**REPORT  
Analytics and Systems of Big Data**

End Sem Lab Exam

**Yutika Kulwe**

**CED15I017**

**16th April, 2019**

1. Generate descriptive statistics to better understand the dataset using python support.

* How to run the program:
  + i. python 1b.py
* Input: Given BlackFriday Dataset.
* Implementation: Switch case is implemented to select the type of plot you want to see.
* Code:

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import re

import pickle

import time

from nltk.tokenize import TweetTokenizer

import nltk

from nltk import Text

from nltk.tokenize import regexp\_tokenize

from nltk.tokenize import word\_tokenize

from nltk.tokenize import sent\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from nltk.stem import PorterStemmer

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import MultinomialNB

from sklearn.multiclass import OneVsRestClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import GridSearchCV

from sklearn.pipeline import make\_pipeline

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the BlackFriday dataset

df=pd.read\_csv('BlackFriday.csv')

df.dropna(inplace=True)

df.drop(df.columns[0],axis=1,inplace=True)

non\_cat = [f for f in df.columns if df.dtypes[f] != 'object']

cat = [f for f in df.columns if df.dtypes[f] == 'object']

def treat\_missing\_numeric(df,columns,how = 'mean'):

'''

Function to treat missing values in numeric columns

Required Input -

- df = Pandas DataFrame

- columns = List input of all the columns need to be imputed

- how = valid values are 'mean', 'mode', 'median','ffill', numeric value

Expected Output -

- Pandas dataframe with imputed missing value in mentioned columns

'''

if how == 'mean':

for i in columns:

print("Filling missing values with mean for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mean())

elif how == 'mode':

for i in columns:

print("Filling missing values with mode for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mode())

elif how == 'median':

for i in columns:

print("Filling missing values with median for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].median())

elif how == 'ffill':

for i in columns:

print("Filling missing values with forward fill for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(method ='ffill')

elif type(how) == int or type(how) == float:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(how)

else:

print("Missing value fill cannot be completed")

return df

def treat\_missing\_categorical(df,columns,how = 'mode'):

'''

Function to treat missing values in numeric columns

Required Input -

- df = Pandas DataFrame

- columns = List input of all the columns need to be imputed

- how = valid values are 'mode', any string or numeric value

Expected Output -

- Pandas dataframe with imputed missing value in mentioned columns

'''

if how == 'mode':

for i in columns:

print("Filling missing values with mode for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mode()[0])

elif type(how) == str:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(how)

elif type(how) == int or type(how) == float:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(str(how))

else:

print("Missing value fill cannot be completed")

return df

treat\_missing\_numeric(df,non\_cat,how = 'mean')

treat\_missing\_categorical(df,cat,how = 'mode')

print df.describe()

l= df['Product\_ID']

print(df.head())

print "\nMedian:\n"

print df.median()

print ()

f = df['Age'].value\_counts()

def numbers\_to\_strings(argument):

switcher = {

1: "Histogram",

2: "Pie Chart",

3: "Box plot",

}

return switcher.get(argument, "nothing")

if \_\_name\_\_ == "\_\_main\_\_":

print "\n ===============================\n"

print "\nPRESS: \n 1. Histogram \n 2. Pie Chart \n 3. Box plt \n"

argument=input()

print numbers\_to\_strings(argument)

if(argument==1):

df.hist()

plt.show()

if argument==2:

f.plot.pie()

