Imports

In [2]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import matplotlib
from matplotlib import pyplot as plt
import matplotlib.pyplot as plt
from datetime import datetime
from sklearn.preprocessing import Imputer
from sklearn.decomposition import PCA
from sklearn import linear_model, decomposition, datasets
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split, cross_val_score,
from sklearn.ensemble import RandomForestClassifier
#from sklearn import cross_validation
from sklearn.metrics import confusion_matrix, roc_curve
from sklearn.neighbors import KNeighborsClassifier
from sklearn import metrics
from sklearn import preprocessing

from sklearn.model_selection import train_test_split
from sklearn import preprocessing
from sklearn import tree
from sklearn import datasets, linear_model
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
```

Read csv file

```
Data set Source from https://www.kaggle.com/arjhbholu/airline-dataset-mining
```

In [3]:
```
airlineData = pd.read_csv('Airline-Dataset.csv')
airlineData.head()
airlineData_orig = airlineData
```

Getting to know columns & Selecting required features

In [4]:
```
# Getting column names
column_names = airlineData.columns
print(column_names)
```

```
Index(['Unnamed: 0', 'airline_name', 'author', 'author_country', 'conten
t',
       'cabin_flown', 'overall_rating', 'seat_comfort_rating',
       'cabin_staff_rating', 'food_beverages_rating',
       'inflight_entertainment_rating', 'value_money_rating', 'Month', 'Y
ear',
       'recommended'],
      dtype='object')
```

In [5]:
```
# Create list comprehension of the columns you want to lose
columns_to_drop = [column_names[i] for i in [0, 2, 12]] # remove 'Unnam
# Dropping unwanted columns
airlineData.drop(columns_to_drop, inplace=True, axis=1)
```

In [6]:
```
airlineData.columns
```

Out[6]:
```
Index(['airline_name', 'author_country', 'content', 'cabin_flown',
       'overall_rating', 'seat_comfort_rating', 'cabin_staff_rating',
       'food_beverages_rating', 'inflight_entertainment_rating',
       'value_money_rating', 'Year', 'recommended'],
      dtype='object')
```

In [7]:
```
# checking data types of columns
airlineData.dtypes
```

Out[7]:
```
airline_name                    object
author_country                  object
content                         object
cabin_flown                     object
overall_rating                 float64
seat_comfort_rating            float64
cabin_staff_rating             float64
food_beverages_rating          float64
inflight_entertainment_rating  float64
value_money_rating             float64
Year                             int64
recommended                      int64
dtype: object
```

Checking null/missing values

In [8]:
```
# checking nulls
airlineData.isnull().sum()
```

Out[8]:
```
airline_name                    0
author_country                  0
content                         0
cabin_flown                     0
overall_rating                  0
seat_comfort_rating             0
cabin_staff_rating              0
food_beverages_rating           0
inflight_entertainment_rating   0
value_money_rating              0
Year                            0
recommended                     0
dtype: int64
```

Getting to know the airlineData dataframe

In [9]:
```
airlineData.describe()
```

Out[9]:

	overall_rating	seat_comfort_rating	cabin_staff_rating	food_beverages_rating	inflight_entertain
count	27284.000000	27284.000000	27284.000000	27284.000000	2
mean	6.067879	3.259566	3.522944	3.016493	
std	3.216066	1.351689	1.460053	1.515096	
min	1.000000	0.000000	0.000000	0.000000	
25%	3.000000	2.000000	2.000000	2.000000	
50%	7.000000	4.000000	4.000000	3.000000	
75%	9.000000	4.000000	5.000000	4.000000	
max	10.000000	5.000000	5.000000	5.000000	

In [10]:
```
1 airlineData.head()
```

Out[10]:

	airline_name	author_country	content	cabin_flown	overall_rating	seat_comfort_rating	cabin_st
0	adria-airways	Germany	Outbound flight FRA/PRN A319. 2 hours 10 min f...	Economy	7.0	4.0	
1	adria-airways	United States	Two short hops ZRH-LJU and LJU-VIE. Very fast ...	Business Class	10.0	4.0	
2	adria-airways	Switzerland	Flew Zurich-Ljubljana on JP365 newish CRJ900. ...	Economy	9.0	5.0	
3	adria-airways	Singapore	Adria serves this 100 min flight from Ljubljan...	Business Class	8.0	4.0	
4	adria-airways	Poland	WAW-SKJ Economy. No free snacks or drinks on t...	Economy	4.0	4.0	

In [11]:
```
1 airlineData.shape
```

Out[11]: (27284, 12)

Covariance & Correlation of features

In [12]:
```
1 # Covariance Matrix
2 airlineData.cov()
```

Out[12]:

	overall_rating	seat_comfort_rating	cabin_staff_rating	food_beverages_ra
overall_rating	10.343081	3.131338	3.642213	3.131
seat_comfort_rating	3.131338	1.827063	1.189771	1.128
cabin_staff_rating	3.642213	1.189771	2.131754	1.399
food_beverages_rating	3.131234	1.128549	1.399871	2.295
inflight_entertainment_rating	2.288042	0.953044	0.908524	1.291
value_money_rating	3.853208	1.376512	1.496718	1.328
Year	-0.167488	-0.073214	-0.068468	-0.053
recommended	1.373956	0.428596	0.506542	0.428

In [13]:
```
1 # Correlation Matrix
2 airlineData.corr()
```

Out[13]:

	overall_rating	seat_comfort_rating	cabin_staff_rating	food_beverages_ra
overall_rating	1.000000	0.720325	0.775661	0.642
seat_comfort_rating	0.720325	1.000000	0.602862	0.551
cabin_staff_rating	0.775661	0.602862	1.000000	0.632
food_beverages_rating	0.642614	0.551066	0.632819	1.000
inflight_entertainment_rating	0.419480	0.415727	0.366893	0.502
value_money_rating	0.831487	0.706743	0.711426	0.608
Year	-0.052101	-0.054188	-0.046915	-0.035
recommended	0.873473	0.648296	0.709330	0.577

Getting to know values of Airline Dataset

In [14]:
```
1 airlineData['cabin_flown'].unique()
```

Out[14]:
```
array(['Economy', 'Business Class', 'Premium Economy', 'First Class'],
      dtype=object)
```

In [15]:
```
1 airlineData['overall_rating'].unique()
```

Out[15]:
```
array([ 7., 10.,  9.,  8.,  4.,  5.,  2.,  3.,  1.,  6.])
```

In [16]:
```
1 airlineData['inflight_entertainment_rating'].unique()
```

Out[16]:
```
array([0., 1., 2., 3., 4., 5.])
```

In [17]:
```
1 airlineData['value_money_rating'].unique()
```

Out[17]:
```
array([4., 5., 2., 3., 1., 0.])
```

In [18]:
```
1 airlineData['Year'].unique()
```

Out[18]:
```
array([2015, 2014, 2013, 2012, 2011, 1970])
```

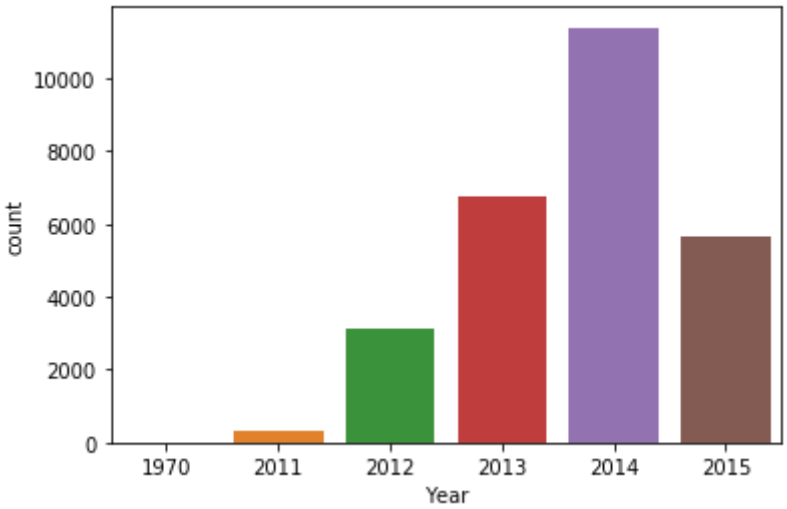
In [19]:
```
1 airlineData['recommended'].unique()
```

Out[19]:
```
array([1, 0])
```

Plotting Airline Dataset

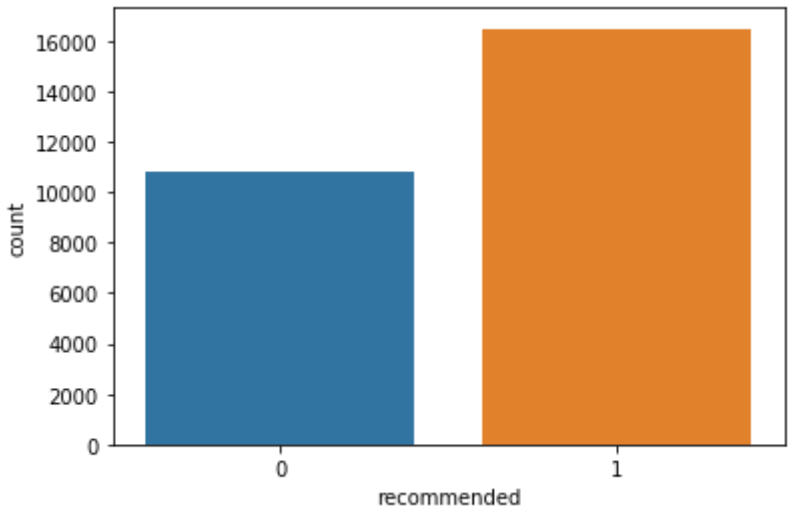
In [20]:
```
1 sns.countplot(x='Year',data=airlineData)
```

Out[20]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1a22331da0>
```



