```
import pandas as pd
import numpy as np
df=pd.read_excel('/content/drive/MyDrive/Datasets/Road.xlsx')
df
```

	Sex_of_driver	Educational_level	Vehicle_driver_relation	Owner_of_vehicle	Are
0	Male	Above high school	Employee	Owner	
1	Male	Junior high school	Employee	Owner	
2	Male	Junior high school	Employee	Owner	
3	Male	Junior high school	Employee	Governmental	
4	Male	Junior high school	Employee	Owner	
...	...	...	...	...	
12311	Male	NaN	Employee	Owner	
12312	Male	Elementary school	Employee	Owner	
12313	Male	Junior high school	Employee	Owner	
12314	Female	Junior high school	Employee	Owner	
12315	Male	Junior high school	Employee	Owner	

12316 rows × 12 columns

Next steps: Generate code with df View recommended plots

```
df.head()
```

	Sex_of_driver	Educational_level	Vehicle_driver_relation	Owner_of_vehicle	Area_acc
0	Male	Above high school	Employee	Owner	F
1	Male	Junior high school	Employee	Owner	
2	Male	Junior high school	Employee	Owner	Re
3	Male	Junior high school	Employee	Governmental	
4	Male	Junior high school	Employee	Owner	

Next steps: Generate code with df View recommended plots

```
df.replace('Unknown',np.NaN,inplace=True)
```

```
df.replace('Fatal injury','Serious Injury',inplace=True)
```

```
df['Accident_severity'].value_counts()
```

```
Accident_severity
Slight Injury     10415
Serious Injury     1901
Name: count, dtype: int64
```

```
df.isna().sum()
```

```
Sex_of_driver                 178
Educational_level             841
Vehicle_driver_relation       593
Owner_of_vehicle              482
Area_accident_occured         261
Light_conditions                0
Weather_conditions            292
```

```
Number_of_vehicles_involved      0
Number_of_casualties             0
Vehicle_movement               396
Cause_of_accident               25
Accident_severity                0
dtype: int64
```

```
df.dropna(inplace=True)
```

```
df.describe()
```

	Number_of_vehicles_involved	Number_of_casualties
count	10064.000000	10064.000000
mean	2.042130	1.546999
std	0.694532	1.010320
min	1.000000	1.000000
25%	2.000000	1.000000
50%	2.000000	1.000000
75%	2.000000	2.000000
max	7.000000	8.000000

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 10064 entries, 0 to 12315
Data columns (total 12 columns):
 #   Column                       Non-Null Count  Dtype
---  ------                       --------------  -----
 0   Sex_of_driver                10064 non-null  object
 1   Educational_level            10064 non-null  object
 2   Vehicle_driver_relation      10064 non-null  object
 3   Owner_of_vehicle             10064 non-null  object
 4   Area_accident_occured        10064 non-null  object
 5   Light_conditions             10064 non-null  object
 6   Weather_conditions           10064 non-null  object
 7   Number_of_vehicles_involved  10064 non-null  int64
 8   Number_of_casualties         10064 non-null  int64
 9   Vehicle_movement             10064 non-null  object
 10  Cause_of_accident            10064 non-null  object
 11  Accident_severity            10064 non-null  object
dtypes: int64(2), object(10)
memory usage: 1022.1+ KB
```

```
df.shape
```

```
(10064, 12)
```

```
df.dtypes
```

```
Sex_of_driver                  object
Educational_level              object
Vehicle_driver_relation        object
Owner_of_vehicle               object
Area_accident_occured          object
Light_conditions               object
Weather_conditions             object
Number_of_vehicles_involved     int64
Number_of_casualties            int64
Vehicle_movement               object
Cause_of_accident              object
Accident_severity              object
dtype: object
```

