1000)]
        print(df)
```

```
     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
```

```
In [7]: #g) Filter the dataframe to select the rows with calories values > 500 and pulse < 100
        df = df[(df['Calories'] > 500) & (df['Pulse'] < 100)]
        print(df)
```

```
     Duration  Pulse  Maxpulse  Calories
65        180     90       130     800.4
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
```

h) Create a new “df_modified” dataframe that contains all the columns from df except for “Maxpulse”.
To do this I have used drop method that removes the specified column or from the dataframe. In this case, we're specifying axis=1 to indicate that we want to drop a column, and the "Maxpulse" column will be dropped from the dataframe. The printed output is as shown below:

```
In [8]: #h) Create a new “df_modified” dataframe that contains all the columns from df except for “Maxpulse”
        df_modified = df.drop("Maxpulse", axis=1)
        print(df_modified)
```

```
     Duration  Pulse  Calories
65        180     90     800.4
73        150     97     953.2
75         90     98     563.2
99         90     93     604.1
103        90     90     500.4
106       180     90     800.3
108        90     90     500.3
```

i) To delete the “Maxpulse” column from the df dataframe I have used the drop method. This method removes the column from the original main dataframe and by printing we show the resultant as shown below:

In [9]:
```
#i) Delete the “Maxpulse” column from the main df dataframe
if "Maxpulse" in df.columns:
    df.drop("Maxpulse", axis=1, inplace=True) #drop method removes the specified column or rows
print(df)
```

```
     Duration  Pulse  Calories
65        180     90     800.4
73        150     97     953.2
75         90     98     563.2
99         90     93     604.1
103        90     90     500.4
106       180     90     800.3
108        90     90     500.3
```

j) To Convert the datatype of Calories column to int datatype, I have used the astype method which changes the datatype of values in the column to integer type. Originally the values in our data file were float. The output after conversion looks like this:

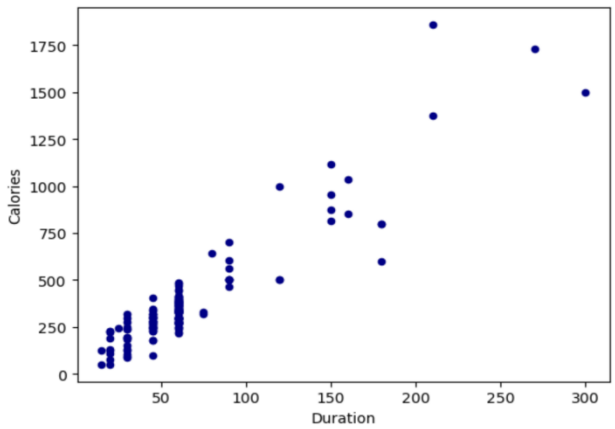
In [10]:
```
#j) Converting the datatype of Calories column to int datatype.
df["Calories"] = df["Calories"].astype(int)
print(df)
```

```
     Duration  Pulse  Calories
65        180     90       800
73        150     97       953
75         90     98       563
99         90     93       604
103        90     90       500
106       180     90       800
108        90     90       500
```

k) Using pandas create a scatter plot for the two columns (Duration and Calories)
we have already imported the matplotlib.pyplot library as plt, which is used to create the scatter plot. We use the plot method to create a scatter plot. Then the we create the scatter plot using the scatter. The resultant scatter plot is as shown below:

In [17]:
```
#k) Using pandas create a scatter plot for the two columns (Duration and Calories).
#plot method to create a scatter plot
data = pd.read_csv("data.csv")
data.plot(x="Duration", y="Calories", kind="scatter", c= "darkblue")
```

Out[17]: <AxesSubplot:xlabel='Duration', ylabel='Calories'>



2. Linear Regression:

a) Import the given “Salary_Data.csv”

In this program I have first imported the required libraries which are numpy, matplotlib.pyplot and pandas. For the first question to read the given csv file. I have used the pd.read_csv method which reads the given Salary_Data.csv file and stores in saldata. Using the info and head() methods I have printed the info from file.

In [20]:
```
#2. Linear Regression

# Importing the libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# a) Importing the datasets

saldata = pd.read_csv('Salary_Data.csv')

