Import the library

In [1]:
```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

Import dataset

In [ ]:
```
```

In [2]:
```
df=pd.read_csv(r"C:\Users\Shubham\Desktop\Data Science\Test\Shubham Singh EDA\Spa
df.head()
```

Out[2]:

	PassengerId	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt
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

Dropping Insignificant Varibles

In [3]:
```
df=df.drop(["PassengerId","Name"],axis=1)
```

In [4]:
```
df.head()
```

Out[4]:

	HomePlanet	CryoSleep	Cabin	Destination	Age	VIP	RoomService	FoodCourt	ShoppingMall
0	Europa	False	B/0/P	TRAPPIST-1e	39.0	False	0.0	0.0	0.0
1	Earth	False	F/0/S	TRAPPIST-1e	24.0	False	109.0	9.0	25.0
2	Europa	False	A/0/S	TRAPPIST-1e	58.0	True	43.0	3576.0	0.0
3	Europa	False	A/0/S	TRAPPIST-1e	33.0	False	0.0	1283.0	371.0
4	Earth	False	F/1/S	TRAPPIST-1e	16.0	False	303.0	70.0	151.0

To find the information about the dataset

In [ ]:
```
```

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8693 entries, 0 to 8692
Data columns (total 12 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   HomePlanet    8492 non-null   object 
 1   CryoSleep     8476 non-null   object 
 2   Cabin         8494 non-null   object 
 3   Destination   8511 non-null   object 
 4   Age           8514 non-null   float64
 5   VIP           8490 non-null   object 
 6   RoomService   8512 non-null   float64
 7   FoodCourt     8510 non-null   float64
 8   ShoppingMall  8485 non-null   float64
 9   Spa           8510 non-null   float64
 10  VRDeck        8505 non-null   float64
 11  Transported   8693 non-null   bool   
dtypes: bool(1), float64(6), object(5)
memory usage: 755.7+ KB
```

In [6]:
```
df.describe()
```

Out[6]:

	Age	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck
count	8514.000000	8512.000000	8510.000000	8485.000000	8510.000000	8505.000000
mean	28.827930	224.687617	458.077203	173.729169	311.138778	304.854791
std	14.489021	666.717663	1611.489240	604.696458	1136.705535	1145.717189
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	19.000000	0.000000	0.000000	0.000000	0.000000	0.000000
50%	27.000000	0.000000	0.000000	0.000000	0.000000	0.000000
75%	38.000000	47.000000	76.000000	27.000000	59.000000	46.000000
max	79.000000	14327.000000	29813.000000	23492.000000	22408.000000	24133.000000

