title: Classification transported prediction in spaceship titanic

author: Wittawat Muangkot

date: 2022-04-1

Description

this paper will analyze spaceship titanic dataset for prediction passenger trasported to an alternate dimension by following step below;

- Import essential package
- Collect Data
- Basic explore dataset and split data into train and test set
- cleaning missing data
- feature engineering
- feature selection
  - constant feature
  - correlation
  - rfe
- model selection
- model pipeline with preprocess
- hyperparameter
- test set prediction

Import essential package

In [143…

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder ,OneHotEncoder, RobustScaler,StandardScaler
from sklearn.pipeline import Pipeline, make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.feature_selection import VarianceThreshold
from sklearn.feature_selection import SelectFromModel
from sklearn.feature_selection import mutual_info_classif
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.feature_selection import SelectKBest, SelectPercentile
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import GradientBoostingClassifier
from sklearn import metrics
from sklearn.metrics import classification_report
```

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
from sklearn.feature_selection import RFE
import missingno as msn

import warnings

warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_colwidth', None)
```

In [144…

```
train=pd.read_csv('/Users/wittawatmuangkot/Dropbox/Mac/Downloads/spaceship-titanic/train.c
test=pd.read_csv('/Users/wittawatmuangkot/Dropbox/Mac/Downloads/spaceship-titanic/test.csv
train
```

Out[144…

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	Sh
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	
…	…	…	…	…	…	…	…	…	…	…
8688	9276_01	Europa	False	A/98/P	55 Cancri e	41.0	True	0.0	6819.0	
8689	9278_01	Earth	True	G/1499/S	PSO J318.5-22	18.0	False	0.0	0.0	
8690	9279_01	Earth	False	G/1500/S	TRAPPIST-1e	26.0	False	0.0	0.0	
8691	9280_01	Europa	False	E/608/S	55 Cancri e	32.0	False	0.0	1049.0	
8692	9280_02	Europa	False	E/608/S	TRAPPIST-1e	44.0	False	126.0	4688.0	

8693 rows × 14 columns

Check overall missing values of dataset

In [145…

```
train.isnull().mean()
```

Out[145…

```
PassengerId     0.000000
HomePlanet      0.023122
CryoSleep       0.024963
Cabin           0.022892
Destination     0.020936
Age             0.020591
VIP             0.023352
RoomService     0.020821
```

```
FoodCourt       0.021051
ShoppingMall    0.023927
Spa             0.021051
VRDeck          0.021627
Name            0.023007
Transported     0.000000
dtype: float64
```

check data types and number of rows and columns

In [146…

```
train.info()
```

```
<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
```

add indicator for missing columns, set 1 for missing and 0 for not missing. and check the number compare between missing and not missing in each column to find the pattern of missing values.

In [147…

```
miss_col=[x for x in train.columns
          if train[x].isnull().any()]
data=train.copy()
for i in miss_col:
    data[str(i)+'_null']=np.where(data[i].isnull(),1,0)
for i in miss_col:
    a=data.groupby('Transported')[str(i)+'_null'].mean()
    print(a)
```

```
Transported
False    0.022711
True     0.023527
Name: HomePlanet_null, dtype: float64
Transported
False    0.025724
True     0.024212
Name: CryoSleep_null, dtype: float64
Transported
False    0.022943
True     0.022841
Name: Cabin_null, dtype: float64
Transported
False    0.020857
```

```
True     0.021014
Name: Destination_null, dtype: float64
Transported
False    0.020626
True     0.020557
Name: Age_null, dtype: float64
Transported
False    0.022943
True     0.023755
Name: VIP_null, dtype: float64
Transported
False    0.022711
True     0.018958
Name: RoomService_null, dtype: float64
Transported
False    0.019467
True     0.022613
Name: FoodCourt_null, dtype: float64
Transported
False    0.021784
True     0.026039
Name: ShoppingMall_null, dtype: float64
Transported
False    0.021321
True     0.020786
Name: Spa_null, dtype: float64
Transported
False    0.020857
True     0.022385
Name: VRDeck_null, dtype: float64
Transported
False    0.022943
True     0.023070
Name: Name_null, dtype: float64
```

In [148…

```
train.info()
```

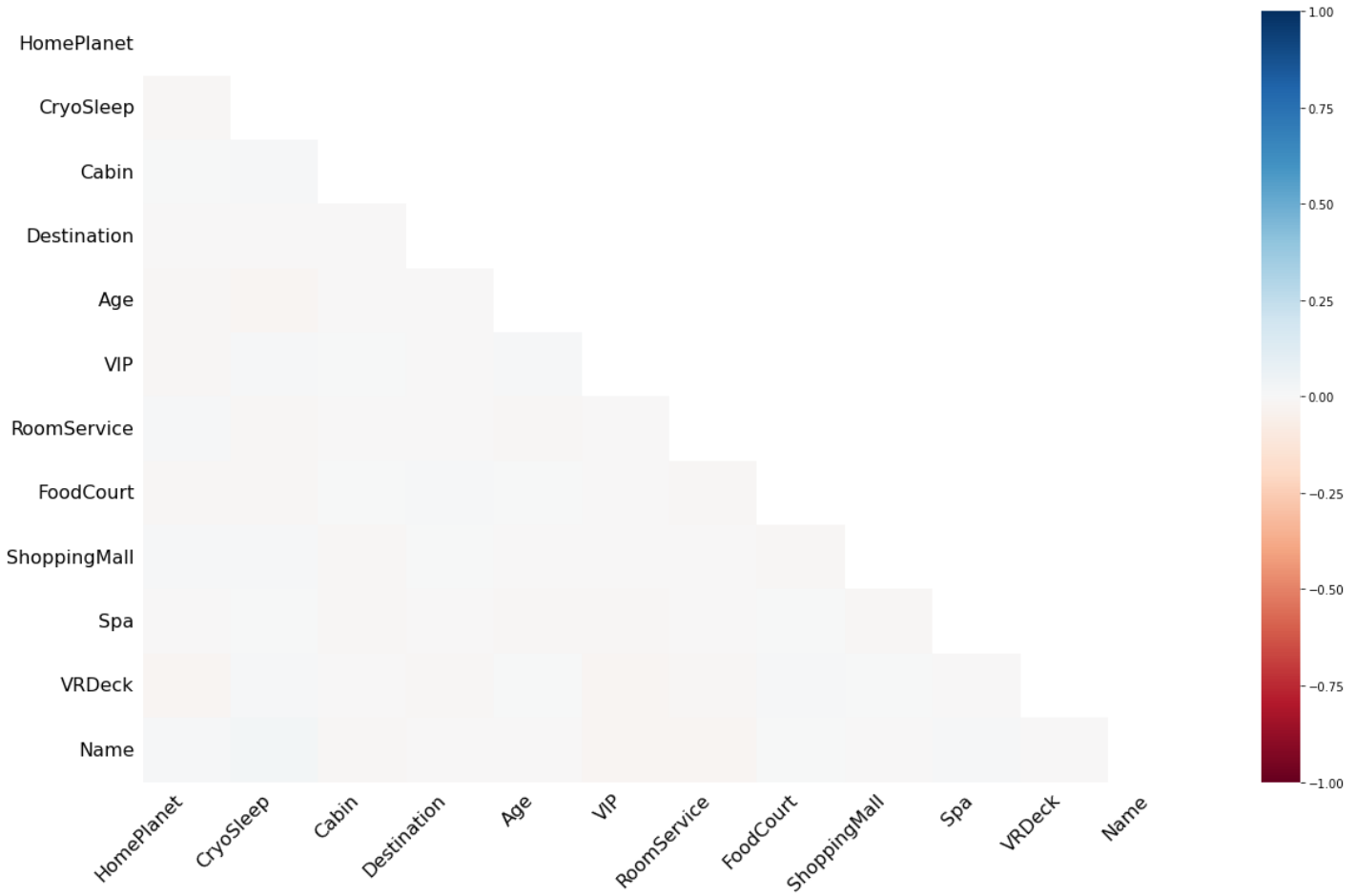
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8693 entries, 0 to 8692
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   PassengerId   8693 non-null   object 
 1   HomePlanet    8492 non-null   object 
 2   CryoSleep     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
```

In [149…

```
msn.heatmap(train)
```

Out[149…

```
<AxesSubplot:>
```

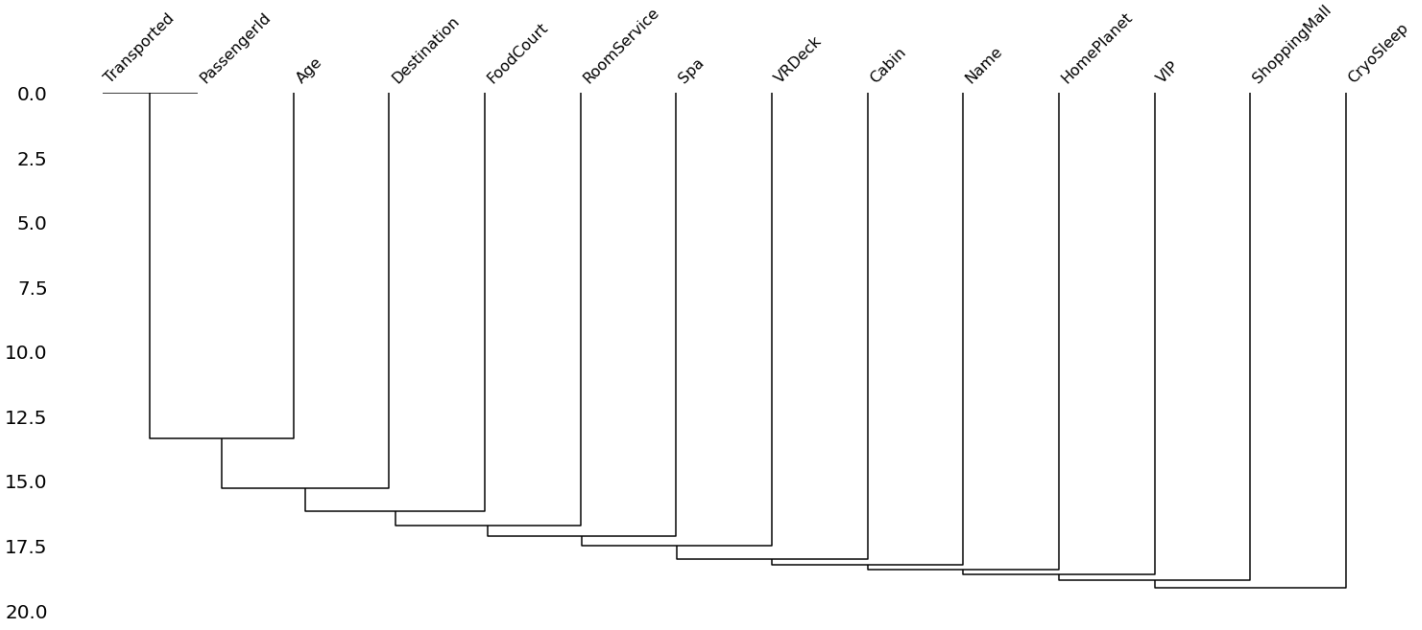


In [150…

```
msn.dendrogram(train)
```

Out[150…

```
<AxesSubplot:>
```

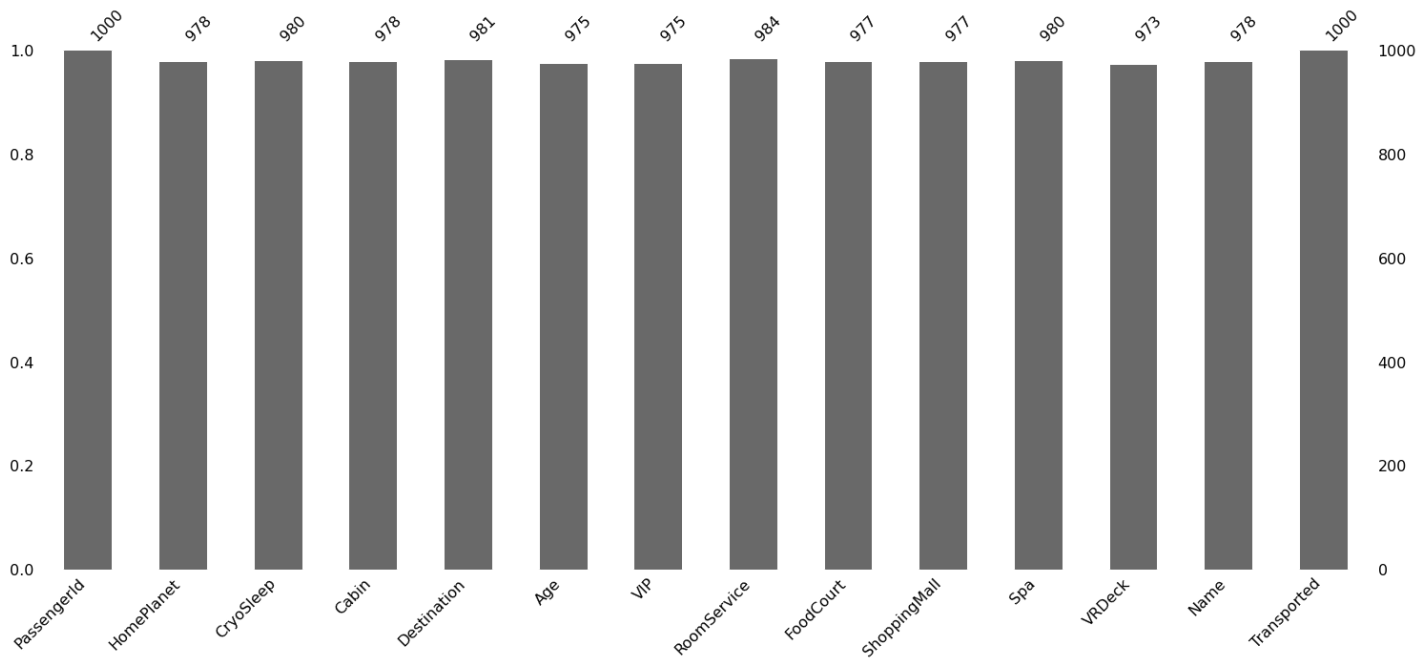


In [151…

```
msn.bar(train.sample(1000))
```

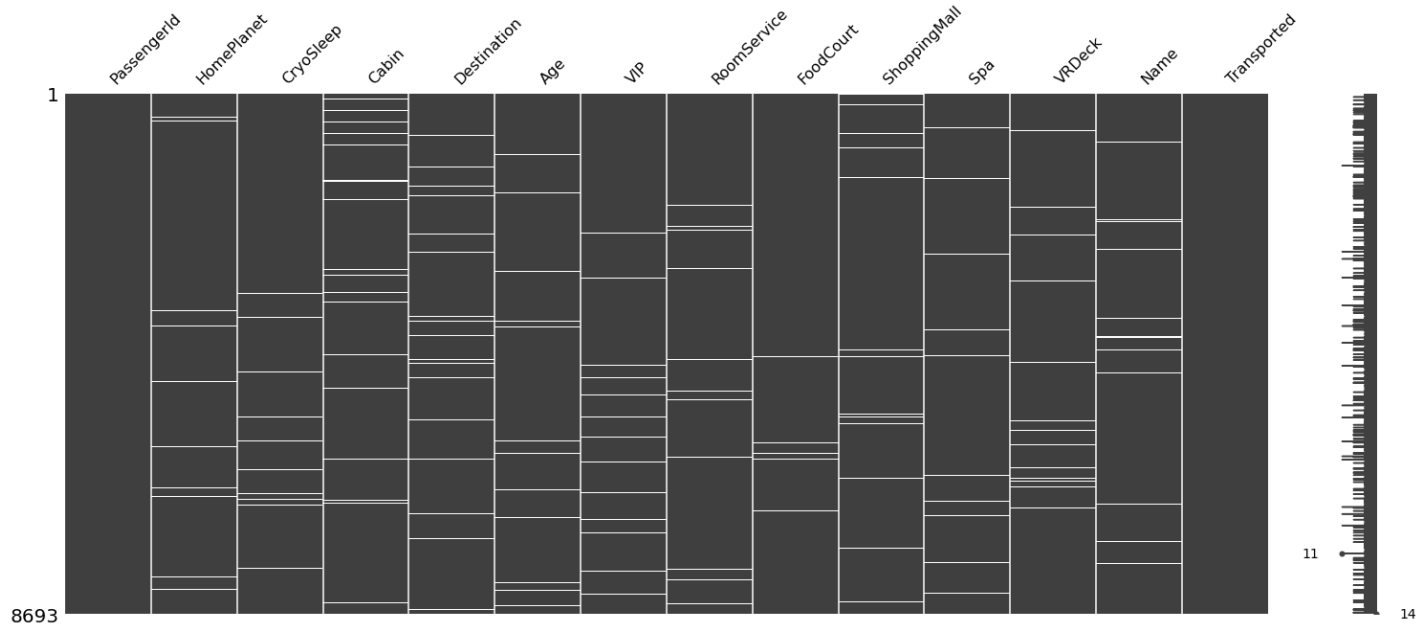
Out[151…