X = saldata.iloc[:, :-1].values #excluding last column
Y = saldata.iloc[:, 1].values #just salary column

saldata.info()
saldata.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 30 entries, 0 to 29
Data columns (total 2 columns):
 #   Column           Non-Null Count  Dtype  
---  ------           --------------  -----  
 0   YearsExperience  30 non-null     float64
 1   Salary           30 non-null     float64
dtypes: float64(2)
memory usage: 608.0 bytes
```

Out[20]:

	YearsExperience	Salary
0	1.1	39343.0
1	1.3	46205.0
2	1.5	37731.0
3	2.0	43525.0
4	2.2	39891.0

b) Split the data in train_test partitions, such that 1/3 of the data is reserved as test subset.
To split the data in partitions we first import the train_test_split function from the sklearn.model_selection. Then I have split the data into training and testing sets using the train_test_split function. The test_size parameter is set to 1/3, meaning that 1/3 of the data will be reserved for testing and the remaining 2/3 will be used for training. The random_state parameter is set to 0 to ensure that the same random partition of the data is obtained each time the code is run. I have then printed the split data as shown below:

In [19]:
```
#b) Splitting the dataset into the Training set and Test set

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=1/3, random_state=0)

## Print the split data
print("Below is the Split Data:")
print("Train features:")
print(pd.DataFrame(X_train).head())

print("Train targets:")
print(pd.DataFrame(Y_train).head())

print("Test features:")
print(pd.DataFrame(X_test).head())

print("Test targets:")
print(pd.DataFrame(Y_test).head())
```

```
Below is the Split Data:
Train features:
     0
0  2.9
1  5.1
2  3.2
3  4.5
4  8.2
Train targets:
          0
0   56642.0
1   66029.0
2   64445.0
3   61111.0
4  113812.0
Test features:
      0
0   1.5
1  10.3
2   4.1
3   3.9
4   9.5
   Test targets:
             0
   0   37731.0
   1  122391.0
   2   57081.0
   3   63218.0
   4  116969.0
```

c) To train and predict the model LinearRegression class is used to create the linear regression model. The fit method is used to train the model on the training data, and the predict method is used to make predictions on the test data. The y_pred variable contains the predicted values for the test data. As shown below:

In [23]:
```
# c) Train and predict the model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(X_train, Y_train) #predict method takes the test set features returns the predicted target variables
Y_pred = model.predict(X_test)
print("Predicted values:", Y_pred)
```