Data Preprocessing

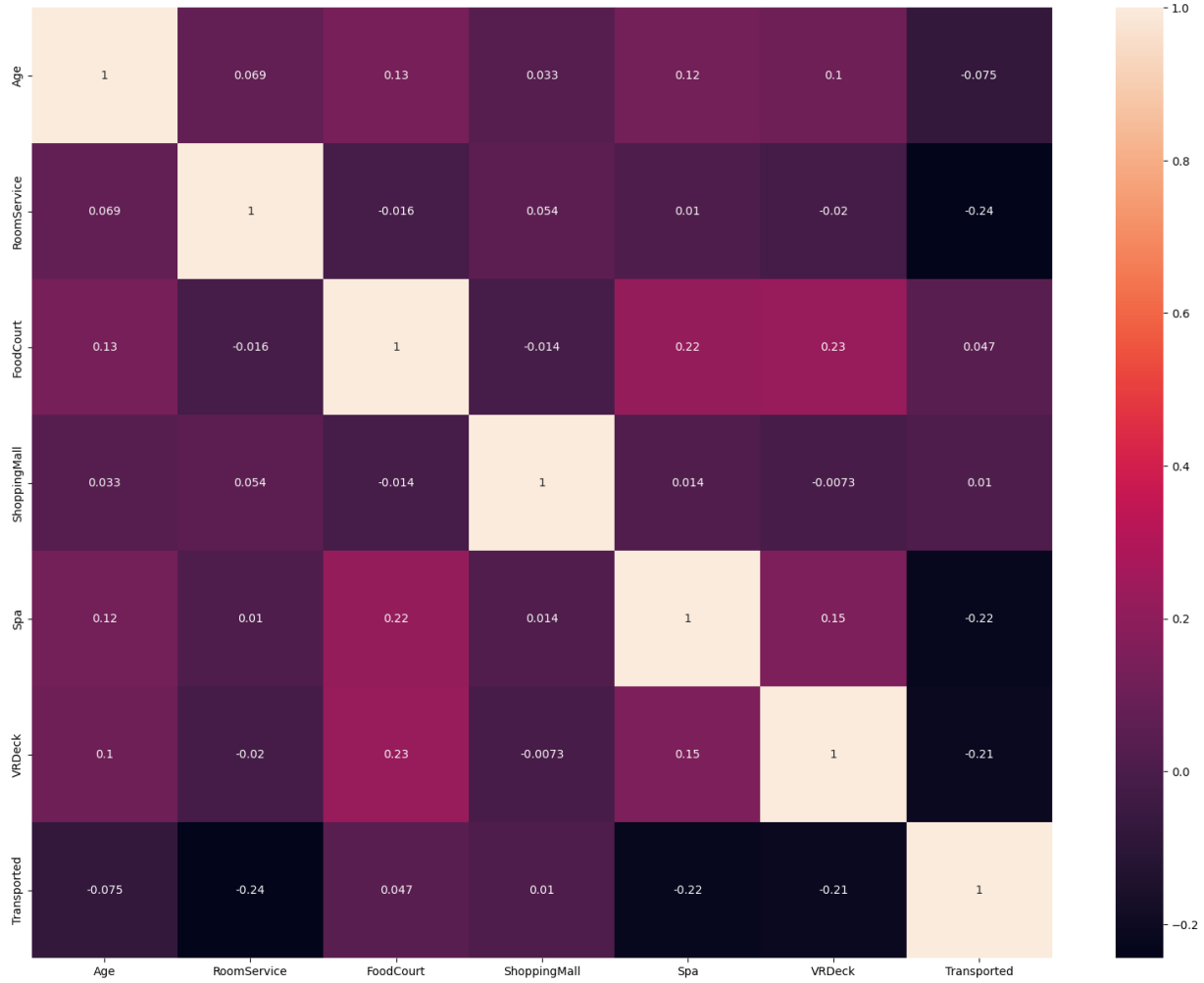
In [ ]:
```

```

In [7]:
```
df.isna().sum()/len(df)*100
```

Out[7]:
```
HomePlanet      2.312205
CryoSleep       2.496261
Cabin           2.289198
Destination     2.093639
Age             2.059128
VIP             2.335212
RoomService     2.082135
FoodCourt       2.105142
ShoppingMall    2.392730
Spa             2.105142
VRDeck          2.162660
Transported     0.000000
dtype: float64
```

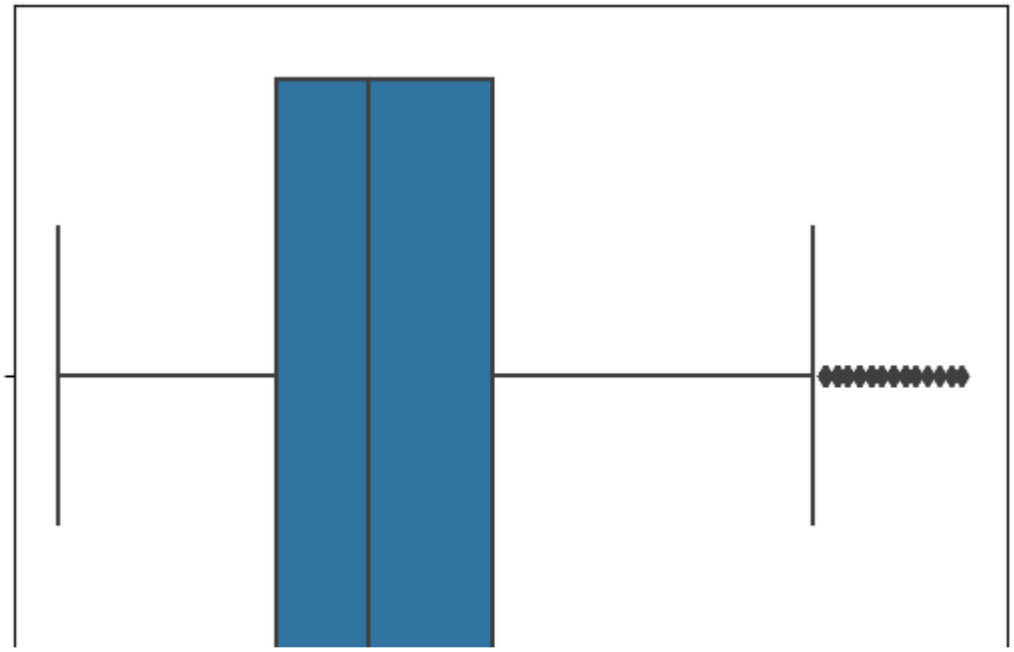
In [ ]:
```

```

In [8]:
```
plt.figure(figsize=(20,15))

sns.heatmap(df.corr(),annot=True)
plt.show()
```



In [ ]:
```
```

In [9]:
```
def boxplot(col):
    sns.boxplot(df[col])
    plt.show()

for i in list(df.select_dtypes(exclude=["object","bool"]).columns)[0:]:
    boxplot(i)
```



Treating Null Values

In [10]:
```
def apply_mode_and_fill_na(df):
    for column_name in df.select_dtypes(include='object').columns:

        mode_value = df[column_name].mode().iloc[0]


        df[column_name].fillna(mode_value, inplace=True)
```

In [11]:
```
apply_mode_and_fill_na(df)
```

In [12]:
```
df.isna().sum()
```

Out[12]:
```
HomePlanet        0
CryoSleep         0
Cabin             0
Destination       0
Age             179
VIP               0
RoomService     181
FoodCourt       183
ShoppingMall    208
Spa             183
VRDeck          188
Transported       0
dtype: int64
```

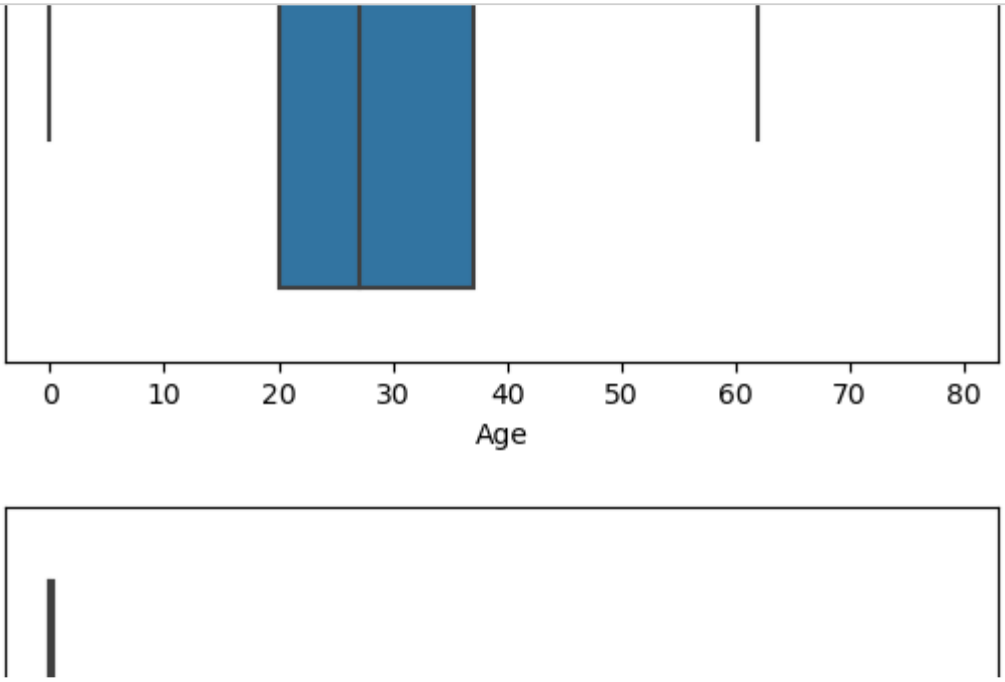
In [13]:
```
def apply_median_and_fill_na(df):
    for column_name in df.select_dtypes(include='float64').columns:

        mode_value = df[column_name].median()


        df[column_name].fillna(mode_value, inplace=True)
```

