***CIS 8045 – Term Project***

***Submitted by:***

***Darshee Mehta, Gautam Shah, Madhur Verma, Venkata Sai Krishna Yadala***

Table of Contents

[**Text Analysis Using Python** 3](#_Toc475841859)

[***Part a*** 3](#_Toc475841860)

[***Part b*** 6](#_Toc475841861)

[**Multiple Factors:** 8](#_Toc475841862)

[**NEO4J** 11](#_Toc475841863)

[***Part 1*** 11](#_Toc475841864)

[***Part 2*** 16](#_Toc475841865)

[***Part 4.1*** 17](#_Toc475841866)

[***Part 4.2*** 18](#_Toc475841867)

# **Text Analysis Using Python**

The project is first setup with dataset consisting of real customer review data collected from Skytrax ([www.airlinequality.com](http://www.airlinequality.com)), a major website for customers to evaluate various airline companies. This dataset contains 41,396 review entries, with each entry capturing the rating of airlines.

First part is a lexicon Analysis where text content of Author review is taken as input for the analysis and each word in the content is subjected to further refinement such as removing stop words, removing words with length less than one and finally applying stemming algorithm to create a better training set.

A dictionary containing list of positive and negative words is created using one of the available free dictionary and the same is used to count positive and negative words in the review to arrive at a sentiment count. An Ordinary Least-Squares (OLS) Regression , Statistical Learning technique is used for estimating values. Then to transform linear regression into probabilities logistic regression is applied across recommended and sentiment columns.

## ***Part a***

#Importing all required libraries for text analytics

import pymongo

import nltk

from pymongo import MongoClient

import pandas as pd

import csv

from nltk import sent\_tokenize, word\_tokenize

from nltk.corpus import stopwords

from nltk.stem.snowball import SnowballStemmer

#connecting to mongoDB for data

con=MongoClient()

airlineDB=con.Airline

reviews=airlineDB.reviews

reviewData=pd.DataFrame(list(reviews.find()))

# We use [General Inquirer](http://www.wjh.harvard.edu/~inquirer/),

# a free dictionary, to analyze review sentiment.

#Import positive and negative words from General Inquirer Dictionary\*

positive=[]

negative=[]

with open(*'general\_inquirer\_dict.txt'*) as fin:

# Reads the csv file delimited by tab, separates the negative

# and positive words and adds them to the list

reader = csv.DictReader(fin,delimiter=*'\t'*)

for i,line in enumerate(reader):

if line[*'Negativ'*]==*'Negativ'*:

if line[*'Entry'*].find(*'#'*)==-1:

negative.append(line[*'Entry'*].lower())

if line[*'Entry'*].find(*'#'*)!=-1: #In General Inquirer, some words have multiple senses. Combine all tags for all senses.

negative.append(line[*'Entry'*].lower()[:line[*'Entry'*].index(*'#'*)])

if line[*'Positiv'*]==*'Positiv'*:

if line[*'Entry'*].find(*'#'*)==-1:

positive.append(line[*'Entry'*].lower())

if line[*'Entry'*].find(*'#'*)!=-1: #In General Inquirer, some words have multiple senses. Combine all tags for all senses.

positive.append(line[*'Entry'*].lower()[:line[*'Entry'*].index(*'#'*)])

#Closes the file

fin.close()

pvocabulary=sorted(list(set(positive)))

nvocabulary=sorted(list(set(negative)))

#creating relevant columns for analysis

reviewData[*'pos'*]=0

reviewData[*'neg'*]=0

reviewData[*'sentiment'*]=0

# Gets all the words in the reviews after tokenizing sentences and words

# and adding it to the list word\_list

def **getWordList**(text, word\_proc = lambda x:x):

word\_list = []

for sent in sent\_tokenize(text):

for word in word\_tokenize(sent):

word\_list.append(word)

return word\_list

# Gets all the english stemmer words in the variable stemmer

stemmer = SnowballStemmer(*"english"*)

pcount\_list = []

ncount\_list = []

senti\_list = []

sample2=reviewData[:7000]#sampling is optional and can be removed

review\_index=0

for text in sample2[*'reviewcontent'*]:

vocabulary = getWordList(text, lambda x:x.lower())

# Remove words with a length of 1

vocabulary = [word for word in vocabulary if len(word) > 1]