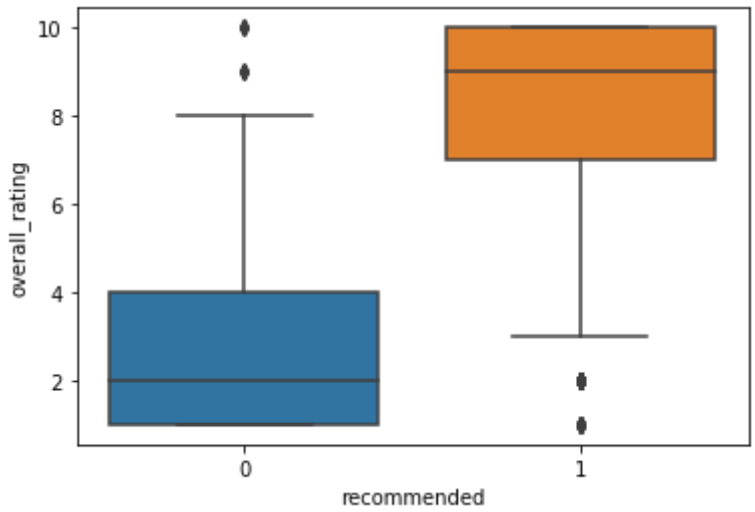
In [21]:
```
1 sns.countplot(x='recommended',data=airlineData)
```

Out[21]: <matplotlib.axes._subplots.AxesSubplot at 0x1a221bdd30>



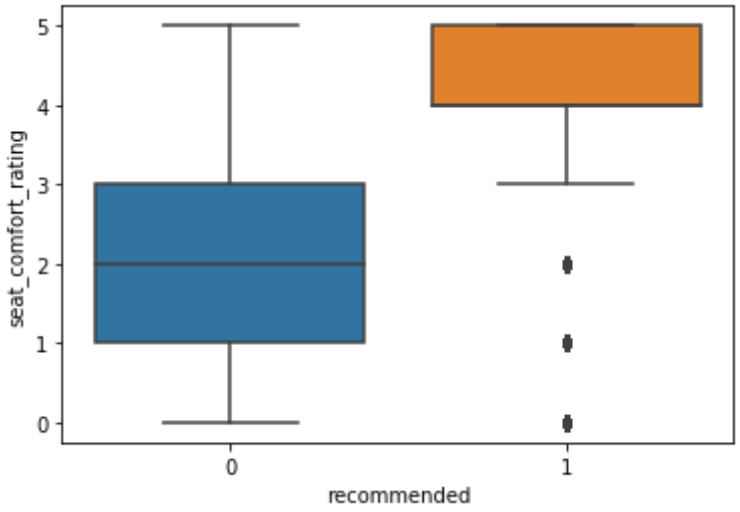
In [22]:
```
1 sns.boxplot(x='recommended',y='overall_rating',data=airlineData)
```

Out[22]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e3a3898>



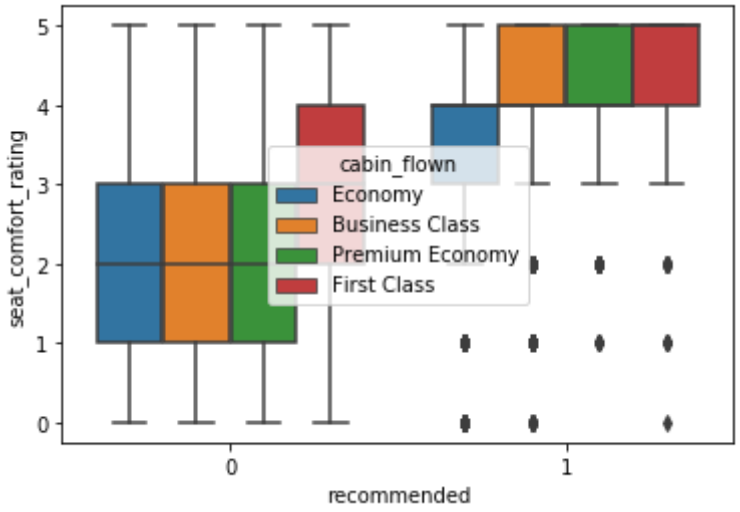
In [23]:
```
1 sns.boxplot(x='recommended',y='seat_comfort_rating',data=airlineData)
```

Out[23]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1dfede48>



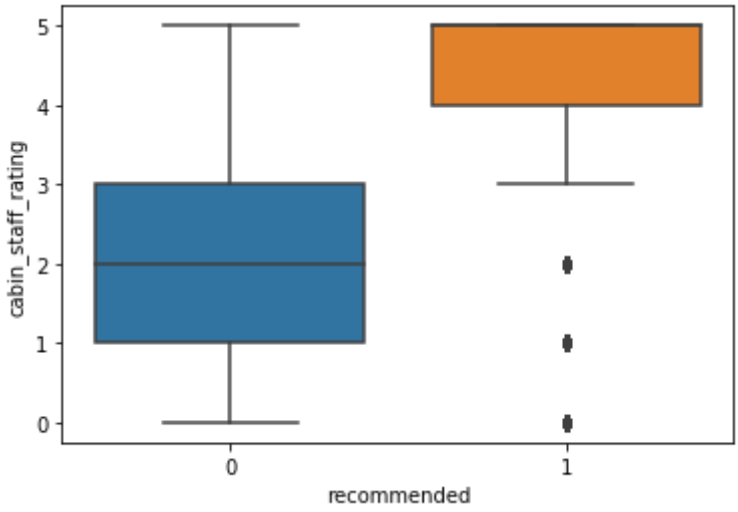
In [24]:
```
1 sns.boxplot(x='recommended',y='seat_comfort_rating',data=airlineData,hu
```

