About Dataset

Context

- This dataset contains an airline passenger satisfaction survey. What factors are highly correlated to a satisfied (or dissatisfied) passenger? We have to predict passenger satisfaction? Gender: Gender of the passengers (Female, Male) ### **Content**
- Customer Type: The customer type (Loyal customer, disloyal customer)
- Age: The actual age of the passengers
- Type of Travel: Purpose of the flight of the passengers (Personal Travel, Business Travel)
- Class: Travel class in the plane of the passengers (Business, Eco, Eco Plus)
- Flight distance: The flight distance of this journey
- Inflight wifi service: Satisfaction level of the inflight wifi service (0:Not Applicable;1-5)
- Departure/Arrival time convenient: Satisfaction level of Departure/Arrival time convenient
- Ease of Online booking: Satisfaction level of online booking
- Gate location: Satisfaction level of Gate location
- Food and drink: Satisfaction level of Food and drink
- Online boarding: Satisfaction level of online boarding
- Seat comfort: Satisfaction level of Seat comfort
- Inflight entertainment: Satisfaction level of inflight entertainment
- On-board service: Satisfaction level of On-board service
- Leg room service: Satisfaction level of Leg room service
- Baggage handling: Satisfaction level of baggage handling
- Check-in service: Satisfaction level of Check-in service
- Inflight service: Satisfaction level of inflight service
- Cleanliness: Satisfaction level of Cleanliness
- Departure Delay in Minutes: Minutes delayed when departure
- Arrival Delay in Minutes: Minutes delayed when Arrival
- Satisfaction: Airline satisfaction level(Satisfaction, neutral or dissatisfaction)

Target

- Satisfied
- Dissatisfied

Approach to the prediction

- 1.import all libaray
- 2.Load and audit the data
- 3.Data prepration and Data Transformation
  - 1.missing value
  - 2.Inconsistent value:Replace all transformation with consistent values
  - 3.outliers
- 4.Data visualization
- 5.Data analysis
  - 1.Uni-variate Analysis(Mean,Median,Mode,Skewness)
  - 2.Bi-variate Analysis(Correlation,Covariance,Chi-Square Test)
  - 3.Multi-Variate Analysis
    - 1.Regression:Not a regressiion
    - 2.Classification
      - 1.Apply Logistic Regression
      - 2.Apply Decision Tree
      - 3.Apply RandomForestClassifier
      - 4.Apply AdaBoostClassifier
      - 5.Apply GradientBoostingClassifier
      - 6.Apply KNeighborsClassifier
      - 7.Apply SVC
      - 8.Evalaute Between Logistic, Decision Tree, RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier, KNeighborsClassifier,SVC Which is the better model

IMPORT ALL LIBRARIES

In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OrdinalEncoder,LabelEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier,AdaBoostClassifier,GradientBoos
from xgboost import XGBClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,confusion_matrix,classification_report

import warnings
```

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

DATA COLLECTION AND ANALYSIS

In [ ]:
```
df=pd.read_csv('train.csv')
```

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

Out[ ]:

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/A time conve
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103904 entries, 0 to 103903
Data columns (total 25 columns):
 #   Column                            Non-Null Count   Dtype  
---  ------                            --------------   -----  
 0   Unnamed: 0                        103904 non-null  int64  
 1   id                                103904 non-null  int64  
 2   Gender                            103904 non-null  object 
 3   Customer Type                     103904 non-null  object 
 4   Age                               103904 non-null  int64  
 5   Type of Travel                    103904 non-null  object 
 6   Class                             103904 non-null  object 
 7   Flight Distance                   103904 non-null  int64  
 8   Inflight wifi service             103904 non-null  int64  
 9   Departure/Arrival time convenient  103904 non-null  int64  
 10  Ease of Online booking            103904 non-null  int64  
 11  Gate location                     103904 non-null  int64  
 12  Food and drink                    103904 non-null  int64  
 13  Online boarding                   103904 non-null  int64  
 14  Seat comfort                      103904 non-null  int64  
 15  Inflight entertainment            103904 non-null  int64  
 16  On-board service                  103904 non-null  int64  
 17  Leg room service                  103904 non-null  int64  
 18  Baggage handling                  103904 non-null  int64  
 19  Checkin service                   103904 non-null  int64  
 20  Inflight service                  103904 non-null  int64  
 21  Cleanliness                       103904 non-null  int64  
 22  Departure Delay in Minutes        103904 non-null  int64  
 23  Arrival Delay in Minutes          103594 non-null  float64
 24  satisfaction                      103904 non-null  object 
dtypes: float64(1), int64(19), object(5)
memory usage: 19.8+ MB
```

Filling Null values in column: Arrival Delay in Minutes

In [ ]:
```
df['Arrival Delay in Minutes']=df['Arrival Delay in Minutes'].fillna(df['Arrival D
```

In [ ]:
```
df.isnull().sum()
```

Out[ ]:
```
Unnamed: 0                           0
id                                   0
Gender                               0
Customer Type                        0
Age                                  0
Type of Travel                       0
Class                                0
Flight Distance                      0
Inflight wifi service                0
Departure/Arrival time convenient    0
Ease of Online booking               0
Gate location                        0
Food and drink                       0
Online boarding                      0
Seat comfort                         0
Inflight entertainment               0
On-board service                     0
Leg room service                     0
Baggage handling                     0
Checkin service                      0
Inflight service                     0
Cleanliness                          0
Departure Delay in Minutes           0
Arrival Delay in Minutes             0
satisfaction                         0
dtype: int64
```

Changing d.Type

In [ ]:
```
# Float to int64
df['Arrival Delay in Minutes']=df['Arrival Delay in Minutes'].astype('Int64')
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103904 entries, 0 to 103903
Data columns (total 25 columns):
 #   Column                            Non-Null Count   Dtype
---  ------                            --------------   -----
 0   Unnamed: 0                        103904 non-null  int64
 1   id                                103904 non-null  int64
 2   Gender                            103904 non-null  object
 3   Customer Type                     103904 non-null  object
 4   Age                               103904 non-null  int64
 5   Type of Travel                    103904 non-null  object
 6   Class                             103904 non-null  object
 7   Flight Distance                   103904 non-null  int64
 8   Inflight wifi service             103904 non-null  int64
 9   Departure/Arrival time convenient 103904 non-null  int64
 10  Ease of Online booking            103904 non-null  int64
 11  Gate location                     103904 non-null  int64
 12  Food and drink                    103904 non-null  int64
 13  Online boarding                   103904 non-null  int64
 14  Seat comfort                      103904 non-null  int64
 15  Inflight entertainment            103904 non-null  int64
 16  On-board service                  103904 non-null  int64
 17  Leg room service                  103904 non-null  int64
 18  Baggage handling                  103904 non-null  int64
 19  Checkin service                   103904 non-null  int64
 20  Inflight service                  103904 non-null  int64
 21  Cleanliness                       103904 non-null  int64
 22  Departure Delay in Minutes        103904 non-null  int64
 23  Arrival Delay in Minutes          103904 non-null  Int64
 24  satisfaction                      103904 non-null  object
dtypes: Int64(1), int64(19), object(5)
memory usage: 19.9+ MB
```

Dropping unnecessary column

In [ ]:
```
df.drop(df.iloc[:,0:2],axis=1,inplace=True)
```

In [ ]:
```
df.sample(10)
```

Out[ ]:

	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking
55403	Male	Loyal Customer	57	Personal Travel	Eco	347	1	5	1
55147	Female	Loyal Customer	44	Business travel	Business	1587	2	3	3
91731	Male	Loyal Customer	26	Business travel	Business	2342	1	1	1
1916	Male	Loyal Customer	40	Personal Travel	Eco	134	3	2	3
48817	Male	disloyal Customer	23	Business travel	Eco	1016	5	5	5
88363	Female	Loyal Customer	37	Business travel	Business	3528	1	1	4
97034	Female	Loyal Customer	41	Personal Travel	Eco Plus	545	5	3	5
96944	Female	disloyal Customer	26	Business travel	Eco Plus	416	1	0	1
96597	Female	Loyal Customer	61	Personal Travel	Eco	1303	2	5	2
21541	Female	disloyal Customer	27	Business travel	Business	639	5	5	5

Since we having [Departure/Arrival time convenient] column so we can remove both ['Departure Delay in Minutes', 'Arrival Delay in Minutes'] column

In [ ]:
```
df.drop(['Departure Delay in Minutes','Arrival Delay in Minutes'],axis=1,inplace=Tr
df.head()
```

Out[ ]:

	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	loc
0	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	3	
1	Male	disloyal Customer	25	Business travel	Business	235	3	2	3	
2	Female	Loyal Customer	26	Business travel	Business	1142	2	2	2	
3	Female	Loyal Customer	25	Business travel	Business	562	2	5	5	
4	Male	Loyal Customer	61	Business travel	Business	214	3	3	3	

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

Out[ ]:

	Age	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate locati
count	103904.000000	103904.000000	103904.000000	103904.000000	103904.000000	103904.000(
mean	39.379706	1189.448375	2.729683	3.060296	2.756901	2.976{
std	15.114964	997.147281	1.327829	1.525075	1.398929	1.277(
min	7.000000	31.000000	0.000000	0.000000	0.000000	0.000(
25%	27.000000	414.000000	2.000000	2.000000	2.000000	2.000(
50%	40.000000	843.000000	3.000000	3.000000	3.000000	3.000(
75%	51.000000	1743.000000	4.000000	4.000000	4.000000	4.000(
max	85.000000	4983.000000	5.000000	5.000000	5.000000	5.000(

RENAME THE COLUMNS

In [ ]:
```
df.rename(columns={'Customer Type':'Customer_type','Type of Travel':'Type_of_Trave
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103904 entries, 0 to 103903
Data columns (total 21 columns):
 #   Column                                  Non-Null Count   Dtype 
---  ------                                  --------------   ----- 
 0   Gender                                  103904 non-null  object
 1   Customer_type                           103904 non-null  object
 2   Age                                     103904 non-null  int64 
 3   Type_of_Travel                          103904 non-null  object
 4   Class                                   103904 non-null  object
 5   Flight_Distance                         103904 non-null  int64 
 6   Inflight_wifi_service                   103904 non-null  int64 
 7   Departure_or_Arrival_time_convenient    103904 non-null  int64 
 8   Ease_of_Online_booking                  103904 non-null  int64 
 9   Gate_location                           103904 non-null  int64 
 10  Food_and_drink                          103904 non-null  int64 
 11  Online_boarding                         103904 non-null  int64 
 12  Seat_comfort                            103904 non-null  int64 
 13  Inflight_entertainment                  103904 non-null  int64 
 14  On_board_service                        103904 non-null  int64 
 15  Leg_room_service                        103904 non-null  int64 
 16  Baggage_handling                        103904 non-null  int64 
 17  Checkin_service                         103904 non-null  int64 
 18  Inflight_service                        103904 non-null  int64 
 19  Cleanliness                             103904 non-null  int64 
 20  satisfaction                            103904 non-null  object
dtypes: int64(16), object(5)
memory usage: 16.6+ MB
```

In [ ]:
```
df['satisfaction'].value_counts()
```

Out[ ]:
```
dissatisfied    58879
satisfied       45025
Name: satisfaction, dtype: int64
```

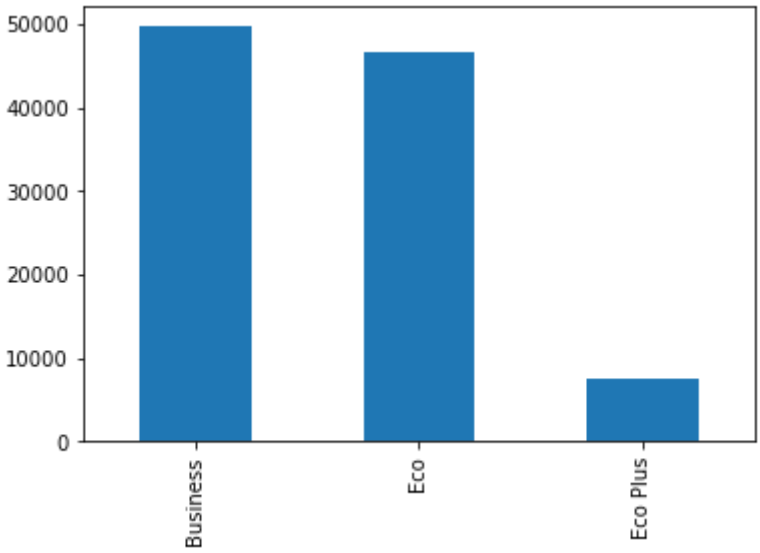
Data Visualization

In [ ]:
```
# # UNIVARIATE
df.hist(figsize=(24,24))
```

Out[ ]:
```
array([[<AxesSubplot:title={'center':'Age'}>,
        <AxesSubplot:title={'center':'Flight_Distance'}>,
        <AxesSubplot:title={'center':'Inflight_wifi_service'}>,
        <AxesSubplot:title={'center':'Departure_or_Arrival_time_convenient'}>],
       [<AxesSubplot:title={'center':'Ease_of_Online_booking'}>,
        <AxesSubplot:title={'center':'Gate_location'}>,
        <AxesSubplot:title={'center':'Food_and_drink'}>,
        <AxesSubplot:title={'center':'Online_boarding'}>],
       [<AxesSubplot:title={'center':'Seat_comfort'}>,
        <AxesSubplot:title={'center':'Inflight_entertainment'}>,
        <AxesSubplot:title={'center':'On_board_service'}>,
        <AxesSubplot:title={'center':'Leg_room_service'}>],
       [<AxesSubplot:title={'center':'Baggage_handling'}>,
        <AxesSubplot:title={'center':'Checkin_service'}>,
        <AxesSubplot:title={'center':'Inflight_service'}>,
        <AxesSubplot:title={'center':'Cleanliness'}>]], dtype=object)
```

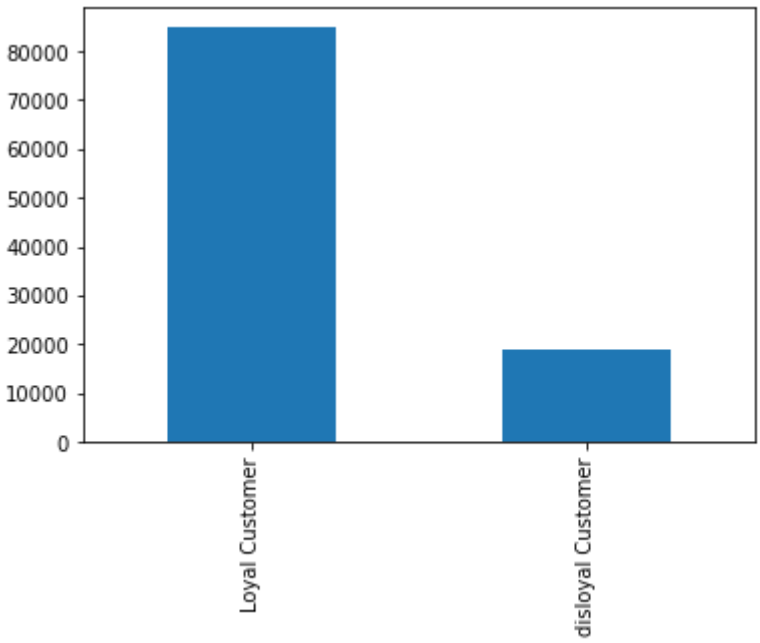
In [ ]:
```
# Very less Eco plus class travellers
df['Class'].value_counts().plot(kind='bar')
```

Out[ ]:
```
<AxesSubplot:>
```



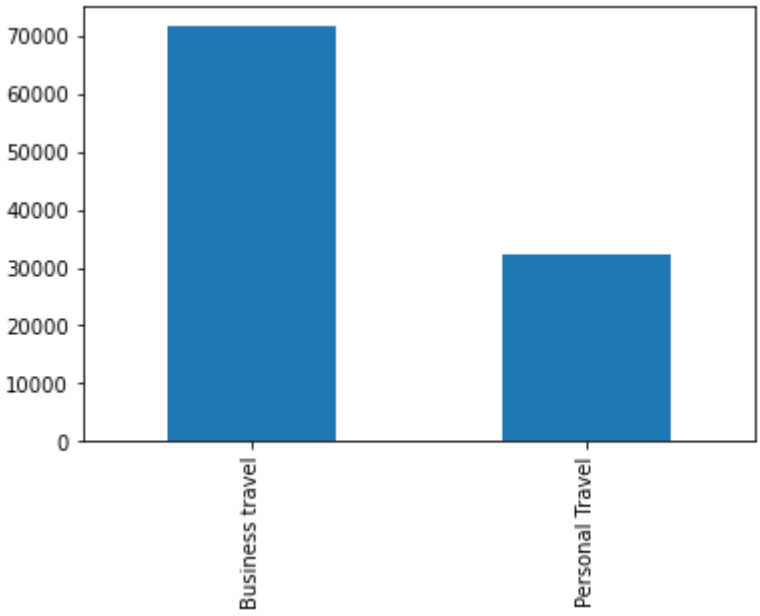
In [ ]:
```
# Most are loyal Customers
df['Customer_type'].value_counts().plot(kind='bar')
```

