```
import pandas as pd

df=pd.read_csv(r"C:\Users\rohan\OneDrive\Desktop\Python Datasets\Heart.csv")

df.head()
```

	Unnamed: 0	Age	Sex	ChestPain	RestBP	Chol	Fbs	RestECG	MaxHR	ExAng	Oldpeak	S
0	1	63	1	typical	145	233	1	2	150	0	2.3	
1	2	67	1	asymptomatic	160	286	0	2	108	1	1.5	
2	3	67	1	asymptomatic	120	229	0	2	129	1	2.6	
3	4	37	1	nonanginal	130	250	0	0	187	0	3.5	

```
#shape of data set
df.shape
```

```
(303, 15)
```

```
#finding missing values
df.isnull().sum()
```

```
Unnamed: 0    0
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
```

```
#data type of each column
df.dtypes
```

```
Unnamed: 0      int64
Age             int64
Sex             int64
```

```
ChestPain      object
RestBP          int64
Chol            int64
Fbs             int64
RestECG         int64
MaxHR           int64
ExAng           int64
Oldpeak       float64
Slope           int64
Ca            float64
Thal           object
AHD            object
dtype: object
```

```
#finding out zero's
df[df==0].count()
```

```
Unnamed: 0      0
Age             0
Sex            97
ChestPain       0
RestBP          0
Chol            0
Fbs           258
RestECG       151
MaxHR           0
ExAng         204
Oldpeak        99
Slope           0
Ca            176
Thal            0
AHD             0
dtype: int64
```

```
#finding out mean age of patients
df['Age'].mean()
```

```
54.43894389438944
```

```
#extracting only Age, Sex, ChestPain, RestBP, Chol columns
newdf = df[['Age', 'Sex', 'ChestPain', 'RestBP', 'Chol']]
newdf.head()
```

```
          Age  Sex    ChestPain  RestBP  Chol
```

```
#Randomly dividing dataset in training (75%) and testing (25%)
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, random_state = 0, test_size = 0.25)
```

```
      2    67    1   asymptomatic      120    229
```

```
train.shape
```

```
     (227, 15)
```

```
test.shape
```

```
     (76, 15)
```

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

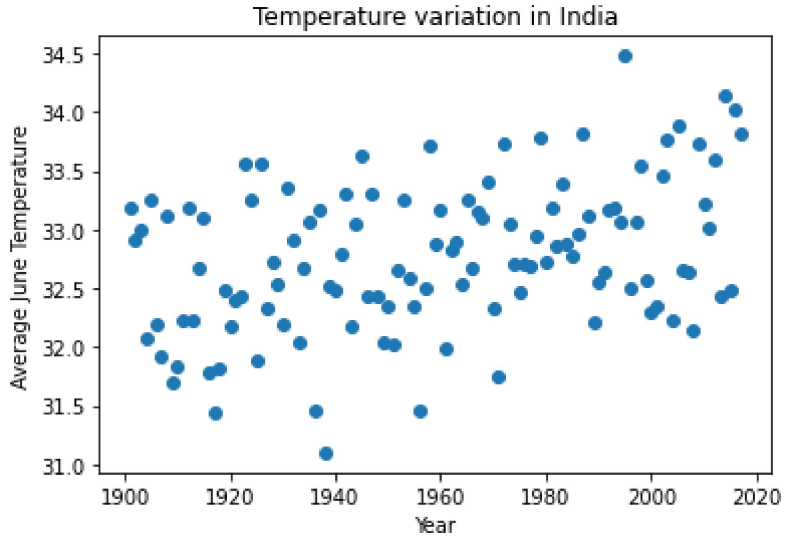
```
df = pd.read_csv(r"C:\Users\rohan\OneDrive\Desktop\Python Datasets\temperatures.csv")
df.head()
```