```
df.reset_index(drop=True,inplace=True)
```

```
df
```

	Sex_of_driver	Educational_level	Vehicle_driver_relation	Owner_of_vehicle	Are
0	Male	Above high school	Employee	Owner	
1	Male	Junior high school	Employee	Owner	
2	Male	Junior high school	Employee	Owner	
3	Male	Junior high school	Employee	Governmental	
4	Male	Junior high school	Employee	Owner	
...	...	...	...	...	
10059	Female	Elementary school	Employee	Owner	
10060	Male	Elementary school	Employee	Owner	
10061	Male	Junior high school	Employee	Owner	
10062	Female	Junior high school	Employee	Owner	
10063	Male	Junior high school	Employee	Owner	

10064 rows × 12 columns

Next steps: Generate code with df View recommended plots

```
df.dtypes
```

```
Sex_of_driver                 object
Educational_level             object
Vehicle_driver_relation       object
Owner_of_vehicle              object
Area_accident_occured         object
Light_conditions              object
Weather_conditions            object
Number_of_vehicles_involved    int64
Number_of_casualties           int64
Vehicle_movement              object
Cause_of_accident             object
Accident_severity             object
dtype: object
```

```
df.shape
```

```
(10064, 12)
```

Label Encoder

```
for i in df:
  if df[i].dtypes=='object':
    print(i,'|',df[i].unique())
```

```
Sex_of_driver | ['Male' 'Female']
Educational_level | ['Above high school' 'Junior high school' 'Elementary school'
 'High school' 'Illiterate' 'Writing & reading']
Vehicle_driver_relation | ['Employee' 'Owner' 'Other']
Owner_of_vehicle | ['Owner' 'Governmental' 'Organization' 'Other']
Area_accident_occured | ['Residential areas' 'Office areas' 'Recreational areas'
 'Industrial areas' 'Other' 'Church areas' 'Market areas'
 'Rural village areas' 'Outside rural areas' 'Hospital areas'
 'School areas' 'Rural village areasOffice areas']
Light_conditions | ['Daylight' 'Darkness - lights lit' 'Darkness - no lighting'
 'Darkness - lights unlit']
Weather_conditions | ['Normal' 'Raining' 'Raining and Windy' 'Cloudy' 'Other' 'Windy' 'Snow'
 'Fog or mist']
Vehicle_movement | ['Going straight' 'Moving Backward' 'U-Turn' 'Turnover' 'Waiting to go'
 'Getting off' 'Reversing' 'Parked' 'Stopping' 'Other' 'Overtaking'
 'Entering a junction']
Cause_of_accident | ['Moving Backward' 'Overtaking' 'Changing lane to the left'
 'Changing lane to the right' 'Other' 'No priority to vehicle'
 'No priority to pedestrian' 'No distancing'
 'Getting off the vehicle improperly' 'Overloading' 'Driving carelessly'
 'Driving at high speed' 'Driving to the left' 'Overturning' 'Turnover'
 'Driving under the influence of drugs' 'Overspeed' 'Drunk driving'
 'Improper parking']
Accident_severity | ['Slight Injury' 'Serious Injury']
```

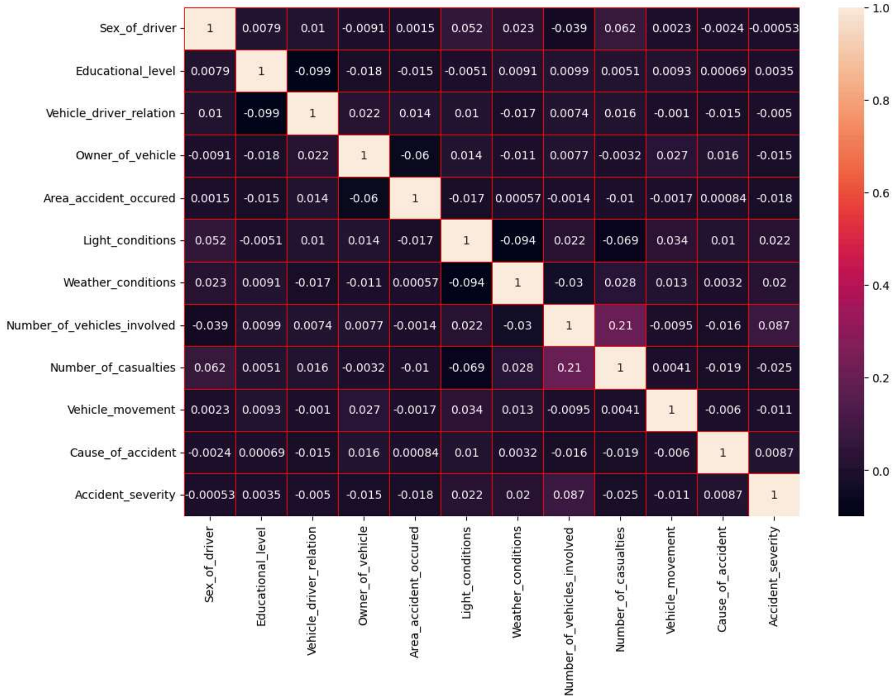
```
from sklearn.preprocessing import LabelEncoder
lst=['Sex_of_driver','Educational_level','Vehicle_driver_relation','Owner_of_vehicle','Area_accident_occured','Light_conditions','Weather
label={}
for col in lst:
  label[col]=LabelEncoder()
  df[col]=label[col].fit_transform(df[col])
label
```