Out[ ]:
```
<AxesSubplot:>
```



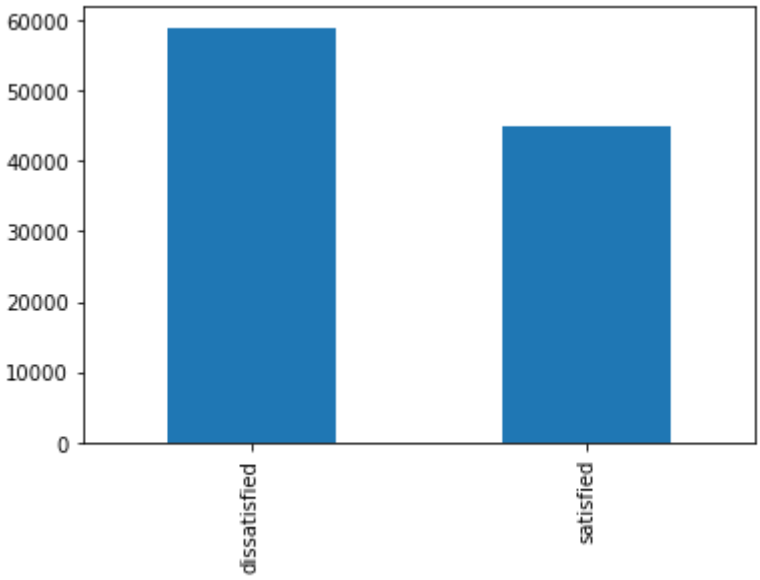
In [ ]:
```
# Most of travellers having business travel
df['Type_of_Travel'].value_counts().plot(kind='bar')
```

Out[ ]:
```
<AxesSubplot:>
```



In [ ]:
```
df['satisfaction'].value_counts().plot(kind='bar')
# it's a balanced data
```

Out[ ]:
```
<AxesSubplot:>
```

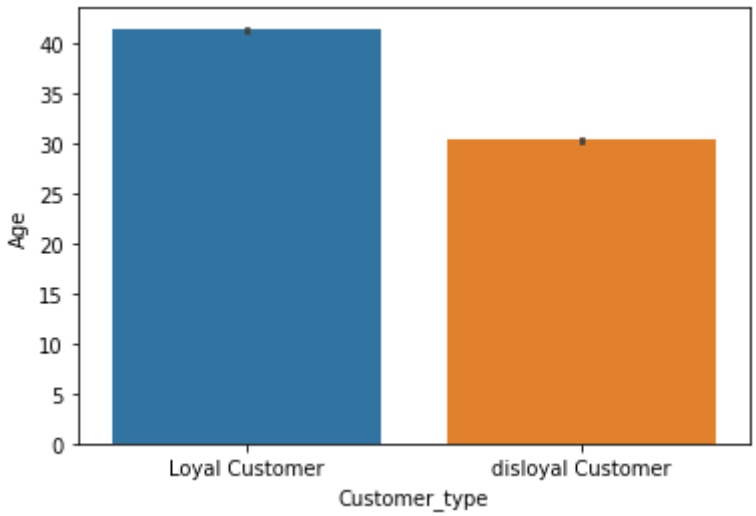


In [ ]:
```
# Multivariate
plt.figure(figsize=(12,12))
sns.scatterplot(df['Flight_Distance'],df['Age'],hue=df['Gender'])
# As we can see aged people travelling less distance comparitvely youngesters(above
# The youngsters travelling high distance who are above 20 and below 60
```

Out[ ]:
```
<AxesSubplot:xlabel='Flight_Distance', ylabel='Age'>
```

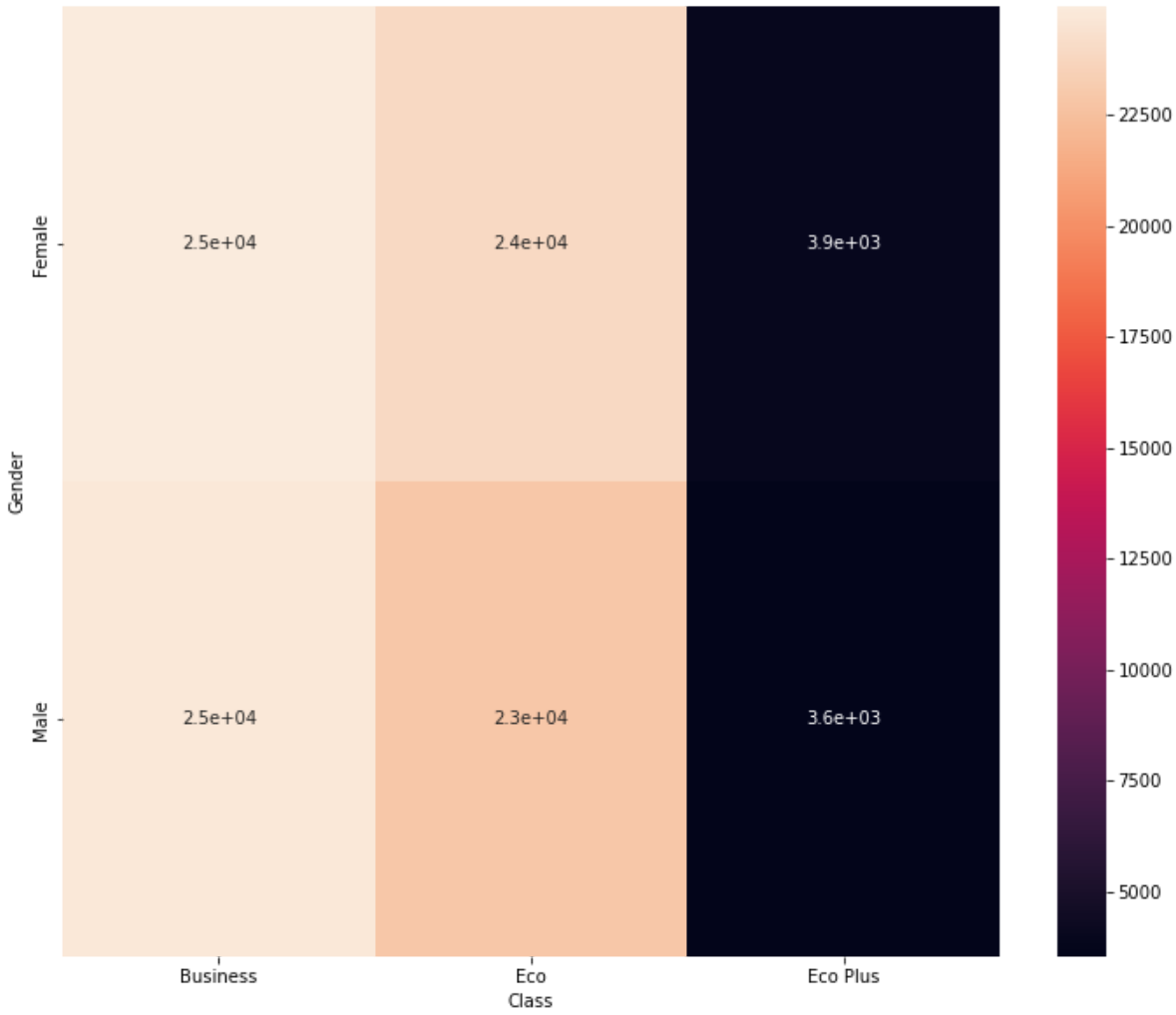
In [ ]:
```
# Bivariate
sns.barplot(df['Customer_type'],df['Age'])
#Below age of 30 are more disloyal
```

