```
# This Python 3 environment comes with many helpful analytics
libraries installed
# It is defined by the kaggle/python Docker image:
https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/)
that gets preserved as output when you create a version using "Save &
Run All"
# You can also write temporary files to /kaggle/temp/, but they won't
be saved outside of the current session
```

```
/kaggle/input/kaartest1/kaar test.csv
/kaggle/input/kaartest1/kaar train.csv
```

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/"
directory
# For example, running this (by clicking run or pressing Shift+Enter)
will list all files under the input directory

import os
for dirname, _, filenames in os.walk('../input/kaartest1'):
    for filename in filenames:
        print(os.path.join(dirname, filename))
```

```
../input/kaartest1/kaar test.csv
../input/kaartest1/kaar train.csv
```

```
df = pd.read_csv('../input/kaartest1/kaar train.csv')

df.columns
```

```
Index(['Gender', 'Age', 'Purchase Amount', 'Sales Amount'],
dtype='object')
```

```
df_orig = df.copy()
```

```
df.head()
```

```
   Gender  Age  Purchase Amount  Sales Amount
0    Male   18               15             1
1    Male   18               15             1
2  Female   18               16             3
3  Female   18               16             4
4  Female   19               17             4
```

```
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier
from sklearn.ensemble import RandomForestRegressor,
RandomForestClassifier
from xgboost import XGBRegressor, XGBClassifier
from sklearn.metrics import mean_absolute_error, accuracy_score,
classification_report,confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier,
KNeighborsRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import
confusion_matrix,accuracy_score,roc_curve,classification_report
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import
GridSearchCV,RandomizedSearchCV,KFold,StratifiedKFold
import pandas_profiling as pp
import warnings
warnings.filterwarnings('ignore')
import missingno as msno #Visualize null
```

```
sns.set_style('ticks') #No grid with ticks
print(sns.__version__)
```

```
0.11.1
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype
```

```
---  ------           --------------  -----
 0   Gender           400 non-null    object
 1   Age              400 non-null    int64
 2   Purchase Amount  400 non-null    int64
 3   Sales Amount     400 non-null    int64
dtypes: int64(3), object(1)
memory usage: 12.6+ KB
```

```
df.isna().any()
```

```
Gender             False
Age                False
Purchase Amount    False
Sales Amount       False
dtype: bool
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 400 entries, 0 to 399
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Gender           400 non-null    object
 1   Age              400 non-null    int64
 2   Purchase Amount  400 non-null    int64
 3   Sales Amount     400 non-null    int64
dtypes: int64(3), object(1)
memory usage: 12.6+ KB
```

```
cols=['Gender','Age','Purchase Amount','Sales Amount']
for i in cols:
  print(df[i].value_counts())
```

```
Female    224
Male      176
Name: Gender, dtype: int64
32    22
35    18
31    16
19    16
30    14
49    14
40    12
38    12
36    12
47    12
23    12
27    12
20    10
48    10
```

```
21    10
34    10
50    10
29    10
28     8
24     8
54     8
67     8
59     8
18     8
68     6
60     6
46     6
43     6
45     6
22     6
25     6
39     6
37     6
33     6
58     4
66     4
65     4
63     4
26     4
57     4
44     4
53     4
52     4
51     4
41     4
42     4
70     4
56     2
55     2
64     2
69     2
Name: Age, dtype: int64
54     24
78     24
48     12
71     12
63     12
       ..
58      4
59      4
16      4
64      4
137     4
Name: Purchase Amount, Length: 64, dtype: int64
```

```
42    16
55    14
46    12
73    12
75    10
      ..
63     2
34     2
44     2
45     2
99     2
Name: Sales Amount, Length: 84, dtype: int64
```