# Remove stopwords

vocabulary = [word for word in vocabulary

if not word in stopwords.words(*'english'*)]

# Stem words

vocabulary = [stemmer.stem(word) for word in vocabulary]

# Counts the total occurrance of positive and negative words

# in the list named vocabulary and appends the number to pcount\_list

# and ncount\_list and stores the overall difference of pcount and ncount

# to get the overall sentiment of each review and stores it to

# senti\_list

pcount = 0

ncount = 0

for pword in pvocabulary:

pcount += vocabulary.count(pword)

for nword in nvocabulary:

ncount += vocabulary.count(nword)

pcount\_list.append(pcount)

ncount\_list.append(ncount)

senti\_list.append(pcount-ncount)

# Adds the sentiment count to the corresponding row and column

sample2.loc[review\_index, *'pos'*] = pcount

sample2.loc[review\_index, *'neg'*] = ncount

sample2.loc[review\_index, *'sentiment'*] = pcount - ncount

review\_index += 1 #Increases the row index

#Applying OLS regression

import statsmodels.formula.api as sm

result1 = sm.ols(formula=*"recommended~sentiment"*, data = sample2).fit()

print(result1.summary())

result2 = sm.ols(formula=*"rating\_overall~sentiment"*, data = sample2).fit()

print(result2.summary())

*OLS Regression Results*

==============================================================================

Dep. Variable: rating\_overall R-squared: 0.146

Model: OLS Adj. R-squared: 0.146

Method: Least Squares F-statistic: 1058.

Date: Sun, 26 Feb 2017 Prob (F-statistic): 2.13e-214

Time: 03:02:39 Log-Likelihood: -15569.

No. Observations: 6203 AIC: 3.114e+04

Df Residuals: 6201 BIC: 3.116e+04

Df Model: 1

Covariance Type: nonrobust

==============================================================================

coef std err t P>|t| [95.0% Conf. Int.]

------------------------------------------------------------------------------

Intercept 5.2055 0.045 116.475 0.000 5.118 5.293

sentiment 0.2968 0.009 32.525 0.000 0.279 0.315

==============================================================================

Omnibus: 1614.881 Durbin-Watson: 1.825

Prob(Omnibus): 0.000 Jarque-Bera (JB): 366.114

Skew: -0.312 Prob(JB): 3.16e-80

Kurtosis: 1.986 Cond. No. 5.86

==============================================================================

#Applying logistic regression

import statsmodels.api as sm2

logit = sm2.Logit(sample2[*'recommended'*], sample2[*'sentiment'*])

result = logit.fit()

print(result.summary())

*Logit Regression Results*

==============================================================================

Dep. Variable: recommended No. Observations: 7000

Model: Logit Df Residuals: 6999

Method: MLE Df Model: 0

Date: Sun, 26 Feb 2017 Pseudo R-squ.: 0.09690

Time: 03:02:40 Log-Likelihood: -4372.6

converged: True LL-Null: -4841.8

LLR p-value: nan

==============================================================================

coef std err z P>|z| [95.0% Conf. Int.]

------------------------------------------------------------------------------

sentiment 0.1758 0.006 27.686 0.000 0.163 0.188

==============================================================================

## ***Part b***

Using the provided dataset, A model is created using Naïve Bayes classifier in order to predict airline recommendation based on author review content. In order to train the model, we prepare training dataset by tokenizing each review into words and refining them by applying various techniques such as stemming and finally ngrams are created using library to have an better input set. Then the input set is converted into format required by the classifier. Once the model is trained, an test data set is applied on the model to predict the recommendation. The performance of the model is now measured using various metrics such positive recall, negative recall. positive precision etc. Finally a CSV is created listing both original values and predicted values of recommendation column.