Out[ ]:
```
<AxesSubplot:xlabel='Customer_type', ylabel='Age'>
```



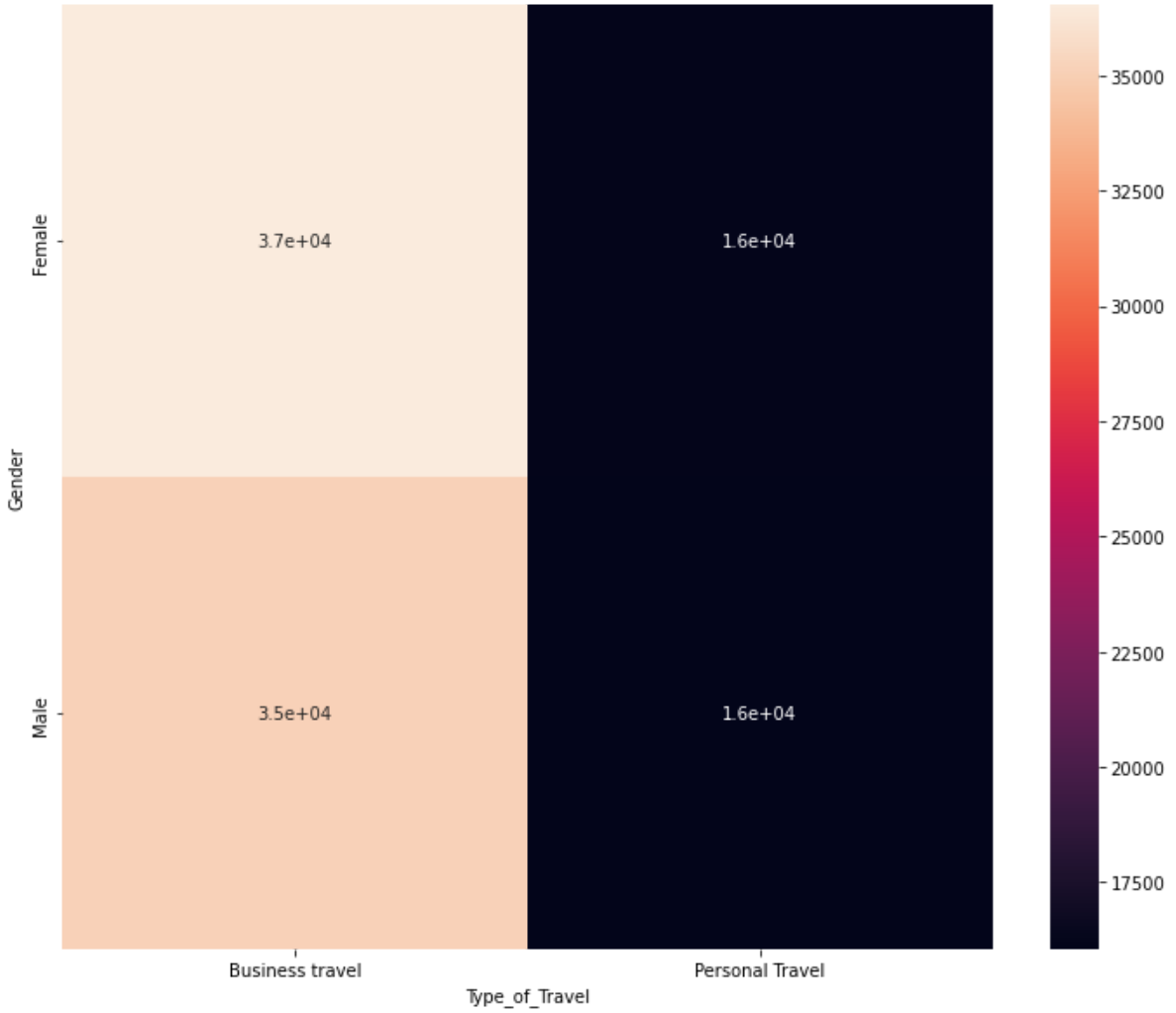
In [ ]:
```
plt.figure(figsize=(12,10))
sns.heatmap(pd.crosstab(df['Gender'],df['Class']),annot=True)
# Mostly Female have chosen Business class
```

Out[ ]: <AxesSubplot:xlabel='Class', ylabel='Gender'>



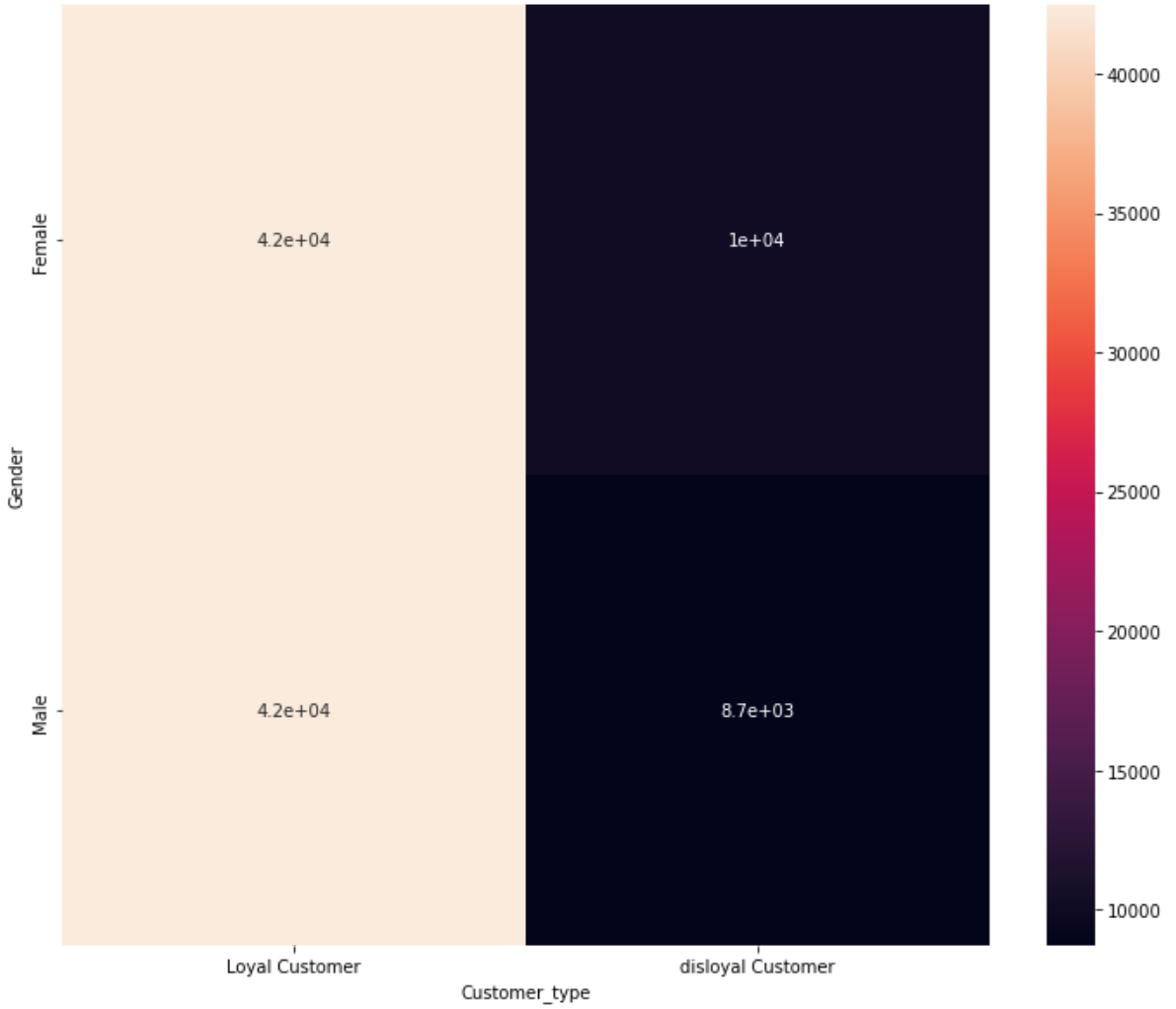
In [ ]:
```
plt.figure(figsize=(12,10))
sns.heatmap(pd.crosstab(df['Gender'],df['Type_of_Travel']),annot=True)
# Here we can see most of the female are Business Traveller
```