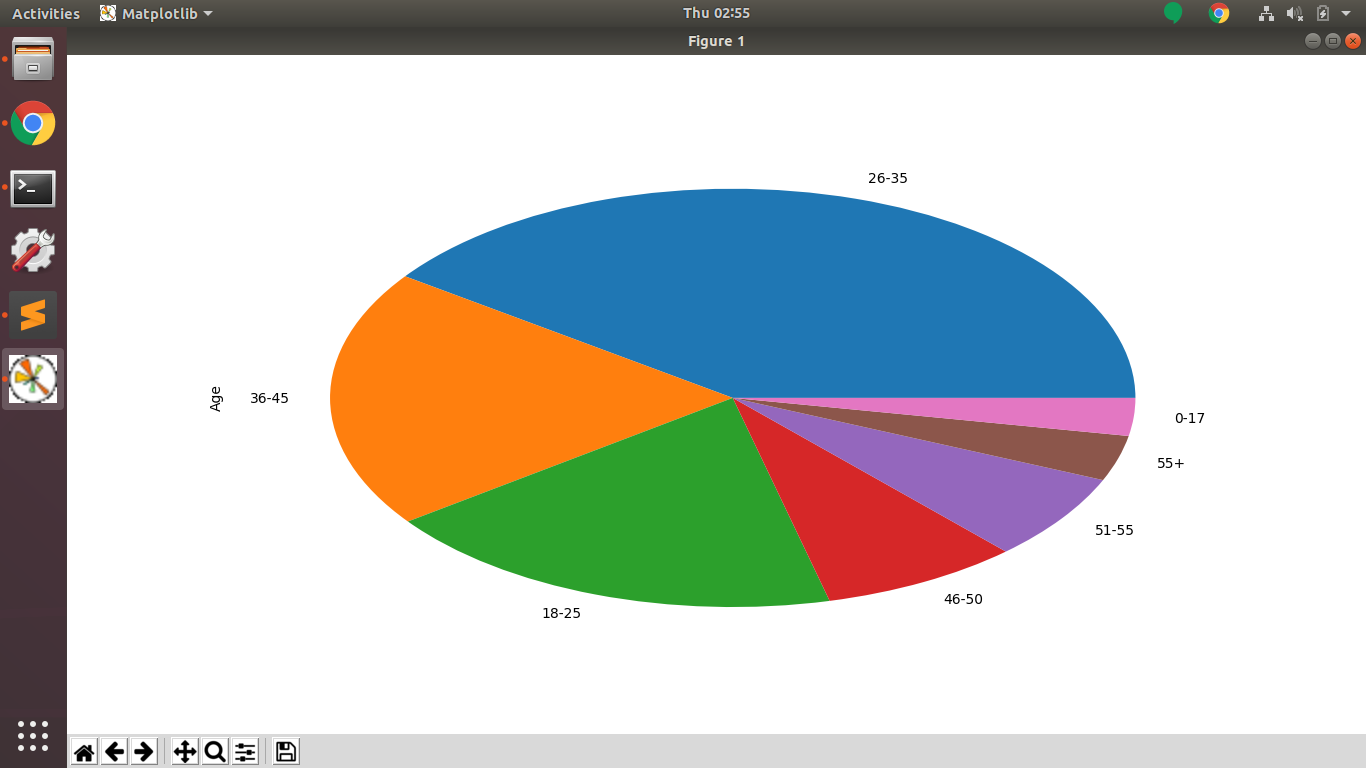
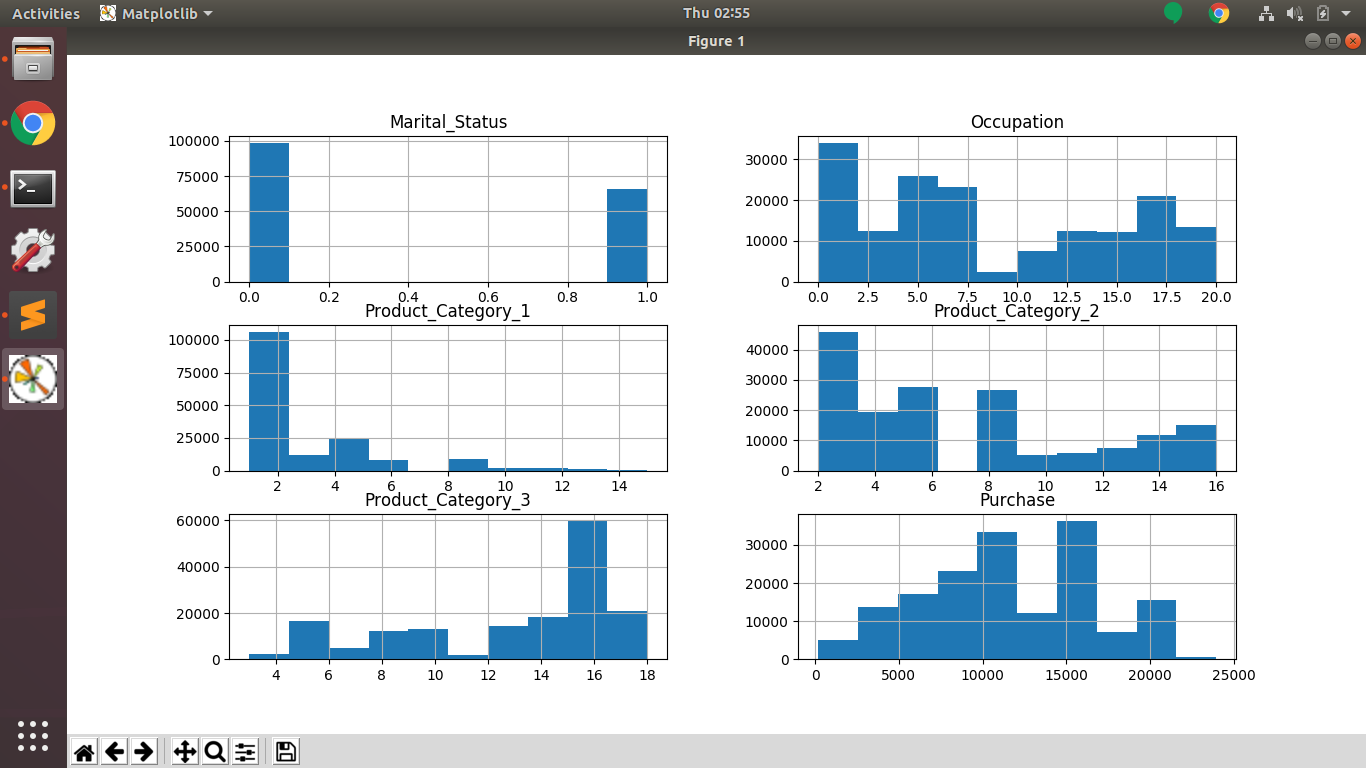
