```
1 import numpy as np
2 import pandas as pd
3 import seaborn as sns
4 import matplotlib.pyplot as plt
```

```
1 train= pd.read_csv('/content/train.csv')
2 test= pd.read_csv('/content/test.csv')
3 sample= pd.read_csv('/content/sample.csv')
4
5 train.info()
6 train.head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8693 entries, 0 to 8692
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   PassengerId   8693 non-null   object
 1   HomePlanet    8492 non-null   object
 2   CryoSleep     8476 non-null   object
 3   Cabin         8494 non-null   object
 4   Destination   8511 non-null   object
 5   Age           8514 non-null   float64
 6   VIP           8490 non-null   object
 7   RoomService   8512 non-null   float64
 8   FoodCourt     8510 non-null   float64
 9   ShoppingMall  8485 non-null   float64
 10  Spa           8510 non-null   float64
 11  VRDeck        8505 non-null   float64
 12  Name          8493 non-null   object
 13  Transported   8693 non-null   bool
dtypes: bool(1), float64(6), object(7)
memory usage: 891.5+ KB
```

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	S
0	0001_01	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	
1	0002_01	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	9.0	
2	0003_01	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	43.0	3576.0	
3	0003_02	Europa	False	A/0/S	TRAPPIST-1e	33.0	False	0.0	1283.0	
4	0004_01	Earth	False	F/1/S	TRAPPIST-1e	16.0	False	303.0	70.0	

```
numerical_columns = train.select_dtypes(include=['float64', 'int64']).columns
for i in numerical_columns:
    train[i].fillna(train[i].mean(), inplace=True)

categorical_columns = train.select_dtypes(include=['object']).columns
for i in categorical_columns:
    train[i].fillna(train[i].mode()[0], inplace=True)
```

```
train.isnull().sum()
```

```
PassengerId     0
HomePlanet      0
CryoSleep       0
Cabin           0
Destination     0
Age             0
VIP             0
RoomService     0
FoodCourt       0
ShoppingMall    0
Spa             0
VRDeck          0
Name            0
Transported     0
dtype: int64
```

```
train.columns
```

```
Index(['PassengerId', 'HomePlanet', 'CryoSleep', 'Cabin', 'Destination', 'Age',
       'VIP', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa', 'VRDeck',
       'Name', 'Transported'],
      dtype='object')
```

```
train.drop(['PassengerId','Name','Cabin'],axis=1,inplace=True)
train
```

	HomePlanet	CryoSleep	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMa
0	Europa	False	TRAPPIST-1e	39.0	False	0.0	0.0	0
1	Earth	False	TRAPPIST-1e	24.0	False	109.0	9.0	25
2	Europa	False	TRAPPIST-1e	58.0	True	43.0	3576.0	0
3	Europa	False	TRAPPIST-1e	33.0	False	0.0	1283.0	371
4	Earth	False	TRAPPIST-1e	16.0	False	303.0	70.0	151
...	...	...	...	...	...	...	...	
8688	Europa	False	55 Cancri e	41.0	True	0.0	6819.0	0
8689	Earth	True	PSO J318.5-22	18.0	False	0.0	0.0	0
8690	Earth	False	TRAPPIST-1e	26.0	False	0.0	0.0	1872
8691	Europa	False	55 Cancri e	32.0	False	0.0	1049.0	0
8692	Europa	False	TRAPPIST-1e	44.0	False	126.0	4688.0	0

8693 rows × 11 columns

```
1 X=train.drop('Transported',axis=1)
2 Y=train['Transported']
3 X
```