```
new_data = df.dropna()
new_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 400 entries, 0 to 399
Data columns (total 4 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Gender           400 non-null    object
 1   Age              400 non-null    int64
 2   Purchase Amount  400 non-null    int64
 3   Sales Amount     400 non-null    int64
dtypes: int64(3), object(1)
memory usage: 15.6+ KB
```

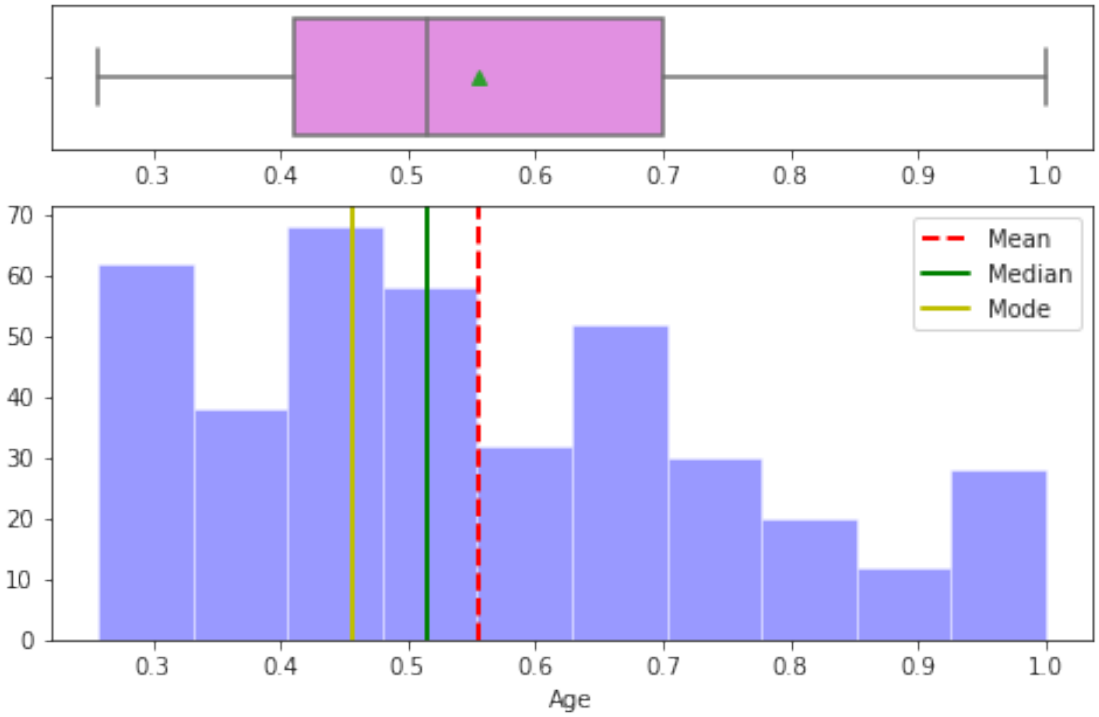
```
labelencoder = LabelEncoder()

df_max_scaled = new_data.copy()

## FEATURE ENGINEERING
df_max_scaled = df_max_scaled.astype({
    'Gender' : 'category'
})

cat_cols = [i for i  in df_max_scaled.columns if
df_max_scaled[i].dtype not in ['int64', 'float64']]

for col in cat_cols:
  df_max_scaled[col + "-cat"] =
labelencoder.fit_transform(df_max_scaled[col])

num_cols = [col for col in df_max_scaled.columns if
df_max_scaled[col].dtype in ['int', 'float']]

for i in cat_cols:
  df_max_scaled.drop([i], axis= 1, inplace= True)
```

```
for col in num_cols:
  df_max_scaled[col] = df_max_scaled[col] /
df_max_scaled[col].abs().max()

df_max_scaled.head()
```

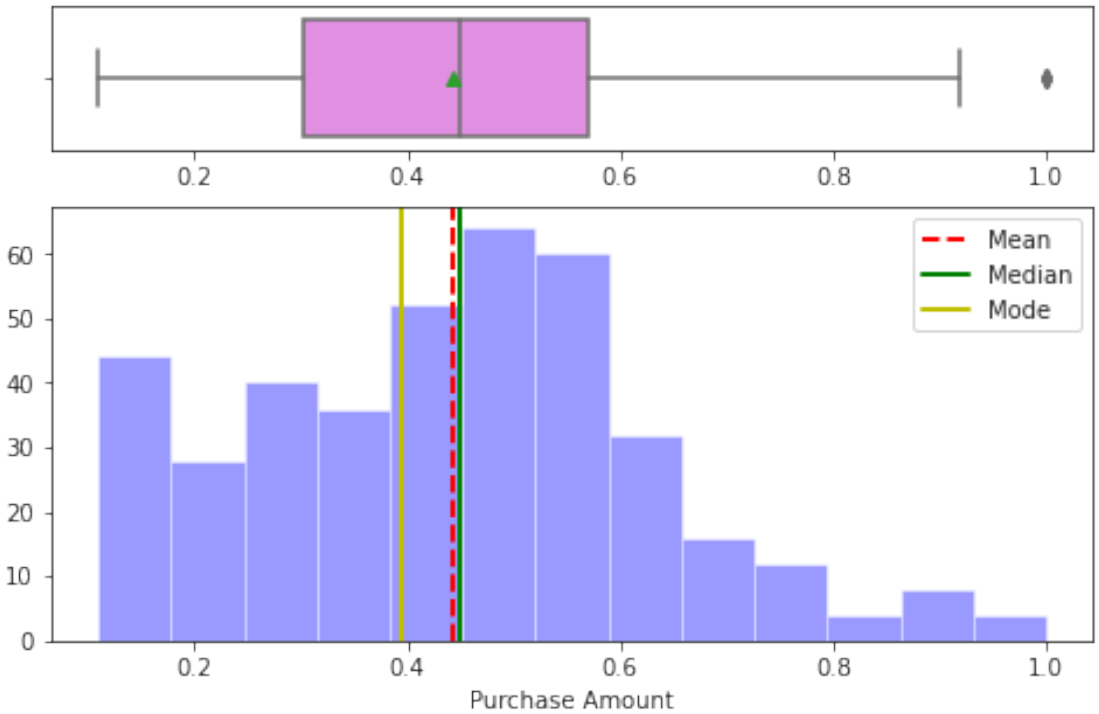
	Age	Purchase Amount	Sales Amount	Gender-cat
0	0.257143	0.109489	0.010101	1.0
1	0.257143	0.109489	0.010101	1.0
2	0.257143	0.116788	0.030303	0.0
3	0.257143	0.116788	0.040404	0.0
4	0.271429	0.124088	0.040404	0.0