In [14]:
```
apply_median_and_fill_na(df)
```

In [15]:
```
df.isna().sum()
```

Out[15]:
```
HomePlanet      0
CryoSleep       0
Cabin           0
Destination     0
Age             0
VIP             0
RoomService     0
FoodCourt       0
ShoppingMall    0
Spa             0
VRDeck          0
Transported     0
dtype: int64
```

In [16]:
```
def boxplots(col):
    sns.boxplot(df[col])
    plt.show()

for i in list(df.select_dtypes(exclude=["object","bool"]).columns)[0:]:
    boxplots(i)
```



Encoding Concept

In [17]:
```
df.CryoSleep.unique()
```

Out[17]:
```
array([False,  True])
```

In [18]:
```
df.CryoSleep = df.CryoSleep.astype('category')
df.CryoSleep = df.CryoSleep.cat.codes
```

In [19]:
```
df.VIP.unique()
```

Out[19]:
```
array([False,  True])
```

In [20]:
```
df.VIP=df.VIP.astype("category")
df.VIP=df.VIP.cat.codes
```

In [21]:
```
df.Destination.unique
```

Out[21]:
```
<bound method Series.unique of 0         TRAPPIST-1e
1         TRAPPIST-1e
2         TRAPPIST-1e
3         TRAPPIST-1e
4         TRAPPIST-1e
            ...
8688      55 Cancri e
8689    PSO J318.5-22
8690      TRAPPIST-1e
8691      55 Cancri e
8692      TRAPPIST-1e
Name: Destination, Length: 8693, dtype: object>
```

In [22]:
```
df.Destination=df.Destination.astype("category")
df.Destination=df.Destination.cat.codes
```

In [23]:
```
df.Destination.unique()
```

Out[23]:
```
array([2, 1, 0], dtype=int8)
```

In [24]:
```
df=pd.get_dummies(df,columns=["Destination"])
```

In [25]:
```
df=df.drop(["Destination_0"],axis=1)
```

In [26]:
```
df.Transported=df.Transported.astype("category")
df.Transported=df.Transported.cat.codes
```

In [27]:
```
df.Transported.unique()
```

Out[27]:
```
array([0, 1], dtype=int8)
```

In [28]:
```
df.Cabin.unique()
```

Out[28]:
```
array(['B/0/P', 'F/0/S', 'A/0/S', ..., 'G/1499/S', 'G/1500/S', 'E/608/S'],
      dtype=object)
```

In [29]:
```
df.Cabin=df.Cabin.astype("category")
df.Cabin=df.Cabin.cat.codes
```

ANOVA Testing - two way or multiple way anova

In [30]:
```
import statsmodels.api as sm
from statsmodels.formula.api import ols

model = ols('Transported ~ Cabin', data=df).fit()
anova_result = sm.stats.anova_lm(model, typ=2)
print(anova_result)
```

```
                sum_sq      df          F        PR(>F)
Cabin         6.097901     1.0  24.455898  7.745434e-07
Residual   2167.037955  8691.0        NaN           NaN
```

As we can conclude P value is greator than 0.5, the column is non-significant we can rop it.

In [ ]:
```

```

In [31]:
```
df=df.drop(["Cabin"],axis=1)
```

In [32]:
```
df.head()
```

Out[32]:

	HomePlanet	CryoSleep	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Tra
0	Europa	0	39.0	0	0.0	0.0	0.0	0.0	0.0	
1	Earth	0	24.0	0	109.0	9.0	25.0	549.0	44.0	
2	Europa	0	58.0	1	43.0	3576.0	0.0	6715.0	49.0	
3	Europa	0	33.0	0	0.0	1283.0	371.0	3329.0	193.0	
4	Earth	0	16.0	0	303.0	70.0	151.0	565.0	2.0	

In [33]:
```
df.Transported.value_counts()
```

Out[33]:
```
1    4378
0    4315
Name: Transported, dtype: int64
```

In [ ]:
```

```

In [34]:
```
df.HomePlanet.unique()
```

Out[34]:
```
array(['Europa', 'Earth', 'Mars'], dtype=object)
```

In [35]:
```
df.HomePlanet=df.HomePlanet.astype("category")
df.HomePlanet=df.HomePlanet.cat.codes
df=pd.get_dummies(df,columns=["HomePlanet"])
```

In [36]:
```
df=df.drop(["HomePlanet_0"],axis=1)
```

In [ ]:
```

```

In [37]:
```
df.head()
```

Out[37]:

	CryoSleep	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Transported	Des
0	0	39.0	0	0.0	0.0	0.0	0.0	0.0	0	
1	0	24.0	0	109.0	9.0	25.0	549.0	44.0	1	
2	0	58.0	1	43.0	3576.0	0.0	6715.0	49.0	0	
3	0	33.0	0	0.0	1283.0	371.0	3329.0	193.0	0	
4	0	16.0	0	303.0	70.0	151.0	565.0	2.0	1	

In [38]:
```
df1=df.copy()
```

In [39]:
```
df1.head()
```

Out[39]:

	CryoSleep	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Transported	Des
0	0	39.0	0	0.0	0.0	0.0	0.0	0.0	0	
1	0	24.0	0	109.0	9.0	25.0	549.0	44.0	1	
2	0	58.0	1	43.0	3576.0	0.0	6715.0	49.0	0	
3	0	33.0	0	0.0	1283.0	371.0	3329.0	193.0	0	
4	0	16.0	0	303.0	70.0	151.0	565.0	2.0	1	

Handling outlier

In [40]:
```
Age_q1 = df1.Age.quantile(0.25)
Age_q3 = df1.Age.quantile(0.75)
Age_iqr = Age_q3 -  Age_q1
Age_upper = Age_q3 + 1.5 * Age_iqr
Age_lower = Age_q1 - 1.5 * Age_iqr
```

In [41]:
```
df1.Age=np.where(df1['Age']>Age_upper,Age_upper,
                 np.where(df1['Age']<Age_lower,Age_lower,
                          df1['Age']))
```

In [ ]:

In [42]:
```
RoomService_q1 = df1.RoomService.quantile(0.25)
RoomService_q3 = df1.RoomService.quantile(0.75)
RoomService_iqr = RoomService_q3 -  RoomService_q1
RoomService_upper = RoomService_q3 + 1.5 * RoomService_iqr
RoomService_lower = RoomService_q1 - 1.5 * RoomService_iqr
```

In [43]:
```
df1.RoomService=np.where(df1['RoomService']>RoomService_upper,RoomService_upper,
                np.where(df1['RoomService']<RoomService_lower,RoomService_lower,
                         df1['RoomService']))
```

In [ ]:

In [44]:
```
FoodCourt_q1 = df1.FoodCourt.quantile(0.25)
FoodCourt_q3 = df1.FoodCourt.quantile(0.75)
FoodCourt_iqr = FoodCourt_q3 -  FoodCourt_q1
FoodCourt_upper = FoodCourt_q3 + 1.5 * FoodCourt_iqr
FoodCourt_lower = FoodCourt_q1 - 1.5 * FoodCourt_iqr
```

In [45]:
```
df1.FoodCourt=np.where(df1['FoodCourt']>FoodCourt_upper,FoodCourt_upper,
                np.where(df1['FoodCourt']<FoodCourt_lower,FoodCourt_lower,
                         df1['FoodCourt']))
```

In [ ]:

In [46]:
```
ShoppingMall_q1 = df1.ShoppingMall.quantile(0.25)
ShoppingMall_q3 = df1.ShoppingMall.quantile(0.75)
ShoppingMall_iqr = ShoppingMall_q3 -  ShoppingMall_q1
ShoppingMall_upper = ShoppingMall_q3 + 1.5 * ShoppingMall_iqr
ShoppingMall_lower = ShoppingMall_q1 - 1.5 * ShoppingMall_iqr
```

In [47]:
```
df1.ShoppingMall=np.where(df1['ShoppingMall']>ShoppingMall_upper,ShoppingMall_upp
                np.where(df1['ShoppingMall']<ShoppingMall_lower,ShoppingMall_lowe
                         df1['ShoppingMall']))
```

In [ ]:

In [48]:
```
Spa_q1 = df1.Spa.quantile(0.25)
Spa_q3 = df1.Spa.quantile(0.75)
Spa_iqr = Spa_q3 -  Spa_q1
Spa_upper = Spa_q3 + 1.5 * Spa_iqr
Spa_lower = Spa_q1 - 1.5 * Spa_iqr
```

In [49]:
```
df1.Spa=np.where(df1['Spa']>Spa_upper,Spa_upper,
                np.where(df1['Spa']<Spa_lower,Spa_lower,
                         df1['Spa']))
```

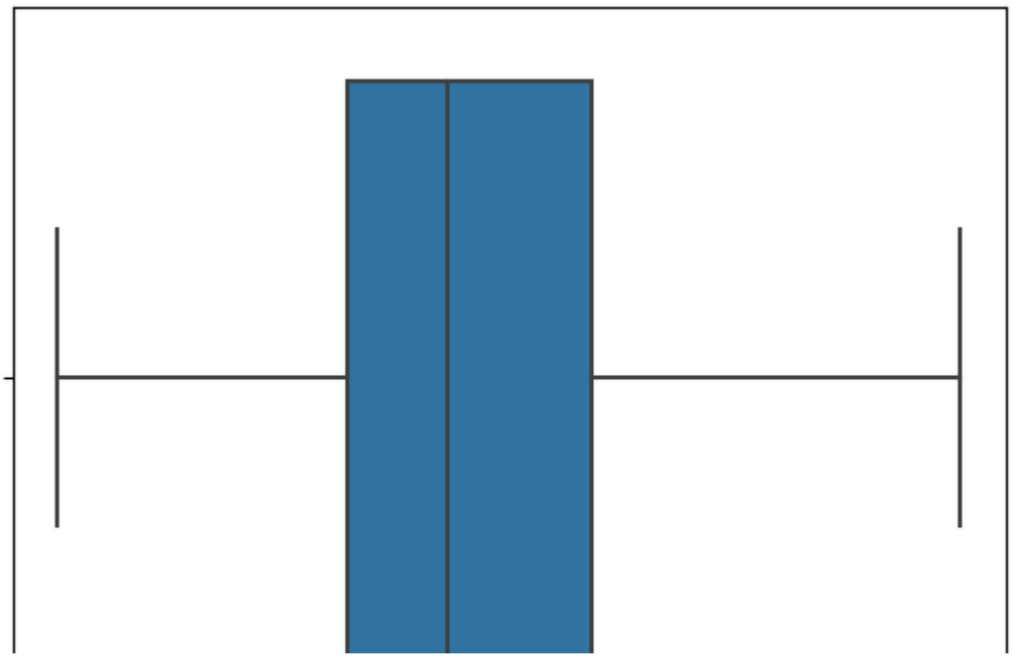
In [ ]:

In [50]:
```
VRDeck_q1 = df1.VRDeck.quantile(0.25)
VRDeck_q3 = df1.VRDeck.quantile(0.75)
VRDeck_iqr = VRDeck_q3 -  VRDeck_q1
VRDeck_upper = VRDeck_q3 + 1.5 * VRDeck_iqr
VRDeck_lower = VRDeck_q1 - 1.5 * VRDeck_iqr
```

In [51]:
```
df1.VRDeck=np.where(df1['VRDeck']>VRDeck_upper,VRDeck_upper,
                np.where(df1['VRDeck']<VRDeck_lower,VRDeck_lower,
                         df1['VRDeck']))
```

In [ ]:

In [52]:
```
def boxplots(col):
    sns.boxplot(df1[col])
    plt.show()

for i in list(df1.select_dtypes(exclude=["object","bool"]).columns)[0:]:
    boxplots(i)
```



Spliting the data into independent variable and dependent variable

In [53]:
```
x=df.drop(["Transported"],axis=1)
y=df[["Transported"]]
```

In [54]:
```
x.head()
```

Out[54]:

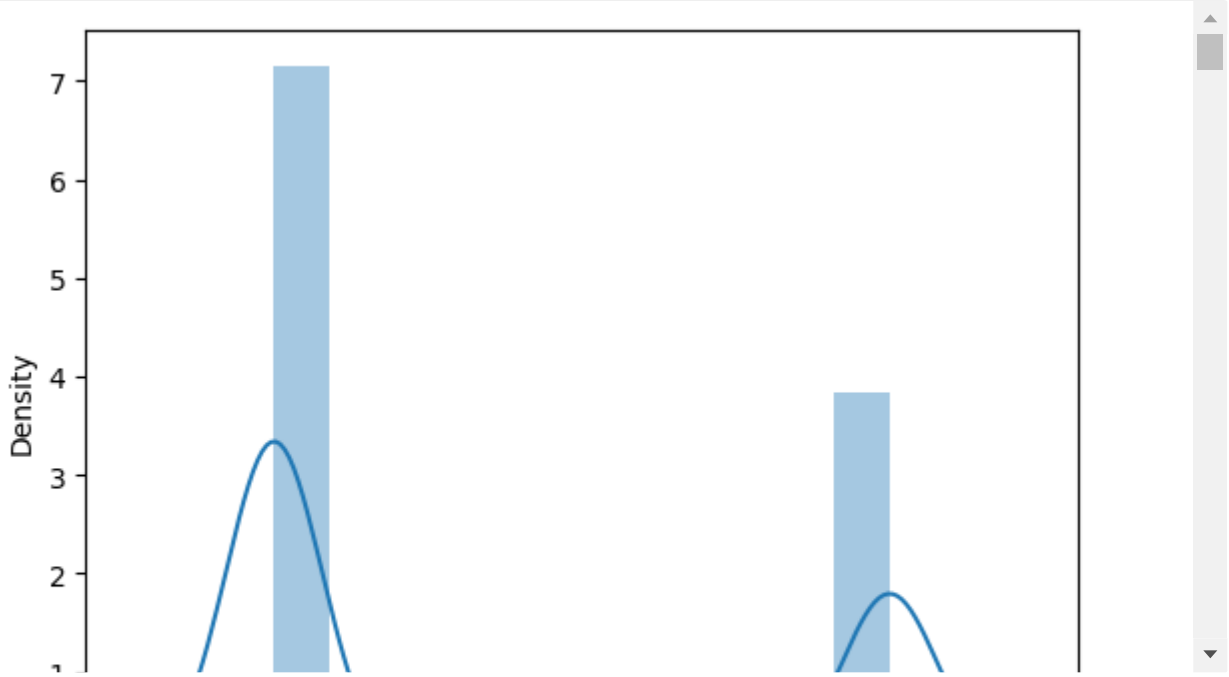
	CryoSleep	Age	VIP	RoomService	FoodCourt	ShoppingMall	Spa	VRDeck	Destination_1	D
0	0	39.0	0	0.0	0.0	0.0	0.0	0.0	0	
1	0	24.0	0	109.0	9.0	25.0	549.0	44.0	0	
2	0	58.0	1	43.0	3576.0	0.0	6715.0	49.0	0	
3	0	33.0	0	0.0	1283.0	371.0	3329.0	193.0	0	
4	0	16.0	0	303.0	70.0	151.0	565.0	2.0	0	

In [55]:
```
y.head()
```

Out[55]:

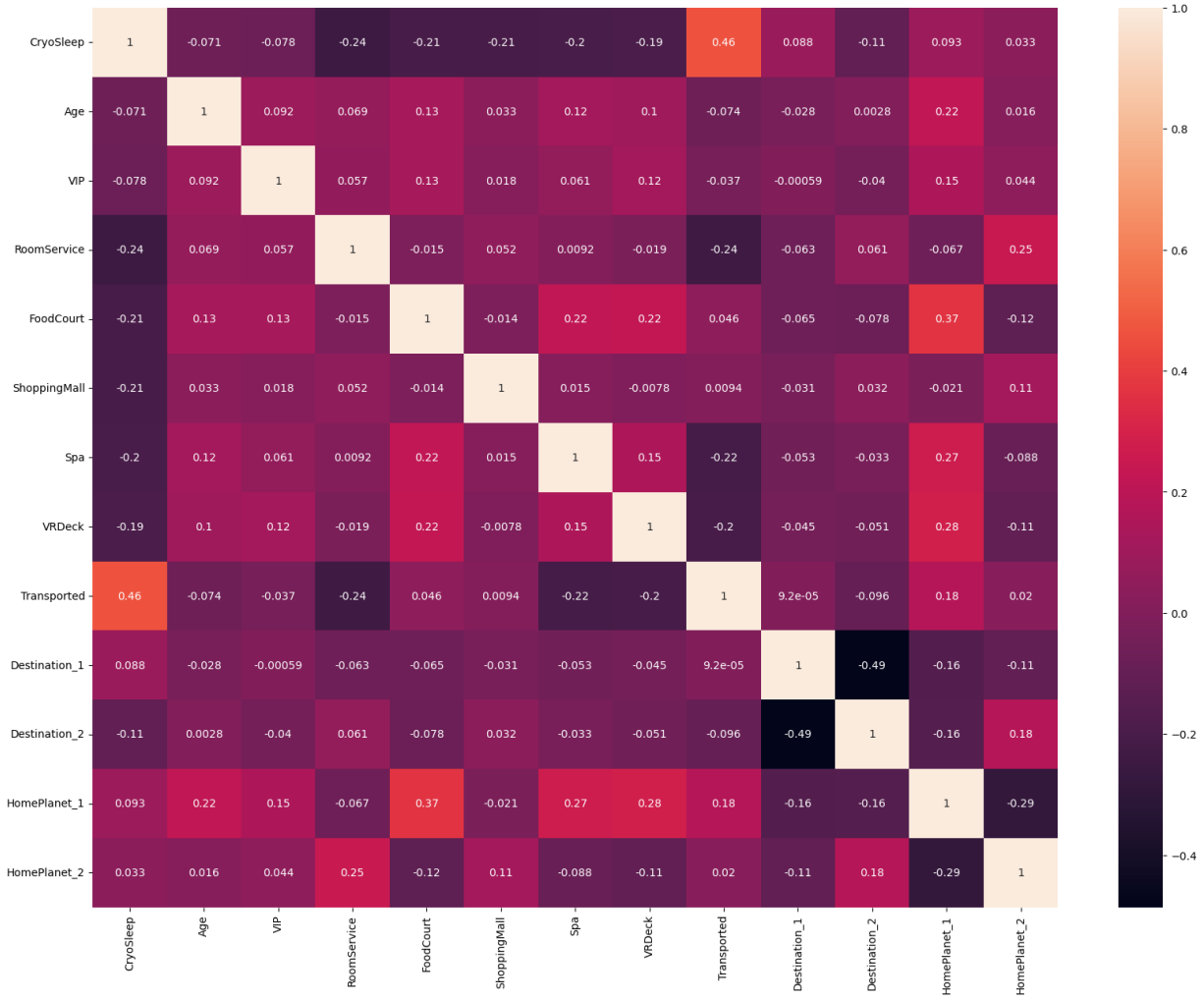
	Transported
0	0
1	1
2	0
3	0
4	1

In [56]:
```
def distplot(col):
    sns.distplot(df[col])
    plt.show()

for i in list(df.columns):
    distplot(i)
```

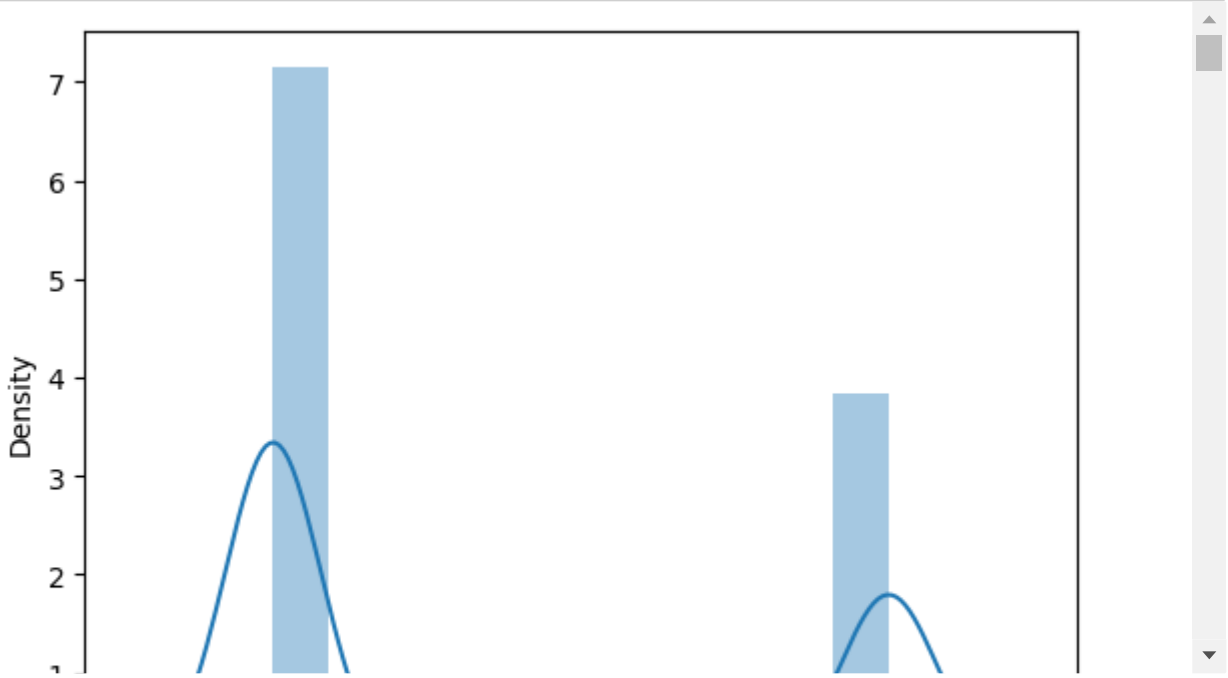


Finding correlation

In [57]:
```
plt.figure(figsize=(20,15))

sns.heatmap(df.corr(),annot=True)
plt.show()
```



