COMS 4995 AML Group Project

Topic: Airline Passenger Satisfaction

Group 2 Members:

Anqi Xue, ax2170
Tanisha Aggrawal, ta2709
Vishal Bhardwaj, vb2573

The Dataset

Description

This data set contains details of an airline passenger satisfaction survey and the target variable is a binary variable reflecting the airline satisfaction level (satisfied, neutral or dissatisfied) of a passenger.

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import time
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder, OrdinalEncoder
from sklearn.compose import make_column_transformer
from sklearn.metrics import precision_score, recall_score, f1_score, accuracy_score, r
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, classification_r
from sklearn.model_selection import GridSearchCV

%matplotlib inline
```

Part 1: Data Preprocessing

1.1: Load the datasets

In [2]:
```
dev_df = pd.read_csv('train_and_validation.csv', index_col = 0)
test_df = pd.read_csv('test.csv', index_col = 0)
```

In [3]:
```
dev_df
```

Out[3]:

	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	I b
0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	
1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2	
2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2	
3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5	
4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3	
...	...	...	...	...	...	...	...	...	...	
103899	94171	Female	disloyal Customer	23	Business travel	Eco	192	2	1	
103900	73097	Male	Loyal Customer	49	Business travel	Business	2347	4	4	
103901	68825	Male	disloyal Customer	30	Business travel	Business	1995	1	1	
103902	54173	Female	disloyal Customer	22	Business travel	Eco	1000	1	1	
103903	62567	Male	Loyal Customer	27	Business travel	Business	1723	1	3	

103904 rows × 24 columns

In [4]:
```
test_df
```

Out[4]:

	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Eas Or boo
0	19556	Female	Loyal Customer	52	Business travel	Eco	160	5	4	
1	90035	Female	Loyal Customer	36	Business travel	Business	2863	1	1	
2	12360	Male	disloyal Customer	20	Business travel	Eco	192	2	0	
3	77959	Male	Loyal Customer	44	Business travel	Business	3377	0	0	
4	36875	Female	Loyal Customer	49	Business travel	Eco	1182	2	3	
...	...	...	...	...	...	...	...	...	...	
25971	78463	Male	disloyal Customer	34	Business travel	Business	526	3	3	
25972	71167	Male	Loyal Customer	23	Business travel	Business	646	4	4	
25973	37675	Female	Loyal Customer	17	Personal Travel	Eco	828	2	5	
25974	90086	Male	Loyal Customer	14	Business travel	Business	1127	3	3	
25975	34799	Female	Loyal Customer	42	Personal Travel	Eco	264	2	5	

25976 rows × 24 columns

1.2: Missing values analysis

We remove the information about the passenger (the whole row) if part of their data is missing.

In [5]:
```
# Checking null values in dev dataset
np.any(dev_df.isnull() == True)
```

Out[5]:
```
True
```

In [6]:
```
#drop the whole row with null value(s) in dev set
dev_df = dev_df.dropna(axis = 0, how = 'any')
dev_df
```

Out[6]:

	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	I b
0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	
1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2	
2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2	
3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5	
4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3	
...	...	...	...	...	...	...	...	...	...	
103899	94171	Female	disloyal Customer	23	Business travel	Eco	192	2	1	
103900	73097	Male	Loyal Customer	49	Business travel	Business	2347	4	4	
103901	68825	Male	disloyal Customer	30	Business travel	Business	1995	1	1	
103902	54173	Female	disloyal Customer	22	Business travel	Eco	1000	1	1	
103903	62567	Male	Loyal Customer	27	Business travel	Business	1723	1	3	

103594 rows × 24 columns

310 rows have been removed from the dev set, that is, 310 samples had at least one null value.

In [7]:
```
# Checking null values in test dataset
np.any(test_df.isnull() == True)
```

Out[7]:
```
True
```

In [8]:
```
#drop the whole row with null value(s) in test set
test_df = test_df.dropna(axis = 0, how = 'any')
test_df
```

Out[8]:

	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Eas Or boo
0	19556	Female	Loyal Customer	52	Business travel	Eco	160	5	4	
1	90035	Female	Loyal Customer	36	Business travel	Business	2863	1	1	
2	12360	Male	disloyal Customer	20	Business travel	Eco	192	2	0	
3	77959	Male	Loyal Customer	44	Business travel	Business	3377	0	0	
4	36875	Female	Loyal Customer	49	Business travel	Eco	1182	2	3	
...	...	...	...	...	...	...	...	...	...	
25971	78463	Male	disloyal Customer	34	Business travel	Business	526	3	3	
25972	71167	Male	Loyal Customer	23	Business travel	Business	646	4	4	
25973	37675	Female	Loyal Customer	17	Personal Travel	Eco	828	2	5	
25974	90086	Male	Loyal Customer	14	Business travel	Business	1127	3	3	
25975	34799	Female	Loyal Customer	42	Personal Travel	Eco	264	2	5	

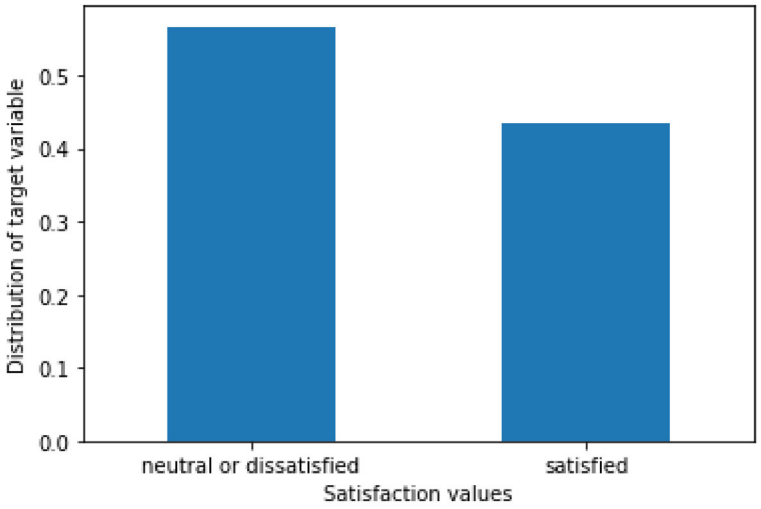
25893 rows × 24 columns

83 rows have been removed from the test set, that is, 83 samples had at least one null value.

1.3: Explain the distribution of the target variable and the dataset

In [9]:
```
dev_df['satisfaction'].value_counts(normalize = True).plot(kind='bar', rot=0)
plt.xlabel('Satisfaction values')
plt.ylabel('Distribution of target variable')
```

Out[9]:
```
Text(0, 0.5, 'Distribution of target variable')
```



1.4: Drop the columns

We drop the id column because this information is irrelevant to the target output

In [10]:
```
X_dev = dev_df.drop(columns = ['id', 'satisfaction'])
X_test = test_df.drop(columns = ['id', 'satisfaction'])
```

In [11]:
```
y_dev = dev_df['satisfaction']
y_test = test_df['satisfaction']
```

1.5: Exploratory Data Analysis

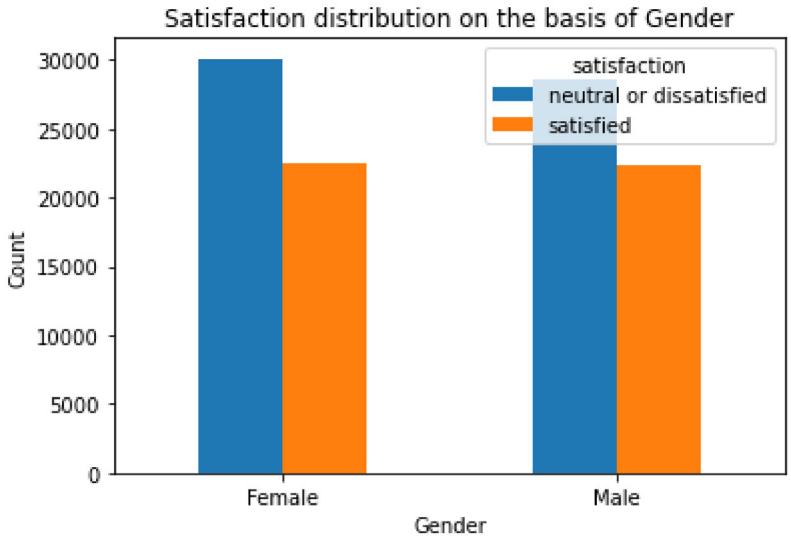
In [12]:
```
#Description of the columns in the Dataframe
X_dev.info()
```

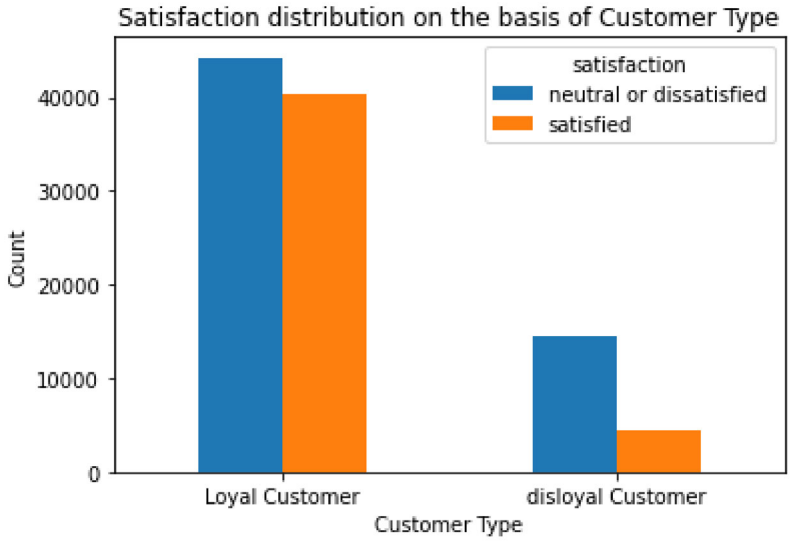
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 103594 entries, 0 to 103903
Data columns (total 22 columns):
 #   Column                            Non-Null Count   Dtype  
---  ------                            --------------   -----  
 0   Gender                            103594 non-null  object 
 1   Customer Type                     103594 non-null  object 
 2   Age                               103594 non-null  int64  
 3   Type of Travel                    103594 non-null  object 
 4   Class                             103594 non-null  object 
 5   Flight Distance                   103594 non-null  int64  
 6   Inflight wifi service             103594 non-null  int64  
 7   Departure/Arrival time convenient  103594 non-null  int64  
 8   Ease of Online booking            103594 non-null  int64  
 9   Gate location                     103594 non-null  int64  
 10  Food and drink                    103594 non-null  int64  
 11  Online boarding                   103594 non-null  int64  
 12  Seat comfort                      103594 non-null  int64  
 13  Inflight entertainment            103594 non-null  int64  
 14  On-board service                  103594 non-null  int64  
 15  Leg room service                  103594 non-null  int64  
 16  Baggage handling                  103594 non-null  int64  
 17  Checkin service                   103594 non-null  int64  
 18  Inflight service                  103594 non-null  int64  
 19  Cleanliness                       103594 non-null  int64  
 20  Departure Delay in Minutes        103594 non-null  int64  
 21  Arrival Delay in Minutes          103594 non-null  float64
dtypes: float64(1), int64(17), object(4)
memory usage: 18.2+ MB
```

