Experiment – 6

Aim: Implement a logistic regression model on given dataset and check the accuracy for test dataset.

Context: Logistic Regression is a statistical model that is used to predict the probability of a binary outcome based on one or more inputs. It is a type of supervised machine learning algorithm. It is based on probability based approach as there are fixed possible outcomes for the output. It is the statistical fitting of a sigmoid curve to a dataset in order to calculate the probability of the occurrence of a specific categorical event based on the values of a set of independent variables. The output of a logistic regression model is a probability value between 0 and 1. Sigmoid function is defined by the formula 1/(1+e^(-x)) where x is the input value.

Dataset Description: The dataset contains 8,124 records and uses 22 attributes to describe the car. Each dataset contains additional attribute to flag if the car is manual or automatic.

Code & Output:

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import LogisticRegression
import io

from google.colab import files
uploaded=files.upload()

33343<IPython.core.display.HTML object>

Saving car_data.csv to car_data.csv

df=pd.read_csv(io.BytesIO(uploaded["car_data.csv"]),encoding='latin1',inde
x_col=0)
df

          Year  Selling_Price  Present_Price  Kms_Driven Fuel_Type  \
Car_Name
ritz      2014           3.35           5.59       27000    Petrol
sx4       2013           4.75           9.54       43000    Diesel
ciaz      2017           7.25           9.85        6900    Petrol
wagon r   2011           2.85           4.15        5200    Petrol
swift     2014           4.60           6.87       42450    Diesel
...        ...            ...            ...         ...       ...
city      2016           9.50          11.60       33988    Diesel
```

```
brio      2015          4.00          5.90       60000    Petrol
city      2009          3.35         11.00       87934    Petrol
city      2017         11.50         12.50        9000    Diesel
brio      2016          5.30          5.90        5464    Petrol

         Seller_Type Transmission  Owner
Car_Name
ritz          Dealer       Manual      0
sx4           Dealer       Manual      0
ciaz          Dealer       Manual      0
wagon r       Dealer       Manual      0
swift         Dealer       Manual      0
...              ...          ...    ...
city          Dealer       Manual      0
brio          Dealer       Manual      0
city          Dealer       Manual      0
city          Dealer       Manual      0
brio          Dealer       Manual      0

[301 rows x 8 columns]
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 301 entries, ritz to brio
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Year           301 non-null    int64
 1   Selling_Price  301 non-null    float64
 2   Present_Price  301 non-null    float64
 3   Kms_Driven     301 non-null    int64
 4   Fuel_Type      301 non-null    object
 5   Seller_Type    301 non-null    object
 6   Transmission   301 non-null    object
 7   Owner          301 non-null    int64
dtypes: float64(2), int64(3), object(3)
memory usage: 21.2+ KB
```

```
df.isnull().sum()
```

```
Year             0
Selling_Price    0
Present_Price    0
Kms_Driven       0
Fuel_Type        0
Seller_Type      0
Transmission     0
Owner            0
dtype: int64
```

```
ed=pd.get_dummies(df, columns=["Fuel_Type"])
ed
```

```
          Year  Selling_Price  Present_Price  Kms_Driven Seller_Type  \
Car_Name
ritz      2014           3.35           5.59       27000      Dealer
sx4       2013           4.75           9.54       43000      Dealer
ciaz      2017           7.25           9.85        6900      Dealer
wagon r   2011           2.85           4.15        5200      Dealer
swift     2014           4.60           6.87       42450      Dealer
...        ...            ...            ...         ...         ...
city      2016           9.50          11.60       33988      Dealer
brio      2015           4.00           5.90       60000      Dealer
city      2009           3.35          11.00       87934      Dealer
city      2017          11.50          12.50        9000      Dealer
brio      2016           5.30           5.90        5464      Dealer

         Transmission  Owner  Fuel_Type_CNG  Fuel_Type_Diesel  \
Car_Name
ritz           Manual      0              0                 0
sx4            Manual      0              0                 1
ciaz           Manual      0              0                 0
wagon r        Manual      0              0                 0
swift          Manual      0              0                 1
...               ...    ...            ...               ...
city           Manual      0              0                 1
brio           Manual      0              0                 0
city           Manual      0              0                 0
city           Manual      0              0                 1
brio           Manual      0              0                 0

          Fuel_Type_Petrol
Car_Name
ritz                     1
sx4                      0
ciaz                     1
wagon r                  1
swift                    0
...                    ...
city                     0
brio                     1
city                     1
city                     0
brio                     1

[301 rows x 10 columns]
```

```
x=ed.iloc[:,[0,1,2,3,7,8,9]]
x
```

```
          Year  Selling_Price  Present_Price  Kms_Driven  Fuel_Type_CNG  \
Car_Name
ritz      2014           3.35           5.59       27000              0
sx4       2013           4.75           9.54       43000              0
ciaz      2017           7.25           9.85        6900              0
wagon r   2011           2.85           4.15        5200              0
```

```
swift     2014          4.60          6.87      42450            0
...        ...           ...           ...        ...          ...
city      2016          9.50         11.60      33988            0
brio      2015          4.00          5.90      60000            0
city      2009          3.35         11.00      87934            0
city      2017         11.50         12.50       9000            0
brio      2016          5.30          5.90       5464            0

          Fuel_Type_Diesel  Fuel_Type_Petrol
Car_Name
ritz                     0                 1
sx4                      1                 0
ciaz                     0                 1
wagon r                  0                 1
swift                    1                 0
...                    ...               ...
city                     1                 0
brio                     0                 1
city                     0                 1
city                     1                 0
brio                     0                 1

[301 rows x 7 columns]
```

```
y=ed["Transmission"].values
y
```

```
array(['Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Automatic', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Automatic', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Automatic', 'Manual',
       'Manual', 'Manual', 'Manual', 'Automatic', 'Manual', 'Manual',
       'Manual', 'Automatic', 'Automatic', 'Automatic', 'Automatic',
       'Automatic', 'Manual', 'Automatic', 'Manual', 'Manual', 'Manual',
       'Automatic', 'Manual', 'Manual', 'Automatic', 'Automatic',
       'Automatic', 'Manual', 'Automatic', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Automatic', 'Automatic', 'Automatic', 'Manual',
       'Manual', 'Automatic', 'Manual', 'Manual', 'Automatic', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Automatic', 'Automatic', 'Manual', 'Automatic', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
```

```
       'Manual', 'Manual', 'Manual', 'Automatic', 'Manual', 'Manual',
       'Manual', 'Automatic', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Automatic', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Automatic', 'Manual', 'Manual',
       'Manual', 'Automatic', 'Automatic', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Automatic', 'Automatic', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Automatic', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Automatic', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Automatic', 'Manual', 'Manual',
       'Automatic', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Automatic', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Automatic', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Automatic', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Automatic', 'Manual', 'Automatic', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual'], dtype=object)
```

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3,rand
om_state=100)
print("x_train shape: ",x_train.shape,"y_train shape: ",y_train.shape)
print("\nx_test shape: ",x_test.shape,"y_test shape: ",y_test.shape)
l=LogisticRegression()
l.fit(x_train,y_train)
y_pred = l.predict(x_test)
print(y_pred)
```

```
x_train shape:  (210, 7) y_train shape:  (210,)

x_test shape:  (91, 7) y_test shape:  (91,)
['Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual'
 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual'
 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual'
 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual'
 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual'
 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual'
 'Manual' 'Manual' 'Automatic' 'Manual' 'Manual' 'Manual' 'Manual'
 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual'
 'Manual' 'Automatic' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual'
 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual'
 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Automatic' 'Manual'
 'Manual' 'Manual' 'Manual' 'Manual' 'Manual' 'Manual']
```

```
from sklearn.metrics import accuracy_score
a=accuracy_score(y_test,y_pred)
print(a)
```

```
0.9340659340659341
```

```
y_pred
```

```
array(['Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Automatic', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Automatic', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Automatic',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual'], dtype=object)
```

```
y_test
```

```
array(['Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Automatic', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Automatic', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Automatic', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Automatic', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Automatic', 'Manual',
       'Manual', 'Manual', 'Automatic', 'Manual', 'Automatic', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual', 'Manual', 'Manual', 'Manual', 'Automatic', 'Automatic',
       'Manual', 'Manual', 'Manual', 'Manual', 'Manual', 'Manual',
       'Manual'], dtype=object)
```

Experiment – 7

Title: To Build a decision tree model for given dataset to predict the target with best accuracy.