Out[24]: <matplotlib.axes._subplots.AxesSubplot at 0x10989b6d8>



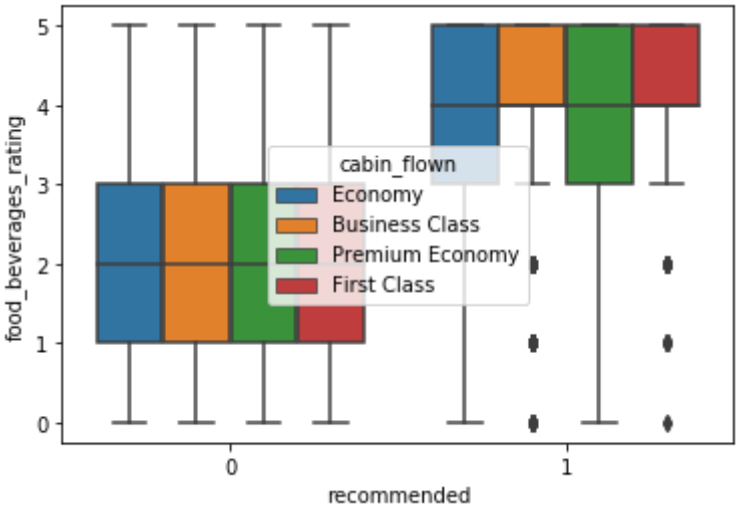
In [25]:
```
1 sns.boxplot(x='recommended',y='cabin_staff_rating',data=airlineData)
```

Out[25]: <matplotlib.axes._subplots.AxesSubplot at 0x1a201b5208>



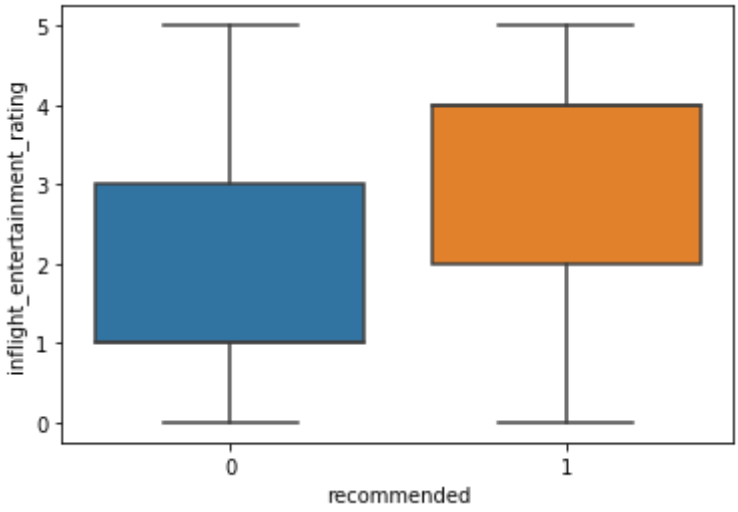
In [26]:
```
1 sns.boxplot(x='recommended',y='food_beverages_rating',data=airlineData,
```

Out[26]: <matplotlib.axes._subplots.AxesSubplot at 0x1a1e5ef6a0>



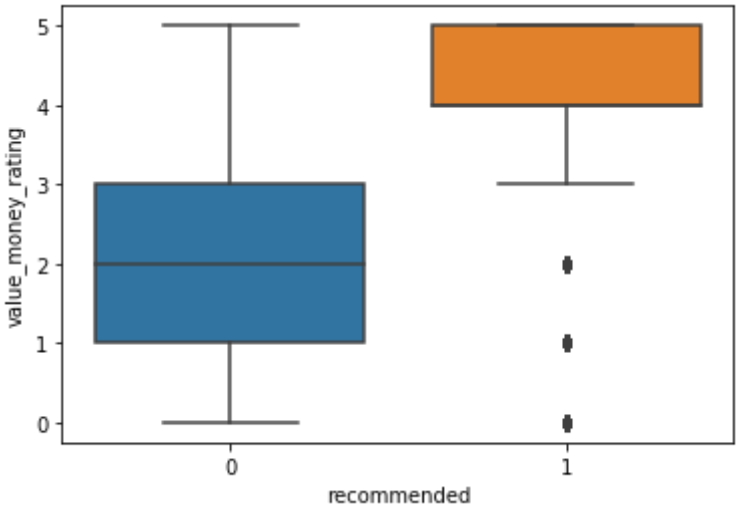
In [27]:
```
1 sns.boxplot(x='recommended',y='inflight_entertainment_rating',data=airl
```

Out[27]: <matplotlib.axes._subplots.AxesSubplot at 0x1a20286d68>



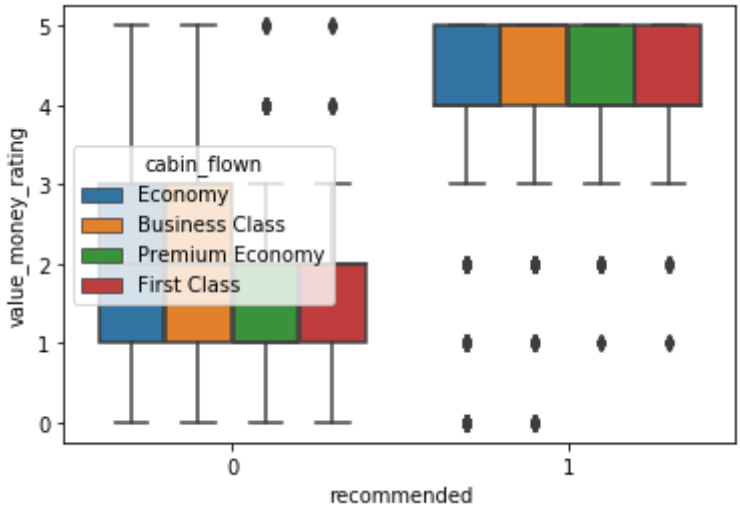
In [28]:
```
1 sns.boxplot(x='recommended',y='value_money_rating',data=airlineData)
```

Out[28]: <matplotlib.axes._subplots.AxesSubplot at 0x1a20354b38>



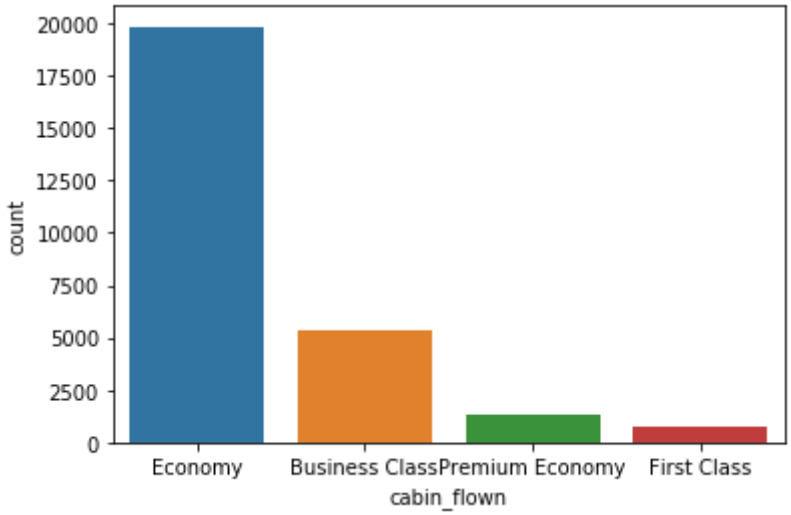
In [29]:
```
1 sns.boxplot(x='recommended',y='value_money_rating',data=airlineData,hue
```