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
0	1901	22.40	24.14	29.07	31.91	33.41	33.18	31.21	30.39	30.47	29.97	27.31	24.49
1	1902	24.93	26.58	29.77	31.78	33.73	32.91	30.92	30.73	29.80	29.12	26.31	24.04
2	1903	23.44	25.03	27.83	31.39	32.91	33.00	31.34	29.98	29.85	29.04	26.08	23.65
3	1904	22.50	24.73	28.21	32.02	32.64	32.07	30.36	30.09	30.04	29.20	26.36	23.63

```
x = df['YEAR'] #input data
y = df['JUN'] #output data
```

```
plt.title("Temperature variation in India")
plt.xlabel('Year')
plt.ylabel('Average June Temperature')
plt.scatter(x, y)
```

```
<matplotlib.collections.PathCollection at 0x2deb0939580>
```




```
x.shape
```

```
(117,)
```

```
x = x.values
```

```
x = x.reshape(117, 1)
```

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

```
regressor = LinearRegression()
```

```
regressor.fit(x, y)
```

```
LinearRegression()
```

```
# Performing the prediction for unseen data
regressor.predict([[2030]])
```

```
array([33.26706749])
```

```
predicted_values = regressor.predict(x)
```

```
# Mean Absolute Error
from sklearn.metrics import mean_absolute_error
mean_absolute_error(y, predicted_values)
```

```
0.48168799010531976
```

```
# Mean Squared Error
from sklearn.metrics import mean_squared_error
mean_squared_error(y, predicted_values)
```

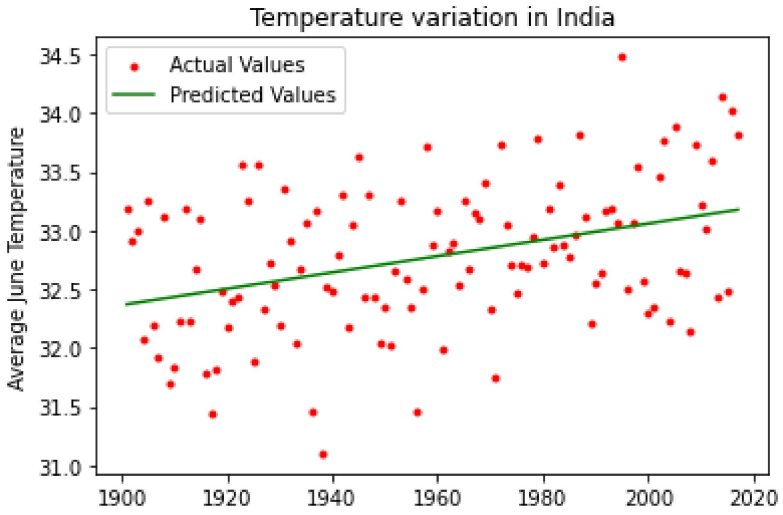
```
0.3424789478589651
```

```
# R-Square metrics
from sklearn.metrics import r2_score
r2_score(y, predicted_values)
```

```
0.1382651229137435
```

```
# Visualizing simple regression model
plt.title("Temperature variation in India")
plt.xlabel('Year')
plt.ylabel('Average June Temperature')
plt.scatter(x, y, label='Actual Values', color = 'r', marker='.')
plt.plot(x, predicted_values, label='Predicted Values', color = 'g')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x2debf4728b0>
```



Colab paid products - Cancel contracts here

```
import pandas as pd
import seaborn as sns
```

```
df=pd.read_csv(r"C:\Users\rohan\OneDrive\Desktop\Python Datasets\Admission_Predict.csv")
df.head()
```

	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research	Chance of Admit
0	1	337	118	4	4.5	4.5	9.65	1	0.92
1	2	324	107	4	4.0	4.5	8.87	1	0.76
2	3	316	104	3	3.0	3.5	8.00	1	0.72
3	4	322	110	3	3.5	2.5	8.67	1	0.80
4	5	314	103	2	2.0	3.0	8.21	0	0.65

```
df.shape
```

```
(400, 9)
```

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

```
Serial No.          0
GRE Score           0
TOEFL Score         0
University Rating   0
SOP                 0
LOR                 0
CGPA                0
Research            0
Chance of Admit     0
dtype: int64
```

```
from sklearn.preprocessing import Binarizer
bi = Binarizer(threshold=0.75)
df['Chance of Admit '] = bi.fit_transform(df[['Chance of Admit ']])
df.head()
```

Serial GRE TOEFL University . Chance of

```
x = df.drop('Chance of Admit ', axis=1)
y = df['Chance of Admit ']
x.head()
```

