Titanic To Space

Presented by Shlok Phulkar

Shlok Phulkar

Aspiring Data and Business Analyst

Education

BE Chemical Engineering

Post Graduaction
Diploma is Data Analyst
and Machine Learning

Skills

Pyhton Programming

Statistics

SQL

Machine Learning

Power BI

Conatct

9146560434

shlok33phulkar@gmail.com

Index

Introduction to dataset

Data points 95623

Shape 8693, 11

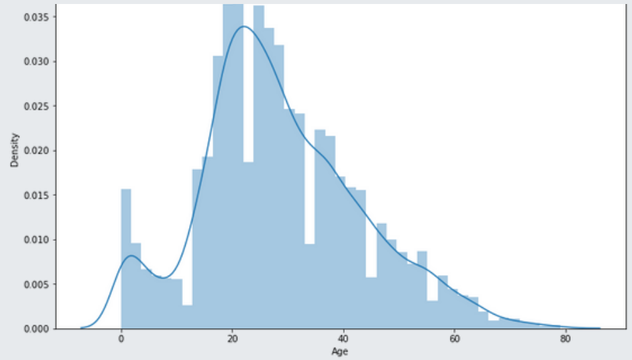
Columns HomePlanet, CryoSleep, Destination, Age, VIP, RoomService,FoodCourt, ShoppingMall, Spa, VRDeck, Transported (TARGET COLUMN)

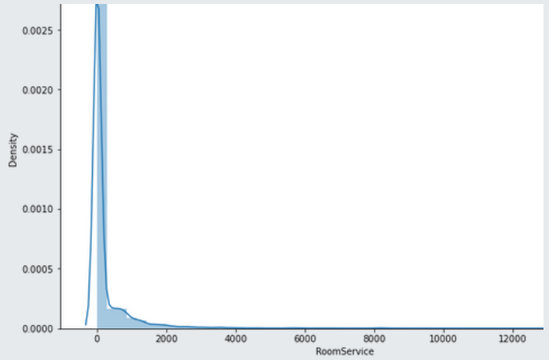
EDA Skewness of all independent columns

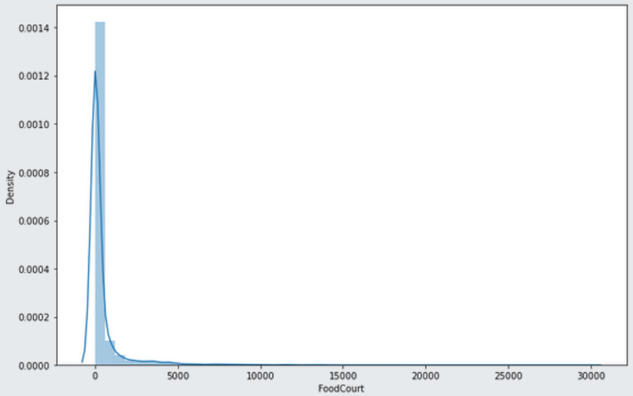
CryoSleep	0.591110
Age	0.419097
VIP	6.300900
RoomService	6.333014
FoodCourt	7.102228
ShoppingMall	12.627562
Spa	7.636020
VRDeck	7.819732
Transported	-0.014497

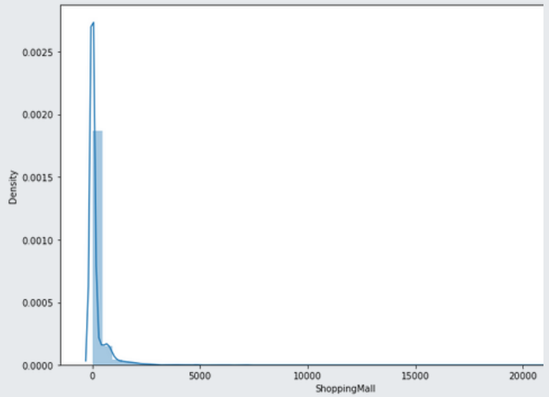
EDA

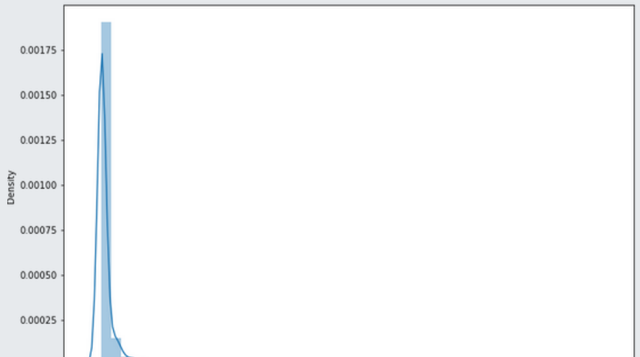
Visualization of all independent columns (Numeric)