Context: A Decision Tree is a supervised machine learning algorithm used for classification and regression tasks. It is a tree-based model that splits the data into smaller subsets based on the most significant feature and continues to split until it reaches the bottom nodes, which are the final decisions. The internal nodes represent the feature splits, and the leaf nodes represent the decisions. The main goal of a decision tree is to create a tree-like model that accurately predicts the target variable for a new data point by following a set of decisions based on the features.

Dataset Description: The dataset contains 8,124 records and uses 22 attributes to describe the mushrooms. Each dataset contains additional attribute to flag if the mushroom is edible or not.

Code & Output:

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn import tree
from sklearn.tree import DecisionTreeClassifier
from sklearn import metrics
from sklearn.metrics import accuracy_score

from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>

Saving mushrooms.csv to mushrooms (1).csv
```

```
import pandas as pd
import io
df=pd.read_csv(io.BytesIO(uploaded['mushrooms.csv']))
df
```

```
     class cap-shape cap-surface cap-color bruises odor gill-attachment  \
0        p         x           s         n       t    p               f
1        e         x           s         y       t    a               f
2        e         b           s         w       t    l               f
3        p         x           y         w       t    p               f
4        e         x           s         g       f    n               f
...    ...       ...         ...       ...     ...  ...             ...
8119     e         k           s         n       f    n               a
8120     e         x           s         n       f    n               a
8121     e         f           s         n       f    n               a
8122     p         k           y         n       f    y               f
8123     e         x           s         n       f    n               a

     gill-spacing gill-size gill-color
```

```
0              c         n          k
1              c         b          k
2              c         b          n
3              c         n          n
4              w         b          k
...          ...       ...        ...
8119           c         b          y
8120           c         b          y
8121           c         b          n
8122           c         n          b
8123           c         b          y

[8124 rows x 10 columns]
```

```
df.shape
```

```
(8124, 10)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   class            8124 non-null   object
 1   cap-shape        8124 non-null   object
 2   cap-surface      8124 non-null   object
 3   cap-color        8124 non-null   object
 4   bruises          8124 non-null   object
 5   odor             8124 non-null   object
 6   gill-attachment  8124 non-null   object
 7   gill-spacing     8124 non-null   object
 8   gill-size        8124 non-null   object
 9   gill-color       8124 non-null   object
dtypes: object(10)
memory usage: 634.8+ KB
```

```
df.describe()
```

```
       class cap-shape cap-surface cap-color bruises  odor gill-attachment  
\
count   8124      8124        8124      8124    8124  8124            8124   
unique     2         6           4        10       2     9               2   
top        e         x           y         n       f     n               f   
freq    4208      3656        3244      2284    4748  3528            7914   

       gill-spacing gill-size gill-color  
count          8124      8124       8124  
unique            2         2         12  
top               c         b          b  
freq           6812      5612       1728  
```

```
df['gill-color'].unique()
```

```
array(['k', 'n', 'g', 'p', 'w', 'h', 'u', 'e', 'b', 'r', 'y', 'o'],
      dtype=object)
```

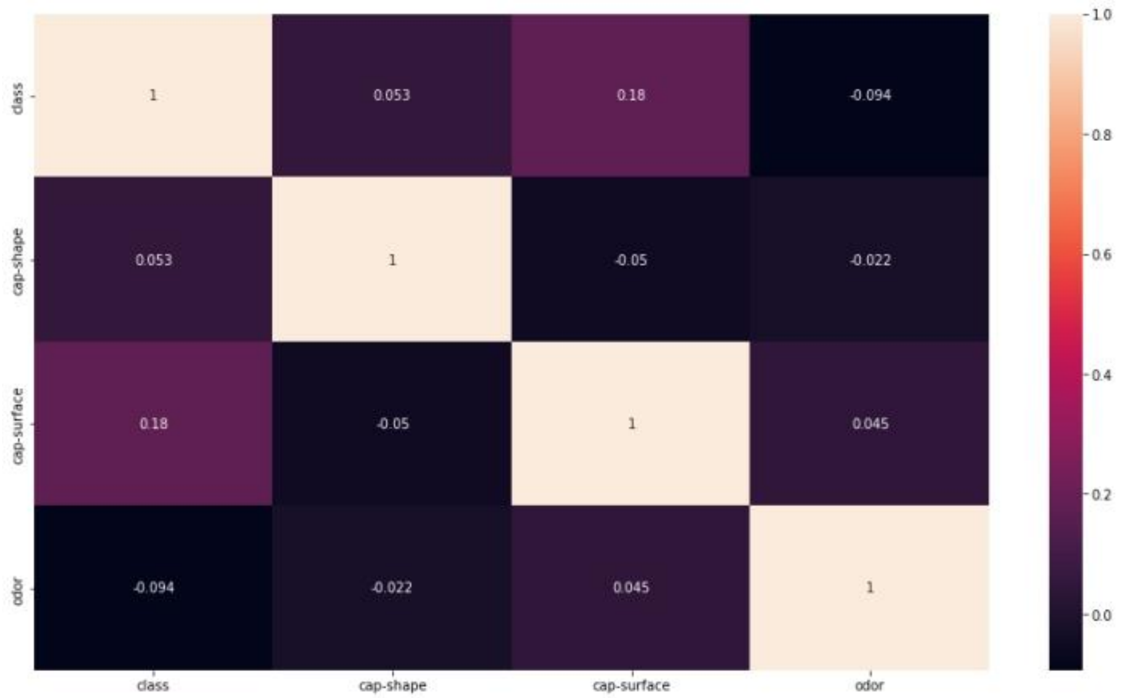
```
df.isnull().sum()
```

```
class              0
cap-shape          0
cap-surface        0
cap-color          0
bruises            0
odor               0
gill-attachment    0
gill-spacing       0
gill-size          0
gill-color         0
dtype: int64
```

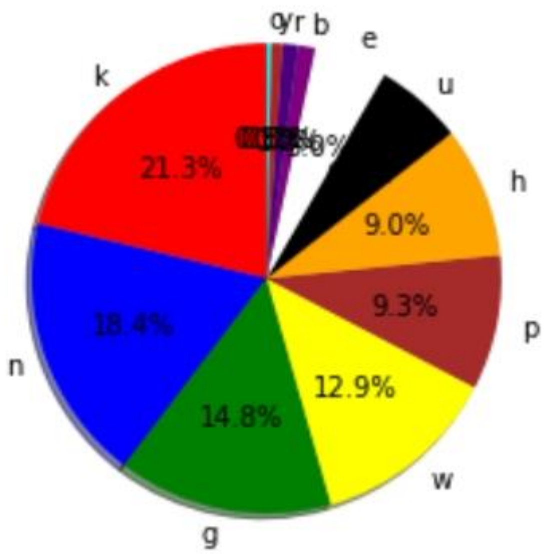
```
drug_label = LabelEncoder()
df['class']=drug_label.fit_transform(df['class'])
df['cap-shape']=drug_label.fit_transform(df['cap-shape'])
df['cap-surface']=drug_label.fit_transform(df['cap-surface'])
df['odor']=drug_label.fit_transform(df['odor'])
```

```
import seaborn as sns
plt.figure(figsize=(16,9))
sns.heatmap(df.corr(),annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f9f9793bb80>
```