```
{'Sex_of_driver': LabelEncoder(),
 'Educational_level': LabelEncoder(),
 'Vehicle_driver_relation': LabelEncoder(),
 'Owner_of_vehicle': LabelEncoder(),
 'Area_accident_occured': LabelEncoder(),
 'Light_conditions': LabelEncoder(),
 'Weather_conditions': LabelEncoder(),
 'Vehicle_movement': LabelEncoder(),
 'Cause_of_accident': LabelEncoder(),
 'Accident_severity': LabelEncoder()}
```

Visualization

Heatmap

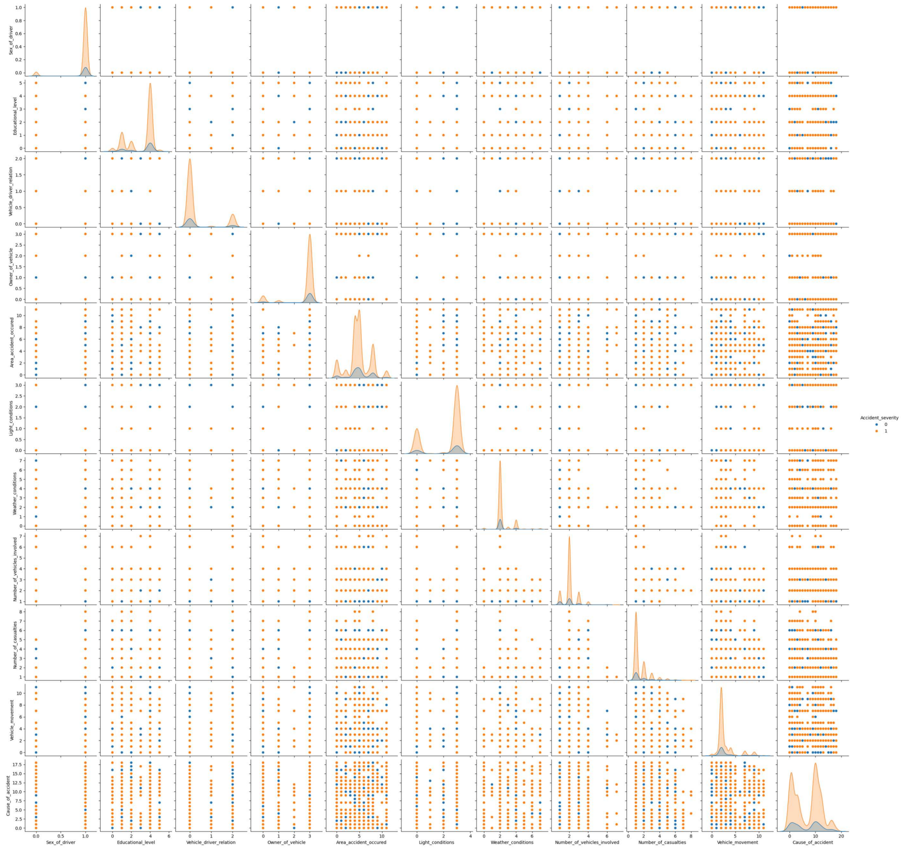
```
import matplotlib.pyplot as plt
import seaborn as sns
plt.subplots(figsize=(12, 8))
sns.heatmap(df.corr(),linewidths=0.5,linecolor='red',annot=True)
plt.show()
```



	Sex_of_driver	Educational_level	Vehicle_driver_relation	Owner_of_vehicle	Area_accident_occured	Light_conditions	Weather_conditions	Number_of_vehicles_involved	Number_of_casualties	Vehicle_movement	Cause_of_accident	Accident_severity
Sex_of_driver	1	0.0079	0.01	-0.0091	0.0015	0.052	0.023	-0.039	0.062	0.0023	-0.0024	-0.00053
Educational_level	0.0079	1	-0.099	-0.018	-0.015	-0.0051	0.0091	0.0099	0.0051	0.0093	0.00069	0.0035