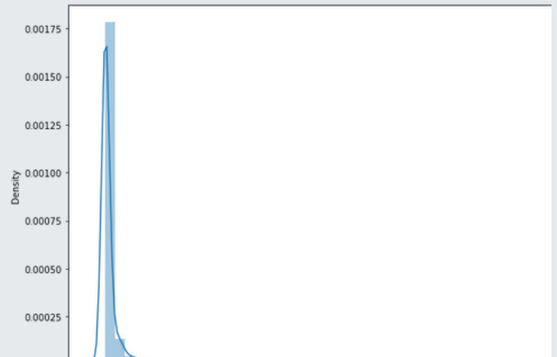






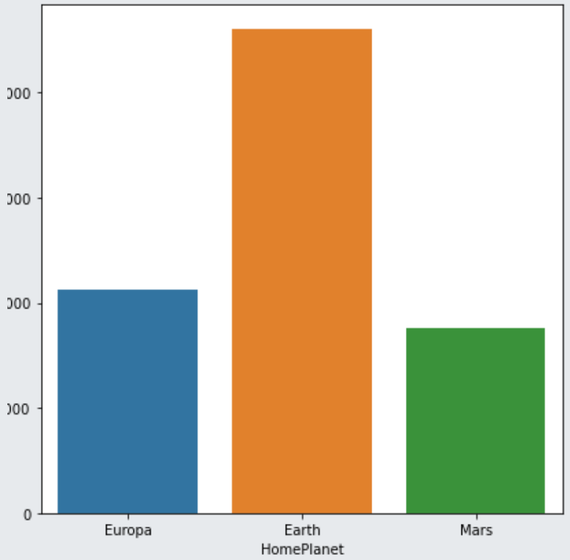


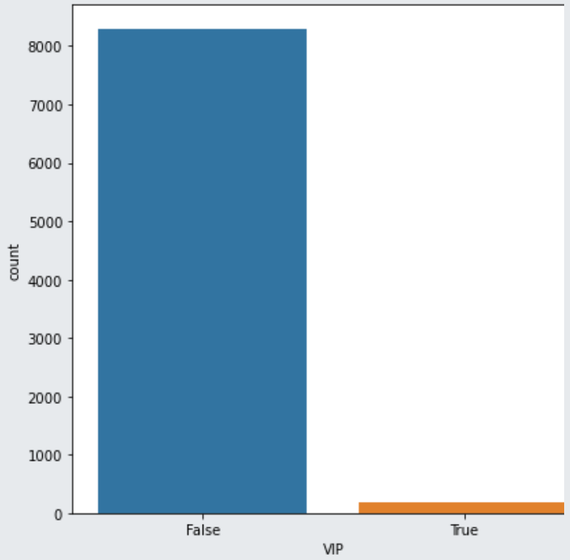


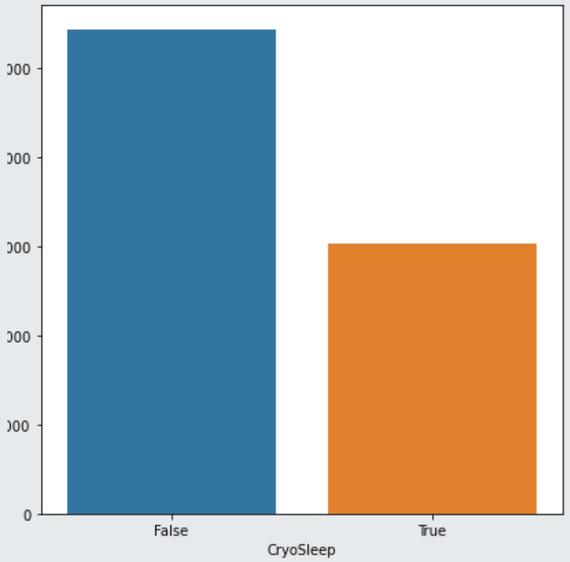


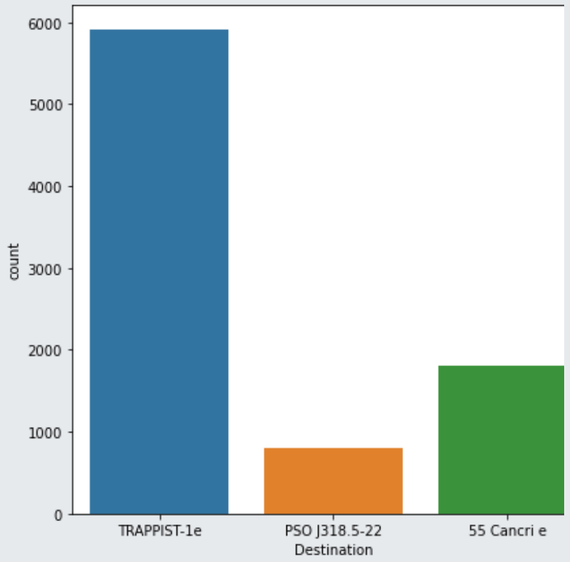
EDA

Visualization of all independent columns (Categorical)