```
labels=['k','n','g','w','p','h', 'u', 'e', 'b', 'r', 'y', 'o']
values=df['gill-color'].value_counts().values
fig1, ax1 = plt.subplots()
colors = ['red','blue','green','yellow','brown','orange','black','white','
purple','indigo','firebrick','cyan']
ax1.pie(values, labels=labels, autopct = '%1.1f%%',shadow=True,startangle=
90,colors=colors)
plt.show()
```



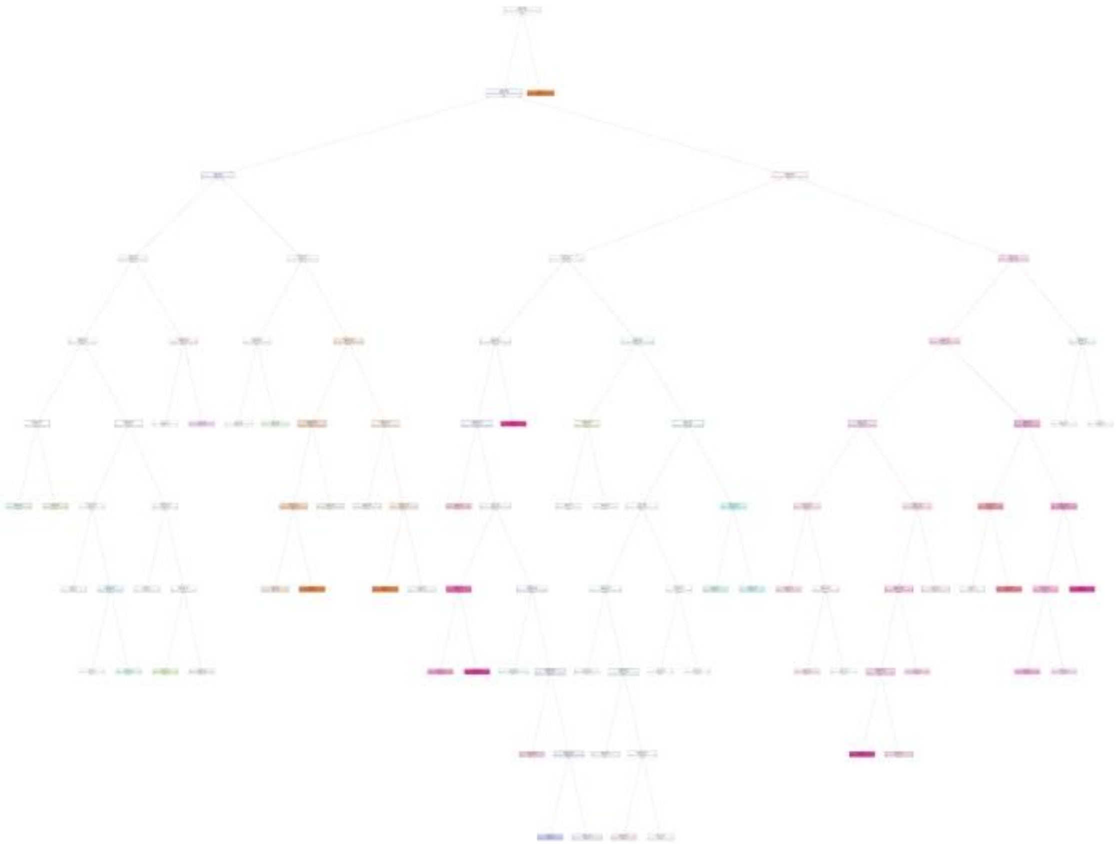

```
df[['class','cap-shape','cap-surface','odor']].head()
```

	class	cap-shape	cap-surface	odor
0	1	5	2	6
1	0	5	2	0
2	0	0	2	3
3	1	5	3	6
4	0	5	2	5

```
x=df[['class','cap-shape','cap-surface','odor']]
y=df['gill-color']
x_train,x_test,y_train,y_test = train_test_split(x,y,train_size = 0.2,rand
om_state = 0)
tree_model = DecisionTreeClassifier()
tree_model = tree_model.fit(x_train,y_train)
y_pred = tree_model.predict(x_test)
accuracy = metrics.accuracy_score(y_test,y_pred)
accuracy
```

0.38676923076923075

```
fig = plt.figure(figsize=(120,100))
tree_fig = tree.plot_tree(tree_model,feature_names = ['class','cap-shape',
'cap-surface','odor'],class_names = ['k','n','g','w','p','h', 'u', 'e', 'b
', 'r', 'y', 'o'],filled = True)
```



Experiment – 8

Aim: Implement KNN model to classify the target in given dataset.

Context: *k*-NN is a type of classification where the function is only approximated locally and all computation is deferred until function evaluation. Since this algorithm relies on distance for classification, if the features represent different physical units or come in vastly different scales then normalizing the training data can improve its accuracy dramatically.

Dataset Description: This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V. It contains 76 attributes, including the predicted attribute, but all published experiments refer to using a subset of 14 of them. The "target" field refers to the presence of heart disease in the patient. It is integer valued 0 = no disease and 1 = disease.

Code & Output:

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import seaborn as sns

from sklearn.neighbors import KNeighborsClassifier

dataset = pd.read_csv('https://raw.githubusercontent.com/kb22/Heart-Diseas
e-Prediction/master/dataset.csv')

dataset
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak  \
0     63    1   3       145   233    1        0      150      0      2.3
1     37    1   2       130   250    0        1      187      0      3.5
2     41    0   1       130   204    0        0      172      0      1.4
3     56    1   1       120   236    0        1      178      0      0.8
4     57    0   0       120   354    0        1      163      1      0.6
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
298   57    0   0       140   241    0        1      123      1      0.2
299   45    1   3       110   264    0        1      132      0      1.2
300   68    1   0       144   193    1        1      141      0      3.4
301   57    1   0       130   131    0        1      115      1      1.2
302   57    0   1       130   236    0        0      174      0      0.0

     slope  ca  thal  target
0        0   0     1       1
1        0   0     2       1
2        2   0     2       1
3        2   0     2       1
4        2   0     2       1
..     ...  ..   ...     ...
```

```
298      1   0     3       0
299      1   0     3       0
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0

[303 rows x 14 columns]
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   age       303 non-null    int64
 1   sex       303 non-null    int64
 2   cp        303 non-null    int64
 3   trestbps  303 non-null    int64
 4   chol      303 non-null    int64
 5   fbs       303 non-null    int64
 6   restecg   303 non-null    int64
 7   thalach   303 non-null    int64
 8   exang     303 non-null    int64
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64
 11  ca        303 non-null    int64
 12  thal      303 non-null    int64
 13  target    303 non-null    int64
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
dataset.describe()
```

```
              age         sex          cp    trestbps        chol
fbs  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000
000
mean    54.366337    0.683168    0.966997  131.623762  246.264026    0.148
515
std      9.082101    0.466011    1.032052   17.538143   51.830751    0.356
198
min     29.000000    0.000000    0.000000   94.000000  126.000000    0.000
000
25%     47.500000    0.000000    0.000000  120.000000  211.000000    0.000
000
50%     55.000000    1.000000    1.000000  130.000000  240.000000    0.000
000
75%     61.000000    1.000000    2.000000  140.000000  274.500000    0.000
000
max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000
000
```

```
          restecg     thalach       exang     oldpeak       slope
ca  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000
000
mean     0.528053  149.646865    0.326733    1.039604    1.399340    0.729
373
std      0.525860   22.905161    0.469794    1.161075    0.616226    1.022
606
min      0.000000   71.000000    0.000000    0.000000    0.000000    0.000
000
25%      0.000000  133.500000    0.000000    0.000000    1.000000    0.000
000
50%      1.000000  153.000000    0.000000    0.800000    1.000000    0.000
000
75%      1.000000  166.000000    1.000000    1.600000    2.000000    1.000
000
max      2.000000  202.000000    1.000000    6.200000    2.000000    4.000
000

             thal      target
count  303.000000  303.000000
mean     2.313531    0.544554
std      0.612277    0.498835
min      0.000000    0.000000
25%      2.000000    0.000000
50%      2.000000    1.000000
75%      3.000000    1.000000
max      3.000000    1.000000
```