```
Predicted values: [ 40835.10590871 123079.39940819  65134.55626083  63265.36777221
 115602.64545369 108125.8914992  116537.23969801  64199.96201652
  76349.68719258 100649.1375447 ]
```

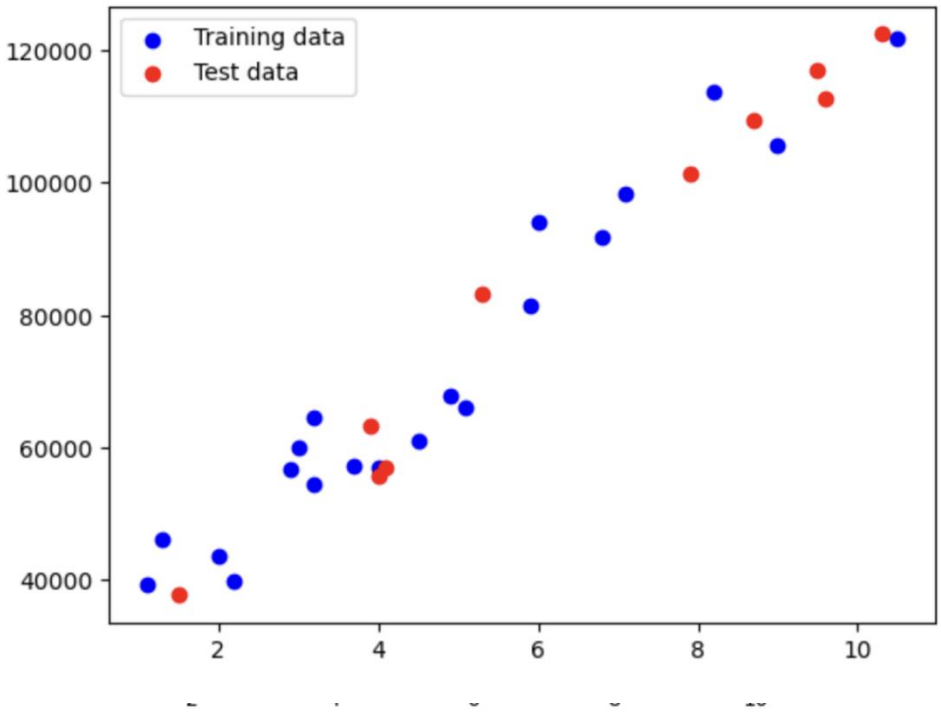
d) To calculate the mean squared error: Here, y_test is the true values, and y_pred is the predicted values. The mean_squared_error function takes these two arrays as input and returns the mean squared error between the two.

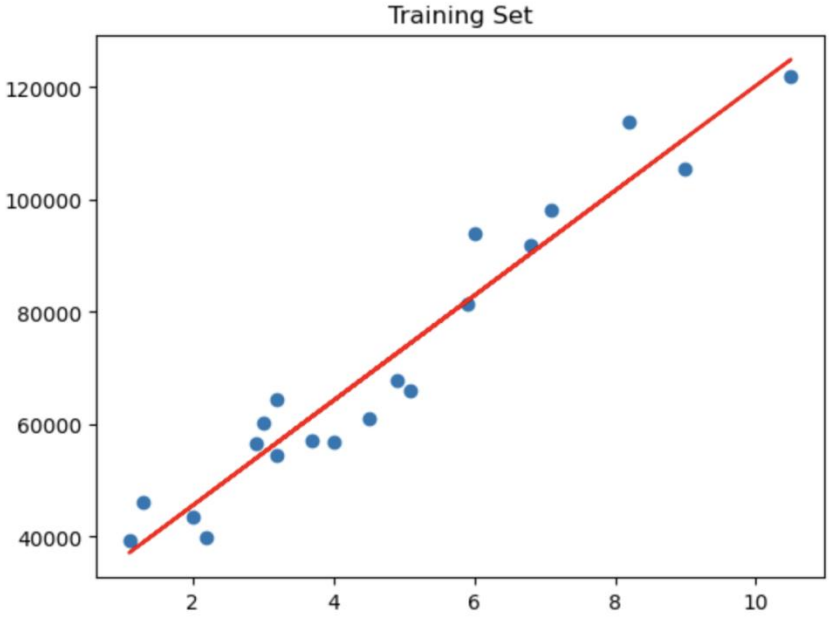
In [24]:
```
#d) Calculate the mean_squared error
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(Y_test, Y_pred)
print("Mean Squared Error:", mse)
```

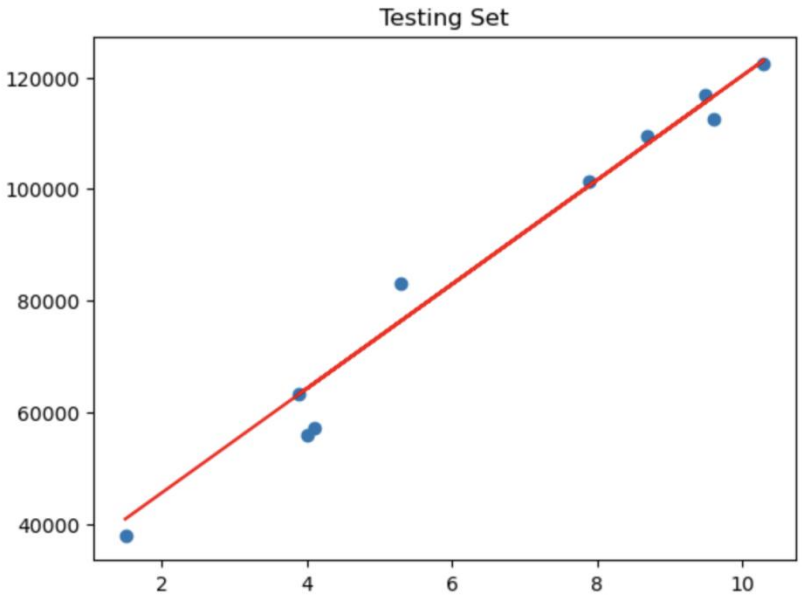
```
Mean Squared Error: 21026037.329511303
```

e) To visualize both the train and test data using the scatter plot I have used the imported library which is matplotlib.pyplot as plt. Below, X_train and y_train are the training data and target variables, and X_test and y_test are the test data and target variables, respectively. The scatter function is used to plot the data, and the color argument is used to specify the color of the data points. The label argument is used to specify the label for the legend. The legend function is used to add a legend to the plot, and the show function is used to display the plot.
The resultant scatter plot is as shown below:

In [28]:
```
#e) Visualize both train and test data using scatter plot.
#The scatter function is used to plot the training data, and the plot function is used to plot the predicted values
import matplotlib.pyplot as plt

plt.scatter(X_train, Y_train, color='blue', label='Training data')
plt.scatter(X_test, Y_test, color='red', label='Test data')
plt.legend()
plt.show()

# Training Data
plt.scatter(X_train, Y_train)
plt.plot(X_train, model.predict(X_train), color='red')
plt.title('Training Set')
plt.show()

# Testing Data
plt.scatter(X_test, Y_test)
plt.plot(X_test, model.predict(X_test), color='red')
plt.title('Testing Set')
plt.show()
```


Training data
Test data
120000
100000
80000
60000
40000
2
4
6
8
10


Training Set
120000
100000
80000
60000
40000
2
4
6
8
10


Testing Set
120000
100000
80000
60000
40000
2
4
6
8
10