Vehicle_driver_relation	0.01	-0.099	1	0.022	0.014	0.01	-0.017	0.0074	0.016	-0.001	-0.015	-0.005
Owner_of_vehicle	-0.0091	-0.018	0.022	1	-0.06	0.014	-0.011	0.0077	-0.0032	0.027	0.016	-0.015
Area_accident_occured	0.0015	-0.015	0.014	-0.06	1	-0.017	0.00057	-0.0014	-0.01	-0.0017	0.00084	-0.018
Light_conditions	0.052	-0.0051	0.01	0.014	-0.017	1	-0.094	0.022	-0.069	0.034	0.01	0.022
Weather_conditions	0.023	0.0091	-0.017	-0.011	0.00057	-0.094	1	-0.03	0.028	0.013	0.0032	0.02
Number_of_vehicles_involved	-0.039	0.0099	0.0074	0.0077	-0.0014	0.022	-0.03	1	0.21	-0.0095	-0.016	0.087
Number_of_casualties	0.062	0.0051	0.016	-0.0032	-0.01	-0.069	0.028	0.21	1	0.0041	-0.019	-0.025
Vehicle_movement	0.0023	0.0093	-0.001	0.027	-0.0017	0.034	0.013	-0.0095	0.0041	1	-0.006	-0.011
Cause_of_accident	-0.0024	0.00069	-0.015	0.016	0.00084	0.01	0.0032	-0.016	-0.019	-0.006	1	0.0087
Accident_severity	-0.00053	0.0035	-0.005	-0.015	-0.018	0.022	0.02	0.087	-0.025	-0.011	0.0087	1

Pair Plot

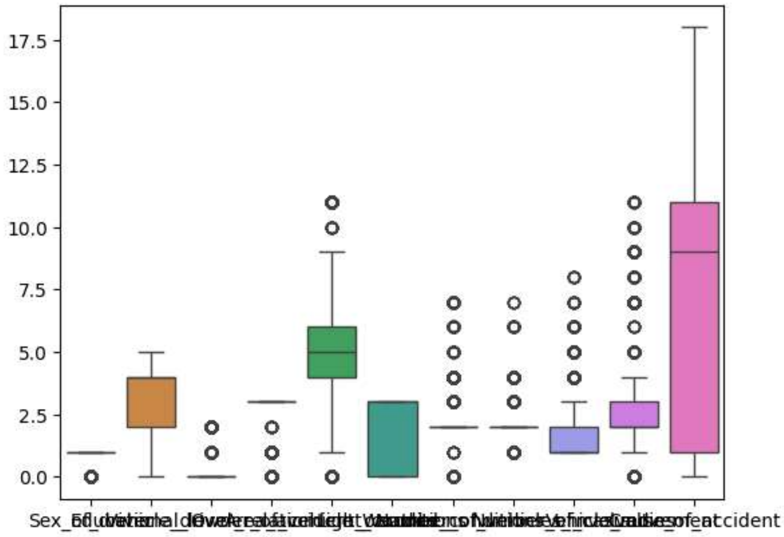
```
sns.pairplot(df,hue='Accident_severity',kind='scatter')
plt.show()
```



SeaBorn Plot