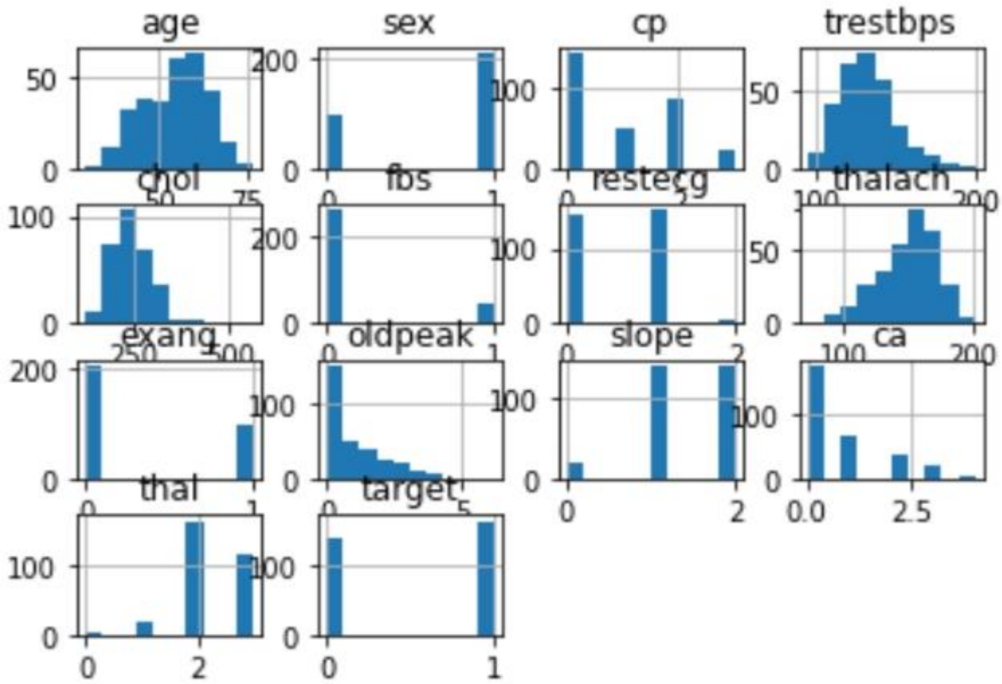
```
dataset.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

```
dataset.hist()
```

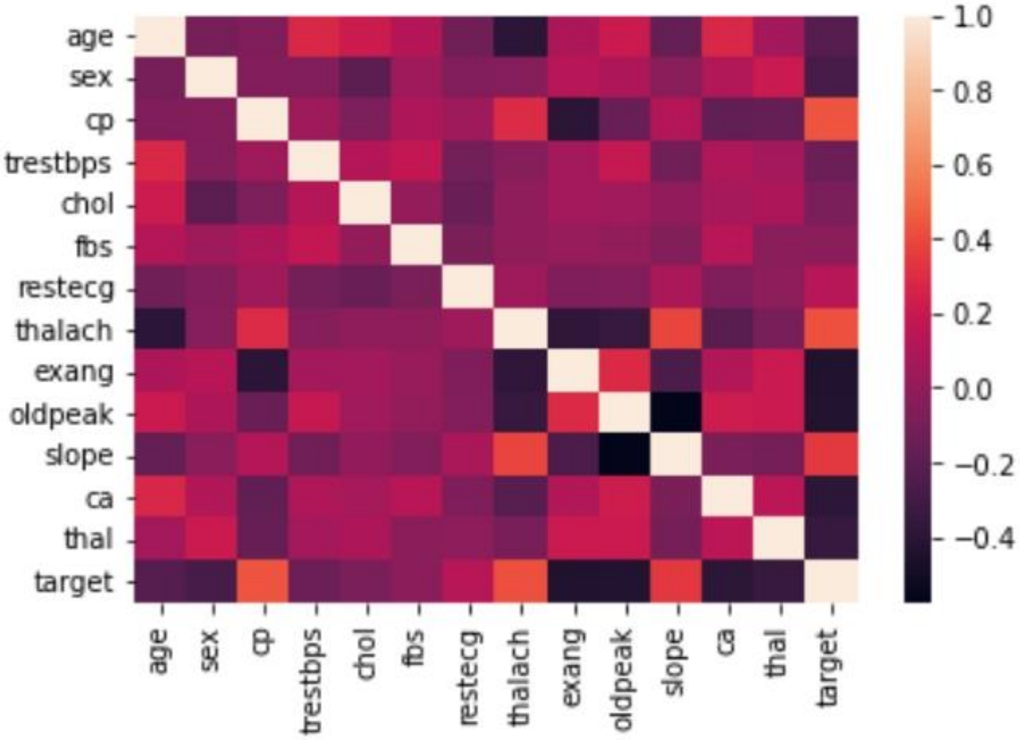
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf7c7b1f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf7771370>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf779f7c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf7756b50>],
```

```
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf7705fa0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf76bf310>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf76bf400>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf766c880>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf76420a0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf75fa430>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf75a6820>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf75d3c70>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf7581160>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf753e4f0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf74eb820>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f1bf750ddc0>]]
,
      dtype=object)
```



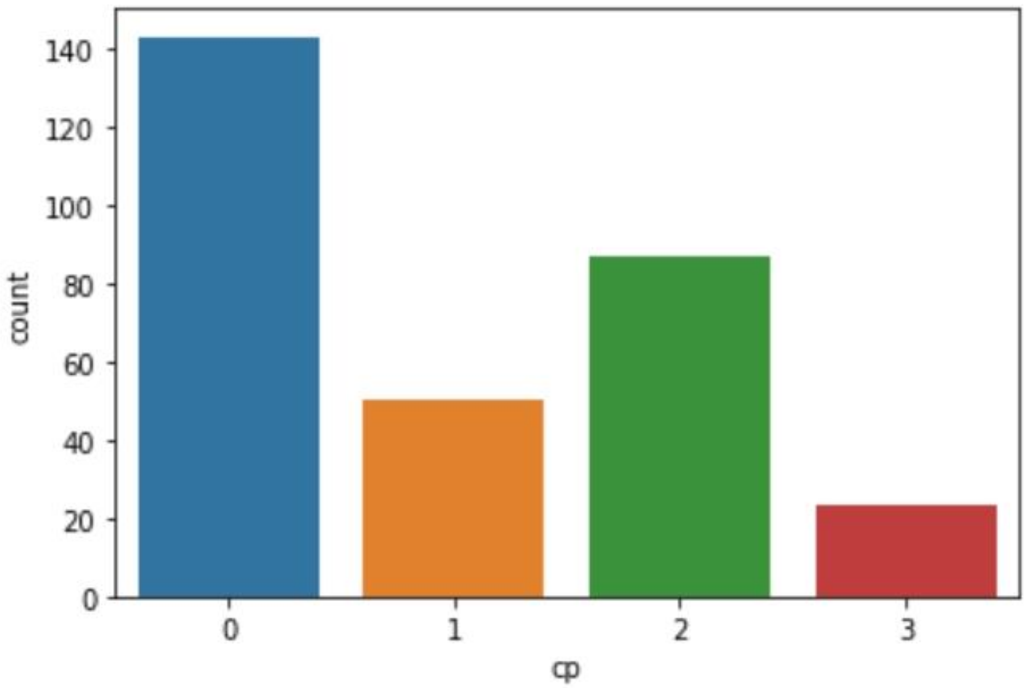

```
hm = dataset.corr()
sns.heatmap(hm)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1bfa584400>
```




```
sns.countplot(x='cp', data=dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1bf728dfd0>
```




```
y = dataset['target']
X = dataset.drop(['target'], axis = 1)
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.33
, random_state = 0)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
```

```
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5)
classifier.fit(x_train, y_train)
```

```
KNeighborsClassifier()
```

```
y_pred = classifier.predict(x_test)
```

```
from sklearn.metrics import classification_report, confusion_matrix, accur
acy_score
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
```

```
Confusion Matrix:
[[38 10]
 [ 5 47]]
```

```
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
```

```
Classification Report:
              precision    recall  f1-score   support

           0       0.88      0.79      0.84        48
           1       0.82      0.90      0.86        52

    accuracy                           0.85       100
   macro avg       0.85      0.85      0.85       100
weighted avg       0.85      0.85      0.85       100
```

```
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)
```

```
Accuracy: 0.85
```

Experiment – 9

Aim: Demonstrate regression and classification metrics using sample data.

Context: Linear Regression and Logistic Regression are the two famous Machine Learning Algorithms which come under supervised learning technique. Since both the algorithms are of supervised in nature hence these algorithms use labeled dataset to make the predictions. But the main difference between them is how they are being used. The Linear Regression is used for solving Regression problems whereas Logistic Regression is used for solving the Classification problems.

Dataset Description: This dataset contains the age,gender,estimated salary of a person and determines whether it is purchased or not.