```
<AxesSubplot:>
```



In [152…

```
miss=msn.matrix(train)
```



AS we basic visualize the missig values of train dataset, notics that the missing values is completely random and in each columns has the missing values only around 2 %

Caculate the statistic values for missing group and not missing group

In [153…

```
miss=train[train['Age'].isnull()]
not_miss=train[~train['Age'].isnull()]
```

In [154…

```
miss.describe(include='all')
```

Out[154…

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	S
count	179	177	178	175	176	0.0	173	178.000000	175.000000	
unique	179	3	2	175	3	NaN	2	NaN	NaN	

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	S
top	0052_01	Earth	False	G/6/S	TRAPPIST-1e	NaN	False	NaN	NaN	
freq	1	88	96	1	128	NaN	172	NaN	NaN	
mean	NaN	NaN	NaN	NaN	NaN	NaN	NaN	166.780899	416.651429	
std	NaN	NaN	NaN	NaN	NaN	NaN	NaN	469.211615	1484.930127	
min	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000	
25%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000	
50%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.000000	0.000000	
75%	NaN	NaN	NaN	NaN	NaN	NaN	NaN	7.500000	25.500000	
max	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3478.000000	13342.000000	

In [155…

```
not_miss.describe(include='all')
```

Out[155…

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	Fo
count	8514	8315	8298	8319	8335	8514.000000	8317	8334.000000	8335
unique	8514	3	2	6451	3	NaN	2	NaN	
top	0001_01	Earth	False	G/734/S	TRAPPIST-1e	NaN	False	NaN	
freq	1	4514	5343	8	5787	NaN	8119	NaN	
mean	NaN	NaN	NaN	NaN	NaN	28.827930	NaN	225.924406	458
std	NaN	NaN	NaN	NaN	NaN	14.489021	NaN	670.267167	1614
min	NaN	NaN	NaN	NaN	NaN	0.000000	NaN	0.000000	0
25%	NaN	NaN	NaN	NaN	NaN	19.000000	NaN	0.000000	0
50%	NaN	NaN	NaN	NaN	NaN	27.000000	NaN	0.000000	0
75%	NaN	NaN	NaN	NaN	NaN	38.000000	NaN	48.000000	79
max	NaN	NaN	NaN	NaN	NaN	79.000000	NaN	14327.000000	29813

Transform the target into the category data type

In [156…

```
train['Transported']=train['Transported'].astype('category')
train['Transported']=train['Transported'].cat.codes
```

split dataset into train and test set for using evaluate model performance

- using the test size 30 % and train size 70 %

In [157…

```
X_train,X_test,y_train,y_test=train_test_split(train.drop('Transported',axis=1),train['Tra
```

In [158…

```
num=[x for x in train.columns
    if train[x].dtypes!='O' and x  != 'Transported']
```

```
cat=[c for c in train.columns
    if c not in num and c != 'Transported']
```

For imputing the Missing values in feature RoomService, FoodCourt, ShoopingMall, Spa, VRDeck we'll fill with 0 as we assume that the missing values is no using service for those service

In [159…

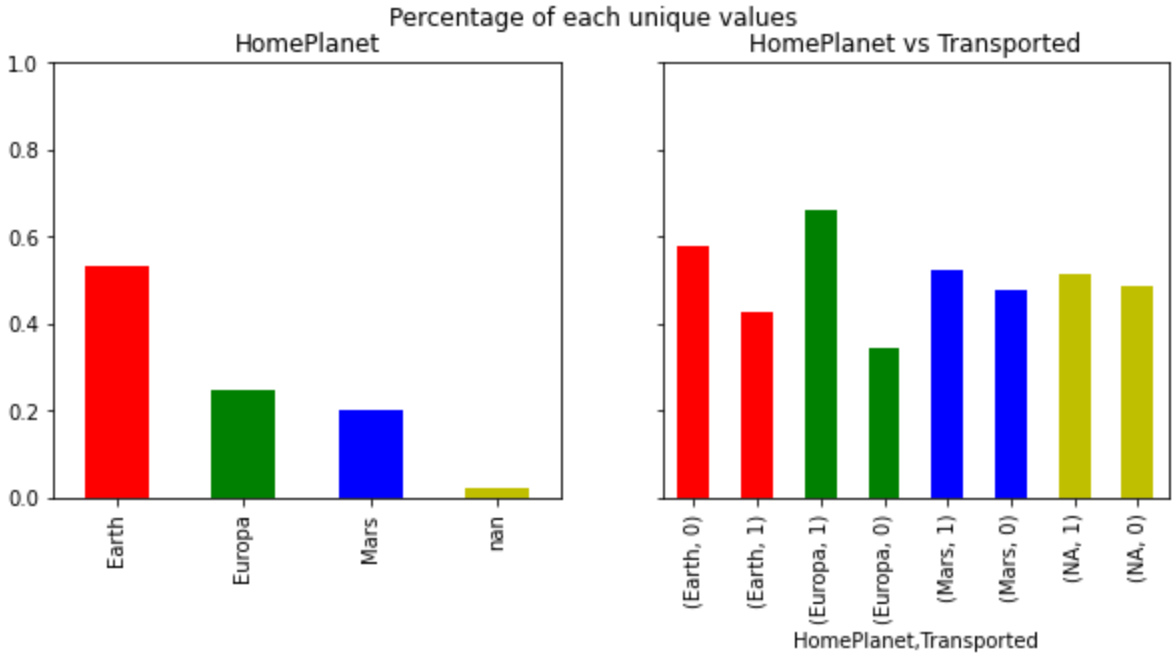
```
list_service=['RoomService', 'FoodCourt', 'ShoppingMall','Spa', 'VRDeck']
for i in list_service:
    X_train[i]=X_train[i].fillna(0)
    X_test[i]=X_test[i].fillna(0)
    test[i]=test[i].fillna(0)
```

For the other missing values columns, as the dataset is completely random missing and only 2% of missing, we'll decide to using the median and mode of each missing columns for filling and set the indicator that tell the missing columns

Homeplanet columns missing imputation

In [160…

```
fig,ax=plt.subplots(1,2,figsize=(10,4),sharey=True)
count_val=train['HomePlanet'].value_counts(dropna=False,normalize=True)
count_val.plot(kind='bar',ax=ax[0],color=['r','g','b','y'])
train_fill=train[['HomePlanet','Transported']]
train_fill['HomePlanet']=train['HomePlanet'].fillna('NA')
train_fill.groupby('HomePlanet')['Transported'].value_counts(normalize=True).plot(kind='ba
plt.ylim((0,1));
plt.suptitle('Percentage of each unique values');
ax[0].title.set_text('HomePlanet')
ax[1].title.set_text('HomePlanet vs Transported')
```



As the above left chart showed the most value of homeplanet columns is "Earth" has around 55 % and the right chart showed the pattern of Homeplanet columns values by group of target columns ("Transported" or

"not Trasported") ,so we'll impute this missing columns as the most frequency values of this columns in train dataset and transform to another dataset.

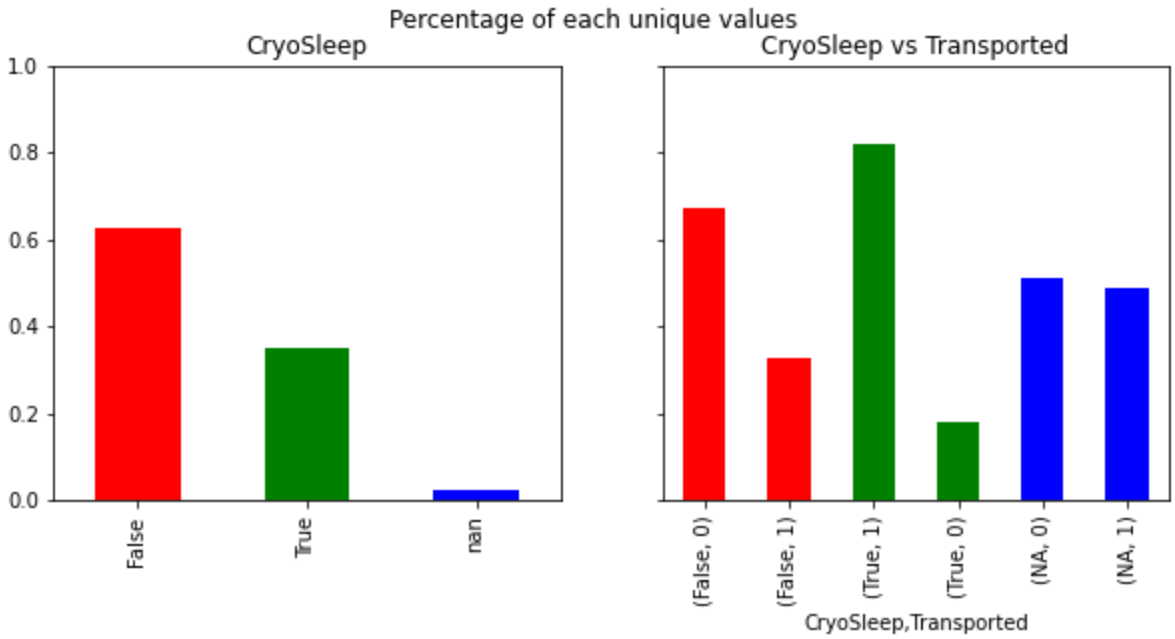
In [161…

```
X_train['HomePlanet'].fillna('Earth',inplace=True)
X_test['HomePlanet'].fillna('Earth',inplace=True)
test['HomePlanet'].fillna('Earth',inplace=True)
```

CryoSleep columns missing imputation

In [162…

```
fig,ax=plt.subplots(1,2,figsize=(10,4),sharey=True)
count_val=train['CryoSleep'].value_counts(dropna=False,normalize=True)
count_val.plot(kind='bar',ax=ax[0],color=['r','g','b'])
train_fill=train[['CryoSleep','Transported']]
train_fill['CryoSleep']=train['CryoSleep'].fillna('NA')
train_fill.groupby('CryoSleep')['Transported'].value_counts(normalize=True).plot(kind='ba
plt.ylim((0,1));
plt.suptitle('Percentage of each unique values');
ax[0].title.set_text('CryoSleep')
ax[1].title.set_text('CryoSleep vs Transported')
```



As the above left chart showed the most value of CryoSleep columns is "False" has around 61 % and the right chart showed the pattern of CryoSleep columns values by group of target columns ("Transported" or "not Trasported") ,so we'll impute this missing columns as the most frequency values of this columns in train dataset and transform to another dataset.

In [163…

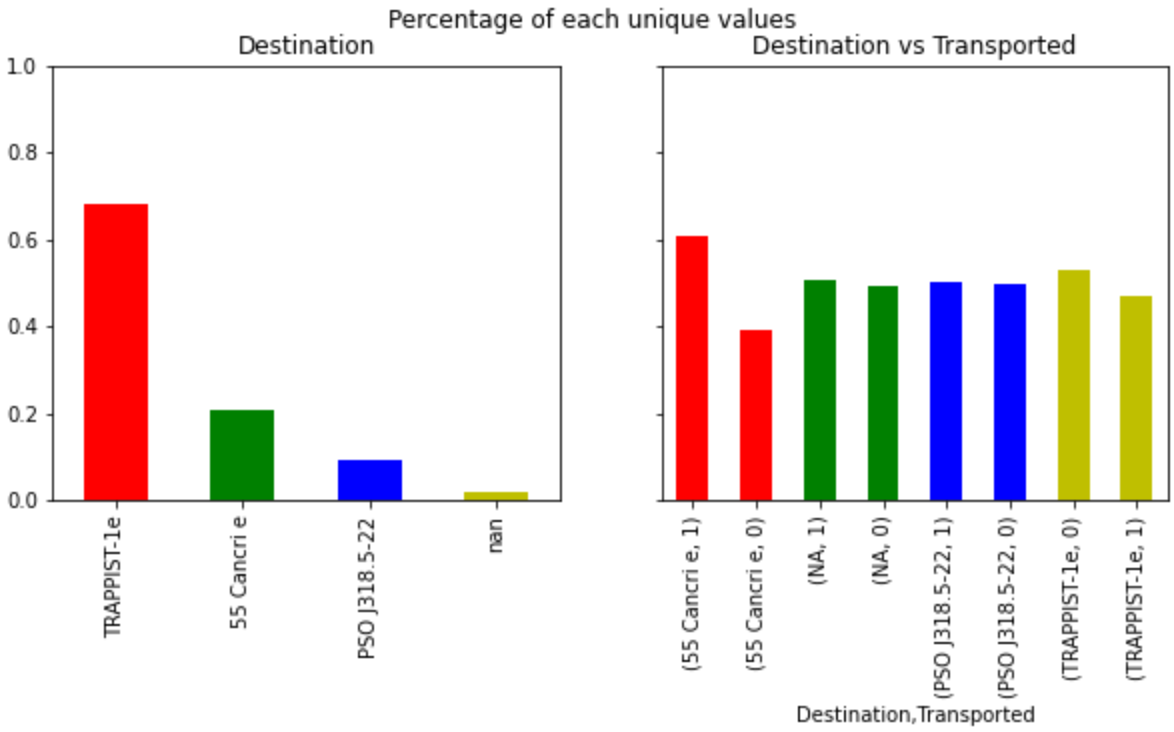
```
X_train['CryoSleep'].fillna(False,inplace=True)
X_test['CryoSleep'].fillna(False,inplace=True)
test['CryoSleep'].fillna(False,inplace=True)
```

Destination columns missing imputation

In [164…

```
fig,ax=plt.subplots(1,2,figsize=(10,4),sharey=True)
count_val=train['Destination'].value_counts(dropna=False,normalize=True)
count_val.plot(kind='bar',ax=ax[0],color=['r','g','b','y'])
train_fill=train[['Destination','Transported']]
train_fill['Destination']=train['Destination'].fillna('NA')
```

```
train_fill.groupby('Destination')['Transported'].value_counts(normalize=True).plot(kind='b
plt.ylim((0,1));
plt.suptitle('Percentage of each unique values');
ax[0].title.set_text('Destination')
ax[1].title.set_text('Destination vs Transported')
```



As the above left chart showed the most value of Destination columns is "TRAPPIST-1e" has around 70 % and the right chart showed the pattern of Destination columns values by group of target columns ("Transported" or "not Trasported") ,so we'll impute this missing columns as the most frequency values of this columns in train dataset and transform to another dataset.

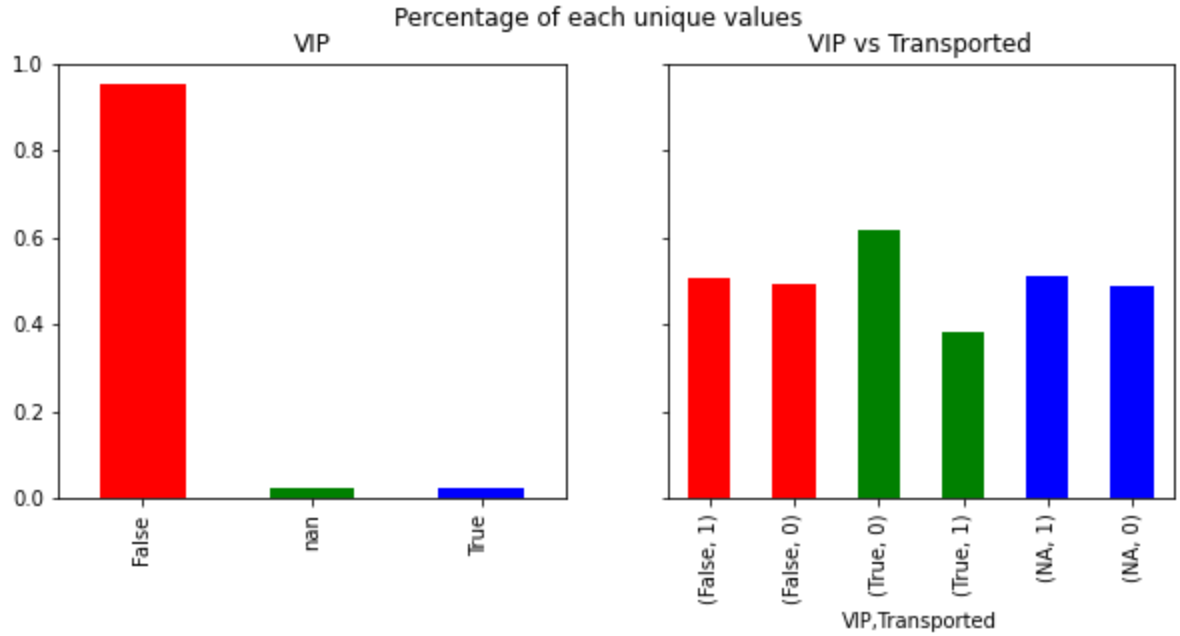
In [165…