```
col=df[['Sex_of_driver','Educational_level','Vehicle_driver_relation','Owner_of_vehicle','Area_accident_occured','Light_conditions','Weat
sns.boxplot(col)
```

```
<Axes: >
```



Remove Outerliers

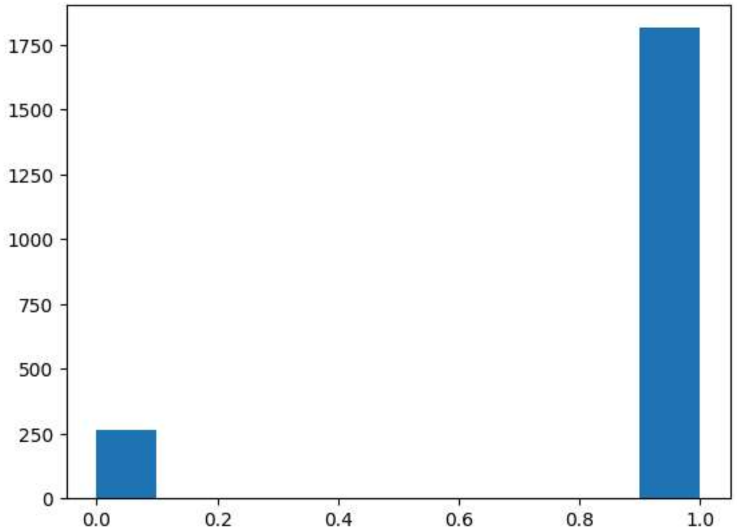
```
lst=['Sex_of_driver','Educational_level','Vehicle_driver_relation','Owner_of_vehicle','Area_accident_occured','Light_conditions','Weather
for i in lst:
  if df[i].dtype in ['int64']:
      Q1 = df[i].quantile(0.25)
      Q3 = df[i].quantile(0.75)
      IQR = Q3 - Q1
      lower_bound = Q1 - 1.5 * IQR
      upper_bound = Q3 + 1.5 * IQR
      df = df[(df[i] >= lower_bound) & (df[i] <= upper_bound)]
```

```
df['Accident_severity'].value_counts()
```

```
Accident_severity
1    1815
0     264
Name: count, dtype: int64
```

Hist Plot

```
plt.hist(df['Accident_severity'])
plt.show()
```



```
df['Accident_severity'].value_counts()
```

```
Accident_severity
1    1815
0     264
Name: count, dtype: int64
```

```
x=df.drop(columns='Accident_severity')
y=df['Accident_severity']
```

OverSampling

```
from imblearn.over_sampling import RandomOverSampler
rand=RandomOverSampler()
x,y=rand.fit_resample(x,y)
```

```
df['Accident_severity'].value_counts()
```

```
Accident_severity
1    1815
0     264
Name: count, dtype: int64
```

TrainTest Splitting

```
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,random_state=42)
```

Normalized Scaling

```
from sklearn.preprocessing import StandardScaler
ss=StandardScaler()
x_train_scaled=ss.fit_transform(x_train)
x_test_scaled=ss.transform(x_test)
```

KNeighborsClassifier

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(x_train_scaled,y_train)
```

```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

```
y_pred=knn.predict(x_test_scaled)
y_pred
```

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

```
y_test.values
```

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

```
from sklearn.metrics import classification_report,accuracy_score
print(accuracy_score(y_test,y_pred)*100)
```

```
67.76859504132231
```

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

```
              precision    recall  f1-score   support

           0       0.63      0.80      0.70       522
           1       0.76      0.56      0.65       567

    accuracy                           0.68      1089
   macro avg       0.69      0.68      0.67      1089
weighted avg       0.69      0.68      0.67      1089
```

```
x_scaled=ss.transform(x)
```

SVM

```
from sklearn.svm import SVC
model=SVC()
model.fit(x_train_scaled,y_train)
```

```
▾ SVC
SVC()
```

```
y_pred=model.predict(x_test_scaled)
y_pred
```

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