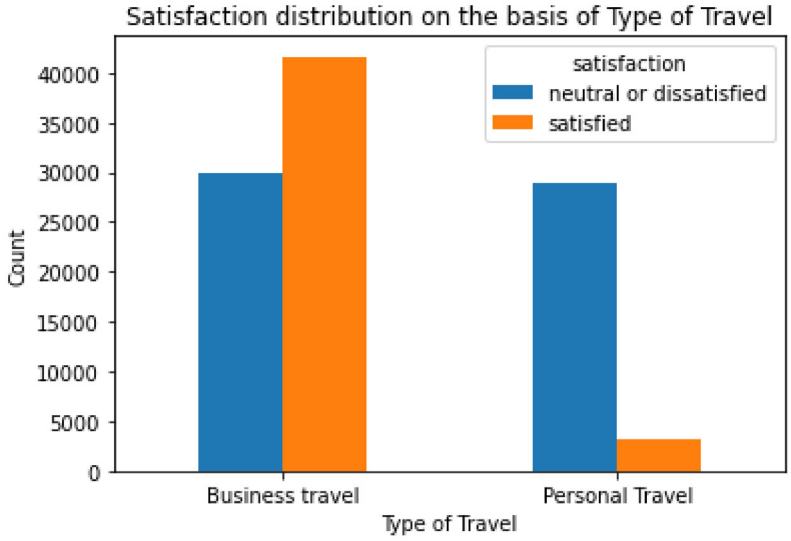
In [13]:
```
#Segragating features into continuous-value features, numerical rating features, and c
num_feats = list(X_dev.select_dtypes(['number']).columns)
continuous_feats = ['Age', 'Flight Distance', 'Departure Delay in Minutes', 'Arrival D
rating_feats = [feat for feat in num_feats if feat not in continuous_feats]
cat_feats = list(X_dev.select_dtypes(['object']).columns)
```

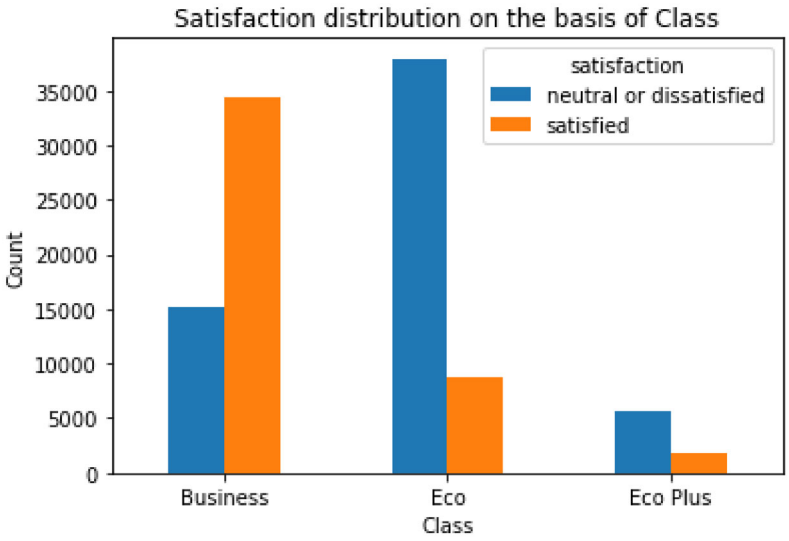
In [14]:
```
#Categorical Features Analysis
for feature in cat_feats:
  print(dev_df.groupby([feature, 'satisfaction']).size().unstack())
  dev_df.groupby([feature, 'satisfaction']).size().unstack().plot(kind='bar', rot=0)
  plt.ylabel('Count')
  plt.title('Satisfaction distribution on the basis of '+ feature)
```

```
satisfaction  neutral or dissatisfied  satisfied
Gender                                          
Female                          30107      22469
Male                            28590      22428
satisfaction       neutral or dissatisfied  satisfied
Customer Type                                       
Loyal Customer                       44249      40413
disloyal Customer                    14448       4484
satisfaction     neutral or dissatisfied  satisfied
Type of Travel                                      
Business travel                    29831      41634
Personal Travel                    28866       3263
satisfaction  neutral or dissatisfied  satisfied
Class                                           
Business                        15143      34390
Eco                             37922       8671
Eco Plus                         5632       1836
```


Satisfaction distribution on the basis of Gender
satisfaction
neutral or dissatisfied
satisfied
Count
30000
25000
20000
15000
10000
5000
0
Female
Male
Gender

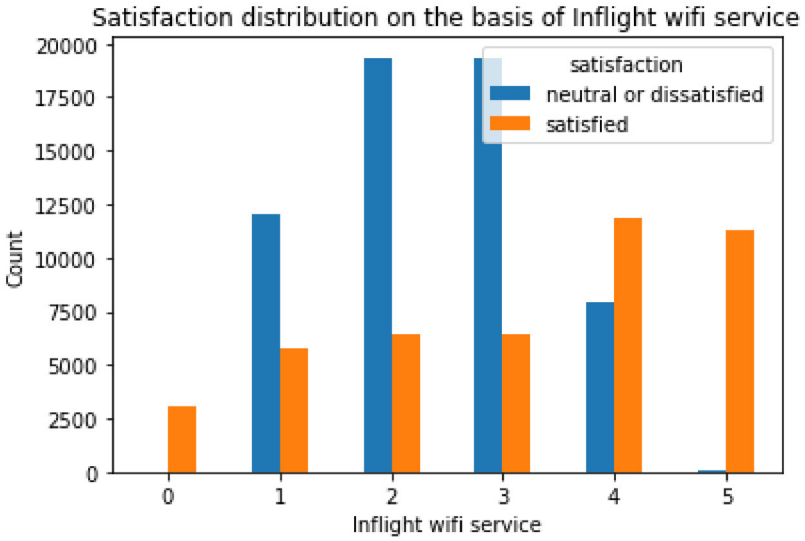

Satisfaction distribution on the basis of Customer Type
satisfaction
neutral or dissatisfied
satisfied
Count
40000
30000
20000
10000
0
Loyal Customer
disloyal Customer
Customer Type

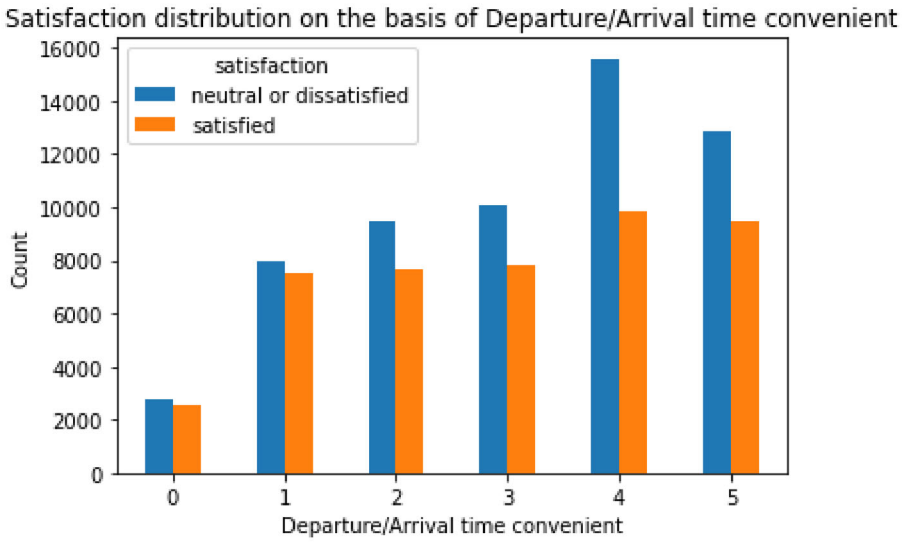

Satisfaction distribution on the basis of Type of Travel
satisfaction
neutral or dissatisfied
satisfied
Count
40000
35000
30000
25000
20000
15000
10000
5000
0
Business travel
Personal Travel
Type of Travel