```
X_train['Destination'].fillna('TRAPPIST-1e',inplace=True)
X_test['Destination'].fillna('TRAPPIST-1e',inplace=True)
test['Destination'].fillna('TRAPPIST-1e',inplace=True)
```

VIP columns missing imputation

In [166…

```
fig,ax=plt.subplots(1,2,figsize=(10,4),sharey=True)
count_val=train['VIP'].value_counts(dropna=False,normalize=True)
count_val.plot(kind='bar',ax=ax[0],color=['r','g','b'])

train_fill=train[['VIP','Transported']]
train_fill['VIP']=train['VIP'].fillna('NA')
train_fill.groupby('VIP')['Transported'].value_counts(normalize=True).plot(kind='bar',ax=a
plt.ylim((0,1));
plt.suptitle('Percentage of each unique values');
ax[0].title.set_text('VIP')
ax[1].title.set_text('VIP vs Transported')
```



As the above left chart showed the most value of VIP columns is "False" has around 95 % and the right chart showed the pattern of VIP columns values by group of target columns ("Transported" or "not Trasported") ,so we'll impute this missing columns as the most frequency values of this columns in train dataset and transform to another dataset.

In [167…
```
X_train['VIP'].fillna(False,inplace=True)
X_test['VIP'].fillna(False,inplace=True)
test['VIP'].fillna(False,inplace=True)
```

Age columns will inputed with the median values

In [168…
```
median=X_train['Age'].median()
median
```

Out[168…
```
27.0
```

In [169…
```
X_train['Age'].fillna(median,inplace=True)
X_test['Age'].fillna(median,inplace=True)
test['Age'].fillna(median,inplace=True)
```

Cabin columns missing impute

as this columns has the combline value string and number, first thing we need to da is to split the value string and number into new columns and next impute the missing valumns in each columns individual

In [170…
```
train[["Deck","Num","Side"]]=train['Cabin'].str.split('/',expand=True)
X_train[["Deck","Num","Side"]]=X_train['Cabin'].str.split('/',expand=True)
X_test[["Deck","Num","Side"]]=X_test['Cabin'].str.split('/',expand=True)
test[["Deck","Num","Side"]]=test['Cabin'].str.split('/',expand=True)
```

Separate cabine columns to 3 new columns

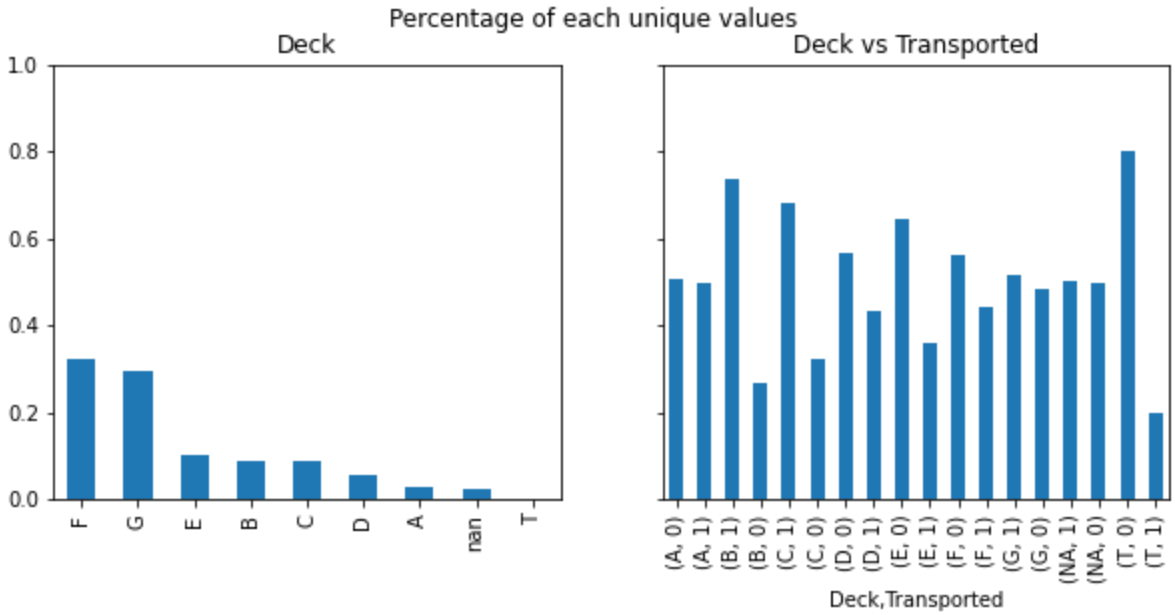
- Deck
- Num

- Side

For deck columns as the chart below showed that the most frequency values in this columns is "F"

In [171…

```
fig,ax=plt.subplots(1,2,figsize=(10,4),sharey=True)
count_val=train['Deck'].value_counts(dropna=False,normalize=True)
count_val.plot(kind='bar',ax=ax[0])

train_fill=train[['Deck','Transported']]
train_fill['Deck']=train['Deck'].fillna('NA')
train_fill.groupby('Deck')['Transported'].value_counts(normalize=True).plot(kind='bar',ax=
plt.ylim((0,1));
plt.suptitle('Percentage of each unique values');
ax[0].title.set_text('Deck')
ax[1].title.set_text('Deck vs Transported')
```



In [172…

```
X_train['Deck'].fillna('F',inplace=True)
X_test['Deck'].fillna('F',inplace=True)
test['Deck'].fillna('F',inplace=True)
```

In [173…

```
X_train.groupby(['VIP','Destination'])['Deck'].value_counts()
```

Out[173…

```
VIP    Destination     Deck
False  55 Cancri e     F        313
                       G        256
                       B        220
                       C        217
                       E         77
                       D         74
                       A         68
       PSO J318.5-22   G        316
                       F        186
                       E         31
                       D          9
                       C          7
                       A          1
                       B          1
       TRAPPIST-1e     F       1601
                       G       1206
                       E        484
                       B        296
```

```
                      C          275
                      D          223
                      A           80
                      T            5
True    55 Cancri e   B           18
                      C            9
                      A            7
                      D            4
                      F            2
                      E            1
        PSO J318.5-22 C            5
                      F            4
                      D            3
                      A            1
                      B            1
                      E            1
        TRAPPIST-1e   F           17
                      D           16
                      A           14
                      C           14
                      B           13
                      E            9
Name: Deck, dtype: int64
```

As Deck T is the Rare labels (only 5 population) so we will group Deck T to Deck F as all Deck T has Destication to TRAPPIST-1e like Deck F the most population has Destination to TRAPPIST-1e

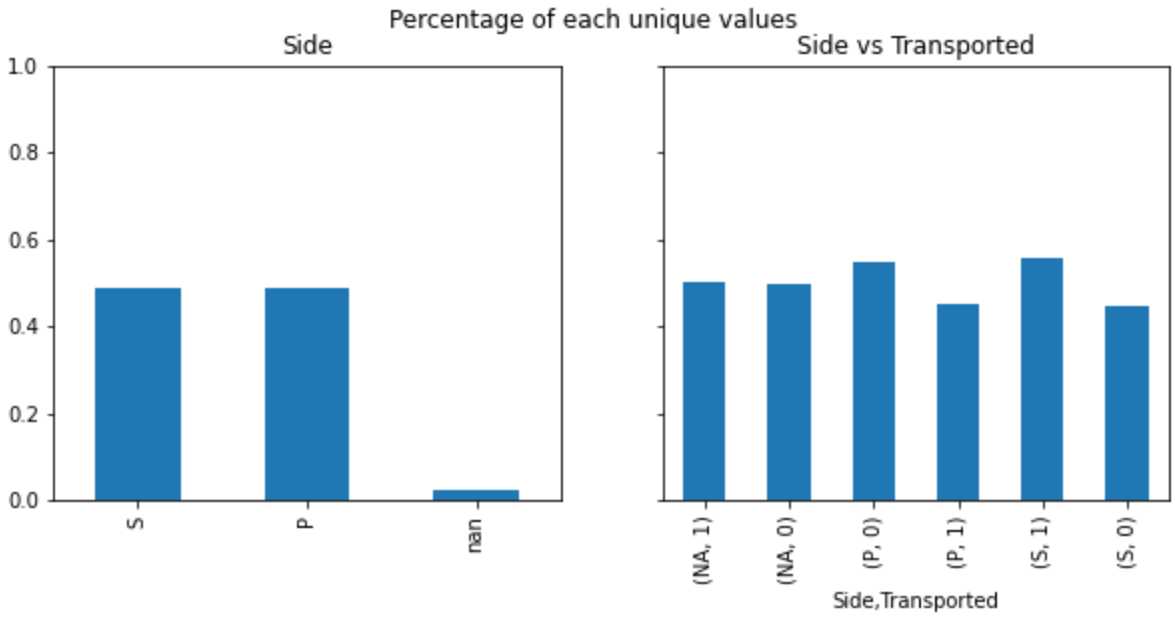
In [174…

```
idx = X_train[X_train['Deck'] == 'T'].index
X_train.loc[idx, 'Deck'] = 'F'

idxt = test[test['Deck'] == 'T'].index
test.loc[idxt, 'Deck'] = 'F'
```

For the side columns imputation, as the chart below showed that the most frequency is "S" and "P" and the trasported pattern of these 2 values are same so we'll use "S" for impute the missing values

In [175…

```
fig,ax=plt.subplots(1,2,figsize=(10,4),sharey=True)
count_val=X_train['Side'].value_counts(dropna=False,normalize=True)
count_val.plot(kind='bar',ax=ax[0])

train_fill=train[['Side','Transported']]
train_fill['Side']=train['Side'].fillna('NA')
train_fill.groupby('Side')['Transported'].value_counts(normalize=True).plot(kind='bar',ax=
plt.ylim((0,1));
plt.suptitle('Percentage of each unique values');
ax[0].title.set_text('Side')
ax[1].title.set_text('Side vs Transported')
```



In [176…

```
X_train['Side'].fillna('S',inplace=True)
X_test['Side'].fillna('S',inplace=True)
test['Side'].fillna('S',inplace=True)
```

For the cardinity of Num columns is high so we'll use the random sampling data for impute this columne

In [177…

```
random_sample_train = X_train['Num'].dropna().sample(
    X_train['Num'].isnull().sum(), random_state=42)

random_sample_test = X_train['Num'].dropna().sample(
    X_test['Num'].isnull().sum(), random_state=42)

random_sample = X_train['Num'].dropna().sample(
    test['Num'].isnull().sum(), random_state=42)

random_sample_train.index = X_train[X_train['Num'].isnull()].index
random_sample_test.index = X_test[X_test['Num'].isnull()].index
random_sample.index = test[test['Num'].isnull()].index

X_train.loc[X_train['Num'].isnull(), 'Num'] = random_sample_train
X_test.loc[X_test['Num'].isnull(), 'Num'] = random_sample_test
test.loc[test['Num'].isnull(), 'Num'] = random_sample
```

Split the Name columns to First name and last name for further analysis

In [178…

```
X_train[["Fname","Lname"]]=X_train['Name'].str.split(' ',expand=True)
X_test[["Fname","Lname"]]=X_test['Name'].str.split(' ',expand=True)
test[["Fname","Lname"]]=test['Name'].str.split(' ',expand=True)
X_train.drop('Fname',axis=1,inplace=True)
X_test.drop('Fname',axis=1,inplace=True)
test.drop('Fname',axis=1,inplace=True)
```

Random sampling for impute missing value in Lname colums for using for invent new columns in feature engineer process to improve model performance

In [179…
```
random_sample_train = X_train['Lname'].dropna().sample(
    X_train['Lname'].isnull().sum(), random_state=42)

random_sample_test = X_train['Lname'].dropna().sample(
    X_test['Lname'].isnull().sum(), random_state=42)

random_sample = X_train['Lname'].dropna().sample(
    test['Lname'].isnull().sum(), random_state=42)

random_sample_train.index = X_train[X_train['Lname'].isnull()].index
random_sample_test.index = X_test[X_test['Lname'].isnull()].index
random_sample.index = test[test['Lname'].isnull()].index

X_train.loc[X_train['Lname'].isnull(), 'Lname'] = random_sample_train
X_test.loc[X_test['Lname'].isnull(), 'Lname'] = random_sample_test
test.loc[test['Lname'].isnull(), 'Lname'] = random_sample
```

drop unuse columns

In [180…
```
X_train.drop(['Cabin','Name'],axis=1,inplace=True)
X_test.drop(['Cabin','Name'],axis=1,inplace=True)
test.drop(['Cabin','Name'],axis=1,inplace=True)
```

Set indicator for missing columns

using 1 for missing indicate and 0 for not missing indicated

In [181…
```
miss_col=['Lname','Num','Side','Deck','Age','CryoSleep','Destination','HomePlanet']
for i in miss_col:
    X_train[str(i)+'_null']=np.where(X_train[i].isnull(),1,0)
    X_test[str(i)+'_null']=np.where(X_test[i].isnull(),1,0)
    test[str(i)+'_null']=np.where(test[i].isnull(),1,0)
```

First model fitting data for first evaluate dataset

In [182…
```
num=[x for x in X_train.columns
    if X_train[x].dtypes!='O' and x  != 'Transported']
cat=[c for c in X_train.columns
    if c not in num and c != 'Transported']
```

we'll using the model pipeline for data preparation in scale and endoing step

In [183…
```
numerical_transformer = Pipeline(steps=[

    ('scale', StandardScaler())
])

categorical_transformer = Pipeline(steps=[

    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num',numerical_transformer,num),
```

```
        ('cat', categorical_transformer, cat)
    ])
```

use the first model the logistic regression to fit this first time model

In [184…
```
model=LogisticRegression(C=0.5, solver='liblinear')
```

In [185…
```
my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('lr', model)
                             ])
```

In [186…
```
my_pipeline.fit(X_train,y_train)
```

Out[186…
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('scale',
                                                                   StandardScaler())]),
                                                  ['CryoSleep', 'Age', 'VIP',
                                                   'RoomService', 'FoodCourt',
                                                   'ShoppingMall', 'Spa',
                                                   'VRDeck', 'Lname_null',
                                                   'Num_null', 'Side_null',
                                                   'Deck_null', 'Age_null',
                                                   'CryoSleep_null',
                                                   'Destination_null',
                                                   'HomePlanet_null']),
                                                 ('cat',
                                                  Pipeline(steps=[('onehot',
                                                                   OneHotEncoder(handle_un
known='ignore'))]),
                                                  ['PassengerId', 'HomePlanet',
                                                   'Destination', 'Deck', 'Num',
                                                   'Side', 'Lname'])])),
                ('lr', LogisticRegression(C=0.5, solver='liblinear'))])
```

In [187…
```
pred=my_pipeline.predict(X_test)
metrics.accuracy_score(y_test,pred)
```

Out[187…
```
0.7802914110429447
```

In [188…
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.75      0.77      1289
           1       0.77      0.81      0.79      1319

    accuracy                           0.78      2608
   macro avg       0.78      0.78      0.78      2608
weighted avg       0.78      0.78      0.78      2608
```

mode show the accuracy around 78.03% for this prediction model we try to predict the trasported of passengers, so focus on the 1

- precision = 77% is the moodel can predict the transported correct 77% from total transported predicted
- recall = 81% is the model can predict the actual transported 79% from total actual transported

- f1-score = 79%

use model to predict the probability use for calculate the ROC curve

In [189…