Out[29]: <matplotlib.axes._subplots.AxesSubplot at 0x1a2043dda0>



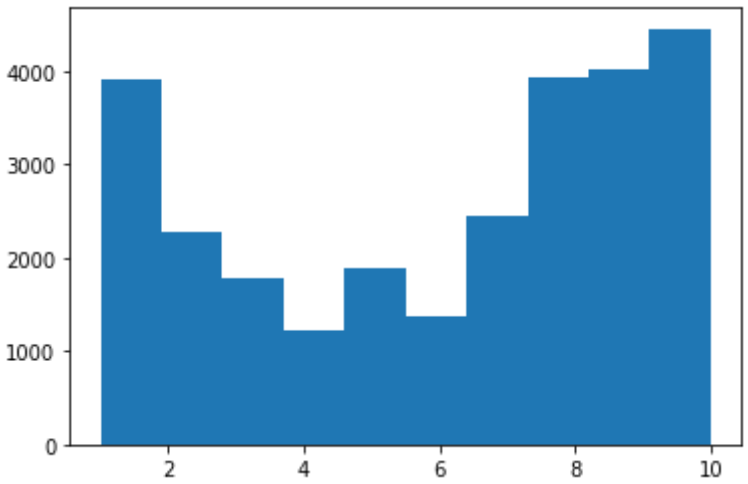
In [30]:
```
1 sns.countplot(x='cabin_flown',data=airlineData)
```

Out[30]: <matplotlib.axes._subplots.AxesSubplot at 0x1a205dacc0>



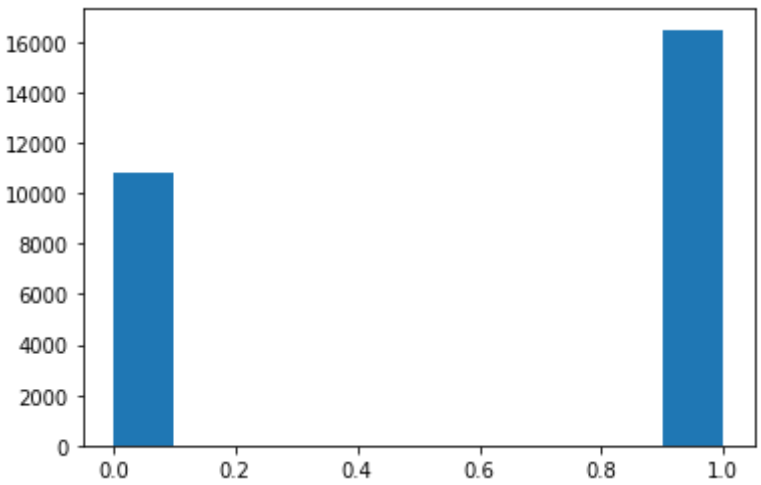
In [31]:
```
1 plt.hist(airlineData.overall_rating.values)
```

Out[31]:
```
(array([3906., 2271., 1782., 1227., 1885., 1369., 2451., 3932., 4008.,
        4453.]),
 array([ 1. ,  1.9,  2.8,  3.7,  4.6,  5.5,  6.4,  7.3,  8.2,  9.1, 10.
]),
 <a list of 10 Patch objects>)
```

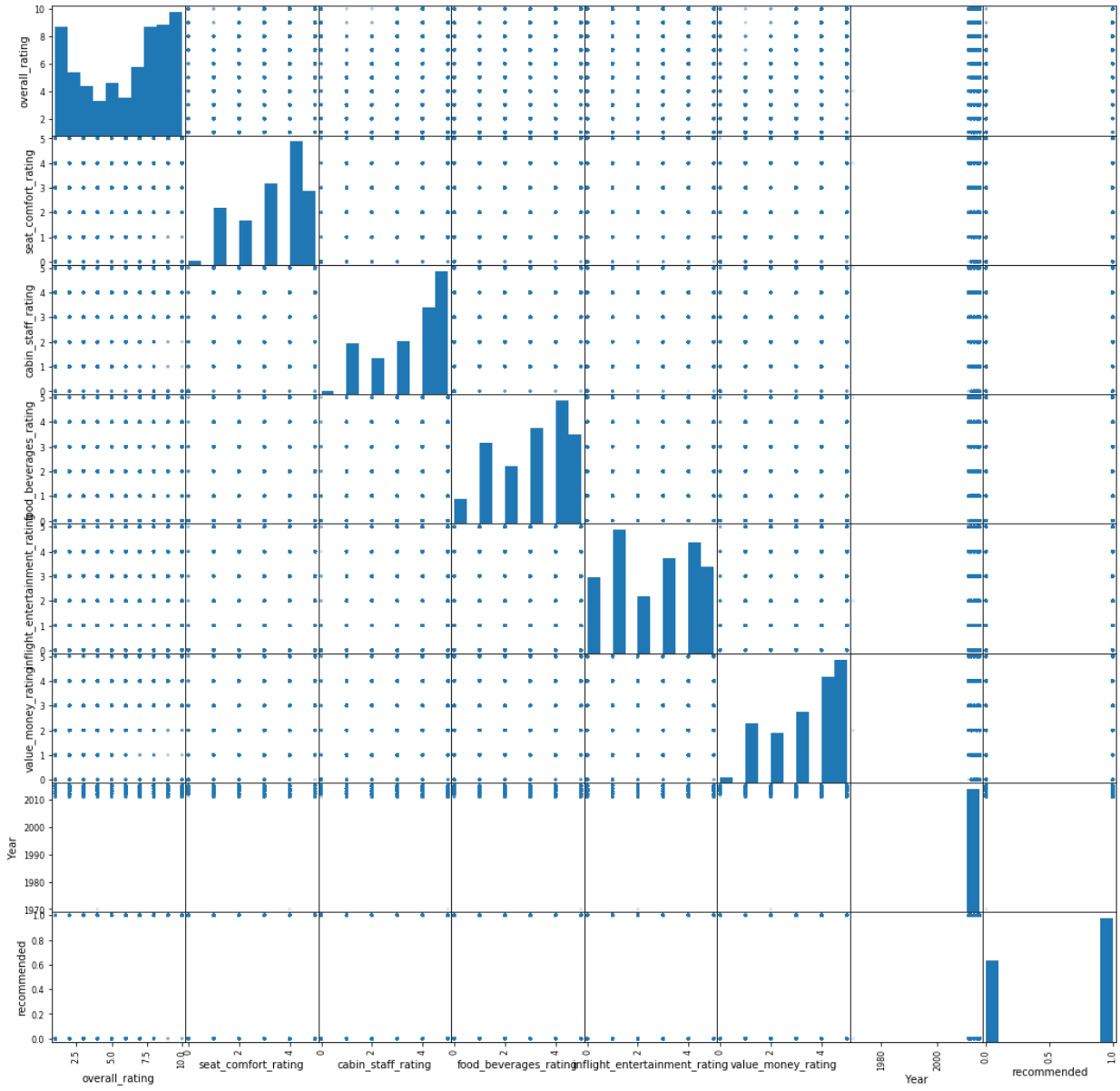


In [32]:
```
1 plt.hist(airlineData.recommended.values)
```

Out[32]:
```
(array([10808.,     0.,     0.,     0.,     0.,     0.,     0.,     0.,
            0., 16476.]),
 array([0. , 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. ]),
 <a list of 10 Patch objects>)
```



In [33]:
```
pd.plotting.scatter_matrix(airlineData, alpha=0.2, figsize=(18, 18))
plt.show()
```



Data Pre- Processing

In [34]:
```
# Finding correlation of features with the target feature
airlineData.corr()["recommended"].sort_values(ascending=False)
```

Out[34]:
```
recommended                    1.000000
overall_rating                 0.873473
value_money_rating             0.767506
cabin_staff_rating             0.709330
seat_comfort_rating            0.648296
food_beverages_rating          0.577775
inflight_entertainment_rating  0.368729
Year                          -0.045218
Name: recommended, dtype: float64
```

