Fall 2023: CS5720

Neural Networks & Deep Learning - ICP-4

Name: Manisha Lakkarsu

Student Id: 700746573

Git link: https://github.com/Mani543/Manisha_NNDL_ICP4.git

Video link: https://drive.google.com/file/d/1URitCku7pPY-qWkxRnlgTgbo6j_99dHM/view?usp=sharing

1. **Data Manipulation**

- Read the provided CSV file 'data.csv'.
- https://drive.google.com/drive/folders/1h8C3mLsso-R-sIOLsvoYwPLzy2fJ4IOF?usp=sharing

```
# Read the provided CSV file 'data.csv'.
user_data = pd.read_csv('data.csv')
user_data.info()
```

Output:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   Duration  169 non-null    int64
 1   Pulse     169 non-null    int64
 2   Maxpulse  169 non-null    int64
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
```

- Show the basic statistical description about the data.

```
# Show the basic statistical description about the data.
user_data.head()
```

Output:

Out[6]:

	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

- Check if the data has null values.

```
# Check if the data has null values
user_data.isnull().any()
user_data.fillna(user_data.mean(), inplace=True)
user_data.isnull().any()
```

Output:

Out[8]:
```
Duration    False
Pulse       False
Maxpulse    False
Calories    False
dtype: bool
```

- Replace the null values with the mean.

```
# Replace the null values with the mean
column_means = user_data.mean()
print(column_means)
user_data = user_data. fillna(column_means)
print(user_data.head(20))
```

Output:

```
Duration     63.846154
Pulse       107.461538
Maxpulse    134.047337
Calories    375.790244
dtype: float64
    Duration  Pulse  Maxpulse    Calories
0         60    110       130  409.100000
1         60    117       145  479.000000
2         60    103       135  340.000000
3         45    109       175  282.400000
4         45    117       148  406.000000
5         60    102       127  300.000000
6         60    110       136  374.000000
7         45    104       134  253.300000
8         30    109       133  195.100000
9         60     98       124  269.000000
10        60    103       147  329.300000
11        60    100       120  250.700000
12        60    106       128  345.300000
13        60    104       132  379.300000
14        60     98       123  275.000000
15        60     98       120  215.200000
16        60    100       120  300.000000
17        45     90       112  375.790244
18        60    103       123  323.000000
19        45     97       125  243.000000
```

- Select at least two columns and aggregate the data using: min, max, count, mean.

```
# Select at least two columns and aggregate the data using: min, max, count,mean.
result_set = user_data.agg({'Calories': ['mean', 'min','max', 'count'],'Pulse': ['mean', 'min', 'max', 'count']})
print(result_set)
```

Output:

```
          Calories       Pulse
mean    375.790244  107.461538
min      50.300000   80.000000
max    1860.400000  159.000000
count   169.000000  169.000000
```

- Filter the data frame to select the rows with calories values between 500 and1000.

```
# Filter the dataframe to select the rows with calories values between 500 and1000.
filter_usr_data1=user_data[(user_data['Calories'] > 500) & (user_data['Calories'] < 1000)]
print(filter_usr_data1)
```

Output:

	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

- Filter the data frame to select the rows with calories values > 500 and pulse.

```
# Filter the dataframe to select the rows with calories values > 500 and pulse <100.
filter_usr_data2=user_data[(user_data['Calories'] > 500) & (user_data['Pulse'] < 100)]
print(filter_usr_data2)
```

Output:

	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

- Create a new “df_modified” dataframe that contains all the columns from df except for “Maxpulse”.

```
# Create a new “df_modified” dataframe that contains all the columns from df exceptfor “Maxpulse”.
df_modified = user_data.loc[:, user_data.columns != 'Maxpulse']
print(df_modified)
```

Output:

```
     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 x 3 columns]
```

- Delete the “Maxpulse” column from the main df data frame.

```
# Delete the “Maxpulse” column from the main df dataframe
user_data.drop('Maxpulse', inplace=True, axis=1)
print(user_data.dtypes)
```

Output:

```
Duration      int64
Pulse         int64
Calories    float64
dtype: object
```

- Convert the datatype of Calories column to int datatype.

```
# Convert the datatype of Calories column to int datatype.
user_data["Calories"] = user_data["Calories"].astype(float).astype(int)
print(user_data.dtypes)
```

Output:

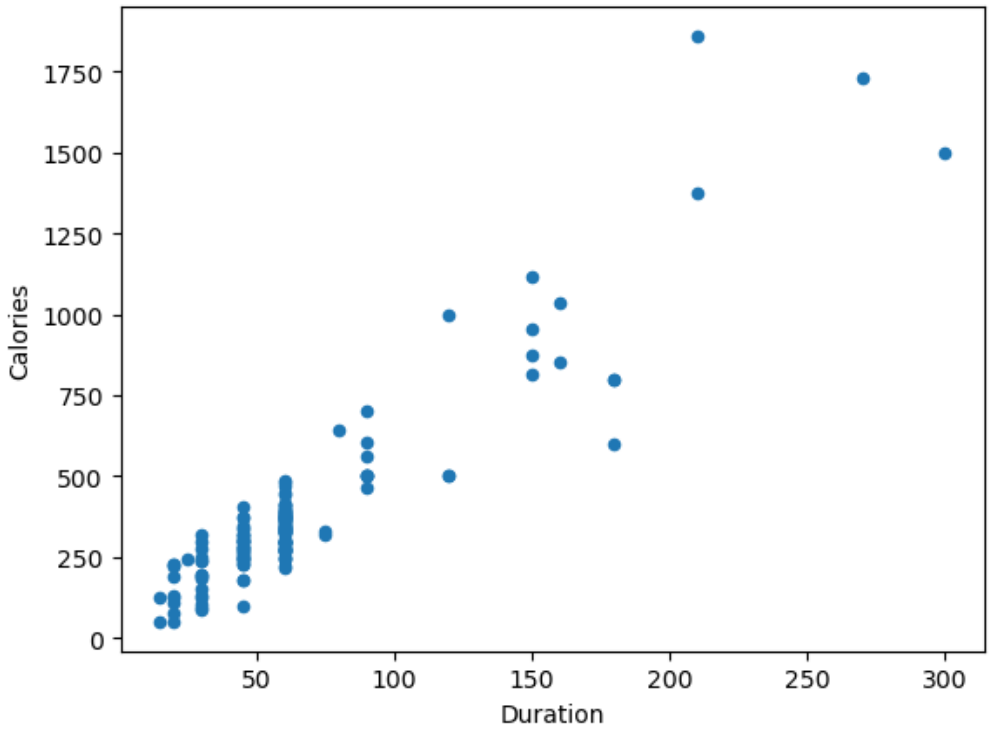
```
Duration     int64
Pulse        int64
Calories     int32
dtype: object
```

- Using pandas create a scatter plot for the two columns (Duration and Calories).