Out[ ]: <AxesSubplot:xlabel='Type_of_Travel', ylabel='Gender'>



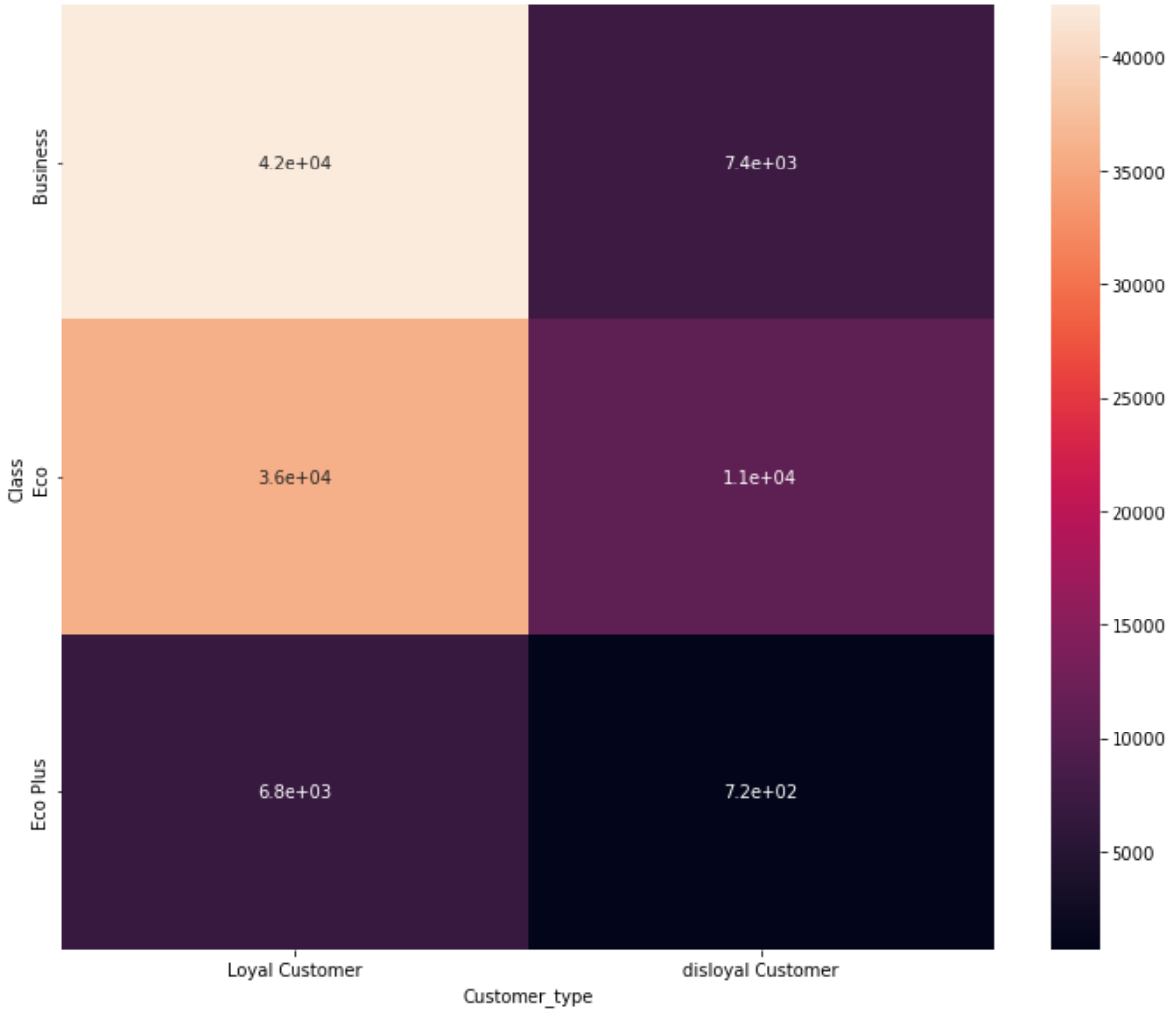
In [ ]:
```
plt.figure(figsize=(12,10))
sns.heatmap(pd.crosstab(df['Gender'],df['Customer_type']),annot=True)
# Mostly both are loyal but if we'll check according to Classes there is a diiffere
```

Out[ ]:
```
<AxesSubplot:xlabel='Customer_type', ylabel='Gender'>
```



In [ ]:
```
plt.figure(figsize=(12,10))
sns.heatmap(pd.crosstab(df['Class'],df['Customer_type']),annot=True)
# we can see Business class travellers are more loyal whereas Eco_plus travellers c
```

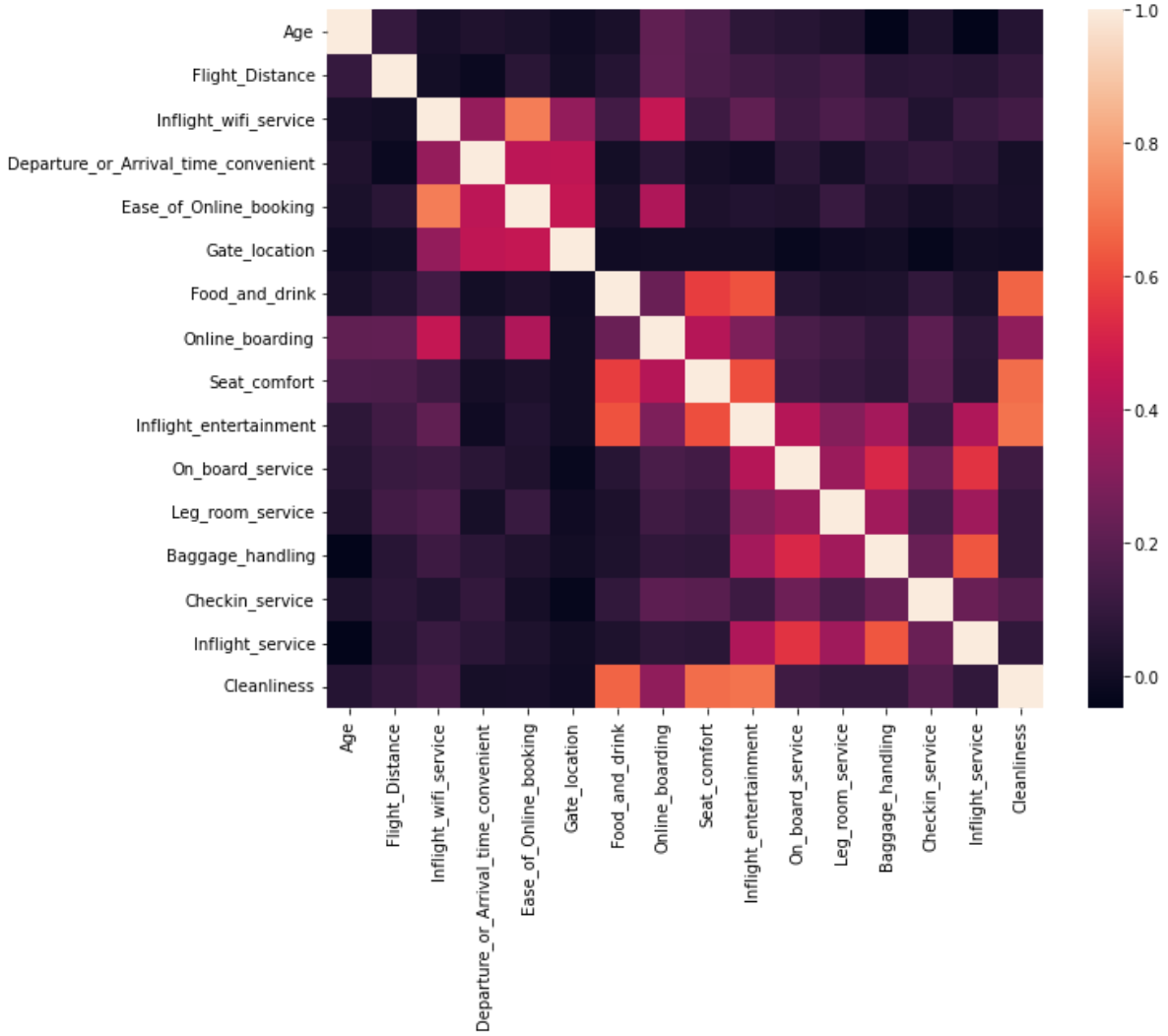
Out[ ]:
```
<AxesSubplot:xlabel='Customer_type', ylabel='Class'>
```



In [ ]:

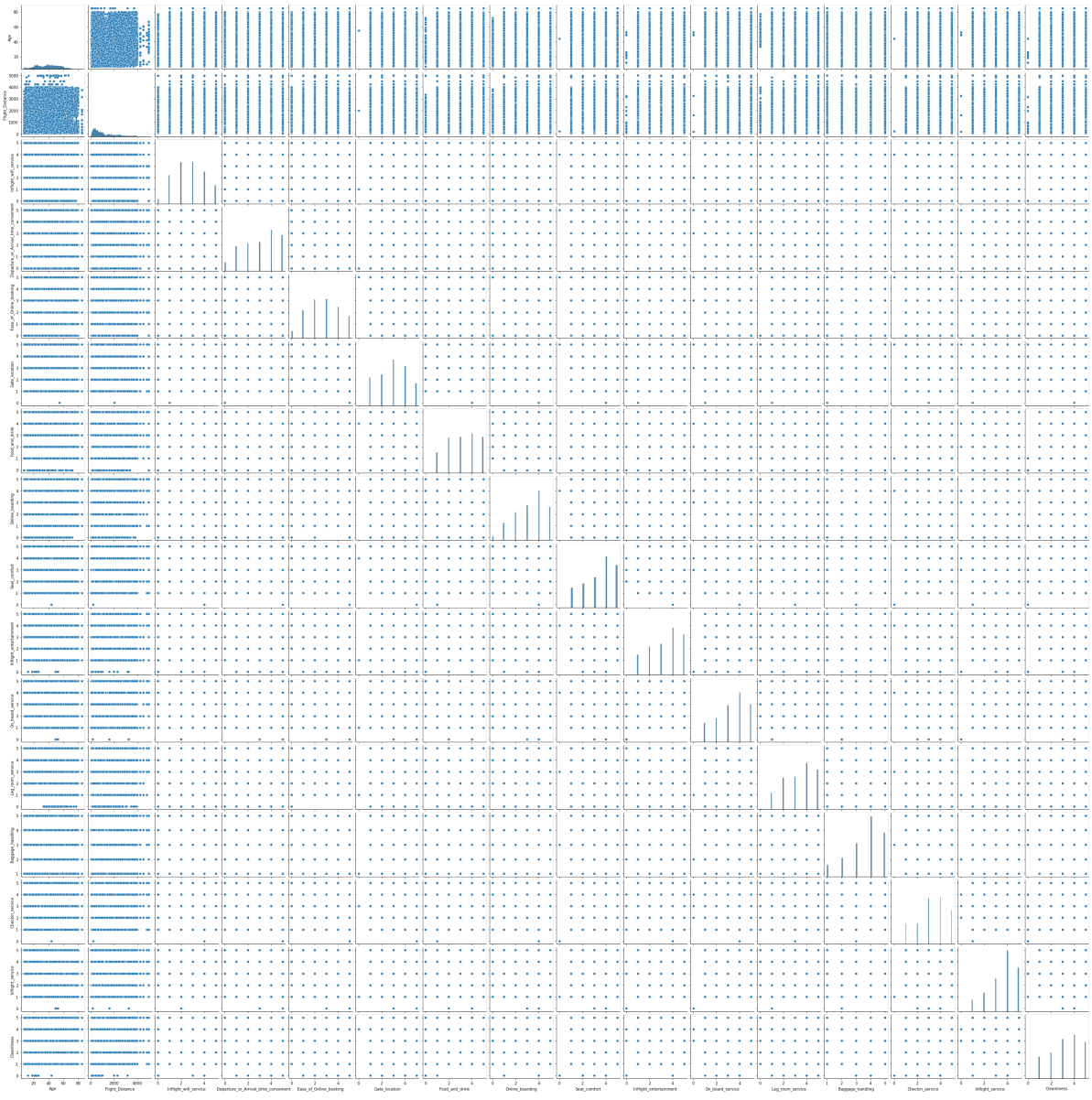
In [ ]:
```
plt.figure(figsize=(10,8))
sns.heatmap(df.corr())
# Not much correlation
```

Out[ ]:
```
<AxesSubplot:>
```



In [ ]:
```
sns.pairplot(df,diag_kind='hist')
```

Out[ ]:
```
<seaborn.axisgrid.PairGrid at 0x1bc6e3e7b20>
```



ENCODING

In [ ]:
```
le=LabelEncoder()
for i in df.columns:
    if df[i].dtype=='object':
        le_enc=le.fit_transform(df[i])
        df[i]=le_enc
```

In [ ]:
```
# Factors are highly correlated to a satisfied (or dissatisfied)

for i in df.columns:
    print(i,df[i].corr(df['satisfaction']))
```

```
Gender 0.012211274713222267
Customer_type -0.18763817141481096
Age 0.1371673049634227
Type_of_Travel -0.44900044984383297
Class -0.4493213256141194
Flight_Distance 0.29877978579988745
Inflight_wifi_service 0.28424504696514713
Departure_or_Arrival_time_convenient -0.051600617697144774
Ease_of_Online_booking 0.1717049784873053
Gate_location 0.0006820274562336509
Food_and_drink 0.2099362404463349
Online_boarding 0.5035573216470103
Seat_comfort 0.34945875813699057
Inflight_entertainment 0.3980594210993003
On_board_service 0.3223825214779897
Leg_room_service 0.3131308016998484
Baggage_handling 0.24774936539485146
Checkin_service 0.2361737428417545
Inflight_service 0.24474073867042667
Cleanliness 0.30519801179700284
satisfaction 1.0
```

Train and Test Split

In [ ]:
```
x=df.iloc[:,:-1]
y=df['satisfaction']
```

In [ ]:
```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.2,random_state=42)
```

In [ ]:
```
print(xtrain.shape)
print(ytrain.shape)
print(xtest.shape)
print(ytest.shape)
```

```
(83123, 20)
(83123,)
(20781, 20)
(20781,)
```

SCALING

In [ ]:
```
from sklearn.preprocessing import StandardScaler
scalar=StandardScaler()
xtrain_scal=scalar.fit_transform(xtrain)
xtest_scal=scalar.fit_transform(xtest)
```

MODELLING

LOGISTIC REGRESSION

In [ ]:
```
lgr=LogisticRegression()
lgr.fit(xtrain_scal,ytrain)
ypred_train_lgr=lgr.predict(xtrain_scal)
ypred_test_lgr=lgr.predict(xtest_scal)
```

ACCURACY CALCULATOR FUNCTION

In [ ]:
```
def acc_report(actual,predicted):
    acc_score=accuracy_score(actual,predicted)
    cm_matrix=confusion_matrix(actual,predicted)
    class_rep=classification_report(actual,predicted)
    print('the accuracy of tha model is ',acc_score)
    print(cm_matrix)
    print(class_rep)
```

In [ ]:
```
acc_report(ytrain,ypred_train_lgr)
```

```
the accuracy of tha model is  0.8740300518508716
[[42658  4508]
 [ 5963 29994]]
              precision    recall  f1-score   support

           0       0.88      0.90      0.89     47166
           1       0.87      0.83      0.85     35957

    accuracy                           0.87     83123
   macro avg       0.87      0.87      0.87     83123
weighted avg       0.87      0.87      0.87     83123
```

In [ ]:
```
acc_report(ytest,ypred_test_lgr)
```

```
the accuracy of tha model is  0.8771473942543669
[[10637  1076]
 [ 1477  7591]]
              precision    recall  f1-score   support

           0       0.88      0.91      0.89     11713
           1       0.88      0.84      0.86      9068

    accuracy                           0.88     20781
   macro avg       0.88      0.87      0.87     20781
weighted avg       0.88      0.88      0.88     20781
```

In [ ]:
```
lgr.coef_
```

Out[ ]:
```
array([[ 0.02561149, -0.79356974, -0.12024004, -1.29693698, -0.34686982,
        -0.00201003,  0.50918017, -0.17620982, -0.19941104,  0.03084588,
        -0.01807444,  0.83140574,  0.09093675,  0.07893738,  0.39917989,
         0.32255273,  0.15079262,  0.40430988,  0.16328931,  0.26863684]])
```

- ## The logistic regression model is not producing the desired outcome.

DECISION TREE

In [ ]:
```
dtr=DecisionTreeClassifier(max_depth=11,min_samples_split=8)
dtr.fit(xtrain,ytrain)
```

Out[ ]:
```
DecisionTreeClassifier(max_depth=11, min_samples_split=8)
```

In [ ]:
```
y_pred_train_dtr=dtr.predict(xtrain)
y_pred_test_dtr=dtr.predict(xtest)
```

In [ ]:
```
acc_report(ytrain,y_pred_train_dtr)
```

```
the accuracy of tha model is  0.9548380111401177
[[46102  1064]
 [ 2690 33267]]
              precision    recall  f1-score   support

           0       0.94      0.98      0.96     47166
           1       0.97      0.93      0.95     35957

    accuracy                           0.95     83123
   macro avg       0.96      0.95      0.95     83123
weighted avg       0.96      0.95      0.95     83123
```

In [ ]:
```
acc_report(ytest,y_pred_test_dtr)
```

```
the accuracy of tha model is  0.9495211972474857
[[11390   323]
 [  726  8342]]
              precision    recall  f1-score   support

           0       0.94      0.97      0.96     11713
           1       0.96      0.92      0.94      9068

    accuracy                           0.95     20781
   macro avg       0.95      0.95      0.95     20781
weighted avg       0.95      0.95      0.95     20781
```

- ## The decision tree model is achieving higher Accuracy than the logistic regression model.