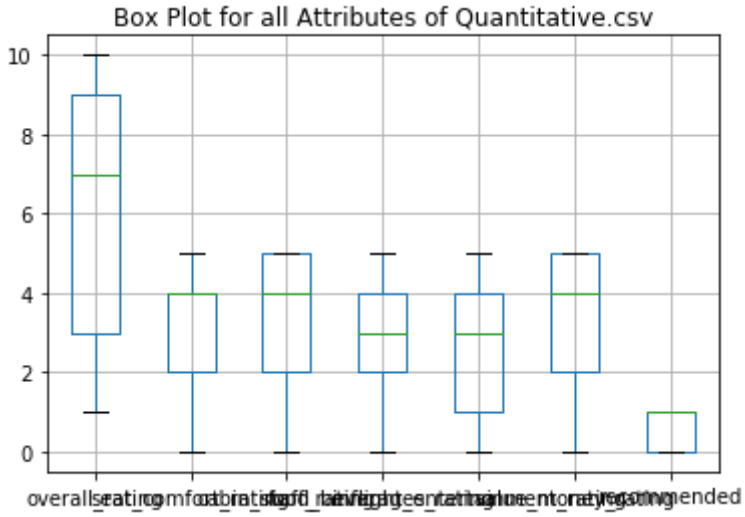
In [35]:
```
# Dropping unwanted/least correlated features
airlineData = airlineData_orig.drop(['author_country','content','Year'],
airlineData
```

Out[35]:

	airline_name	cabin_flown	overall_rating	seat_comfort_rating	cabin_staff_rating	food_bevera
0	adria-airways	Economy	7.0	4.0	4.0	
1	adria-airways	Business Class	10.0	4.0	5.0	
2	adria-airways	Economy	9.0	5.0	5.0	
3	adria-airways	Business Class	8.0	4.0	4.0	
4	adria-airways	Economy	4.0	4.0	2.0	
...	...	...	...	...	...	
27279	wizz-air	Economy	7.0	3.0	3.0	
27280	wizz-air	Economy	2.0	1.0	3.0	
27281	wizz-air	Economy	1.0	2.0	1.0	
27282	wizz-air	Economy	5.0	3.0	4.0	
27283	wizz-air	Economy	1.0	2.0	3.0	

27284 rows × 9 columns

In [36]:
```
# Quantitative Dataframe Boxplot
boxplot = airlineData.boxplot()
plt.title("Box Plot for all Attributes of Quantitative.csv")
plt.show()
```



Since the dataset contains only ratings ranging from 0 to 10, there are no outlier data to be cleaned/ removed. Pre-processing data with Label Encoder and Min-Max normalization is done below.

In [38]:
```
# Data preprocessing
airlineDataTemp = airlineData.iloc[:, :9]

le = preprocessing.LabelEncoder()
for column_name in airlineData.columns:
    if airlineData[column_name].dtype == object:
        airlineData[column_name] = le.fit_transform(airlineData[column_

# Normalizing data with min-max normalization
x = airlineData.iloc[:, :9].values
min_max_scaler = preprocessing.MinMaxScaler(feature_range=[0, 3]) # nor
x_scaled = min_max_scaler.fit_transform(x)
dataNormalizedDf = pd.DataFrame(x_scaled, columns=airlineDataTemp.colum
dataNormalizedDf
```

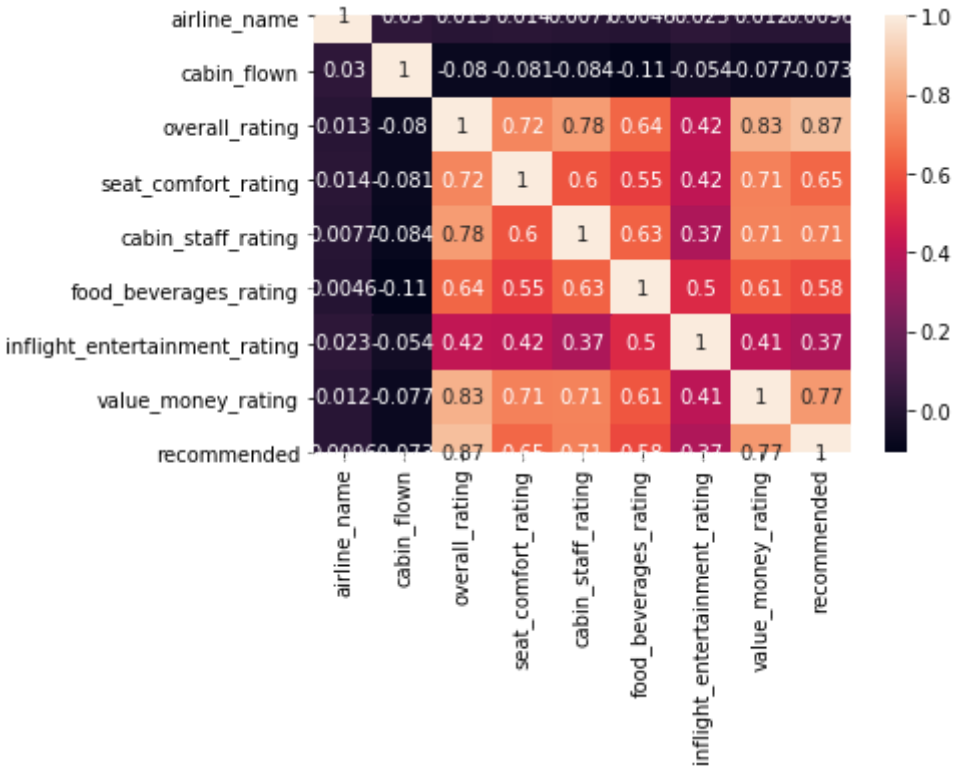
Out[38]:

	airline_name	cabin_flown	overall_rating	seat_comfort_rating	cabin_staff_rating	food_bevera
0	0.000000	1.0	2.000000	2.4	2.4	
1	0.000000	0.0	3.000000	2.4	3.0	
2	0.000000	1.0	2.666667	3.0	3.0	
3	0.000000	0.0	2.333333	2.4	2.4	
4	0.000000	1.0	1.000000	2.4	1.2	
...	...	...	...	...	...	
27279	2.958763	1.0	2.000000	1.8	1.8	
27280	2.958763	1.0	0.333333	0.6	1.8	
27281	2.958763	1.0	0.000000	1.2	0.6	
27282	2.958763	1.0	1.333333	1.8	2.4	
27283	2.958763	1.0	0.000000	1.2	1.8	

27284 rows × 9 columns

In [39]:
```
1 sns.heatmap(airlineData.corr(),annot=True)
```

Out[39]: <matplotlib.axes._subplots.AxesSubplot at 0x1a23bf7358>



Classification Algorithms

Splitting Data for Training & Testing Dataset

In [40]:
```
X = dataNormalizedDf.iloc[:, :-1].values
y = dataNormalizedDf['recommended'].values

# Applying Stratified sampling
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3
print("Shape after split::", X_train.shape, X_test.shape, y_train.shape
```

```
Shape after split:: (18280, 8) (9004, 8) (18280,) (9004,)
```

Decision Trees

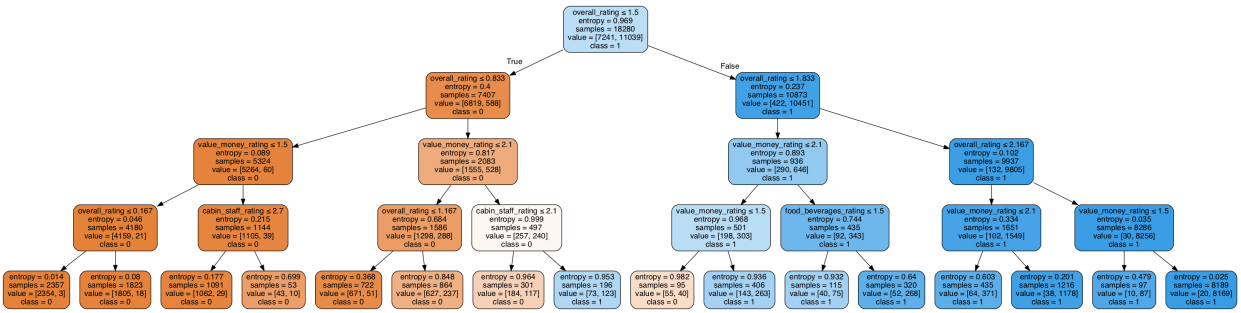
In [41]:
```
# Decision Tree with Entropy
dct = tree.DecisionTreeClassifier(criterion='entropy', max_depth=4, ran
dct = dct.fit(X_train, y_train)
scoreTrain =dct.score(X_train, y_train)
scoreTest = dct.score(X_test, y_test)
print("Decision Tree Classifier Model:::")
print("scoreTrain=", scoreTrain)
print("scoreTest=", scoreTest)
```

```
Decision Tree Classifier Model:::
scoreTrain= 0.9483041575492341
scoreTest= 0.9461350510884051
```