	HomePlanet	CryoSleep	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMa
0	Europa	False	TRAPPIST-1e	39.0	False	0.0	0.0	0
1	Earth	False	TRAPPIST-1e	24.0	False	109.0	9.0	25
2	Europa	False	TRAPPIST-1e	58.0	True	43.0	3576.0	0
3	Europa	False	TRAPPIST-1e	33.0	False	0.0	1283.0	371
4	Earth	False	TRAPPIST-1e	16.0	False	303.0	70.0	151
...	...	...	...	...	...	...	...	
8688	Europa	False	55 Cancri e	41.0	True	0.0	6819.0	0
8689	Earth	True	PSO J318.5-22	18.0	False	0.0	0.0	0
8690	Earth	False	TRAPPIST-1e	26.0	False	0.0	0.0	1872
8691	Europa	False	55 Cancri e	32.0	False	0.0	1049.0	0
8692	Europa	False	TRAPPIST-1e	44.0	False	126.0	4688.0	0

8693 rows × 10 columns

```
from sklearn.preprocessing import LabelEncoder
# Initialize LabelEncoder
label_encoder = LabelEncoder()
# Iterate over each column in your dataset
for column in X.columns:
    # Check if the column dtype is 'object' (i.e., categorical)
    if X[column].dtype == 'object':
        # Encode the values in the column and replace them with encoded values
        X[column] = label_encoder.fit_transform(X[column])
```

```
# This will be needed when we have string values in target column

# Y_encoder = LabelEncoder()
# Y = Y_encoder.fit_transform(train['SalePrice'])
```

```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.2,random_state=1)
```

Classification

```
from sklearn.metrics import accuracy_score #higher is good
from sklearn.metrics import mean_squared_error #lower is good
```

```
from sklearn.linear_model import LogisticRegression
lr= LogisticRegression()
lr.fit(x_train, y_train)
y_pred = lr.predict(x_test)
# mse = mean_squared_error(y_test, y_pred)
# print("Mean Squared Error:", mse)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 79.30%
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458: ConvergenceWarni
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

```
# Decision Tree Classification
from sklearn.tree import DecisionTreeClassifier
dt = DecisionTreeClassifier()
dt.fit(x_train, y_train)
y_pred = dt.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 74.99%
```

```
from sklearn.ensemble import RandomForestClassifier
# rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf = RandomForestClassifier()
rf.fit(x_train, y_train)
y_pred = rf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 78.55%
```

```
from sklearn.ensemble import GradientBoostingClassifier
gb = GradientBoostingClassifier()
gb.fit(x_train,y_train)
y_pred=gb.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 79.70%
```

```
from sklearn.svm import SVC
svc=SVC()
svc.fit(x_train,y_train)
y_pred=svc.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 79.93%
```

```
from xgboost import XGBClassifier
xgb = XGBClassifier()
xgb.fit(x_train,y_train)
y_pred=xgb.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 78.95%
```

```
from xgboost import XGBRFClassifier
xgbrf = XGBRFClassifier()
xgbrf.fit(x_train,y_train)
y_pred=xgbrf.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 79.70%
```

```
from lightgbm import LGBMClassifier
lgb = LGBMClassifier()
lgb.fit(x_train,y_train)
y_pred=lgb.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
[LightGBM] [Info] Number of positive: 3482, number of negative: 3472
[LightGBM] [Info] Auto-choosing row-wise multi-threading, the overhead of testing was 0.001320
You can set `force_row_wise=true` to remove the overhead.
And if memory is not enough, you can set `force_col_wise=true`.
[LightGBM] [Info] Total Bins 1363
[LightGBM] [Info] Number of data points in the train set: 6954, number of used features: 10
[LightGBM] [Info] [binary:BoostFromScore]: pavg=0.500719 -> initscore=0.002876
[LightGBM] [Info] Start training from score 0.002876
Accuracy: 79.36%
```

```
from sklearn.neural_network import MLPClassifier
# mlp = MLPClassifier(hidden_layer_sizes=(100,), max_iter=500)
mlp = MLPClassifier()
mlp.fit(x_train,y_train)
y_pred=mlp.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 78.61%
```

```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(x_train, y_train)
y_pred = knn.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 78.32%
```