RANDOM FOREST

In [ ]:
```
rf=RandomForestClassifier(n_estimators=20,max_depth=11,verbose=1)
rf.fit(xtrain,ytrain)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:    1.1s finished
```

Out[ ]:
```
RandomForestClassifier(max_depth=11, n_estimators=20, verbose=1)
```

In [ ]:
```
y_pred_train_rf=rf.predict(xtrain)
y_pred_test_rf=rf.predict(xtest)
```

```
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:    0.1s finished
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done  20 out of  20 | elapsed:    0.0s finished
```

In [ ]:
```
acc_report(ytrain,y_pred_train_rf)
```

```
the accuracy of tha model is  0.9578095112062847
[[45806  1360]
 [ 2147 33810]]
              precision    recall  f1-score   support

           0       0.96      0.97      0.96     47166
           1       0.96      0.94      0.95     35957

    accuracy                           0.96     83123
   macro avg       0.96      0.96      0.96     83123
weighted avg       0.96      0.96      0.96     83123
```

In [ ]:
```
acc_report(ytest,y_pred_test_rf)
```

```
the accuracy of tha model is  0.9519753621096194
[[11315   398]
 [  600  8468]]
              precision    recall  f1-score   support

           0       0.95      0.97      0.96     11713
           1       0.96      0.93      0.94      9068

    accuracy                           0.95     20781
   macro avg       0.95      0.95      0.95     20781
weighted avg       0.95      0.95      0.95     20781
```

- ## Random forest also producing good Accuracy score

IMPLEMENTATION OF ADABOOST

In [ ]:
```
ada=AdaBoostClassifier(n_estimators=200,learning_rate=0.9623)
ada.fit(xtrain,ytrain)
```

Out[ ]:
```
AdaBoostClassifier(learning_rate=0.9623, n_estimators=200)
```

In [ ]:
```
y_pred_train_ada=ada.predict(xtrain)
y_pred_test_ada=ada.predict(xtest)
```

In [ ]:
```
acc_report(ytrain,y_pred_train_ada)
```

```
the accuracy of tha model is  0.929814852688185
[[44461  2705]
 [ 3129 32828]]
              precision    recall  f1-score   support

           0       0.93      0.94      0.94     47166
           1       0.92      0.91      0.92     35957

    accuracy                           0.93     83123
   macro avg       0.93      0.93      0.93     83123
weighted avg       0.93      0.93      0.93     83123
```

In [ ]:
```
acc_report(ytest,y_pred_test_ada)
```

```
the accuracy of tha model is  0.9301284827486647
[[11054   659]
 [  793  8275]]
              precision    recall  f1-score   support

           0       0.93      0.94      0.94     11713
           1       0.93      0.91      0.92      9068

    accuracy                           0.93     20781
   macro avg       0.93      0.93      0.93     20781
weighted avg       0.93      0.93      0.93     20781
```

- ## ADABOOST also producing good Accuracy score

GRADIENT BOOSTING ALGORITHM

In [ ]:
```
gb=GradientBoostingClassifier(learning_rate=0.9445)
gb.fit(xtrain,ytrain)
```

Out[ ]:
```
GradientBoostingClassifier(learning_rate=0.9445)
```

In [ ]:
```
y_pred_train_gb=gb.predict(xtrain)
y_pred_test_gb=gb.predict(xtest)
```

In [ ]:
```
acc_report(ytrain,y_pred_train_gb)
```

```
the accuracy of tha model is  0.9634156611286888
[[46107  1059]
 [ 1982 33975]]
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     47166
           1       0.97      0.94      0.96     35957

    accuracy                           0.96     83123
   macro avg       0.96      0.96      0.96     83123
weighted avg       0.96      0.96      0.96     83123
```

In [ ]:
```
acc_report(ytest,y_pred_test_gb)
```

```
the accuracy of tha model is  0.9573167797507338
[[11372   341]
 [  546  8522]]
              precision    recall  f1-score   support

           0       0.95      0.97      0.96     11713
           1       0.96      0.94      0.95      9068

    accuracy                           0.96     20781
   macro avg       0.96      0.96      0.96     20781
weighted avg       0.96      0.96      0.96     20781
```

- ## GRADIENT BOOSTING also producing good Accuracy score

XGBOOST

In [ ]:
```
xgb=XGBClassifier()
xgb.fit(xtrain,ytrain)
```

Out[ ]:
```
XGBClassifier(base_score=None, booster=None, callbacks=None,
              colsample_bylevel=None, colsample_bynode=None,
              colsample_bytree=None, early_stopping_rounds=None,
              enable_categorical=False, eval_metric=None, feature_types=None,
              gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
              interaction_constraints=None, learning_rate=None, max_bin=None,
              max_cat_threshold=None, max_cat_to_onehot=None,
              max_delta_step=None, max_depth=None, max_leaves=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              predictor=None, random_state=None, ...)
```

In [ ]:
```
y_pred_xtrain_xgb=xgb.predict(xtrain)
y_pred_xtest_xgb=xgb.predict(xtest)
```

In [ ]:
```
acc_report(ytrain,y_pred_xtrain_xgb)
```

```
the accuracy of tha model is  0.9761558172828219
[[46638   528]
 [ 1454 34503]]
              precision    recall  f1-score   support

           0       0.97      0.99      0.98     47166
           1       0.98      0.96      0.97     35957

    accuracy                           0.98     83123
   macro avg       0.98      0.97      0.98     83123
weighted avg       0.98      0.98      0.98     83123
```

In [ ]:
```
acc_report(ytest,y_pred_xtest_xgb)
```

```
the accuracy of tha model is  0.9620807468360522
[[11463   250]
 [  538  8530]]
              precision    recall  f1-score   support

           0       0.96      0.98      0.97     11713
           1       0.97      0.94      0.96      9068

    accuracy                           0.96     20781
   macro avg       0.96      0.96      0.96     20781
weighted avg       0.96      0.96      0.96     20781
```

- ## XGBoost model is achieving good accuracy score but there is some variance in accuracy.

KNN

In [ ]:
```
np.sqrt(100000)
```

Out[ ]:
```
316.22776601683796
```

In [ ]:
```
knn=KNeighborsClassifier(n_neighbors=5)
knn.fit(xtrain,ytrain)
```

Out[ ]:
```
KNeighborsClassifier()
```

In [ ]:
```
y_pred_xtrain_knn=knn.predict(xtrain)
y_pred_xtest_knn=knn.predict(xtest)
```

In [ ]:
```
acc_report(ytrain,y_pred_xtrain_knn)
```

```
the accuracy of tha model is  0.8688449646908798
[[42935  4231]
 [ 6671 29286]]
              precision    recall  f1-score   support

           0       0.87      0.91      0.89     47166
           1       0.87      0.81      0.84     35957

    accuracy                           0.87     83123
   macro avg       0.87      0.86      0.87     83123
weighted avg       0.87      0.87      0.87     83123
```

In [ ]:
```
acc_report(ytest,y_pred_xtest_knn)
```

```
the accuracy of tha model is  0.8093450748279678
[[10074  1639]
 [ 2323  6745]]
              precision    recall  f1-score   support

           0       0.81      0.86      0.84     11713
           1       0.80      0.74      0.77      9068

    accuracy                           0.81     20781
   macro avg       0.81      0.80      0.80     20781
weighted avg       0.81      0.81      0.81     20781
```

- ## KNN is not performing well it's achieving high bias and high variance

SVC

In [ ]:
```
svc=SVC()
svc.fit(xtrain,ytrain)
```

Out[ ]:
```
SVC()
```

In [ ]:
```
y_pred_train_svc=svc.predict(xtrain)
y_pred_test_svc=svc.predict(xtest)
```

In [ ]:
```
acc_report(ytrain,y_pred_train_svc)
```

```
the accuracy of tha model is  0.6687920310864622
[[39440  7726]
 [19805 16152]]
              precision    recall  f1-score   support

           0       0.67      0.84      0.74     47166
           1       0.68      0.45      0.54     35957

    accuracy                           0.67     83123
   macro avg       0.67      0.64      0.64     83123
weighted avg       0.67      0.67      0.65     83123
```

In [ ]:
```
acc_report(ytest,y_pred_test_svc)
```

```
the accuracy of tha model is  0.6677734468986093
[[9799 1914]
 [4990 4078]]
              precision    recall  f1-score   support

           0       0.66      0.84      0.74     11713
           1       0.68      0.45      0.54      9068

    accuracy                           0.67     20781
   macro avg       0.67      0.64      0.64     20781
weighted avg       0.67      0.67      0.65     20781
```

- ## SVC is not performing well it's achieving bad Accuracy Score

NAIVE BAYES

In [ ]:
```
from sklearn.naive_bayes import GaussianNB
```

In [ ]:
```
gnb=GaussianNB()
gnb.fit(xtrain,ytrain)
```

Out[ ]:
```
GaussianNB()
```

In [ ]:
```
y_pred_train_gnb=gnb.predict(xtrain)
y_pred_test_gnb=gnb.predict(xtest)
```

In [ ]:
```
acc_report(ytrain,y_pred_train_gnb)
```

```
the accuracy of tha model is  0.8659456468125549
[[42587  4579]
 [ 6564 29393]]
              precision    recall  f1-score   support

           0       0.87      0.90      0.88     47166
           1       0.87      0.82      0.84     35957

    accuracy                           0.87     83123
   macro avg       0.87      0.86      0.86     83123
weighted avg       0.87      0.87      0.87     83123
```