```
y_scores = my_pipeline.predict_proba(X_test)
print(y_scores)
```

```
[[7.14063073e-01 2.85936927e-01]
 [4.47058244e-01 5.52941756e-01]
 [2.33135858e-01 7.66864142e-01]
 ...
 [9.99999575e-01 4.24749875e-07]
 [1.62029936e-01 8.37970064e-01]
 [3.60969608e-01 6.39030392e-01]]
```

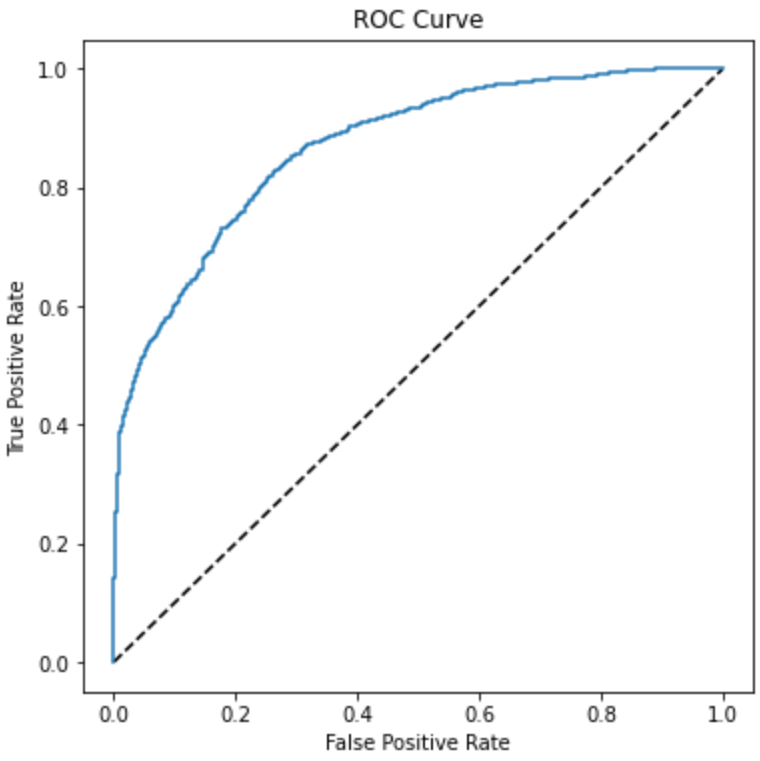
In [190…

```
auc = roc_auc_score(y_test,y_scores[:,1])
print(auc)
```

```
0.8690529475805954
```

In [191…

```
fpr, tpr, thresholds = roc_curve(y_test, y_scores[:,1])

# plot ROC curve
fig = plt.figure(figsize=(6, 6))
# Plot the diagonal 50% line
plt.plot([0, 1], [0, 1], 'k--')
# Plot the FPR and TPR achieved by our model
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```



Feature Egineering

Let improve model performance by invent the new columnns

In [192…

```
X_train['Avg_amenity_bill']=X_train[['RoomService','FoodCourt','ShoppingMall','Spa','VRDec
X_test['Avg_amenity_bill']=X_test[['RoomService','FoodCourt','ShoppingMall','Spa','VRDeck'
test['Avg_amenity_bill']=test[['RoomService','FoodCourt','ShoppingMall','Spa','VRDeck']].m

X_train['total_amenity_bill']=X_train[['RoomService','FoodCourt','ShoppingMall','Spa','VRD
X_test['total_amenity_bill']=X_test[['RoomService','FoodCourt','ShoppingMall','Spa','VRDec
test['total_amenity_bill']=test[['RoomService','FoodCourt','ShoppingMall','Spa','VRDeck']]
```

In [193…

```
X_train['RoomService_use']=X_train['RoomService']!=0
X_train['FoodCourt_use']=X_train['FoodCourt']!=0
X_train['ShoppingMall_use']=X_train['ShoppingMall']!=0
X_train['Spa_use']=X_train['Spa']!=0
X_train['VRDeck_use']=X_train['VRDeck']!=0
X_train['Tol_amenity_use']=X_train[['RoomService_use','FoodCourt_use','ShoppingMall_use','

X_test['RoomService_use']=X_test['RoomService']!=0
X_test['FoodCourt_use']=X_test['FoodCourt']!=0
X_test['ShoppingMall_use']=X_test['ShoppingMall']!=0
X_test['Spa_use']=X_test['Spa']!=0
X_test['VRDeck_use']=X_test['VRDeck']!=0
X_test['Tol_amenity_use']=X_test[['RoomService_use','FoodCourt_use','ShoppingMall_use','Sp

test['RoomService_use']=test['RoomService']!=0
test['FoodCourt_use']=test['FoodCourt']!=0
test['ShoppingMall_use']=test['ShoppingMall']!=0
test['Spa_use']=test['Spa']!=0
test['VRDeck_use']=test['VRDeck']!=0
test['Tol_amenity_use']=test[['RoomService_use','FoodCourt_use','ShoppingMall_use','Spa_us
```

In [194…

```
X_train[["Group","Id"]]=X_train['PassengerId'].str.split('_',expand=True)
X_train['Group_no']=X_train.groupby('Group')['Group'].transform('count')

X_test[["Group","Id"]]=X_test['PassengerId'].str.split('_',expand=True)
X_test['Group_no']=X_test.groupby('Group')['Group'].transform('count')

test[["Group","Id"]]=test['PassengerId'].str.split('_',expand=True)
test['Group_no']=test.groupby('Group')['Group'].transform('count')
```

In [195…

```
X_train['Family_no']=X_train.groupby(['Group','Lname'])['Lname'].transform('count')

X_test['Family_no']=X_test.groupby(['Group','Lname'])['Lname'].transform('count')

test['Family_no']=test.groupby(['Group','Lname'])['Lname'].transform('count')
```

In [196…

```
X_train['Travel_alone']=X_train['Group_no']==1

X_test['Travel_alone']=X_test['Group_no']==1

test['Travel_alone']=test['Group_no']==1
```

In [197…

```
X_train['Nofamily']=X_train['Family_no']==1

X_test['Nofamily']=X_test['Family_no']==1

test['Nofamily']=test['Family_no']==1
```

In [198…

```
X_train
```

Out[198…

	PassengerId	HomePlanet	CryoSleep	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMal
3032	3282_03	Europa	False	TRAPPIST-1e	43.0	False	0.0	1440.0	0.0
7757	8276_02	Europa	True	TRAPPIST-1e	23.0	False	0.0	0.0	0.0
1795	1911_01	Earth	False	TRAPPIST-1e	46.0	False	8.0	652.0	0.0
1702	1808_01	Earth	False	TRAPPIST-1e	33.0	False	0.0	763.0	8.0
6634	6995_01	Earth	False	55 Cancri e	24.0	False	0.0	58.0	618.0
...	...	...	...	...	...	...	...	...	..
5734	6076_01	Earth	False	TRAPPIST-1e	18.0	False	14.0	2.0	144.0
5191	5537_01	Mars	False	TRAPPIST-1e	50.0	False	690.0	0.0	30.0
5390	5756_06	Earth	False	PSO J318.5-22	22.0	False	158.0	0.0	476.0
860	0925_01	Mars	False	TRAPPIST-1e	34.0	False	379.0	0.0	1626.0
7270	7775_01	Europa	False	55 Cancri e	28.0	False	7.0	489.0	0.0

6085 rows × 37 columns

In [199…

```
num=[x for x in X_train.columns
    if X_train[x].dtypes!='O' and x  != 'Transported']
cat=[c for c in X_train.columns
    if c not in num and c != 'Transported']
```

In [200…

```
numerical_transformer = Pipeline(steps=[

    ('scale', StandardScaler())
])

categorical_transformer = Pipeline(steps=[

    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num',numerical_transformer,num),

        ('cat', categorical_transformer, cat)
    ])
```

In [201…

```
model=LogisticRegression(C=0.5, solver='liblinear')
```

In [202…

```
my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('lr', model)
```

```
                                ])
```

In [203…

```
my_pipeline.fit(X_train,y_train)
```

Out[203…

```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('scale',
                                                                   StandardScaler())]),
                                                  ['CryoSleep', 'Age', 'VIP',
                                                   'RoomService', 'FoodCourt',
                                                   'ShoppingMall', 'Spa',
                                                   'VRDeck', 'Lname_null',
                                                   'Num_null', 'Side_null',
                                                   'Deck_null', 'Age_null',
                                                   'CryoSleep_null',
                                                   'Destination_null',
                                                   'HomePlanet_null',
                                                   'Avg_amenity_bill',
                                                   'total_amenity_bill',
                                                   'RoomService_use',
                                                   'FoodCourt_use',
                                                   'ShoppingMall_use',
                                                   'Spa_use', 'VRDeck_use',
                                                   'Tol_amenity_use',
                                                   'Group_no', 'Family_no',
                                                   'Travel_alone',
                                                   'Nofamily']),
                                                 ('cat',
                                                  Pipeline(steps=[('onehot',
                                                                   OneHotEncoder(handle_un
known='ignore'))]),
                                                  ['PassengerId', 'HomePlanet',
                                                   'Destination', 'Deck', 'Num',
                                                   'Side', 'Lname', 'Group',
                                                   'Id'])])),
                ('lr', LogisticRegression(C=0.5, solver='liblinear'))])
```

In [204…

```
pred=my_pipeline.predict(X_test)
metrics.accuracy_score(y_test,pred)
```

Out[204…

```
0.7848926380368099
```

In [205…

```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.80      0.75      0.78      1289
           1       0.77      0.82      0.79      1319

    accuracy                           0.78      2608
   macro avg       0.79      0.78      0.78      2608
weighted avg       0.79      0.78      0.78      2608
```

In [206…

```
y_scores = my_pipeline.predict_proba(X_test)
print(y_scores)
```

```
[[7.94416390e-01 2.05583610e-01]
 [5.19895469e-01 4.80104531e-01]
 [2.27865288e-01 7.72134712e-01]
 ...
```

```
[9.99999072e-01 9.27734671e-07]
[1.22782561e-01 8.77217439e-01]
[2.82050515e-01 7.17949485e-01]]
```

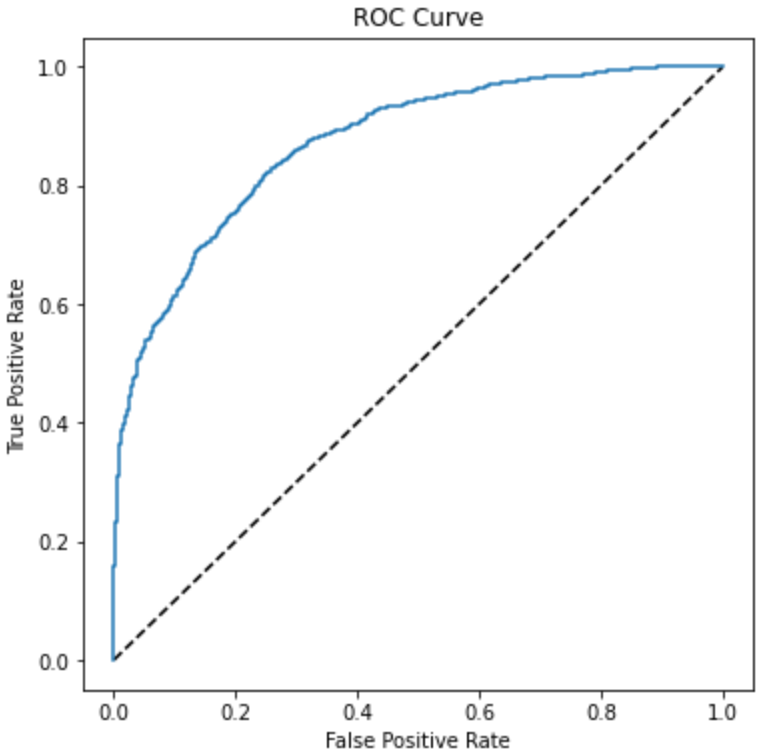
In [207…

```
auc = roc_auc_score(y_test,y_scores[:,1])
print(auc)
```

```
0.8724960901451662
```

In [208…

```
fpr, tpr, thresholds = roc_curve(y_test, y_scores[:,1])

# plot ROC curve
fig = plt.figure(figsize=(6, 6))
# Plot the diagonal 50% line
plt.plot([0, 1], [0, 1], 'k--')
# Plot the FPR and TPR achieved by our model
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```



After completed the process add new columns, we can a bit increase the model score, next time we will discrete the continue columns to imporve the model performance

first discrete columns age and num

In [209…

```
labels= ['A'+str(i) for i in range(0,10)]
labels
```

Out[209…

```
['A0', 'A1', 'A2', 'A3', 'A4', 'A5', 'A6', 'A7', 'A8', 'A9']
```

In [210…

```
X_train['Num']=X_train['Num'].astype('int')
X_test['Num']=X_test['Num'].astype('int')
test['Num']=test['Num'].astype('int')
```

In [211…
```
discrete, Interval= pd.qcut(X_train['Age'],q=10,
                            labels=labels,retbins=True,
                            precision=3,duplicates='raise')
Interval[0]=float('-inf')
Interval[len(Interval)-1]=float('inf')
Interval

X_train['Age'+'_discrete']=pd.cut(x=X_train['Age'],bins=Interval,labels=labels)


X_test['Age'+'_discrete']=pd.cut(x=X_test['Age'],bins=Interval,labels=labels)

test['Age'+'_discrete']=pd.cut(x=test['Age'],bins=Interval,labels=labels)



discrete, Interval= pd.qcut(X_train['Num'],q=10,
                            labels=labels,retbins=True,
                            precision=3,duplicates='raise')
Interval[0]=float('-inf')
Interval[len(Interval)-1]=float('inf')
Interval

X_train['Num'+'_discrete']=pd.cut(x=X_train['Num'],bins=Interval,labels=labels)


