

Spring 2024: CS5720

NEURAL NETWORK AND DEEP LEARNING

ICP-4 CRN: 22317 Name: Afroz Mohammad [700758012]

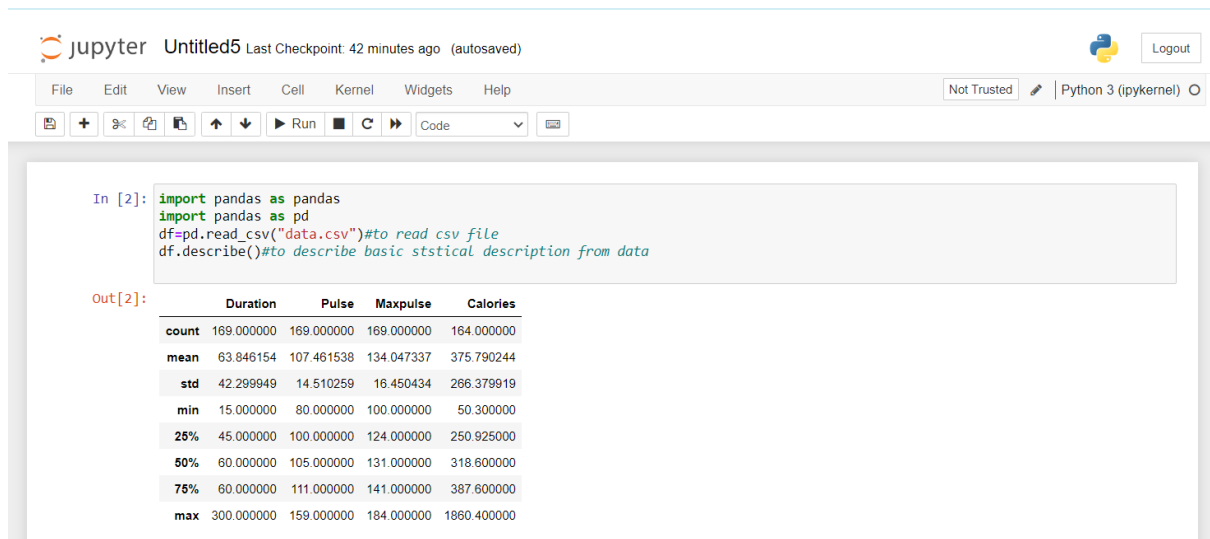
Git Hub Link:<https://github.com/Afrozmoshammad19/Assignment4>

Video Link:

<https://drive.google.com/file/d/1He1NNrQDuSi2Fsl9iPuXktPdkTGXc07Z/view?usp=sharing>

- Read the provided CSV file 'data.csv'.
- <https://drive.google.com/drive/folders/1h8C3mLsso-R-slOLsvoYwPLzy2fJ4IOF?usp=sharing>
- Show the basic statistical description about the data.

Output:



The screenshot shows a Jupyter Notebook interface with a code cell and its output. The code cell contains the following Python code:

```
In [2]: import pandas as pandas
import pandas as pd
df=pd.read_csv("data.csv")#to read csv file
df.describe()#to describe basic ststical description from data
```

The output of the code is a statistical summary of the data, displayed as a table:

| | 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 |

- Check if the data has null values.
- Replace the null values with the mean



The screenshot shows a Jupyter Notebook interface with a code cell and its output. The code cell contains the following Python code:

```
In [3]: show_null=df.isnull().sum()#display null values
print(show_null)#print the null values
```

The output of the code is a single line of text:

```
Duration    0
Pulse       0
Maxpulse    0
Calories    5
dtype: int64
```

```
In [4]: df.fillna(df.mean(),inplace=True)#replace null values with mean value
print(df)#print after replacing with mean
```

| | 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]

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

```
In [5]: df=df[["Duration","Pulse","Maxpulse","Calories"]]
aggregate={"Duration":["max","min","count","mean"],
           "Pulse":["max","min","count","mean"],
           "Maxpulse":["max","min","count","mean"],
           "Calories":["max","min","count","mean"]}# to find max,min,count,mean of all the columns
aggregate_df=df.agg(aggregate)#function to aggregate
print(aggregate_df)#print the aggregate
```

| | Duration | Pulse | Maxpulse | Calories |
|-------|------------|------------|------------|-------------|
| max | 300.000000 | 159.000000 | 184.000000 | 1850.400000 |
| min | 15.000000 | 80.000000 | 100.000000 | 50.300000 |
| count | 169.000000 | 169.000000 | 169.000000 | 169.000000 |
| mean | 63.846154 | 107.461538 | 134.047337 | 375.790244 |

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

```
In [6]: calories_in_range=(df["Calories"]>500) & (df["Calories"]<=1000)#defining range of values to be displayed
filters_result=df[calories_in_range]#adding the defined range to new variable
print(filters_result)#printing the new result
```

| | 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 |

g. Filter the dataframe to select the rows with calories values > 500 and pulse < 100.

```
In [7]: calories_pulse_filter=(df["calories"]>500)&(df["pulse"]<100)#defining range
filters_result=df[calories_pulse_filter]#adding the result to new variable
print(filters_result)#printing the result
```

| | 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 |

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

```
In [8]: df_modified=df.drop(columns=["Maxpulse"])#displaying every column except Maxpulse
print(df_modified)#printing the result
```

| | 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]

i. Delete the “Maxpulse” column from the main df dataframe

```
In [9]: del df["Maxpulse"]#command to delete entire row
print(df)
```

| | 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]

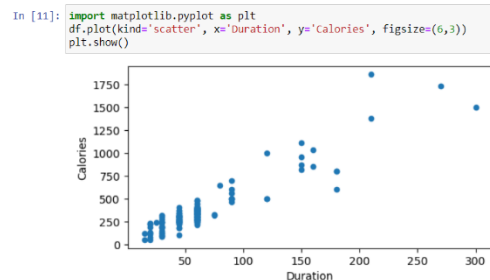
j. Convert the datatype of Calories column to int datatype.

```
In [10]: df['Calories'] = df['Calories'].fillna(0).astype(int)#converting to int data type
print(df)
```

| | 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 x 3 columns]

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



2. Linear Regression

a) Import the given “Salary_Data.csv”

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

c) Train and predict the model.

d) Calculate the mean_squared error

e) Visualize both train and test data using scatter plot.

Output:

UPDATE Read the migration plan to Notebook 7 to learn about the new features and the actions to take if you are using extensions - Please note that updating to Notebook 7 might break some of your extensions. Don't show anymore

jupyter Untitled5 Last Checkpoint: an hour ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Not Trusted Python 3 (ipykernel)

Duration

```
In [13]: import pandas as pd
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split

salariesData = pd.read_csv('Salary_Data (2).csv') #importing data from the csv file
df.describe()
```

Out[13]:

| | Duration | Pulse | Calories |
|-------|------------|------------|-------------|
| count | 169.000000 | 169.000000 | 169.000000 |
| mean | 63.846154 | 107.461538 | 375.544379 |
| std | 42.299949 | 14.510259 | 262.411685 |
| min | 15.000000 | 80.000000 | 50.000000 |
| 25% | 45.000000 | 100.000000 | 253.000000 |
| 50% | 60.000000 | 105.000000 | 321.000000 |
| 75% | 80.000000 | 111.000000 | 384.000000 |
| max | 300.000000 | 159.000000 | 1860.000000 |

```
In [14]: #splitting the data in to training and testing
X = salariesData.iloc[:, :-1].values
Y = salariesData.iloc[:, 1].values
```

```
In [14]: #splitting the data in to training and testing
X = salariesData.iloc[:, :-1].values
Y = salariesData.iloc[:, 1].values

In [15]: #splitting 1/3 of the data
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 1/3, random_state = 0)

In [16]: # Fitting Simple Linear Regression to the training set
reg = LinearRegression()
reg.fit(X_train, Y_train)

Out[16]:
LinearRegression
LinearRegression()

In [17]: # Predicting the Test set result
pred = reg.predict(X_test)

In [18]: # Calculating the Mean_squared_error
mse = mean_squared_error(Y_test, pred)
```