Code:

#Importing all required libraries for text analytics

import pymongo

import nltk

from pymongo import MongoClient

import pandas as pd

import csv

from nltk import sent\_tokenize, word\_tokenize

from nltk.corpus import stopwords

from nltk.stem.snowball import SnowballStemmer

from nltk.util import ngrams

#connecting to mongoDB for data

con=MongoClient()

airlineDB=con.Airline

reviews=airlineDB.reviews

reviewData=pd.DataFrame(list(reviews.find()))

posResult=pd.DataFrame(list(reviews.find({*'recommended'*:1})))

negResult=pd.DataFrame(list(reviews.find({*'recommended'*:0})))

#Sampling done here is optional and can be removed if needed

posSample=posResult[:5000]

negSample=negResult[:5000]

stemmer = SnowballStemmer(*"english"*)

def **getWordList**(text, word\_proc = lambda x:x):

*'''*

*this function takes each review and breaks them into words and then*

*further refines these list of words by removing stopwords,applying stemming*

*and creating ngrams*

*'''*

word\_list=[]

for sent in sent\_tokenize(text):

for word in word\_tokenize(sent):

word\_list.append(word)

vocabulary = [word for word in word\_list if len(word) > 2]

vocabulary = [word for word in vocabulary

if not word in stopwords.words(*'english'*)]

vocabulary = [stemmer.stem(word) for word in vocabulary]

ngramsize=2

if ngramsize>1:

vocabulary=[word for word in ngrams(

vocabulary,ngramsize)]

return vocabulary

def **word\_feats**(words):

if words != words:

words = *""*

vocabulary = getWordList(words, lambda x:x.lower())

return dict([(word,True) for word in vocabulary])#creating list of words in format suitable for classifier

pos\_feat=[]

neg\_feat=[]

pos\_feat=[(word\_feats(f),*'1'*)

for f in posSample[*"reviewcontent"*]]

neg\_feat=[(word\_feats(f),*'0'*)

for f in negSample[*"reviewcontent"*]]

#creating test and train data set

train=pos\_feat[:4000]+neg\_feat[:4000]

test=pos\_feat[4000:]+neg\_feat[4000:]

#using Naive Bayes classifier for creating model

from nltk.classify import NaiveBayesClassifier

classifier=NaiveBayesClassifier.train(train)

import collections

refsets=collections.defaultdict(set)

testsets=collections.defaultdict(set)

result=[]

for i,(feats,label) in enumerate(test):

refsets[label].add(i)

observed=classifier.classify(feats)#using trained model on test data

testsets[observed].add(i)

result.append(observed)

#calculating various metrics to measure the performance of model

import math

from nltk.metrics import precision

from nltk.metrics import recall

pos\_pre=precision(refsets[*'1'*],testsets[*'1'*])

neg\_pre=precision(refsets[*'0'*],testsets[*'0'*])

pos\_rec=recall(refsets[*'1'*],testsets[*'1'*])

neg\_rec=recall(refsets[*'0'*],testsets[*'0'*])

gperf=math.sqrt(pos\_pre\*neg\_rec)

print(*'Positive Precision: '*,pos\_pre)

print(*'Negative Precision: '*,neg\_pre)

print(*'Positive Recall: '*,pos\_rec)

print(*'Negative Recall: '*,neg\_rec)

print(*'G-Performance: '*,gperf)

df = pd.DataFrame()

se2=pd.Series([*'1'*]\*5000+[*'0'*]\*5000)

df[*'Original\_recomendation'*]=se2

se3=pd.Series([*''*]\*4000+result[0:1000]+[*''*]\*4000+result[1000:2000])

df[*'Predicted\_Class'*]=se3

df.to\_csv(*"Results.csv"*)#it must be an relative path with respect to this file location

### **Multiple Factors:**

For this analysis, we identified the three most contributing factors towards a positive or negative recommendation. We calculated the correlation values between all rating parameters and the recommendation values. As per our analysis, *rating\_cabinstaff, rating\_valuemoney and rating\_overall* came out to be the most strongly correlated, and hence were used for training the Naïve Bayes Classifier.

#Importing all required libraries for text analytics

import pymongo

import nltk

from pymongo import MongoClient

import pandas as pd

import csv

from nltk import sent\_tokenize, word\_tokenize

from nltk.corpus import stopwords

from nltk.stem.snowball import SnowballStemmer

#connecting to mongoDB for data

con=MongoClient()

airlineDB=con.Airline

reviews=airlineDB.reviews

reviewData=pd.DataFrame(list(reviews.find()))

posResult=pd.DataFrame(list(reviews.find({*'recommended'*:1})))

negResult=pd.DataFrame(list(reviews.find({*'recommended'*:0})))

