Code in plain text for viewing:

Code in file finaImpl/preprocessing.ipynb:

from sklearn.linear\_model import SGDClassifier

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Visualization Library

import matplotlib.pyplot as plt

import seaborn as sns

from collections import Counter

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split,StratifiedKFold,GridSearchCV

import pdb

import tensorflow.contrib.keras as keras

from tensorflow.contrib.keras import losses,optimizers,metrics

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import confusion\_matrix,classification\_report

from tensorboard.plugins.hparams import api as hp

train\_df=pd.read\_csv("original\_data/train.csv")

test\_df=pd.read\_csv("original\_data/test.csv")

train\_df = train\_df.drop(['Unnamed: 0', 'id'], axis=1)

test\_df = test\_df.drop(['Unnamed: 0', 'id'], axis=1)

train\_df.columns=[each.replace(" ","\_") for each in train\_df.columns]

test\_df.columns=[each.replace(" ","\_") for each in train\_df.columns]

##Feature engineering

train\_df=pd.get\_dummies(train\_df,columns=["Customer\_Type"])

train\_df=pd.get\_dummies(train\_df,columns=["Type\_of\_Travel"])

train\_df=pd.get\_dummies(train\_df,columns=["Class"])

train\_df=pd.get\_dummies(train\_df,columns=["Gender"])

test\_df=pd.get\_dummies(test\_df,columns=["Customer\_Type"])

test\_df=pd.get\_dummies(test\_df,columns=["Type\_of\_Travel"])

test\_df=pd.get\_dummies(test\_df,columns=["Class"])

test\_df=pd.get\_dummies(test\_df,columns=["Gender"])

training\_data = np.asarray(train\_df)

training\_count = len(training\_data[:,0])

testing\_data = np.asarray(test\_df)

testing\_count = len(testing\_data[:,0])

def outlierIdentifier(dataFrame,columns):

outlier\_indices=[]

for f in columns:

Quartile1 = np.percentile(dataFrame[f],25)

Quartile3 = np.percentile(dataFrame[f],75)

interquartile\_range = Quartile3-Quartile1

outlier\_step= interquartile\_range \* 1.5

outlier\_list\_col = dataFrame[(dataFrame[f]< Quartile1 - outlier\_step)|( dataFrame[f] > Quartile3 + outlier\_step)].index

outlier\_indices.extend(outlier\_list\_col)

outliers\_indices = Counter(outlier\_indices)

multiple\_outliers = list(i for i , v in outliers\_indices.items() if v>2 )

return multiple\_outliers

# remove outliers

train\_df = train\_df.drop(outlierIdentifier(train\_df,[ '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', 'On-board\_service', 'Leg\_room\_service',

'Baggage\_handling', 'Checkin\_service', 'Inflight\_service',

'Cleanliness', 'Departure\_Delay\_in\_Minutes',

'Arrival\_Delay\_in\_Minutes']),axis = 0).reset\_index(drop = True)

train\_df.isna().any()

test\_df.isna().any()

train\_df\_len=len(train\_df)

train\_df= pd.concat([train\_df,test\_df],axis=0).reset\_index(drop=True)

train\_df["Arrival\_Delay\_in\_Minutes"]=train\_df["Arrival\_Delay\_in\_Minutes"].fillna(np.mean(train\_df["Arrival\_Delay\_in\_Minutes"]))

train\_df.isna().any()

X\_test=train\_df[train\_df\_len:]

X\_train=train\_df[:train\_df\_len]

y\_train = X\_train.satisfaction

y\_test = X\_test.satisfaction

X\_train.drop(labels=["satisfaction"],axis=1,inplace=True)

X\_test.drop(labels=["satisfaction"],axis=1,inplace=True)

#Below causes problem when using pd.dummies later

#y\_train = y\_train.eq('satisfied').mul(1)

#y\_test = y\_test.eq('satisfied').mul(1)

try:

os.stat('processed\_data')

except:

os.mkdir('processed\_data')

X\_train.to\_csv("processed\_data/X\_train.csv")

X\_test.to\_csv("processed\_data/X\_test.csv")

y\_train.to\_csv("processed\_data/y\_train.csv")

y\_test.to\_csv("processed\_data/y\_test.csv")

print("X\_train",len(X\_train))

print("X\_test",len(X\_test))

print("y\_train",len(y\_train))

print("y\_test",len(y\_test))

Code in file finaImpl/HyperParameterTuning.ipynb :

from sklearn.linear\_model import SGDClassifier

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Visualization Library

import matplotlib.pyplot as plt

import seaborn as sns

from collections import Counter

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split,StratifiedKFold,GridSearchCV

import pdb

import tensorflow.contrib.keras as keras

from tensorflow.contrib.keras import losses,optimizers,metrics

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import confusion\_matrix,classification\_report

from tensorboard.plugins.hparams import api as hp

import os.path

X\_train=pd.read\_csv("processed\_data/X\_train.csv")

X\_test=pd.read\_csv("processed\_data/X\_test.csv")

y\_train=pd.read\_csv("processed\_data/y\_train.csv")

y\_test=pd.read\_csv("processed\_data/y\_test.csv")

try:

os.stat('saved\_models')

except:

os.mkdir('saved\_models')

scaler = MinMaxScaler()

X\_train = X\_train.drop(['Unnamed: 0'], axis=1)

X\_train = scaler.fit\_transform(X\_train)

y\_train = pd.get\_dummies(y\_train).astype(float)

y\_train = y\_train.drop(['Unnamed: 0'], axis=1)

X\_test = X\_test.drop(['Unnamed: 0'], axis=1)

X\_test = scaler.fit\_transform(X\_test)

y\_test = pd.get\_dummies(y\_test).astype(float)

y\_test = y\_test.drop(['Unnamed: 0'], axis=1)

print("X\_train",len(X\_train))

print("X\_test",len(X\_test))

print("y\_train",len(y\_train))

print("y\_test",len(y\_test))

np.random.seed(101)

tf.set\_random\_seed(101)

HP\_OPTIMIZER = hp.HParam('optimizer', hp.Discrete(['adam', 'sgd']))

HP\_NUM\_UNITS1 = hp.HParam('num\_units 1', hp.Discrete([50,100, 150]))

HP\_NUM\_UNITS2 = hp.HParam('num\_units 2', hp.Discrete([50,100, 150]))

METRIC\_ACCURACY = 'accuracy'

def train\_test\_model(run\_dir, hparams, identifier):

model = tf.keras.Sequential([

keras.layers.Dense(len(X\_train), input\_dim=27, activation='relu'),

keras.layers.Dense(hparams[HP\_NUM\_UNITS1], activation='relu'),

keras.layers.Dense(hparams[HP\_NUM\_UNITS2], activation='relu'),

keras.layers.Dense(2, activation='softmax')

])

model.compile(optimizer=hparams[HP\_OPTIMIZER],

loss='binary\_crossentropy',

metrics=['accuracy'])

tensorboard\_callback = keras.callbacks.TensorBoard(log\_dir=run\_dir)

training\_history = model.fit(X\_train,y\_train, epochs=50, callbacks=[tensorboard\_callback])

model.save('saved\_models/'+identifier+'.h5')

metrics = model.evaluate(X\_test, y\_test)

return metrics

def run(run\_dir, hparams, identifier):

with tf.summary.FileWriter(run\_dir) as writer:

hp.hparams(hparams) # record the values used in this trial

metrics = train\_test\_model(run\_dir, hparams, identifier)

tf.summary.scalar(METRIC\_ACCURACY, metrics)

writer.flush()

return metrics

hparam\_df = pd.DataFrame(columns = ['identifier','num\_units1', 'num\_units2', 'optimizer', 'loss', 'accuracy'])

if os.path.isfile("hparam.csv"):

hparam\_df=pd.read\_csv("hparam.csv")

else:

hparam\_df.to\_csv("hparam.csv")

for num\_units1 in HP\_NUM\_UNITS1.domain.values:

for num\_units2 in HP\_NUM\_UNITS2.domain.values:

for optimizer in HP\_OPTIMIZER.domain.values:

identifier = str(num\_units1)+"\_"+str(num\_units2)+"\_"+optimizer;

if os.path.isfile('saved\_models/'+identifier+'.h5') and hparam\_df['identifier'].str.contains(identifier).any() and os.path.isdir('logs/hparam\_tuning/'+identifier+'/train/'):

continue

hparams = {

HP\_NUM\_UNITS1: num\_units1,

HP\_NUM\_UNITS2: num\_units2,

HP\_OPTIMIZER: optimizer

}

print('--- Starting trial: %s' % identifier)

print({h.name: hparams[h] for h in hparams})

metrics = run('logs/hparam\_tuning/' + identifier, hparams, identifier)

hp\_append = {'identifier' : identifier,'num\_units1': num\_units1, 'num\_units2': num\_units2, 'optimizer': optimizer, 'loss':metrics[0], 'accuracy':metrics[1]}

hparam\_df = hparam\_df.append(hp\_append, ignore\_index = True)

hparam\_df.to\_csv("hparam.csv")

combined\_X = np.concatenate((X\_train, X\_test))

combined\_y = np.concatenate((y\_train, y\_test))

final\_mlp\_model = tf.keras.models.load\_model('saved\_models/50\_50\_adam.h5')

history = final\_mlp\_model.fit(combined\_X,combined\_y, epochs=50, validation\_split=0.200061,)

plt.plot(history.history['acc'])

plt.plot(history.history['val\_acc'])

plt.title('model accuracy')

plt.ylabel('accuracy')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

Code in file finaImpl/ Model Evaluation and Comparisons.ipynb:

from sklearn.linear\_model import SGDClassifier

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Visualization Library

import matplotlib.pyplot as plt

import seaborn as sns

from collections import Counter

import tensorflow as tf

import pdb

import tensorflow.contrib.keras as keras

from tensorflow.contrib.keras import losses,optimizers,metrics

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import confusion\_matrix,classification\_report, plot\_confusion\_matrix, plot\_roc\_curve, precision\_recall\_curve

from tensorboard.plugins.hparams import api as hp

import os.path

from sklearn import model\_selection

from sklearn.model\_selection import train\_test\_split,StratifiedKFold,GridSearchCV

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier,VotingClassifier, GradientBoostingClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.svm import SVC

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.metrics import cohen\_kappa\_score

from sklearn.metrics import roc\_auc\_score

from sklearn.metrics import roc\_curve

from sklearn.metrics import auc

from os import listdir

from os.path import isfile, join

import joblib

from datetime import datetime

from sklearn.neural\_network import MLPClassifier

import sys

from io import StringIO

X\_train=pd.read\_csv("processed\_data/X\_train.csv")

X\_test=pd.read\_csv("processed\_data/X\_test.csv")

y\_train=pd.read\_csv("processed\_data/y\_train.csv")

y\_test=pd.read\_csv("processed\_data/y\_test.csv")

try:

os.stat('comparison\_saved\_models')

except:

os.mkdir('comparison\_saved\_models')

scaler = MinMaxScaler()

X\_train = X\_train.drop(['Unnamed: 0'], axis=1)

X\_train = scaler.fit\_transform(X\_train)

y\_train\_bool = y\_train.satisfaction.eq('satisfied').mul(1)

y\_train = pd.get\_dummies(y\_train).astype(float)

y\_train = y\_train.drop(['Unnamed: 0'], axis=1)

X\_test = X\_test.drop(['Unnamed: 0'], axis=1)

X\_test = scaler.fit\_transform(X\_test)

y\_test\_bool = y\_test.satisfaction.eq('satisfied').mul(1)

y\_test = pd.get\_dummies(y\_test).astype(float)

y\_test = y\_test.drop(['Unnamed: 0'], axis=1)

print("X\_train",len(X\_train))

print("X\_test",len(X\_test))

print("y\_train",len(y\_train))

print("y\_test",len(y\_test))

print("y\_train\_bool",len(y\_train\_bool))

print("y\_test\_bool",len(y\_test\_bool))

np.random.seed(101)

tf.set\_random\_seed(101)

y\_train\_bool

LogisticRegression().fit

classifier\_hparam\_matrix = [

[LogisticRegression(random\_state=101), {"C":np.logspace(-1,1,7), "penalty":["l1","l2"]}, 'LogisticRegression'],

[RandomForestClassifier(random\_state=101), {"max\_features":[2,4,10],

"min\_samples\_split":[1,4,10],

"bootstrap":[False]}, 'RandomForestClassifier'],

[GaussianNB(), {'var\_smoothing': np.logspace(0,-11, num=100)}, 'GaussianNB'],

[DecisionTreeClassifier(random\_state=101), {"min\_samples\_split":range(10,100,20),

"max\_depth":range(1,25,2)}, 'DecisionTreeClassifier'],

[KNeighborsClassifier(), {"n\_neighbors": np.linspace(5,15,10,dtype=int).tolist(),

"weights":["distance"],

"metric":["manhattan"]}, 'KNeighborsClassifier'],

[GradientBoostingClassifier(random\_state=101), {

"min\_samples\_split": [2,5,10],

"min\_samples\_leaf": [1,2,5],

"max\_depth":[3,5],

"max\_features":["log2","sqrt"],

"n\_estimators":[10, 100]

}, 'GradientBoostingClassifier'],

[SVC(random\_state=101),{'C': [0.5,1, 10],

'kernel': ['rbf']},

'SVC'],

]

grid\_scores = pd.DataFrame(columns = ['identifier','best\_score', 'best\_estimator'])

if os.path.isfile("gridCVscores.csv"):

grid\_scores=pd.read\_csv("gridCVscores.csv")

else:

grid\_scores.to\_csv("gridCVscores.csv")

for i in range(len(classifier\_hparam\_matrix)):

if os.path.isfile('comparison\_saved\_models/'+classifier\_hparam\_matrix[i][2]+'.pkl') and grid\_scores['identifier'].str.contains(classifier\_hparam\_matrix[i][2]).any():

continue

now = datetime.now()

current\_time = now.strftime("%H:%M:%S")

print(classifier\_hparam\_matrix[i][2]+' start :::::: '+current\_time)

grid=GridSearchCV(classifier\_hparam\_matrix[i][0],param\_grid=classifier\_hparam\_matrix[i][1],cv=StratifiedKFold(n\_splits=10),scoring="accuracy",n\_jobs=-1,verbose=1)

grid.fit(X\_train,y\_train\_bool)

joblib.dump(grid.best\_estimator\_, 'comparison\_saved\_models/'+classifier\_hparam\_matrix[i][2]+'.pkl')

grid\_scores\_append = {'identifier' : classifier\_hparam\_matrix[i][2],'best\_score': grid.best\_score\_, 'best\_estimator': grid.best\_estimator\_}

grid\_scores = grid\_scores.append(grid\_scores\_append, ignore\_index = True)

grid\_scores.to\_csv("gridCVscores.csv")

now = datetime.now()

current\_time = now.strftime("%H:%M:%S")

print(classifier\_hparam\_matrix[i][2]+' End \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* '+current\_time)

onlyfiles = [f for f in listdir('comparison\_saved\_models/') if isfile(join('comparison\_saved\_models/', f))]

compare\_models = []

compare\_model\_names=[]

for i in range(len(onlyfiles)):

loaded\_model = joblib.load('comparison\_saved\_models/'+onlyfiles[i])

compare\_models.append(loaded\_model)

compare\_model\_names.append(onlyfiles[i].split('.')[0])

models\_roc\_auc = []

models\_prec\_recall = []

for i in range(len(compare\_model\_names)):

print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

print(compare\_model\_names[i])

predictions = compare\_models[i].predict(X\_test)

fpr, tpr, thresholds = roc\_curve(y\_test\_bool, predictions)

auc\_val = auc(fpr, tpr)

models\_roc\_auc.append([fpr, tpr, thresholds, auc\_val])

precision, recall, thresholds = precision\_recall\_curve(y\_test\_bool, predictions)

models\_prec\_recall.append([precision, recall, thresholds])

print(classification\_report(predictions,y\_test\_bool))

print(confusion\_matrix(predictions,y\_test\_bool))

plot\_confusion\_matrix(compare\_models[i], X\_test, y\_test\_bool,

display\_labels=['neutral or \ndissatisfied', 'satisfied'],

cmap=plt.cm.Blues, values\_format = 'd')

mlp\_model = tf.keras.models.load\_model('saved\_models/50\_50\_adam.h5')

predictions = mlp\_model.predict\_classes(X\_test)

fpr\_mlp, tpr\_mlp, thresholds\_mlp = roc\_curve(y\_test\_bool, predictions)

auc\_mlp = auc(fpr\_mlp, tpr\_mlp)

mlp\_precision, mlp\_recall, mlp\_thresholds = precision\_recall\_curve(y\_test\_bool, predictions)

print(classification\_report(predictions,y\_test\_bool))

print(confusion\_matrix(predictions,y\_test\_bool))

plt.figure(1)

plt.plot([0, 1], [0, 1], 'k--')

plt.plot(fpr\_mlp, tpr\_mlp, label='MLP (area = {:.3f})'.format(auc\_mlp))

for i in range(len(models\_roc\_auc)):

plt.plot(models\_roc\_auc[i][0], models\_roc\_auc[i][1], label=compare\_model\_names[i]+' (area = {:.3f})'.format(models\_roc\_auc[i][3]))

plt.xlabel('False positive rate')

plt.ylabel('True positive rate')

plt.title('ROC curve')

plt.legend(loc='best')

plt.show()

plt.figure(2)

plt.xlim(0, 0.2)

plt.ylim(0.8, 1)

plt.plot([0, 1], [0, 1], 'k--')

plt.plot(fpr\_mlp, tpr\_mlp, label='MLP (area = {:.3f})'.format(auc\_mlp))

for i in range(len(models\_roc\_auc)):

plt.plot(models\_roc\_auc[i][0], models\_roc\_auc[i][1], label=compare\_model\_names[i]+' (area = {:.3f})'.format(models\_roc\_auc[i][3]))

plt.xlabel('False positive rate')

plt.ylabel('True positive rate')

plt.title('ROC curve (zoomed in at top left)')

plt.legend(loc='best')

plt.show()

baseline = len(y\_test\_bool[y\_test\_bool==1]) / len(y\_test\_bool)

plt.figure(1)

plt.plot(mlp\_recall, mlp\_precision, label='MLP')

for i in range(len(models\_prec\_recall)):

plt.plot(models\_prec\_recall[i][1], models\_prec\_recall[i][0], label=compare\_model\_names[i])

plt.plot([0, 1], [baseline, baseline], linestyle='--', label='Baseline')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision Recall curve')

plt.legend(loc='best')

plt.show()

plt.figure(2)

plt.xlim(0, 0.2)

plt.ylim(0.8, 1)

plt.plot(mlp\_recall, mlp\_precision, label='MLP')

for i in range(len(models\_roc\_auc)):

plt.plot(models\_prec\_recall[i][1], models\_prec\_recall[i][0], label=compare\_model\_names[i])

plt.plot([0, 1], [baseline, baseline], linestyle='--', label='Baseline')

plt.xlabel('Recall')

plt.ylabel('Precision')

plt.title('Precision Recall curve (zoomed in at top left)')

plt.legend(loc='best')

plt.show()

Code in file finaImpl/EDA.ipynb:

from sklearn.linear\_model import SGDClassifier

import numpy as np # linear algebra

import pandas as pd # data processing, CSV file I/O (e.g. pd.read\_csv)

# Visualization Library

import matplotlib.pyplot as plt

import seaborn as sns

from collections import Counter

import tensorflow as tf

from sklearn.model\_selection import train\_test\_split,StratifiedKFold,GridSearchCV

import pdb

import tensorflow.contrib.keras as keras

from tensorflow.contrib.keras import losses,optimizers,metrics

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import confusion\_matrix,classification\_report

from tensorboard.plugins.hparams import api as hp

import os.path

import math

import statsmodels.api as sm

from statsmodels.formula.api import ols

train\_df=pd.read\_csv("original\_data/train.csv")

test\_df=pd.read\_csv("original\_data/test.csv")

combined\_df= pd.concat([train\_df,test\_df],axis=0).reset\_index(drop=True)

try:

os.stat('saved\_stats')

except:

os.mkdir('saved\_stats')

combined\_df.shape

combined\_df.info()

combined\_df.columns

combined\_df.describe()

combined\_df.dtypes

combined\_df.head(5)

combined\_df = combined\_df.drop(['Unnamed: 0', 'id'], axis=1)

combined\_df.columns=[each.replace(" ","\_") for each in combined\_df.columns]

combined\_df['satisfaction\_int'] = combined\_df.satisfaction.eq('satisfied').mul(1)

num\_feat = ['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', 'On-board\_service', 'Leg\_room\_service',

'Baggage\_handling', 'Checkin\_service', 'Inflight\_service',

'Cleanliness', 'Departure\_Delay\_in\_Minutes', 'Arrival\_Delay\_in\_Minutes']

fig = plt.figure(figsize=(15,30))

i = 1

for each\_feature in num\_feat:

fig.add\_subplot(8,3,i)

sns.boxplot(y=combined\_df[each\_feature])

i = i+1

plt.show()

def outlierIdentifier(dataFrame,columns):

outlier\_indices=[]

for f in columns:

Quartile1 = np.percentile(dataFrame[f],25)

Quartile3 = np.percentile(dataFrame[f],75)

interquartile\_range = Quartile3-Quartile1

outlier\_step= interquartile\_range \* 1.5

outlier\_list\_col = dataFrame[(dataFrame[f]< Quartile1 - outlier\_step)|( dataFrame[f] > Quartile3 + outlier\_step)].index

outlier\_indices.extend(outlier\_list\_col)

outliers\_indices = Counter(outlier\_indices)

multiple\_outliers = list(i for i , v in outliers\_indices.items() if v>2 )

return multiple\_outliers

# remove outliers

combined\_df = combined\_df.drop(outlierIdentifier(combined\_df,[ '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', 'On-board\_service', 'Leg\_room\_service',

'Baggage\_handling', 'Checkin\_service', 'Inflight\_service',

'Cleanliness', 'Departure\_Delay\_in\_Minutes',

'Arrival\_Delay\_in\_Minutes']),axis = 0).reset\_index(drop = True)

combined\_df.isna().any()

combined\_df["Arrival\_Delay\_in\_Minutes"]=combined\_df["Arrival\_Delay\_in\_Minutes"].fillna(np.mean(combined\_df["Arrival\_Delay\_in\_Minutes"]))

combined\_df.shape

combined\_df.describe()

columns = combined\_df.columns

customer\_details\_columns = ['Gender', 'Customer\_Type',

'Type\_of\_Travel', 'Class']

fig,axes = plt.subplots(2,2,figsize=(12,15))

for i,cat in enumerate(customer\_details\_columns):

row,col = i//2,i%2

sns.countplot(x=cat,data=combined\_df,hue='satisfaction',ax=axes[row,col])

plt.subplots\_adjust(hspace=1)

service\_rating\_columns = ['Inflight\_wifi\_service',

'Departure/Arrival\_time\_convenient', 'Ease\_of\_Online\_booking',

'Gate\_location', 'Food\_and\_drink', 'Online\_boarding', 'Seat\_comfort',

'Inflight\_entertainment', 'On-board\_service', 'Leg\_room\_service',

'Baggage\_handling', 'Checkin\_service', 'Inflight\_service',

'Cleanliness']

disloyal\_pax = combined\_df[combined\_df['Customer\_Type']=='disloyal Customer']

fig,axes = plt.subplots(7,2,figsize=(12,20))

for i,cat in enumerate(service\_rating\_columns):

row,col = i//2,i%2

sns.countplot(x=cat,data=disloyal\_pax,hue='satisfaction',ax=axes[row,col])

plt.subplots\_adjust(hspace=1)

service\_rating\_columns = ['Inflight\_wifi\_service',

'Departure/Arrival\_time\_convenient', 'Ease\_of\_Online\_booking',

'Gate\_location', 'Food\_and\_drink', 'Online\_boarding', 'Seat\_comfort',

'Inflight\_entertainment', 'On-board\_service', 'Leg\_room\_service',

'Baggage\_handling', 'Checkin\_service', 'Inflight\_service',

'Cleanliness']

personal\_trvl\_pax = combined\_df[combined\_df['Type\_of\_Travel']=='Personal Travel']

fig,axes = plt.subplots(7,2,figsize=(12,20))

for i,cat in enumerate(service\_rating\_columns):

row,col = i//2,i%2

sns.countplot(x=cat,data=personal\_trvl\_pax,hue='satisfaction',ax=axes[row,col])

plt.subplots\_adjust(hspace=1)

service\_rating\_columns = ['Inflight\_wifi\_service',

'Departure/Arrival\_time\_convenient', 'Ease\_of\_Online\_booking',

'Gate\_location', 'Food\_and\_drink', 'Online\_boarding', 'Seat\_comfort',

'Inflight\_entertainment', 'On-board\_service', 'Leg\_room\_service',

'Baggage\_handling', 'Checkin\_service', 'Inflight\_service',

'Cleanliness']

eco\_pax = combined\_df[combined\_df['Class']!='Business']

fig,axes = plt.subplots(7,2,figsize=(12,20))

for i,cat in enumerate(service\_rating\_columns):

row,col = i//2,i%2

sns.countplot(x=cat,data=personal\_trvl\_pax,hue='satisfaction',ax=axes[row,col])

plt.subplots\_adjust(hspace=1)

service\_rating\_columns = [ 'Inflight\_wifi\_service',

'Departure/Arrival\_time\_convenient', 'Ease\_of\_Online\_booking',

'Gate\_location', 'Food\_and\_drink', 'Online\_boarding', 'Seat\_comfort',

'Inflight\_entertainment', 'On-board\_service', 'Leg\_room\_service',

'Baggage\_handling', 'Checkin\_service', 'Inflight\_service',

'Cleanliness']

fig,axes = plt.subplots(7,2,figsize=(12,20))

for i,cat in enumerate(service\_rating\_columns):

row,col = i//2,i%2

sns.countplot(x=cat,data=combined\_df,hue='satisfaction',ax=axes[row,col])

plt.subplots\_adjust(hspace=1)

sns.distplot(combined\_df[combined\_df['satisfaction']=='satisfied'].Age, label='satisfied')

sns.distplot(combined\_df[combined\_df['satisfaction']!='satisfied'].Age, label='neutral or dissatisfied')

plt.legend()

plt.show()

sns.distplot(combined\_df[combined\_df['satisfaction']=='satisfied'].Flight\_Distance, label='satisfied')

sns.distplot(combined\_df[combined\_df['satisfaction']!='satisfied'].Flight\_Distance, label='neutral or dissatisfied')

plt.legend()

plt.show()

sns.distplot(combined\_df[combined\_df['satisfaction']=='satisfied'].Departure\_Delay\_in\_Minutes, label='satisfied')

sns.distplot(combined\_df[combined\_df['satisfaction']!='satisfied'].Departure\_Delay\_in\_Minutes, label='neutral or dissatisfied')

plt.legend()

plt.show()

sns.distplot(combined\_df[combined\_df['satisfaction']=='satisfied'].Arrival\_Delay\_in\_Minutes, label='satisfied')

sns.distplot(combined\_df[combined\_df['satisfaction']!='satisfied'].Arrival\_Delay\_in\_Minutes, label='neutral or dissatisfied')

plt.legend()

plt.show()

service\_rating\_columns = ['Inflight\_wifi\_service',

'Departure/Arrival\_time\_convenient', 'Ease\_of\_Online\_booking',

'Gate\_location', 'Food\_and\_drink', 'Online\_boarding', 'Seat\_comfort',

'Inflight\_entertainment', 'On-board\_service', 'Leg\_room\_service',

'Baggage\_handling', 'Checkin\_service', 'Inflight\_service',

'Cleanliness']

fig,axes = plt.subplots(7,2,figsize=(12,30))

for i, ser\_col in enumerate(service\_rating\_columns):

#plt.figure(i)

row,col = i//2,i%2

#sns.countplot(x = ser\_col, color='blue', data=combined\_df,ax=axes[row,col])

#sns.palplot(sns.color\_palette("RdBu", n\_colors=7))

sns.countplot(x = ser\_col, palette="vlag", data=combined\_df,ax=axes[row,col])

plt.subplots\_adjust(hspace=0.2)

combined\_df.Inflight\_wifi\_service.value\_counts()

customer\_details\_columns = [ 'Gender', 'Customer\_Type',

'Type\_of\_Travel', 'Class']

for each\_col in customer\_details\_columns:

col\_grp = combined\_df[[each\_col,'satisfaction\_int']].groupby([each\_col])['satisfaction\_int'].agg('mean').reset\_index(name='satisfied')

count\_grp = combined\_df[[each\_col,'satisfaction\_int']].groupby([each\_col])['satisfaction\_int'].count().reset\_index(name='Count')

satisfied\_count = combined\_df[combined\_df['satisfaction\_int']==1].groupby([each\_col])['satisfaction\_int'].count().reset\_index(name='Count')

col\_grp['satisfied\_count'] = satisfied\_count.Count.values

col\_grp['tot\_count'] = count\_grp.Count.values

col\_grp['population\_percentage'] = ((col\_grp['tot\_count']\*100)/len(combined\_df))/100

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(col\_grp)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

col\_grp.to\_csv("saved\_stats/"+each\_col+".csv")

len(combined\_df)

numeric\_columns = ['Age', 'Flight\_Distance']

for each\_col in numeric\_columns:

bins = []

labels = []

last\_i = 0

for i in range(math.floor(combined\_df[each\_col].min()),math.ceil(combined\_df[each\_col].max()),int(combined\_df[each\_col].std())+1) :

range\_start = i

bins.append(i)

std\_val = int(combined\_df[each\_col].std())

range\_end = (i + std\_val) if (i + std\_val) < combined\_df[each\_col].max() else combined\_df[each\_col].max()

labels.append(str(range\_start) + '-' + str(range\_end))

last\_i = i

bins.append(combined\_df[each\_col].max())

temp\_df = combined\_df

temp\_col = each\_col + '\_range'

temp\_df[temp\_col] = pd.cut(temp\_df[each\_col], bins, labels = labels,include\_lowest = True)

col\_grp = temp\_df[[temp\_col,'satisfaction\_int']].groupby([temp\_col])['satisfaction\_int'].agg('mean').reset\_index(name='satisfied')

count\_grp = temp\_df[[temp\_col,'satisfaction\_int']].groupby([temp\_col])['satisfaction\_int'].count().reset\_index(name='Count')

satisfied\_count = temp\_df[temp\_df['satisfaction\_int']==1].groupby([temp\_col])['satisfaction\_int'].count().reset\_index(name='Count')

col\_grp['satisfied\_count'] = satisfied\_count.Count.values

col\_grp['tot\_count'] = count\_grp.Count.values

col\_grp['population\_percentage'] = ((col\_grp['tot\_count']\*100)/len(combined\_df))/100

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(col\_grp)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

col\_grp.to\_csv("saved\_stats/"+each\_col+".csv")

numeric\_columns = ['Departure\_Delay\_in\_Minutes', 'Arrival\_Delay\_in\_Minutes']

for each\_col in numeric\_columns:

bins = []

labels = []

last\_i = 0

for i in range(math.floor(combined\_df[each\_col].min()),math.ceil(combined\_df[each\_col].max()),int(combined\_df[each\_col].std()\*10)+1) :

range\_start = i

bins.append(i)

std\_val = int(combined\_df[each\_col].std()\*10)

range\_end = (i + std\_val) if (i + std\_val) < combined\_df[each\_col].max() else combined\_df[each\_col].max()

labels.append(str(range\_start) + '-' + str(range\_end))

last\_i = i

bins.append(combined\_df[each\_col].max())

temp\_df = combined\_df

temp\_col = each\_col + '\_range'

temp\_df[temp\_col] = pd.cut(temp\_df[each\_col], bins, labels = labels,include\_lowest = True)

col\_grp = temp\_df[[temp\_col,'satisfaction\_int']].groupby([temp\_col])['satisfaction\_int'].agg('mean').reset\_index(name='satisfied')

count\_grp = temp\_df[[temp\_col,'satisfaction\_int']].groupby([temp\_col])['satisfaction\_int'].count().reset\_index(name='Count')

satisfied\_count = temp\_df[temp\_df['satisfaction\_int']==1].groupby([temp\_col])['satisfaction\_int'].count().reset\_index(name='Count')

col\_grp['satisfied\_count'] = satisfied\_count.Count.values

col\_grp['tot\_count'] = count\_grp.Count.values

col\_grp['population\_percentage'] = ((col\_grp['tot\_count']\*100)/len(combined\_df))/100

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(col\_grp)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

col\_grp.to\_csv("saved\_stats/"+each\_col+".csv")

serv\_columns = [ 'Inflight\_wifi\_service',

'Departure/Arrival\_time\_convenient', 'Ease\_of\_Online\_booking',

'Gate\_location', 'Food\_and\_drink', 'Online\_boarding', 'Seat\_comfort',

'Inflight\_entertainment', 'On-board\_service', 'Leg\_room\_service',

'Baggage\_handling', 'Checkin\_service', 'Inflight\_service',

'Cleanliness']

for each\_col in serv\_columns:

col\_grp = combined\_df[[each\_col,'satisfaction\_int']].groupby([each\_col])['satisfaction\_int'].agg('mean').reset\_index(name='satisfied')

count\_grp = combined\_df[[each\_col,'satisfaction\_int']].groupby([each\_col])['satisfaction\_int'].count().reset\_index(name='Count')

satisfied\_count = combined\_df[combined\_df['satisfaction\_int']==1].groupby([each\_col])['satisfaction\_int'].count().reset\_index(name='Count')

if 0 not in satisfied\_count[each\_col].values :

missing\_rating = []

missing\_rating.insert(0, {each\_col: 0, 'Count': 0})

satisfied\_count = pd.concat([pd.DataFrame(missing\_rating), satisfied\_count], ignore\_index=True)

if 0 not in col\_grp[each\_col].values :

missing\_rating = []

missing\_rating.insert(0, {each\_col: 0, 'satisfied': 0})

col\_grp = pd.concat([pd.DataFrame(missing\_rating), col\_grp], ignore\_index=True)

if 0 not in count\_grp[each\_col].values :

missing\_rating = []

missing\_rating.insert(0, {each\_col: 0, 'Count': 0})

count\_grp = pd.concat([pd.DataFrame(missing\_rating), count\_grp], ignore\_index=True)

#if(each\_col=='Baggage\_handling'):

# pdb.set\_trace()

col\_grp['satisfied\_count'] = satisfied\_count.Count.values

col\_grp['tot\_count'] = count\_grp.Count.values

col\_grp['population\_percentage'] = ((col\_grp['tot\_count']\*100)/len(combined\_df))/100

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

print(col\_grp)

print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n")

col\_grp.to\_csv("saved\_stats/"+each\_col.replace("/","")+".csv")

demograph\_grp = combined\_df[['Gender', 'Customer\_Type', 'Type\_of\_Travel', 'Class','satisfaction\_int']].groupby(['Gender', 'Customer\_Type', 'Type\_of\_Travel', 'Class'])['satisfaction\_int'].mean().reset\_index(name='Mean')

model = ols('Mean ~ C(Gender) + C(Customer\_Type) + C(Type\_of\_Travel) + C(Class) + C(Gender):C(Customer\_Type) + C(Gender):C(Type\_of\_Travel) + C(Gender):C(Class) + C(Customer\_Type):C(Type\_of\_Travel) + C(Customer\_Type):C(Class) + C(Type\_of\_Travel):C(Class)', data=demograph\_grp).fit()

sm.stats.anova\_lm(model, typ=2)

temp\_df = combined\_df

temp\_df.columns=[each.replace("/","\_") for each in temp\_df.columns]

temp\_df.columns=[each.replace("-","\_") for each in temp\_df.columns]

service\_grp = temp\_df[['Inflight\_wifi\_service','Departure\_Arrival\_time\_convenient', 'Ease\_of\_Online\_booking','Gate\_location', 'Food\_and\_drink', 'Online\_boarding', 'Seat\_comfort','Inflight\_entertainment', 'On\_board\_service', 'Leg\_room\_service','Baggage\_handling', 'Checkin\_service', 'Inflight\_service','Cleanliness','satisfaction\_int']].groupby(['Inflight\_wifi\_service','Departure\_Arrival\_time\_convenient', 'Ease\_of\_Online\_booking','Gate\_location', 'Food\_and\_drink', 'Online\_boarding', 'Seat\_comfort','Inflight\_entertainment', 'On\_board\_service', 'Leg\_room\_service','Baggage\_handling', 'Checkin\_service', 'Inflight\_service','Cleanliness'])['satisfaction\_int'].mean().reset\_index(name='Mean')

model = ols('Mean ~ C(Inflight\_wifi\_service) + C(Departure\_Arrival\_time\_convenient) + C(Ease\_of\_Online\_booking) + C(Gate\_location) + C(Food\_and\_drink) + C(Online\_boarding) + C(Seat\_comfort) + C(Inflight\_entertainment) + C(On\_board\_service) + C(Leg\_room\_service) + C(Baggage\_handling) + C(Checkin\_service) + C(Inflight\_service) + C(Cleanliness) ', data=service\_grp).fit()

sm.stats.anova\_lm(model, typ=1)

combined\_df.hist(bins=50, figsize=(15,15));

corr\_mat = combined\_df.corr(method='spearman')

f, ax = plt.subplots(figsize=(12, 10))

sns.heatmap(corr\_mat, ax=ax, annot=True, linewidths=0.5,fmt=".2f")

corrmat = combined\_df.corr(method='spearman')

cg = sns.clustermap(corrmat, annot=True, linewidths=0.5);

plt.setp(cg.ax\_heatmap.yaxis.get\_majorticklabels(), rotation=0)

cg