In [58]:
```
def distplot(col):
    sns.distplot(df1[col])
    plt.show()

for i in list(df1.columns):
    distplot(i)
```



Split the data into training and test for building the model and for prediction

In [ ]:
```
```

In [59]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.25,random_state=1
```

Logistic Regression Model

In [60]:
```
from sklearn.linear_model import LogisticRegression
```

In [61]:
```
logit = LogisticRegression(multi_class='multinomial')
logit.fit(x_train, y_train)
```

Out[61]:
```
LogisticRegression(multi_class='multinomial')
```

Predict the data

In [62]:
```
y_pred_train = logit.predict(x_train)
y_pred_test = logit.predict(x_test)
```

Evaluate the model

In [63]:
```
from sklearn.metrics import accuracy_score, classification_report, confusion_matr
```

In [64]:
```
print("Trainging Accuracy Score :", accuracy_score(y_train, y_pred_train))
print("****************"*10)
print("Test Accuracy Score :", accuracy_score(y_test, y_pred_test))
```

```
Trainging Accuracy Score : 0.786470317533364
********************************************************************************
********************************************************************************
**
Test Accuracy Score : 0.7916283348666053
```

In [65]:
```
print( classification_report(y_train, y_pred_train))
print("****************"*10)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.79      0.77      0.78      3227
           1       0.78      0.80      0.79      3292

    accuracy                           0.79      6519
   macro avg       0.79      0.79      0.79      6519
weighted avg       0.79      0.79      0.79      6519

********************************************************************************
********************************************************************************
**
              precision    recall  f1-score   support

           0       0.80      0.77      0.79      1088
           1       0.78      0.81      0.80      1086

    accuracy                           0.79      2174
   macro avg       0.79      0.79      0.79      2174
weighted avg       0.79      0.79      0.79      2174
```

In [66]:
```
print( confusion_matrix(y_train, y_pred_train))
print("****************"*10)
print(confusion_matrix(y_test, y_pred_test))
```

```
[[2498  729]
 [ 663 2629]]
********************************************************************************
********************************************************************************
**
[[841 247]
 [206 880]]
```

ROC AND AUC

In [67]:
```
from sklearn.metrics import roc_auc_score
logit_roc_auc = roc_auc_score(y_test, y_pred_test)
logit_roc_auc
```

Out[67]:
```
0.7916455083414582
```

In [68]:
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_test)
display(fpr[:10])
display(tpr[:10])
display(thresholds[:10])
```

```
array([0.        , 0.22702206, 1.        ])

array([0.        , 0.81031308, 1.        ])

array([2, 1, 0], dtype=int8)
```