#Finding Correlations to get most influencing factors driving recommendation

cabinToR=reviewData[*'rating\_cabinstaff'*].corr(reviewData[*'recommended'*])

foodToR=reviewData[*'rating\_foodbeverage'*].corr(reviewData[*'recommended'*])

inflightToR=reviewData[*'rating\_inflightEnt'*].corr(reviewData[*'recommended'*])

overallToR=reviewData[*'rating\_overall'*].corr(reviewData[*'recommended'*])

seatToR=reviewData[*'rating\_seatcomfort'*].corr(reviewData[*'recommended'*])

valueToR=reviewData[*'rating\_valuemoney'*].corr(reviewData[*'recommended'*])

#Sampling done here is optional and can be removed if needed

posSample=posResult[:5000]

negSample=negResult[:5000]

def **Nancheck**(content):

if content != content:

return *""*

return content

def **num\_feats**(words,j):

list1 = []

list1.append(Nancheck(words[*"rating\_cabinstaff"*][j]))

list1.append(Nancheck(words[*"rating\_overall"*][j]))

list1.append(Nancheck(words[*"rating\_valuemoney"*][j]))

return dict([(word,True) for word in list1])

#creating test and train data set

pos\_feat=[(num\_feats(posSample,j),*'1'*)

for j in range(0,len(posSample))]

neg\_feat=[(num\_feats(negSample,j),*'0'*)

for j in range(0,len(negSample))]

train=pos\_feat[:4000]+neg\_feat[:4000]

test=pos\_feat[4000:]+neg\_feat[4000:]

#using Naive Bayes classifier for creating model

from nltk.classify import NaiveBayesClassifier

classifier=NaiveBayesClassifier.train(train)

import collections

refsets=collections.defaultdict(set)

testsets=collections.defaultdict(set)

result=[]

for i,(feats,label) in enumerate(test):

refsets[label].add(i)

observed=classifier.classify(feats)#using trained model on test data

testsets[observed].add(i)

result.append(observed)

#calculating various metrics to measure the performance of model

import math

from nltk.metrics import precision

from nltk.metrics import recall

pos\_pre=precision(refsets[*'1'*],testsets[*'1'*])

neg\_pre=precision(refsets[*'0'*],testsets[*'0'*])

pos\_rec=recall(refsets[*'1'*],testsets[*'1'*])

neg\_rec=recall(refsets[*'0'*],testsets[*'0'*])

gperf=math.sqrt(pos\_pre\*neg\_rec)

print(*'Positive Precision: '*,pos\_pre)

print(*'Negative Precision: '*,neg\_pre)

print(*'Positive Recall: '*,pos\_rec)

print(*'Negative Recall: '*,neg\_rec)

print(*'G-Performance: '*,gperf)

df = pd.DataFrame()

se2=pd.Series([*'1'*]\*5000+[*'0'*]\*5000)

df[*'Original\_recomendation'*]=se2

se3=pd.Series([*''*]\*4000+result[0:1000]+[*''*]\*4000+result[1000:2000])

df[*'Predicted\_Class'*]=se3

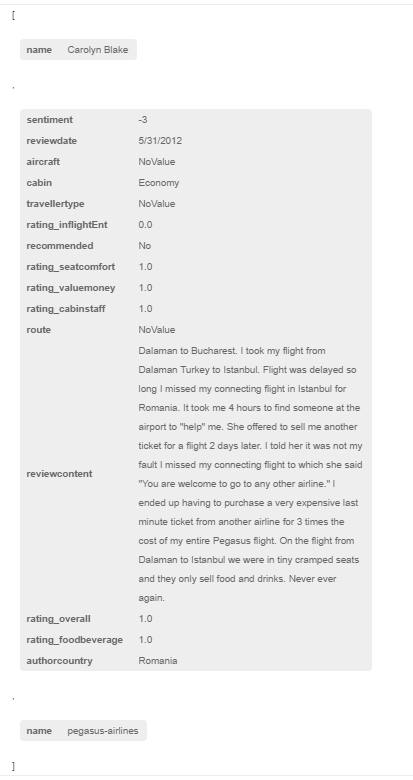
df.to\_csv(*"Results.csv"*)#it must be an raltive path with respect to this file location