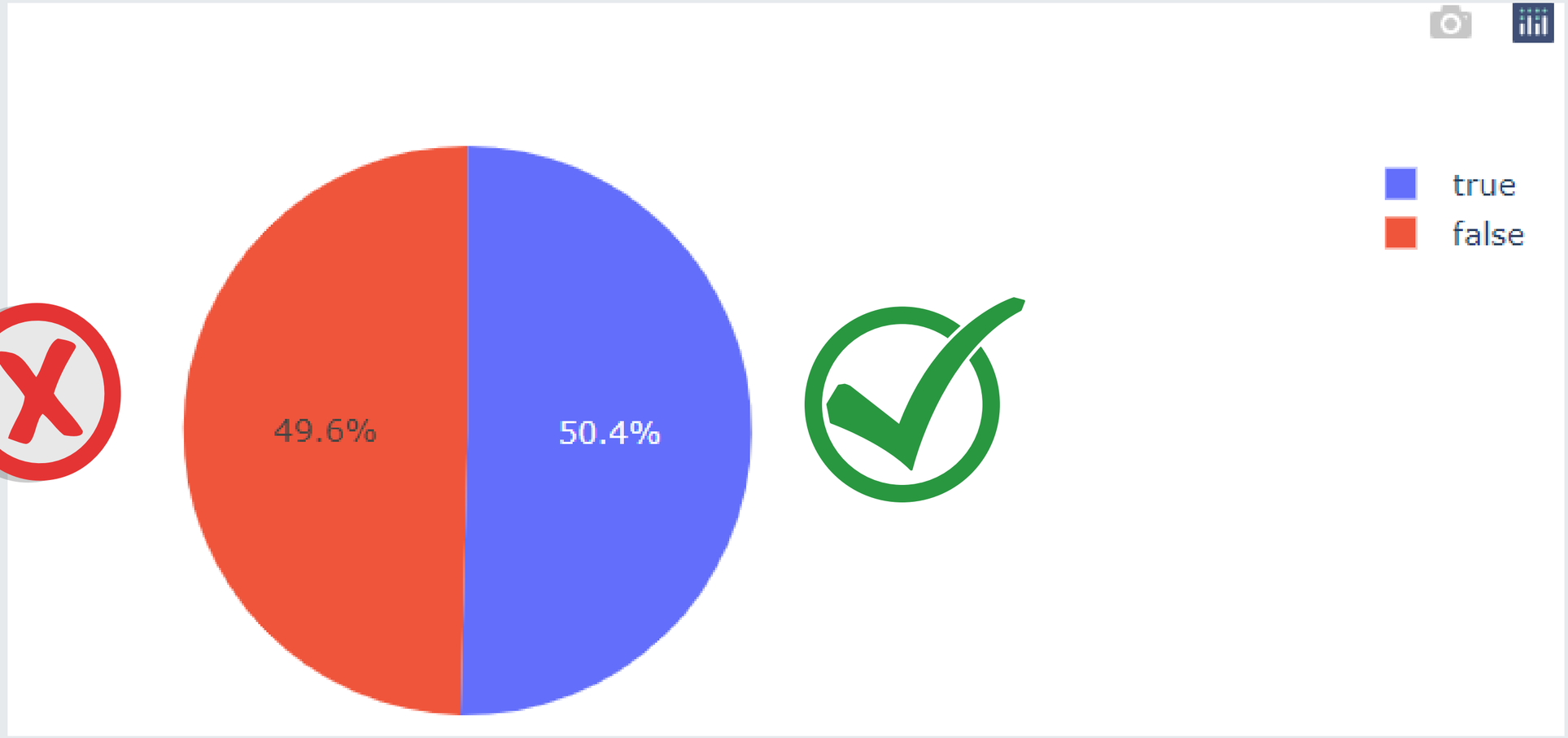




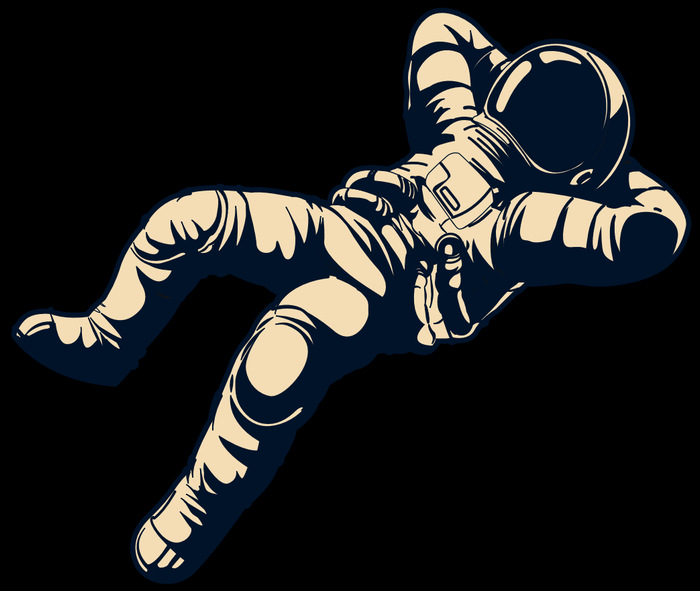


EDA

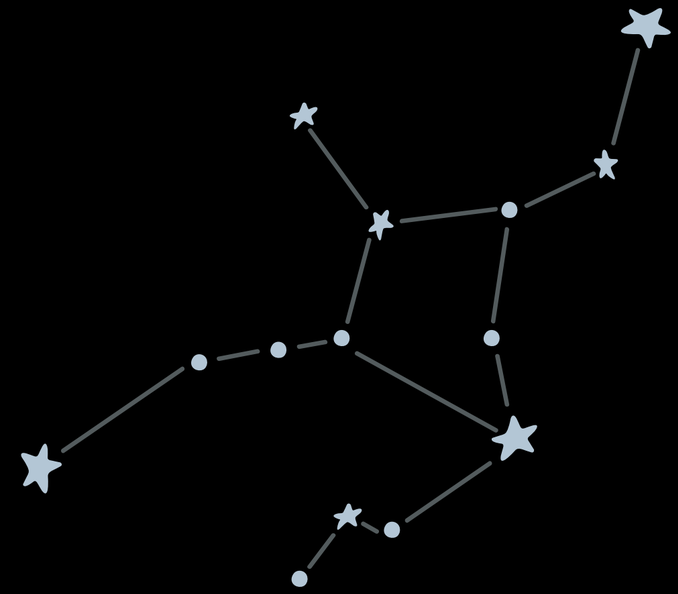
Visualization of target column



Check of missing values



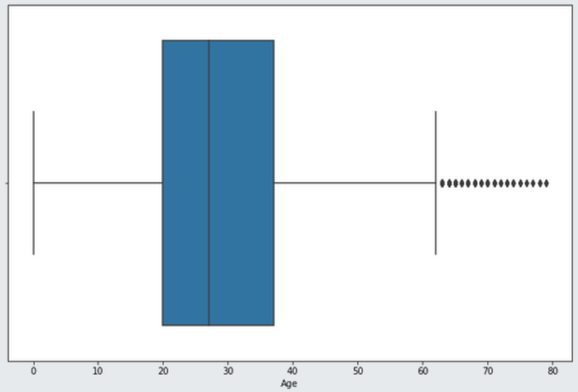
HomePlanet	201
CryoSleep	217
Destination	182
Age	179
VIP	203
RoomService	181
FoodCourt	183
ShoppingMall	208
Spa	183
VRDeck	188
Transported	0

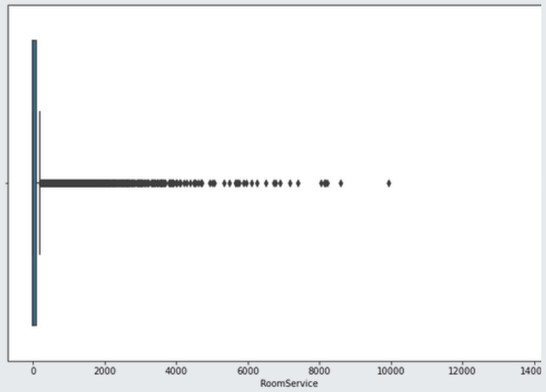


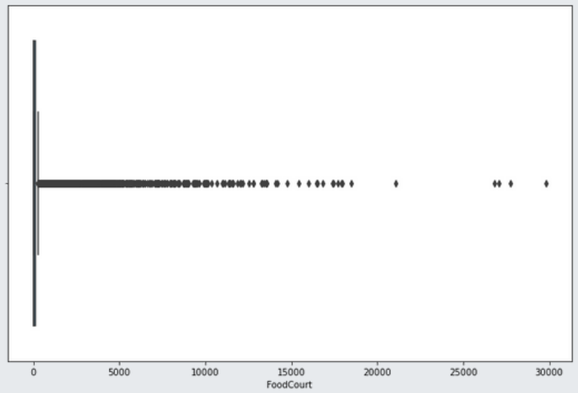
Missing values Treatement

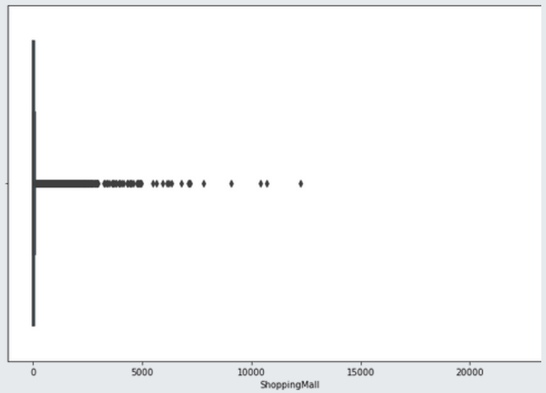
HomePlanet	0
CryoSleep	0
Destination	0
Age	0
VIP	0
RoomService	0
FoodCourt	0
ShoppingMall	0
Spa	0
VRDeck	0
Transported	0

Check of outliers

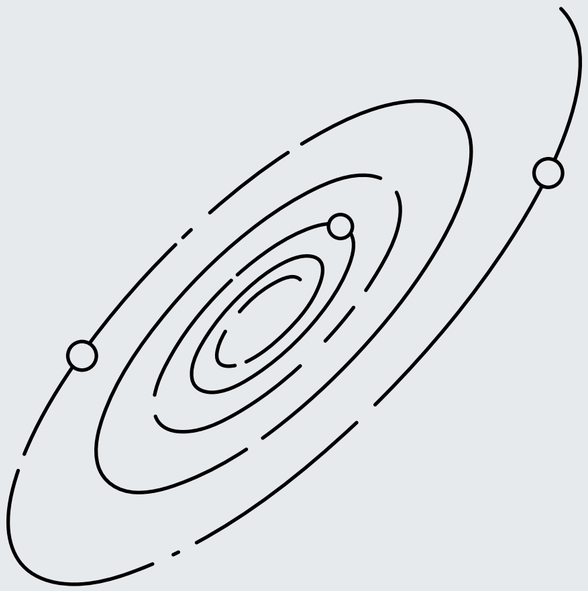


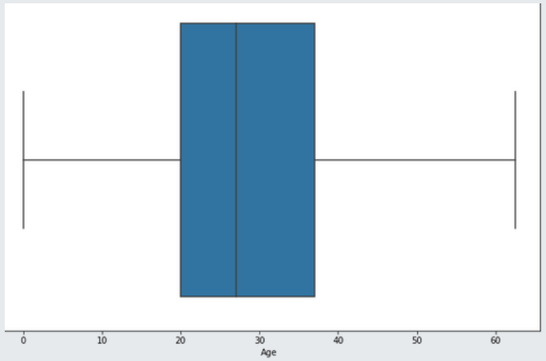


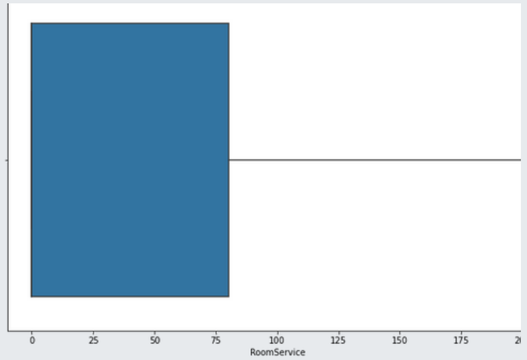


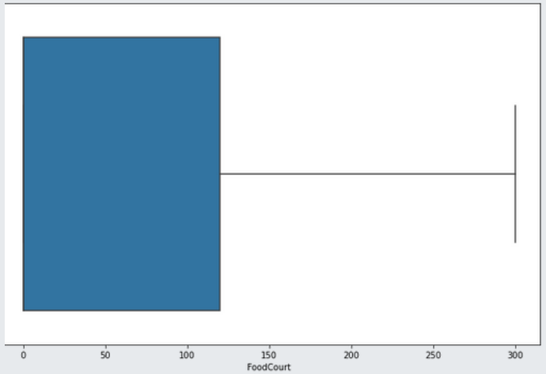


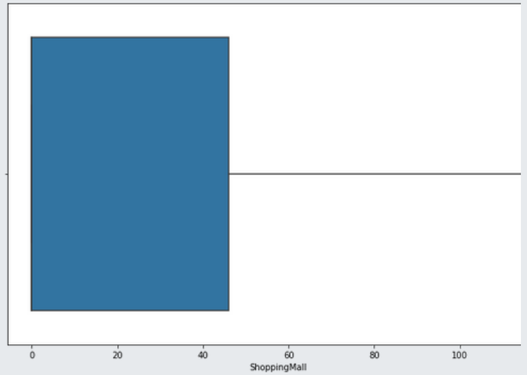
Treatment of missing values

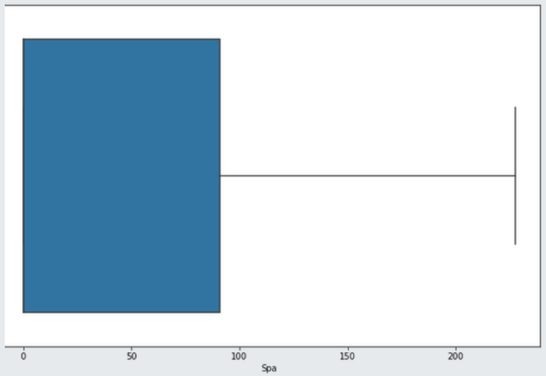


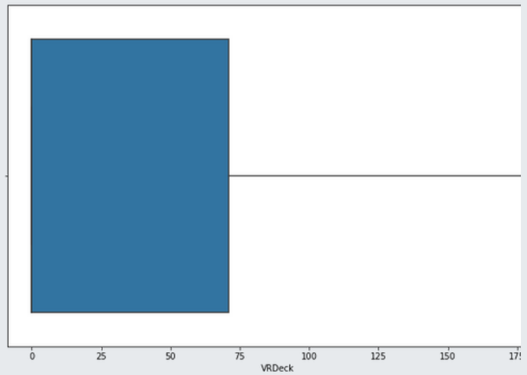












Standardization of Data

For each feature, the Standard Scaler scales the values such that the mean is 0 and the standard deviation is 1(or the variance)