In [69]:
```
tpr
```

Out[69]:
```
array([0.        , 0.81031308, 1.        ])
```

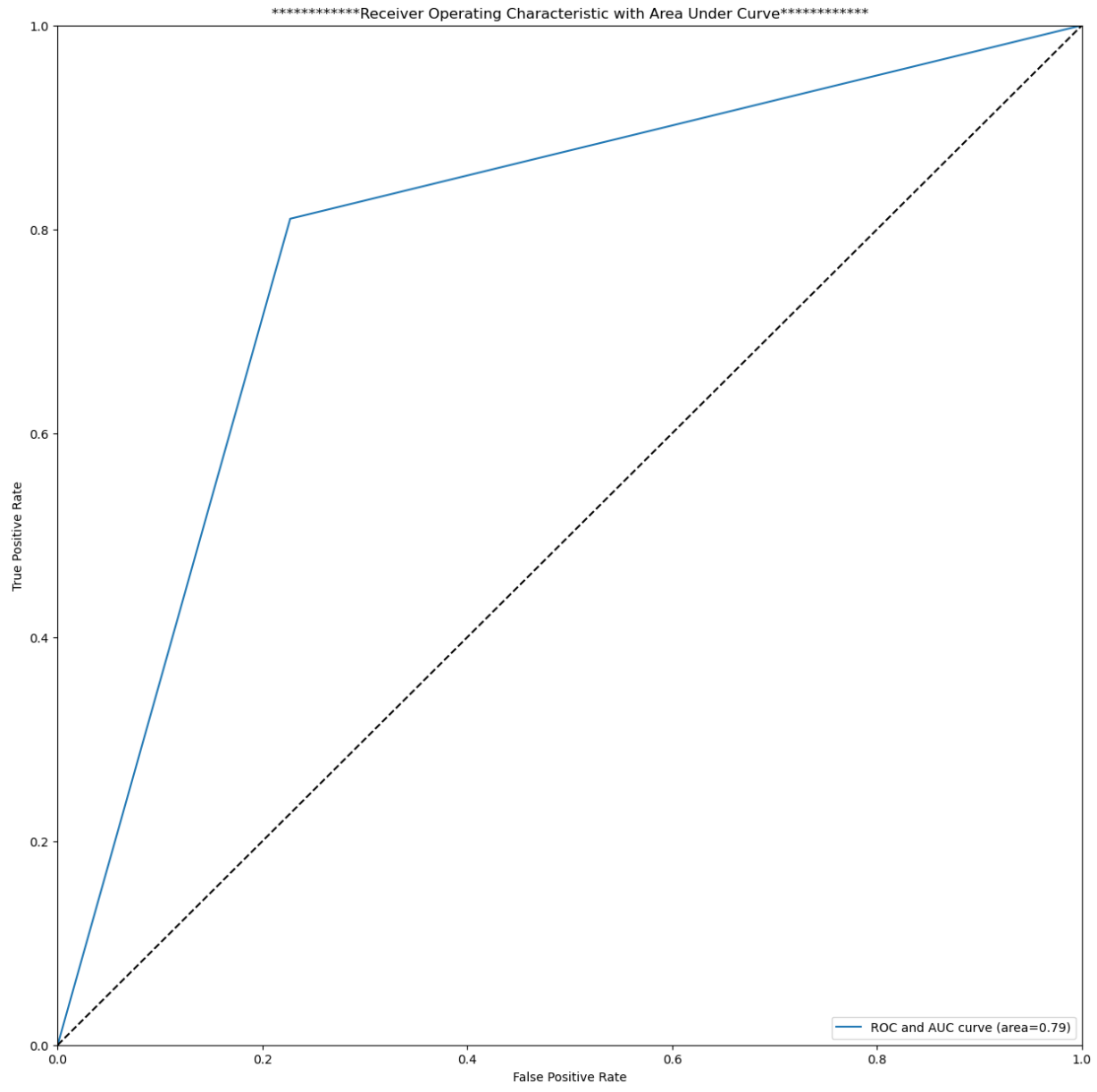
In [70]:
```
thresholds
```

Out[70]:
```
array([2, 1, 0], dtype=int8)
```

In [ ]:
```

```

Plotting ROC and AUC curve

In [71]:
```
plt.figure(figsize=(15,15))
plt.plot(fpr, tpr, label="ROC and AUC curve (area=%0.2f)" % logit_roc_auc)
plt.plot([0,1],[0,1], 'k--')
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title("************Receiver Operating Characteristic with Area Under Curve***
plt.legend(loc='lower right')
plt.show()
```



Cross Validation approach - K-Fold Method

In [78]:
```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(logit, x_train, y_train, cv=10)
test_accuracy = cross_val_score(logit, x_test, y_test, cv=10)
print(training_accuracy)
print()
print(test_accuracy)
print()
print("Training Avg Accuracy", training_accuracy.mean())
print()
print("Test Avg Accuracy", test_accuracy.mean())
```

```
[0.80214724 0.79141104 0.80828221 0.78067485 0.79601227 0.75153374
 0.76226994 0.79601227 0.77760736 0.78033794]

[0.79816514 0.78899083 0.80733945 0.80733945 0.77419355 0.83870968
 0.76497696 0.80184332 0.77419355 0.80184332]

Training Avg Accuracy 0.7846288861873663

Test Avg Accuracy 0.7957595231048915
```

Building Logistic Regression - MultiClass Classification

OVR/OVA

In [73]:
```
from sklearn.linear_model import LogisticRegression
logit_ovr = LogisticRegression(multi_class='ovr')
logit_ovr.fit(x_train, y_train)
```

Out[73]:
```
LogisticRegression(multi_class='ovr')
```

In [74]:
```
y_pred_train = logit_ovr.predict(x_train)
y_pred_test = logit_ovr.predict(x_test)
```

In [ ]:
```

```

In [75]:
```
print("Trainging Accuracy Score :", accuracy_score(y_train, y_pred_train))
print("****************"*10)
print("Test Accuracy Score :", accuracy_score(y_test, y_pred_test))
```

```
Trainging Accuracy Score : 0.7853965332106151
********************************************************************************
********************************************************************************
**
Test Accuracy Score : 0.7916283348666053
```

In [76]:
```
print( classification_report(y_train, y_pred_train))
print("****************"*10)
print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.79      0.77      0.78      3227
           1       0.78      0.80      0.79      3292

    accuracy                           0.79      6519
   macro avg       0.79      0.79      0.79      6519
weighted avg       0.79      0.79      0.79      6519

**********************************************************************
**********************************************************************
**
              precision    recall  f1-score   support

           0       0.80      0.77      0.79      1088
           1       0.78      0.81      0.80      1086

    accuracy                           0.79      2174
   macro avg       0.79      0.79      0.79      2174
weighted avg       0.79      0.79      0.79      2174
```

In [77]:
```
print( confusion_matrix(y_train, y_pred_train))
print("****************"*10)
print(confusion_matrix(y_test, y_pred_test))
```

```
[[2498  729]
 [ 670 2622]]
**********************************************************************
**********************************************************************
**
[[842 246]
 [207 879]]
```

Cross Validation approach - K-Fold Method

In [ ]:
```
```

In [79]:
```
from sklearn.model_selection import cross_val_score
training_accuracy = cross_val_score(logit_ovr, x_train, y_train, cv=10)
test_accuracy = cross_val_score(logit_ovr, x_test, y_test, cv=10)
print(training_accuracy)
print()
print(test_accuracy)
print()
print("Training Avg Accuracy", training_accuracy.mean())
print()
print("Test Avg Accuracy", test_accuracy.mean())
```

```
[0.80214724 0.79141104 0.80828221 0.78527607 0.81441718 0.75153374
 0.77147239 0.79601227 0.77607362 0.78494624]

[0.79357798 0.77522936 0.80275229 0.82110092 0.77419355 0.83870968
 0.76497696 0.80184332 0.77419355 0.79262673]

Training Avg Accuracy 0.7881572003430305

Test Avg Accuracy 0.7939204329260559
```

In [ ]: