CST 383 - Final Project

    BayTech

- Warren Ngoun (wngoun@csumb.edu)
- Yukio Rivera (yrivera@csumb.edu)
- Jennah Yasin (jyasin@csumb.edu)
- Luis Jimenez Barrios (ljimenezbarrios@csumb.edu)

Table of Contents:

Intro

Back to Top

We chose the Airline Passenger Satisfaction dataset for our final project because we are all interested in satisfying flight experiences and believe that it is important for airlines to take their passenger's reviews into consideration.

- Link to Kaggle Repo: https://www.kaggle.com/datasets/teejmahal20/airline-passenger-satisfaction

With this dataset, we are going to predict what factors may be most relevant and most correlated to the passenger's satisfaction. We will be analyzing which features are most correlated to our topic and dropping those that aren't as relevant/needed.

Also a note, the file was pre-processed by our Kaggle author, he split the original data set into two separate files (test and train) while also removing a vast majority of NaN and bad data, so most of that work was already done for us by the repo author.

We plan to use the "satisfaction" feature as our target label for this project. The initial features that we plan to use as predictors for our project are: Seat comfort, in-flight entertainment, cleanliness, and food & drinks, but that may change as we test them out later.

Column Contents

Taken from the Kaggle Repo description.

- Gender: Gender of the passengers (Female, Male)
- 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 stars)
- 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 (0: Not Applicable; 1-5 stars)
- 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 or `0`, neutral/dissatisfied or `1`)

Imports

All the necessary imports we need for the project.

In [625…

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# For ML Work
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import learning_curve
from sklearn.metrics import confusion_matrix
```

From our dataset, we have two files from the original Kaggle repo, an already split training and test csv, so we made two separate data frames and all the pre-processing and changes we do will be applied to both so later tests and predictions all have the same columns to draw from.

In [626…

```
df = pd.read_csv('https://raw.githubusercontent.com/BayTech-CSUMB/CST383Final/main/train
dfTest = pd.read_csv('https://raw.githubusercontent.com/BayTech-CSUMB/CST383Final/main/t
```

Data Investigation and Preprocessing

Back to Top

First we do some preliminary looking at our dataset with `info()` and `describe()`.

In [627…

```
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
```

Our below describe shows us that a lot of our column data is from a range of 0.0 to 5.0, akin to a star rating. Others are in full ints like *flight distance* or *food and drink*. We will z-score normalize our data when we start the Machine Learning training.

In [628…

```
df.describe().round(1)
```

Out[628]:

	Unnamed: 0	id	Age	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	Foo an drin
count	103904.0	103904.0	103904.0	103904.0	103904.0	103904.0	103904.0	103904.0	103904.

mean	51951.5	64924.2	39.4	1189.4	2.7	3.1	2.8	3.0	3.
std	29994.6	37463.8	15.1	997.1	1.3	1.5	1.4	1.3	1.
min	0.0	1.0	7.0	31.0	0.0	0.0	0.0	0.0	0.
25%	25975.8	32533.8	27.0	414.0	2.0	2.0	2.0	2.0	2.
50%	51951.5	64856.5	40.0	843.0	3.0	3.0	3.0	3.0	3.
75%	77927.2	97368.2	51.0	1743.0	4.0	4.0	4.0	4.0	4.
max	103903.0	129880.0	85.0	4983.0	5.0	5.0	5.0	5.0	5.

Lets see which columns are categorical.

In [629…

```
df.dtypes[df.dtypes == 'object']
```

Out[629]:

```
Gender              object
Customer Type       object
Type of Travel      object
Class               object
satisfaction        object
dtype: object
```

For each of those above categories, lets investigate the values.

In [630…

```
print(df['Gender'].value_counts())
```

```
Female     52727
Male       51177
Name: Gender, dtype: int64
```

In [631…

```
print(df['Customer Type'].value_counts())
```

```
Loyal Customer       84923
disloyal Customer    18981
Name: Customer Type, dtype: int64
```

In [632…

```
print(df['Type of Travel'].value_counts())
```

```
Business travel    71655
Personal Travel    32249
Name: Type of Travel, dtype: int64
```

In [633…

```
print(df['Class'].value_counts())
```

```
Business    49665
Eco         46745
Eco Plus     7494
Name: Class, dtype: int64
```

In [634…

```
print(df['Type of Travel'].value_counts())
```

```
Business travel    71655
Personal Travel    32249
Name: Type of Travel, dtype: int64
```

Data Encoding

Now we'll go ahead and One-Hot Encode all of those categorical values and then label encode *satisfaction* so we can use a single column for the machine learning.

In [635…

```
cols = ['Gender', 'Type of Travel', 'Class', 'Customer Type']
# Keeping these as backups for graphing later.
```

```python
classForGraphing = df['Class']
genderForGraphing = df['Gender']

for col in cols:
    catCol = pd.get_dummies(df[col], prefix=col)
    df.drop(col, axis=1, inplace=True)
    df = pd.concat([df, catCol], axis=1)
    # Repeat but for our test data set too.
    catCol2 = pd.get_dummies(dfTest[col], prefix=col)
    dfTest.drop(col, axis=1, inplace=True)
    dfTest = pd.concat([dfTest, catCol2], axis=1)
```

In [636…

```python
# Here we made a copy so that when we later try and visualize the data we can get proper
satisfactionForGraphing = df['satisfaction'].copy()
# Checking before and after. 0 = neutral/dissatisfied 1 = satisfied
print(df['satisfaction'].value_counts())
df['satisfaction'] = df['satisfaction'].astype('category').cat.codes
print(df['satisfaction'].value_counts())
```

```
neutral or dissatisfied    58879
satisfied                  45025
Name: satisfaction, dtype: int64
0    58879
1    45025
Name: satisfaction, dtype: int64
```

Lets check out the different columns from both of our data frames now.

In [637…

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103904 entries, 0 to 103903
Data columns (total 30 columns):
 #   Column                             Non-Null Count   Dtype
---  ------                             --------------   -----
 0   Unnamed: 0                         103904 non-null  int64
 1   id                                 103904 non-null  int64
 2   Age                                103904 non-null  int64
 3   Flight Distance                    103904 non-null  int64
 4   Inflight wifi service              103904 non-null  int64
 5   Departure/Arrival time convenient  103904 non-null  int64
 6   Ease of Online booking             103904 non-null  int64
 7   Gate location                      103904 non-null  int64
 8   Food and drink                     103904 non-null  int64
 9   Online boarding                    103904 non-null  int64
 10  Seat comfort                       103904 non-null  int64
 11  Inflight entertainment             103904 non-null  int64
 12  On-board service                   103904 non-null  int64
 13  Leg room service                   103904 non-null  int64
 14  Baggage handling                   103904 non-null  int64
 15  Checkin service                    103904 non-null  int64
 16  Inflight service                   103904 non-null  int64
 17  Cleanliness                        103904 non-null  int64
 18  Departure Delay in Minutes         103904 non-null  int64
 19  Arrival Delay in Minutes           103594 non-null  float64
 20  satisfaction                       103904 non-null  int8
 21  Gender_Female                      103904 non-null  uint8
 22  Gender_Male                        103904 non-null  uint8
 23  Type of Travel_Business travel     103904 non-null  uint8
 24  Type of Travel_Personal Travel     103904 non-null  uint8
 25  Class_Business                     103904 non-null  uint8
 26  Class_Eco                          103904 non-null  uint8
 27  Class_Eco Plus                     103904 non-null  uint8
 28  Customer Type_Loyal Customer       103904 non-null  uint8
 29  Customer Type_disloyal Customer    103904 non-null  uint8
```

```
dtypes: float64(1), int64(19), int8(1), uint8(9)
memory usage: 16.8 MB
```

In [638…

```
dfTest.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 25976 entries, 0 to 25975
Data columns (total 30 columns):
 #   Column                             Non-Null Count  Dtype  
---  ------                             --------------  -----  
 0   Unnamed: 0                         25976 non-null  int64  
 1   id                                 25976 non-null  int64  
 2   Age                                25976 non-null  int64  
 3   Flight Distance                    25976 non-null  int64  
 4   Inflight wifi service              25976 non-null  int64  
 5   Departure/Arrival time convenient  25976 non-null  int64  
 6   Ease of Online booking             25976 non-null  int64  
 7   Gate location                      25976 non-null  int64  
 8   Food and drink                     25976 non-null  int64  
 9   Online boarding                    25976 non-null  int64  
 10  Seat comfort                       25976 non-null  int64  
 11  Inflight entertainment             25976 non-null  int64  
 12  On-board service                   25976 non-null  int64  
 13  Leg room service                   25976 non-null  int64  
 14  Baggage handling                   25976 non-null  int64  
 15  Checkin service                    25976 non-null  int64  
 16  Inflight service                   25976 non-null  int64  
 17  Cleanliness                        25976 non-null  int64  
 18  Departure Delay in Minutes         25976 non-null  int64  
 19  Arrival Delay in Minutes           25893 non-null  float64
 20  satisfaction                       25976 non-null  object 
 21  Gender_Female                      25976 non-null  uint8  
 22  Gender_Male                        25976 non-null  uint8  
 23  Type of Travel_Business travel     25976 non-null  uint8  
 24  Type of Travel_Personal Travel     25976 non-null  uint8  
 25  Class_Business                     25976 non-null  uint8  
 26  Class_Eco                          25976 non-null  uint8  
 27  Class_Eco Plus                     25976 non-null  uint8  
 28  Customer Type_Loyal Customer       25976 non-null  uint8  
 29  Customer Type_disloyal Customer    25976 non-null  uint8  
dtypes: float64(1), int64(19), object(1), uint8(9)
memory usage: 4.4+ MB
```

We went from 24 columns to 29 now after encoding.

NaN Processing

Now we're interested if there are still any NaN values left in our dataset and then if the data will be suitable for imputation.

In [639…

```
print(f'There are {df.isna().sum().sum()} NaN values in df.')
df.isna().sum()
```

```
There are 310 NaN values in df.
```

Out[639]:

```
Unnamed: 0                           0
id                                   0
Age                                  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            310
satisfaction                          0
Gender_Female                         0
Gender_Male                           0
Type of Travel_Business travel        0
Type of Travel_Personal Travel        0
Class_Business                        0
Class_Eco                             0
Class_Eco Plus                        0
Customer Type_Loyal Customer          0
Customer Type_disloyal Customer       0
dtype: int64
```

In [640…

```
print(f'There are {dfTest.isna().sum().sum()} NaN values in our test df.')
dfTest.isna().sum()
```

```
There are 83 NaN values in our test df.
```

Out[640]:

```
Unnamed: 0                           0
id                                   0
Age                                  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            83
satisfaction                         0
Gender_Female                        0
Gender_Male                          0
Type of Travel_Business travel       0
Type of Travel_Personal Travel       0
Class_Business                       0
Class_Eco                            0
Class_Eco Plus                       0
Customer Type_Loyal Customer         0
Customer Type_disloyal Customer      0
dtype: int64
```

In [641…

```
# Calculate the average of the column
average_delay = df['Arrival Delay in Minutes'].mean()
average_delay_test = dfTest['Arrival Delay in Minutes'].mean()
# Impute & replace NaNs with the average value
df['Arrival Delay in Minutes'].fillna(value=average_delay, inplace=True)
dfTest['Arrival Delay in Minutes'].fillna(value=average_delay_test, inplace=True)
```

In [642…
```
print(f'There are {df.isna().sum().sum()} NaN values in df.')
print(f'There are {dfTest.isna().sum().sum()} NaN values in our test df.')
```

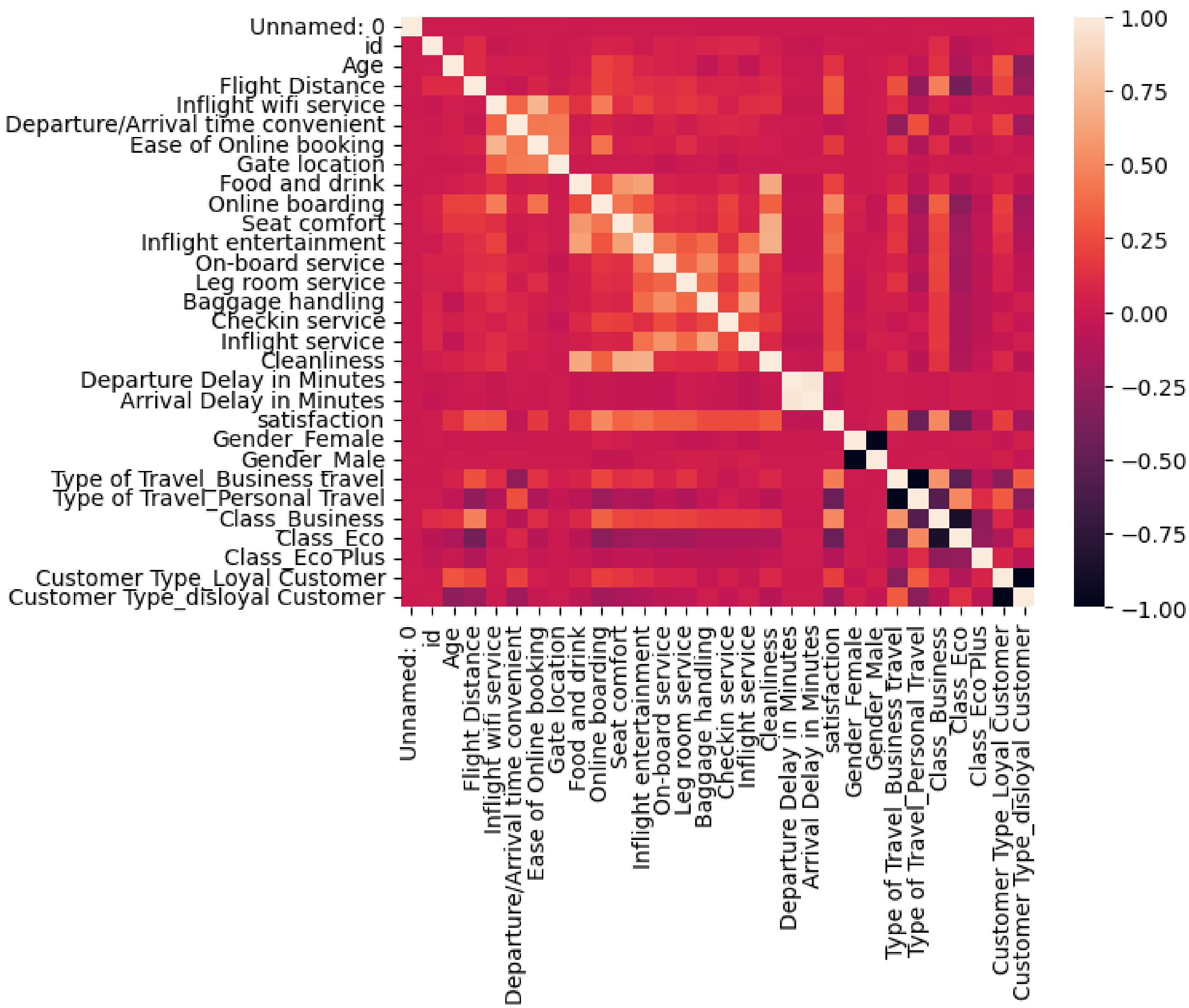
```
There are 0 NaN values in df.
There are 0 NaN values in our test df.
```

Correlation Heatmap & Column Dropping

We're interested in also trimming out unnecessary columns from the dataset so there are less potential noisy predictors. Here we have two correlation heat maps, one of before and after a purging of potential columns.

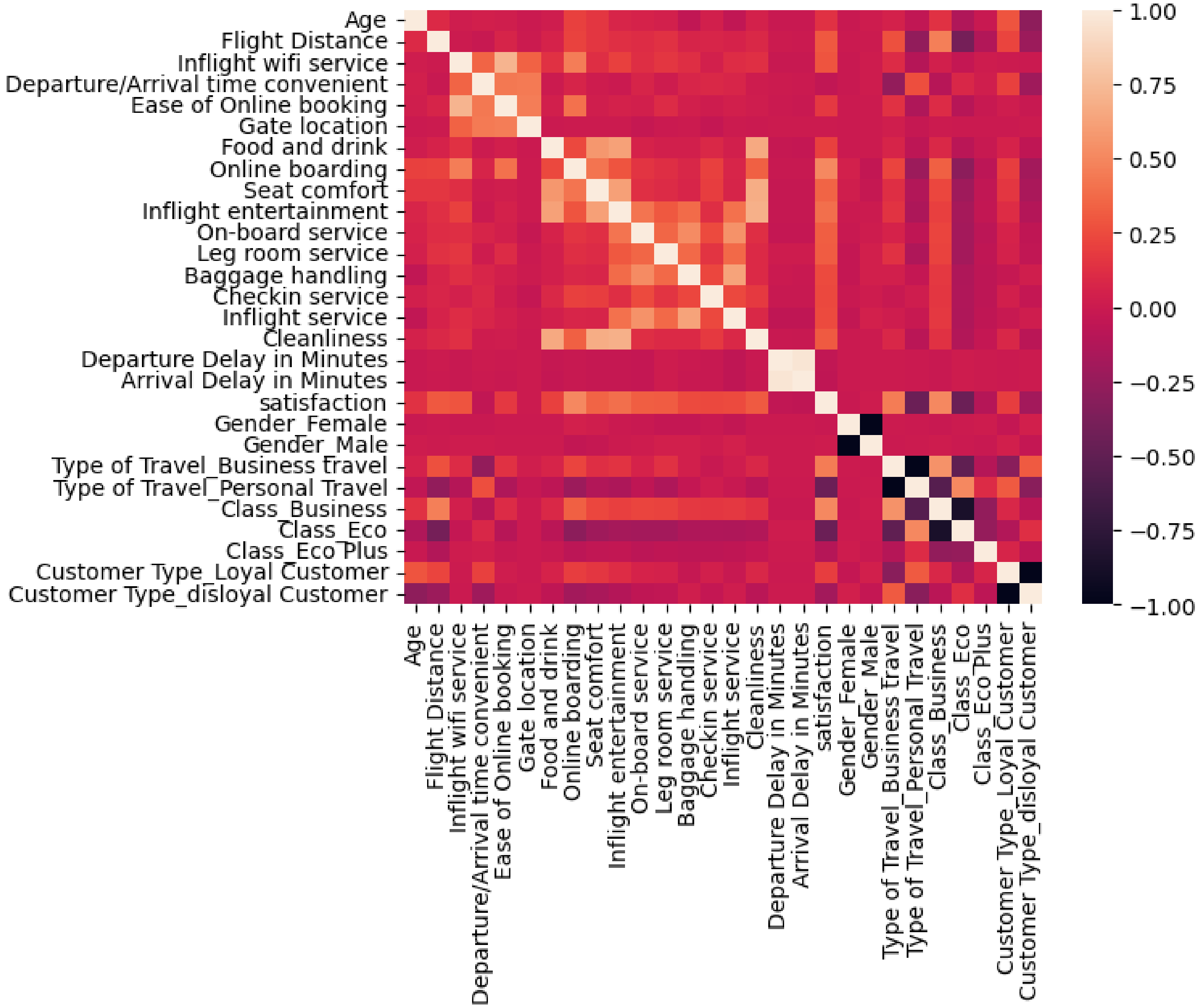
In [643…
```
corrDfOld = df.corr()
sns.heatmap(corrDfOld, xticklabels=corrDfOld.columns, yticklabels=corrDfOld.columns)
```

Out[643]: <AxesSubplot:>



In [644…
```
corrDf = df.copy()
corrDf.drop(['id', 'Unnamed: 0'], axis=1, inplace=True)
correlation = corrDf.corr()
sns.heatmap(correlation, xticklabels=correlation.columns, yticklabels=correlation.column
```

Out[644]: <AxesSubplot:>



There weren't too many hard correlations in our data now after encoding, other than some with the data formerly in encoded in binary categories like *Gender*. So considering that, we officially drop the "extra" columns from our dataset and investigate what's left again, leaving a majority of the data intact after all. We are now ultimately left with 27 columns, 26 potential predictors and 1 label.

In [645…

```
toDropCols = ['id', 'Unnamed: 0']
df.drop(toDropCols, axis=1, inplace=True)
dfTest.drop(toDropCols, axis=1, inplace=True)
```

In [646…

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 103904 entries, 0 to 103903
Data columns (total 28 columns):
 #   Column                             Non-Null Count   Dtype
---  ------                             --------------   -----
 0   Age                                103904 non-null  int64
 1   Flight Distance                    103904 non-null  int64
 2   Inflight wifi service              103904 non-null  int64
 3   Departure/Arrival time convenient  103904 non-null  int64
 4   Ease of Online booking             103904 non-null  int64
 5   Gate location                      103904 non-null  int64
 6   Food and drink                     103904 non-null  int64
 7   Online boarding                    103904 non-null  int64
 8   Seat comfort                       103904 non-null  int64
 9   Inflight entertainment             103904 non-null  int64
```

```
 10  On-board service                  103904 non-null  int64
 11  Leg room service                  103904 non-null  int64
 12  Baggage handling                  103904 non-null  int64
 13  Checkin service                   103904 non-null  int64
 14  Inflight service                  103904 non-null  int64
 15  Cleanliness                       103904 non-null  int64
 16  Departure Delay in Minutes        103904 non-null  int64
 17  Arrival Delay in Minutes          103904 non-null  float64
 18  satisfaction                      103904 non-null  int8
 19  Gender_Female                     103904 non-null  uint8
 20  Gender_Male                       103904 non-null  uint8
 21  Type of Travel_Business travel    103904 non-null  uint8
 22  Type of Travel_Personal Travel    103904 non-null  uint8
 23  Class_Business                    103904 non-null  uint8
 24  Class_Eco                         103904 non-null  uint8
 25  Class_Eco Plus                    103904 non-null  uint8
 26  Customer Type_Loyal Customer      103904 non-null  uint8
 27  Customer Type_disloyal Customer   103904 non-null  uint8
dtypes: float64(1), int64(17), int8(1), uint8(9)
memory usage: 15.3 MB
```

Data Exploration

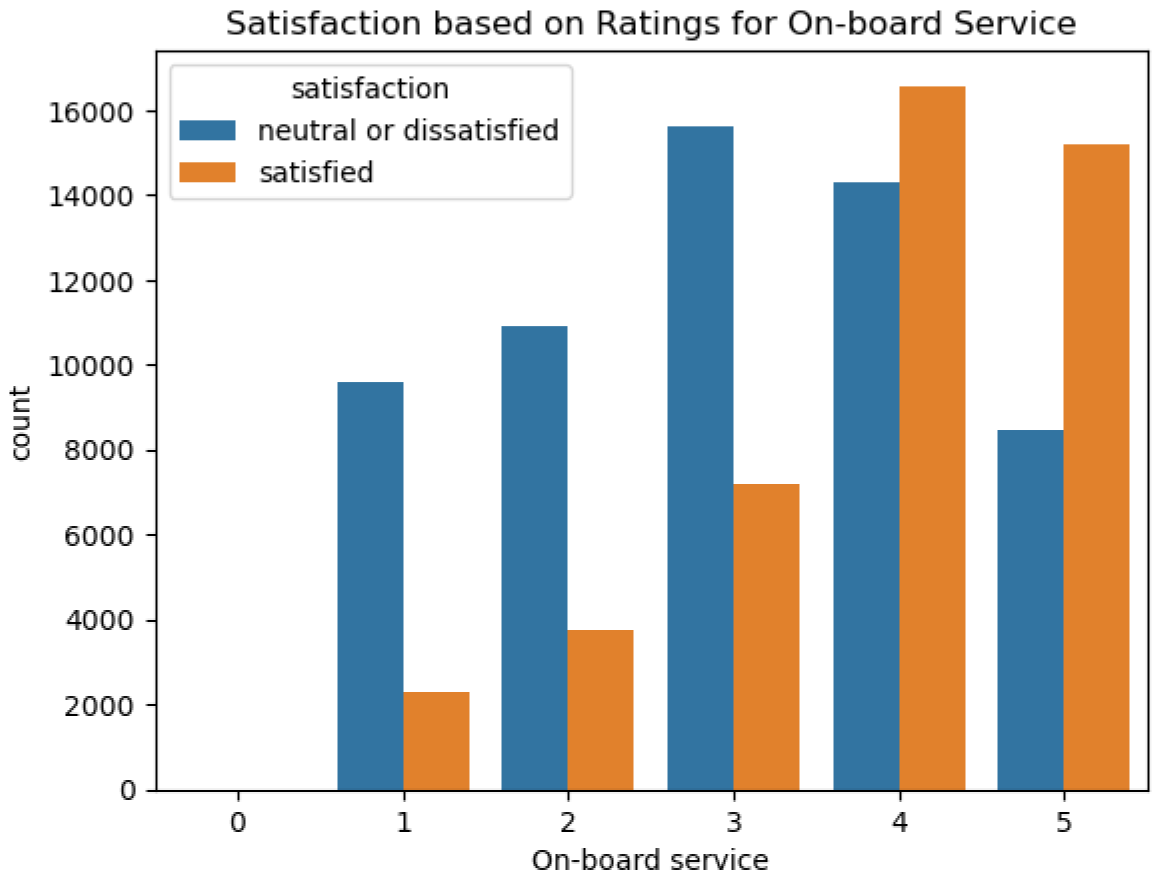
Back to Top

Now we look at certain columns and the relationships between them via graphing. Our satisfaction (which will be our labels) are a binary option with only two answers (neutral or dissatisfied or satisfied).

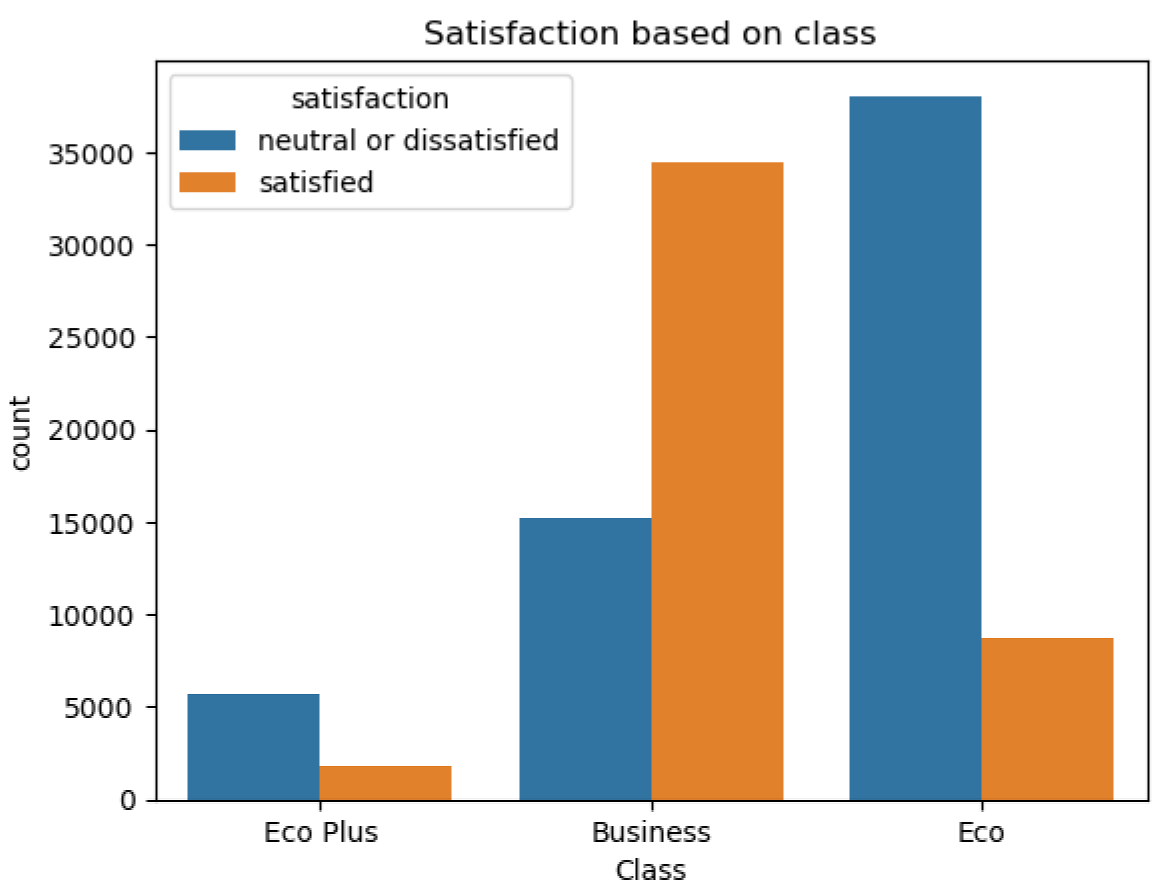
Visualizations

In [647…

```
# Passenger's satisfaction based on the Ratings of on-board service.
sns.countplot(x=df['On-board service'], hue=satisfactionForGraphing)
plt.title('Satisfaction based on Ratings for On-board Service')
plt.show()
```

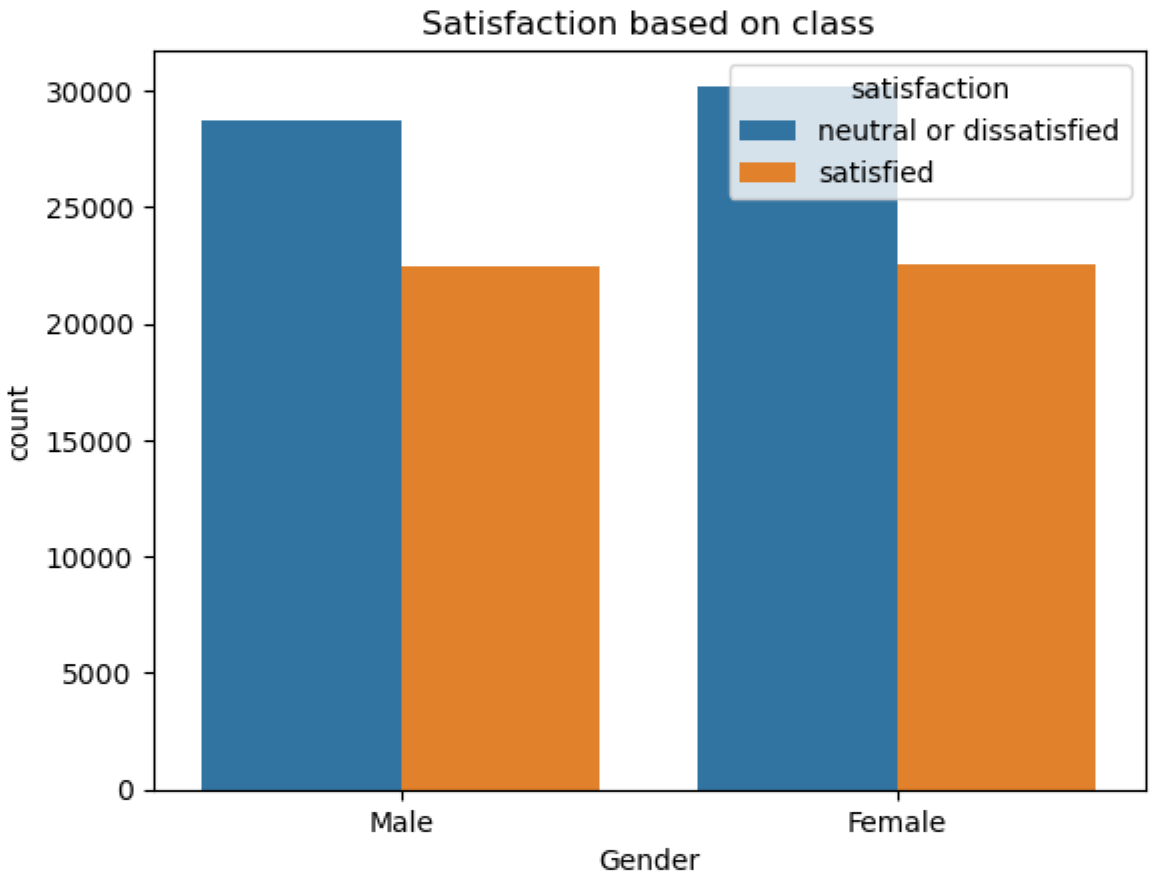


In [648…
```
# Passenger's satisfaction based off of Class
sns.countplot(x=classForGraphing, hue=satisfactionForGraphing)
plt.title('Satisfaction based on class')
plt.show()
```



In [649…
```
# Passenger's satisfaction based off of their gender. To see if the graphs are near equa
sns.countplot(x=genderForGraphing, hue=satisfactionForGraphing)
```

```
plt.title('Satisfaction based on class')
plt.show()
```

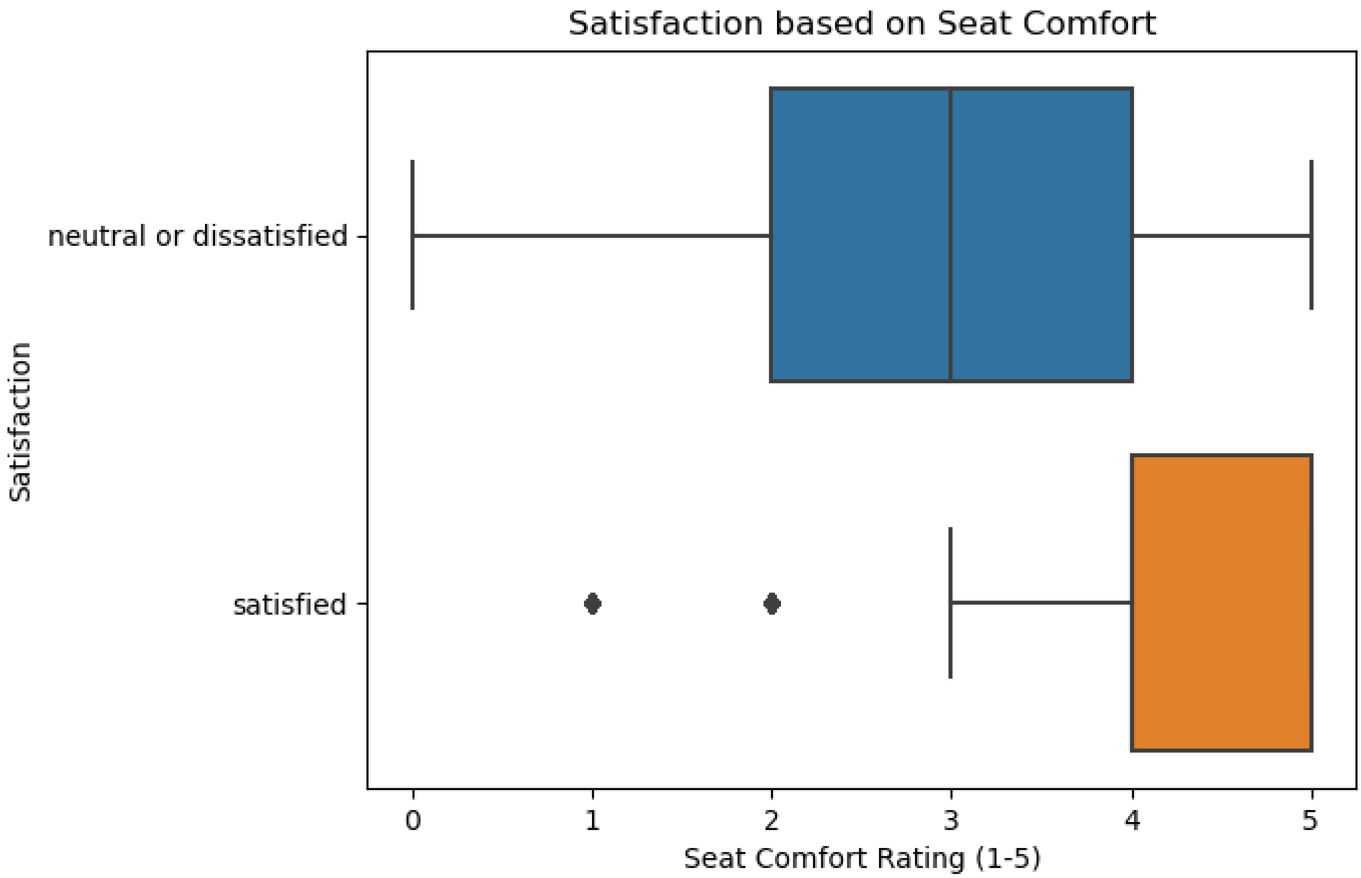


In [650…

```
# Passenger's Satisfaction based off of Seat Comfort Ratings
sns.boxplot(x=df['Seat comfort'], y=satisfactionForGraphing)
plt.title('Satisfaction based on Seat Comfort')
plt.xlabel('Seat Comfort Rating (1-5)')
plt.ylabel('Satisfaction')
```

Out[650]:

```
Text(0, 0.5, 'Satisfaction')
```



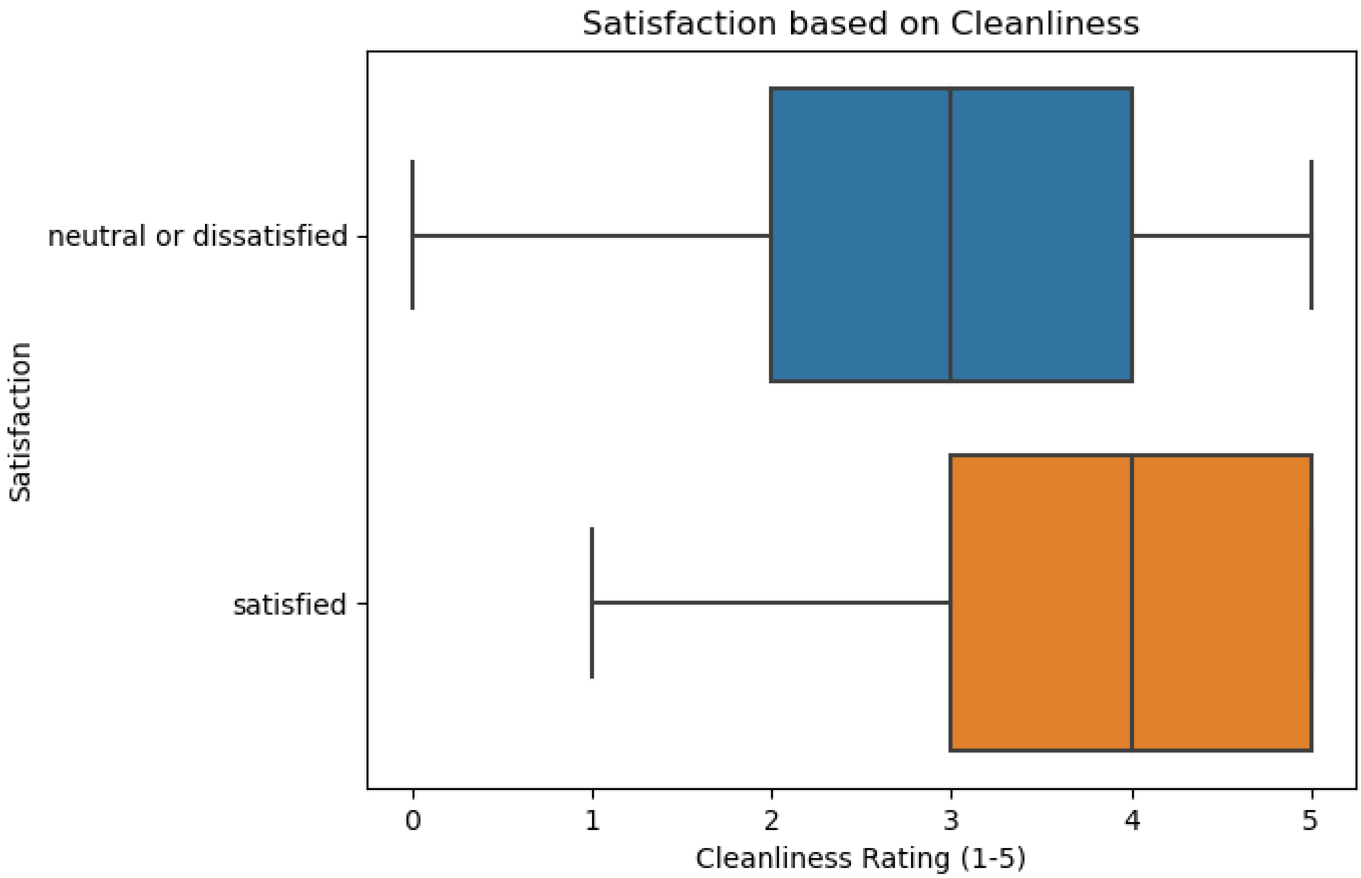
Based on the results, it appears that passengers who rated Seat Comfort between a 4-5 (high comfort) were satisfied compared to those who rated the seat comfort less than a 4, were not satisfied. There are two outliers that have chosen to be satisfied although they rated the seat comfort a 1 or 2.

In [651…

```
# Passenger's Satisfaction based off of Cleanliness Ratings
sns.boxplot(data=df, x=df['Cleanliness'], y=satisfactionForGraphing)
plt.xlabel('Cleanliness Rating (1-5)')
plt.ylabel('Satisfaction')
plt.title('Satisfaction based on Cleanliness')
```

Out[651]:

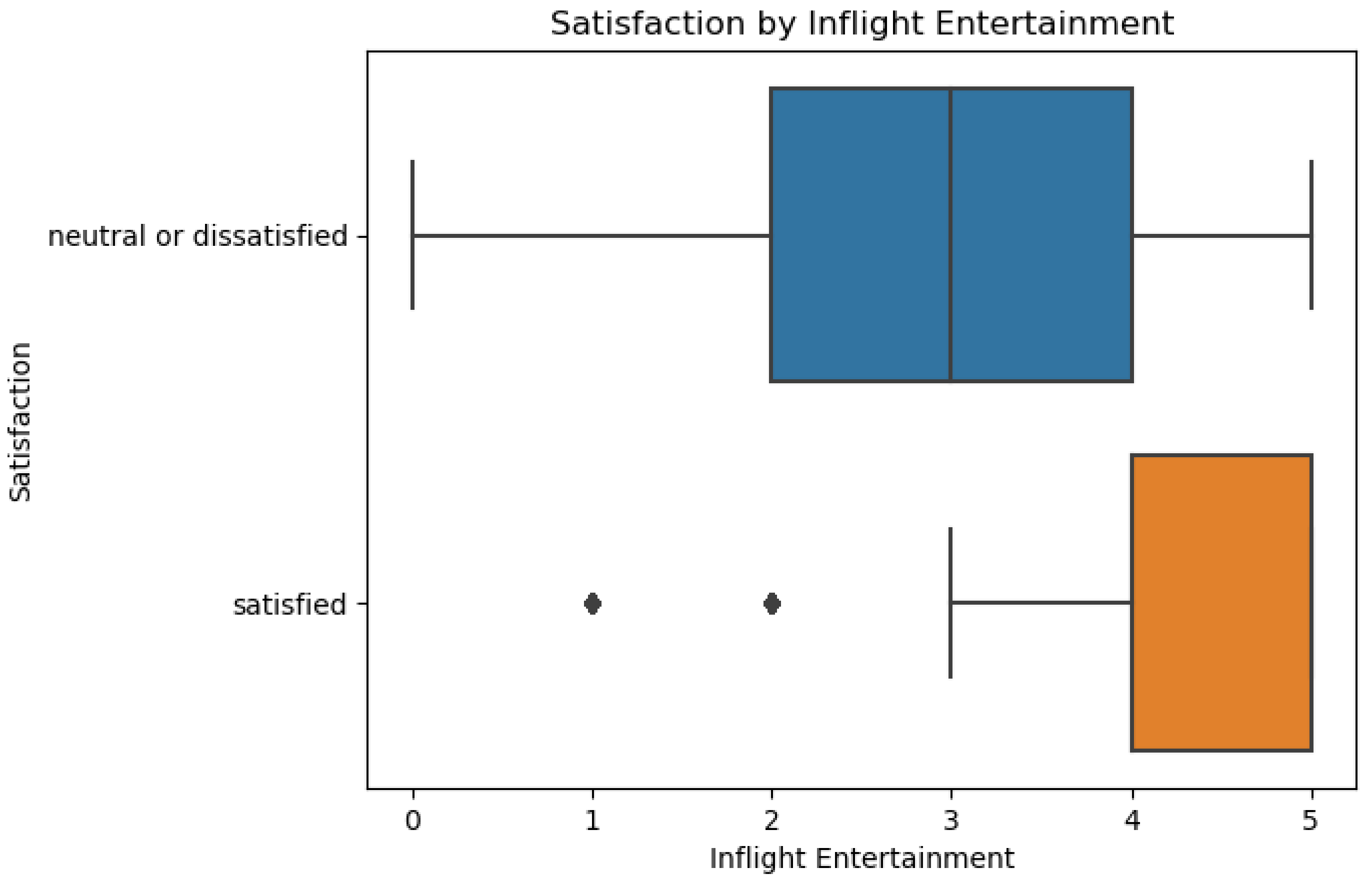
```
Text(0.5, 1.0, 'Satisfaction based on Cleanliness')
```



This plot shows the higher the cleanliness rating (3-5), the more likely satisfied the passenger was and the lower the cleanliness rating(1-~3), the more likely they weren't satisfied.

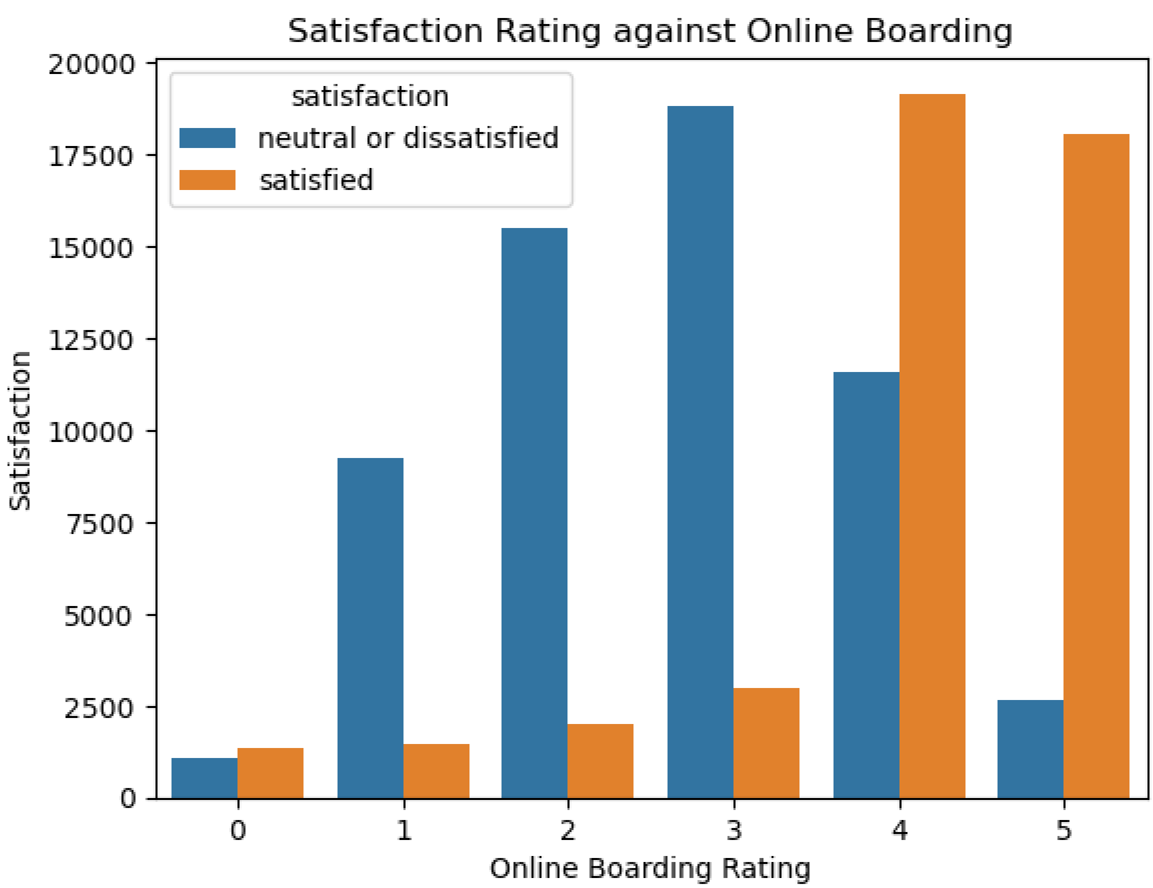
In [652…

```
# Passenger's Satisfaction based off of Inflight Entertainment
sns.boxplot(x='Inflight entertainment', y=satisfactionForGraphing, data=df)
plt.title('Satisfaction by Inflight Entertainment')
plt.xlabel('Inflight Entertainment')
plt.ylabel('Satisfaction')
plt.show()
```



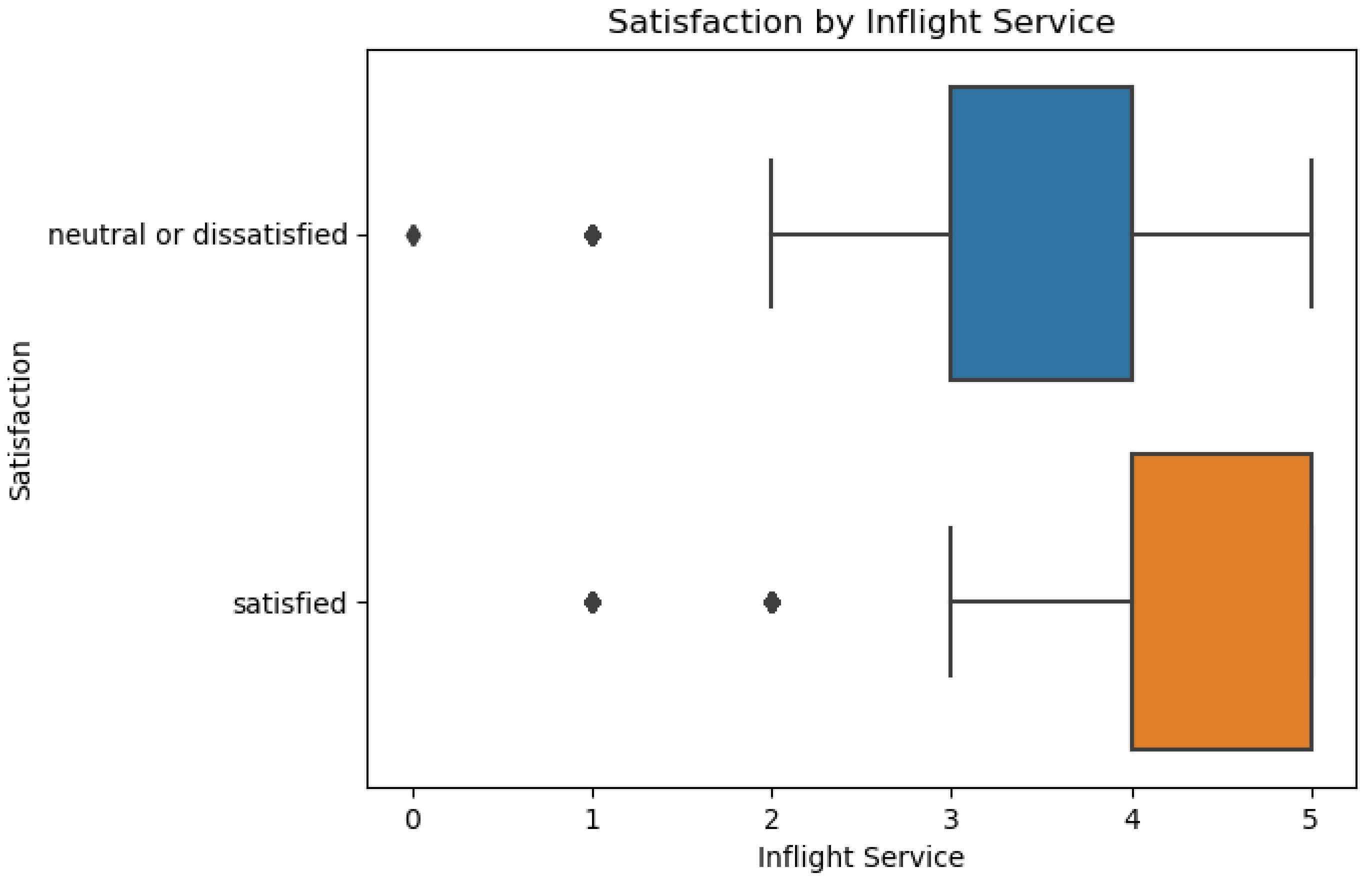
In [653…

```
# Checking overall satisfaction for all online boarders.
sns.countplot(x='Online boarding', hue=satisfactionForGraphing, data=df)
plt.title('Satisfaction Rating against Online Boarding')
plt.xlabel('Online Boarding Rating')
plt.ylabel('Satisfaction')
plt.show()
```



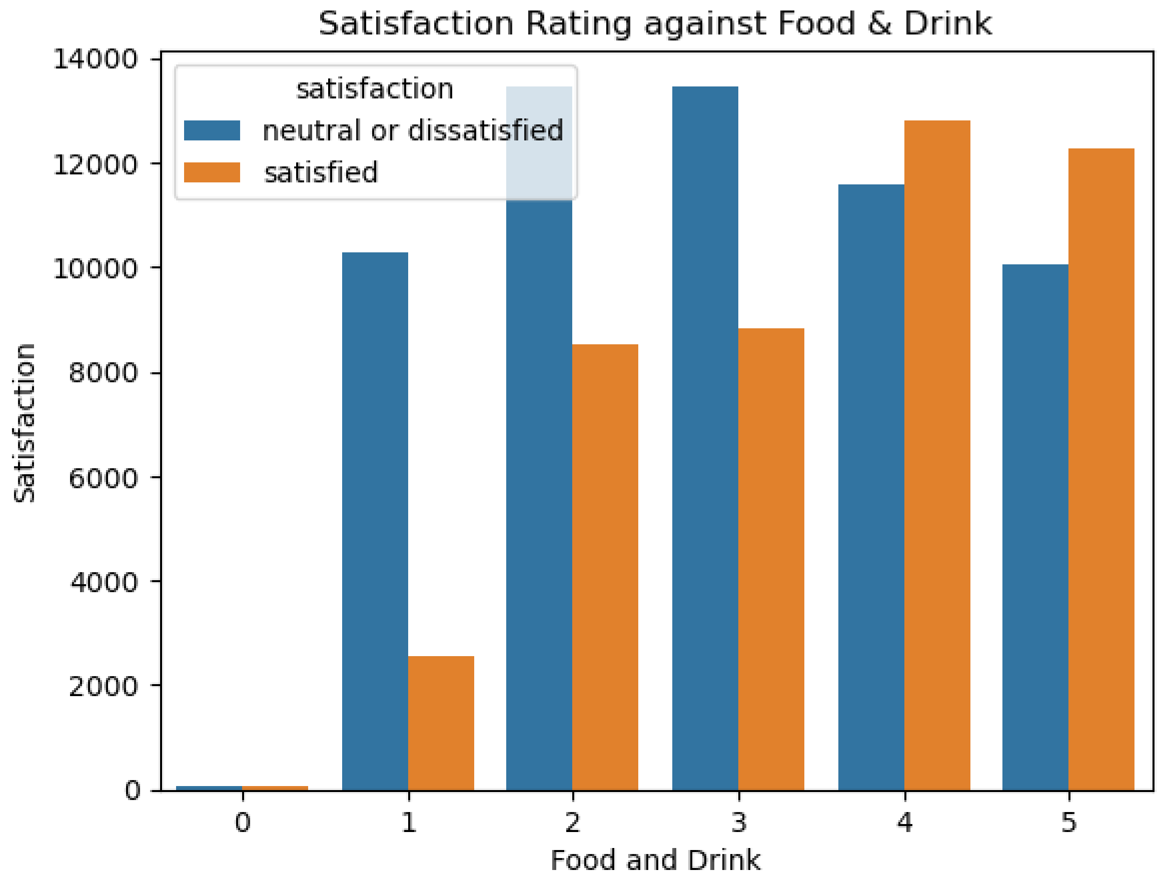
In [654…

```
# Passenger's Satisfaction based off of Inflight Service
sns.boxplot(x='Inflight service', y=satisfactionForGraphing, data=df)
plt.title('Satisfaction by Inflight Service')
plt.xlabel('Inflight Service')
plt.ylabel('Satisfaction')
plt.show()
```



In [655…

```
sns.countplot(x='Food and drink', hue=satisfactionForGraphing, data=df)
plt.title('Satisfaction Rating against Food & Drink')
plt.xlabel('Food and Drink')
plt.ylabel('Satisfaction')
plt.show()
```



Training

Back to Top

Since we have only two results in our *satisfaction* column (neutral or dissatisfied & satisfied), we are going to use either kNN Classifier or a Decision Tree Classifier for our project.

To prep for the actual machine learning, we'll prep our y_train & y_test so that we can reuse them in all subsequent trials without having to redeclare/tweak it.

In [656…
```
y_train = df['satisfaction'].values
y_test = dfTest['satisfaction'].astype('category').cat.codes
```

A helper function that simulates train_test_split but also with the added feature of normalizing our data. Also only works for X since we already did Y earlier.

In [657…
```
def x_train_test_split(predictors):
    scaler = StandardScaler()
    xPred = scaler.fit_transform(df[predictors].values)
    testPred = scaler.fit_transform(dfTest[predictors].values)
    return (xPred, testPred)
```

Also a note for these future cells, some of them take a generous amount of computational time due to large amounts of data neededing to be processed.

In [658…
```
predictors = ['Inflight entertainment', 'On-board service', 'Leg room service','Baggage
X_train, X_test = x_train_test_split(predictors)
```

```
knn = KNeighborsClassifier()
knn.fit(X_train, y_train)

predictions = knn.predict(X_test)
accuracy = (predictions == y_test).mean()
print(f'kNN 5-feature accuracy: {accuracy.round(2)}')
```

```
c:\Users\warre\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
kNN 5-feature accuracy: 0.73
```

Now that we have a simple kNN Classifier model setup, we decided to run another test with all the columns as potential predictors. We also decided to tweak the hyper-parameters to see if we could speed up the kNN model.

In [659…

```
dfCopy = df.copy()
dfCopy.drop('satisfaction', inplace=True, axis=1)
predictors = dfCopy.columns
X_train, X_test = x_train_test_split(predictors)

knn = KNeighborsClassifier(algorithm='brute')   # 25s
# knn = KNeighborsClassifier(algorithm='ball_tree') # over a min
# knn = KNeighborsClassifier(algorithm='kd_tree') # 40s
knn.fit(X_train, y_train)

predictions = knn.predict(X_test)
accuracy = (predictions == y_test).mean()
print(f'kNN all-feature accuracy: {accuracy.round(2)}')
```

```
c:\Users\warre\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:228: FutureWarning: Unlike other reduction functions (e.g. `skew`, `kurtosis`), the default behavior of `mode` typically preserves the axis it acts along. In SciPy 1.11.0, this behavior will change: the default value of `keepdims` will become False, the `axis` over which the statistic is taken will be eliminated, and the value None will no longer be accepted. Set `keepdims` to True or False to avoid this warning.
  mode, _ = stats.mode(_y[neigh_ind, k], axis=1)
kNN all-feature accuracy: 0.93
```

In [660…

```
dfCopy = df.copy()
dfCopy.drop('satisfaction', inplace=True, axis=1)
predictors = dfCopy.columns
X_train, X_test = x_train_test_split(predictors)

# tree = DecisionTreeClassifier(max_depth=2) # 0.86
tree = DecisionTreeClassifier(max_depth=4) # 0.89, close enough to kNN accuracy but keep
tree.fit(X_train, y_train)

predictions = tree.predict(X_test)
accuracy = (predictions == y_test).mean()
print(f'Decision Tree all-feature accuracy: {accuracy.round(2)}')
```

```
Decision Tree all-feature accuracy: 0.89
```

Considering the lengthy runtime of kNN compared to the decision trees (30-40s vs ~0.3s) we decided to use `Decision Trees` from now on as our main model going forward.

We'll start by investigating hyper parameters to improve our models accuracy. First, we'll tweak the `max_depth` and create a learning curve graph to pick out the optimal one.

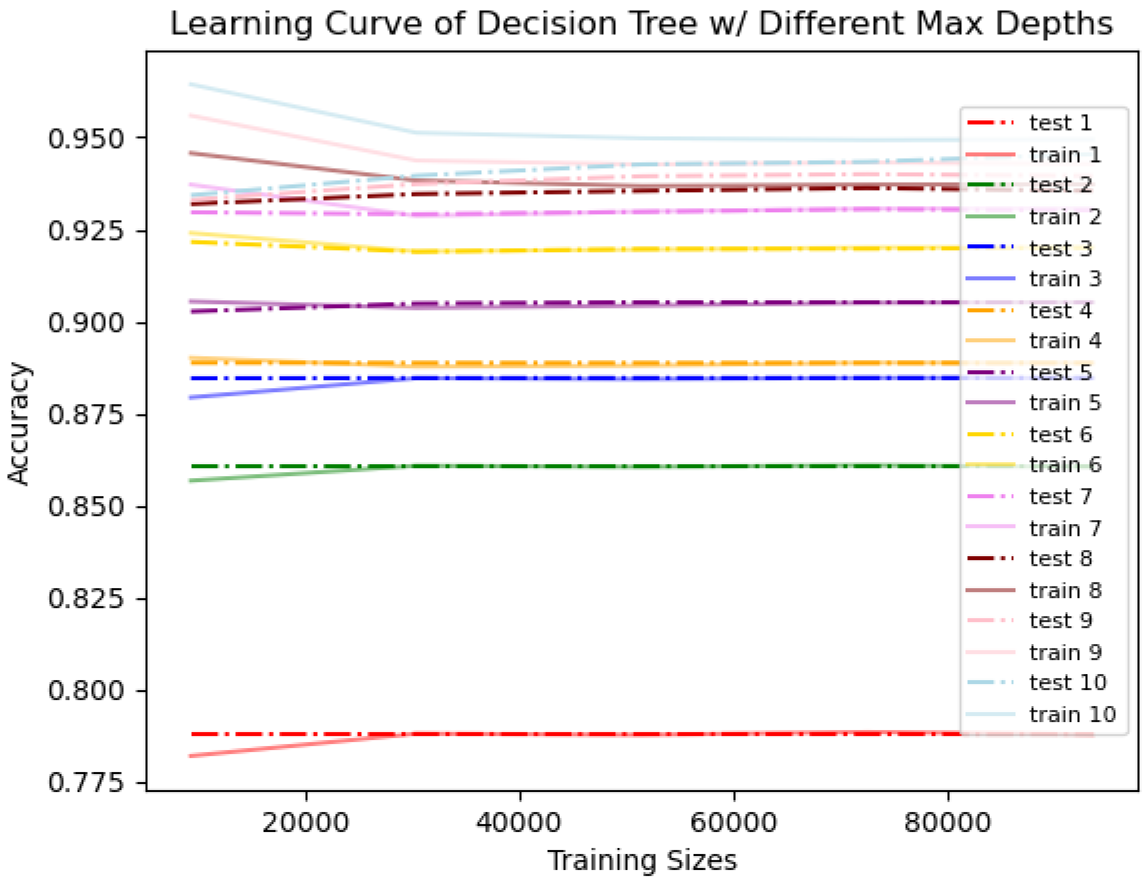
In [661…

```
# Code snippet from previous lab. Code to run plenty of tests with our model that'll map
k = 10
overallTE = []
overallTR = []
dfCopy = df.copy()
dfCopy.drop('satisfaction', inplace=True, axis=1)
predictors = dfCopy.columns
X_train, X_test = x_train_test_split(predictors)

for i in range(1, k+1, 1):
    knn = DecisionTreeClassifier(max_depth=i)
    te_errs = []
    tr_errs = []
    tr_sizes = np.linspace(100, X_train.shape[0], 10).astype(int)
    for tr_size in tr_sizes:
        # train model on a subset of the training data
        X_train1 = X_train[:tr_size,:]
        y_train1 = y_train[:tr_size]
        knn.fit(X_train1, y_train1)
        # Errors from Training & Test Data
        tr_predicted = knn.predict(X_train1)
        err = (tr_predicted != y_train1).mean()
        tr_errs.append(err)
        te_predicted = knn.predict(X_test)
        err = (te_predicted != y_test).mean()
        te_errs.append(err)
    # Calc the learning curve values and append them for later.
    tr_sizes, tr_errs, te_errs = learning_curve(
        knn, X_train, y_train, cv=10, scoring='accuracy')
    overallTR.append(np.mean(tr_errs, axis=1))
    overallTE.append(np.mean(te_errs, axis=1))
```

In [662…

```
# Same snippet from lab, but separated for easier tweaking of graphs.
# Make the resulting pairs "easier" to interpret
color = ['red', 'green', 'blue', 'orange', 'purple', 'gold', 'violet', 'maroon', 'pink',
k = 1
for i in range(0, len(overallTE), 1):
    plt.plot(tr_sizes, overallTE[i], label=f'test {k}', color=color[i], ls='dashdot')
    plt.plot(tr_sizes, overallTR[i], label=f'train {k}', color=color[i], linewidth=1.5,
    k = k+1
plt.legend(loc='right', prop={'size': 8})
plt.xlabel('Training Sizes')
plt.ylabel('Accuracy')
plt.title('Learning Curve of Decision Tree w/ Different Max Depths')
plt.show()
```



From this graph, we can see all the different gaps between the `max_depth` values and how the overall accuracy goes up as the depth increases. This can definitely be over fit if we go too far up, so we'll pick something in the middle with a small gap and then use that parameter for subsequent tests.

So going forward, `max_depth` will be set to *4*.

We had also experimented with `GridSearchCV` to test out potential hyper-parameter improvements, but not only did the cell take around a minute to run, the results didn't improve our prediction accuracy much so we stuck with the depth parameter only.

In [663…
```
dfCopy = df.copy()
dfCopy.drop('satisfaction', inplace=True, axis=1)
predictors = dfCopy.columns
X_train, X_test = x_train_test_split(predictors)

parameters = [{'min_samples_leaf': [0.1, 0.2, 0.3], 'max_leaf_nodes': [4, 8, 16]}]
tree = DecisionTreeClassifier(max_depth=4)
test = GridSearchCV(tree, parameters, scoring='accuracy', cv=10)
test.fit(X_train, y_train)

print(f'Our best score was: {test.best_score_} and the best params were {test.best_param
```

```
Our best score was: 0.8435478840845556 and the best params were {'max_leaf_nodes': 4, 'm
in_samples_leaf': 0.1}.
```

Now we have figured out our major hyper-parameter and can begin checking for the best predictors/features.

Finding best feature that has highest accuracy:

In [664…
```
dfCopy = df.copy()
dfCopy.drop(columns='satisfaction', axis=1, inplace=True)
predictors = dfCopy.columns
```

```
# Was unable to setup the proper file split here for some reason.
X_train_feat, X_test_feat, y_train_feat, y_test_feat = train_test_split(dfCopy, df['sati

colName = []
currentAccuracy = 0
for col in predictors:
    X_train_1 = X_train_feat[[col]]
    scores = cross_val_score(DecisionTreeClassifier(random_state = 42), X_train_1, y_tra
    accuracy = scores.mean()
    if (accuracy > currentAccuracy):
        currentAccuracy = accuracy
        colName = col
print('Best Feature: {}, Best Accuracy: {:.2f}%'.format(colName, currentAccuracy))
```

```
Best Feature: Online boarding, Best Accuracy: 0.79%
```

Top 10 combined features that have highest accuracy using forward feature search:

In [665…

```
dfCopy = df.copy()
dfCopy.drop(columns='satisfaction', axis=1, inplace=True)
predictors = dfCopy.columns
# Same as unable to setup the proper file split, but its fine for training
X_train_feat, X_test_feat, y_train_feat, y_test_feat = train_test_split(dfCopy, df['sati

remaining = list(predictors)
selected = []
n = 10

while len(selected) < n:
    currentAccuracy = 0
    colName = ''
    for feature in remaining:
        X_selected = X_train_feat[selected + [feature]]
        scores = cross_val_score(DecisionTreeClassifier(random_state = 42), X_selected,
        accuracy = scores.mean()
        if (accuracy > currentAccuracy):
            currentAccuracy = accuracy
            colName = feature

    remaining.remove(colName)
    selected.append(colName)
    print('Feature: {}, Accuracy: {:.2f}'.format(colName, currentAccuracy))
```

```
Feature: Online boarding, Accuracy: 0.79
Feature: Type of Travel_Business travel, Accuracy: 0.85
Feature: Inflight wifi service, Accuracy: 0.89
Feature: Gate location, Accuracy: 0.92
Feature: Baggage handling, Accuracy: 0.93
Feature: Customer Type_disloyal Customer, Accuracy: 0.94
Feature: Class_Business, Accuracy: 0.95
Feature: Inflight service, Accuracy: 0.95
Feature: Seat comfort, Accuracy: 0.95
Feature: Customer Type_Loyal Customer, Accuracy: 0.95
```

From the previous cell, we determined we only need around roughly ~90% accuracy, so we decided to use just the top 5 features instead of all 10.

In [666…

```
predictors = ['Online boarding', 'Type of Travel_Business travel', 'Inflight wifi servic
# True file split was achieved for the final model run
X_train, X_test = x_train_test_split(predictors)

tree = DecisionTreeClassifier(max_depth=4, random_state=42)
tree.fit(X_train, y_train)

predictions = tree.predict(X_test)
```

```
accuracy = (predictions == y_test).mean()
print(f'Final Decision Tree Accuracy: {accuracy.round(2)}%')
```

```
Final Decision Tree Accuracy: 0.88%
```

Conclusions

Back to Top

Here we test our model by adding in parameters (the predictors) that signaled a `dissatisfied` customer (Arrival Delay in Minutes, Baggage Handling, etc.) to see if our model would most predict an unhappy customer. We did the same with `satisfied` and got the output we expected.

In [667…

```
# Create a dictionary with the feature values for a single demo customer.
demoCustomer = {'Gender_Female': 0, 'Gender_Male': 1,
                 'Type of Travel_Business travel': 0,
            'Type of Travel_Personal Travel': 0, 'Class_Business': 0, 'Class_Eco': 1,
            'Class_Eco Plus': 0, 'Customer Type_Loyal Customer': 1, 'Customer Type_dislo
            'Age': 35, 'Flight Distance': 1000, 'Inflight wifi service': 1, 'Departure/A
            'Ease of Online booking': 4, 'Gate location': 1, 'Food and drink': 4, 'Onlin
            'Seat comfort': 3, 'Inflight entertainment': 4, 'On-board service': 4, 'Leg
            'Baggage handling': 0, 'Checkin service': 5, 'Inflight service': 5, 'Cleanli
            'Departure Delay in Minutes': 1000, 'Arrival Delay in Minutes': 1000}

# create a DataFrame with the new data
custDf = pd.DataFrame(demoCustomer, index=[0])
# get the predicted satisfaction value for the new customer
prediction = tree.predict(custDf[predictors].values)
# convert the predicted value to a string
satisfaction = 'satisfied' if prediction[0] == 1 else 'neutral/dissatisfied'

print(f'The prediction for our demo customer is: {satisfaction}')
```

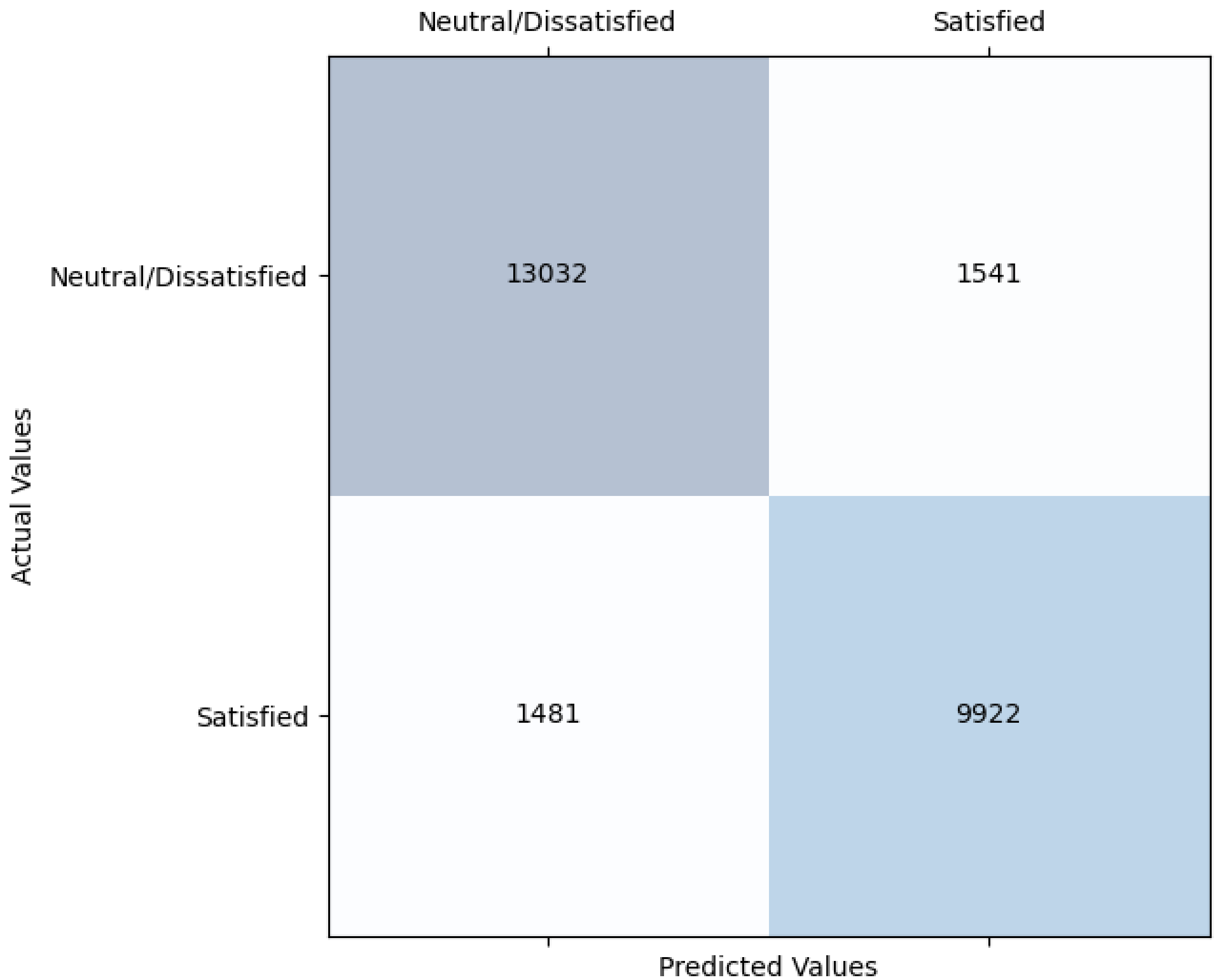
```
The prediction for our demo customer is: satisfied
```

Lastly, we have a confusion matrix to display our models accuracy.

In [668…

```
# Predictions here were from our final model before conclusion.
confusion = confusion_matrix(y_test, predictions)
# convert 0 to "neutral/dissatisfied", and 1 to "satisfied"
predictions = [0 if p==0 else 1 for p in predictions]
# convert 0 to "neutral/dissatisfied", and 1 to "satisfied"
y_test = [0 if y==0 else 1 for y in y_test]
# 1 if correct, 0 if incorrect
correct_predictions = [1 if p==t else 0 for p, t in zip(predictions, y_test)]
accuracy = sum(correct_predictions) / len(correct_predictions)
# Setup the matrix
fig, ax = plt.subplots(figsize=(6, 6))
ax.matshow(confusion, cmap=plt.cm.Blues, alpha=0.3)
for i in range(confusion.shape[0]):
    for j in range(confusion.shape[1]):
        ax.text(x=j, y=i, s=confusion[i, j], va='center', ha='center')
# Actually set the labels
tick_labels = ['Satisfied', 'Neutral/Dissatisfied']
ax.set_xticks([1, 0])
ax.set_yticks([1, 0])
ax.set_xticklabels(tick_labels)
ax.set_yticklabels(tick_labels)
# Axis & Display
plt.xlabel('Predicted Values')
```

```
plt.ylabel('Actual Values')
plt.show()
```



Our matrix results show our 88% accuracy rating, as our previous model results had outputted. With 1541 false positives (predicted satisfied but actually dissatisfied) and 1481 false negatives (actual satisfied but predicted dissatisfied) for a total of 3022 incorrect predictions out of 25976 customers.

In conclusion, our group had learned a great deal from this class and project, we experimented with `kNN` and `Decision Trees` and found that due to the speed of Trees, we got a decent accurate result at a fraction of the time that kNN would calculate, even with less predictors. Specific to our project, while many of us can assume features like `Online boarding`, `Gate Location`, `Inflight Wifi`, `Type of Travel_Business travel`, or `Baggage Handling` would be quality predictors for ones enjoyment of a flight; those were also different than our initial hypothesis (Seat comfort, in-flight entertainment, cleanliness, and food & drinks). Even though we had found 10 features that were really accurate together, we were able to derive the best 5 features from our those and our overall list of 26 and determined which ones were stronger predictors and more relevant.

Back to Top