Code & Output:

Import libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score
from sklearn.metrics import classification_report,confusion_matrix,accurac
y_score
```

Reading dataset

```
df=pd.read_csv("Social_Network_Ads.csv")
```

Data Preprocessing

```
df.head()
```

```
    User ID  Gender  Age  EstimatedSalary  Purchased
0  15624510    Male   19            19000          0
1  15810944    Male   35            20000          0
2  15668575  Female   26            43000          0
3  15603246  Female   27            57000          0
4  15804002    Male   19            76000          0
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    object
```

```
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(4), object(1)
memory usage: 15.8+ KB
```

```
df.isnull().sum()
```

```
User ID            0
Gender             0
Age                0
EstimatedSalary    0
Purchased          0
dtype: int64
```

Encoding the data

```
le = LabelEncoder()
df['Gender']= le.fit_transform(df['Gender'])
df['Gender'].unique()
```

```
array([1, 0])
```

```
df.head()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	1	19	19000	0
1	15810944	1	35	20000	0
2	15668575	0	26	43000	0
3	15603246	0	27	57000	0
4	15804002	1	19	76000	0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   User ID          400 non-null    int64
 1   Gender           400 non-null    int64
 2   Age              400 non-null    int64
 3   EstimatedSalary  400 non-null    int64
 4   Purchased        400 non-null    int64
dtypes: int64(5)
memory usage: 15.8 KB
```

```
df.describe()
```

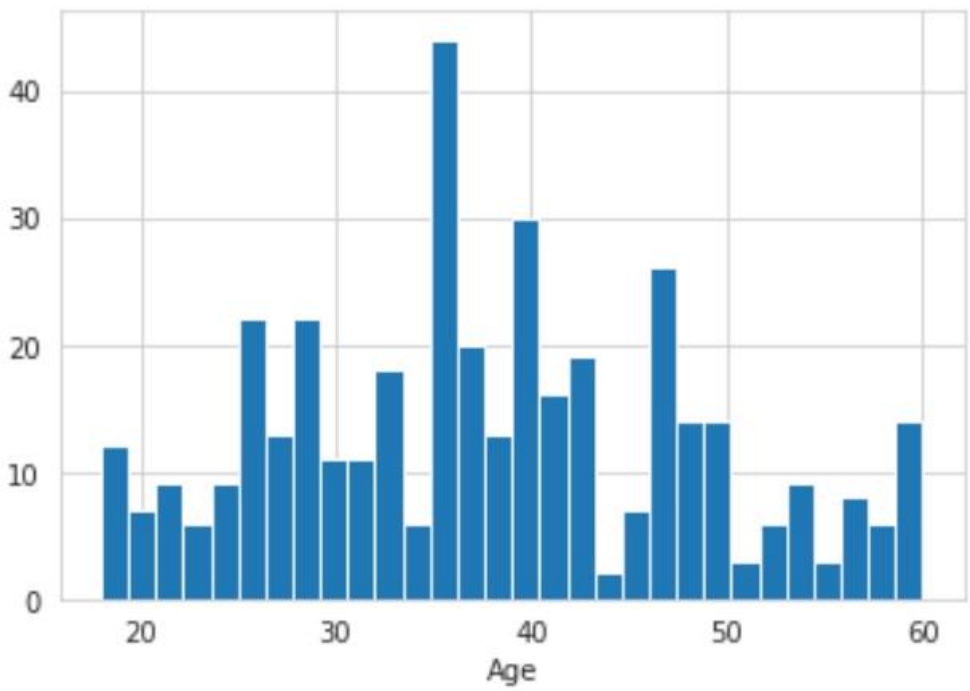
	User ID	Gender	Age	EstimatedSalary	Purchased
count	4.000000e+02	400.000000	400.000000	400.000000	400.000000
mean	1.569154e+07	0.490000	37.655000	69742.500000	0.357500
std	7.165832e+04	0.500526	10.482877	34096.960282	0.479864

```
min     1.556669e+07    0.000000   18.000000     15000.000000    0.000000
25%     1.562676e+07    0.000000   29.750000     43000.000000    0.000000
50%     1.569434e+07    0.000000   37.000000     70000.000000    0.000000
75%     1.575036e+07    1.000000   46.000000     88000.000000    1.000000
max     1.581524e+07    1.000000   60.000000    150000.000000    1.000000
```

Exploratory Data Analysis

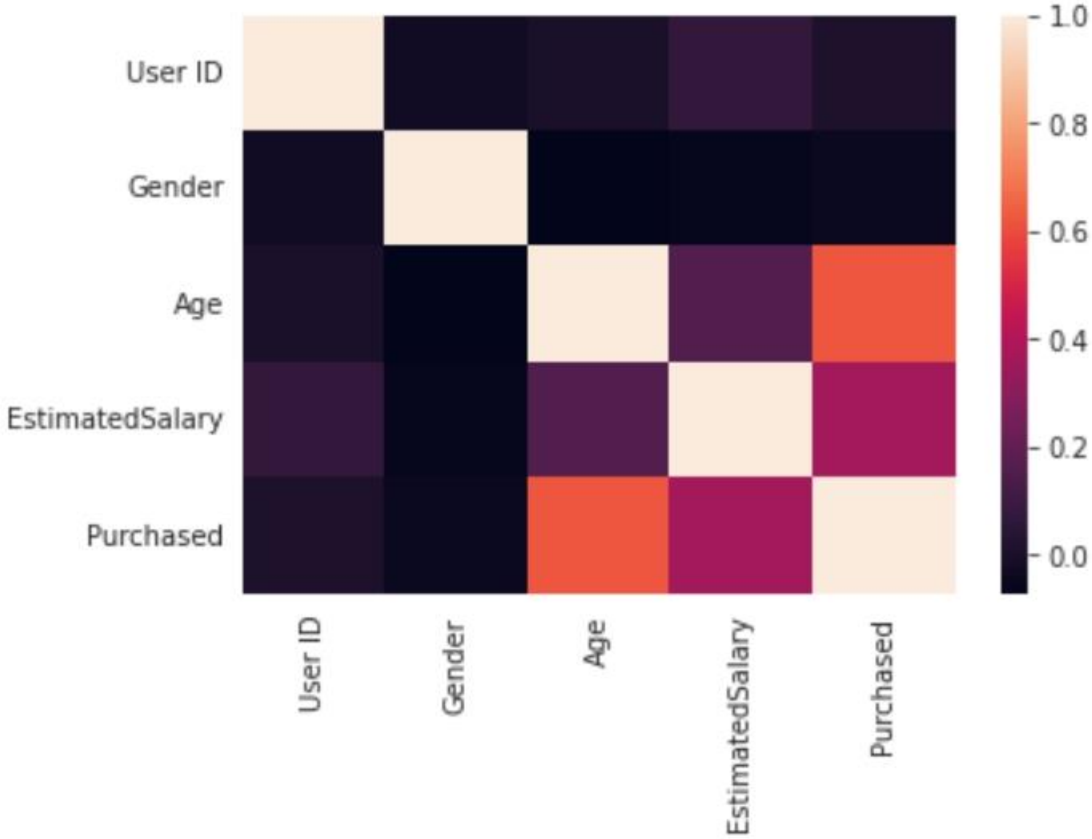
```
sns.set_style('whitegrid')
df['Age'].hist(bins=30)
plt.xlabel('Age')
```