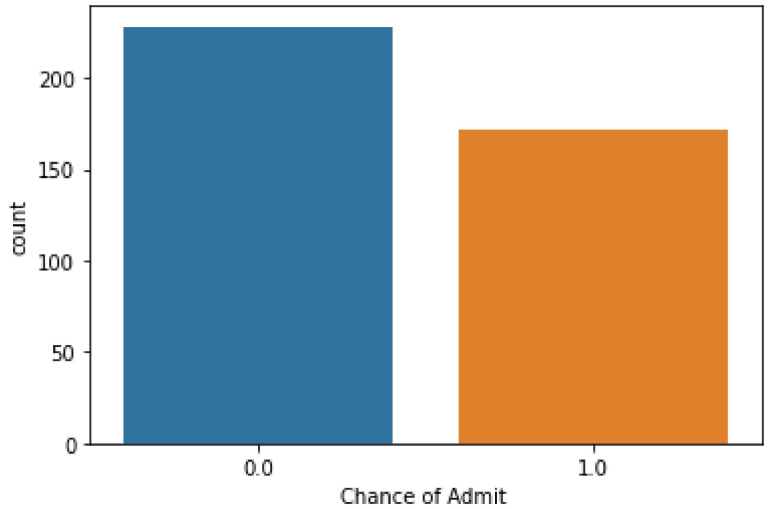
	Serial No.	GRE Score	TOEFL Score	University Rating	SOP	LOR	CGPA	Research
0	1	337	118	4	4.5	4.5	9.65	1
1	2	324	107	4	4.0	4.5	8.87	1
2	3	316	104	3	3.0	3.5	8.00	1
3	4	322	110	3	3.5	2.5	8.67	1
4	5	314	103	2	2.0	3.0	8.21	0

```
y.astype('int')
```

```
0      1
1      1
2      0
3      1
4      0
      ..
395    1
396    1
397    1
398    0
399    1
Name: Chance of Admit , Length: 400, dtype: int32
```

```
sns.countplot(x=y)
```

```
<AxesSubplot:xlabel='Chance of Admit ', ylabel='count'>
```




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

```
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=0, train_size=0.75)
```

```
from sklearn.tree import DecisionTreeClassifier
classifier = DecisionTreeClassifier(random_state=0)
classifier.fit(x_train, y_train)
```

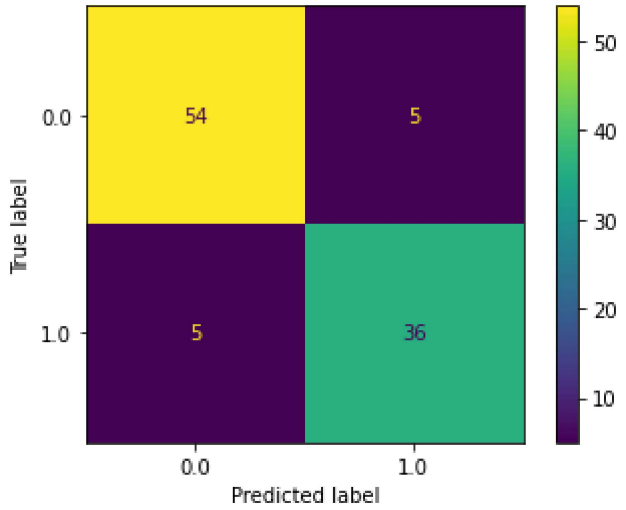
```
DecisionTreeClassifier(random_state=0)
```

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

```
result = pd.DataFrame({
    'actual' : y_test,
    'predicted' : y_pred
})
```

```
from sklearn.metrics import ConfusionMatrixDisplay, accuracy_score
from sklearn.metrics import classification_report
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
```

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x1a4243effd0>
```




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

```
0.9
```

```
print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.92      0.92      0.92        59
         1.0       0.88      0.88      0.88        41

    accuracy                           0.90       100
   macro avg       0.90      0.90      0.90       100
```

```
weighted avg       0.90      0.90      0.90       100
```

Colab paid products - Cancel contracts here

```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib.lines import Line2D
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mou
```

```
cust = pd.read_csv("gdrive/My Drive/Mall_Customers.csv")
cust.head()
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
cust.shape
```

```
(200, 5)
```

```
cust.isnull().sum()
```

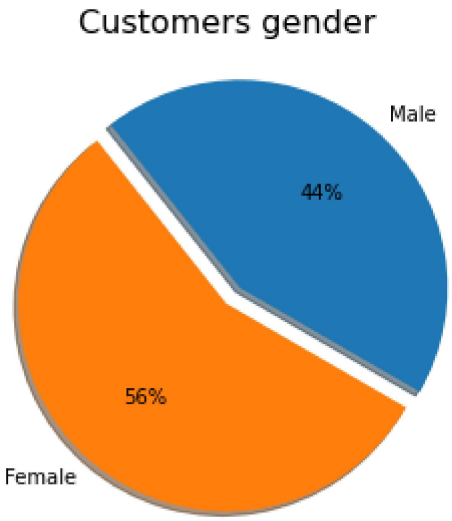
```
CustomerID                0
Genre                     0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

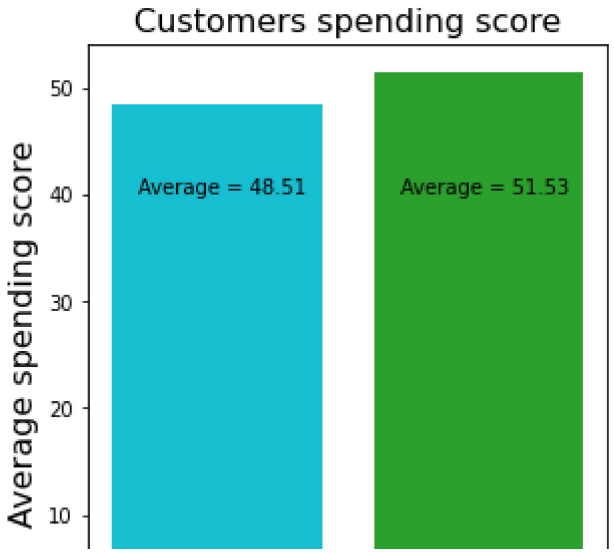
```
cust.rename(columns = {"Genre":"Gender"}, inplace = True)
cust.drop(labels = 'CustomerID', axis = 1 , inplace = True)
```

```
cust["Gender"].replace({"Male":1, "Female":0}, inplace = True)
```

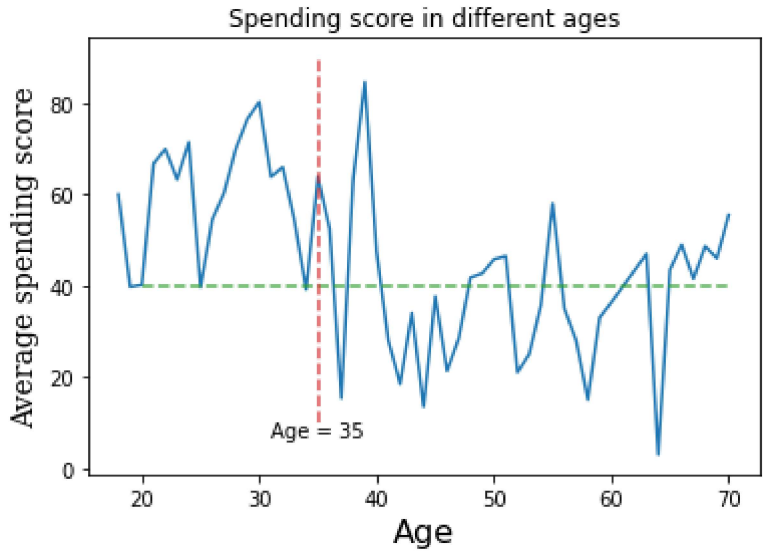
```
sns.heatmap(data = cust.corr(), annot = True, fmt = ".2f", cmap = "cividis_r")
font = {"family":"Sherif", "size":16}
```


```
plt.subplots_adjust(left = 1, bottom = 1,right = 2.5, top = 2, wspace = 0.5, hspace = None)
plt.subplot(1,2,1)
plt.pie(x = [len(cust[cust.Gender == 1]) , len(cust[cust.Gender == 0])] , labels = ['Male' ,
plt.title("Customers gender", fontdict = font)
plt.subplot(1,2,2)
male_avg_score = cust[cust.Gender == 1]['Spending Score (1-100)'].mean()
female_avg_score = cust[cust.Gender == 0]['Spending Score (1-100)'].mean()
plt.bar(x = ['Male' , 'Female'] , height = [male_avg_score , female_avg_score]
, color = ['tab:cyan' , 'tab:green'])
plt.title('Customers spending score' , fontdict = font)
plt.ylabel('Average spending score' , fontdict = font)
plt.xlabel('Gender' , fontdict = font)
plt.text(-0.3 , 40 , 'Average = {:.2f}'.format(male_avg_score))
plt.text(0.7 , 40 , 'Average = {:.2f}'.format(female_avg_score))
plt.show()
```



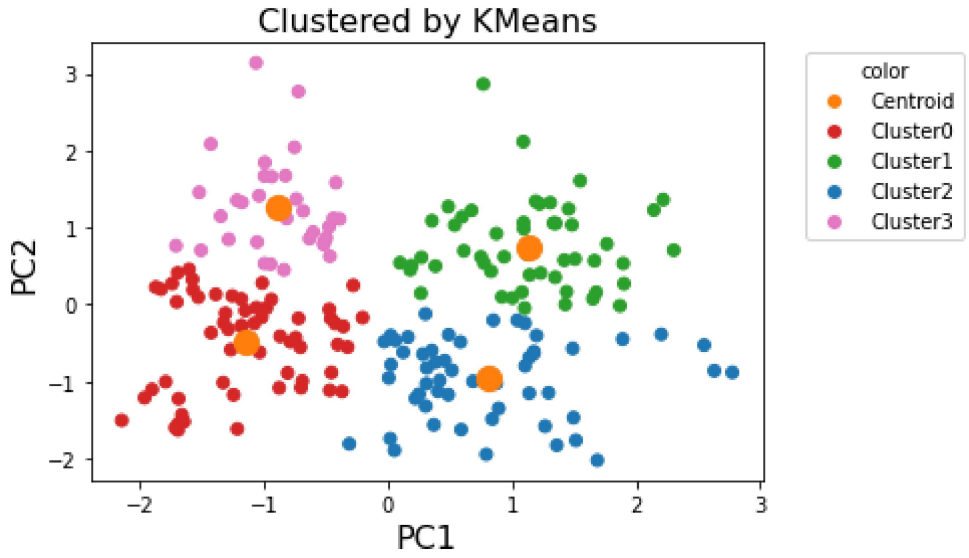



```
age_list = cust.Age.unique()
age_list.sort()
avg_list = []
for age in age_list:
  avg_list.append(cust[cust.Age == age]['Spending Score (1-100)'].mean())
plt.plot(age_list,avg_list)
plt.xlabel('Age' , fontdict = font)
plt.ylabel('Average spending score' , fontdict = {'family':'serif' , 'size':14
})
plt.title('Spending score in different ages')
plt.plot([20,70] , [40,40] , linestyle = '--' , c = 'tab:green' , alpha = 0.8)
plt.plot([35,35] , [10,90] , linestyle = '--' , c = 'tab:red' , alpha = 0.8)
plt.text(31,7,'Age = 35')
plt.show()
```




```
kmeans = KMeans(n_clusters = 4 , init = 'k-means++' , random_state = 1)
kmeans.fit(data_pca)
cluster_id = kmeans.predict(data_pca)
result_data = pd.DataFrame()
```

```
result_data = pd.DataFrame()
result_data['PC1'] = data_pca[:,0]
result_data['PC2'] = data_pca[:,1]
result_data['ClusterID'] = cluster_id
cluster_colors = {0:'tab:red' , 1:'tab:green' , 2:'tab:blue' , 3:'tab:pink'}
cluster_dict = {'Centroid':'tab:orange','Cluster0':'tab:red' , 'Cluster1':'tab:green', 'Clust
plt.scatter(x = result_data['PC1'] , y = result_data['PC2'] , c = result_data[
'ClusterID'].map(cluster_colors))
handles = [Line2D([0], [0], marker='o', color='w', markerfacecolor=v, label=k,
markersize=8) for k, v in cluster_dict.items()]
plt.legend(title='color', handles=handles, bbox_to_anchor=(1.05, 1), loc='upper left')
plt.scatter(x = kmeans.cluster_centers_[:,0] , y = kmeans.cluster_centers_[:,1
] , marker = 'o' , c = 'tab:orange', s = 150 , alpha = 1)
plt.title("Clustered by KMeans" , fontdict = font)
plt.xlabel("PC1" , fontdict = font)
plt.ylabel("PC2" , fontdict = font)
plt.show()
```



```
import pandas as pd
import csv
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules


dataset = []
with open('Market_Basket_Optimisation.csv') as file:
  reader = csv.reader(file, delimiter=',')
  for row in reader:
    dataset += [row]



dataset[0:10]
```

```
[['shrimp',
  'almonds',
  'avocado',
  'vegetables mix',
  'green grapes',
  'whole weat flour',
  'yams',
  'cottage cheese',
  'energy drink',
  'tomato juice',
  'low fat yogurt',
  'green tea',
  'honey',
  'salad',
  'mineral water',
  'salmon',
  'antioxydant juice',
  'frozen smoothie',
  'spinach',
  'olive oil'],
 ['burgers', 'meatballs', 'eggs'],
 ['chutney'],
 ['turkey', 'avocado'],
 ['mineral water', 'milk', 'energy bar', 'whole wheat rice', 'green tea'],
 ['low fat yogurt'],
 ['whole wheat pasta', 'french fries'],
 ['soup', 'light cream', 'shallot'],
 ['frozen vegetables', 'spaghetti', 'green tea'],
 ['french fries']]
```

```
len(dataset)
```

```
7501
```

```
te = TransactionEncoder()
x= te.fit_transform(dataset)
x
```

```
array([[False,  True,  True, ...,  True, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       ...,
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False, False, False],
       [False, False, False, ..., False,  True, False]])
```

```
df = pd.DataFrame(x, columns=te.columns_)
```

```
#1. frequent itemsets
freq_itemset = apriori(df, min_support=0.01, use_colnames=True)
freq_itemset
```

	support	itemsets
0	0.020397	(almonds)
1	0.033329	(avocado)
2	0.010799	(barbecue sauce)
3	0.014265	(black tea)
4	0.011465	(body spray)
...	...	...
252	0.011065	(milk, ground beef, mineral water)
253	0.017064	(spaghetti, ground beef, mineral water)
254	0.015731	(spaghetti, milk, mineral water)
255	0.010265	(spaghetti, olive oil, mineral water)
256	0.011465	(pancakes, spaghetti, mineral water)

257 rows × 2 columns

```
#Find the rules
rules = association_rules(freq_itemset, metric='confidence', min_threshold=0.25)
rules = rules[['antecedents', 'consequents','support','confidence']]
rules.head()
```

	antecedents	consequents	support	confidence
0	(avocado)	(mineral water)	0.011598	0.348000
1	(burgers)	(eggs)	0.028796	0.330275
2	(burgers)	(french fries)	0.021997	0.252294

```
rules[rules['antecedents']=={'cake'}]['consequents']
```

```
4    (mineral water)
Name: consequents, dtype: object
```

Colab paid products - Cancel contracts here

✓ 0s completed at 1:09 PM

```
import pandas as pd
import numpy as np


df = pd.read_csv(r"C:\Users\rohan\OneDrive\Desktop\Python Datasets\pima-indians-diabetes.csv"
df.head()
```

	6	148	72	35	0	33.6	0.627	50	1
0	1	85	66	29	0	26.6	0.351	31	0
1	8	183	64	0	0	23.3	0.672	32	1
2	1	89	66	23	94	28.1	0.167	21	0
3	0	137	40	35	168	43.1	2.288	33	1
4	5	116	74	0	0	25.6	0.201	30	0

```
x= df.iloc[:,:8]
y= df.iloc[:,8]
```

+ Code + Text

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
```

```
#create model
model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))

#hidden layers
model.add(Dense(8, activation='relu'))
model.add(Dense(8, activation='relu'))

#output layer
model.add(Dense(1, activation='sigmoid'))


#compile model
model.compile(loss='binary_crossentropy', optimizer='adam',
              metrics=['accuracy'])