```
print(accuracy_score(y_test,y_pred)*100)
```

```
59.87144168962351
```

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

```
              precision    recall  f1-score   support

           0       0.57      0.69      0.62       522
           1       0.64      0.51      0.57       567

    accuracy                           0.60      1089
   macro avg       0.61      0.60      0.60      1089
weighted avg       0.61      0.60      0.60      1089
```

Naive Bayes

```
from sklearn.naive_bayes import MultinomialNB,GaussianNB,BernoulliNB
model1=BernoulliNB()
model1.fit(x_train_scaled,y_train)
```

```
▾ BernoulliNB
BernoulliNB()
```

```
y_pred=model1.predict(x_test_scaled)
```

```
print(accuracy_score(y_test,y_pred)*100)
```

```
53.168044077134994
```

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

```
              precision    recall  f1-score   support

           0       0.51      0.83      0.63       522
           1       0.62      0.25      0.36       567

    accuracy                           0.53      1089
   macro avg       0.57      0.54      0.50      1089
weighted avg       0.57      0.53      0.49      1089
```

Decision Tree

```
from sklearn.tree import DecisionTreeClassifier,plot_tree
model2=DecisionTreeClassifier()
model2.fit(x_train,y_train)
```

```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

```
y_pred=model2.predict(x_test)
y_pred
```

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

```
print(accuracy_score(y_test,y_pred)*100)
```

```
73.27823691460054
```

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

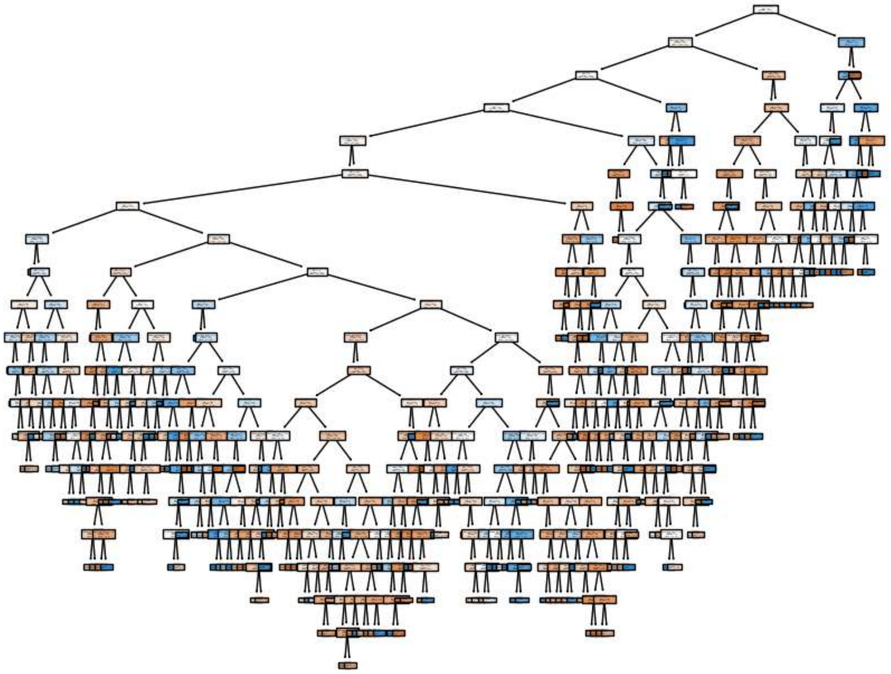
```
              precision    recall  f1-score   support

           0       0.66      0.90      0.76       522
           1       0.86      0.58      0.69       567

    accuracy                           0.73      1089
   macro avg       0.76      0.74      0.73      1089
weighted avg       0.77      0.73      0.73      1089
```

Decision Tree Plot

```
plt.figure(figsize=(10,8))
plot_tree(model2,filled=True,feature_names=x_train.columns)
plt.show()
```



RandomForest

```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(x_train,y_train)
```

```
▾ RandomForestClassifier
RandomForestClassifier()
```

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

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

```
print(accuracy_score(y_test,y_pred)*100)
```

```
73.46189164370982
```

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

```
              precision    recall  f1-score   support

           0       0.67      0.89      0.76       522
           1       0.85      0.59      0.70       567

    accuracy                           0.73      1089
   macro avg       0.76      0.74      0.73      1089
weighted avg       0.76      0.73      0.73      1089
```