```
Text(0.5, 0, 'Age')
```



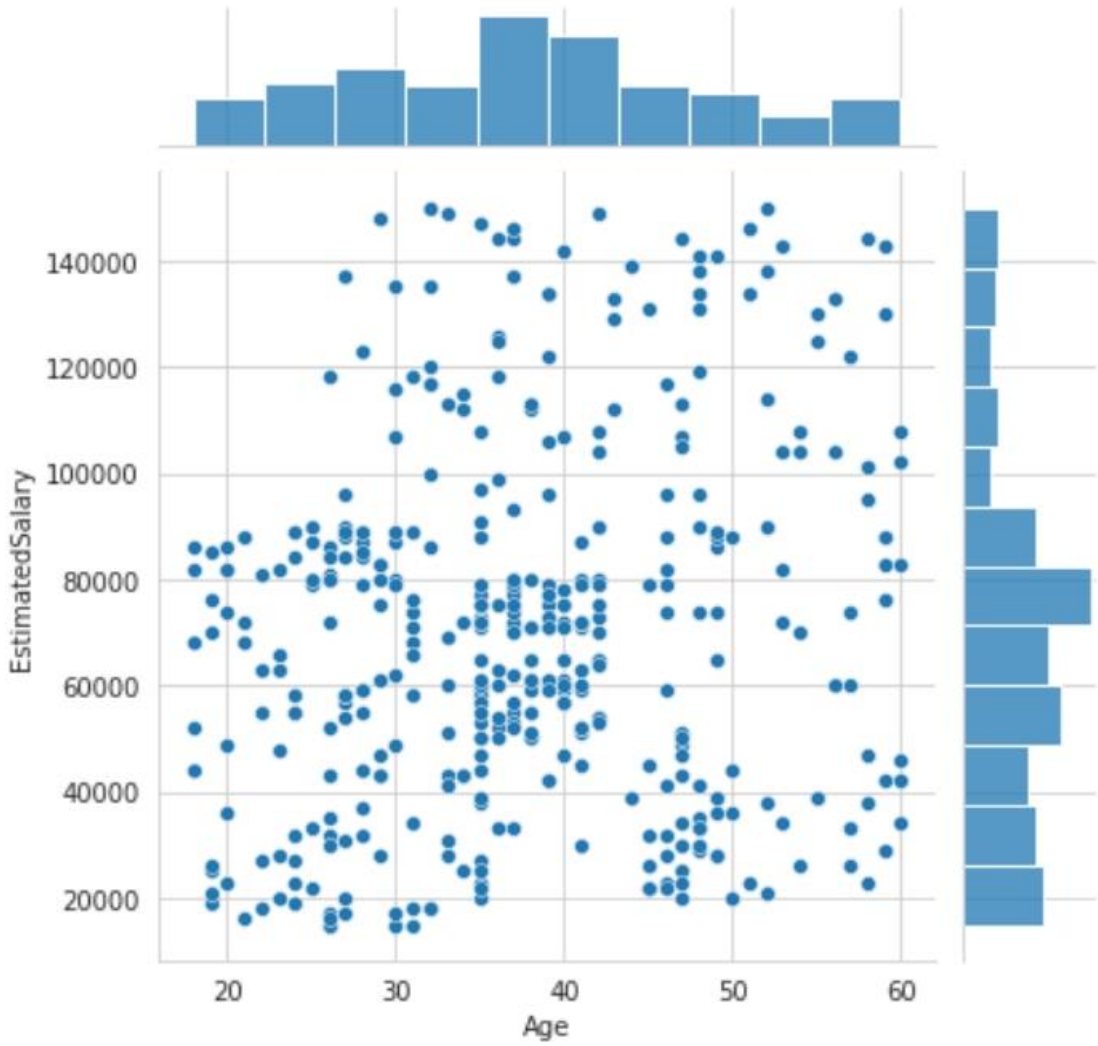

```
sns.heatmap(df.corr())
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8b04f88c70>
```

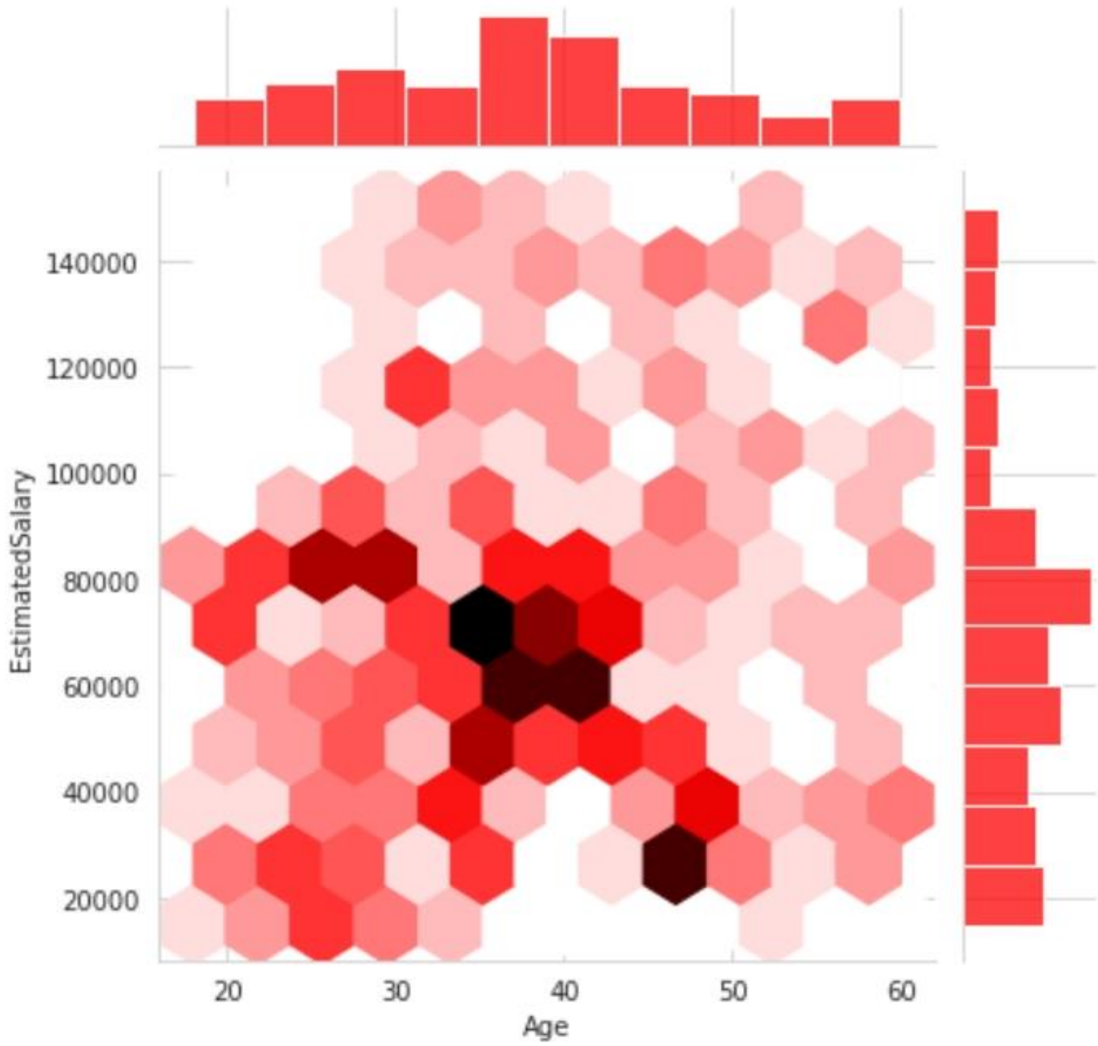



```
sns.jointplot(x='Age',y='EstimatedSalary',data=df)
```

```
<seaborn.axisgrid.JointGrid at 0x7f8b04a16070>
```




```
sns.jointplot(x='Age',y='EstimatedSalary',data=df,color='red',kind='hex');
```



Splitting the data

```
x=df.iloc[:,2:4]
y=df.iloc[:,-1]
```

```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)
```

Logistic Regression Model

```
lr=LogisticRegression()
lr.fit(x_train,y_train)
```

```
LogisticRegression()
```

Prediction and Evaluation

```
pred=lr.predict(x_test)
```

```
print(classification_report(y_test,pred,zero_division=0))
```

```
              precision    recall  f1-score   support

           0       0.64      1.00      0.78        77
           1       0.00      0.00      0.00        43

    accuracy                           0.64       120
   macro avg       0.32      0.50      0.39       120
weighted avg       0.41      0.64      0.50       120
```

```
confusion_matrix(y_test,pred)
```

```
array([[77,  0],
       [43,  0]])
```

```
accuracy_score(y_test,pred)
```

```
0.6416666666666667
```

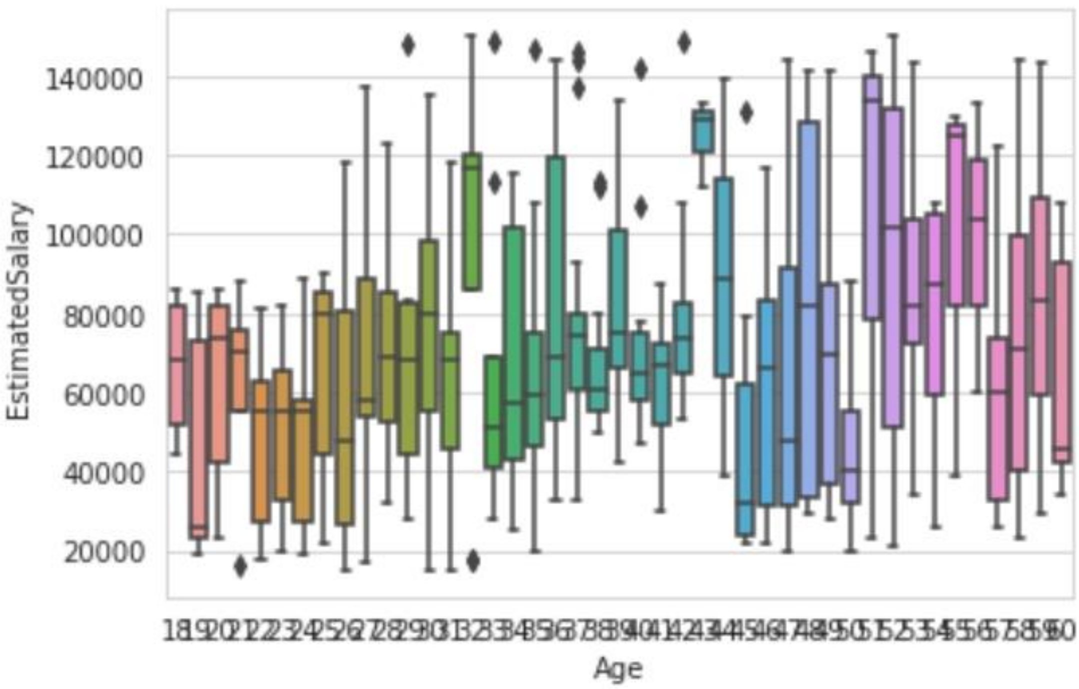
Linear Regression Model

```
num=df.select_dtypes(exclude=[object])
num.corr(method='pearson')
num.corr()
```

	User ID	Gender	Age	EstimatedSalary	Purchased
User ID	1.000000	-0.025249	-0.000721	0.071097	0.007120
Gender	-0.025249	1.000000	-0.073741	-0.060435	-0.042469
Age	-0.000721	-0.073741	1.000000	0.155238	0.622454
EstimatedSalary	0.071097	-0.060435	0.155238	1.000000	0.362083
Purchased	0.007120	-0.042469	0.622454	0.362083	1.000000

```
sns.boxplot(x="Age",y="EstimatedSalary",data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f8b020fc400>
```




```
x=df.iloc[:,2:3]
y=df.iloc[:,3:4]

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)

lir=LinearRegression()
lir.fit(x_train,y_train)
```

```
LinearRegression()
```

```
y_pred=lir.predict(x_test)
r2_score(y_test,y_pred)
```

```
0.0414978360689332
```

Multiple Linear Regression

```
x=df.iloc[:,1:3]
y=df.iloc[:,3:4]

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3)

mlr=LinearRegression()
mlr.fit(x_train,y_train)
```

```
LinearRegression()
```

```
y_pred=mlr.predict(x_test)
r2_score(y_test,y_pred)
```

```
0.016502206176493117
```

ACCURACY COMPARISIONS:

Logistic Regression: 64.2%, Linear Regression: 4.2%, Multiple Linear Regression: 1.7%

Experiment – 10

Aim: Build SVM model with various kernels and select best kernel for given dataset.

Context: In machine learning, **support vector machines** (**SVMs**, also **support vector networks**) are supervised learning models with associated learning algorithms that analyze data for classification and regression analysis. SVM maps training examples to points in space so as to maximise the width of the gap between the two categories. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Dataset Description: This data set dates from 1988 and consists of four databases: Cleveland, Hungary, Switzerland, and Long Beach V. It contains 76 attributes, including the predicted attribute, but all published experiments refer to using a subset of 14 of them. The "target" field refers to the presence of heart disease in the patient. It is integer valued 0 = no disease and 1 = disease.

Code & Output:

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

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import seaborn as sns

dataset = pd.read_csv('https://raw.githubusercontent.com/kb22/Heart-Diseas
e-Prediction/master/dataset.csv')

dataset
```

```
     age  sex  cp  trestbps  chol  fbs  restecg  thalach  exang  oldpeak
\
0     63    1   3       145   233    1        0      150      0      2.3
1     37    1   2       130   250    0        1      187      0      3.5
2     41    0   1       130   204    0        0      172      0      1.4
3     56    1   1       120   236    0        1      178      0      0.8
4     57    0   0       120   354    0        1      163      1      0.6
..   ...  ...  ..       ...   ...  ...      ...      ...    ...      ...
298   57    0   0       140   241    0        1      123      1      0.2
299   45    1   3       110   264    0        1      132      0      1.2
300   68    1   0       144   193    1        1      141      0      3.4
301   57    1   0       130   131    0        1      115      1      1.2
302   57    0   1       130   236    0        0      174      0      0.0

     slope  ca  thal  target
0        0   0     1       1
1        0   0     2       1
2        2   0     2       1
3        2   0     2       1
4        2   0     2       1
..     ...  ..   ...     ...
298      1   0     3       0
299      1   0     3       0
```

```
300      1   2     3       0
301      1   1     3       0
302      1   1     2       0

[303 rows x 14 columns]
```

```
dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 303 entries, 0 to 302
Data columns (total 14 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   age       303 non-null    int64  
 1   sex       303 non-null    int64  
 2   cp        303 non-null    int64  
 3   trestbps  303 non-null    int64  
 4   chol      303 non-null    int64  
 5   fbs       303 non-null    int64  
 6   restecg   303 non-null    int64  
 7   thalach   303 non-null    int64  
 8   exang     303 non-null    int64  
 9   oldpeak   303 non-null    float64
 10  slope     303 non-null    int64  
 11  ca        303 non-null    int64  
 12  thal      303 non-null    int64  
 13  target    303 non-null    int64  
dtypes: float64(1), int64(13)
memory usage: 33.3 KB
```

```
dataset.describe()
```

```
              age         sex          cp    trestbps        chol  
fbs  \
count  303.000000  303.000000  303.000000  303.000000  303.000000  303.000
000   
mean    54.366337    0.683168    0.966997  131.623762  246.264026    0.148
515   
std      9.082101    0.466011    1.032052   17.538143   51.830751    0.356
198   
min     29.000000    0.000000    0.000000   94.000000  126.000000    0.000
000   
25%     47.500000    0.000000    0.000000  120.000000  211.000000    0.000
000   
50%     55.000000    1.000000    1.000000  130.000000  240.000000    0.000
000   
75%     61.000000    1.000000    2.000000  140.000000  274.500000    0.000
000   
max     77.000000    1.000000    3.000000  200.000000  564.000000    1.000
000   

          restecg     thalach       exang     oldpeak       slope  
ca  \
```

```
count   303.000000  303.000000  303.000000  303.000000  303.000000  303.000
000
mean      0.528053  149.646865    0.326733    1.039604    1.399340    0.729
373
std       0.525860   22.905161    0.469794    1.161075    0.616226    1.022
606
min       0.000000   71.000000    0.000000    0.000000    0.000000    0.000
000
25%       0.000000  133.500000    0.000000    0.000000    1.000000    0.000
000
50%       1.000000  153.000000    0.000000    0.800000    1.000000    0.000
000
75%       1.000000  166.000000    1.000000    1.600000    2.000000    1.000
000
max       2.000000  202.000000    1.000000    6.200000    2.000000    4.000
000

             thal      target
count  303.000000  303.000000
mean     2.313531    0.544554
std      0.612277    0.498835
min      0.000000    0.000000
25%      2.000000    0.000000
50%      2.000000    1.000000
75%      3.000000    1.000000
max      3.000000    1.000000
```