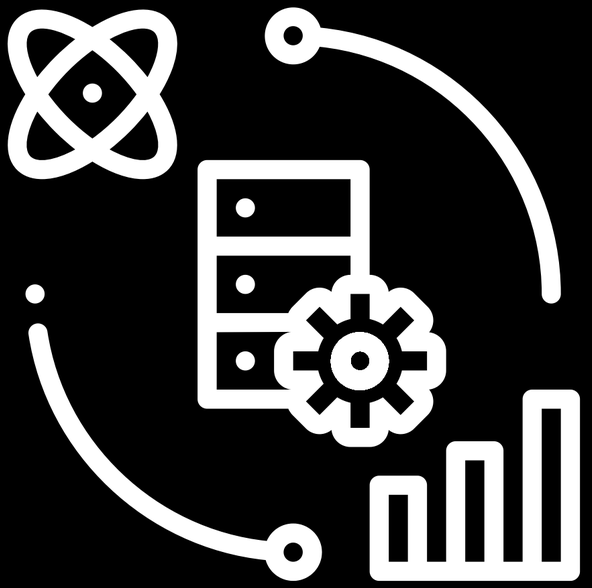
x_scaled=x-mean/std_dev

Standard Scaler assumes that the distribution of the variable is normal

```
#Data standardization
ss=StandardScaler()
X=df.drop("Transported_True",axis=1)
y=df["Transported_True"]
ScaledX=ss.fit_transform(X)
```

Implementation of ML algorithm

1) Logistic Regression
2) KNN
3) SVM
4) Naïve Bayes
5) Decision tree
6) Random Forest
7) Bagging Regressor
8) Extra Trees Regression
9) Adaboost
10) Gradient Boost
11) XgBoost



Logistic Regression

```
#Algoritmn
LR=LogisticRegression()
#Fit
LR.fit(X_train,y_train)
```

```
LogisticRegression()
```

```
#Training and testing score
print("Training score is",LR.score(X_train,y_train))
print("Testing score is",LR.score(X_test,y_test))
```

```
Training score is 0.7727962245069948
Testing score is 0.7632717263075108
```

```
#Performance Parameters
print("Accuracy is ",metrics.accuracy_score(y_test,predict_LR_x))
print("precsion is ",metrics.precision_score(y_test,predict_LR_x))
print("recall is ",metrics.recall_score(y_test,predict_LR_x))
```

```
Accuracy is  0.7632717263075108
precsion is  0.7546296296296297
recall is  0.7749603803486529
```

Naive Bayes

```
#Algoritm
NB=BernoulliNB()
#Fit
NB.fit(X_train,y_train)
```

```
BernoulliNB()
```

```
#Training and testing score
print("Training score is",NB.score(X_train,y_train))
print("Testing score is",NB.score(X_test,y_test))
```

```
Training score is 0.7490308444294623
Testing score is 0.7416437278804562
```

```
#Performance Parameters
print("Accuracy is ",metrics.accuracy_score(y_test,predict_LR_x))
print("precsion is ",metrics.precision_score(y_test,predict_LR_x))
print("recall is ",metrics.recall_score(y_test,predict_LR_x))
```

```
Accuracy is  0.7632717263075108
precsion is  0.7546296296296297
recall is  0.7749603803486529
```

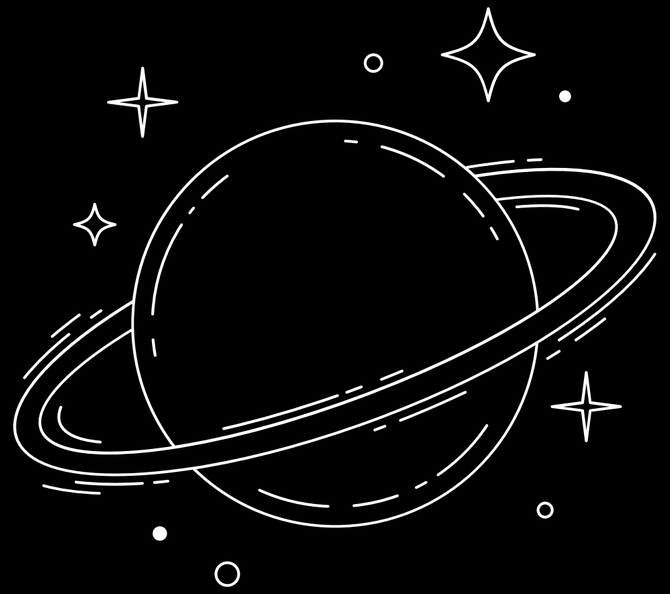
KNN

```
#Algoritm
KNN=KNeighborsClassifier()
#Fit
KNN.fit(X_train,y_train)
```

```
KNeighborsClassifier()
```

```
#Training and testing score
print("Training score is",KNN.score(X_train,y_train))
print("Testing score is",KNN.score(X_test,y_test))
```

```
Training score is 0.803472105174448
Testing score is 0.7412504915454188
```



```
#Performance parameters
print("Accuracy is ",metrics.accuracy_score(y_test,predict_KNN_x))
print("precsion is ",metrics.precision_score(y_test,predict_KNN_x))
print("recall is ",metrics.recall_score(y_test,predict_KNN_x))
```

```
Accuracy is  0.7412504915454188
precsion is  0.7443365695792881
recall is  0.7290015847860539
```