X_test['Num'+'_discrete']=pd.cut(x=X_test['Num'],bins=Interval,labels=labels)

test['Num'+'_discrete']=pd.cut(x=test['Num'],bins=Interval,labels=labels)
```

for the other continue values columns, also descrete as the columns that was find the more than 19 unique values

In [212…
```
from feature_engine.discretisation import EqualFrequencyDiscretiser
```

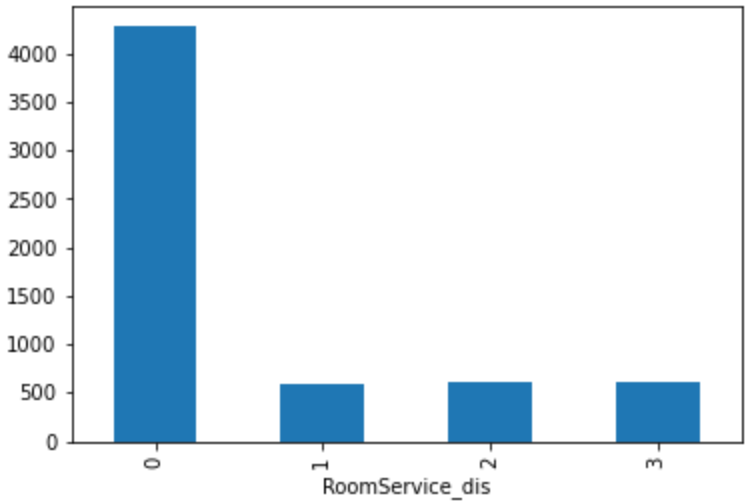
In [213…
```
col_con=[x for x in X_train.columns
        if X_train[x].dtypes in ['int64','float64'] and X_train[x].nunique()>= 20 and x!=
col_dis=[str(i)+'_dis' for i in col_con]
col_dis
```

Out[213…
```
['RoomService_dis',
 'FoodCourt_dis',
 'ShoppingMall_dis',
 'Spa_dis',
 'VRDeck_dis',
 'Avg_amenity_bill_dis',
 'total_amenity_bill_dis']
```

In [214…
```
disc = EqualFrequencyDiscretiser(q=10,variables=col_con)
disc.fit(X_train[col_con])
X_train[col_dis]=disc.transform(X_train[col_con])
X_test[col_dis]=disc.transform(X_test[col_con])
test[col_dis]=disc.transform(test[col_con])
```

In [215…
```
X_train.groupby('RoomService_dis')['RoomService'].count().plot(kind='bar')
```

Out[215…
```
<AxesSubplot:xlabel='RoomService_dis'>
```



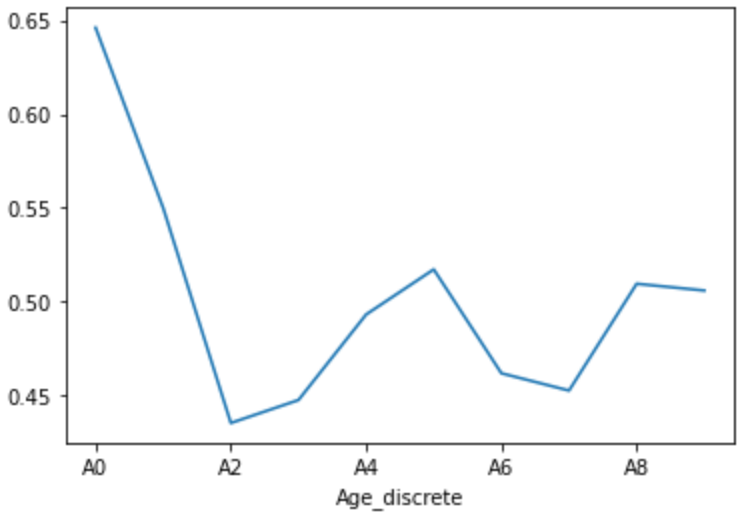
In [216…
```
disc.binner_dict_
```

Out[216…
```
{'RoomService': [-inf, 6.0, 173.0, 737.0, inf],
 'FoodCourt': [-inf, 13.0, 282.1999999999998, 1027.6000000000004, inf],
 'ShoppingMall': [-inf, 4.0, 84.0, 600.0, inf],
 'Spa': [-inf, 12.0, 173.0, 734.6000000000004, inf],
 'VRDeck': [-inf, 9.0, 162.19999999999982, 728.2000000000007, inf],
 'Avg_amenity_bill': [-inf,
  144.2,
  169.2,
  231.4,
  385.6799999999997,
  789.0,
  inf],
 'total_amenity_bill': [-inf,
  721.0,
  846.0,
  1157.0,
  1928.3999999999987,
  3945.0,
  inf]}
```

As the chart below showed the columns before/after descrete and encode that has the smooth line than before, so it might increase the model performance

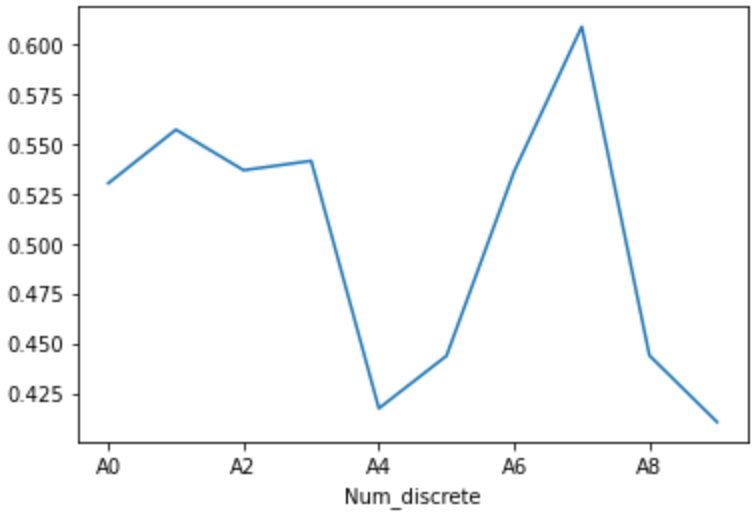
In [217…
```
data=pd.concat([X_train,y_train],axis=1)
data.groupby('Age_discrete')['Transported'].mean().plot()
```

Out[217…
```
<AxesSubplot:xlabel='Age_discrete'>
```



In [218…
```
data.groupby('Num_discrete')['Transported'].mean().plot()
```

Out[218…
```
<AxesSubplot:xlabel='Num_discrete'>
```



In [219…
```
from feature_engine.encoding import OrdinalEncoder
enc = OrdinalEncoder(encoding_method = 'ordered')
enc.fit(X_train,y_train)
```

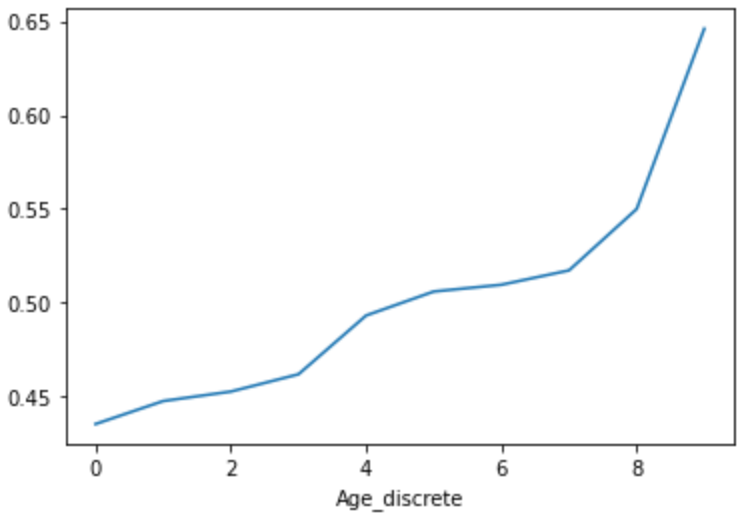
Out[219…
```
OrdinalEncoder()
```

In [220…
```
data_ec=enc.transform(X_train)
```

In [221…
```
data_ec=pd.concat([data_ec,y_train],axis=1)
```

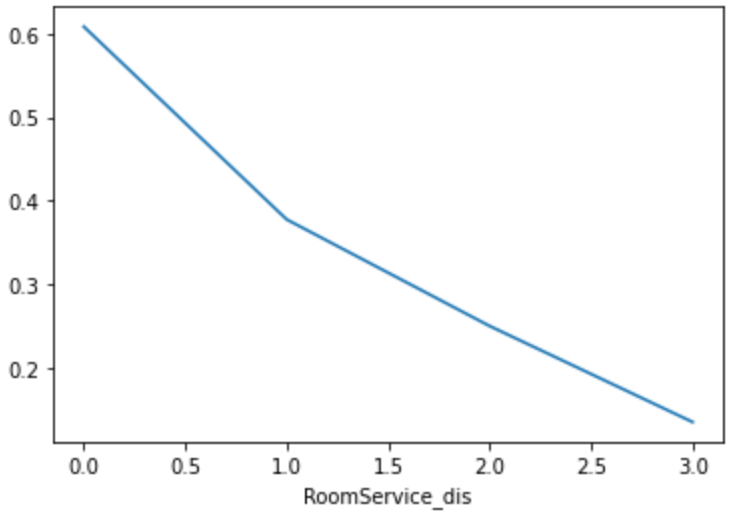
In [222…
```
data_ec.groupby('Age_discrete')['Transported'].mean().plot()
```

Out[222…
```
<AxesSubplot:xlabel='Age_discrete'>
```



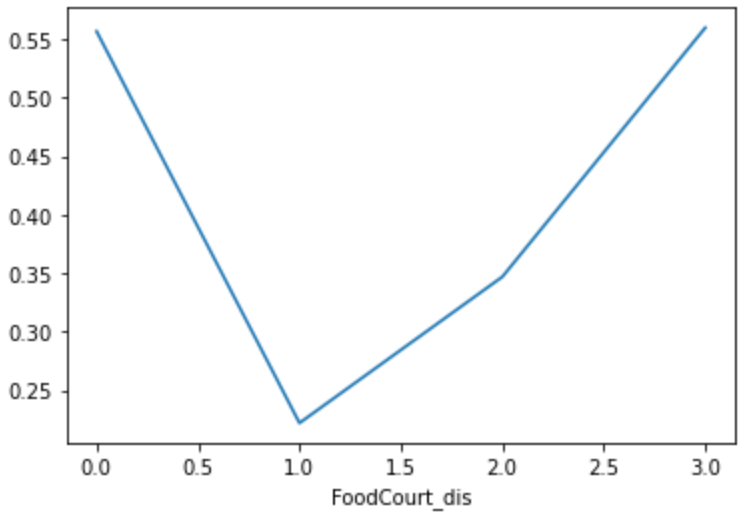
In [223…
```
data_ec.groupby('RoomService_dis')['Transported'].mean().plot()
```

Out[223…
```
<AxesSubplot:xlabel='RoomService_dis'>
```



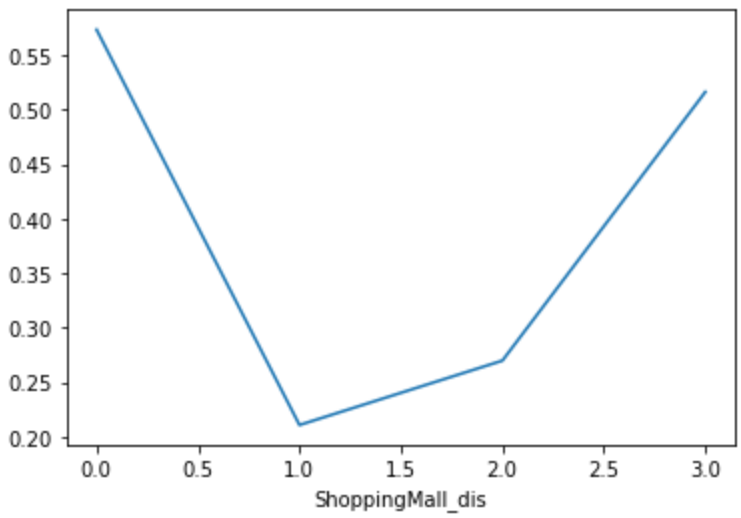
In [224…
```
data_ec.groupby('FoodCcourt_dis')['Transported'].mean().plot()
```

Out[224…
```
<AxesSubplot:xlabel='FoodCourt_dis'>
```



In [225…
```
data_ec.groupby('ShoppingMall_dis')['Transported'].mean().plot()
```

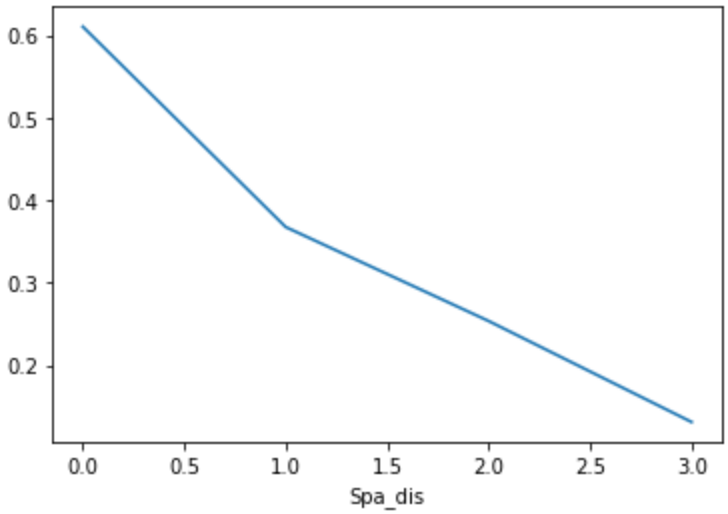
Out[225…
```
<AxesSubplot:xlabel='ShoppingMall_dis'>
```



In [226…
```
data_ec.groupby('Spa_dis')['Transported'].mean().plot()
```

```
<AxesSubplot:xlabel='Spa_dis'>
```

Out[226…

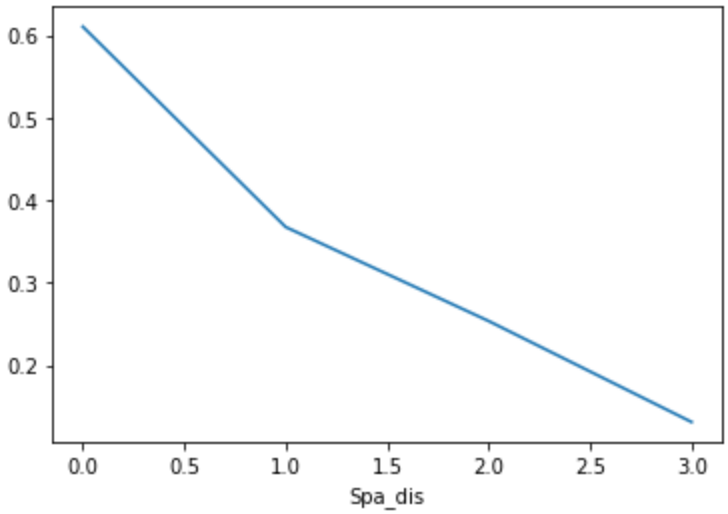


In [227…

```
data_ec.groupby('Spa_dis')['Transported'].mean().plot()
```

Out[227…

```
<AxesSubplot:xlabel='Spa_dis'>
```

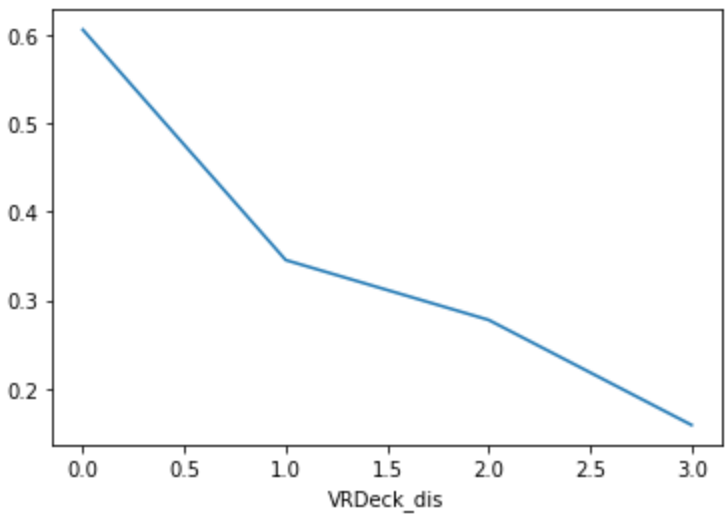


In [228…

```
data_ec.groupby('VRDeck_dis')['Transported'].mean().plot()
```

Out[228…

```
<AxesSubplot:xlabel='VRDeck_dis'>
```

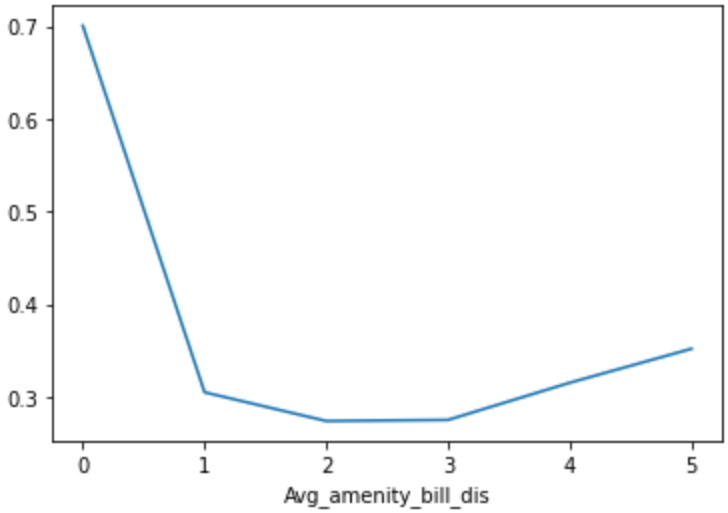


In [229…

```
data_ec.groupby('Avg_amenity_bill_dis')['Transported'].mean().plot()
```

Out[229…

```
<AxesSubplot:xlabel='Avg_amenity_bill_dis'>
```

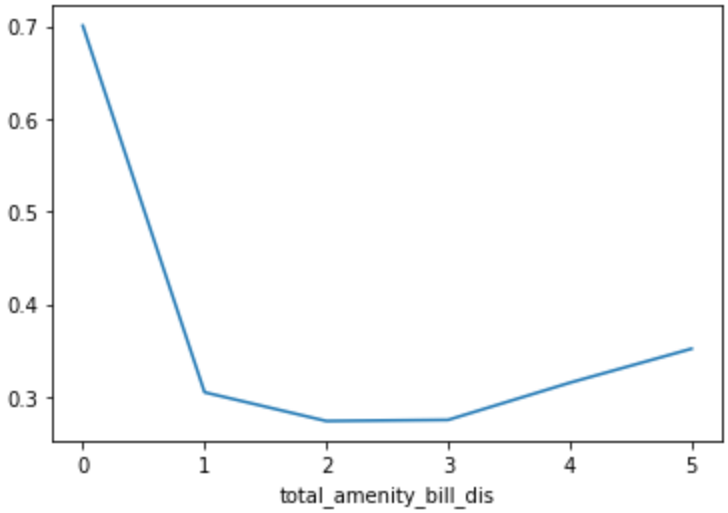


In [230…