In [42]:
```
!pip install graphviz
!pip install pydotplus

from IPython.display import Image
import pydotplus
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz

dot_data = StringIO()
export_graphviz(dct, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = airlineDataTemp
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('airlineDct.png')
Image(graph.create_png())
```

```
Requirement already satisfied: graphviz in /Library/anaconda3/lib/python
3.7/site-packages (0.13)
Requirement already satisfied: pydotplus in /Library/anaconda3/lib/python
3.7/site-packages (2.0.2)
Requirement already satisfied: pyparsing>=2.0.1 in /Library/anaconda3/li
b/python3.7/site-packages (from pydotplus) (2.4.2)
```

Out[42]:



In [43]:
```
# Predict
y_pred = dct.predict(X_test)

print("Accuracy of Decision Tree Classifier:",metrics.accuracy_score(y_
```

```
Accuracy of Decision Tree Classifier: 0.9461350510884051
```

In [44]:
```
from sklearn.metrics import classification_report
print("Classification Report of Decision Tree Classifier Prediction mod
print(classification_report(y_test,y_pred))
```

```
Classification Report of Decision Tree Classifier Prediction model::
              precision    recall  f1-score   support

         0.0       0.93      0.94      0.93      3567
         3.0       0.96      0.95      0.96      5437

    accuracy                           0.95      9004
   macro avg       0.94      0.94      0.94      9004
weighted avg       0.95      0.95      0.95      9004
```

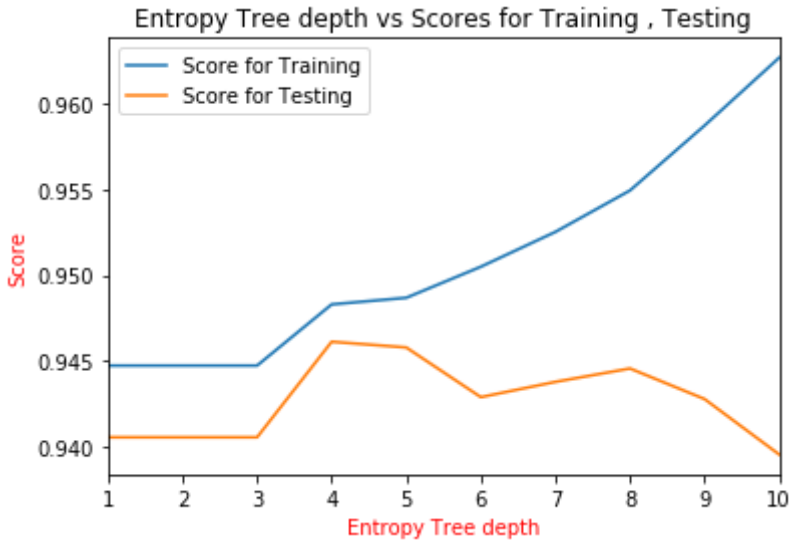
In [45]:

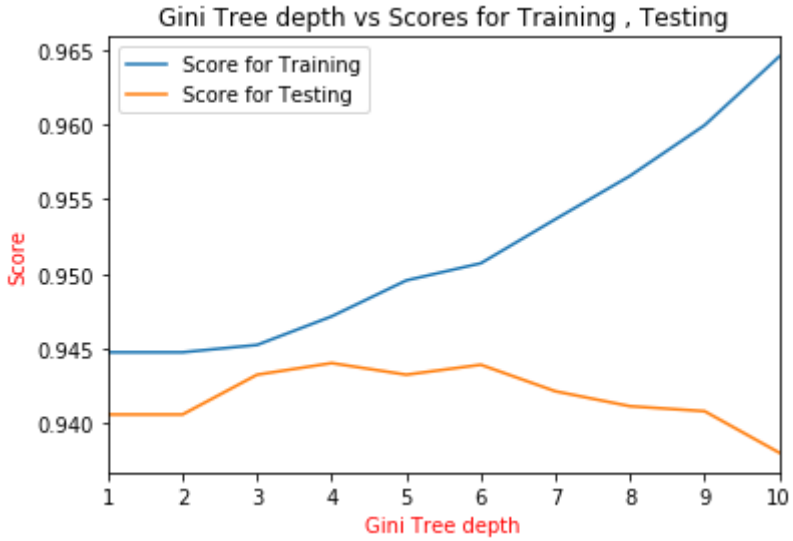
```
# Entropy with different depths from 1 to 10
resultsEntropy = pd.DataFrame(columns=['LevelLimit', 'Score for Trainin
for treeDepth in range (1, 11):
    dct = tree.DecisionTreeClassifier(criterion='entropy', max_depth=tr
    dct = dct.fit(X_train, y_train)
    dct.predict(X_test)
    scoreTrain =dct.score(X_train, y_train)
    scoreTest = dct.score(X_test, y_test)
    resultsEntropy.loc[treeDepth] = [treeDepth, scoreTrain, scoreTest]

print("Decision Tree 'Entropy' :::")
print(resultsEntropy.head(11))
resultsEntropy.pop('LevelLimit')
ax = resultsEntropy.plot(title="Entropy Tree depth vs Scores for Traini
ax.set_xlabel("Entropy Tree depth", color="red")
ax.set_ylabel("Score", color="red")


# Gini index
resultsGini = pd.DataFrame(columns=['LevelLimit', 'Score for Training',
for treeDepth in range (1, 11):
    dct = tree.DecisionTreeClassifier(criterion='gini', max_depth=treeD
    dct = dct.fit(X_train, y_train)
    dct.predict(X_test)
    scoreTrain =dct.score(X_train, y_train)
    scoreTest = dct.score(X_test, y_test)
    resultsGini.loc[treeDepth] = [treeDepth, scoreTrain, scoreTest]

print("Decision Tree 'Gini' :::")
print(resultsGini.head(11))
resultsGini.pop('LevelLimit')
ax = resultsGini.plot(title="Gini Tree depth vs Scores for Training , T
ax.set_xlabel("Gini Tree depth", color="red")
ax.set_ylabel("Score", color="red")
```

```
Decision Tree 'Entropy' :::
    LevelLimit  Score for Training  Score for Testing
1          1.0            0.944748           0.940582
2          2.0            0.944748           0.940582
3          3.0            0.944748           0.940582
4          4.0            0.948304           0.946135
5          5.0            0.948687           0.945802
6          6.0            0.950492           0.942914
7          7.0            0.952516           0.943803
8          8.0            0.954923           0.944580
9          9.0            0.958698           0.942803
10        10.0            0.962637           0.939582
Decision Tree 'Gini' :::
    LevelLimit  Score for Training  Score for Testing
1          1.0            0.944748           0.940582
2          2.0            0.944748           0.940582
3          3.0            0.945241           0.943247
4          4.0            0.947155           0.944025
5          5.0            0.949562           0.943247
6          6.0            0.950711           0.943914
7          7.0            0.953665           0.942137
```

```
8          8.0          0.956565          0.941137
9          9.0          0.959956          0.940804
10        10.0          0.964551          0.938028
```

Out[45]: Text(0, 0.5, 'Score')