SVM

```
#Algoritm
SVM=SVC(probability=True)
#Fit
SVM.fit(X_train,y_train)
```

```
SVC(probability=True)
```

```
#Training and testing score
print("Training score is",SVM.score(X_train,y_train))
print("Testing score is",SVM.score(X_test,y_test))
```

```
Training score is 0.786111579302208
Testing score is 0.7679905623279591
```

```
#Performance Parameters
print("Accuracy is ",metrics.accuracy_score(y_test,predict_SVM_x))
print("precsion is ",metrics.precision_score(y_test,predict_SVM_x))
print("recall is ",metrics.recall_score(y_test,predict_SVM_x))
```

```
Accuracy is  0.7679905623279591
precsion is  0.7522522522522522
recall is  0.7939778129952456
```

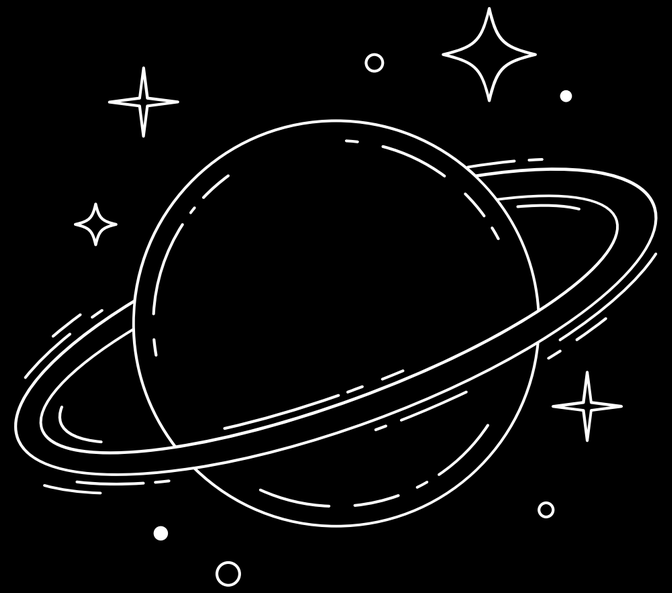
Decision Tree

```
#Algoritm
DT=DecisionTreeClassifier()
#Fit
DT.fit(X_train,y_train)
```

```
DecisionTreeClassifier()
```

```
#Traing and testing score
print("Traning score is",DT.score(X_train,y_train))
print("Testing score is",DT.score(X_test,y_test))
```

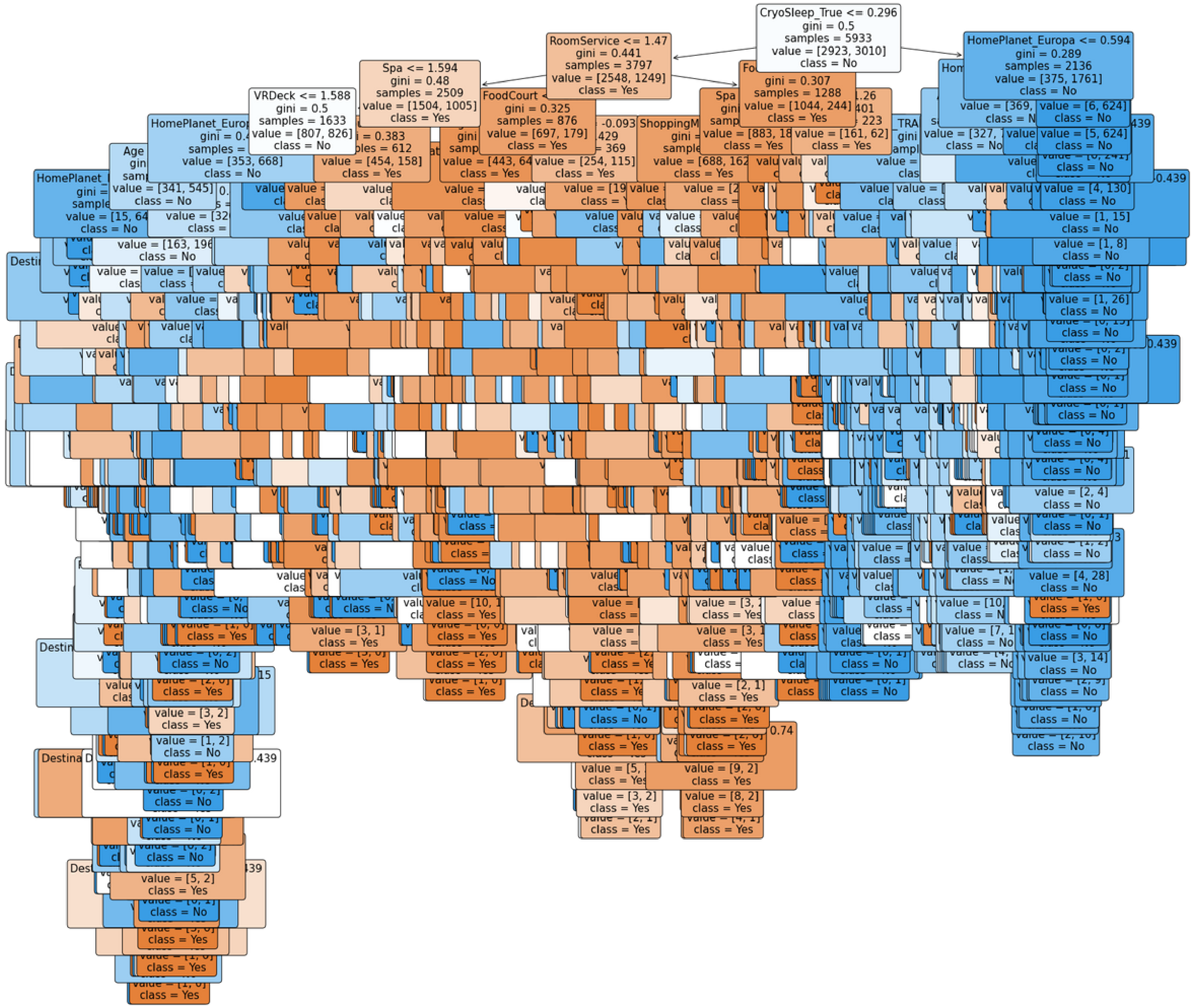
```
Traning score is 0.9367942019214562
Testing score is 0.7046795123869446
```



```
#Performance Parameters
print("Accuracy is ",metrics.accuracy_score(y_test,predict_DT))
print("precsion is ",metrics.precision_score(y_test,predict_DT))
print("recall is ",metrics.recall_score(y_test,predict_DT))
```

```
Accuracy is  0.7046795123869446
precsion is  0.6928301886792453
recall is  0.7274167987321711
```