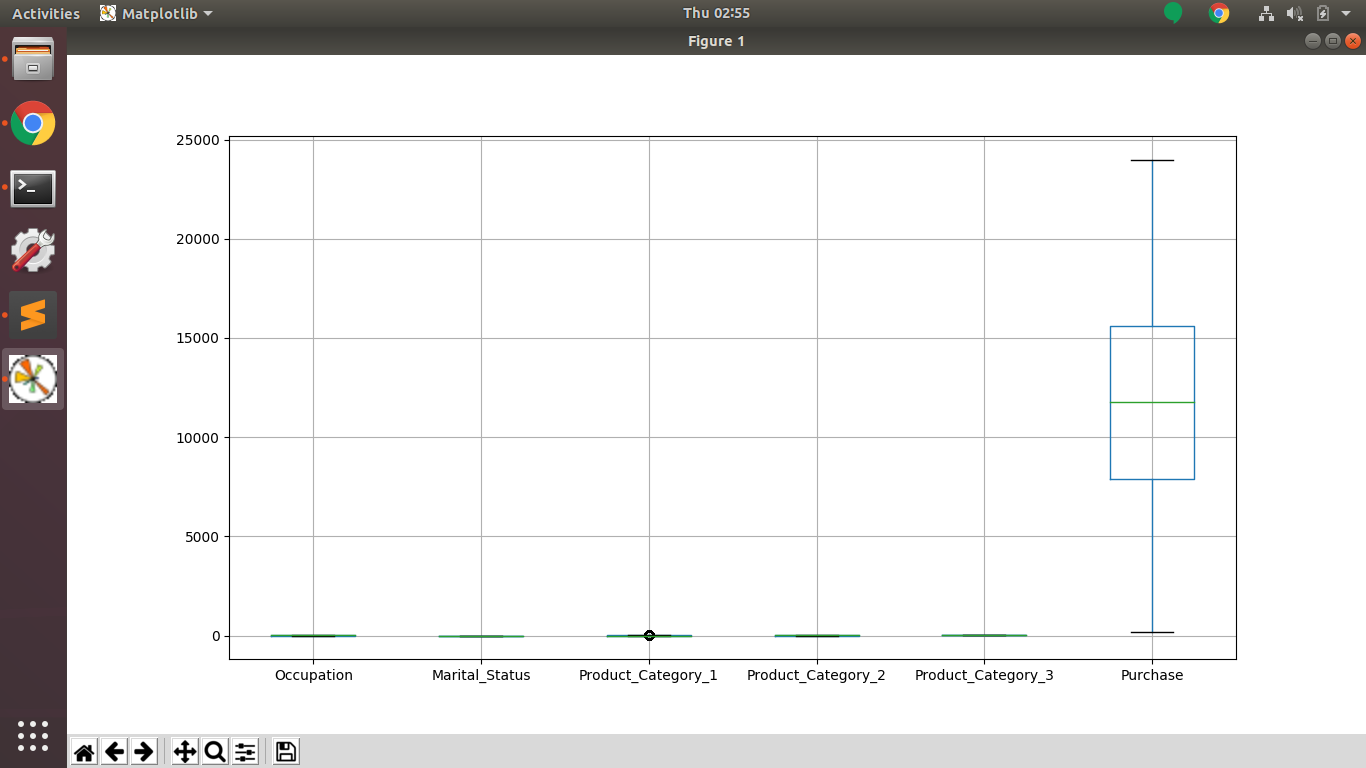
plt.show()