```
num_cols = [col for col in df_max_scaled.columns if
df_max_scaled[col].dtype in ['int64','float64']]

cat_cols = [col for col in df_max_scaled.columns if
df_max_scaled[col].dtype not in ['int64', 'float64']]

def dist_box(data):
 # function plots a combined graph for univariate analysis of
continous variable
 #to check spread, central tendency , dispersion and outliers
    Name=data.name.upper()
    fig,(ax_box,ax_dis)  =plt.subplots(2,1,gridspec_kw =
{"height_ratios": (.25, .75)},figsize=(8, 5))
    mean=data.mean()
    median=data.median()
    mode=data.mode().tolist()[0]
    fig.suptitle("SPREAD OF DATA FOR "+ Name  , fontsize=18,
fontweight='bold')
    sns.boxplot(x=data,showmeans=True,
orient='h',color="violet",ax=ax_box)
    ax_box.set(xlabel='')
    sns.distplot(data,kde=False,color='blue',ax=ax_dis)
    ax_dis.axvline(mean, color='r', linestyle='--',linewidth=2)
    ax_dis.axvline(median, color='g', linestyle='-',linewidth=2)
    ax_dis.axvline(mode, color='y', linestyle='-',linewidth=2)
    plt.legend({'Mean':mean,'Median':median,'Mode':mode})