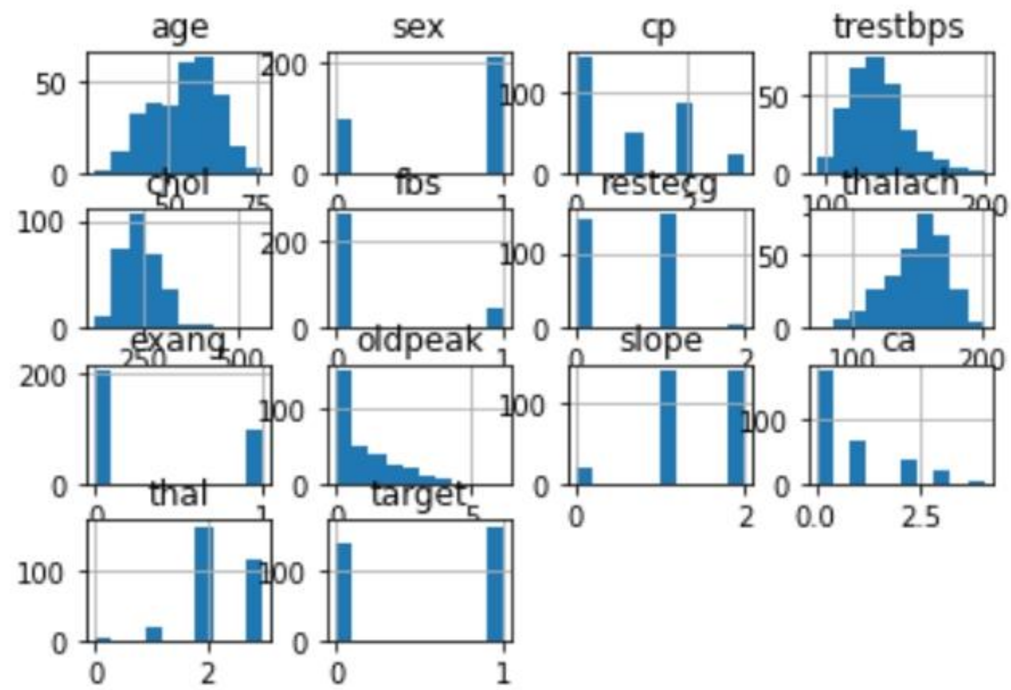
```
dataset.isnull().sum()
```

```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

```
dataset.hist()
```

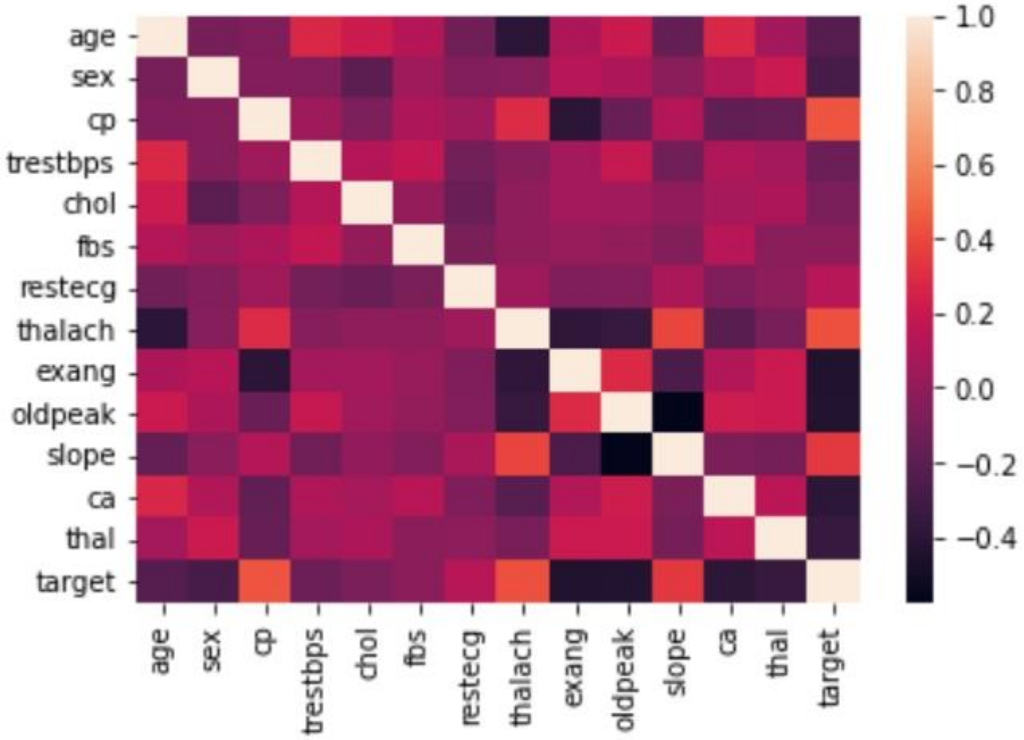
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x7f0144c3e910>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0144c0ed60>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0144bc81c0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0144b775b0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0144b249a0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0144b51cd0>,
```

```
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0144b51dc0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0144b0a250>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0144a659d0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0144a93df0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f0144a4d250>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f01449f9640>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x7f0144afb250>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f01449caf70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f01449842e0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x7f01449238b0>]]
,
      dtype=object)
```



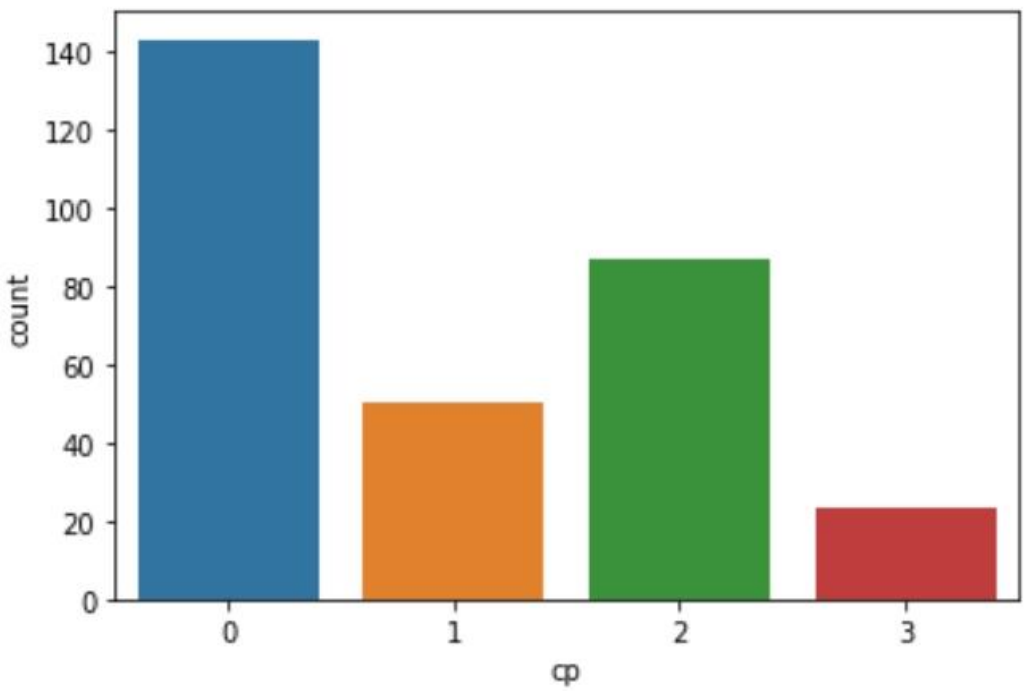

```
hm = dataset.corr()
sns.heatmap(hm)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f01447f84c0>
```




```
sns.countplot(x='cp', data=dataset)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0141979c70>
```




```
y = dataset['target']
X = dataset.drop(['target'], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.33
, random_state = 0)
```

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
from sklearn.svm import SVC
from sklearn import preprocessing,svm
```

```
svm = SVC(kernel='linear', C=1, random_state=50)
svm.fit(X_train, y_train)
```

```
SVC(C=1, kernel='linear', random_state=50)
```

```
from sklearn.metrics import accuracy_score
```

```
y_pred = svm.predict(X_test)
```

```
accuracy_score(y_test,y_pred)
```

```
0.81
```