Predicting Airline Customer Satisfaction

Group 1

Changa Fernando s3842381

Athan Katanos s3777056

Submission Mode: **Regular**

We hereby declare that this project report submission is an original research contribution.

Table of Contents

Introduction

Phase 1 Summary

Phase 1 started with finding a dataset. The dataset found was from www.kaggle.com and the specific data set relates the customer satisfaction surveys from airline passengers. The specific data set can be found at the following link: www.kaggle.com/teejmahal20/airline-passenger-satisfaction (Klein, 2020).

Next the dataset was inspected. The target feature was defined as the satisfaction. After that, the dataset was cleaned. To clean the dataset, the first two columns were removed, and secondly the dataset was scanned for missing values. It was found that there were 310 missing values in the *Arrival Delay in Minutes* column so the rows containing these observations were removed. Following the data cleaning, data visualisaton and data exploration were conducted.

When making 1 variable plots, *Satisfaction*, *Flight Distance* and *Age* were plotted. For **Figure 1**'s *Satisfaction*, it was found that more people are likely to be 'neutral or dissatisfied' rather than 'satisfied'. For **Figure 2**'s *Flight distance*, it was recognised that most people travel less than 1000 miles. For **Figure 3**'s *Age*, it was identified that mean of ages was 39 and the mean of ages is almost normally distributed.

When making 2 variable plots, there were three plots produced. **Figure 4** was a Violin plot of Satisfaction against Flight Distance. This plot showed that people who travel less than 1000 miles are more likely to be "neutral or dissatisfied", indicating that may affect the satisfaction of passengers. **Figure 5** was a Violin plot of Satisfaction against arrival delay and **Figure 6** was Violin plot of Satisfaction against Departure Delay. Both

of these plots show that the less time passengers are delayed in arrival and departure, the more likely they are to be 'satisfied'. There may be a strong correlation between delay and satisfaction.

When making 3 variable plots, there were three plots produced. **Figure 7** was a Violin plot of *Gender* against *Flight Distance* against *Satisfaction*. This plot indicates that gender does not appear to impact satisfaction as the violin plots for both 'Male' and 'Female' appear to be relatively the same. **Figure 8** shows a Box plot of *Flight Distance* against *Ease of Online Booking* against *Satisfaction*. This plot confirms what was found in **Figure 4** in that people who travel short distances are more likely to be 'neutral or dissatisfied'. *Ease of Online Booking* appears to no have no impact on *Satisfaction*. The last plot, **Figure 9**, shows a Scatter Plot of *Age* against *Flight Distance* against *Satisfaction* with kernel density estimates. While supporting what was found in **Figure 4**, *Age* was more difficult to analyse. Through the kernel density estimates, it appears that passengers aged between 40 to 60 are more likely to be 'satisfied' with their experience. The trend is unclear and it is unknown if *Age* has a noticeable impact on *Satisfaction*.

Phase 1 found that *Flight Distance* and *Age* may be significant in determining the *Satisfaction* of passengers. It is highly unlikely that *Gender*, *Arrival Delay*, *Departure Delay* and *Ease of Online Booking* contribute to customer satisfaction. Feature selection will determine if these features are important for determining customer satisfaction and will be done in section 1.8 of this report.

Report Overview

Phase 2 of Predicting Airline Customer Satisfaction is using what we have learnt from Phase 1 and applying it to various machine learning algorithms that can predict how any customer will or will not be satisfied by this US airline company. In doing so, we will also be using multiple methods of machine learning performance comparisons in order to best verify which algorithm is the most accurate and effective.

As previously identified in Phase 1, there are 103,903 rows and 25 columns. However, due to computational power constrictions, we will be reducing it to a randomly sampled 5000 rows and retaining the 25 original columns strictly for the machine learning. We will be using the full dataset for the feature selection predictions. More data will definitely be preferred to carry out any kind of machine learning algorithm. However, 5000 randomly sampled rows will be enough to conduct and carry out meaningful conclusions on the performance of these algorithms as well as produce significant findings.

Overview of Methodology

With our target feature of 'satisfaction', we're going to dive into the classifiers of:

- K-Nearest Neighbours (KNN)
- Random Forest (RF)
- Decision Tree (DT)

Before performing any kind of machine learning algorithm and performance comparisons, we will be cleaning the dataset, as performed in Phase 1. This report will include the further transformation step of one-hot encoding and scaling features of the necessary variables that will strengthen the machine learning algorithms later on - making the computer read our data easier. We will also be placing a large emphasis on the proportions of the target feature, which will be explained in its section.

We will then conducting feature selections on the full cleaned dataset as we're trying to identify which rating category carries the most weight in determining the satisfaction of a customer. Although there are different types of FS methods, we will be computing the Mutual Information and the F-Score feature selection algorithms for the best 10 features. We will then compute a comparison score by cross-validating each of the feature selections. The highest score identifies which FS algorithm is the best at predicting the rating category weight in the dataset.

We're going to be testing the dataset by splitting up the dataset into a training dataset and a testing dataset by randomly sampling the dataset into a 70:30 ratio respectively. As previously mentioned, we're randomly sampling 5000 rows to save computational power strictly for the machine learning. With this in mind, we will be demonstrating that:

- 3500 rows will be used to train the dataset
- 1500 rows will be used to test the dataset

The training dataset will be heavily involved in the hyper-parameter tuning phase of this report, whereas the testing dataset will be heavily involved in the performance comparison phase of this report.

In the hyper-parameter search portion of this report, we conduct a 5-fold stratified cross-validation to fine-tune hyper-parameters of each classifier (listed above) using the area under the receiver operating characteristic curve as the performance metric. As we're conducting these python codes from home, we're using our home devices which aren't as high-end as the machines at workplaces or at RMIT. With this, we build each model using parallel processing with "-2" cores which enables our devices to use all the CPU cores except for 1. This will allow us to comprehend/not freeze the machine learning algorithm, at the cost of some added computational time.

To identify which classifier has the best set of hyper-parameter values, we perform a grid search on the training dataset. After this, they're considered *tuned* classifiers. We then 'fit' these *tuned* classifiers onto the test dataset using 5-fold cross validation method, to grant a fair and unified performance test. With the performance characteristics, we can then perform a t-test to test each test's significance. In addition to this, we use a classification report and confusion matrix to help differentiate each classifier's performance.

Data Cleaning and Preprocessing

Much like Phase 1, we load the dataset in for initial cleaning and preprocessing.

In [1]:
```
import warnings
warnings.filterwarnings('ignore')
import pandas as pd
import numpy as np
data = pd.read_csv('Phase2_Group1.csv')
data.head()
```

Out[1]:

	Unnamed: 0	id	Gender	Customer Type	Age	Type of Travel	Class	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	...	ent
0	0	70172	Male	Loyal Customer	13	Personal Travel	Eco Plus	460	3	4	...	
1	1	5047	Male	disloyal Customer	25	Business travel	Business	235	3	2	...	
2	2	110028	Female	Loyal Customer	26	Business travel	Business	1142	2	2	...	
3	3	24026	Female	Loyal Customer	25	Business travel	Business	562	2	5	...	
4	4	119299	Male	Loyal Customer	61	Business travel	Business	214	3	3	...	

5 rows × 25 columns

From the table above the first two columns ('Unamed' and 'id') need to be removed as they are not descriptive features.

In [2]:
```
data.drop(['Unnamed: 0', 'id'], axis = 'columns', inplace = True)
data.head()
```

Out[2]:

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

5 rows × 23 columns

Now that the data set has been cleaned, missing values are checked for.

In [3]:
```
data.isna().sum()
```

Out[3]:
```
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            310
satisfaction                          0
dtype: int64
```

'Arrival Delay in Minutes' column has 310 missing values. The rows containing these columns will be deleted from the dataset.

In [4]:
```
data_no_na = data.dropna()
data_no_na.isna().sum()
```

Out[4]:
```
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
```

In [5]:
```
data_no_na.dtypes
```

Out[5]:
```
Gender                              object
Customer Type                       object
Age                                  int64
Type of Travel                      object
Class                               object
Flight Distance                      int64
Inflight wifi service                int64
Departure/Arrival time convenient    int64
Ease of Online booking               int64
Gate location                        int64
Food and drink                       int64
Online boarding                      int64
Seat comfort                         int64
Inflight entertainment               int64
On-board service                     int64
Leg room service                     int64
Baggage handling                     int64
Checkin service                      int64
Inflight service                     int64
Cleanliness                          int64
Departure Delay in Minutes           int64
Arrival Delay in Minutes           float64
satisfaction                        object
dtype: object
```

In [6]:
```
convert_dict = {'Arrival Delay in Minutes' : int}
Data2 = data_no_na.astype(convert_dict)
print(Data2.dtypes)
```

```
Gender                              object
Customer Type                       object
Age                                  int64
Type of Travel                      object
Class                               object
Flight Distance                      int64
Inflight wifi service                int64
Departure/Arrival time convenient    int64
Ease of Online booking               int64
Gate location                        int64
Food and drink                       int64
Online boarding                      int64
Seat comfort                         int64
Inflight entertainment               int64
On-board service                     int64
Leg room service                     int64
Baggage handling                     int64
Checkin service                      int64
Inflight service                     int64
Cleanliness                          int64
Departure Delay in Minutes           int64
Arrival Delay in Minutes             int32
satisfaction                        object
dtype: object
```

Now that the dataset has been cleaned, one-hot encoding and feature scaling can be done on the appropriate variables.

In [7]:
```
from sklearn import preprocessing
target = Data2.iloc[:, -1]

Data3 = Data2.iloc[:, :-1]
Data3 = pd.get_dummies(Data3)
Data_cols = Data3.columns
```

```
Data3 = pd.DataFrame(preprocessing.MinMaxScaler().fit_transform(Data3), columns=Data_cols)
Data3.head()
```

Out[7]:

	Age	Flight Distance	Inflight wifi service	Departure/Arrival time convenient	Ease of Online booking	Gate location	Food and drink	Online boarding	Seat comfort	Inflight entertainment
0	0.076923	0.086632	0.6	0.8	0.6	0.2	1.0	0.6	1.0	1.0
1	0.230769	0.041195	0.6	0.4	0.6	0.6	0.2	0.6	0.2	0.2
2	0.243590	0.224354	0.4	0.4	0.4	0.4	1.0	1.0	1.0	1.0
3	0.230769	0.107229	0.4	1.0	1.0	1.0	0.4	0.4	0.4	0.4
4	0.692308	0.036955	0.6	0.6	0.6	0.6	0.8	1.0	1.0	0.6

5 rows × 27 columns

All the variables that require one-hot encoding, such as Gender, Type of Travel, and Class, have been encoded correctly. On top of that, all the variables that require feature scaling, which are all the variables that have ratings 0-5, have been scaled correctly.

We now have a clean and usable dataset ready for machine learning algorithms.

In [8]:
```
print(target.value_counts())
```

```
neutral or dissatisfied    58697
satisfied                  44897
Name: satisfaction, dtype: int64
```

In [9]:
```
le = preprocessing.LabelEncoder()
le_fit = le.fit(target)
target_encoded_le = le_fit.transform(target)
```

In [10]:
```
print("Counts Using NumPy:")
print(np.unique(target_encoded_le, return_counts = True))
```

```
Counts Using NumPy:
(array([0, 1]), array([58697, 44897], dtype=int64))
```

From these codes, we can make the assumption that the target value is split evenly across the full dataset. This will be useful for the machine learning algorithms to be trained for both outcomes of satisfaction evenly. Meaning, there are no biases in the machine learning models.

Predictive Modelling

- Feature Selection
- Model Fitting and Tuning
- Model Comparison

Feature Selection (FS)

The process of deciding the most significant features of a dataset that influences the outcome of the target feature can be calculated in multiple ways. In this section, we will be selecting the 10 best features via the following methods:

- Mutual Information
- F-Score

We will then determine which method performs best.

Feature Selection using Mutual Information

The Mutual Information method will compare the relationship between each rating category with the target feature of satisfaction using the concept of entropy.

The code below will select the best 10 features that have the highest mutual information value. A histogram has been plotted to signal how each category compares to each other.

In [11]:
```
from sklearn.tree import DecisionTreeClassifier
clf = DecisionTreeClassifier(random_state=999)

from sklearn.model_selection import cross_val_score, StratifiedKFold

cv_method = StratifiedKFold(n_splits=5, shuffle=True, random_state=999)
scoring_metric = 'accuracy'
```

In [12]:
```
from sklearn import feature_selection as fs

num_features = 10

fs_fit_mutual_info = fs.SelectKBest(fs.mutual_info_classif, k=num_features)
fs_fit_mutual_info.fit_transform(Data3, target)
fs_indices_mutual_info = np.argsort(fs_fit_mutual_info.scores_)[::-1][0:num_features]
```

Here are the 10 best features it produced:

In [13]:
```
best_features_mutual_info = Data3.columns[fs_indices_mutual_info].values
best_features_mutual_info
```

Out[13]:
```
array(['Online boarding', 'Inflight wifi service', 'Class_Business',
       'Type of Travel_Business travel', 'Type of Travel_Personal Travel',
       'Class_Eco', 'Inflight entertainment', 'Seat comfort',
       'Flight Distance', 'Leg room service'], dtype=object)
```

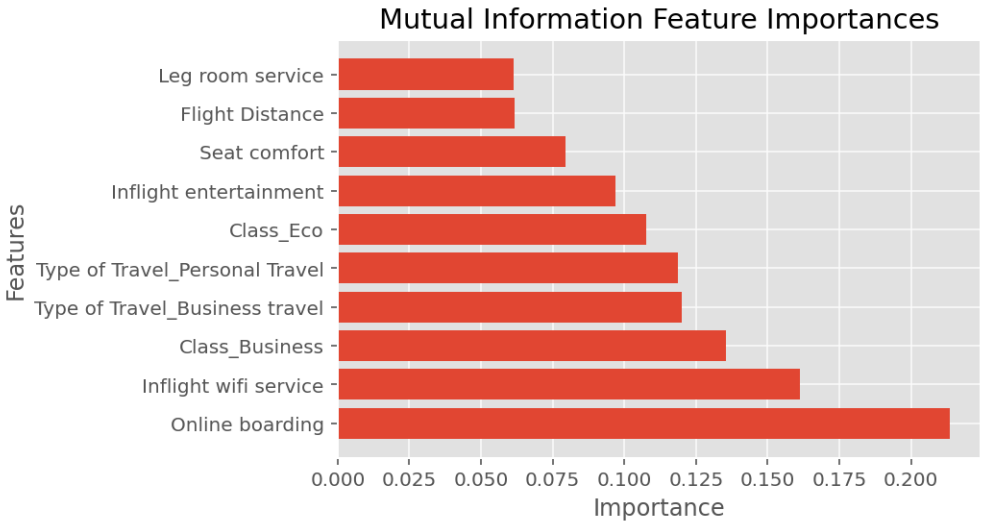
In [14]:
```
feature_importances_mutual_info = fs_fit_mutual_info.scores_[fs_indices_mutual_info]
feature_importances_mutual_info
```

Out[14]:
```
array([0.21372538, 0.16123227, 0.13563033, 0.12001187, 0.11858272,
       0.10773525, 0.09697464, 0.07961822, 0.06159915, 0.06132751])
```

In [15]:
```
import matplotlib.pyplot as plt
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
plt.style.use("ggplot")

def plot_imp(best_features, scores, method_name):
    plt.barh(best_features, scores)
    plt.title(method_name + ' Feature Importances')
    plt.xlabel("Importance")
    plt.ylabel("Features")
    plt.show()

plot_imp(best_features_mutual_info, feature_importances_mutual_info, 'Mutual Information')
```



From the above graph it is evident that the most important feature is online boarding (0.213), followed by inflight wifi service (0.162), and if the customer has traveled in business class (0.136).

Now the performance of the mutual information method will be tested using cross validation with the decision tree classifier.

In [16]:
```
cv_results_mutual_info  = cross_val_score(estimator=clf,
                            X=Data3.iloc[:, fs_indices_mutual_info],
                            y=target,
                            cv=cv_method,
                            scoring=scoring_metric)
cv_results_mutual_info.mean().round(3)
```

Out[16]:
```
0.909
```

The cross validation shows that the accuracy of the mutual information importance has an average of 91% with the 10 best features.

Feature Selection using F-Score

The F-Score method will compare the relationship between each rating category with the target feature of satisfaction using the F-Distribution.

The code below will select the best 10 features that have the highest F-Score. A histogram has been plotted to signal how each category compares to each other.

We also convert any 'NaN' value to zero, as the F-Score produces these values as a result of its technique.

In [17]:
```
from sklearn import feature_selection as fs
fs_fit_fscore = fs.SelectKBest(fs.f_classif, k=num_features)
fs_fit_fscore.fit_transform(Data3, target)
fs_indices_fscore = np.argsort(np.nan_to_num(fs_fit_fscore.scores_))[::-1][0:num_features]
```

Here are the 10 best features it produced:

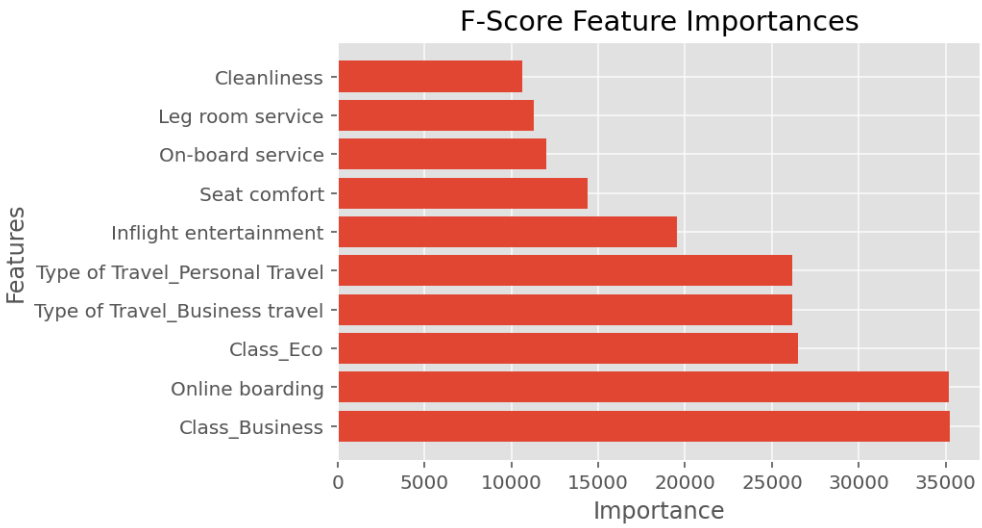
In [18]:
```
best_features_fscore = Data3.columns[fs_indices_fscore].values
best_features_fscore
```

Out[18]:
```
array(['Class_Business', 'Online boarding', 'Class_Eco',
       'Type of Travel_Business travel', 'Type of Travel_Personal Travel',
       'Inflight entertainment', 'Seat comfort', 'On-board service',
       'Leg room service', 'Cleanliness'], dtype=object)
```

In [19]:
```
feature_importances_fscore = fs_fit_fscore.scores_[fs_indices_fscore]
feature_importances_fscore
```

Out[19]:
```
array([35263.68611526, 35170.63342419, 26477.19375366, 26156.91806264,
       26156.91806264, 19521.61547268, 14378.12032998, 12020.7425102 ,
       11265.60989663, 10628.85476075])
```

In [20]:
```
plot_imp(best_features_fscore, feature_importances_fscore, 'F-Score')
```



From the above graph it is evident that the most important feature is if the customer has traveled in business class (35263.69), followed by online boarding (35170.63), and if the customer has traveled in economy class (26477.19).

Now the performance of the F-Score method will be tested using cross validation with the decision tree classifier.

In [21]:
```
cv_results_fscore = cross_val_score(estimator=clf,
                                    X=Data3.iloc[:, fs_indices_fscore],
                                    y=target,
                                    cv=cv_method,
                                    scoring=scoring_metric)
cv_results_fscore.mean().round(3)
```

Out[21]:
```
0.902
```

The cross validation shows that the accuracy of the F-Score importance has an average of 90% with the 10 best features.

Comparing feature selections

In [22]:
```
print('F-Score:', cv_results_fscore.mean().round(3))
print('Mutual Information:', cv_results_mutual_info.mean().round(3))
```

```
F-Score: 0.902
Mutual Information: 0.909
```

From this, we can see that the Mutual Information method just out performs the F-Score method. We can also see that the Mutual Information's performance score of 91% and beats the F-Score's performance score of 90%.

A paired t-test will be conducted to see if there is a significant differnece in these results.

In [23]:
```
from scipy import stats
print(stats.ttest_rel(cv_results_mutual_info,cv_results_fscore).pvalue.round(3))
```

```
0.002
```

Since the p-value is less than 0.05, we can conclude that the mutual information feature selections is statistically better than the feature selection

Model Fitting and Tuning

Sampling the Data

As stated earlier, we're going to reduce the current dataset of 103,903 rows to 5000 as it's computationally expensive to run the full dataset. To eliminate the bias in row selection, we're going to randomly select the number of samples for both the data and the target feature.

In [24]:
```
n_samples = 5000

Data_sample = pd.DataFrame(Data3).sample(n=n_samples, random_state=999).values
target_sample = pd.DataFrame(target).sample(n=n_samples, random_state=999).values

print(Data_sample.shape)
print(target_sample.shape)
```

```
(5000, 27)
(5000, 1)
```

Train-Test Split

Now that we've successfully reduced the dataset to a computationally safe amount, we can now partition the dataset into a training set and a testing set.

Below the data will be split into training and testing partitions. 70% of the data will be used for training and 30% of the data will be used for testing

In [25]:
```
from sklearn.model_selection import train_test_split

Data_sample_train, Data_sample_test, \
target_sample_train, target_sample_test = train_test_split(Data_sample, target_sample,
                                                           test_size = 0.3, random_state=999,
                                                           stratify = target_sample, shuffle=True)

print(Data_sample_train.shape)
print(Data_sample_test.shape)
print(target_sample_train.shape)
print(target_sample_test.shape)
```

```
(3500, 27)
(1500, 27)
(3500, 1)
(1500, 1)
```

Using a K-Nearest Neighbor Classifier

The following code uses a nearest neighbor classifier with 2 neighbors using the Euclidean distance.

In [26]:
```
from sklearn.neighbors import KNeighborsClassifier

knn_classifier = KNeighborsClassifier(n_neighbors=2, p=2)
```

In [27]:
```
knn_classifier.fit(Data_sample_train, target_sample_train);
```

Now the training data will be tested on the testing data.

In [28]:
```
knn_classifier.score(Data_sample_test, target_sample_test)
```

Out[28]:
```
0.876
```

The nearest neighbor classifier scores an accuracy rate of 0.88% on the test data.

Hyper Parameter Tuning for K-Nearest Neighbor

The following code is now hyper parameter tuning for K-Nearest Neighbors. This search will consider K values between 5 and 10 and p values of 1 (Manhattan), 2 (Euclidean), and 5 (Minkowski). This will be done with a 3 folds.

In [29]:
```
params_KNN = {'n_neighbors': [1, 2, 3, 5, 8, 10],
              'p': [1, 2, 5]}
```

In [30]:
```
from sklearn.model_selection import StratifiedKFold, GridSearchCV

gs_KNN = GridSearchCV(estimator=KNeighborsClassifier(),
                      param_grid=params_KNN,
                      cv=cv_method,
                      n_jobs = -2,
                      verbose=1,
                      scoring='accuracy',
                      return_train_score=True)
```

In [31]:
```
gs_KNN.fit(Data_sample, target_sample);
```

```
Fitting 5 folds for each of 18 candidates, totalling 90 fits
```

In [32]:
```
gs_KNN.best_params_
```

Out[32]:
```
{'n_neighbors': 5, 'p': 1}
```

In [33]:
```
gs_KNN.best_score_
```

Out[33]:
```
0.9172
```

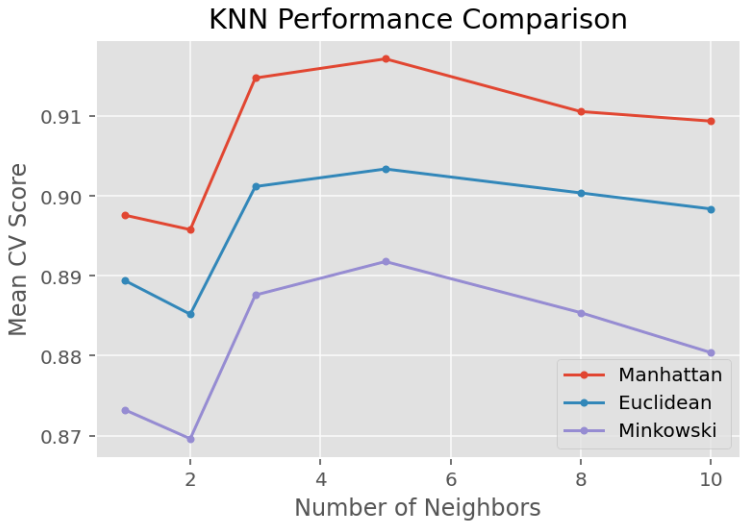
The best parameters for KNN are 5 neighbors using the Manhattan distance. This achieves an accuracy of 0.9172.

Let's visualise the KNN scores.

In [34]:
```
results_KNN = pd.DataFrame(gs_KNN.cv_results_['params'])
results_KNN['test_score'] = gs_KNN.cv_results_['mean_test_score']
results_KNN['metric'] = results_KNN['p'].replace([1,2,5], ["Manhattan", "Euclidean", "Minko
```

In [35]:
```
for i in ["Manhattan", "Euclidean", "Minkowski"]:
    temp = results_KNN[results_KNN['metric'] == i]
    plt.plot(temp['n_neighbors'], temp['test_score'], marker = '.', label = i)

plt.legend()
plt.xlabel('Number of Neighbors')
plt.ylabel("Mean CV Score")
plt.title("KNN Performance Comparison")
plt.show()
```



From the above, we can visualise how each distance method compares to one another. It also shows how the Manhattan distance uses 5 neighbors to achieve the score of 0.9172.

Decision Tree classifier

The following code uses a decision tree classifier with a max depth of 10 using the entropy criterion to maximise information gain.

In [36]:
```
dt_classifier = DecisionTreeClassifier(max_depth=10,
                                       criterion='entropy',
                                       random_state = 999)
```

In [37]:
```
dt_classifier.fit(Data_sample_train, target_sample_train);
```

In [38]:
```
dt_classifier.score(Data_sample_test, target_sample_test)
```

Out[38]:
```
0.9206666666666666
```

Using the decision tree classifier with a max depth of 10 and entropy as the information gain, the accuracy was 92% on the test data.

Hyper Parameter Tuning for Decision Tree Classifier

The following code is now hyper parameter tuning the decision tree classifier with the max depth ranging from 1 to 15, with the number of splits ranging from 1 to 5.

In [39]:
```
from sklearn.tree import DecisionTreeClassifier

df_classifier = DecisionTreeClassifier(random_state=999)

params_DT = {
             'criterion': ['gini', 'entropy'],
             'max_depth': range(1,15),
             'min_samples_split': range(1,5)}

gs_DT = GridSearchCV(estimator=df_classifier,
                     param_grid=params_DT,
                     cv=cv_method,
                     n_jobs = -2,
                     verbose=1,
                     scoring='accuracy')
```

In [40]:
```
gs_DT.fit(Data_sample, target_sample);
```

```
Fitting 5 folds for each of 112 candidates, totalling 560 fits
```

In [41]:
```
gs_DT.best_estimator_
```

Out[41]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=9, random_state=999)
```

In [42]:
```
gs_DT.best_score_
```

Out[42]:
```
0.9284000000000001
```

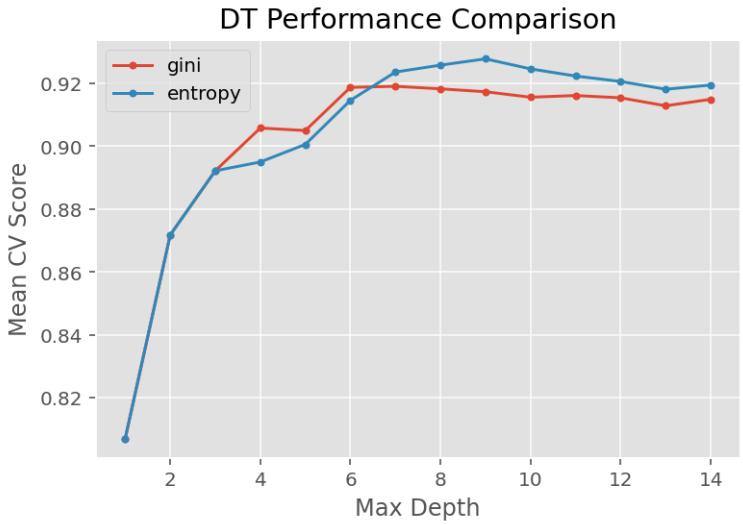
The best decision tree has maximum depth of 9 and uses entropy for information gain. It has an accuracy of 0.9284.

Let's visualise the grid search results.

In [43]:
```
results_DT = pd.DataFrame(gs_DT.cv_results_['params'])
results_DT['test_score'] = gs_DT.cv_results_['mean_test_score']
```

In [44]:
```
for i in ['gini', 'entropy']:
    temp = results_DT[results_DT['criterion'] == i]
    temp_average = temp.groupby('max_depth').agg({'test_score': 'mean'})
    plt.plot(temp_average, marker = '.', label = i)


plt.legend()
plt.xlabel('Max Depth')
plt.ylabel("Mean CV Score")
plt.title("DT Performance Comparison")
plt.show()
```



From the above, we can visualise how each information gain method compares to one another. It also shows how the entropy uses a max depth of 9 to achieve the score of 0.9284.

Random Forest Classifier

The following code uses a random forest classifier with the number of estimators 100.

In [45]:
```
from sklearn.ensemble import RandomForestClassifier

rf_classifier = RandomForestClassifier(n_estimators= 100, random_state=999)
```

In [46]:
```
rf_classifier.fit(Data_sample_train, target_sample_train)
```

Out[46]:
```
RandomForestClassifier(random_state=999)
```

In [47]:
```
rf_classifier.score(Data_sample_test, target_sample_test)
```

Out[47]:
```
0.9306666666666666
```

Using the random forest classifier with the number of estimators 100, the accuracy was 93% on the test data.

Hyper Parameter Tuning for Random Forest Classifier

The following code is now hyper parameter tuning the random forest classifier with n estimators being 100, 200, 500, 1000, 1500 and the max depth ranging from 10 to 21.

In [48]:
```
rf1_classifier = RandomForestClassifier(random_state=999)

params_RF = {'n_estimators': [100, 200, 500, 1000, 1500],
             'max_depth': range(10, 21)}

gs_RF = GridSearchCV(estimator=rf_classifier,
                     param_grid=params_RF,
                     cv=cv_method,
                     verbose=1,
                     n_jobs=-2,
                     scoring=scoring_metric)
```

In [49]:
```
gs_RF.fit(Data_sample, target_sample);
```

```
Fitting 5 folds for each of 55 candidates, totalling 275 fits
```

In [50]:
```
gs_RF.best_params_
```

Out[50]:
```
{'max_depth': 18, 'n_estimators': 1000}
```

In [51]:
```
gs_RF.best_score_
```

Out[51]:
```
0.9452
```

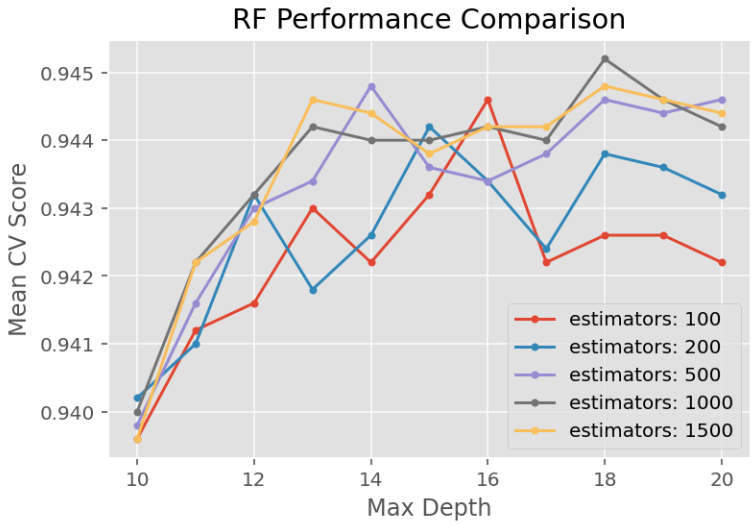
The best random forest has a number of estimators as 1000 with a max depth of 18. It has an accuracy of 0.9452.

Let's visualise the grid search results.

In [52]:
```
results_RF = pd.DataFrame(gs_RF.cv_results_['params'])
results_RF['test_score'] = gs_RF.cv_results_['mean_test_score']
```

In [53]:
```
for i in results_RF['n_estimators'].unique():
    temp = results_RF[results_RF['n_estimators'] == i]
    plt.plot(temp['max_depth'], temp['test_score'], marker = '.', label = 'estimators: ' +

plt.legend()
plt.xlabel('Max Depth')
plt.ylabel("Mean CV Score")
plt.title("RF Performance Comparison")
plt.show()
```



From the above, we can visualise how each number of estimators compare to one another. It also shows how the estimators of 1000 uses a max depth of 18 to achieve the score of 0.9452.

Model Comparison

During the Model Fitting and Tuning phase, we demonstrated how each of the parameters for each of the classifier parameters were manipulated to best fit the training set of 1,500 rows. Do note that this was 70% of the randomly sampled dataset of 5,000 rows.

Here's a quick summary of what we found with each of the hyper-parameter tuned classifiers:

- K-Nearest Neighbours: 5 neighbors using the Manhattan distance, 98% accuracy.
- Decision Tree: maximum depth of 9 with entropy method, 93% accuracy.
- Random Forest: 1000 estimators with a max depth of 18, 95% accuracy.

We're going to be performing a cross validation method with 5 splits on the testing set when using the hyper-parameter tuned classifiers. This will iron out any irregularities within the performance and select their best AUC score. Afterwards, we will perform multiple t-tests to identify which classifier works best. Finally, we will inspect each classifier's statistics using classification reports and confusion matrices. These results will help identify which classifier is best and further our findings.

Cross Validation

In [54]:
```
cv_method
```

Out[54]:
```
StratifiedKFold(n_splits=5, random_state=999, shuffle=True)
```

In [55]:
```
cv_results_KNN = cross_val_score(estimator=gs_KNN.best_estimator_,
                                 X=Data_sample_test,
                                 y=target_sample_test,
                                 cv=cv_method,
                                 scoring='roc_auc')
```

In [56]:
```
cv_results_DT = cross_val_score(estimator=gs_DT.best_estimator_,
                                X=Data_sample_test,
                                y=target_sample_test,
                                cv=cv_method,
                                scoring='roc_auc')
```

In [57]:
```
cv_results_RF = cross_val_score(estimator=gs_RF.best_estimator_,
                                X=Data_sample_test,
                                y=target_sample_test,
                                cv=cv_method,
```

```
                                    n_jobs=-2,
                                    scoring='roc_auc')
```

In [58]:
```
print('KNN AUC score:', cv_results_KNN.mean().round(3))
print('DT AUC score:', cv_results_DT.mean().round(3))
print('RF AUC score:', cv_results_RF.mean().round(3))
```

```
KNN AUC score: 0.941
DT AUC score: 0.925
RF AUC score: 0.977
```

The method with the best AUC score after cross validation is the Random Forest with an average AUC score of 0.977.

Performing Paired T tests

In [59]:
```
from scipy import stats
print(stats.ttest_rel(cv_results_RF, cv_results_KNN))
print(stats.ttest_rel(cv_results_RF, cv_results_DT))
print(stats.ttest_rel(cv_results_DT, cv_results_KNN))
```

```
Ttest_relResult(statistic=6.696875076725628, pvalue=0.0025865010220924006)
Ttest_relResult(statistic=6.793506671800106, pvalue=0.0024519265162140456)
Ttest_relResult(statistic=-1.5894205970124182, pvalue=0.18716769285892243)
```

From these outputs, we can detail that the RF/KNN and RF/DT are both significant as their p-values are less than 0.05, indicating these comparisons are statistically significant. With the 95% significant level, we can conclude that the common classifier of the Random Forest is the best model in terms of AUC when compared to the test data.

Granted we have proof of how significant each classifier's performance is when compared to each other, we can compute some of the following metrics to help further analysis:

- Accuracy
- Precision
- Recall
- F1 Score
- Confusion Matrix

Classification Report

In [60]:
```
pred_KNN = gs_KNN.predict(Data_sample_test)
```

In [61]:
```
pred_RF = gs_RF.predict(Data_sample_test)
```

In [62]:
```
pred_DT = gs_DT.predict(Data_sample_test)
```

In [63]:
```
from sklearn import metrics
print("\nClassification report for K-Nearest Neighbor")
print(metrics.classification_report(target_sample_test, pred_KNN))
print("\nClassification report for Random Forest")
print(metrics.classification_report(target_sample_test, pred_RF))
print("\nClassification report for Decision Tree")
print(metrics.classification_report(target_sample_test, pred_DT))
```

```
Classification report for K-Nearest Neighbor
                         precision    recall  f1-score   support

neutral or dissatisfied       0.92      0.97      0.94       838
              satisfied       0.96      0.89      0.92       662

               accuracy                           0.93      1500
```

```
                macro avg       0.94      0.93      0.93      1500
             weighted avg       0.93      0.93      0.93      1500


Classification report for Random Forest
                          precision    recall  f1-score   support

neutral or dissatisfied       1.00      1.00      1.00       838
              satisfied       1.00      1.00      1.00       662

               accuracy                           1.00      1500
              macro avg       1.00      1.00      1.00      1500
           weighted avg       1.00      1.00      1.00      1500


Classification report for Decision Tree
                          precision    recall  f1-score   support

neutral or dissatisfied       0.95      0.96      0.96       838
              satisfied       0.95      0.94      0.95       662

               accuracy                           0.95      1500
              macro avg       0.95      0.95      0.95      1500
           weighted avg       0.95      0.95      0.95      1500
```

From these classification reports for each of the classifiers, we can firmly identify that the Random Forest algorithm produced the perfect classification report, followed by the Decision Tree and the K-Nearest Neighbor. The Random Forest classifier scored the perfect 100% for every performance metric, meaning that this classifier is the best choice in detecting how each rating plays its part in determining satisfaction. Although the Decision Tree and K-Nearest Neighbor trail behind, they both achieve scores in the 90%-100% range. They both aren't terrible choices in providing accurate and precise results, but not as good as the Random Forest.

Confusion Matrices

In [64]:

```
from sklearn import metrics
print("\nConfusion matrix for K-Nearest Neighbor")
print(metrics.confusion_matrix(target_sample_test, pred_KNN))
print("\nConfusion matrix for Random Forest")
print(metrics.confusion_matrix(target_sample_test, pred_RF))
print("\nConfusion matrix for Decision Tree")
print(metrics.confusion_matrix(target_sample_test, pred_DT))
```

```
Confusion matrix for K-Nearest Neighbor
[[811  27]
 [ 73 589]]

Confusion matrix for Random Forest
[[838   0]
 [  0 662]]

Confusion matrix for Decision Tree
[[808  30]
 [ 39 623]]
```

From these matricies for each of the classifiers, we can draw the same conclusion as the classification reports - the Random Forest model provides the highest accuracy and precision. The classification reports use the confusion matricies to compute their precision and recall results. It uses formulas to generate proportions of a classifier's result accuracy.

The precision decimal describes how precise the results of the classifier is at producing truly positive results. The recall decimal describes the proportion of the true positives the classifier can find.

In this case, the Random Forest yielded 0 False Negatives and 0 False Positives which is exactly what we wanted.

Critique and Limitations

One of the benefits of having categorical features in a dataset is that decision trees and random forest work very well. Random forest is an extremely powerful classifier and in this case, it was a perfect classifier on the training data. Overall, the KNN, DT and Random Forest all had very good very good accuracy and AUC scores.

While the Random Forest was considered a perfect classifier, there may be other classifiers that exist that perform just as well on the training data and perform better on the test data. This would result in trialling more classifiers to see if they can perform better on the test data, which would result in more computation.

The obvious constraint of randomly sampling the original dataset of 103,903 rows to 5,000 rows can possibly play a significant role in the way each of the classifiers performed. That is, there still could have been a larger bias for certain rating categories for the flight. This can be true as we're dealing with less than 5% of what was originally given in the dataset. This bias can easily be removed if we were to increase the dataset we're training and testing with - which cannot happen due to computational performance constraints.

It is worth noting that there could be other features at play the affect customer satisfaction that are not features of this dataset such as seasonality, or availability of flights. This would require an extended survey with more questions.

Summary and Conclusion

Project Summary

In complete totality, Phases 1 and 2 of Predicting Airline Customer Satisfaction was a large success.

Phase 1

Phase 1 began with identifying the essential facts and cleaning the composition of the dataset. This would be identifying the target feature and cleaning the dataset. The target feature was defined as the satisfaction. To clean the dataset, the first two columns were removed, and secondly the dataset was scanned for missing values. It was found that there were 310 missing values in the *Arrival Delay in Minutes* column so the rows containing these observations were removed. Following the data cleaning, data visualisaton and data exploration were conducted.

Next came the plots for 1 variable. Singe variable plots were made for *Satisfaction*, *Flight Distance* and *Age*. For 2 variable plots, *Flight Distance*, *Arrival Delay* and *Departure Delay* were all plotted against *Satisfaction*. In the 3 variable plots section, there was a plot of *Gender* against *Flight Distance* against *Satisfaction*, a plot of *Flight Distance* against *Ease of Online Booking* against *Satisfaction*, and a plot of *Age* against *Flight Distance* against *Satisfaction*.

After the series of singular, dual and multi-variable plotting, Phase 1 found that *Flight Distance* and *Age* may be significant in determining the *Satisfaction* of passengers. It is highly unlikely that *Gender*, *Arrival Delay*, *Departure Delay* and *Ease of Online Booking* contribute to customer satisfaction. Feature selection will determine if these features are important for determining customer satisfaction, which has been explored in this report.

Phase 2

After the completion of Phase 1, Phase 2 began. Phase 2 started with the same cleaning as Phase 1, with the additional step of one-hot encoding. This prepared the data for the feature selections and hyper-parameter tuning of the machine learning algorithms. Next came the feature selections.

Using the Mutual Information and F-Score methods, we found that the Mutual Information method was the most effective in selecting which 10 features were most significant in predicting the customer satisfaction. This outcome was determined by a series of cross-validation scores using the decision tree method. To

determine if the cross-validation scores between the Mutual Information and F-Score were significant, a t-test was performed. Afterwards came the model fitting and hyper-parameter tuning.

The data was then partitioned into a training and testing set with 5,000 rows with a 70:30 ratio. The K-Nearest Neighbor, Random Forest and Decision Tree classifier algorithms were performed. After initialising each classifier with a singular set of parameters, we ran more tests on each of the classifiers. This was to identify which parameters best suit the classifier to increase its succession rate by fitting its training data to the training target. Each of the classifiers' performance for each change in parameters was calculated and plotted for visualisation. After conducting the hyper-parameter tuning for each of the models, comparisons were made.

Using cross-validations and AUC scores, t-tests, classification reports and confusion matricies on the partitioned test set, the Random Forest classifier remained the most dominant and promising machine learning algorithm to predict the likelihood of any customer's satisfaction.

Summary of Findings

The beginning of this Phase 2 report highlighted the feature selection algorithms. In this section, the following feature selection methods were used; Mutual Information, and F-Score. Through conducting cross-validation scores and each method's program, the Mutual Information method produced the most likely 10 features of the dataset with a score of 0.909 followed by the F-Score's 0.902. From the paired t-test, Mutual Information was statistically better method compared to F-score when selecting features. The 10 features that were selected in order of importance by Mutual Information goes as follows: *Online boarding, Inflight wifi service, Class_Business, Type of Travel_Business travel, Type of Travel_Personal Travel, Class_Eco, Inflight entertainment, Seat comfort, Leg room service* and *Flight Distance*. These features are determined as the most important in determining customer satisfaction.

The K-Nearest Neighbor classifier was trained using 1, 2, 3, 5, 8, 10 neighbors and the distance measuring of Manhattan, Euclidean and Minkowski. After using cross validation, the best parameters yielding the highest accuracy for the KNN classifier was using 5 neighbors with the Manhattan distance measuring. Its accuracy score was 0.9172.

The Decision Tree classifier was trained using the gini index and the entropy criteria with the max depth ranging from 1 to 15. After using cross validation, the best parameters yielding the highest accuracy for the DT classifier was using the entropy criteria with the depth of 9. Its accuracy score was 0.9284.

The Random Forest classifier was trained using 100, 200, 500, 1000, 1500 estimators and the max depth ranging from 10-21. After using cross validation, the best parameters yielding the highest accuracy for the RF classifer was 1000 estimators with the max depth of 18. Its accuracy score was 0.9452.

From the testing of hyper-parameter tuned classifiers and their comparisons, we found that the Random Forest is the best model in predicting the satisfaction of airline customers. The Random Forest scored almost perfect scores in the t-tests and the classification report metrics.

The hyper-parameter tuned Random Forest classifier had an accuracy score of 97%, with 100% precision and 100% recall.

Conclusion

Our main goal of this machine learning project was to predict this airline's customers' level of satisfaction using various models, algorithms and machine learning techniques. From our summaries and findings, we have strong confidence that we accomplished this goal.

We were able to visually model how the characteristics of some passengers influenced the likelihood of them being satisfied with their overall flight experience. We were able to use machine learning algorithms to identify which of these characteristics played the largest role in determining the satisfaction of a passenger.

We were able to hyper-parameter tune machine learning classifiers to fit the dataset, which will grant the airline company the chance of determining the likelihood of a customer feeling satisfied based on a few characteristics.

As stated in our findings, the Random Forest model was the most significant model compared to the others. We hope the airline company appreciates our work and uses it to develop a stronger relationship between them and their consumers.

References

- Klein, R. (2020, February). Airline Passenger Satisfaction Version 1. Retrieved August 6, 2021 from https://www.kaggle.com/teejmahal20/airline-passenger-satisfaction/metadata