```
data_ec.groupby('total_amenity_bill_dis')['Transported'].mean().plot()
```

Out[230…

```
<AxesSubplot:xlabel='total_amenity_bill_dis'>
```

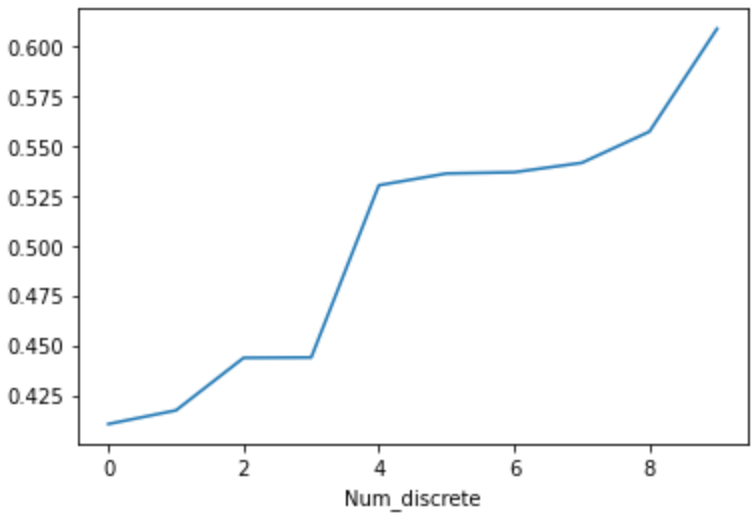


In [231…

```
data_ec.groupby('Num_discrete')['Transported'].mean().plot()
```

Out[231…

```
<AxesSubplot:xlabel='Num_discrete'>
```



Transform the descrete valune into the dataset

In [232…
```
enc.fit(X_train[['Age_discrete','Num_discrete']],y_train)
X_train[['Age_discrete','Num_discrete']]=enc.transform(X_train[['Age_discrete','Num_discre
X_test[['Age_discrete','Num_discrete']]=enc.transform(X_test[['Age_discrete','Num_discrete
test[['Age_discrete','Num_discrete']]=enc.transform(test[['Age_discrete','Num_discrete']])
```

In [233…
```
X_train.drop(['PassengerId','Id','Lname','Lname_null','Group'],axis=1,inplace=True)
X_test.drop(['PassengerId','Id','Lname','Lname_null','Group'],axis=1,inplace=True)
test.drop(['PassengerId','Id','Lname','Lname_null','Group'],axis=1,inplace=True)
```

built model for evaluate after descrete continue columns

In [234…
```
num=[x for x in X_train.columns
    if X_train[x].dtypes!='O' and x  != 'Transported']
cat=[c for c in X_train.columns
    if c not in num and c != 'Transported']
```

In [235…
```
numerical_transformer = Pipeline(steps=[

    ('scale', StandardScaler())
])

categorical_transformer = Pipeline(steps=[

    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num',numerical_transformer,num),

        ('cat', categorical_transformer, cat)
    ])
```

In [236…
```
model=LogisticRegression(C=0.5, solver='liblinear')
```

In [237…
```
my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('lr', model)
                             ])
```

In [238…
```
my_pipeline.fit(X_train,y_train)
```

Out[238…
```
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                  Pipeline(steps=[('scale',
                                                                   StandardScaler())]),
                                                  ['CryoSleep', 'Age', 'VIP',
                                                   'RoomService', 'FoodCourt',
                                                   'ShoppingMall', 'Spa',
                                                   'VRDeck', 'Num', 'Num_null',
                                                   'Side_null', 'Deck_null',
                                                   'Age_null', 'CryoSleep_null',
                                                   'Destination_null',
                                                   'HomePlanet_null',
                                                   'Avg_amenity_bill',
                                                   'total_amenity_bill',
```

```
                                                  'RoomService_use',
                                                  'FoodCourt_use',
                                                  'ShoppingMall_use',
                                                  'Spa_use', 'VRDeck_use',
                                                  'Tol_amenity_use',
                                                  'Group_no', 'Family_no',
                                                  'Travel_alone', 'Nofamily',
                                                  'Age_discrete',
                                                  'Num_discrete', ...]),
                                                 ('cat',
                                                  Pipeline(steps=[('onehot',
                                                                   OneHotEncoder(handle_un
known='ignore'))]),
                                                  ['HomePlanet', 'Destination',
                                                   'Deck', 'Side'])])),
                ('lr', LogisticRegression(C=0.5, solver='liblinear'))])
```

In [239…

```
pred=my_pipeline.predict(X_test)
metrics.accuracy_score(y_test,pred)
```

Out[239…

```
0.786042944785276
```

In [240…

```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.73      0.77      1289
           1       0.76      0.84      0.80      1319

    accuracy                           0.79      2608
   macro avg       0.79      0.79      0.79      2608
weighted avg       0.79      0.79      0.79      2608
```

In [241…

```
y_scores = my_pipeline.predict_proba(X_test)
print(y_scores)
```

```
[[9.17633315e-01 8.23666855e-02]
 [3.44360963e-01 6.55639037e-01]
 [2.65233608e-01 7.34766392e-01]
 ...
 [9.99963216e-01 3.67842327e-05]
 [2.23133045e-01 7.76866955e-01]
 [4.35373498e-01 5.64626502e-01]]
```

In [242…

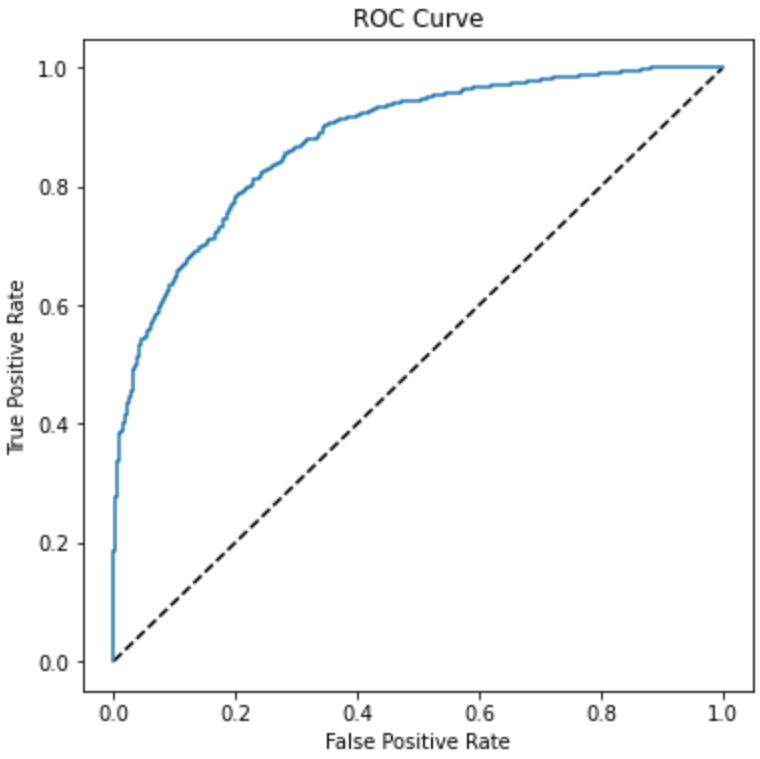
```
auc = roc_auc_score(y_test,y_scores[:,1])
print(auc)
```

```
0.8788242026925209
```

In [243…

```
fpr, tpr, thresholds = roc_curve(y_test, y_scores[:,1])

# plot ROC curve
fig = plt.figure(figsize=(6, 6))
# Plot the diagonal 50% line
plt.plot([0, 1], [0, 1], 'k--')
# Plot the FPR and TPR achieved by our model
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('ROC Curve')
plt.show()
```



add some dummie columns for one hot ecode category columns

In [244…
```
bool_col=[x for x in X_train.columns
          if X_train[x].dtypes == bool]
X_train[bool_col]=X_train[bool_col].astype('category')
X_test[bool_col]=X_test[bool_col].astype('category')
test[bool_col]=test[bool_col].astype('category')
```

In [245…
```
for i in bool_col:

    X_train[i]=X_train[i].cat.codes
    X_test[i]=X_test[i].cat.codes
    test[i]=test[i].cat.codes
```

In [246…
```
ob_col=[x for x in X_train.columns
          if X_train[x].dtypes == 'O']
X_train[ob_col]=X_train[ob_col].astype('category')
X_test[ob_col]=X_test[ob_col].astype('category')
test[ob_col]=test[ob_col].astype('category')
```

In [247…
```
X_train_dm=pd.get_dummies(X_train[['HomePlanet','Destination','Deck','Side']],drop_first=T
X_test_dm=pd.get_dummies(X_test[['HomePlanet','Destination','Deck','Side']],drop_first=Tru
test_dm=pd.get_dummies(test[['HomePlanet','Destination','Deck','Side']],drop_first=True)
```

In [248…
```
X_train=pd.concat([X_train,X_train_dm],axis=1)
X_test=pd.concat([X_test,X_test_dm],axis=1)
test=pd.concat([test,test_dm],axis=1)
```

In [249…

```
X_train.drop(['HomePlanet','Destination','Deck','Side'],axis=1,inplace=True)
X_test.drop(['HomePlanet','Destination','Deck','Side'],axis=1,inplace=True)
test.drop(['HomePlanet','Destination','Deck','Side'],axis=1,inplace=True)
```

In [250…

```
num=[x for x in train.columns
    if train[x].dtypes!='O' and x  != 'Transported']
cat=[c for c in train.columns
    if c not in num and c != 'Transported']
```

In [251…

```
X_train_original = X_train.copy()
X_test_original = X_test.copy()
test_original = test.copy()
```

Feature selection process

select the feature that without

- constant columns
- duplicate columns
- more correlation in independents variable

FInd the constant columns and filter out

In [252…

```
constant_features = [
    feat for feat in X_train.columns if X_train[feat].std() == 0
]

X_train.drop(labels=constant_features, axis=1, inplace=True)
X_test.drop(labels=constant_features, axis=1, inplace=True)
test.drop(labels=constant_features, axis=1, inplace=True)

X_train.shape, X_test.shape
```

Out[252… ((6085, 41), (2608, 41))

In [253…

```
sel = VarianceThreshold(
    threshold=0.01)

sel.fit(X_train)

sum(sel.get_support())
```

Out[253… 41

In [254…

```
features_to_keep = X_train.columns[sel.get_support()]
```

In [255…

```
X_train = sel.transform(X_train)
X_test = sel.transform(X_test)
test = sel.transform(test)



X_train.shape, X_test.shape
```

Out[255… ((6085, 41), (2608, 41))

In [256…
```
X_train= pd.DataFrame(X_train)
X_train.columns = features_to_keep

X_test= pd.DataFrame(X_test)
X_test.columns = features_to_keep

test= pd.DataFrame(test)
test.columns = features_to_keep
```

Find the duplicate columns and filter out off dataset

In [257…
```
duplicated_feat = []

for i in range(0, len(X_train.columns)):
    if i % 10 == 0:
        print(i)

    col_1 = X_train.columns[i]

    for col_2 in X_train.columns[i + 1:]:
        if X_train[col_1].equals(X_train[col_2]):
            duplicated_feat.append(col_2)

len(duplicated_feat)
```

```
0
10
20
30
40
```
Out[257… 1

In [258…
```
X_train.drop(labels=duplicated_feat, axis=1, inplace=True)
X_test.drop(labels=duplicated_feat, axis=1, inplace=True)
test.drop(labels=duplicated_feat, axis=1, inplace=True)


X_train.shape, X_test.shape
```

Out[258… ((6085, 40), (2608, 40))

In [259…
```
X_train_basic_filter = X_train.copy()
X_test_basic_filter = X_test.copy()
test_basic_filter = test.copy()
```

find the correlation columns and filter out

In [260…
```
def correlation(dataset, threshold):
    col_corr = set()
    corr_matrix = dataset.corr()
    for i in range(len(corr_matrix.columns)):
        for j in range(i):

            if abs(corr_matrix.iloc[i, j]) > threshold:
                colname = corr_matrix.columns[i]
```

```
                col_corr.add(colname)
    return col_corr


corr_features = correlation(X_train, 0.8)
print('correlated features: ', len(set(corr_features)))
```

```
correlated features:  5
```

In [261…

```
X_train.drop(labels=corr_features, axis=1, inplace=True)
X_test.drop(labels=corr_features, axis=1, inplace=True)
test.drop(labels=corr_features, axis=1, inplace=True)

X_train.shape, X_test.shape
```

Out[261…

```
((6085, 35), (2608, 35))
```

In [262…

```
X_train_corr = X_train.copy()
X_test_corr = X_test.copy()
test_corr = test.copy()
```

In [263…

```
scaler = StandardScaler()
scaler.fit(X_train)
```

Out[263…

```
StandardScaler()
```

Select the best feature for model by using lasso method

In [264…

```
sel_ = SelectFromModel(
    LogisticRegression(C=0.5,
                       penalty='l1',
                       solver='liblinear',
                       random_state=10))

sel_.fit(scaler.transform(X_train), y_train)



X_train_lasso = pd.DataFrame(sel_.transform(X_train))
X_test_lasso = pd.DataFrame(sel_.transform(X_test))
test_lasso = pd.DataFrame(sel_.transform(test))


X_train_lasso.columns = X_train.columns[(sel_.get_support())]
X_test_lasso.columns = X_train.columns[(sel_.get_support())]
test_lasso.columns = X_train.columns[(sel_.get_support())]
```

In [265…

```
X_train_lasso.shape, X_test_lasso.shape
```

Out[265…

```
((6085, 32), (2608, 32))
```

In [266…

```
def run_logistic(X_train, X_test, y_train, y_test):


    scaler = StandardScaler().fit(X_train)

    logit = LogisticRegression(random_state=44, max_iter=500)
```

```
    logit.fit(scaler.transform(X_train), y_train)

    print('Train set')
    pred = logit.predict_proba(scaler.transform(X_train))
    print('Logistic Regression roc-auc: {}'.format(
        roc_auc_score(y_train, pred[:, 1])))

    print('Test set')
    pred = logit.predict_proba(scaler.transform(X_test))
    print('Logistic Regression roc-auc: {}'.format(
        roc_auc_score(y_test, pred[:, 1])))
```

Compare accuracy by each feature selection method

In [267…

```
print('Original train shape is {} rows {} columns'.format(X_train_original.shape[0],X_trai
print('Original test shape is {} rows {} columns'.format(X_test_original.shape[0],X_test_c
run_logistic(X_train_original,
             X_test_original,
             y_train,
             y_test)
```

```
Original train shape is 6085 rows 48 columns
Original test shape is 2608 rows 48 columns
Train set
Logistic Regression roc-auc: 0.8894646743586746
Test set
Logistic Regression roc-auc: 0.8788230263540979
```

In [268…

```
print('Basic filter train shape is {} rows {} columns'.format(X_train_basic_filter.shape[0
print('Basicfilter test shape is {} rows {} columns'.format(X_test_basic_filter.shape[0],X
run_logistic(X_train_basic_filter,
             X_test_basic_filter,
             y_train,
             y_test)
```

```
Basic filter train shape is 6085 rows 40 columns
Basicfilter test shape is 2608 rows 40 columns
Train set
Logistic Regression roc-auc: 0.8894648904222683
Test set
Logistic Regression roc-auc: 0.8788171446619821
```

In [269…

```
print('Correlation filter train shape is {} rows {} columns'.format(X_train_corr.shape[0],
print('Correlation filter test shape is {} rows {} columns'.format(X_test_corr.shape[0],X_

run_logistic(X_train_corr,
             X_test_corr,
             y_train,
             y_test)
```

```
Correlation filter train shape is 6085 rows 35 columns
Correlation filter test shape is 2608 rows 35 columns
Train set
Logistic Regression roc-auc: 0.8875990732600344
Test set
Logistic Regression roc-auc: 0.8808527983032495
```

In [270…