Decision Tree



Decision Tree

After Hyperparametric Tunning

```
#Traing and testing score
print("Traning score is",DT_P.score(X_train,y_train))
print("Testing score is",DT_P.score(X_test,y_test))
```

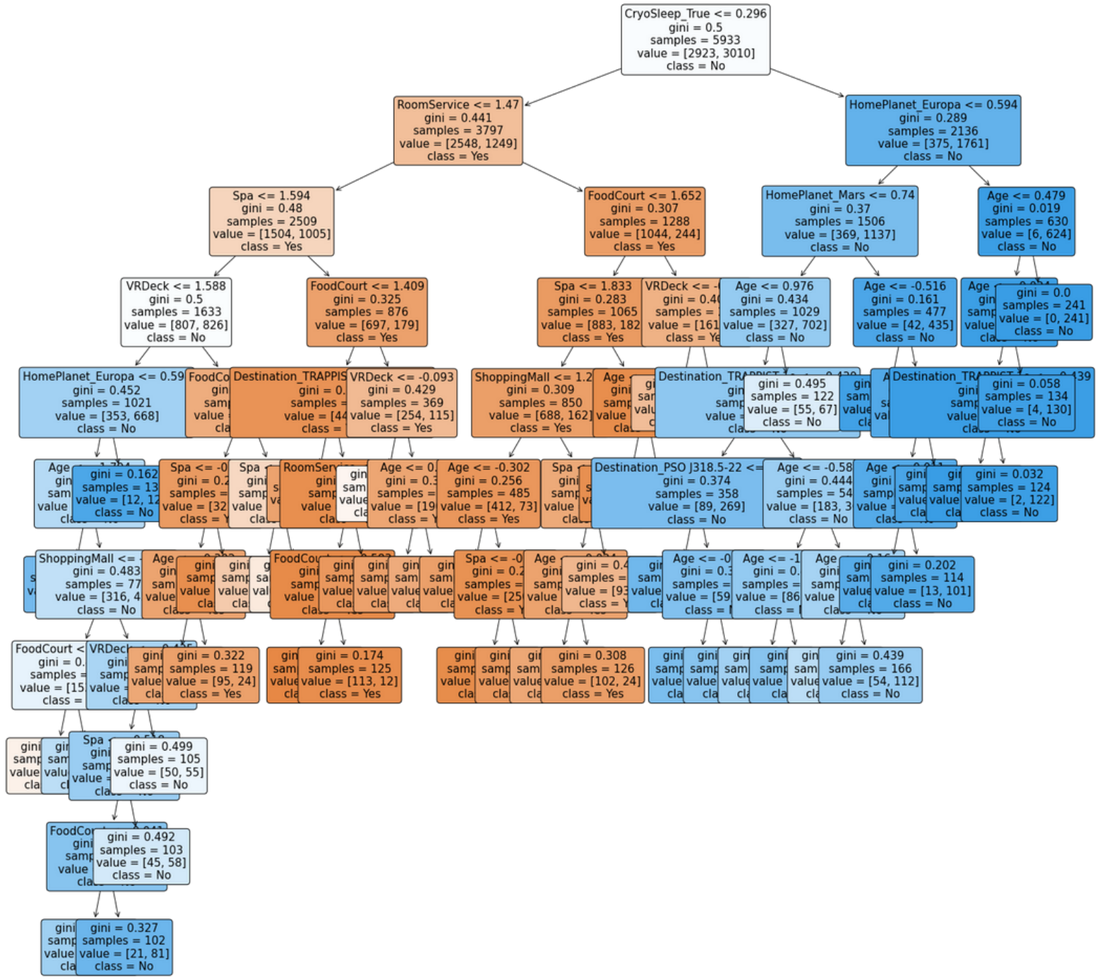
```
Traning score is 0.7810551154559245
Testing score is 0.7660243806527723
```

```
#Performance
print("Accuracy is ",metrics.accuracy_score(y_test,predict_DT_P))
print("precsion is ",metrics.precision_score(y_test,predict_DT_P))
print("recall is ",metrics.recall_score(y_test,predict_DT_P))
```

```
Accuracy is  0.7660243806527723
precsion is  0.7636363636363637
recall is  0.7654516640253566
```

Decision Tree

After Hyperparametric Tunning



Random Forest

```
#Agorithm
RF=RandomForestClassifier(n_estimators=1000)
#Fit
RF.fit(X_train,y_train)
```

```
RandomForestClassifier(n_estimators=1000)
```

```
#Traning and testing score
print("Training score is",RF.score(X_train,y_train))
print("Testing score is",RF.score(X_test,y_test))
```

```
Training score is 0.9367942019214562
Testing score is 0.7526543452615022
```

```
#Performance
print("Accuracy is ",metrics.accuracy_score(y_test,predict_RF))
print("precsion is ",metrics.precision_score(y_test,predict_RF))
print("recall is ",metrics.recall_score(y_test,predict_RF))
```

```
Accuracy is  0.7526543452615022
precsion is  0.7538091419406576
recall is  0.7448494453248812
```