```
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(x_train, y_train)
y_pred = nb.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 70.85%
```

```
from sklearn.ensemble import AdaBoostClassifier
ada = AdaBoostClassifier(n_estimators=100)
ada.fit(x_train, y_train)
y_pred = ada.predict(x_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 79.82%
```

```
t=test
```

```
numerical_columns = t.select_dtypes(include=['float64', 'int64']).columns
for i in numerical_columns:
    t[i].fillna(t[i].mean(), inplace=True)

categorical_columns = t.select_dtypes(include=['object']).columns
for i in categorical_columns:
    t[i].fillna(t[i].mode()[0], inplace=True)
```

```
t.isnull().sum()
```

```
PassengerId     0
HomePlanet      0
CryoSleep       0
Cabin           0
Destination     0
Age             0
VIP             0
RoomService     0
FoodCourt       0
ShoppingMall    0
Spa             0
VRDeck          0
Name            0
dtype: int64
```

```
t.drop(['PassengerId','Name','Cabin'],axis=1,inplace=True)
t
```

	HomePlanet	CryoSleep	Destination	Age	VIP	RoomService	FoodCourt	Shopp
0	Earth	True	TRAPPIST-1e	27.000000	False	0.0	0.0	
1	Earth	False	TRAPPIST-1e	19.000000	False	0.0	9.0	
2	Europa	True	55 Cancri e	31.000000	False	0.0	0.0	
3	Europa	False	TRAPPIST-1e	38.000000	False	0.0	6652.0	
4	Earth	False	TRAPPIST-1e	20.000000	False	10.0	0.0	
...	...	...	...	...	...	...	...	
4272	Earth	True	TRAPPIST-1e	34.000000	False	0.0	0.0	
4273	Earth	False	TRAPPIST-1e	42.000000	False	0.0	847.0	
4274	Mars	True	55 Cancri e	28.658146	False	0.0	0.0	
4275	Europa	False	TRAPPIST-1e	28.658146	False	0.0	2680.0	
4276	Earth	True	PSO J318.5-22	43.000000	False	0.0	0.0	

4277 rows × 10 columns

```
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
label_encoder = LabelEncoder()

# Iterate over each column in your dataset
for column in t.columns:
    # Check if the column dtype is 'object' (i.e., categorical)
    if t[column].dtype == 'object':
        # Encode the values in the column and replace them with encoded values
        t[column] = label_encoder.fit_transform(t[column])
```

```
svc.fit(X,Y)
y_pred=svc.predict(test)
```

```
y_pred
```

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

```
# it will bw only used when we have label encoded object Target column

# y_pred_decoded = Y_encoder.inverse_transform(y_pred)
```

```
sample1=sample
```

```
1 sample1['Transported']=y_pred
```

```
1 sample1
```

	PassengerId	Transported
0	0013_01	True
1	0018_01	False
2	0019_01	True
3	0021_01	True
4	0023_01	True
...	...	...
4272	9266_02	True
4273	9269_01	True
4274	9271_01	True
4275	9273_01	True
4276	9277_01	True

4277 rows × 2 columns

```
1 sample1.to_csv('submit_1.csv',index=False)
```

ANN

```
1 x_train = x_train.astype(np.int32)
2 y_train = y_train.astype(np.int32)
3 x_test = x_test.astype(np.int32)
4 y_test = y_test.astype(np.int32)
```

```
1 import tensorflow
2 from tensorflow import keras
3 from tensorflow.keras import Sequential
4 from tensorflow.keras.layers import Dense
```

```
1 model=Sequential()
```

```
model.add(Dense(1000, activation='relu',input_dim=10))
model.add(Dense(1000, activation='relu'))
model.add(Dense(1000, activation='relu'))
model.add(Dense(500, activation='relu'))
model.add(Dense(500, activation='relu'))
model.add(Dense(100, activation='relu'))
model.add(Dense(1, activation='relu'))
```