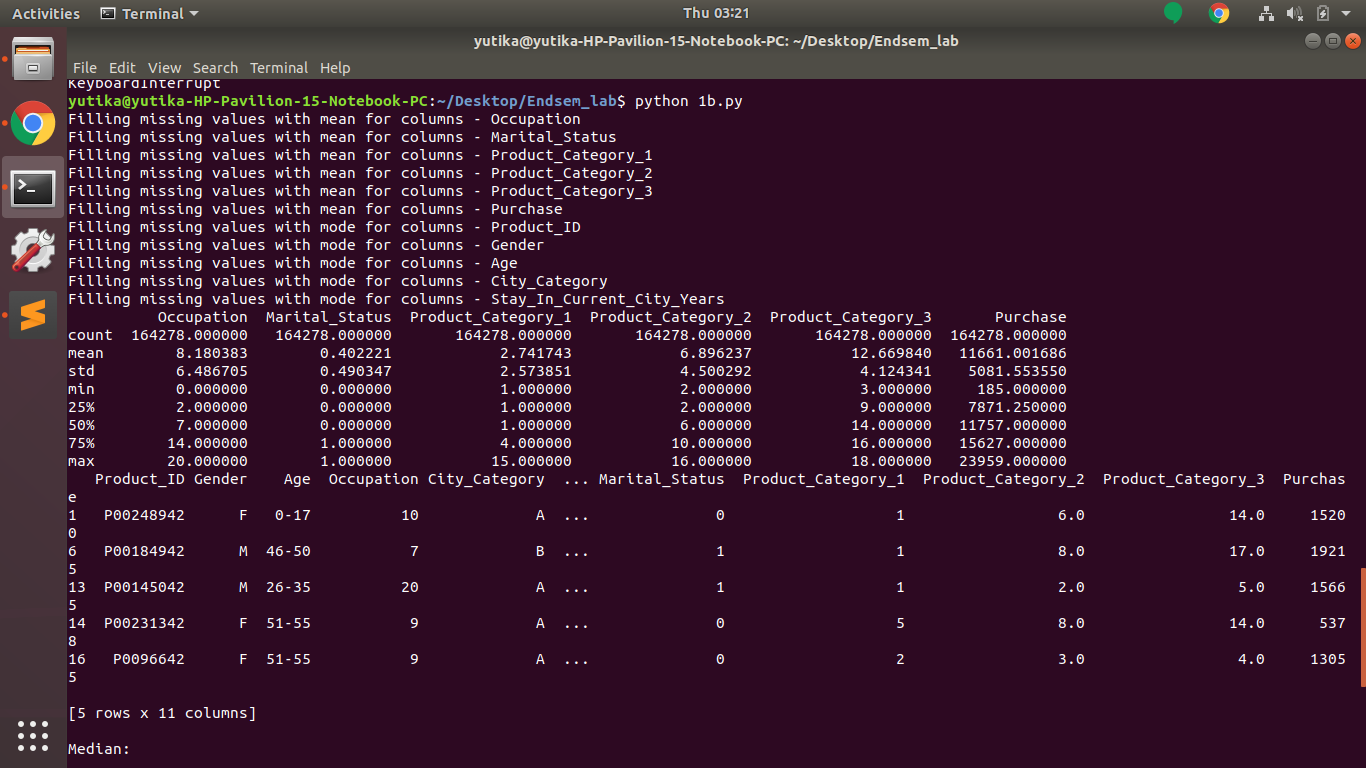
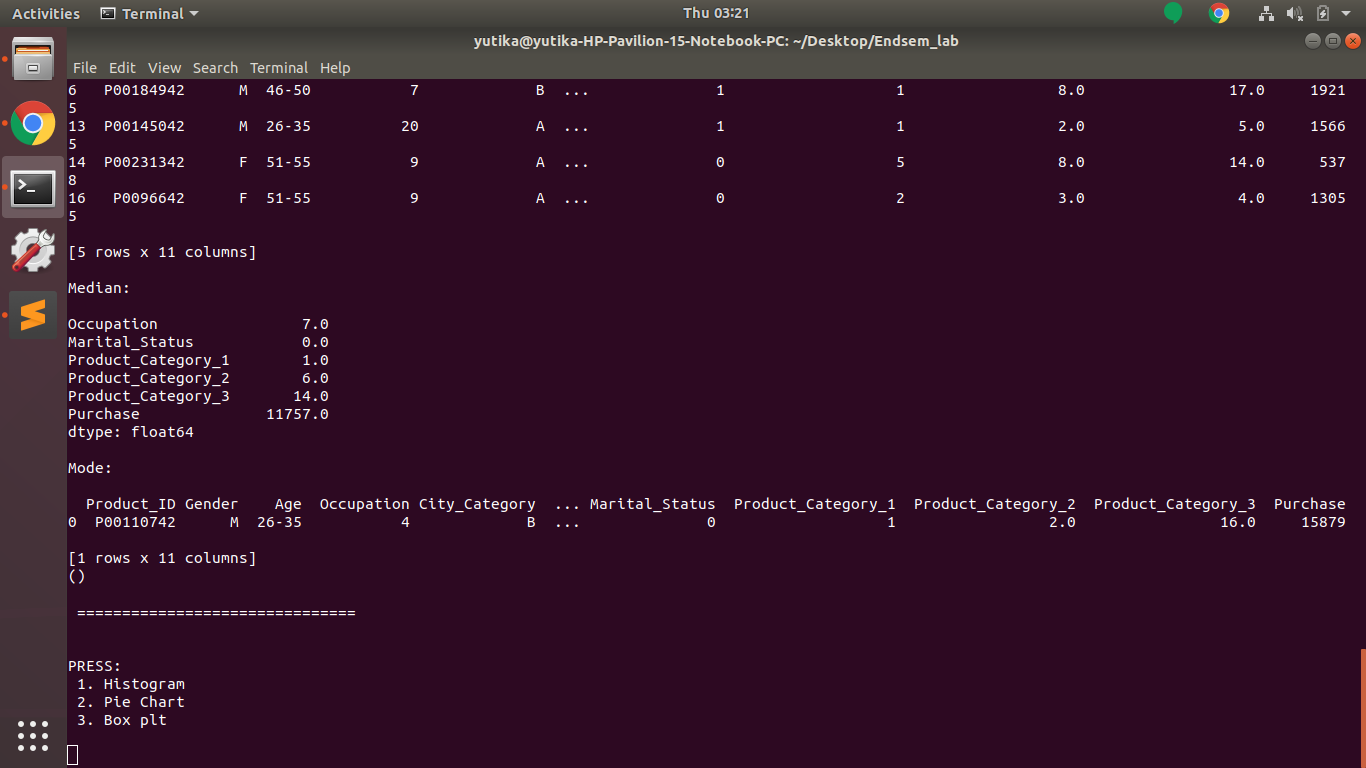
if argument==3:

df.boxplot()

plt.show()

* Output:





2. Handle missing values using any three techniques.

* How to run the program:
  + i. python 2a.py
* Input: Given BlackFriday Dataset.
* Implementation: Handles missing numerical and categorical values by replacing it with mean, mode and median of the data in the columns.
* Code:

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import re

import pickle

import time

from nltk.tokenize import TweetTokenizer # doesn't split at apostrophes

import nltk

from nltk import Text

from nltk.tokenize import regexp\_tokenize

from nltk.tokenize import word\_tokenize

from nltk.tokenize import sent\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from nltk.stem import PorterStemmer

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import MultinomialNB

from sklearn.multiclass import OneVsRestClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import GridSearchCV

from sklearn.pipeline import make\_pipeline

df=pd.read\_csv('BlackFriday.csv')

df.dropna(inplace=True)

#df.drop(df.columns[0],axis=1,inplace=True)

non\_cat = [f for f in df.columns if df.dtypes[f] != 'object']

cat = [f for f in df.columns if df.dtypes[f] == 'object']

def treat\_missing\_numeric(df,columns,how = 'mean'):

'''

Function to treat missing values in numeric columns

Required Input -

- df = Pandas DataFrame

- columns = List input of all the columns need to be imputed

- how = valid values are 'mean', 'mode', 'median','ffill', numeric value

Expected Output -

- Pandas dataframe with imputed missing value in mentioned columns

'''

if how == 'mean':

for i in columns:

print("Filling missing values with mean for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mean())

elif how == 'mode':

for i in columns:

print("Filling missing values with mode for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mode())

elif how == 'median':

for i in columns:

print("Filling missing values with median for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].median())

elif how == 'ffill':

for i in columns:

print("Filling missing values with forward fill for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(method ='ffill')

elif type(how) == int or type(how) == float:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(how)

else:

print("Missing value fill cannot be completed")

return df

def treat\_missing\_categorical(df,columns,how = 'mode'):

'''

Function to treat missing values in numeric columns

Required Input -

- df = Pandas DataFrame

- columns = List input of all the columns need to be imputed

- how = valid values are 'mode', any string or numeric value

Expected Output -

- Pandas dataframe with imputed missing value in mentioned columns

'''

if how == 'mode':

for i in columns:

print("Filling missing values with mode for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mode()[0])

elif type(how) == str:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(how)

elif type(how) == int or type(how) == float:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(str(how))

else:

print("Missing value fill cannot be completed")

return df

# treat\_missing\_numeric(df,non\_cat,how = 'mean')

treat\_missing\_numeric(df,non\_cat,how = 'median')

treat\_missing\_categorical(df,cat,how = 'mode')