Extra Trees

```
#Algoritm
ET=ExtraTreesClassifier(n_estimators=10)
#Fit
ET.fit(X_train,y_train)
```

```
ExtraTreesClassifier(n_estimators=10)
```

```
#Traning and testing score
print("Training score is",ET.score(X_train,y_train))
print("Testing score is",ET.score(X_test,y_test))
```

```
Training score is 0.9367942019214562
Testing score is 0.7369248918600079
```

```
#Performance
print("Accuracy is ",metrics.accuracy_score(y_test,predict_ET))
print("precsion is ",metrics.precision_score(y_test,predict_ET))
print("recall is ",metrics.recall_score(y_test,predict_ET))
```

```
Accuracy is  0.7369248918600079
precsion is  0.7424366312346689
recall is  0.7194928684627575
```

Bagging Classification

```
#Algorithm
BC=BaggingClassifier()
#Fit
BC.fit(X_train,y_train)
```

```
BaggingClassifier()
```

```
#Traning and testing score
print("Training score is",BC.score(X_train,y_train))
print("Testing score is",BC.score(X_test,y_test))
```

```
Training score is 0.9216248103826058
Testing score is 0.7404640188753441
```

```
#Performance
print("Accuracy is ",metrics.accuracy_score(y_test,predict_BC))
print("precsion is ",metrics.precision_score(y_test,predict_BC))
print("recall is ",metrics.recall_score(y_test,predict_BC))
```

```
Accuracy is  0.7404640188753441
precsion is  0.7443181818181818
recall is  0.7266244057052298
```

Adaboost

```
#Algorithm
AB=AdaBoostClassifier()
#Fit
AB.fit(X_train,y_train)
```

```
AdaBoostClassifier()
```

```
#Traning and testing score
print("Training score is",AB.score(X_train,y_train))
print("Testing score is",AB.score(X_test,y_test))
```

```
Training score is 0.7726276757121187
Testing score is 0.7636649626425481
```

```
#Performance
print("Accuracy is ",metrics.accuracy_score(y_test,predict_AB))
print("precsion is ",metrics.precision_score(y_test,predict_AB))
print("recall is ",metrics.recall_score(y_test,predict_AB))
```

```
Accuracy is  0.7636649626425481
precsion is  0.7540353574173713
recall is  0.777337559429477
```

Gradient Boost

```
#Algorithm
GB=GradientBoostingClassifier()
#Fit
GB.fit(X_train,y_train)
```

```
GradientBoostingClassifier()
```

```
#Traning and testing score
print("Training score is",GB.score(X_train,y_train))
print("Testing score is",GB.score(X_test,y_test))
```

```
Training score is 0.7930220798921288
Testing score is 0.7754620526936689
```

```
#Performance
print("Accuracy is ",metrics.accuracy_score(y_test,predict_GB))
print("precsion is ",metrics.precision_score(y_test,predict_GB))
print("recall is ",metrics.recall_score(y_test,predict_GB))
```

```
Accuracy is  0.7754620526936689
precsion is  0.7584143605086013
recall is  0.803486529318542
```

XgBoost

```
#Algorithm
XG=XGBClassifier()
#Fit
XG.fit(X_train,y_train)
```

```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=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,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

```
#Traning and testing score
print("Training score is",XG.score(X_train,y_train))
print("Testing score is",XG.score(X_test,y_test))
```

```
Training score is 0.8877465026125063
Testing score is 0.7550137632717263
```

```
#Performance
print("Accuracy is ",metrics.accuracy_score(y_test,predict_XG))
print("precsion is ",metrics.precision_score(y_test,predict_XG))
print("recall is ",metrics.recall_score(y_test,predict_XG))
```

```
Accuracy is  0.7550137632717263
precsion is  0.7486381322957198
recall is  0.7622820919175911
```

Result

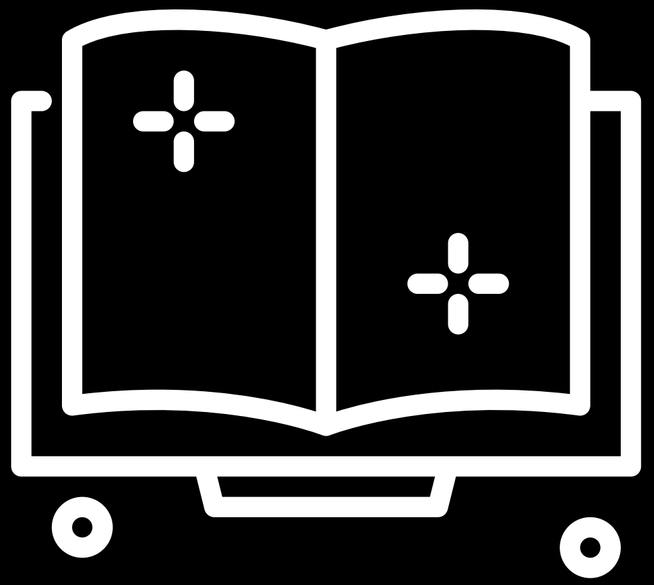
	Algoritm	Accuracy	Precision	Recall	AUC
1	Logistic Regression	0.76	0.74	0.770	0.83
2	Naive Bayes	0.73	0.77	0.690	0.79
3	KNN	0.73	0.74	0.718	0.79
4	SVM	0.76	0.75	0.780	0.82
5	Decision Tree	0.69	0.68	0.740	0.71
6	Decision Tree after Hyerparametric Tunning	0.75	0.74	0.770	0.83
7	Random Forest	0.73	0.73	0.740	0.81
8	Extra Trees	0.72	0.72	0.710	0.78
9	Bagging Classification	0.73	0.72	0.740	0.79
10	Adaboost	0.76	0.74	0.788	0.83
11	Gradient Boost	0.77	0.75	0.790	0.84
12	XgBoost	0.74	0.73	0.770	0.82

Conclusion

The performance of all the alogorithm is good enough.

We are choosing Gradient Boosting for future predictions as it has the highest AUC value which is the best deciding factor when it comes to binomial classification.

Prediction



	Actual value	Predicted value
5245	1	1
1861	0	0
4970	1	0
3064	0	0
7054	1	1
5144	0	0
5474	1	1
3921	0	0
4076	1	0
3500	1	1