```
# Using pandas create a scatter plot for the two columns (Duration andCalories).
scatter_plot = user_data.plot.scatter(x='Duration',y='Calories')
print(scatter_plot)
```

Output:

```
Axes(0.125,0.11;0.775x0.77)
```



2. **Linear Regression**

- Import the given “Salary_Data.csv”

```
# Import the given “Salary_Data.csv”
sal_dataset = pd.read_csv('Salary_Data.csv')
sal_dataset.info()
sal_dataset.head()
```

Output:

```
<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
```

[21]:

	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

- Split the data in train test partitions, such that 1/3 of the data is reserved as test subset.

```
# Split the data in train_test partitions, such that 1/3 of the data is reserved as test subset.

#excluding last column i.e., years of experience column
set1 = sal_dataset.iloc[:, :-1].values

#only salary column
set2 = sal_dataset.iloc[:, 1].values
```

```
from sklearn.model_selection import train_test_split
```

```
set1_train, set1_test, set2_train, set2_test = train_test_split(set1, set2, test_size=1/3, random_state=0)
```

```
from sklearn.linear_model import LinearRegression
```

- Train and predict the model.

```
# Train and predict the model.
reg = LinearRegression()
reg.fit(set1_train, set2_train)
set2_Prediction = reg.predict(set1_test)
set2_Prediction
```

Output:

```
Out[35]: array([ 40835.10590871, 123079.39940819,  65134.55626083,  63265.36777221,
                115602.64545369, 108125.8914992 , 116537.23969801,  64199.96201652,
                 76349.68719258, 100649.1375447 ])
```

- Calculate the mean squared error.

```
In [36]:
# Calculate the mean_squared error
square_error = (set2_Prediction - set2_test) ** 2
sum_error = np.sum(square_error)
mean_squared_error = sum_error / set2_test.size
mean_squared_error
```

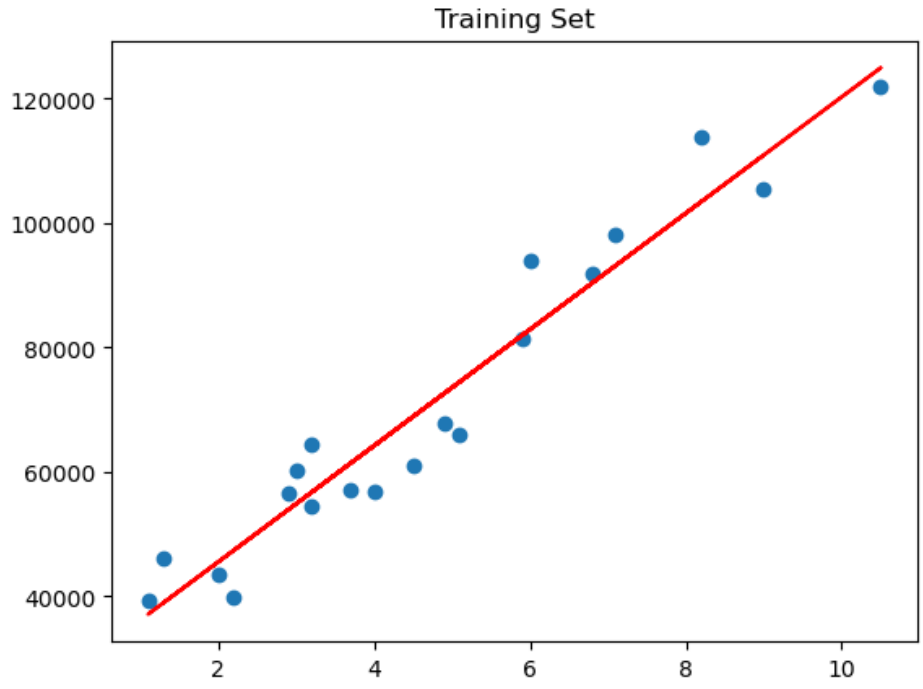
```
Out[36]: 21026037.329511296
```

- Visualize both train and test data using scatter plot.

```
In [39]:
# Visualize both train and test data using scatter plot.
import matplotlib.pyplot as scatter_plot
```

```
In [40]:
# Training Data set
scatter_plot.scatter(set1_train, set2_train)
scatter_plot.plot(set1_train, reg.predict(set1_train), color='red')
scatter_plot.title('Training Set')
scatter_plot.show()
```

Output:



In [41]:
```
# Testing Data set
scatter_plot.scatter(set1_test, set2_test)
scatter_plot.plot(set1_test, reg.predict(set1_test), color='red')
scatter_plot.title('Testing Set')
scatter_plot.show()
```