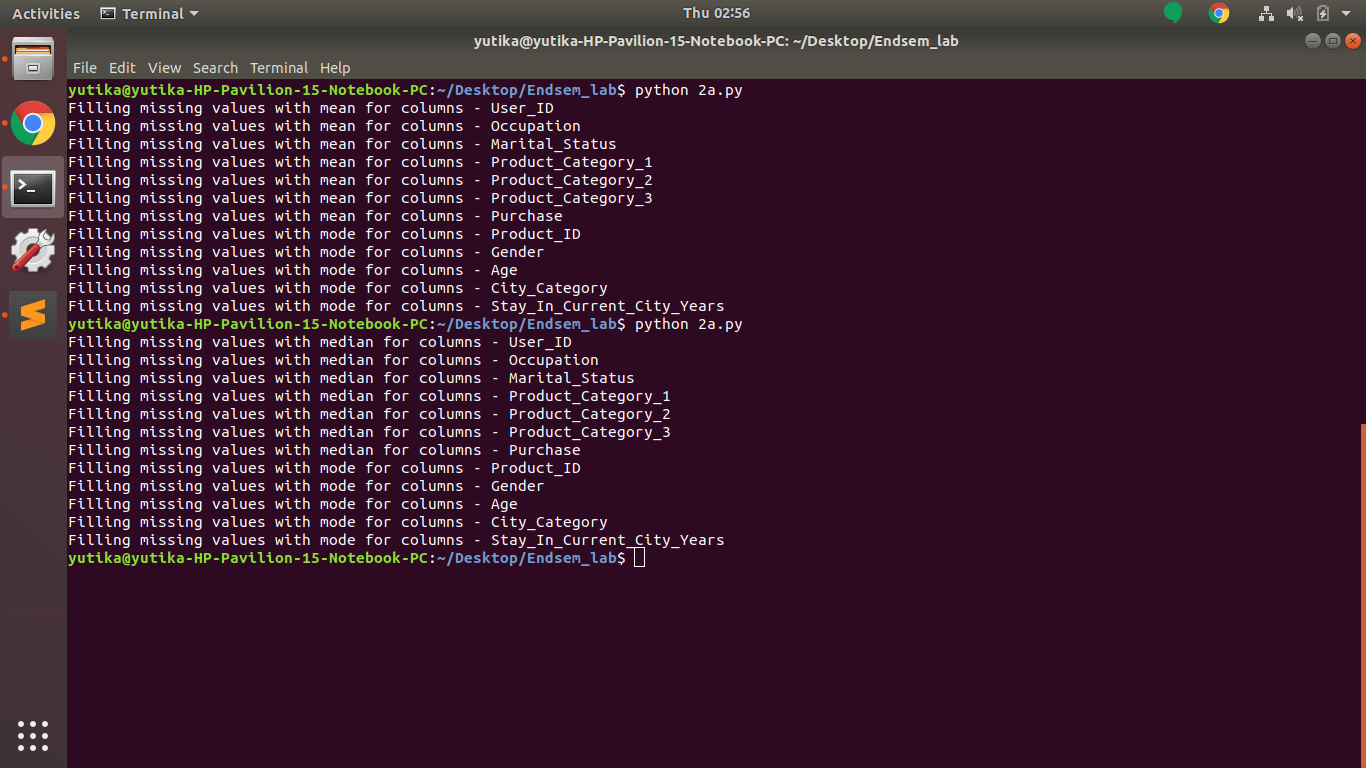
X = df.copy()

def holdout\_cv(X,y,size = 0.3, seed = 1):

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = size, random\_state = seed)

return X\_train, X\_test, y\_train, y\_test

#X\_train, X\_test, y\_train, y\_test = holdout\_cv(X, y, size = 0.3, seed = 1)

* Output:

3. Fit a regression model to predict the credit limit based on relevant parameters.

* How to run the program:
  + python 3c.py
* Input: Given BlackFriday Dataset. Columns to be considered are Purchase on X-axis and Age-Group on Y-axis.
* Implementation: scikit-learn library is used to implement Linear regression model. The data is split into training and testing dataset. The model is first trained and then the tested by predicting the credit limit using test dataset.
* Code:

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import re

import pickle

import time

from nltk.tokenize import TweetTokenizer

import nltk

from nltk import Text

from nltk.tokenize import regexp\_tokenize

from nltk.tokenize import word\_tokenize

from nltk.tokenize import sent\_tokenize

from nltk.corpus import stopwords

from nltk.stem import WordNetLemmatizer

from nltk.stem import PorterStemmer

from sklearn.feature\_extraction.text import CountVectorizer

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

from sklearn.naive\_bayes import MultinomialNB

from sklearn.multiclass import OneVsRestClassifier

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

from sklearn.metrics import classification\_report

from sklearn.metrics import confusion\_matrix

from sklearn.model\_selection import GridSearchCV

from sklearn.pipeline import make\_pipeline

from sklearn.linear\_model import LinearRegression

from sklearn import datasets, linear\_model

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load the BlackFriday dataset

df=pd.read\_csv('big\_BlackFriday.csv')

df.dropna(inplace=True)

#df.drop(df.columns[0],axis=1,inplace=True)

non\_cat = [f for f in df.columns if df.dtypes[f] != 'object']

cat = [f for f in df.columns if df.dtypes[f] == 'object']

def treat\_missing\_numeric(df,columns,how = 'mean'):

'''

Function to treat missing values in numeric columns

Required Input -

- df = Pandas DataFrame

- columns = List input of all the columns need to be imputed

- how = valid values are 'mean', 'mode', 'median','ffill', numeric value

Expected Output -

- Pandas dataframe with imputed missing value in mentioned columns

'''

if how == 'mean':

for i in columns:

print("Filling missing values with mean for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mean())

elif how == 'mode':

for i in columns:

print("Filling missing values with mode for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mode())

elif how == 'median':

for i in columns:

print("Filling missing values with median for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].median())

elif how == 'ffill':

for i in columns:

print("Filling missing values with forward fill for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(method ='ffill')

elif type(how) == int or type(how) == float:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(how)

else:

print("Missing value fill cannot be completed")

return df

def treat\_missing\_categorical(df,columns,how = 'mode'):

'''