import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns

for i in range(len(num_cols)):
    dist_box(df_max_scaled[num_cols[i]])
```

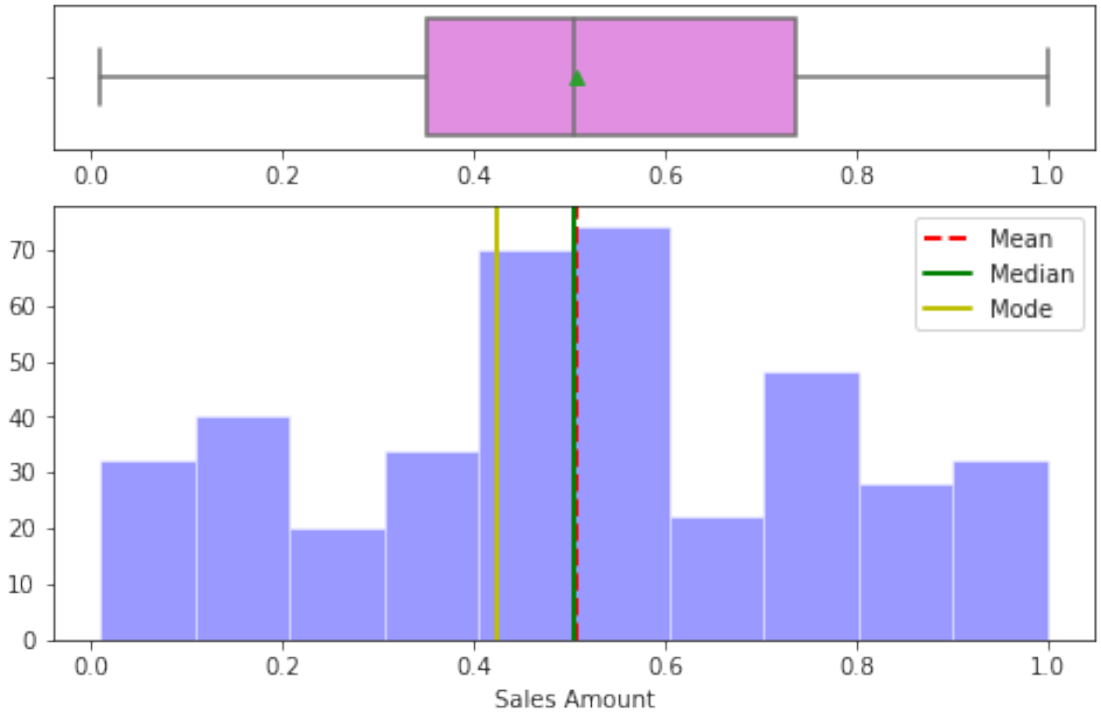
SPREAD OF DATA FOR AGE



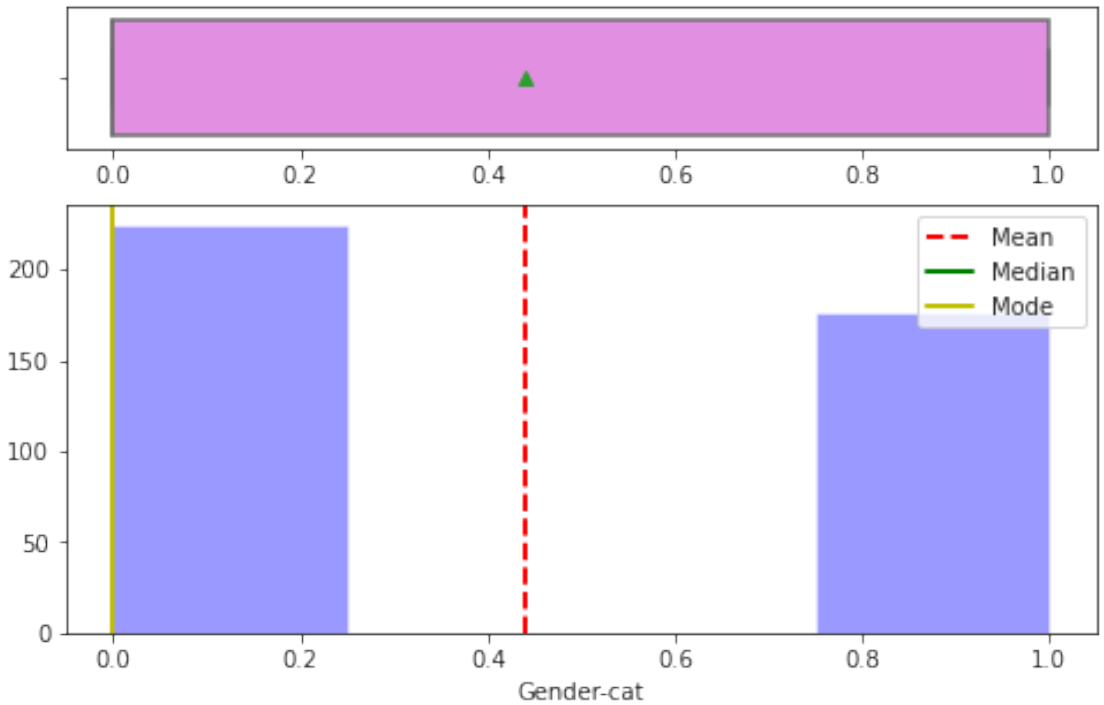
SPREAD OF DATA FOR PURCHASE AMOUNT



SPREAD OF DATA FOR SALES AMOUNT



SPREAD OF DATA FOR GENDER-CAT



```
data = df_max_scaled.drop_duplicates()
data.head()
```

	Age	Purchase Amount	Sales Amount	Gender-cat
0	0.257143	0.109489	0.010101	1.0
2	0.257143	0.116788	0.030303	0.0
3	0.257143	0.116788	0.040404	0.0
4	0.271429	0.124088	0.040404	0.0
5	0.271429	0.124088	0.050505	0.0

```
y = data["Sales Amount"]
X = data.drop('Sales Amount',axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.5)

df_max_scaled.shape
```

```
(400, 4)
```

```
from sklearn import preprocessing
from sklearn import utils
lab_enc = preprocessing.LabelEncoder()
training_scores_encoded = lab_enc.fit_transform(y_train)
print(utils.multiclass.type_of_target(y_train))
print(utils.multiclass.type_of_target(y_train.astype('int')))
print(utils.multiclass.type_of_target(training_scores_encoded))
```