```
print('Lasso filter train shape is {} rows {} columns'.format(X_train_lasso.shape[0],X_tra
print('Lasso filter test shape is {} rows {} columns'.format(X_test_lasso.shape[0],X_test_
run_logistic(X_train_lasso,
             X_test_lasso,
```

```
                y_train,
                y_test)
```

```
Lasso filter train shape is 6085 rows 32 columns
Lasso filter test shape is 2608 rows 32 columns
Train set
Logistic Regression roc-auc: 0.8875950760835535
Test set
Logistic Regression roc-auc: 0.8808480929495569
```

In [271…

```
sel_ = SelectFromModel(RandomForestClassifier(n_estimators=50, random_state=10))
sel_.fit(X_train_corr, y_train)



X_train_rf = pd.DataFrame(sel_.transform(X_train_corr))
X_test_rf = pd.DataFrame(sel_.transform(X_test_corr))
test_rf = pd.DataFrame(sel_.transform(test_corr))

X_train_rf.columns = X_train_corr.columns[(sel_.get_support())]
X_test_rf.columns = X_test_corr.columns[(sel_.get_support())]
test_rf.columns = test_corr.columns[(sel_.get_support())]
```

In [272…

```
def run_randomForests(X_train, X_test, y_train, y_test):

    rf = RandomForestClassifier(n_estimators=200, random_state=39, max_depth=4)
    rf.fit(X_train, y_train)

    print('Train set')
    pred = rf.predict_proba(X_train)
    print('Random Forests roc-auc: {}'.format(roc_auc_score(y_train, pred[:,1])))

    print('Test set')
    pred = rf.predict_proba(X_test)
    print('Random Forests roc-auc: {}'.format(roc_auc_score(y_test, pred[:,1])))
```

In [273…

```
print('Randomforest filter train shape is {} rows {} columns'.format(X_train_rf.shape[0],X
print('Randomforest filter test shape is {} rows {} columns'.format(X_test_rf.shape[0],X_t
run_randomForests(X_train_rf,
                  X_test_rf,
                  y_train, y_test)
```

```
Randomforest filter train shape is 6085 rows 13 columns
Randomforest filter test shape is 2608 rows 13 columns
Train set
Random Forests roc-auc: 0.854474201682833
Test set
Random Forests roc-auc: 0.8380337856158514
```

After compare score in each method notice that the best score for test model is using lasso method to select feature

In [274…

```
scaler = StandardScaler()
scaler.fit(X_train_lasso)
```

Out[274…

```
StandardScaler()
```

In [275…

```
X_train_lasso_scale=scaler.transform(X_train_lasso)
X_test_lasso_scale=scaler.transform(X_test_lasso)
```

In [276…
```
model=LogisticRegression(C=0.5, solver='liblinear')
```

In [277…
```
model.fit(X_train_lasso,y_train)
```

Out[277…
```
LogisticRegression(C=0.5, solver='liblinear')
```

In [278…
```
pred=model.predict(X_test_lasso)
metrics.accuracy_score(y_test,pred)
```

Out[278…
```
0.7871932515337423
```

In [279…
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.74      0.77      1289
           1       0.76      0.84      0.80      1319

    accuracy                           0.79      2608
   macro avg       0.79      0.79      0.79      2608
weighted avg       0.79      0.79      0.79      2608
```

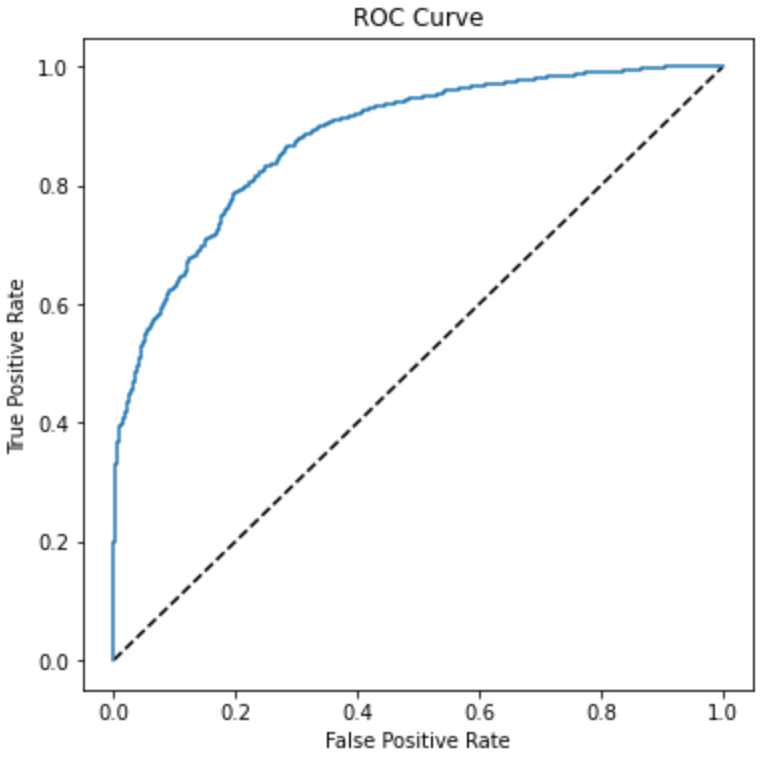
In [280…
```
y_scores = model.predict_proba(X_test_lasso)
print(y_scores)
```

```
[[8.85740526e-01 1.14259474e-01]
 [5.00195933e-01 4.99804067e-01]
 [2.67113873e-01 7.32886127e-01]
 ...
 [9.99977010e-01 2.29899578e-05]
 [2.32652966e-01 7.67347034e-01]
 [4.63357275e-01 5.36642725e-01]]
```

In [281…
```
auc = roc_auc_score(y_test,y_scores[:,1])
print(auc)
```

```
0.8796699900187684
```

In [282…
```
fpr, tpr, thresholds = roc_curve(y_test, y_scores[:,1])

# plot ROC curve
fig = plt.figure(figsize=(6, 6))
# Plot the diagonal 50% line
plt.plot([0, 1], [0, 1], 'k--')
# Plot the FPR and TPR achieved by our model
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```



Try to use Robustscale to use with outliner data in dataset to improve performance

In [283…
```
robust = RobustScaler()
robust.fit(X_train_lasso)
```

Out[283…
```
RobustScaler()
```

In [284…
```
X_train_lasso_rb=robust.transform(X_train_lasso)
X_test_lasso_rb=robust.transform(X_test_lasso)
```

In [285…
```
model=LogisticRegression(C=0.5, solver='liblinear')
```

In [286…
```
model.fit(X_train_lasso_rb,y_train)
```

Out[286…
```
LogisticRegression(C=0.5, solver='liblinear')
```

In [287…
```
pred=model.predict(X_test_lasso_rb)
metrics.accuracy_score(y_test,pred)
```

Out[287…
```
0.7864263803680982
```

In [288…
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.81      0.74      0.77      1289
           1       0.76      0.84      0.80      1319

    accuracy                           0.79      2608
   macro avg       0.79      0.79      0.79      2608
weighted avg       0.79      0.79      0.79      2608
```

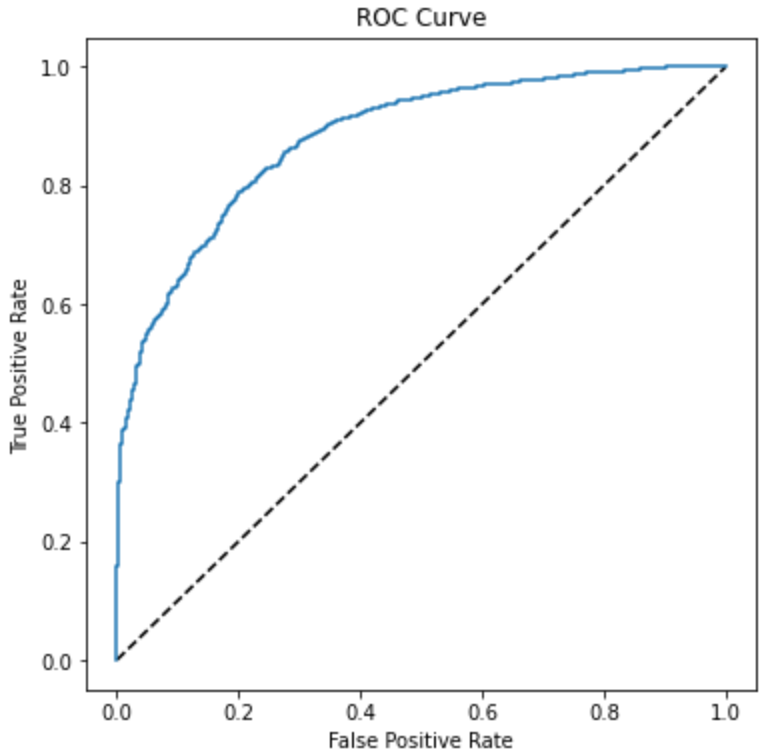
In [289…
```
y_scores = model.predict_proba(X_test_lasso_rb)
print(y_scores)
```

```
[[8.86982583e-01 1.13017417e-01]
 [4.89133658e-01 5.10866342e-01]
 [2.65751643e-01 7.34248357e-01]
 ...
 [9.99977744e-01 2.22564632e-05]
 [2.42154524e-01 7.57845476e-01]
 [4.60876629e-01 5.39123371e-01]]
```

In [290…
```
auc = roc_auc_score(y_test,y_scores[:,1])
print(auc)
```

```
0.8808292715347863
```

In [291…
```
fpr, tpr, thresholds = roc_curve(y_test, y_scores[:,1])

# plot ROC curve
fig = plt.figure(figsize=(6, 6))
# Plot the diagonal 50% line
plt.plot([0, 1], [0, 1], 'k--')
# Plot the FPR and TPR achieved by our model
plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```



Try another method of feature selection by filter out the feature with top twenty mutual information score

In [292…
```
mi = mutual_info_classif(X_train_original, y_train)
mi
```

Out[292…
```
array([0.11273595, 0.01613687, 0.0008033 , 0.06115884, 0.04110562,
       0.04096195, 0.07693499, 0.05909672, 0.008674  , 0.        ,
       0.00292106, 0.00822654, 0.        , 0.00808601, 0.        ,
       0.00489482, 0.14212358, 0.14100974, 0.06047344, 0.0270929 ,
```

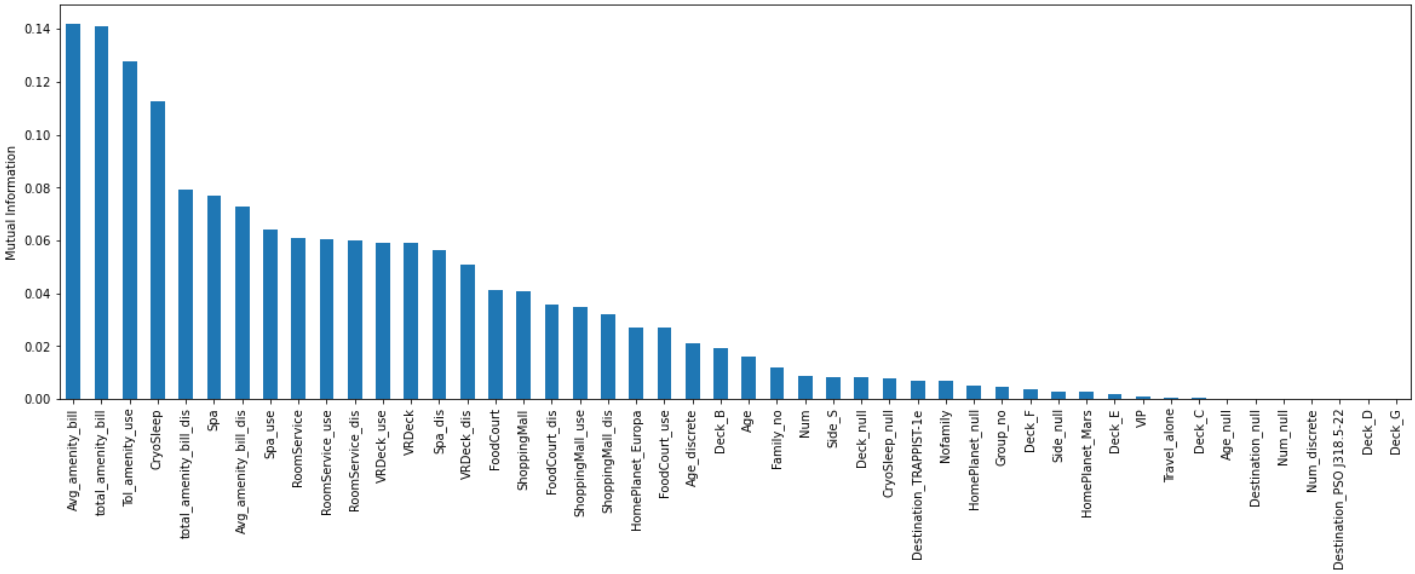
```
       0.03500445, 0.06425629, 0.05928002, 0.12762305, 0.0047867 ,
       0.01203065, 0.00058837, 0.00711027, 0.02101067, 0.        ,
       0.06003794, 0.03556696, 0.03208875, 0.05646061, 0.05085286,
       0.07284768, 0.07913301, 0.02718867, 0.00282516, 0.        ,
       0.00713484, 0.01906922, 0.00050501, 0.        , 0.00203508,
       0.00395238, 0.        , 0.00846092])
```

In [293…

```
mi = pd.Series(mi)
mi.index = X_train_original.columns
mi.sort_values(ascending=False).plot.bar(figsize=(20, 6))
plt.ylabel('Mutual Information')
```

Out[293…

```
Text(0, 0.5, 'Mutual Information')
```



In [294…

```
sel_ = SelectKBest(mutual_info_classif, k=20).fit(X_train_original, y_train)

X_train_original.columns[sel_.get_support()]
```

Out[294…

```
Index(['CryoSleep', 'RoomService', 'FoodCourt', 'ShoppingMall', 'Spa',
       'VRDeck', 'Avg_amenity_bill', 'total_amenity_bill', 'RoomService_use',
       'FoodCourt_use', 'Spa_use', 'VRDeck_use', 'Tol_amenity_use',
       'RoomService_dis', 'FoodCourt_dis', 'ShoppingMall_dis', 'Spa_dis',
       'VRDeck_dis', 'Avg_amenity_bill_dis', 'total_amenity_bill_dis'],
      dtype='object')
```

In [295…

```
X_train_mi = sel_.transform(X_train_original)
X_test_mi = sel_.transform(X_test_original)
```

In [296…

```
numerical_transformer = Pipeline(steps=[

    ('robust', RobustScaler())
])

categorical_transformer = Pipeline(steps=[

    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num',numerical_transformer,num),
```

```
        ('cat', categorical_transformer, cat)
    ])
```

In [297…
```
model=LogisticRegression(C=0.5, solver='liblinear')
```

In [298…
```
my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('lr', model)
                             ])
```

In [299…
```
model.fit(X_train_mi,y_train)
```

Out[299…
```
LogisticRegression(C=0.5, solver='liblinear')
```

In [300…
```
pred=model.predict(X_test_mi)
metrics.accuracy_score(y_test,pred)
```

Out[300…
```
0.785659509202454
```

In [301…
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.83      0.71      0.77      1289
           1       0.75      0.86      0.80      1319

    accuracy                           0.79      2608
   macro avg       0.79      0.78      0.78      2608
weighted avg       0.79      0.79      0.78      2608
```