K-Fold on Decision Trees

In [46]:
```
# Cross Validation

# K-fold
from sklearn.model_selection import KFold # import KFold

kf = KFold(n_splits=2) # Define the split - into 2 folds
kf.get_n_splits(X) # returns the number of splitting iterations in the
print(kf)
```

```
KFold(n_splits=2, random_state=None, shuffle=False)
```

In [47]:
```
for train_index, test_index in kf.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
```

```
TRAIN: [13642 13643 13644 ... 27281 27282 27283] TEST: [    0     1     2
... 13639 13640 13641]
TRAIN: [    0     1     2 ... 13639 13640 13641] TEST: [13642 13643 13644
... 27281 27282 27283]
```

In [48]:
```
# Cross Validation LOOCV
from sklearn.model_selection import LeaveOneOut

loo = LeaveOneOut()
loo.get_n_splits(X)
for train_index, test_index in loo.split(X):
    print("TRAIN:", train_index, "TEST:", test_index)
print(X_train, X_test, y_train, y_test)
```

```
TRAIN: [    1     2     3 ... 27281 27282 27283] TEST: [0]
TRAIN: [    0     2     3 ... 27281 27282 27283] TEST: [1]
TRAIN: [    0     1     3 ... 27281 27282 27283] TEST: [2]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [3]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [4]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [5]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [6]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [7]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [8]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [9]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [10]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [11]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [12]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [13]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [14]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [15]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [16]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [17]
TRAIN: [    0     1     2 ... 27281 27282 27283] TEST: [18]
```

In [49]:
```
from sklearn.model_selection import cross_val_score, cross_val_predict
from sklearn import metrics

# Perform 10-fold cross validation
scores = cross_val_score(dct, X, y, cv=10)
k = 1
for i in scores:
    print("K-Fold level = ", k , " Score = ", i)
    k=k+1
```

```
K-Fold level =  1  Score =  0.8937339684866251
K-Fold level =  2  Score =  0.929278123854892
K-Fold level =  3  Score =  0.9212165628435325
K-Fold level =  4  Score =  0.929278123854892
K-Fold level =  5  Score =  0.9384389886405277
K-Fold level =  6  Score =  0.9263466471234885
K-Fold level =  7  Score =  0.9226539589442815
K-Fold level =  8  Score =  0.8856304985337243
K-Fold level =  9  Score =  0.9251925192519251
K-Fold level =  10  Score =  0.8558855885588559
```

In [ ]:
```

```

Naive Bayes

In [50]:
```
from sklearn.naive_bayes import MultinomialNB
Nb_model = MultinomialNB()
Nb_model.fit(X_train,y_train)
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

Out[50]:
```
MultinomialNB(alpha=1.0, class_prior=None, fit_prior=True)
```

In [51]:
```
pred=Nb_model.predict(X_test)
pred
```

Out[51]:
```
array([3., 0., 3., ..., 0., 3., 3.])
```

In [52]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test,pred)
```

Out[52]:
```
array([[2306, 1261],
       [ 142, 5295]])
```

In [53]:
```
from sklearn.metrics import accuracy_score
accuracy_score(y_test,pred)
```

Out[53]:
```
0.8441803642825411
```

In [54]:
```
from sklearn.model_selection import cross_val_score
score_nb=cross_val_score(Nb_model, X, y, cv=10, scoring='accuracy')
score_nb.mean()
```

Out[54]:
```
0.8382716837969056
```

Random Forest

In [55]:
```
from sklearn.ensemble import RandomForestClassifier
RF_model = RandomForestClassifier(n_estimators=10,random_state=42,boots
RF_model.fit(X_train, y_train)
RF_model.score(X_test, y_test)
```

Out[55]:
```
0.9404709018214127
```

In [56]:
```
rf_pred = RF_model.predict(X_test)
```

In [57]:
```
rf_cm = confusion_matrix(y_test, rf_pred)
rf_cm
```

Out[57]:
```
array([[3339,  228],
       [ 308, 5129]])
```

In [58]:
```
!pip install pydot

from IPython.display import Image
from sklearn.externals.six import StringIO
from sklearn.tree import export_graphviz
import pydot
```

```
Requirement already satisfied: pydot in /Library/anaconda3/lib/python3.7/
site-packages (1.4.1)
Requirement already satisfied: pyparsing>=2.1.4 in /Library/anaconda3/li
b/python3.7/site-packages (from pydot) (2.4.2)
```

In [59]:
```
features=list(airlineDataTemp.iloc[:, :-1].columns)
features
```

Out[59]:
```
['airline_name',
 'cabin_flown',
 'overall_rating',
 'seat_comfort_rating',
 'cabin_staff_rating',
 'food_beverages_rating',
 'inflight_entertainment_rating',
 'value_money_rating']
```

In [60]:
```
tree = RF_model.estimators_[5]
dot_data=StringIO()
export_graphviz(tree,out_file=dot_data,feature_names=features,class_nam
graph=pydot.graph_from_dot_data(dot_data.getvalue())
Image(graph[0].create_png())
```

Out[60]:

In [61]:
```
score_rf=cross_val_score(RF_model, X, y, cv=10, scoring='accuracy')
score_rf.mean()
```

Out[61]:
```
0.9218535724008221
```

SVM

In [62]:
```
df_cor= airlineData[['overall_rating','cabin_flown','seat_comfort_ratin
```

In [63]:
```
X=df_cor[['overall_rating','cabin_flown','seat_comfort_rating','cabin_s
y=df_cor['recommended']
```

In [64]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,test_size=0.3)
```

In [65]:
```
from sklearn.svm import SVC
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)

from sklearn.metrics import classification_report, confusion_matrix
print ('Linear Kernel Performance')
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

```

```
Linear Kernel Performance
[[2988  217]
 [ 238 4743]]
              precision    recall  f1-score   support

           0       0.93      0.93      0.93      3205
           1       0.96      0.95      0.95      4981

    accuracy                           0.94      8186
   macro avg       0.94      0.94      0.94      8186
weighted avg       0.94      0.94      0.94      8186
```

In [66]:
```
svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)

print ('Linear Kernel Performance')
print(confusion_matrix(y_test,y_pred))
print(classification_report(y_test,y_pred))

```

```
Linear Kernel Performance
[[2988  217]
 [ 238 4743]]
              precision    recall  f1-score   support

           0       0.93      0.93      0.93      3205
           1       0.96      0.95      0.95      4981

    accuracy                           0.94      8186
   macro avg       0.94      0.94      0.94      8186
weighted avg       0.94      0.94      0.94      8186
```

In [67]:
```
g_svclassifier = SVC(kernel='rbf')
g_svclassifier.fit(X_train, y_train)
g_y_pred = g_svclassifier.predict(X_test)

print ('Gussain Kernel Performance')
print(confusion_matrix(y_test, g_y_pred))
print(classification_report(y_test, g_y_pred))
```

```
Gussain Kernel Performance
[[3002  203]
 [ 245 4736]]
              precision    recall  f1-score   support

           0       0.92      0.94      0.93      3205
           1       0.96      0.95      0.95      4981

    accuracy                           0.95      8186
   macro avg       0.94      0.94      0.94      8186
weighted avg       0.95      0.95      0.95      8186
```

In [68]:
```
sig_svclassifier = SVC(kernel='sigmoid')
sig_svclassifier.fit(X_train, y_train)
sig_y_pred = sig_svclassifier.predict(X_test)

print ('Sigmoid Kernel Performance')
print(confusion_matrix(y_test, sig_y_pred))
print(classification_report(y_test, sig_y_pred))
```

```
Sigmoid Kernel Performance
[[   1 3204]
 [2299 2682]]
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      3205
           1       0.46      0.54      0.49      4981

    accuracy                           0.33      8186
   macro avg       0.23      0.27      0.25      8186
weighted avg       0.28      0.33      0.30      8186
```

In [ ]:

```
%matplotlib inline

print(__doc__)

import numpy as np
import matplotlib.pyplot as plt
from sklearn import svm, datasets