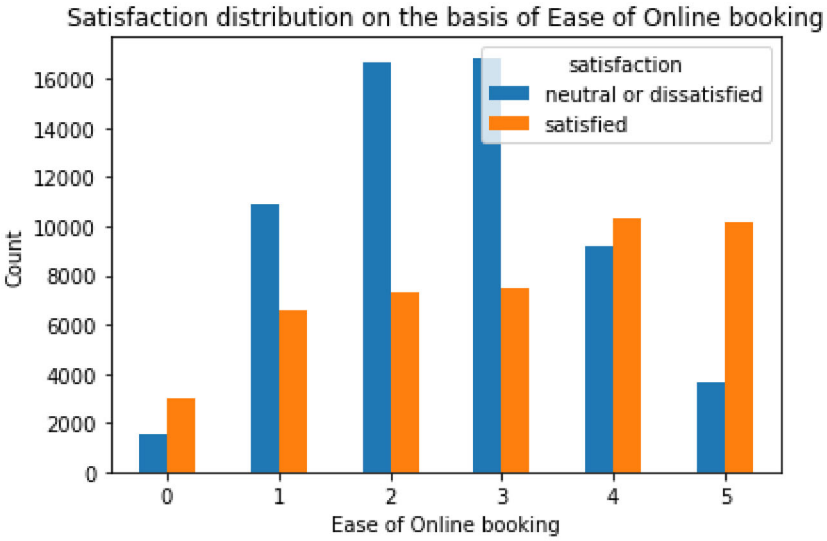


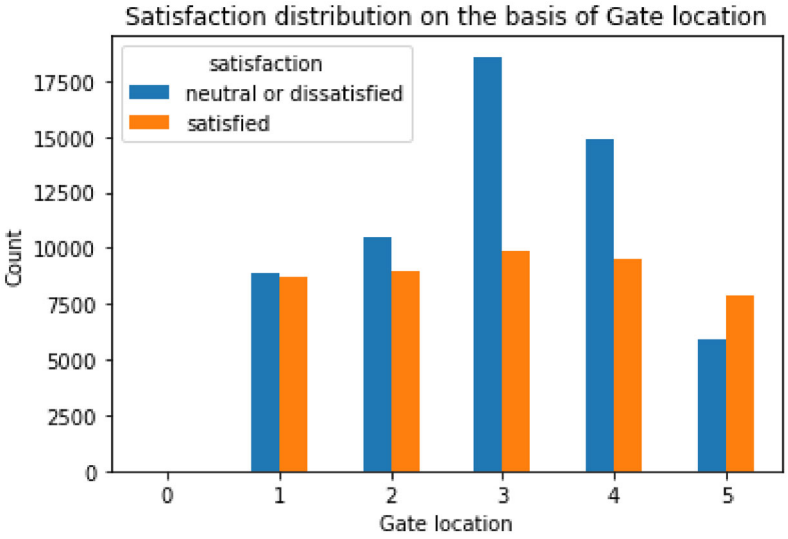
- Consistent distribution of target variable in each category for the features Gender and Customer Type.
- Similar number of 'Males' and 'Females' in the Dev set.
- Almost 4.5 times more loyal customers than disloyal customers. *Number of dissatisfied customers is similar for customers travelling for business and personal purposes.
- 58% customers travelling for business purposes are satisfied whereas only 10% of customers travelling for personal purposes are satisfied.
- Only 2940 more customers travelled in Business than Eco.
- 69.5% customers travelling in Business class were satisfied whereas only 18.6% customers travelling in Eco class were satisfied.
- Only 7.2% customers travelled in Eco Plus.

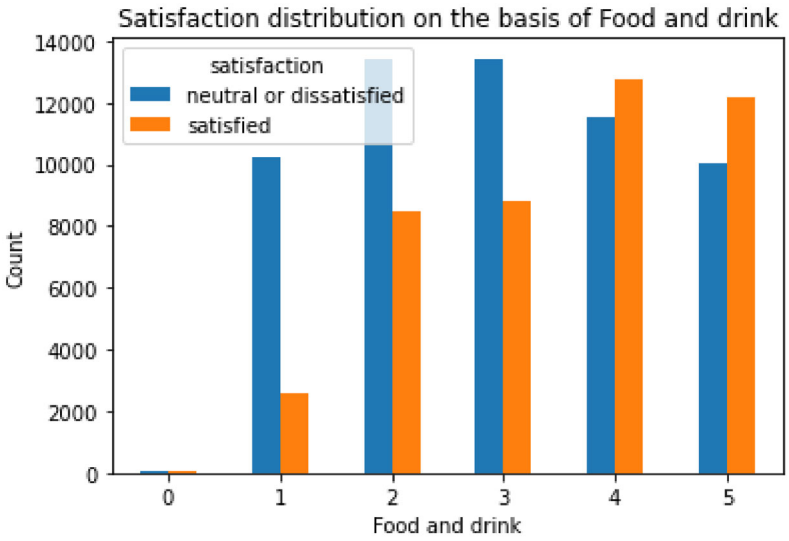
In [15]:
```
#Numerical Ratings based features Analysis
for feature in rating_feats:
  dev_df.groupby([feature, 'satisfaction']).size().unstack().plot(kind='bar', rot=0)
  plt.ylabel('Count')
  plt.title('Satisfaction distribution on the basis of '+ feature)
```

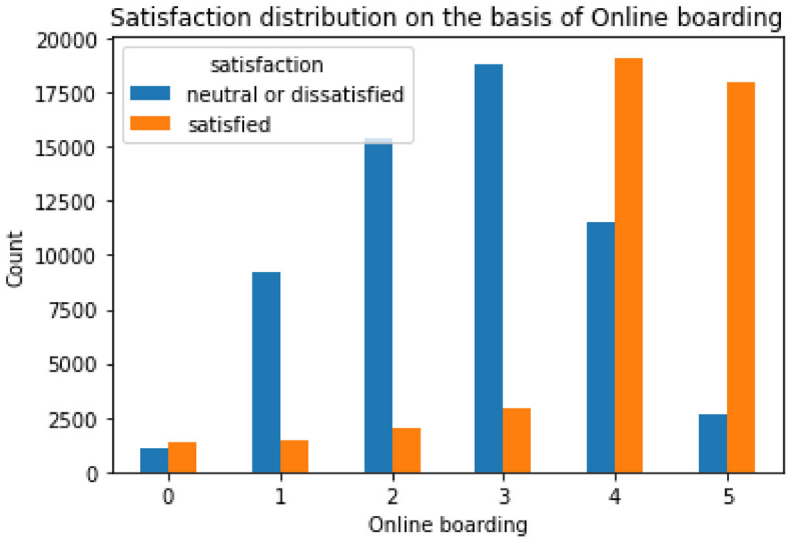


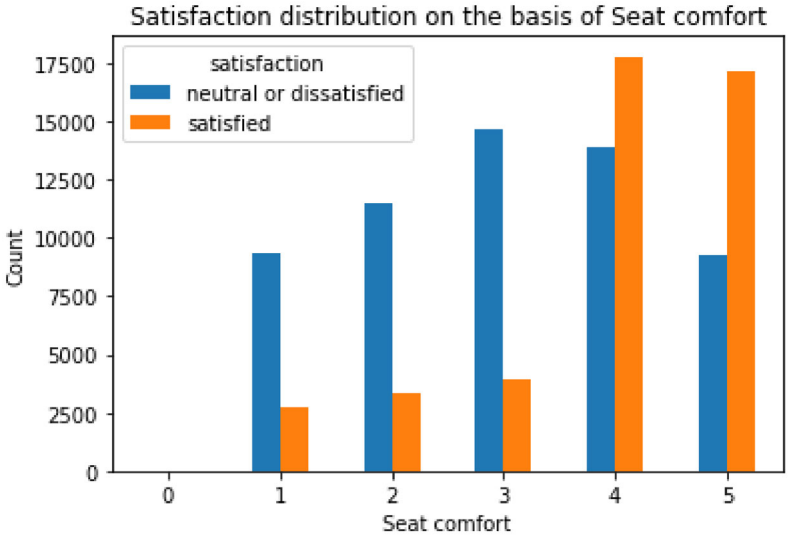

Satisfaction distribution on the basis of Departure/Arrival time convenient
satisfaction
neutral or dissatisfied
satisfied
Count
Departure/Arrival time convenient

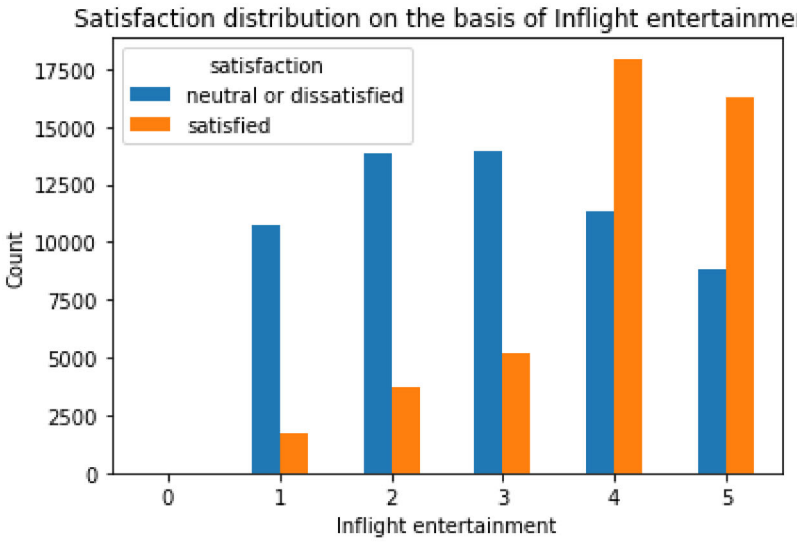

Satisfaction distribution on the basis of Ease of Online booking
satisfaction
neutral or dissatisfied
satisfied
Count
Ease of Online booking

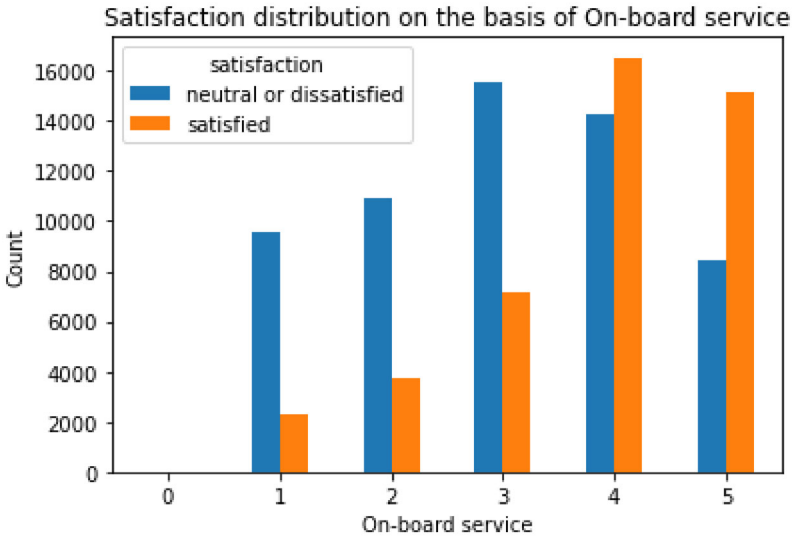

Satisfaction distribution on the basis of Gate location
satisfaction
neutral or dissatisfied
satisfied
Count
Gate location

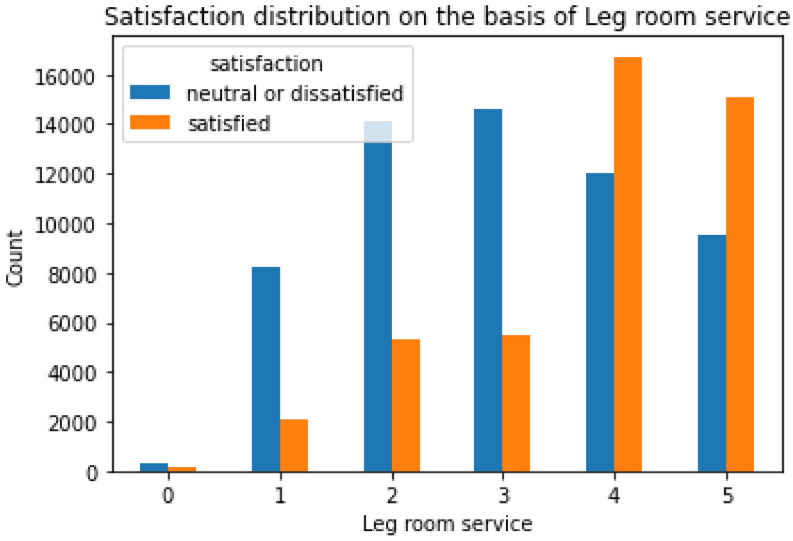

Satisfaction distribution on the basis of Food and drink
satisfaction
neutral or dissatisfied
satisfied
Count
Food and drink
0
2000
4000
6000
8000
10000
12000
14000
0
1
2
3
4
5

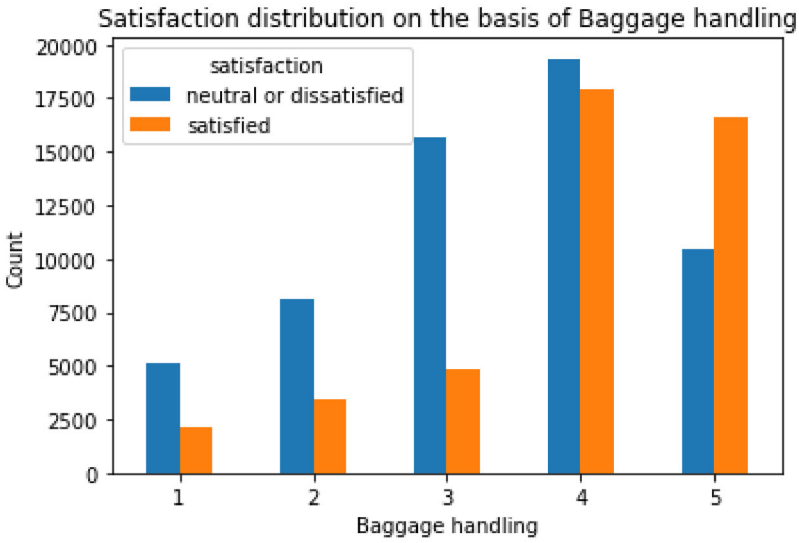

Satisfaction distribution on the basis of Online boarding
satisfaction
neutral or dissatisfied
satisfied
Count
Online boarding
0
2500
5000
7500
10000
12500
15000
17500
20000
0
1
2
3
4
5

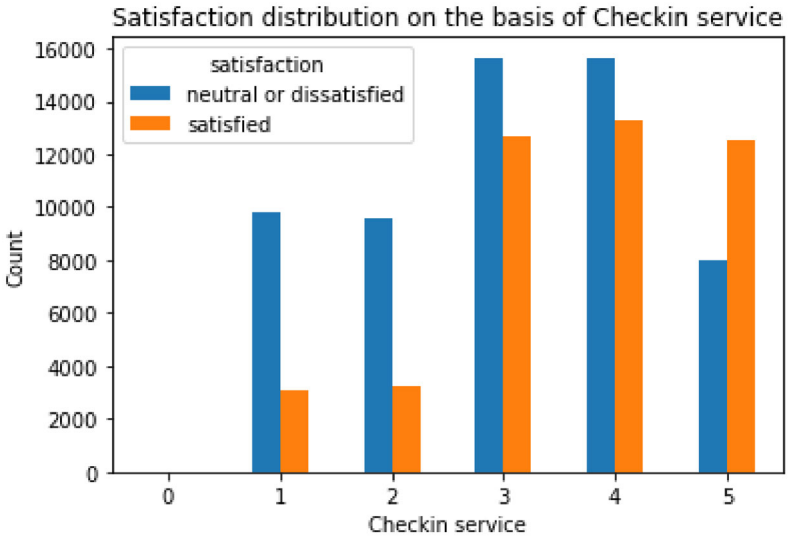

Satisfaction distribution on the basis of Seat comfort
satisfaction
neutral or dissatisfied
satisfied
Count
Seat comfort
0
2500
5000
7500
10000
12500
15000
17500
0
1
2
3
4
5

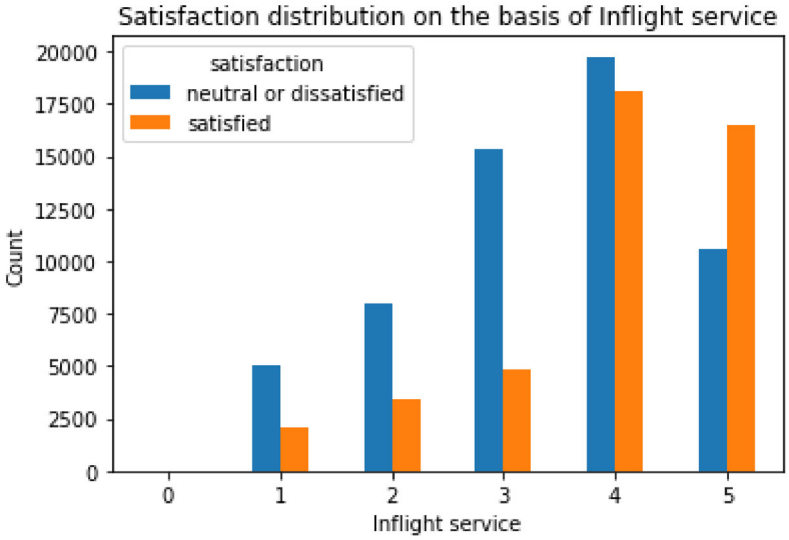

Satisfaction distribution on the basis of Inflight entertainment
satisfaction
neutral or dissatisfied
satisfied
Count
Inflight entertainment

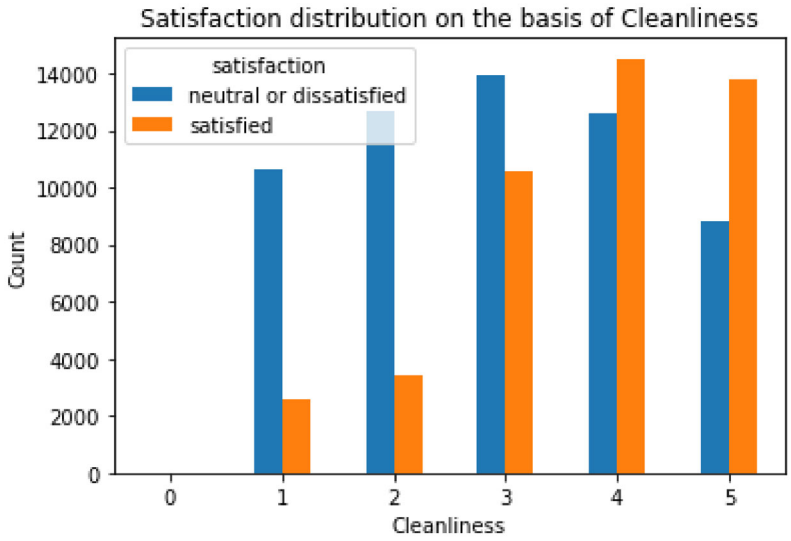

Satisfaction distribution on the basis of On-board service
satisfaction
neutral or dissatisfied
satisfied
Count
On-board service


Satisfaction distribution on the basis of Leg room service
satisfaction
neutral or dissatisfied
satisfied
Count
Leg room service


Satisfaction distribution on the basis of Baggage handling
satisfaction
neutral or dissatisfied
satisfied
Count
Baggage handling
0
2500
5000
7500
10000
12500
15000
17500
20000
1
2
3
4
5

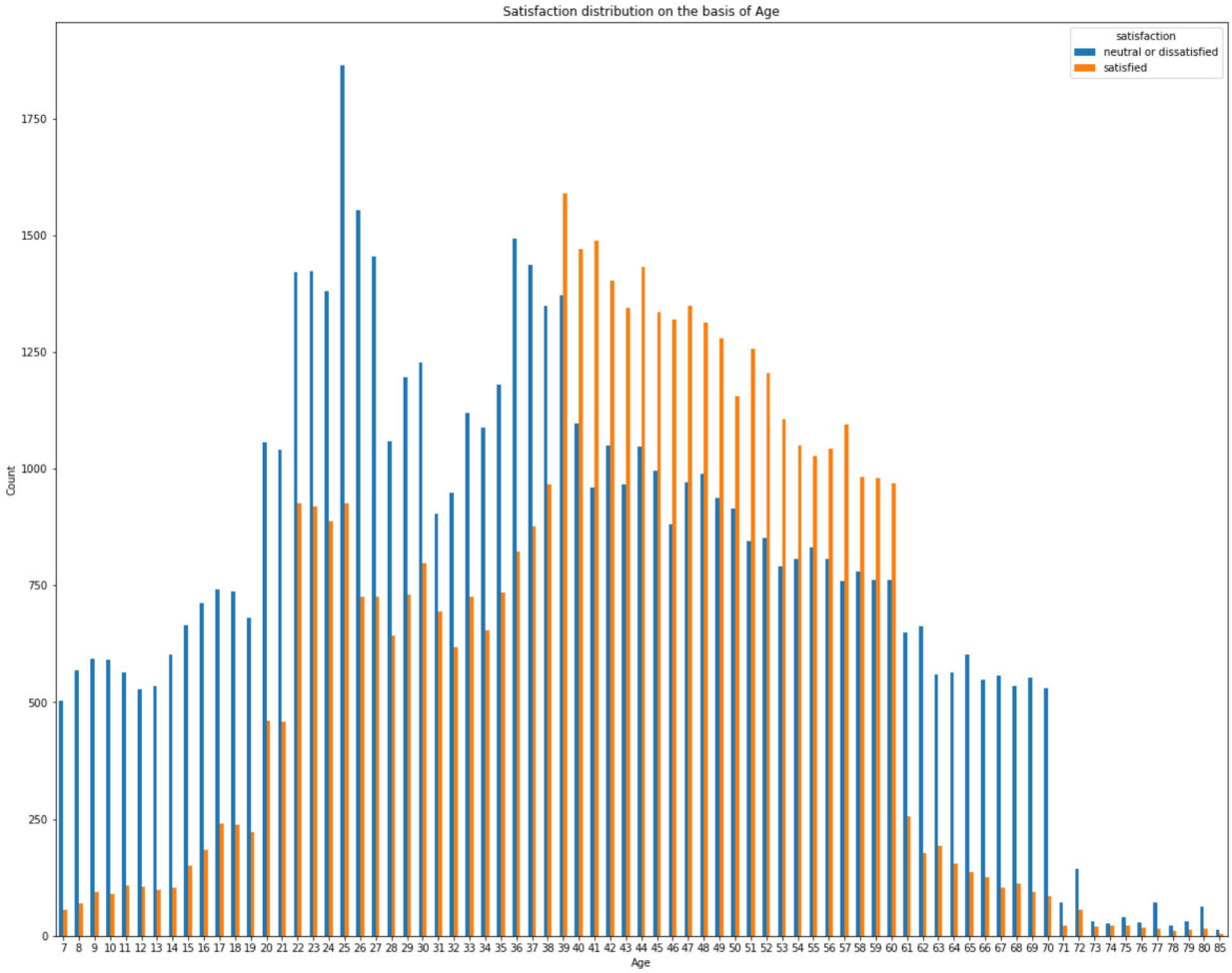

Satisfaction distribution on the basis of Checkin service
satisfaction
neutral or dissatisfied
satisfied
Count
Checkin service
0
2000
4000
6000
8000
10000
12000
14000
16000
0
1
2
3
4
5


Satisfaction distribution on the basis of Inflight service
satisfaction
neutral or dissatisfied
satisfied
Count
Inflight service
0
2500
5000
7500
10000
12500
15000
17500
20000
0
1
2
3
4
5



In [16]:
```
#Age vs Satisfaction Analysis
dev_df.groupby(['Age', 'satisfaction']).size().unstack().plot(kind='bar', rot=0, figsi
plt.ylabel('Count')
plt.title('Satisfaction distribution on the basis of Age')
```

Out[16]:
```
Text(0.5, 1.0, 'Satisfaction distribution on the basis of Age')
```



In [17]:
```
dev_df[(dev_df['Age'] >= 39) & (dev_df['Age']<=60)]['satisfaction'].value_counts()
```

Out[17]:
```
satisfied                 27197
neutral or dissatisfied   20182
Name: satisfaction, dtype: int64
```

In [18]:
```
dev_df[~(dev_df['Age'] >= 39) | ~(dev_df['Age']<=60)]['satisfaction'].value_counts()
```

Out[18]:
```
neutral or dissatisfied   38515
satisfied                 17700
Name: satisfaction, dtype: int64
```

In [19]:
```
dev_df['satisfaction'].replace({'neutral or dissatisfied': 0, 'satisfied': 1},inplace
test_df['satisfaction'].replace({'neutral or dissatisfied': 0, 'satisfied': 1},inplace
```

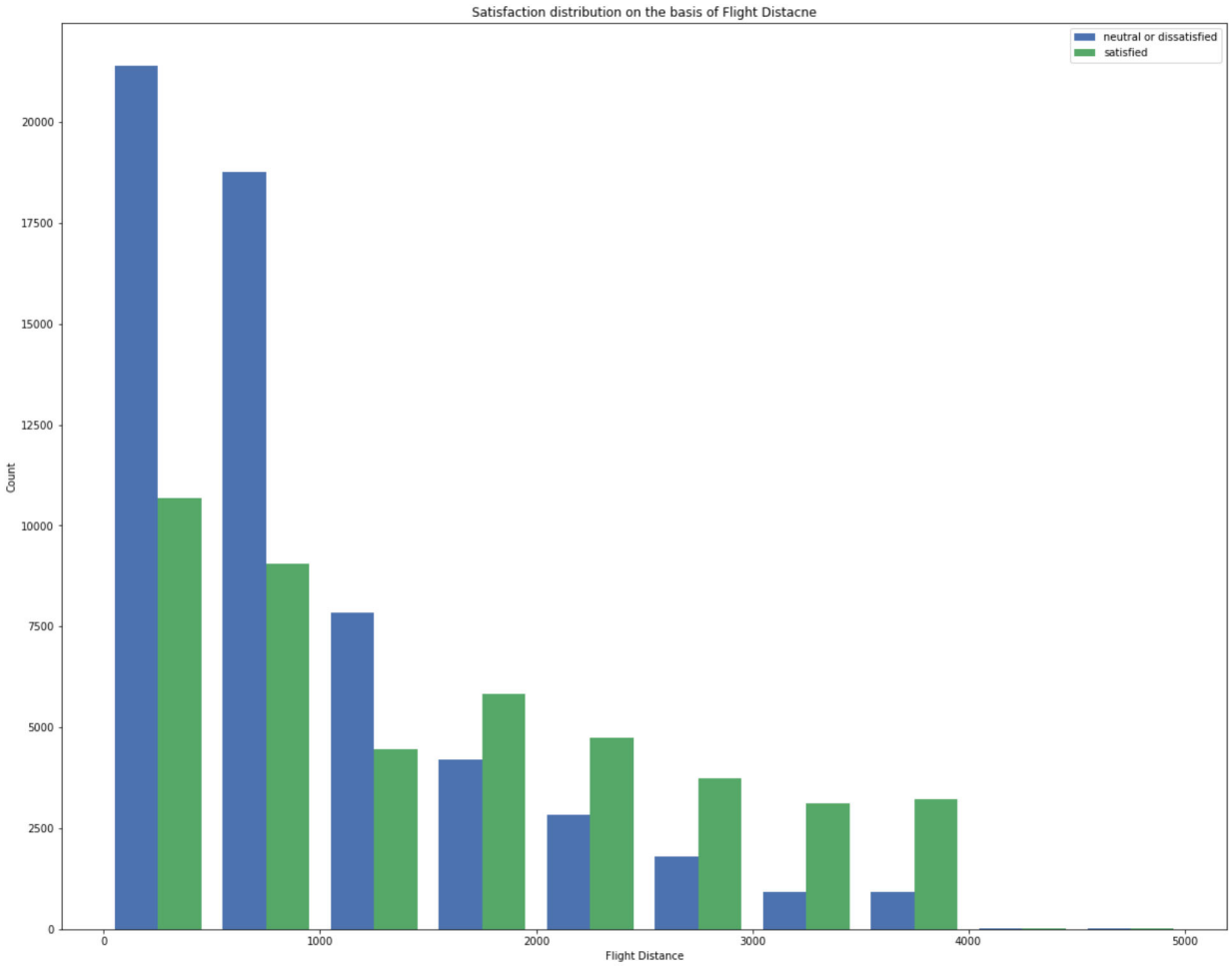
```
C:\Users\16207\AppData\Local\Temp\ipykernel_8336\2439646969.py:1: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  dev_df['satisfaction'].replace({'neutral or dissatisfied': 0, 'satisfied': 1},inpla
ce = True)
C:\Users\16207\AppData\Local\Temp\ipykernel_8336\2439646969.py:2: SettingWithCopyWarn
ing:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/us
er_guide/indexing.html#returning-a-view-versus-a-copy
  test_df['satisfaction'].replace({'neutral or dissatisfied': 0, 'satisfied': 1},inpl
ace = True)
```

In [20]:
```
X_dev['Flight Distance'].describe()
```

Out[20]:
```
count    103594.000000
mean       1189.325202
std         997.297235
min          31.000000
25%         414.000000
50%         842.000000
75%        1743.000000
max        4983.000000
Name: Flight Distance, dtype: float64
```

In [21]:
```
plt.style.use('seaborn-deep')
plt.figure(figsize=(20,16))
y0 = dev_df[dev_df['satisfaction'] == 0]['Flight Distance']
y1 = dev_df[dev_df['satisfaction'] == 1]['Flight Distance']
plt.hist([y0, y1], bins=np.linspace(0, 5000, 11))
plt.legend(['neutral or dissatisfied', 'satisfied'])
plt.xlabel('Flight Distance')
plt.ylabel('Count')
plt.title('Satisfaction distribution on the basis of Flight Distacne')
```

Out[21]:
```
Text(0.5, 1.0, 'Satisfaction distribution on the basis of Flight Distacne')
```



In [22]:
```
dev_df[(dev_df['Flight Distance'] <= 1500)]['satisfaction'].value_counts()
```

Out[22]:
```
0    48008
1    24213
Name: satisfaction, dtype: int64
```

In [23]:
```
dev_df[(dev_df['Flight Distance'] > 1500)]['satisfaction'].value_counts()
```

Out[23]:
```
1    20684
0    10689
Name: satisfaction, dtype: int64
```

In [24]:
```
dev_df[(dev_df['Flight Distance'] > 4000)]['satisfaction'].value_counts()
```

Out[24]:
```
1    32
0    26
Name: satisfaction, dtype: int64
```

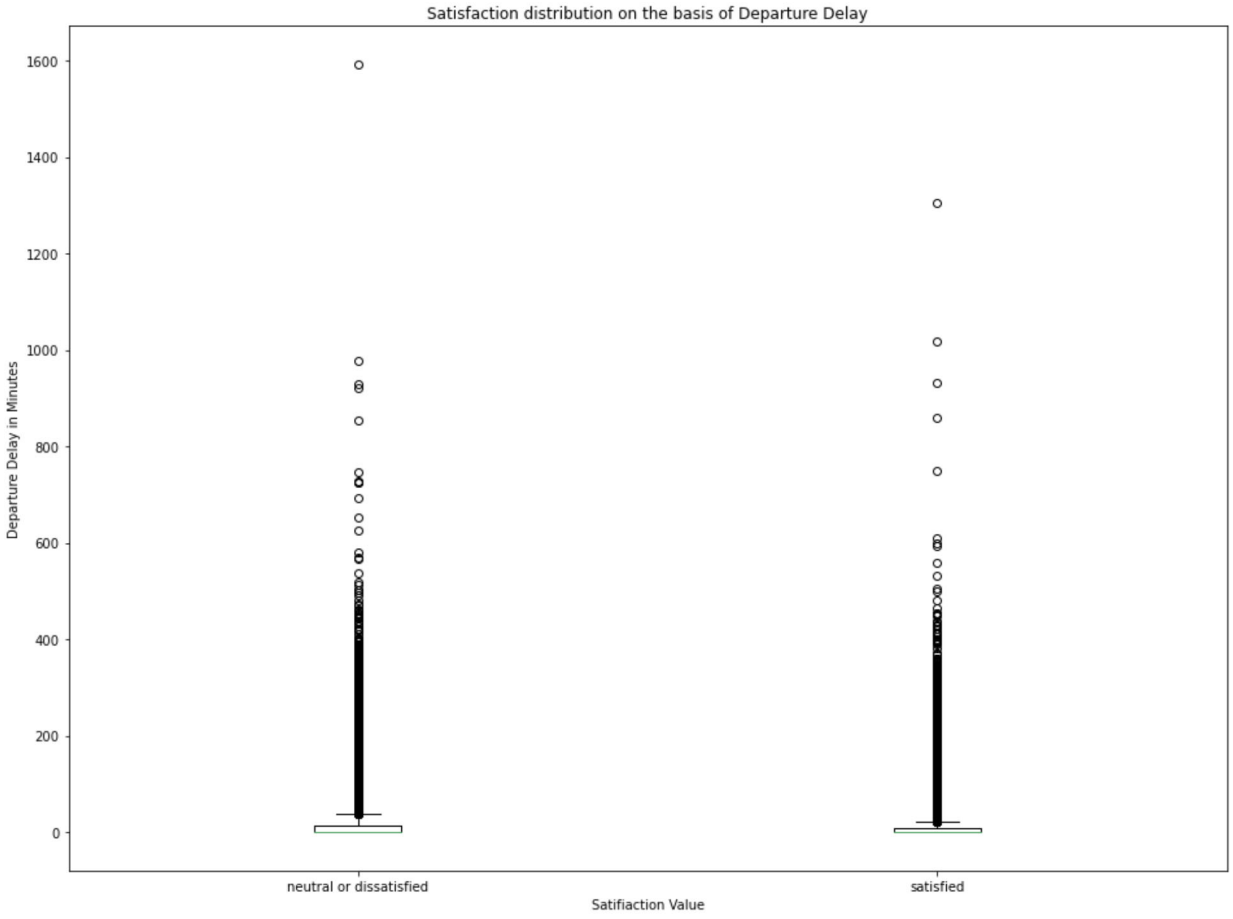
- 69.7% customers in the survey travelled less than 1500 miles.
- Two-thirds of the customers travelling less than 1500 miles were neutral or dissatisfied.
- Amongst customers who travelled more than 1500 miles, the number of satisfied customers is twice as many as dissatisfied customers.
- The number of customers who travelled more than 4000 miles is just 58, which is ~0.056% of the total sample size.

In [25]:
```
X_dev['Departure Delay in Minutes'].describe()
```

Out[25]:
```
count    103594.000000
mean         14.747939
std          38.116737
min           0.000000
25%           0.000000
50%           0.000000
75%          12.000000
max        1592.000000
Name: Departure Delay in Minutes, dtype: float64
```

In [26]:
```
# plt.figure(figsize=(24,18))
y00 = dev_df[dev_df['satisfaction'] == 0]['Departure Delay in Minutes']
y01 = dev_df[dev_df['satisfaction'] == 1]['Departure Delay in Minutes']
plt.figure(figsize=(16,12))
plt.boxplot([y00, y01])
plt.xticks([1,2], ['neutral or dissatisfied', 'satisfied'])
plt.xlabel('Satifiaction Value')
plt.ylabel('Departure Delay in Minutes')
plt.title('Satisfaction distribution on the basis of Departure Delay')
```

Out[26]:
```
Text(0.5, 1.0, 'Satisfaction distribution on the basis of Departure Delay')
```

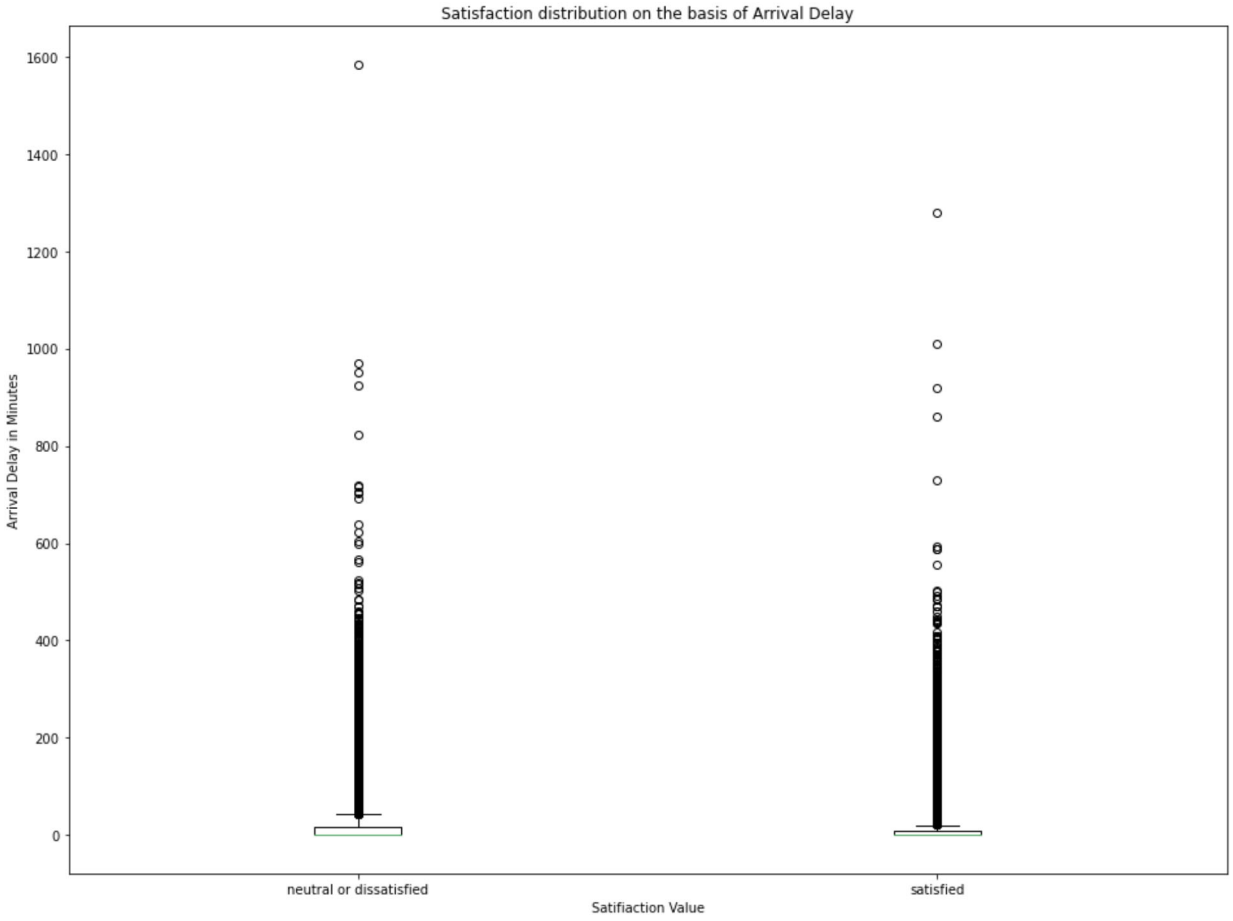


In [27]:
```
X_dev['Arrival Delay in Minutes'].describe()
```

Out[27]:
```
count    103594.000000
mean         15.178678
std          38.698682
min           0.000000
25%           0.000000
50%           0.000000
75%          13.000000
max        1584.000000
Name: Arrival Delay in Minutes, dtype: float64
```

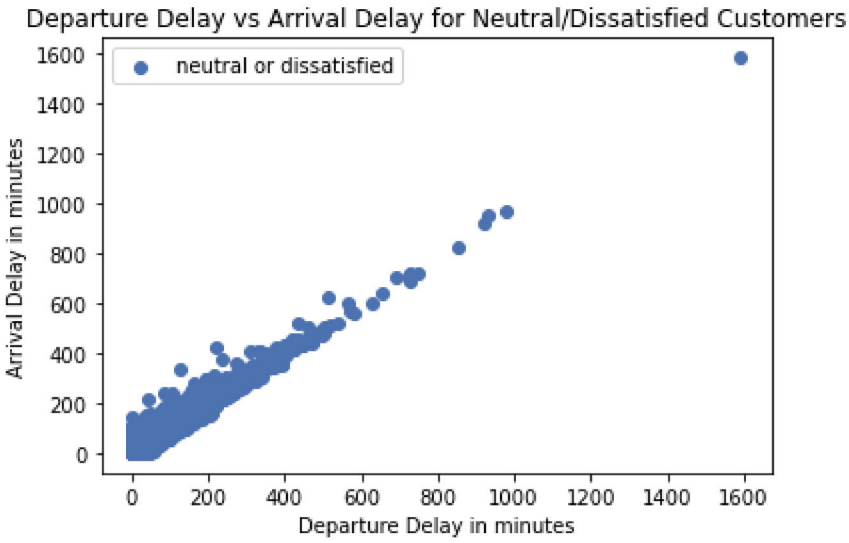
In [28]:
```
y10 = dev_df[dev_df['satisfaction'] == 0]['Arrival Delay in Minutes']
y11 = dev_df[dev_df['satisfaction'] == 1]['Arrival Delay in Minutes']
plt.figure(figsize=(16,12))
plt.boxplot([y10, y11])
plt.xticks([1,2], ['neutral or dissatisfied', 'satisfied'])
plt.xlabel('Satifiaction Value')
plt.ylabel('Arrival Delay in Minutes')
plt.title('Satisfaction distribution on the basis of Arrival Delay')
```

Out[28]:
```
Text(0.5, 1.0, 'Satisfaction distribution on the basis of Arrival Delay')
```



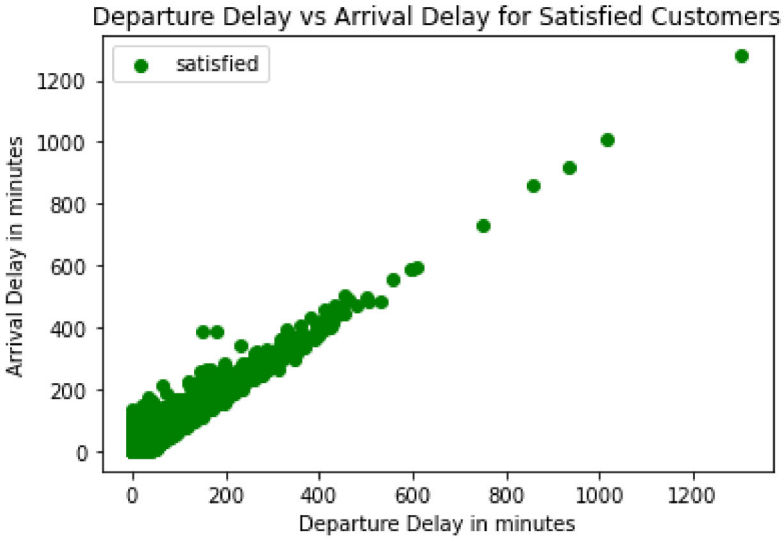
In [29]:
```
plt.scatter(y00, y10)
plt.legend(['neutral or dissatisfied'])
plt.ylabel('Arrival Delay in minutes')
plt.xlabel('Departure Delay in minutes')
plt.title('Departure Delay vs Arrival Delay for Neutral/Dissatisfied Customers')
```

Out[29]:
```
Text(0.5, 1.0, 'Departure Delay vs Arrival Delay for Neutral/Dissatisfied Customers')
```



In [30]:
```
plt.scatter(y01, y11, color='green')
plt.legend(['satisfied'])
plt.ylabel('Arrival Delay in minutes')
plt.xlabel('Departure Delay in minutes')
plt.title('Departure Delay vs Arrival Delay for Satisfied Customers')
```

Out[30]:
```
Text(0.5, 1.0, 'Departure Delay vs Arrival Delay for Satisfied Customers')
```



- Highly skewed
  - Departure delay: first quartile = 0.00 ; third quartile = 12.00
  - Arrival delay: first quartile = 0.00 ; third quartile = 13.00

1.6: Preprocess the data (Handle the categorical variable and apply scaling)

In [31]:
```
# Encoding Categorical features
for feature in cat_feats:
  string=feature+'_encoded'
  enc = OrdinalEncoder()
  X_dev[string] = enc.fit_transform(np.asarray(X_dev[feature]).reshape(-1,1))
  X_test[string] = enc.transform(np.asarray(X_test[feature]).reshape(-1,1))
```

In [32]:
```
#Replacing categorical features with encoded counterparts and plotting Correlation mat
X_dev = X_dev.drop(columns=cat_feats)
X_dev.corr().abs().style.background_gradient(cmap='coolwarm')
```

Out[32]:

	Age	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	Food and drink	O boar
Age	1.000000	0.099838	0.017470	0.038038	0.024461	0.001558	0.022920	0.20
Flight Distance	0.099838	1.000000	0.007050	0.019908	0.065697	0.004732	0.056957	0.21
Inflight wifi service	0.017470	0.007050	1.000000	0.343758	0.715848	0.336127	0.134603	0.45
Departure/Arrival time convenient	0.038038	0.019908	0.343758	1.000000	0.437021	0.444601	0.005189	0.06
Ease of Online booking	0.024461	0.065697	0.715848	0.437021	1.000000	0.458746	0.031940	0.40
Gate location	0.001558	0.004732	0.336127	0.444601	0.458746	1.000000	0.001170	0.00
Food and drink	0.022920	0.056957	0.134603	0.005189	0.031940	0.001170	1.000000	0.23
Online boarding	0.208681	0.215191	0.457002	0.069990	0.404093	0.001451	0.234492	1.00
Seat comfort	0.160302	0.157517	0.122617	0.011416	0.030021	0.003383	0.574561	0.42
Inflight entertainment	0.076380	0.128645	0.209513	0.004683	0.047185	0.003564	0.622374	0.28
On-board service	0.057123	0.109540	0.121484	0.068604	0.038759	0.028532	0.058999	0.15
Leg room service	0.040498	0.133839	0.160485	0.012461	0.107431	0.005868	0.032415	0.12
Baggage handling	0.047619	0.063222	0.121060	0.071901	0.038851	0.002421	0.034811	0.08
Checkin service	0.035003	0.073224	0.043178	0.093329	0.010957	0.035451	0.087055	0.20
Inflight service	0.049899	0.057430	0.110626	0.073227	0.035330	0.001742	0.034077	0.07
Cleanliness	0.053493	0.093121	0.132652	0.014337	0.016192	0.004015	0.657648	0.33
Departure Delay in Minutes	0.010150	0.001906	0.017451	0.000791	0.006292	0.005533	0.029983	0.01
Arrival Delay in Minutes	0.012147	0.002426	0.019095	0.000864	0.007984	0.005143	0.032524	0.02
Gender_encoded	0.008921	0.006079	0.008964	0.008846	0.007166	0.000213	0.005707	0.04
Customer Type_encoded	0.281821	0.225363	0.007706	0.207007	0.019627	0.006294	0.059554	0.18
Type of Travel_encoded	0.048593	0.267642	0.104879	0.259829	0.133399	0.030802	0.063124	0.22
Class_encoded	0.117423	0.427509	0.023046	0.089793	0.094323	0.004532	0.076834	0.29

In [33]:
```
#Replacing categorical features with encoded counterparts in test set
X_test = X_test.drop(columns=cat_feats)
```

In [34]:
```
#Dropping 'Departure Delay in Minutes' due to high correlation with 'Arrival Delay in
X_dev = X_dev.drop(columns=['Departure Delay in Minutes'])
X_test = X_test.drop(columns=['Departure Delay in Minutes'])
```

In [35]:
```
#Scaling the values
scaler = StandardScaler()
X_dev_scaled = scaler.fit_transform(X_dev)
X_test_scaled = scaler.fit(X_test)
```

Part 2: Logistic Regression

2.1: Add a column of ones to the feature matrices for the bias term

In [36]:
```
X_dev_1 = np.hstack([np.ones((X_dev.shape[0], 1)), X_dev])
X_test_1 = np.hstack([np.ones((X_test.shape[0], 1)), X_test])
```

2.2: Grid search to find the best parameters for logistic regression

In [37]:
```
from sklearn.linear_model import LogisticRegression

# Define the hyperparameters to tune
param_grid = {
    'C': [0.1, 1.0, 10.0, 100.0, 1000.0],
    'penalty': ['l2'],
}

# Create a LogisticRegression object
log_reg = LogisticRegression(random_state=0)

# Create a GridSearchCV object with 3-fold cross-validation
grid_search = GridSearchCV(log_reg, param_grid=param_grid, cv=3)

# Fit the GridSearchCV object on the development dataset
grid_search.fit(X_dev_1, y_dev)

# # Print the best hyperparameters and the corresponding mean cross-validation score
print(f"Best hyperparameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_:.3f}")

best_model_log_reg = grid_search.best_estimator_
y_pred_log_reg = best_model_log_reg.predict(X_test_1)
prob_y_pred_log_reg = best_model_log_reg.predict_proba(X_test_1)
test_accuracy = accuracy_score(y_test, y_pred_log_reg)
print(f"Test accuracy of the best model: {test_accuracy:.3f}")
```

```
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Best hyperparameters: {'C': 100.0, 'penalty': 'l2'}
Best cross-validation score: 0.840
Test accuracy of the best model: 0.819
C:\Users\16207\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: Con
vergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

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

Part 3: Random Forest

3.1: Grid search to find the best parameters for random forest

In [38]:

```python
from sklearn.ensemble import RandomForestClassifier

# Define the hyperparameters to tune
param_grid = {
    'max_features': ['sqrt', 'log2', 0.3, 0.6, 0.9],
    'n_estimators': [30, 60, 90, 120, 150]
}

# Create a RandomForestClassifier object
rf = RandomForestClassifier(random_state=0)

# Create a GridSearchCV object with 3-fold cross-validation
grid_search = GridSearchCV(rf, param_grid=param_grid, cv=3)

# Fit the GridSearchCV object on the development dataset
grid_search.fit(X_dev, y_dev)

# # Print the best hyperparameters and the corresponding mean cross-validation score
print(f"Best hyperparameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_:.3f}")

best_model_rf = grid_search.best_estimator_
y_pred_rf = best_model_rf.predict(X_test)
prob_y_pred_rf = best_model_rf.predict_proba(X_test)
```

```
test_accuracy = accuracy_score(y_test, y_pred_rf)
print(f"Test accuracy of the best model: {test_accuracy:.3f}")
```

```
Best hyperparameters: {'max_features': 0.3, 'n_estimators': 150}
Best cross-validation score: 0.963
Test accuracy of the best model: 0.964
```

Part 4: XGBoost

4.1: Grid search to find the best parameters for XGBoost

In [39]:
```
from xgboost import XGBClassifier

# Define the hyperparameters to tune
param_grid = {
    'learning_rate': [0.001, 0.01, 0.1, 1],
    'max_depth': [6, 9, 12, 15],
    'n_estimators': [50, 100, 150]
}

# Create a XGBoostingClassifier object
clf = XGBClassifier(random_state=0)

# Create a GridSearchCV object with 3-fold cross-validation
grid_search = GridSearchCV(clf, param_grid=param_grid, cv=3)

# Fit the GridSearchCV object on the development dataset
grid_search.fit(X_dev, y_dev)

# # Print the best hyperparameters and the corresponding mean cross-validation score
print(f"Best hyperparameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_:.3f}")

best_model_xgb = grid_search.best_estimator_
y_pred_xgb = best_model_xgb.predict(X_test)
prob_y_pred_xgb = best_model_xgb.predict_proba(X_test)
test_accuracy = accuracy_score(y_test, y_pred_xgb)
print(f"Test accuracy of the best model: {test_accuracy:.3f}")
```

```
Best hyperparameters: {'learning_rate': 0.1, 'max_depth': 9, 'n_estimators': 150}
Best cross-validation score: 0.964
Test accuracy of the best model: 0.964
```

Part 5: Support Vector Machines

5.1: Grid search to find the best parameters for SVM using SGDClassifier

In [59]:
```
import time
from sklearn.linear_model import SGDClassifier

# record start time
s_time = time.time()

# Create a SGDClassifier object, stochastic gradient descent
SGD = SGDClassifier(random_state=0)
```

```
# Create a GridSearchCV object with 3-fold cross-validation
param_grid = {
    'loss':['hinge', 'squared_hinge'] ,
    'alpha':[1e-3, 1e-4, 1e-5]
}

# Fit the GridSearchCV object on the development dataset
grid_search = GridSearchCV(SGD, param_grid=param_grid, cv=3)
grid_search.fit(X_dev, y_dev)
# Print the best hyperparameters and the corresponding mean cross-validation score
print(f"Best hyperparameters: {grid_search.best_params_}")
print(f"Best cross-validation score: {grid_search.best_score_:.3f}")

best_model_SGD = grid_search.best_estimator_

print('Training time: {} seconds'.format(time.time() - s_time))
```

```
Best hyperparameters: {'alpha': 0.001, 'loss': 'squared_hinge'}
Best cross-validation score: 0.815
Training time: 50.13587141036987 seconds
```

In [60]:
```
y_pred_SGD = best_model_SGD.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred_SGD)
print(f"Test accuracy of the best model: {test_accuracy:.3f}")
```

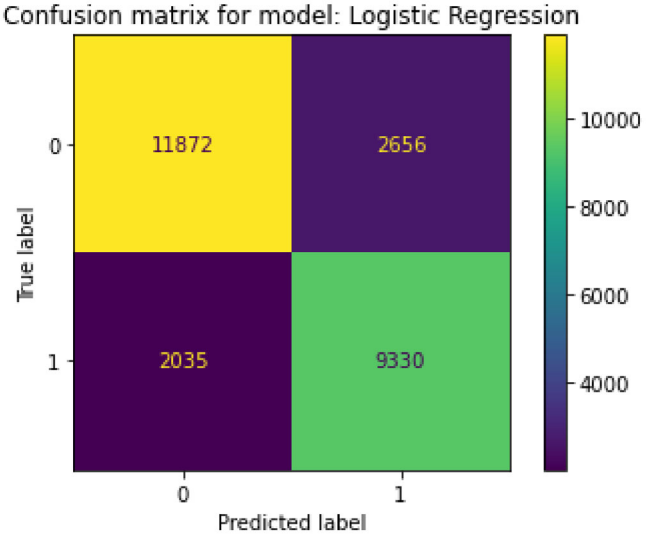
```
Test accuracy of the best model: 0.829
```

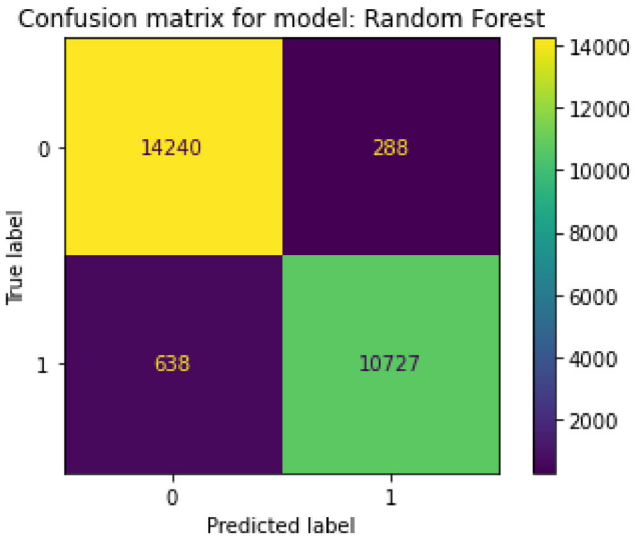
Part 6: Result Analysis (Comparing Different Methods)

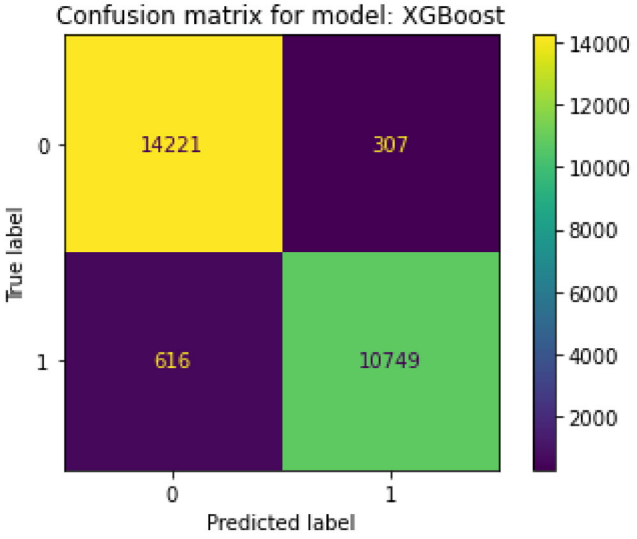
In [63]:
```
predictions = [y_pred_log_reg, y_pred_rf, y_pred_xgb, y_pred_SGD]
models = ['Logistic Regression', 'Random Forest', 'XGBoost', 'Support Vector Machines'
```

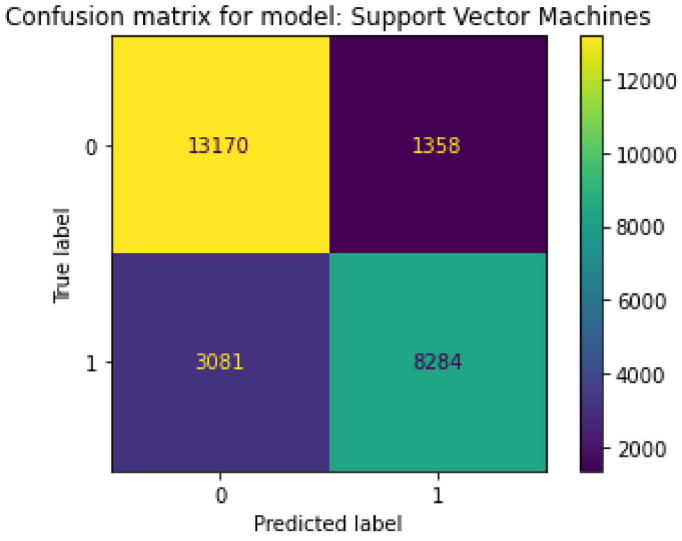
Confusion Matrix

In [64]:
```
for i,pred in enumerate(predictions):
    cm = confusion_matrix(y_test, pred)
    disp = ConfusionMatrixDisplay(confusion_matrix=cm)
    disp.plot()
    disp.ax_.set_title('Confusion matrix for model: ' + models[i])
```









Classification Report

In [65]:
```
for i,pred in enumerate(predictions):
    print('Classification report for model: ' + models[i])
    cr = pd.DataFrame(classification_report(y_test, pred, output_dict=True))
    display(cr)
```

```
Classification report for model: Logistic Regression
```

	0	1	accuracy	macro avg	weighted avg
precision	0.853671	0.778408	0.818831	0.816039	0.820636
recall	0.817181	0.820941	0.818831	0.819061	0.818831
f1-score	0.835027	0.799109	0.818831	0.817068	0.819262
support	14528.000000	11365.000000	0.818831	25893.000000	25893.000000

```
Classification report for model: Random Forest
```

	0	1	accuracy	macro avg	weighted avg
precision	0.957118	0.973854	0.964237	0.965486	0.964464
recall	0.980176	0.943863	0.964237	0.962019	0.964237
f1-score	0.968510	0.958624	0.964237	0.963567	0.964171
support	14528.000000	11365.000000	0.964237	25893.000000	25893.000000

```
Classification report for model: XGBoost
```

	0	1	accuracy	macro avg	weighted avg
precision	0.958482	0.972232	0.964353	0.965357	0.964517
recall	0.978868	0.945799	0.964353	0.962333	0.964353
f1-score	0.968568	0.958833	0.964353	0.963701	0.964295
support	14528.000000	11365.000000	0.964353	25893.000000	25893.000000

```
Classification report for model: Support Vector Machines
```

	0	1	accuracy	macro avg	weighted avg
precision	0.810412	0.859158	0.828564	0.834785	0.831807
recall	0.906525	0.728905	0.828564	0.817715	0.828564
f1-score	0.855778	0.788689	0.828564	0.822234	0.826332
support	14528.000000	11365.000000	0.828564	25893.000000	25893.000000