In [ ]:
```
acc_report(ytest,y_pred_test_gnb)
```

```
the accuracy of tha model is  0.8679563062412781
[[10584  1129]
 [ 1615  7453]]
              precision    recall  f1-score   support

           0       0.87      0.90      0.89     11713
           1       0.87      0.82      0.84      9068

    accuracy                           0.87     20781
   macro avg       0.87      0.86      0.86     20781
weighted avg       0.87      0.87      0.87     20781
```

- ## Naive Bayes is not producing desired outcome

Depolyment of the model

In [ ]:
```
import pickle
import gradio as gr
```

In [ ]:
```
with open('gb_model.pkl','wb') as f:
    pickle.dump(gb,f)
```

In [ ]:
```
def make_prediction(Gender,Customer_type,Age,Type_of_Travel,Class,
       Flight_Distance,Inflight_wifi_service,
       Departure_or_Arrival_time_convenient,Ease_of_Online_booking,
       Gate_location,Food_and_drink,Online_boarding,Seat_comfort,
       Inflight_entertainment, On_board_service,Leg_room_service,
       Baggage_handling,Checkin_service,Inflight_service,
       Cleanliness):
    with open('gb_model.pkl','rb')as f:
        clf=pickle.load(f)
        pred=clf.predict([[Gender,Customer_type,Age,Type_of_Travel,Class,
       Flight_Distance,Inflight_wifi_service,
       Departure_or_Arrival_time_convenient,Ease_of_Online_booking,
       Gate_location,Food_and_drink,Online_boarding,Seat_comfort,
       Inflight_entertainment, On_board_service,Leg_room_service,
       Baggage_handling,Checkin_service,Inflight_service,
       Cleanliness]])
    if pred==0:
        return('Dissatisfied')
    else:
        return('satisfied')
```

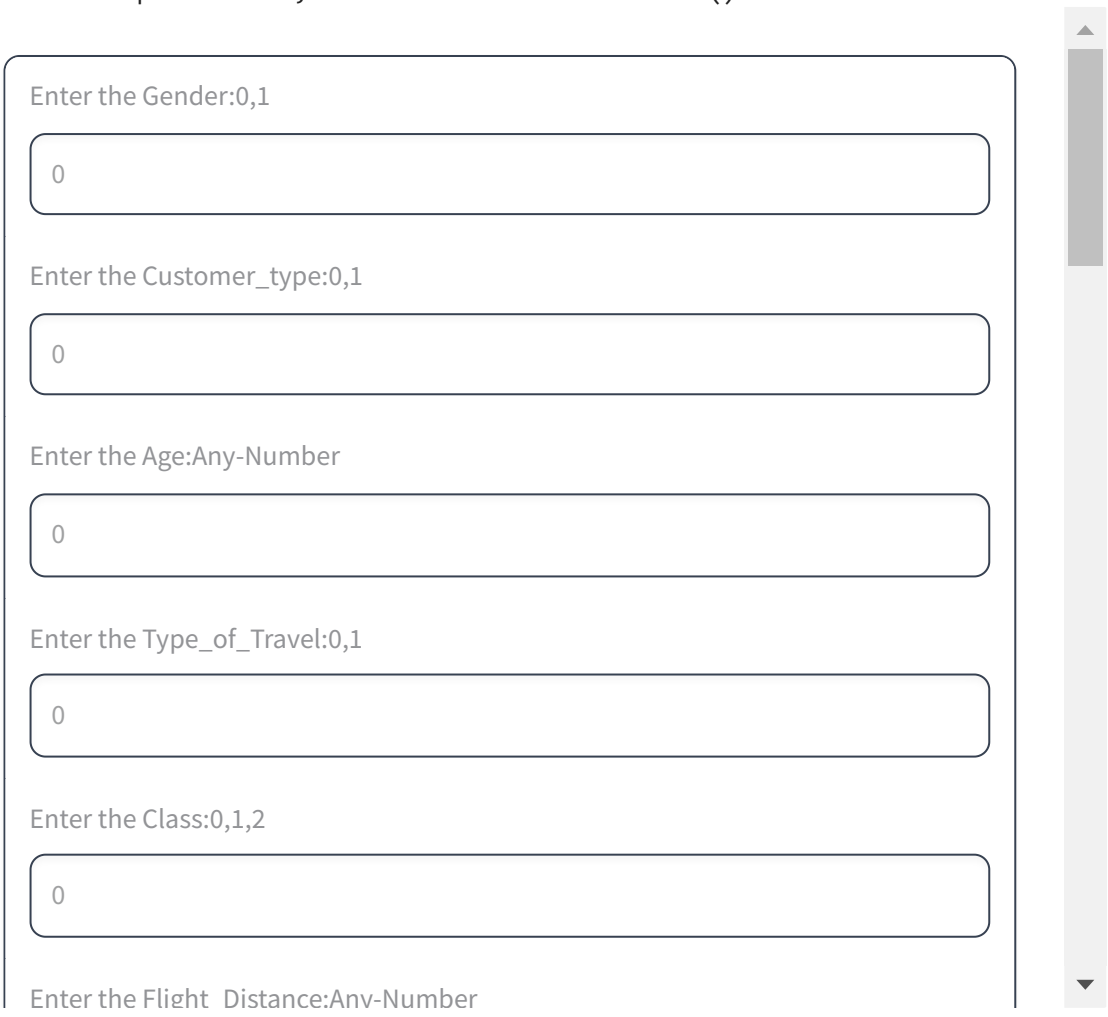
In [ ]:
```
Gender=gr.Number(label='Enter the Gender:0,1')
Customer_type=gr.Number(label='Enter the Customer_type:0,1')
Age=gr.Number(label='Enter the Age:Any-Number')
Type_of_Travel=gr.Number(label='Enter the Type_of_Travel:0,1')
Class=gr.Number(label='Enter the Class:0,1,2')
Flight_Distance=gr.Number(label='Enter the Flight_Distance:Any-Number')
Inflight_wifi_service=gr.Number(label='Enter the satisfaction for Inflight_wifi_se
Departure_or_Arrival_time_convenient=gr.Number(label='Enter the satisfaction for D
Ease_of_Online_booking=gr.Number(label='Enter the Ease_of_Online_booking:0,1,2,3,4
Gate_location=gr.Number(label='Enter the satisfaction for Gate_location:0,1,2,3,4,
Food_and_drink=gr.Number(label='Enter the satisfaction for Food_and_drink:0,1,2,3,
Online_boarding=gr.Number(label='Enter the satisfaction for Online_boarding:0,1,2,
Seat_comfort=gr.Number(label='Enter the satisfaction for Seat_comfort:0,1,2,3,4,5'
Inflight_entertainment=gr.Number(label='Enter the satisfaction for Inflight_enterta
On_board_service=gr.Number(label='Enter the satisfaction for On_board_service:0,1,
Inflight_entertainment=gr.Number(label='Enter the satisfaction for Inflight_enterta
```

```
Leg_room_service=gr.Number(label='Enter the satisfaction for Leg_room_service:0,1,2
Baggage_handling=gr.Number(label='Enter the satisfaction for Baggage_handling:0,1,2
Checkin_service=gr.Number(label='Enter the satisfaction for Checkin_service:0,1,2,3
Inflight_service=gr.Number(label='Enter the satisfaction for Inflight_service:0,1,2
Cleanliness=gr.Number(label='Enter the satisfaction for Cleanliness:0,1,2,3,4,5')

# OUTPUT
output=gr.Textbox()
```

In [ ]:

```
app=gr.Interface(fn=make_prediction,inputs=[Gender,Customer_type,Age,Type_of_Travel
       Flight_Distance,Inflight_wifi_service,
       Departure_or_Arrival_time_convenient,Ease_of_Online_booking,
       Gate_location,Food_and_drink,Online_boarding,Seat_comfort,
       Inflight_entertainment, On_board_service,Leg_room_service,
       Baggage_handling,Checkin_service,Inflight_service,
       Cleanliness],outputs=output)
app.launch(debug=True)
```

```
Running on local URL:  http://127.0.0.1:7860

To create a public link, set `share=True` in `launch()`.
```



CONCLUSION

Low performance: Logistic regression(87,87), KNN(86,80), SVC(66,66), Naive Bayes(86,86)

Good performance: Decision Tree(95,94), Random Forest(95,95), Adaboosting(92,93), Gradient Boosting(96,95),

XGboost(97,96)

DECISION TREE, XG BOOST, RANDOM FOREST & Gradient Boosting Performing Good