# import some data to play with
#iris = datasets.load_iris()
#print (X_train)
X = X_train[['overall_rating','value_money_rating']]  # we only take th
                      # avoid this ugly slicing by using a two-dim data
X = X.to_numpy()
y = y_train
y = y.to_numpy()

h = .02  # step size in the mesh

# we create an instance of SVM and fit out data. We do not scale our
# data since we want to plot the support vectors
C = 1.0  # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=C).fit(X, y)
rbf_svc = svm.SVC(kernel='rbf', gamma=0.7, C=C).fit(X, y)
poly_svc = svm.SVC(kernel='poly', degree=3, C=C).fit(X, y)
lin_svc = svm.LinearSVC(C=C).fit(X, y)

# create a mesh to plot in
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))

# title for the plots
titles = ['SVC with linear kernel',
          'LinearSVC (linear kernel)',
          'SVC with RBF kernel',
          'SVC with polynomial (degree 3) kernel']


for i, clf in enumerate((svc, lin_svc, rbf_svc, poly_svc)):
    # Plot the decision boundary. For that, we will assign a color to e
    # point in the mesh [x_min, x_max]x[y_min, y_max].
    plt.subplot(2, 2, i + 1)
    plt.subplots_adjust(wspace=0.4, hspace=0.4)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

    # Put the result into a color plot
    Z = Z.reshape(xx.shape)
    plt.contourf(xx, yy, Z, cmap=plt.cm.coolwarm, alpha=0.8)

    # Plot also the training points
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap=plt.cm.coolwarm)
    plt.xlabel('Overall Rating')
    plt.ylabel('Value Money Rating')
```

```
    plt.xlim(xx.min(), xx.max())
    plt.ylim(yy.min(), yy.max())
    plt.xticks(())
    plt.yticks(())
    plt.title(titles[i])

plt.show()
```

```
Automatically created module for IPython interactive environment
```

K Means and PCA

In [74]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x_scaled)
principalDf = pd.DataFrame(data = principalComponents, columns = ['prin
principalDf
```

Out[74]:

	principal component 1	principal component 2
0	-0.983866	-1.958869
1	-1.967212	-1.635383
2	-1.840427	-2.076037
3	-1.098725	-1.659318
4	2.300094	-0.137943
...	...	...
27279	-0.128850	-1.971489
27280	2.819787	-0.156139
27281	3.521786	-0.565983
27282	-0.325211	-1.038528
27283	2.980598	-0.254663

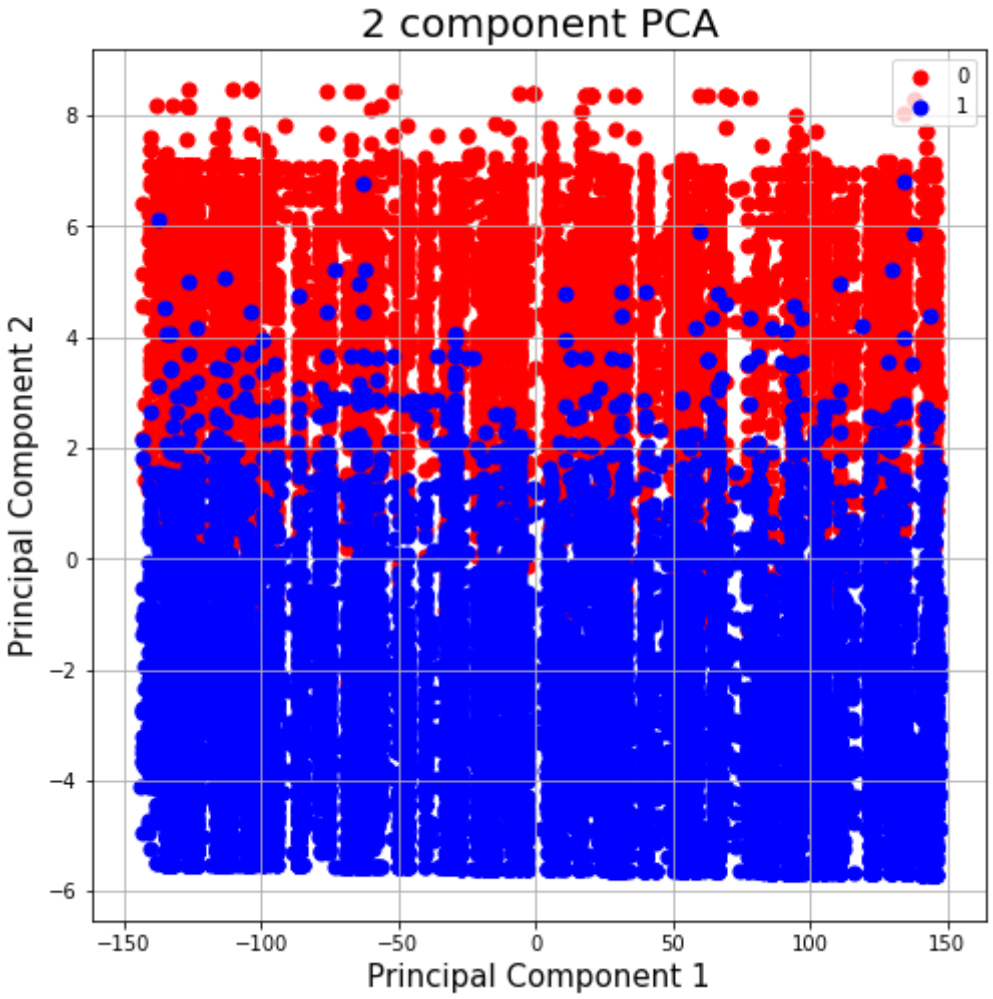
27284 rows × 2 columns

In [75]:
```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
principalComponents = pca.fit_transform(x)
principalDf = pd.DataFrame(data = principalComponents, columns = ['prin
```

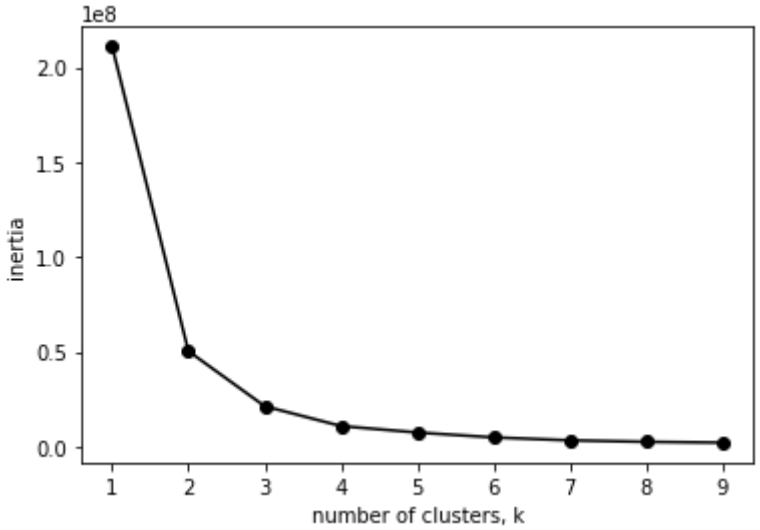
In [76]:
```
finalDf = pd.concat([principalDf, airlineData[['recommended']]], axis =
```

In [77]:

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)
targets = [0, 1]
colors = ['r', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['recommended'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



In [78]:
```
from sklearn.cluster import KMeans

ks = range(1, 10)
inertias = []
for k in ks:
    # Create a KMeans instance with k clusters: model
    model = KMeans(n_clusters=k)

    # Fit model to samples
    model.fit(principalDf.iloc[:,:3])

    # Append the inertia to the list of inertias
    inertias.append(model.inertia_)

plt.plot(ks, inertias, '-o', color='black')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```



In [ ]:
```
1
```

In [ ]:
```
1
```