#train model
model.fit(x, y, epochs = 100, batch_size=10)
```

```
Epoch 1/100
77/77 [==============================] - 1s 1ms/step - loss: 3.6775 - accuracy: 0.52
Epoch 2/100
77/77 [==============================] - 0s 1ms/step - loss: 0.9684 - accuracy: 0.58
Epoch 3/100
77/77 [==============================] - 0s 1ms/step - loss: 0.8022 - accuracy: 0.60
Epoch 4/100
77/77 [==============================] - 0s 1ms/step - loss: 0.7610 - accuracy: 0.63
Epoch 5/100
77/77 [==============================] - 0s 1ms/step - loss: 0.7086 - accuracy: 0.64
Epoch 6/100
77/77 [==============================] - 0s 1ms/step - loss: 0.6650 - accuracy: 0.64
Epoch 7/100
77/77 [==============================] - 0s 1ms/step - loss: 0.6642 - accuracy: 0.65
Epoch 8/100
77/77 [==============================] - 0s 1ms/step - loss: 0.6600 - accuracy: 0.67
Epoch 9/100
77/77 [==============================] - 0s 1ms/step - loss: 0.6835 - accuracy: 0.64
Epoch 10/100
77/77 [==============================] - 0s 1ms/step - loss: 0.6275 - accuracy: 0.69
Epoch 11/100
77/77 [==============================] - 0s 1ms/step - loss: 0.6416 - accuracy: 0.65
Epoch 12/100
77/77 [==============================] - 0s 1ms/step - loss: 0.7130 - accuracy: 0.64
Epoch 13/100
77/77 [==============================] - 0s 1ms/step - loss: 0.6222 - accuracy: 0.69
Epoch 14/100
77/77 [==============================] - 0s 1ms/step - loss: 0.6081 - accuracy: 0.70
Epoch 15/100
77/77 [==============================] - 0s 1ms/step - loss: 0.5889 - accuracy: 0.69
Epoch 16/100
77/77 [==============================] - 0s 1ms/step - loss: 0.5949 - accuracy: 0.69
Epoch 17/100
77/77 [==============================] - 0s 1ms/step - loss: 0.6329 - accuracy: 0.69
Epoch 18/100
77/77 [==============================] - 0s 1ms/step - loss: 0.5820 - accuracy: 0.71
Epoch 19/100
77/77 [==============================] - 0s 1ms/step - loss: 0.5829 - accuracy: 0.70
Epoch 20/100
77/77 [==============================] - 0s 1ms/step - loss: 0.5822 - accuracy: 0.70
Epoch 21/100
77/77 [==============================] - 0s 1ms/step - loss: 0.6209 - accuracy: 0.67
Epoch 22/100
77/77 [==============================] - 0s 1ms/step - loss: 0.5659 - accuracy: 0.73
Epoch 23/100
77/77 [==============================] - 0s 2ms/step - loss: 0.5715 - accuracy: 0.72
Epoch 24/100
77/77 [==============================] - 0s 1ms/step - loss: 0.6128 - accuracy: 0.69
Epoch 25/100
77/77 [==============================] - 0s 1ms/step - loss: 0.5899 - accuracy: 0.68
Epoch 26/100
77/77 [==============================] - 0s 1ms/step - loss: 0.5763 - accuracy: 0.70
Epoch 27/100
77/77 [==============================] - 0s 1ms/step - loss: 0.5665 - accuracy: 0.71
Epoch 28/100
77/77 [==============================] - 0s 1ms/step - loss: 0.5881 - accuracy: 0.71
```

```
Epoch 29/100
77/77 [==============================] - 0s 1ms/step - loss: 0.5756 - accuracy: 0.71
```

```
#evaluate
model.evaluate(x,y)
```

```
24/24 [==============================] - 0s 1ms/step - loss: 0.4629 - accuracy: 0.7771
[0.46289893984794617, 0.7770534753799438]
```

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 12)                108

 dense_1 (Dense)             (None, 8)                 104

 dense_2 (Dense)             (None, 8)                 72

 dense_3 (Dense)             (None, 1)                 9

=================================================================
Total params: 293
Trainable params: 293
Non-trainable params: 0
_________________________________________________________________
```