DSBDA Assignment 5

Details

1. Author : Aditya Muthal
2. Roll Number : 33249
3. Batch : M10
4. Class : TE10

Problem Statement

Perform the following operations using Python on the Air quality and Heart Diseases data sets

1. Data cleaning
2. Data integration
3. Data transformation
4. Error correcting
5. Data model building

Implementation details

1. Dataset URL : https://archive.ics.uci.edu/ml/datasets/Heart+Disease
2. Python version : 3.7.4
3. Imports :
    1. pandas
    2. numpy
    3. matplotlib.pyolot
    4. seaborn
    5. sklearn.linear_model.LogisticRegression

Dataset details

1. This database contains 76 attributes, but all published experiments refer to using a subset of 14 of them. In particular, the Cleveland database is the only one that has been used by ML researchers to this date.
2. The "goal" field refers to the presence of heart disease in the patient.
3. It is integer valued from 0 (no presence) to 4. Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0).

4. The names and social security numbers of the patients were recently removed from the database, replaced with dummy values.

Importing required libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Loading the dataset

```
dataset = pd.read_csv("/content/Heart.csv", index_col=0)
```

Displaying metadata for dataset (Statistical)

```
dataset.shape
```

```
(303, 14)
```

```
dataset.isnull().sum()
```

```
Age          0
Sex          0
ChestPain    0
RestBP       0
Chol         0
Fbs          0
RestECG      0
MaxHR        0
ExAng        0
Oldpeak      0
Slope        0
Ca           4
Thal         2
AHD          0
dtype: int64
```

```
dataset.head(15)
```

	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	Slop
1	63	1	typical	145	233	1	2	150	0	2.3	
2	67	1	asymptomatic	160	286	0	2	108	1	1.5	
3	67	1	asymptomatic	120	229	0	2	129	1	2.6	
4	37	1	nonanginal	130	250	0	0	187	0	3.5	
5	41	0	nontypical	130	204	0	2	172	0	1.4	
6	56	1	nontypical	120	236	0	0	178	0	0.8	
7	62	0	asymptomatic	140	268	0	2	160	0	3.6	
8	57	0	asymptomatic	120	354	0	0	163	1	0.6	
9	63	1	asymptomatic	130	254	0	2	147	0	1.4	
10	53	1	asymptomatic	140	203	1	2	155	1	3.1	
11	57	1	asymptomatic	140	192	0	0	148	0	0.4	
12	56	0	nontypical	140	294	0	2	153	0	1.3	
13	56	1	nonanginal	130	256	1	2	142	1	0.6	
14	44	1	nontypical	120	263	0	0	173	0	0.0	

Observations :

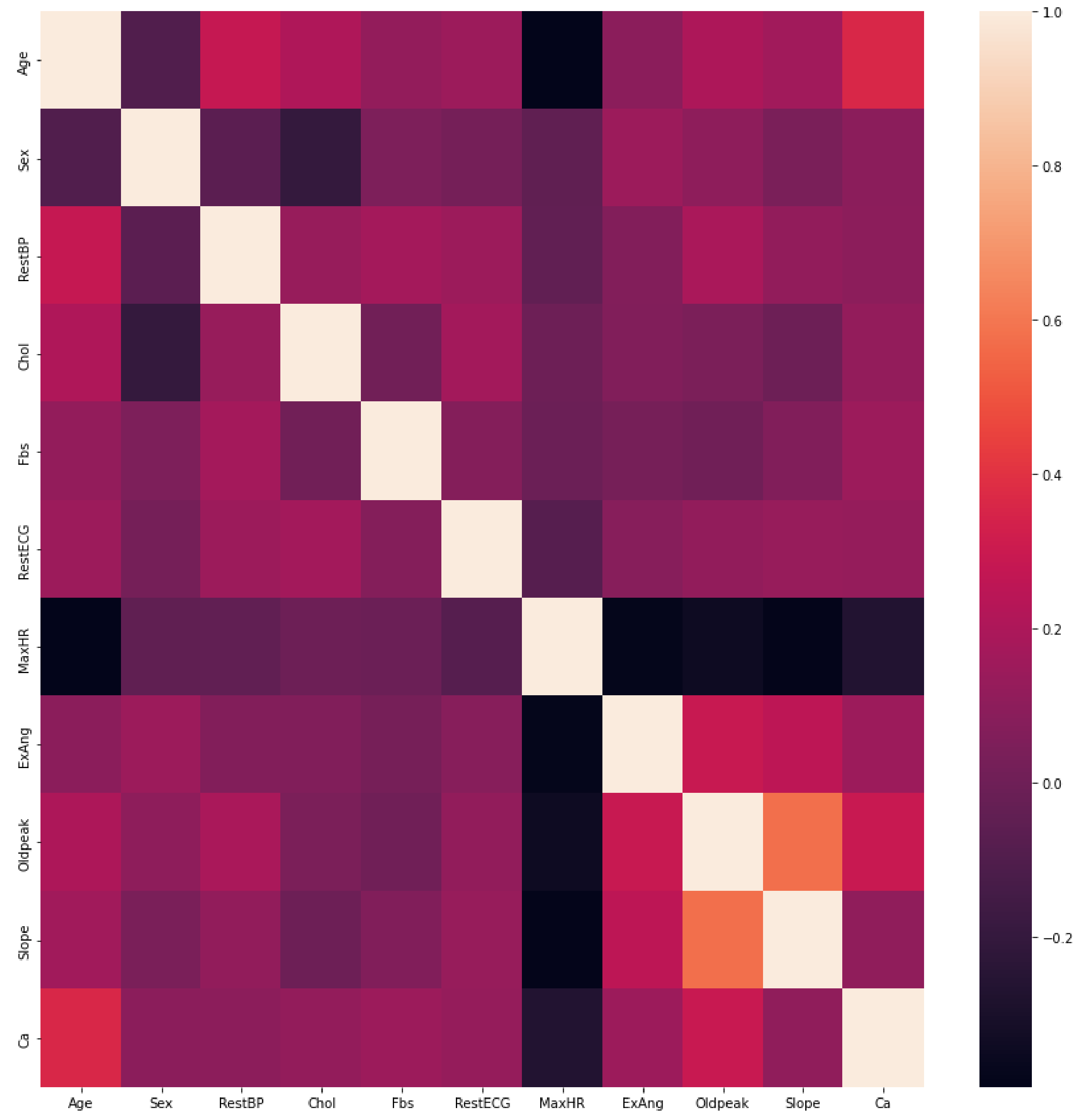
1. There are 682 data points with 14 columns (including target column)
2. Null values are removed / replaced and dataset is scaled for numerical variables

A) Feature selection

```
# Displaying heatmap for correlation matrix
fig = plt.figure(figsize=(15, 15))

# Adds subplot on position 1
ax = fig.add_subplot(111)

sns.heatmap(dataset.corr())
plt.show()
```



```
dataset.corr()
```

	Age	Sex	RestBP	Chol	Fbs	RestECG	MaxHR
Age	1.000000	-0.097542	0.284946	0.208950	0.118530	0.148868	-0.393806
Sex	-0.097542	1.000000	-0.064456	-0.199915	0.047862	0.021647	-0.048663

Note :

1. The above heatmap and the correlation table suggests that all of the data features are significantly correlated with the target variables
2. The following features are negatively correlated with the target variable :
    1. cholestrol
    2. thalach

Further action :

1. No feature drop is necessary due to significant correlation with target variable
2. The target variable is considered to be "num" (last column of the dataset)

B) Building the data model

1) Understanding the target variable

```
# checking the unique values (categories in the dataset)
dataset.num.unique()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-16-d95b6be9efae> in <module>()
      1 # checking the unique values (categories in the dataset)
----> 2 dataset.num.unique()

/usr/local/lib/python3.7/dist-packages/pandas/core/generic.py in
__getattr__(self, name)
   5485         ):
   5486             return self[name]
-> 5487         return object.__getattribute__(self, name)
   5488 
   5489     def __setattr__(self, name: str, value) -> None:

AttributeError: 'DataFrame' object has no attribute 'num'
```

SEARCH STACK OVERFLOW

Note :

1. The values beyond 0 are indicative of the fact that there is presence of heart disease

Further action :

1. Binarize the target variable for the classes as presence or absence of heart disease

2) Binarizing the target variable

```
dataset["num"] = dataset["num"].replace([2, 3, 4], 1)
```

```
dataset.num.unique()
```

3) Checking distribution of target variable

```
# Plotting the count plot for target variable
fig = plt.figure(figsize=(8, 8))

# Adds subplot on position 1
ax = fig.add_subplot(111)

plt.xlabel("num", fontsize=20)
plt.ylabel("count", fontsize=20)

sns.countplot(x=dataset.num)
plt.show()
```

```
# creating subsets for target variables for fair distribution in training and testing data
dataset_target_0 = dataset[dataset.num == 0]
dataset_target_1 = dataset[dataset.num == 1]
print("Shape of target 1 data : ", dataset_target_1.shape)
print("Shape of target 0 data : ", dataset_target_0.shape)
```

```
# Shuffling the data subsets
dataset_target_1 = dataset_target_1.sample(frac=1)
dataset_target_0 = dataset_target_0.sample(frac=1)

# Confirming shapes for no value loss
print("Shape of target 1 data : ", dataset_target_1.shape)
print("Shape of target 0 data : ", dataset_target_0.shape)
```

4) Creating training and testing data with 80:20 ratio

```
# Calculating 80 percent mark
target_0_mark = int(dataset_target_0.shape[0]*0.8)
target_1_mark = int(dataset_target_1.shape[0]*0.8)

# Generating train data
train_data = pd.concat(
    objs=[
        dataset_target_0.iloc[:target_0_mark, :],
        dataset_target_1.iloc[:target_1_mark, :]
    ],
    axis=0
)

# Generating test data
test_data = pd.concat(
    objs=[
        dataset_target_0.iloc[target_0_mark:, :],
        dataset_target_1.iloc[target_1_mark:, :]
    ],
    axis=0
)

# Shuffling the training and testing data
train_data = train_data.sample(frac=1)
test_data = test_data.sample(frac=1)

# Checking data shapes
print("Training data shape : ", train_data.shape)
print("Testing data shape  : ", test_data.shape)
```

5) Splitting training and testing inputs and targets

```
# Splitting training data
train_inputs = train_data.iloc[:, :-1]
train_targets = train_data.iloc[:, -1]

# Splitting testing data
test_inputs = test_data.iloc[:, :-1]
test_targets = test_data.iloc[:, -1]

# Checking shape of data
print("Train inputs shape : ", train_inputs.shape)
print("Train targets shape : ", train_targets.shape)
print("Test inputs shape : ", test_inputs.shape)
print("Test targets shape : ", test_targets.shape)
```

6) Building the data model

```
# Importing model
from sklearn.linear_model import LogisticRegression

logReg_model = LogisticRegression()
```

```
# Training the model
logReg_model.fit(train_inputs, train_targets)
print("Model trained")
```

7) Checking accuracy of model on testing data

```
logReg_model.score(test_inputs, test_targets)
```

```
logReg_model.score(train_inputs, train_targets)
```

Conclusion

1. The logistic regression model was fit on the given dataset
2. The model gave 89.85% accuracy on testing data and 82.53% accuracy on testing data

End of Notebook