# **NEO4J**

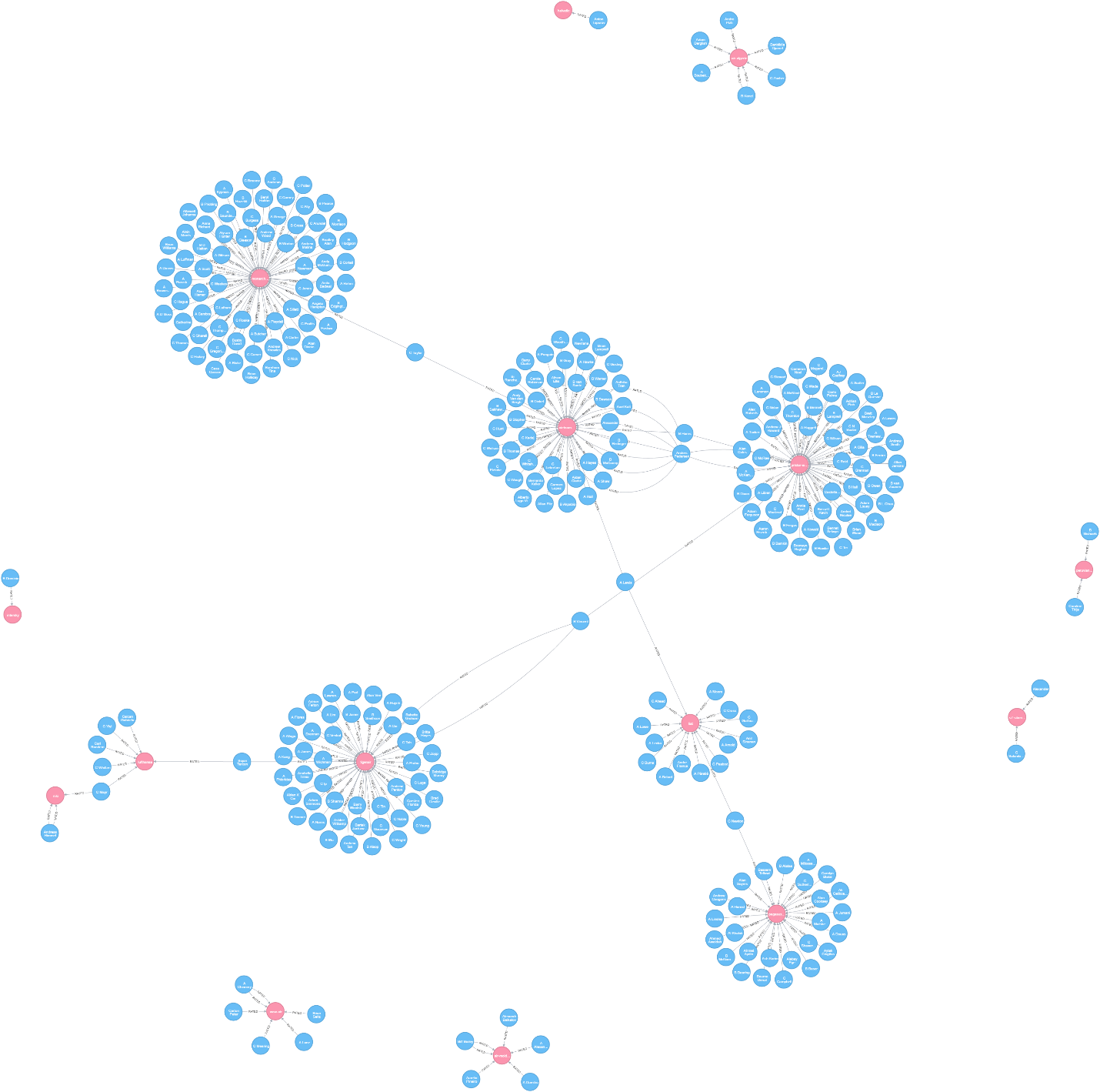
## ***Part 1***

**Inserting the data in Neo4j after getting the sentiment for each review from MongoDB**

* Py2neo version 3.1.2 has been used
* Created a python code using py2neo to import data from a csv file and created two nodes: Airline and Author and created a relationship called RATED between these nodes
* As a part of data clean, all null values were substituted with String ‘NoValue’ and used Latin encoding to make sure the parser parses characters such as è, ä, ø, etc.
* To make the data uniform, all number values were converted to String and the Recommend value of 0 and 1 were substituted with No and Yes
* Since the we are sampling 10000 records from the database of 40000 records, the program takes approximately 10 mins to create the nodes and relationships
* A sample relationship is shown below:



* The sample database structure in Neo4j has been shown below:



**Python Code:**

import pandas as pd

import csv

import math

#py2neo 3.1.2

import py2neo

from py2neo import \*

py2neo.authenticate(*"localhost:7474"*, *"neo4j"*, *"termproject"*)

graph = Graph(*"http://localhost:7474/db/data/"*)

#graph.delete\_all()

#1 Creating database in Neo4j from .csv file

# Reading records from the csv file

review = pd.read\_csv(*'AirlineSentimentResults.csv'*, encoding=*'latin-1'*)

#Using only the first 10000 reviews for analysis

review = review[:10000]

# Creating a set of unique author and airline names

setAuthor = set(review[*'authorname'*])

setAirline = set(review[*'airlinename'*])

# Insert Author Nodes

for i in setAuthor:

i = Node(*"Author"*, name=i)

graph.create(i)

# Insert Airline Nodes

for j in setAirline:

j = Node(*"Airline"*, name=j)

graph.create(j)

# Create relationships between Author and Airline and give values to its parameters

statement = *"MATCH (a {name:{A}}), (b {name:{B}}) \*

*CREATE (a)-[:RATED {authorcountry:{C}, \*

*aircraft:{D}, \*

*route:{E}, \*

*travellertype:{F}, \*

*cabin:{G}, \*

*rating\_cabinstaff:{H}, \*

*rating\_foodbeverage:{I}, \*

*rating\_inflightEnt:{J}, \*

*rating\_overall:{K}, \*

*rating\_seatcomfort:{L}, \*

*rating\_valuemoney:{M}, \*

*recommended:{N}, \*

*reviewcontent:{O}, \*

*reviewdate:{P}, \*

*sentiment:{Q}}]->(b)"*

authorcountry = []

aircraft = []

route = []

travellertype = []

cabin = []

rating\_cabinstaff = []

rating\_foodbeverage = []

rating\_inflightEnt = []

rating\_overall = []

rating\_seatcomfort = []

rating\_valuemoney = []

recommended = []

sentiment = []

for k in range(0, len(review)):

if review.loc[k, *'authorcountry'*] != review.loc[k, *'authorcountry'*]:

authorcountry.append(*'NoValue'*)

else:

authorcountry.append(review.loc[k, *'authorcountry'*])

if review.loc[k, *'aircraft'*] != review.loc[k, *'aircraft'*]:

aircraft.append(*'NoValue'*)

else:

aircraft.append(review.loc[k, *'aircraft'*])

if review.loc[k, *'route'*] != review.loc[k, *'route'*]:

route.append(*'NoValue'*)

else:

route.append(review.loc[k, *'route'*])

if review.loc[k, *'travellertype'*] != review.loc[k, *'travellertype'*]:

travellertype.append(*'NoValue'*)

else:

travellertype.append(review.loc[k, *'travellertype'*])

if review.loc[k, *'cabin'*] != review.loc[k, *'cabin'*]:

cabin.append(*'NoValue'*)

else:

cabin.append(review.loc[k, *'cabin'*])

if (math.isnan(review.loc[k, *'rating\_cabinstaff'*])):

rating\_cabinstaff.append(*'NoValue'*)

else:

rating\_cabinstaff.append(str(review.loc[k, *'rating\_cabinstaff'*]))

if (math.isnan(review.loc[k, *'rating\_foodbeverage'*])):

rating\_foodbeverage.append(*'NoValue'*)

else:

rating\_foodbeverage.append(str(review.loc[k, *'rating\_foodbeverage'*]))

if (math.isnan(review.loc[k, *'rating\_inflightEnt'*])):

rating\_inflightEnt.append(*'NoValue'*)

else:

rating\_inflightEnt.append(str(review.loc[k, *'rating\_inflightEnt'*]))

if (math.isnan(review.loc[k, *'rating\_overall'*])):

rating\_overall.append(*'NoValue'*)

else:

rating\_overall.append(str(review.loc[k, *'rating\_overall'*]))

if (math.isnan(review.loc[k, *'rating\_seatcomfort'*])):

rating\_seatcomfort.append(*'NoValue'*)

else:

rating\_seatcomfort.append(str(review.loc[k, *'rating\_seatcomfort'*]))

if (math.isnan(review.loc[k, *'rating\_valuemoney'*])):

rating\_valuemoney.append(*'NoValue'*)

else:

rating\_valuemoney.append(str(review.loc[k, *'rating\_valuemoney'*]))

if (math.isnan(review.loc[k, *'recommended'*])):

recommended.append(*'NoValue'*)

elif review.loc[k, *'recommended'*] == 1:

recommended.append(*'Yes'*)

elif review.loc[k, *'recommended'*] == 0:

recommended.append(*'No'*)

sentiment.append(str(review.loc[k, *'sentiment'*]))

tx = graph.begin()

for k in range(0, len(review)):

tx.run(statement, {*"A"*: review.loc[k,*'authorname'*],

*"B"*: review.loc[k,*'airlinename'*],

*"C"*: authorcountry[k],

*"D"*: aircraft[k],

*"E"*: route[k],

*"F"*: travellertype[k],

*"G"*: cabin[k],

*"H"*: rating\_cabinstaff[k],

*"I"*: rating\_foodbeverage[k],

*"J"*: rating\_inflightEnt[k],

*"K"*: rating\_overall[k],

*"L"*: rating\_seatcomfort[k],

*"M"*: rating\_valuemoney[k],

*"N"*: recommended[k],

*"O"*: review.loc[k,*'reviewcontent'*],

*"P"*: review.loc[k,*'reviewdate'*],

*"Q"*: sentiment[k]})

tx.commit()

## ***Part 2***

**Comparison of direct airline competitors as rated and perceived by common raters**

**Python Code:**

import pandas as pd

import csv

import math

#py2neo 3.1.2

import py2neo

from py2neo import \*

py2neo.authenticate(*"localhost:7474"*, *"neo4j"*, *"termproject"*)

graph = Graph(*"http://localhost:7474/db/data/"*)

#2 Comparison between direct competitors

results = graph.run(*"""MATCH (ai1: Airline)<-[r1:RATED]-(au: Author)-[r2:RATED]->(ai2: Airline)*

*WHERE NOT (ai1 = ai2)*

*WITH ai1, ai2, au, r1, r2,*

*CASE toInt(r1.sentiment) > toInt(r2.sentiment)*

*WHEN True THEN ai1.name*

*ELSE ai2.name END as BetterAirline,*

*CASE toInt(r1.rating\_cabinstaff) > toInt(r2.rating\_cabinstaff)*

*WHEN True THEN ai1.name*

*ELSE ai2.name END as BetterCabinStaff,*

*CASE toInt(r1.rating\_foodbeverage) > toInt(r2.rating\_foodbeverage)*

*WHEN True THEN ai1.name*

*ELSE ai2.name END as BetterFoodBeverage,*

*CASE toInt(r1.rating\_inflightEnt) > toInt(r2.rating\_inflightEnt)*

*WHEN True THEN ai1.name*

*ELSE ai2.name END as BetterInFlightEntertainment,*

*CASE toInt(r1.rating\_seatcomfort) > toInt(r2.rating\_seatcomfort)*

*WHEN True THEN ai1.name*

*ELSE ai2.name END as BetterSeatComfort,*

*CASE toInt(r1.rating\_valuemoney) > toInt(r2.rating\_valuemoney)*

*WHEN True THEN ai1.name*

*ELSE ai2.name END as BetterValueForMoney,*

*CASE toInt(r1.recommended) > toInt(r2.recommended)*

*WHEN True THEN ai1.name*

*ELSE ai2.name END as RecommendedAirline*

*Return au.name, ai1.name, r1.sentiment,*

*ai2.name, r2.sentiment, BetterAirline,*

*RecommendedAirline, BetterCabinStaff, BetterFoodBeverage,*

*BetterInFlightEntertainment, BetterSeatComfort, BetterValueForMoney"""*)

for result in results:

print(result)

print()

**Sample Output (Detailed output is mentioned in OutputResults.docx):**

('au.name': 'Anders Pedersen',

'ai1.name': 'vietjetair',

'r1.sentiment': '-5',

'ai2.name': 'turkish-airlines',

'r2.sentiment': '5',

'BetterAirline': 'turkish-airlines',

'RecommendedAirline': 'turkish-airlines',

'BetterCabinStaff': 'vietjetair',

'BetterFoodBeverage': 'turkish-airlines',

'BetterInFlightEntertainment': 'turkish-airlines',

'BetterSeatComfort': 'turkish-airlines',

'BetterValueForMoney': 'turkish-airlines')

**Justification of solution:**

The comparison is done between direct airline competitors i.e. airlines having common raters. We have compared the ratings of these competitor airlines and shown which out of the two airlines wins in different rating categories.