In [302…
```
y_scores = model.predict_proba(X_test_mi)
print(y_scores)
```

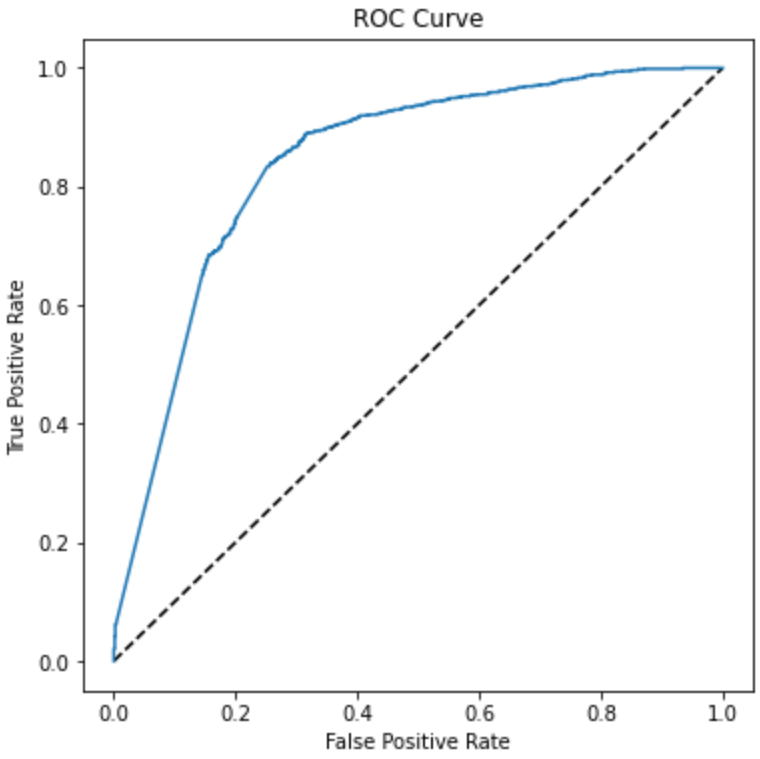
```
[[8.93595271e-01 1.06404729e-01]
 [2.72980563e-01 7.27019437e-01]
 [1.79950743e-01 8.20049257e-01]
 ...
 [9.99966881e-01 3.31192984e-05]
 [4.73791058e-01 5.26208942e-01]
 [4.34311376e-01 5.65688624e-01]]
```

In [303…
```
auc = roc_auc_score(y_test,y_scores[:,1])
print(auc)
```

```
0.8418848235286507
```

In [304…
```
fpr, tpr, thresholds = roc_curve(y_test, y_scores[:,1])


fig = plt.figure(figsize=(6, 6))

plt.plot([0, 1], [0, 1], 'k--')

plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
```

```
plt.title('ROC Curve')
plt.show()
```



Try the RFE method to find the best feature

In [305…

```
rfe_gb = RFE(estimator=GradientBoostingClassifier(),
             n_features_to_select=19, step=3, verbose=1)
rfe_gb.fit(X_train, y_train)


score = rfe_gb.score(X_test, y_test)
print('The model can explain {0:.1%} of the variance in the test set'.format(score))


gb_mask = rfe_gb.support_!=0
```

```
Fitting estimator with 35 features.
Fitting estimator with 32 features.
Fitting estimator with 29 features.
Fitting estimator with 26 features.
Fitting estimator with 23 features.
Fitting estimator with 20 features.
The model can explain 79.9% of the variance in the test set
```

In [306…

```
rfe_rf = RFE(estimator=RandomForestClassifier(),
             n_features_to_select=29,step=3,verbose=1)

rfe_rf.fit(X_train, y_train)
r_squared = rfe_rf.score(X_test, y_test)
print('The model can explain {0:.1%} of the variance in the test set'.format(r_squared))


rf_mask = rfe_rf.support_
```

```
Fitting estimator with 35 features.
Fitting estimator with 32 features.
The model can explain 79.5% of the variance in the test set
```

Notice that the RFE gb model can best explain the test set

In [307…

```
X_train_gb=X_train.loc[:,gb_mask]
X_test_gb=X_test.loc[:,gb_mask]
test=test.loc[:,gb_mask]
```

In [308…

```
X_train_rf=X_train.loc[:,rf_mask]
X_test_rf=X_test.loc[:,rf_mask]
```

In [309…

```
model_pipeline=[]
model_pipeline.append(RandomForestClassifier())
model_pipeline.append(LogisticRegression())
model_pipeline.append(SVC())
model_pipeline.append(GradientBoostingClassifier())
model_pipeline.append(GaussianNB())
model_pipeline.append(KNeighborsClassifier())
```

In [310…

```
model_list=('RandomForestClassifier','LogisticRegression','SVC','GradientBoostingClassifie
score_list=[]

for model in model_pipeline:
    model.fit(X_train_gb,y_train)
    pred=model.predict(X_test_gb)

    score_list.append(metrics.accuracy_score(y_test,pred))
```

find the best model performance by compare RFE from gb and RFE from randomforest

In [311…

```
result_model=pd.DataFrame({'Model':model_list,'Score':score_list})
result_model
```

Out[311…

	Model	Score
0	RandomForestClassifier	0.800230
1	LogisticRegression	0.790644
2	SVC	0.777607
3	GradientBoostingClassifier	0.798696
4	GaussianNB	0.746933
5	KNeighborsClassifier	0.757669

In [312…

```
model_list=('RandomForestClassifier','LogisticRegression','SVC','GradientBoostingClassifie
score_list=[]
auc_list=[]
for model in model_pipeline:
    model.fit(X_train_rf,y_train)
    pred=model.predict(X_test_rf)

    score_list.append(metrics.accuracy_score(y_test,pred))
```

In [313…

```
#result_model=pd.DataFrame({'Model':model_list,'Score':score_list})
```

```
#result_model
```

In [314…

```
X_train_gb
```

Out[314…

	CryoSleep	Age	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Num	Avg_amenity_bill	To
0	0.0	43.0	0.0	1440.0	0.0	85.0	150.0	120.0	335.0	
1	1.0	23.0	0.0	0.0	0.0	0.0	0.0	273.0	0.0	
2	0.0	46.0	8.0	652.0	0.0	5.0	90.0	300.0	151.0	
3	0.0	33.0	0.0	763.0	8.0	2.0	30.0	346.0	160.6	
4	0.0	24.0	0.0	58.0	618.0	0.0	41.0	1334.0	143.4	
…	…	…	…	…	…	…	…	…	…	
6080	0.0	18.0	14.0	2.0	144.0	610.0	0.0	988.0	154.0	
6081	0.0	50.0	690.0	0.0	30.0	762.0	428.0	1063.0	382.0	
6082	0.0	22.0	158.0	0.0	476.0	0.0	26.0	1194.0	132.0	
6083	0.0	34.0	379.0	0.0	1626.0	0.0	0.0	191.0	401.0	
6084	0.0	28.0	7.0	489.0	0.0	4.0	6027.0	253.0	1305.4	

6085 rows × 19 columns

Hyperparameter tuning

In [315…

```
from sklearn.preprocessing import PowerTransformer
```

In [316…

```
pipeline= Pipeline([

    ('Robust',
    RobustScaler()),
    ('yeojohnson',PowerTransformer(method='yeo-johnson', standardize=False)),

    ('gb',
    GradientBoostingClassifier()),
])
```

Set the hyperparameter and using the random grid to find the best tuning by random hyperparameter?

In [317…

```
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import loguniform
from sklearn.ensemble import GradientBoostingRegressor

param={
    "gb__n_estimators":[1,5,10,20,50,200,500,700],
    "gb__max_leaf_nodes":[2,5,10,20,50,100],
    "gb__learning_rate":loguniform(0.01,1),
    "gb__max_depth":[1,2,5,7,9]
    }
random_cv=RandomizedSearchCV(
    estimator=pipeline,param_distributions=param,
    scoring='roc_auc',n_iter=20,
    random_state=42,n_jobs=-1,cv=5,verbose=2
```

```
)
random_cv.fit(X_train_gb,y_train)
```

Out[317…
```
Fitting 5 folds for each of 20 candidates, totalling 100 fits
RandomizedSearchCV(cv=5,
                   estimator=Pipeline(steps=[('Robust', RobustScaler()),
                                             ('yeojohnson',
                                              PowerTransformer(standardize=False)),
                                             ('gb',
                                              GradientBoostingClassifier())]),
                   n_iter=20, n_jobs=-1,
                   param_distributions={'gb__learning_rate': <scipy.stats._distn_infrastru
cture.rv_frozen object at 0x7f8dd24f0520>,
                                        'gb__max_depth': [1, 2, 5, 7, 9],
                                        'gb__max_leaf_nodes': [2, 5, 10, 20, 50,
                                                               100],
                                        'gb__n_estimators': [1, 5, 10, 20, 50,
                                                             200, 500, 700]},
                   random_state=42, scoring='roc_auc', verbose=2)
```

In [318…
```
random_cv.best_params_
```

Out[318…
```
{'gb__learning_rate': 0.013066739238053278,
 'gb__max_depth': 9,
 'gb__max_leaf_nodes': 20,
 'gb__n_estimators': 700}
```

In [319…
```
model_gb=random_cv.best_estimator_
model_gb.fit(X_train_gb,y_train)
pred=model_gb.predict(X_test_gb)
metrics.accuracy_score(y_test,pred)
```

Out[319…
```
0.8071319018404908
```

After got the best performance from the random grid, we will scope small number of hyperparameter using Grid search

In [320…
```
from sklearn.model_selection import GridSearchCV
```

In [321…
```
pipeline= Pipeline([

    ('standard',
    StandardScaler()),
    ('yeojohnson',PowerTransformer(method='yeo-johnson', standardize=False)),

    ('gb',
    GradientBoostingClassifier(learning_rate=0.013066739238053278)),
])
```

In [322…
```
from scipy.stats import loguniform
from sklearn.ensemble import GradientBoostingRegressor

param={
    "gb__n_estimators":[720],
    "gb__max_leaf_nodes":[20],
    "gb__max_depth":[10]
    }
grid_cv=GridSearchCV(
    estimator=pipeline,param_grid=param,
    scoring='roc_auc',
```

```
    n_jobs=-1,cv=5,verbose=2
)
grid_cv.fit(X_train_gb,y_train)
```

Out[322…
```
Fitting 5 folds for each of 1 candidates, totalling 5 fits
GridSearchCV(cv=5,
             estimator=Pipeline(steps=[('standard', StandardScaler()),
                                       ('yeojohnson',
                                        PowerTransformer(standardize=False)),
                                       ('gb',
                                        GradientBoostingClassifier(learning_rate=0.0130667
39238053278))]),
             n_jobs=-1,
             param_grid={'gb__max_depth': [10], 'gb__max_leaf_nodes': [20],
                         'gb__n_estimators': [720]},
             scoring='roc_auc', verbose=2)
```

In [323…
```
grid_cv.best_params_
```

Out[323…
```
{'gb__max_depth': 10, 'gb__max_leaf_nodes': 20, 'gb__n_estimators': 720}
```

In [324…
```
model_gb=grid_cv.best_estimator_
model_gb.fit(X_train_gb,y_train)
pred=model_gb.predict(X_test_gb)
metrics.accuracy_score(y_test,pred)
```

Out[324…
```
0.8090490797546013
```

In [325…
```
print(classification_report(y_test,pred))
```

```
              precision    recall  f1-score   support

           0       0.82      0.79      0.80      1289
           1       0.80      0.83      0.81      1319

    accuracy                           0.81      2608
   macro avg       0.81      0.81      0.81      2608
weighted avg       0.81      0.81      0.81      2608
```

In [326…
```
y_scores = model_gb.predict_proba(X_test_gb)
print(y_scores)
```

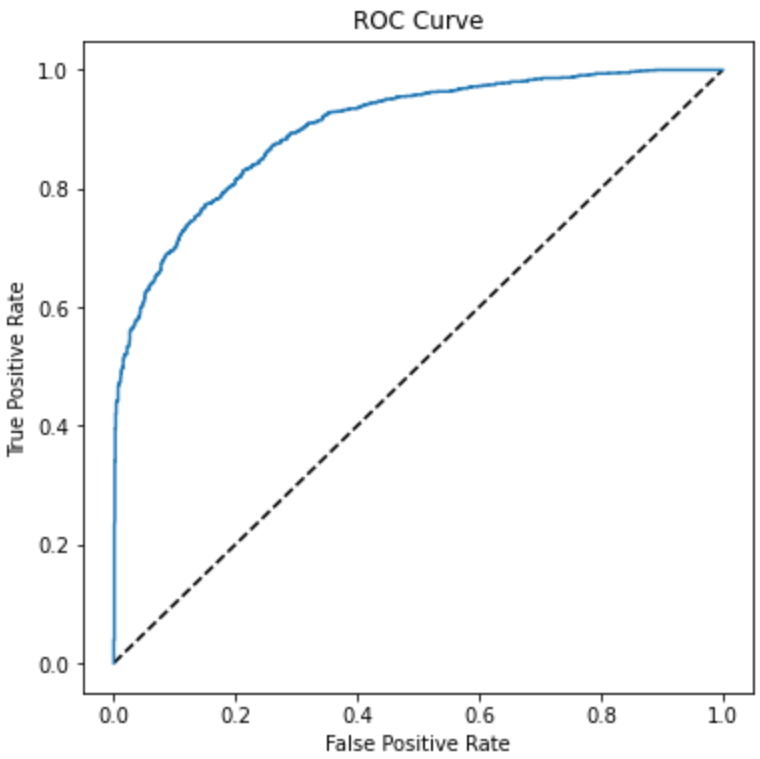
```
[[0.88080423 0.11919577]
 [0.44706292 0.55293708]
 [0.29547367 0.70452633]
 ...
 [0.97888421 0.02111579]
 [0.30827924 0.69172076]
 [0.25212637 0.74787363]]
```

In [327…
```
auc = roc_auc_score(y_test,y_scores[:,1])
print(auc)
```

```
0.9013454958884031
```

In [328…
```
fpr, tpr, thresholds = roc_curve(y_test, y_scores[:,1])


fig = plt.figure(figsize=(6, 6))
```

```
plt.plot([0, 1], [0, 1], 'k--')

plt.plot(fpr, tpr)
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.show()
```



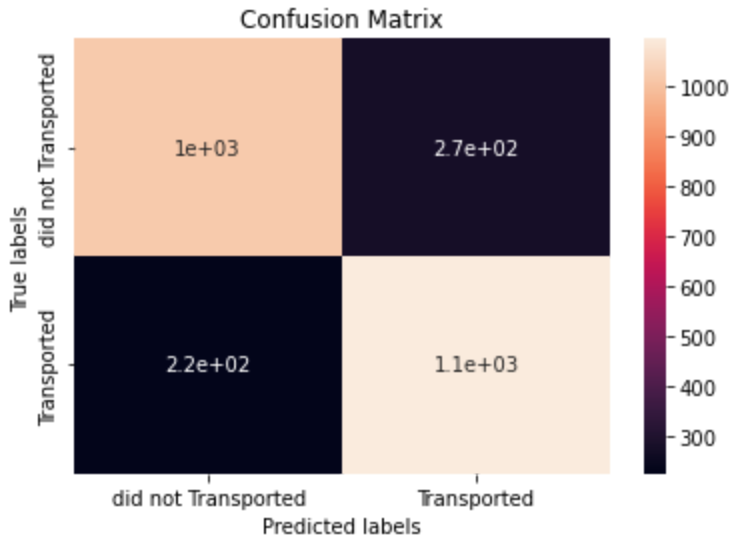
Confusion matrix

In [329…

```
def plot_confusion_matrix(y,y_predict):

    from sklearn.metrics import confusion_matrix

    cm = confusion_matrix(y, y_predict)
    ax= plt.subplot()
    sns.heatmap(cm, annot=cm, ax = ax);
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['did not Transported', 'Transported']); ax.yaxis.set_ticklabe
```

In [330…

```
plot_confusion_matrix(y_test,pred)
```



conclusion

mode show the accuracy around 81% for the final prediction model we try to predict the trasported of passengers, so focus on the 1

- precision = 80% is the moodel can predict the transported correct 80% from total transported predicted
- recall = 83% is the model can predict the actual transported 83% from total actual transported
- f1-score = 81% and as we predict the probability to calculate the AUC and plot the ROC curve we can got AUC socre = 90.1 %

In [331…

```
import joblib
file = './Transported.pkl'
joblib.dump(model_gb, file)
```

Out[331…

```
['./Transported.pkl']
```

In [332…

```
loaded_model = joblib.load(file)
predict_data=test

result = loaded_model.predict(predict_data)
result[0:100]
```

Out[332…

```
array([1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0,
       1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       0, 1, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0,
       1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0,
       1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1], dtype=int8)
```

In [333…

```
data=pd.DataFrame({'Predict':result})
a=data['Predict'].value_counts()
a
```

Out[333…

```
1    2240
0    2037
Name: Predict, dtype: int64
```

In [334…

```
tol=sum(a)
base_test=(a[1]/tol)*100
base_test
```

Out[334…

```
52.373158756137485
```