```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_14 (Dense)            (None, 1000)              11000

 dense_15 (Dense)            (None, 1000)              1001000

 dense_16 (Dense)            (None, 1000)              1001000

 dense_17 (Dense)            (None, 500)               500500

 dense_18 (Dense)            (None, 500)               250500

 dense_19 (Dense)            (None, 100)               50100

 dense_20 (Dense)            (None, 1)                 101

=================================================================
Total params: 2814201 (10.74 MB)
Trainable params: 2814201 (10.74 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
model.compile(loss='binary_crossentropy',optimizer='Adam',metrics=['accuracy'])
```

```
model.fit(x_train, y_train, batch_size=200, validation_data=(x_test, y_test), epochs=
```

```
Epoch 1/10
35/35 [==============================] - 3s 96ms/step - loss: 7.7236 - accuracy: 0.4993 - val_l
Epoch 2/10
35/35 [==============================] - 5s 138ms/step - loss: 7.7236 - accuracy: 0.4993 - val_
Epoch 3/10
35/35 [==============================] - 3s 85ms/step - loss: 7.7236 - accuracy: 0.4993 - val_l
Epoch 4/10
35/35 [==============================] - 3s 95ms/step - loss: 7.7236 - accuracy: 0.4993 - val_l
Epoch 5/10
35/35 [==============================] - 3s 90ms/step - loss: 7.7236 - accuracy: 0.4993 - val_l
Epoch 6/10
35/35 [==============================] - 4s 129ms/step - loss: 7.7236 - accuracy: 0.4993 - val_
Epoch 7/10
35/35 [==============================] - 3s 86ms/step - loss: 7.7236 - accuracy: 0.4993 - val_l
Epoch 8/10
35/35 [==============================] - 4s 107ms/step - loss: 7.7236 - accuracy: 0.4993 - val_
Epoch 9/10
35/35 [==============================] - 3s 96ms/step - loss: 7.7236 - accuracy: 0.4993 - val_l
```

```
Epoch 10/10
35/35 [==============================] - 4s 128ms/step - loss: 7.7236 - accuracy: 0.4993 - val_
<keras.src.callbacks.History at 0x7b8f5a3e8550>
```

```
1 X =X.astype(np.int32)
2 Y = Y.astype(np.int32)
3 model.fit(X,Y,epochs=10)
```

```
Epoch 1/10
272/272 [==============================] - 11s 40ms/step - loss: 7.7684 - accuracy: 0.4964
Epoch 2/10
272/272 [==============================] - 12s 43ms/step - loss: 7.7684 - accuracy: 0.4964
Epoch 3/10
272/272 [==============================] - 12s 43ms/step - loss: 7.7684 - accuracy: 0.4964
Epoch 4/10
272/272 [==============================] - 12s 43ms/step - loss: 7.7684 - accuracy: 0.4964
Epoch 5/10
272/272 [==============================] - 13s 49ms/step - loss: 7.7684 - accuracy: 0.4964
Epoch 6/10
272/272 [==============================] - 12s 43ms/step - loss: 7.7684 - accuracy: 0.4964
Epoch 7/10
272/272 [==============================] - 17s 62ms/step - loss: 7.7684 - accuracy: 0.4964
Epoch 8/10
272/272 [==============================] - 14s 50ms/step - loss: 7.7684 - accuracy: 0.4964
Epoch 9/10
272/272 [==============================] - 20s 73ms/step - loss: 7.7684 - accuracy: 0.4964
Epoch 10/10
272/272 [==============================] - 19s 69ms/step - loss: 7.7684 - accuracy: 0.4964
<keras.src.callbacks.History at 0x7b8f480e2bc0>
```

```
1 test = test.astype(np.int32)
2 y_pred=model.predict(test)
```

```
134/134 [==============================] - 1s 8ms/step
```

```
1 sample2 = sample
```

```
1 sample2['Transported']=y_pred
```

```
1 sample2
```