For example in the above sample output, author/customer ‘Anders Pedersen’ has given the rating for Airlines ‘vietjetair’ and ‘turkish-airlines’. While comparing the ratings, the following results come up:

* After doing sentiment analysis on the review content, Turkish airlines has better or more positive sentiment that vietjetair
* Turkish airlines is recommended over vietjetair and has better In-Flight Food and Beverages, In-Flight Entertainment, Seat Comfort and Value for Money than Vietjetair whereas Vietjetair has better Cabin Staff than Turkish airlines

## ***Part 4.1***

**Finding the count of common airlines used by two raters**

**Python Code:**

import pandas as pd

import csv

import math

#py2neo 3.1.2

import py2neo

from py2neo import \*

py2neo.authenticate(*"localhost:7474"*, *"neo4j"*, *"termproject"*)

graph = Graph(*"http://localhost:7474/db/data/"*)

results = graph.run(*"""MATCH (au1:Author)-[r1:RATED]->(ai:Airline)<-[r2:RATED]-(au2:Author)*

*WHERE NOT (au1 = au2)*

*WITH count(ai) as SimilarityIndex, au1, au2*

*ORDER BY SimilarityIndex DESC LIMIT 20*

*CREATE UNIQUE (au1)-[d:SimilarityIndex]->(au2)*

*SET d.count=SimilarityIndex*

*RETURN d.count as SimilarityIndex, au1.name, au2.name"""*)

for result in results:

print(result)

**Sample Output (Detailed output is mentioned in OutputResults.docx):**

('(au2.name)': 'Bob Motto', '(au1.name)': 'C Cutts', 'SimilarityIndex': 90)

**Justification of solution:**

The query is written to compare two authors/customers in such a way so as to find the number of common airlines rated/used by these authors. For example: Bob Motto and C Cutts both rated 90 common airlines. After getting this index, we are creating a relationship called SimilarityIndex between these authors to store the number of common airlines rated.

## ***Part 4.2***

**Finding the top common flyers for each airline**

**Python Code:**

import pandas as pd

import csv

import math

#py2neo 3.1.2

import py2neo

from py2neo import \*

py2neo.authenticate(*"localhost:7474"*, *"neo4j"*, *"termproject"*)

graph = Graph(*"http://localhost:7474/db/data/"*)

#1 Creating database in Neo4j from .csv file

# Reading records from the csv file

review = pd.read\_csv(*AirlineSentimentResults.csv'*, encoding=*'latin-1'*)

#Using only the first 10000 reviews for analysis

review = review[:10000]

# Creating a set of unique airline names

setAirline = set(review[*'airlinename'*])

statement = *"MATCH (au: Author)-[r:RATED]->(ai: Airline {name:{A}})<-[r2:RATED]-(au2: Author) \*

*WITH count(r) as NumberOfTravels, count(r2) as NumberOfTravels2, ai, au, au2 \*

*WHERE NumberOfTravels > 5 AND NOT (au.name = au2.name)\*

*RETURN DISTINCT(ai.name) as Airline, au.name as Author, \*

*au2.name as Author2, NumberOfTravels, NumberOfTravels2 \*

*ORDER BY NumberOfTravels DESC LIMIT 1"*

tx = graph.begin()

for k in setAirline:

result = tx.run(statement, {*"A"*: k})

print(result.data())

print()

**Sample Output (Detailed output is mentioned in OutputResults.docx):**

[{'Airline': 'british-airways', 'Author': 'Bob Motto', 'Author2': 'C Cutts', 'NumberOfTravels': 90, 'NumberOfTravels2': 90}]

**Justification of solution:**

The query is written to compare two authors/customers in such a way so as to find the top travelers for each airline. This shows the likeness between the authors for each airline.