```
continuous
binary
multiclass
```

```
lab_enc = preprocessing.LabelEncoder()
training_scores_encoded1 = lab_enc.fit_transform(y_test)
print(utils.multiclass.type_of_target(y_test))
print(utils.multiclass.type_of_target(y_test.astype('int')))
print(utils.multiclass.type_of_target(training_scores_encoded1))
```

```
continuous
binary
multiclass
```

```
m1 = 'Random Forest Classfier'
rf = RandomForestClassifier(n_estimators=20, max_depth=5)
rf.fit(X_train,training_scores_encoded)
rf_predicted = rf.predict(X_test)
rf_conf_matrix = confusion_matrix(training_scores_encoded1,
rf_predicted)
rf_acc_score = accuracy_score(training_scores_encoded1, rf_predicted)
print("confussion matrix")
print(rf_conf_matrix)
print("\n")
print("Accuracy of Random Forest:",rf_acc_score*100,'\n')
```

```
print(classification_report(training_scores_encoded1,rf_predicted))

kfold = KFold(n_splits=10, random_state=None)
cv_results = cross_val_score(rf, X_train, training_scores_encoded,
cv=kfold, scoring='accuracy')
msg = "%s: %f (%f)" % (m1, cv_results.mean(), cv_results.std())
print(msg)
```

```
confussion matrix
[[1 0 0 ... 0 0 0]
 [1 0 0 ... 0 0 0]
 [2 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]


Accuracy of Random Forest: 1.2658227848101267
```

	precision	recall	f1-score	support
0	0.11	1.00	0.20	1
1	0.00	0.00	0.00	1
2	0.00	0.00	0.00	2
3	0.00	0.00	0.00	1
4	0.00	0.00	0.00	2
5	0.00	0.00	0.00	1
6	0.00	0.00	0.00	1
7	0.00	0.00	0.00	1
8	0.00	0.00	0.00	1
9	0.00	0.00	0.00	1
10	0.00	0.00	0.00	3
11	0.00	0.00	0.00	1
12	0.00	0.00	0.00	1
13	0.00	0.00	0.00	1
14	0.00	0.00	0.00	1
15	0.00	0.00	0.00	2
16	0.00	0.00	0.00	1
17	0.00	0.00	0.00	1
18	0.00	0.00	0.00	1
19	0.00	0.00	0.00	1
20	0.00	0.00	0.00	1
21	0.00	0.00	0.00	1
22	0.00	0.00	0.00	2
23	0.00	0.00	0.00	2
24	0.00	0.00	0.00	1
25	0.00	0.00	0.00	2
26	0.00	0.00	0.00	2
27	0.00	0.00	0.00	2

```
           28       0.00      0.00      0.00         1
           29       0.00      0.00      0.00         1
           30       0.00      0.00      0.00         3
           31       0.00      0.00      0.00         3
           32       0.00      0.00      0.00         1
           33       0.00      0.00      0.00         2
           34       0.00      0.00      0.00         1
           35       0.00      0.00      0.00         2
           36       0.00      0.00      0.00         1
           37       0.00      0.00      0.00         1
           38       0.00      0.00      0.00         1
           39       0.00      0.00      0.00         1
           40       0.00      0.00      0.00         1
           41       0.00      0.00      0.00         2
           42       0.00      0.00      0.00         1
           43       0.00      0.00      0.00         1
           44       0.00      0.00      0.00         1
           45       0.00      0.00      0.00         1
           46       0.00      0.00      0.00         2
           47       0.00      0.00      0.00         2
           48       0.00      0.00      0.00         1
           49       0.00      0.00      0.00         2
           50       0.00      0.00      0.00         1
           51       0.00      0.00      0.00         1
           52       0.00      0.00      0.00         1
           53       0.00      0.00      0.00         1
           54       0.00      0.00      0.00         1
           55       0.00      0.00      0.00         2
           56       0.00      0.00      0.00         1
           57       0.00      0.00      0.00         1

    accuracy                           0.01        79
   macro avg       0.00      0.02      0.00        79
weighted avg       0.00      0.01      0.00        79
```

```
Random Forest Classfier: 0.176786 (0.113908)
```

```
import pickle
pickle.dump(rf, open('rfmodel.pkl','wb'))
```

```
loaded_model=pickle.load(open('./rfmodel.pkl','rb'))
```

```
loaded_model.predict([np.array([1,19,19])])[0]
```

```
50
```