GradientBoosting

```
from sklearn.ensemble import GradientBoostingClassifier
gc=GradientBoostingClassifier()
gc.fit(x_train,y_train)
gc.fit(x_train,y_train)
```

```
▾ GradientBoostingClassifier
GradientBoostingClassifier()
```

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

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

```
print(accuracy_score(y_test,y_pred)*100)
```

```
61.89164370982553
```

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

```
              precision    recall  f1-score   support

           0       0.58      0.73      0.65       522
           1       0.68      0.52      0.59       567

    accuracy                           0.62      1089
   macro avg       0.63      0.62      0.62      1089
weighted avg       0.63      0.62      0.62      1089
```

XGBooster

```
from xgboost import XGBClassifier
xg=XGBClassifier()
xg.fit(x_train,y_train)
```

```
▾                              XGBClassifier
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
```

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

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

```
print(accuracy_score(y_test,y_pred)*100)
```

```
61.89164370982553
```

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

```
              precision    recall  f1-score   support

           0       0.58      0.73      0.65       522
           1       0.68      0.52      0.59       567

    accuracy                           0.62      1089
   macro avg       0.63      0.62      0.62      1089
weighted avg       0.63      0.62      0.62      1089
```

AdaBooster

```
from sklearn.ensemble import AdaBoostClassifier
```

```
ada=AdaBoostClassifier()
ada.fit(x_train,y_train)
```

```
▾ AdaBoostClassifier
AdaBoostClassifier()
```

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

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

```
print(accuracy_score(y_test,y_pred)*100)
```

```
56.84113865932048
```

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

```
              precision    recall  f1-score   support

           0       0.54      0.63      0.58       522
           1       0.60      0.51      0.55       567

    accuracy                           0.57      1089
   macro avg       0.57      0.57      0.57      1089
weighted avg       0.57      0.57      0.57      1089
```

```
from sklearn.model_selection import RandomizedSearchCV
params={'criterion':['entropy','gini'],
        'max_depth':[5,10,30,40,50],
       'min_samples_split':[10,20,25,30,45,60,100] }
rs=RandomizedSearchCV(RandomForestClassifier(),params,cv=10,n_iter=15)
rs.fit(x_train,y_train)
```

```
▸ RandomizedSearchCV
▸ estimator: RandomForestClassifier
    ▸ RandomForestClassifier
```

```
rs.best_params_
```

```
{'min_samples_split': 20, 'max_depth': 30, 'criterion': 'gini'}
```

```
rs.best_score_
```

```
0.7044681179558437
```

```
knn=KNeighborsClassifier()
svc=SVC()
naive=BernoulliNB()
decision=DecisionTreeClassifier()
rf=RandomForestClassifier()
gcv=GradientBoostingClassifier()
xgb=XGBClassifier()
ada=AdaBoostClassifier()
```

```
model_lst=[knn,svc,naive,decision,rf,gcv,xgb,ada]
for model in model_lst:
  model.fit(x_train,y_train)
  y_pred=model.predict(x_test)
  print(model)
  print('Accuracy_Score:',accuracy_score(y_test,y_pred)*100)
  print(classification_report(y_test,y_pred))
  print('-------------------------------------------------------------------')
```

```
    accuracy                           0.73      1089
   macro avg       0.76      0.74      0.73      1089
weighted avg       0.77      0.73      0.73      1089

------------------------------------------------------------------
GradientBoostingClassifier()
Accuracy_Score: 61.89164370982553
              precision    recall  f1-score   support

           0       0.58      0.73      0.65       522
           1       0.68      0.52      0.59       567

    accuracy                           0.62      1089
   macro avg       0.63      0.62      0.62      1089
weighted avg       0.63      0.62      0.62      1089

------------------------------------------------------------------
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, device=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              multi_strategy=None, n_estimators=None, n_jobs=None,
              num_parallel_tree=None, random_state=None, ...)
Accuracy_Score: 71.71717171717171
              precision    recall  f1-score   support
```