Function to treat missing values in numeric columns

Required Input -

- df = Pandas DataFrame

- columns = List input of all the columns need to be imputed

- how = valid values are 'mode', any string or numeric value

Expected Output -

- Pandas dataframe with imputed missing value in mentioned columns

'''

if how == 'mode':

for i in columns:

print("Filling missing values with mode for columns - {0}".format(i))

df.ix[:,i] = df.ix[:,i].fillna(df.ix[:,i].mode()[0])

elif type(how) == str:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(how)

elif type(how) == int or type(how) == float:

for i in columns:

print("Filling missing values with {0} for columns - {1}".format(how,i))

df.ix[:,i] = df.ix[:,i].fillna(str(how))

else:

print("Missing value fill cannot be completed")

return df

treat\_missing\_numeric(df,non\_cat,how = 'mean')

treat\_missing\_categorical(df,cat,how = 'mode')

#print(df)

from sklearn.model\_selection import train\_test\_split

X = df['Purchase']

y, z = pd.factorize(df['Age'])

X, y = X.tolist(), y.tolist()

#print(X, y)

x = [[0 for i in range(2)] for j in range(len(X))]

for i in range(len(X)):

x[i][0] = X[i]

x[i][1] = y[i]

# X = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])

#print (x)

def holdout\_cv(x,y,size = 0.3, seed = 1):

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = size, random\_state = seed)

return x\_train, x\_test, y\_train, y\_test

x\_train, x\_test, y\_train, y\_test = holdout\_cv(x, y, size = 0.3, seed = 1)

reg = LinearRegression().fit(x\_train, y\_train)

print "\n Score:", reg.score(x, y)

print "\n Coefficient:", reg.coef\_

print "\n Intercept:", reg.intercept\_

#print "\n Fit:", reg.fit(list(x\_test),list(y\_test))

print "\n Predict:", reg.predict(list(x\_test))

# Plot outputs

plt.scatter(X,y, color='black')

#plt.plot(X,y, color='blue', linewidth=3)

plt.title('LinearRegression model')

labels=['X-axis:Purchase,Y-axis:Age']

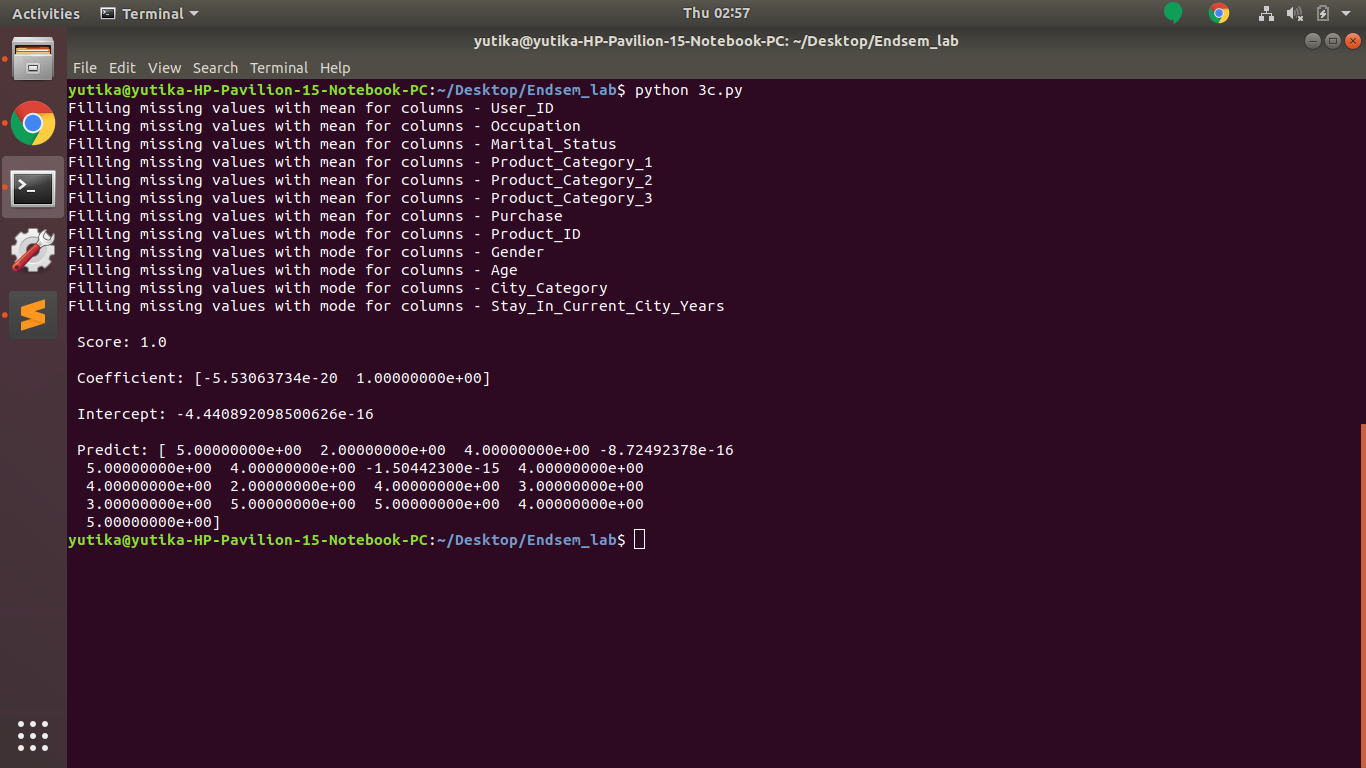
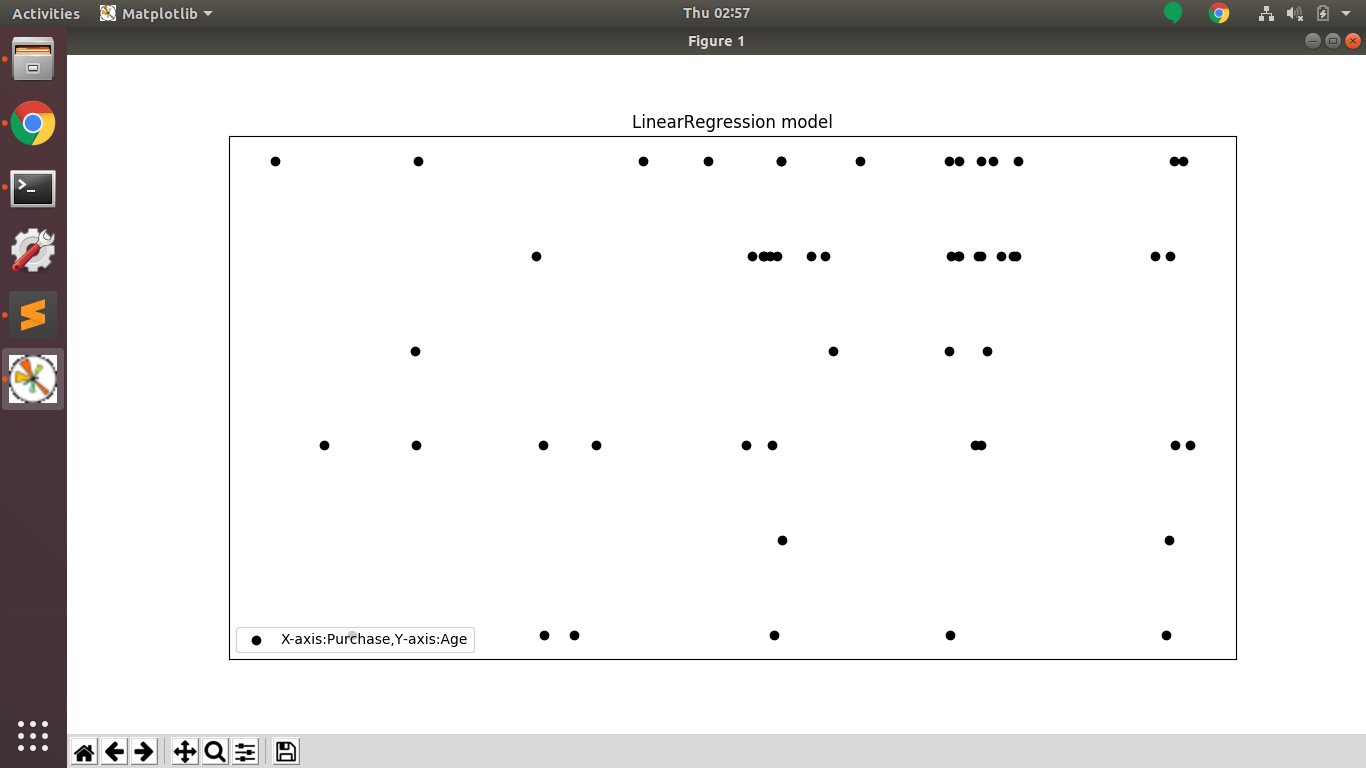
plt.legend(labels,loc=3)

plt.xticks(())

plt.yticks(())

plt.show()

* Output:



4. Establish associations between age, stay in city, marital status with purchase amount using FP growth algorithm.

* How to run the program:
  + python 4a.py
* Input: Age-Group, Stay in city, Marital status with Purchase amount columns of Black friday dataset is used.
* Implementation: pyfpgrowth inbuilt library is used to implement FP Growth Algorithm.
* Code:

import pyfpgrowth

import pandas as pd

store\_data = pd.read\_csv('small\_BlackFriday.csv')

records = []

for i in range(148):

records.append([str(store\_data.values[i,j]) for j in range(3)])

patterns = pyfpgrowth.find\_frequent\_patterns(records, 0.1)

rules = pyfpgrowth.generate\_association\_rules(patterns, 1)

print "=============Patterns============"

print "\n"

print patterns

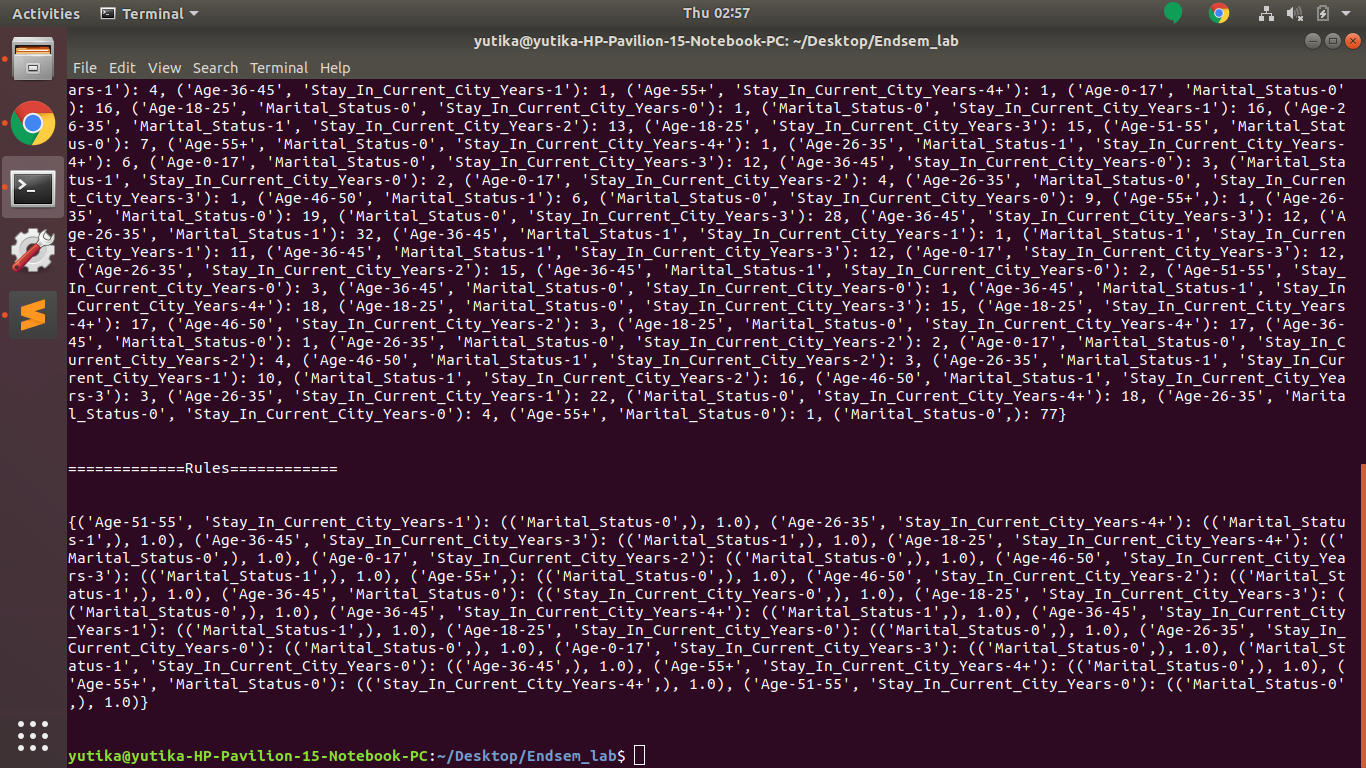
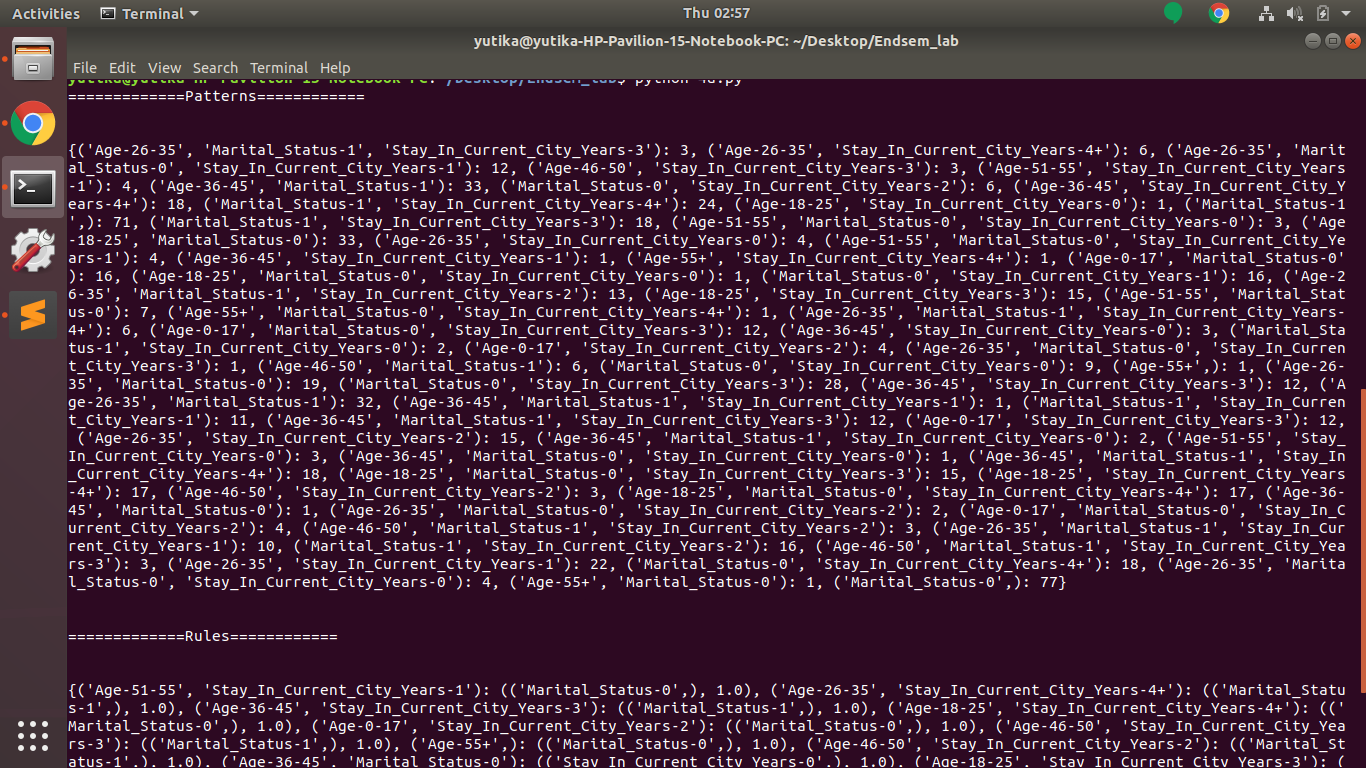
print "\n"

print "=============Rules============"

print "\n"

print rules

print "\n"

* Output:

5. Mine interesting patterns about each product id maintained by the organization.

* How to run the program:
  + python 5a.py
* Input: Given BlackFriday Dataset.
* Implementation:
* Code:

from sklearn import metrics

from collections import Counter

from scipy.spatial.distance import pdist,squareform

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from apyori import apriori

from tqdm import tqdm

store\_data = pd.read\_csv('patterns\_BlackFriday.csv')

records = []

for i in range(199):

records.append([str(store\_data.values[i,j]) for j in range(8)])

association\_rules = apriori(records, min\_support=0.029, min\_lift=2, min\_length=3)

association\_results = list(association\_rules)

for item in association\_results:

pair = item[0]

items = [x for x in pair]

confidence = item[2][0][2]

lift = item[2][0][3]

print("Rule: " + items[0] + " -> " + items[1])

#second index of the inner list

support\_AC = item[1]

support\_A = support\_AC / confidence

support\_C = confidence / lift

leverage = support\_AC - support\_A\*support\_C

#print("Support: " , support\_AC)

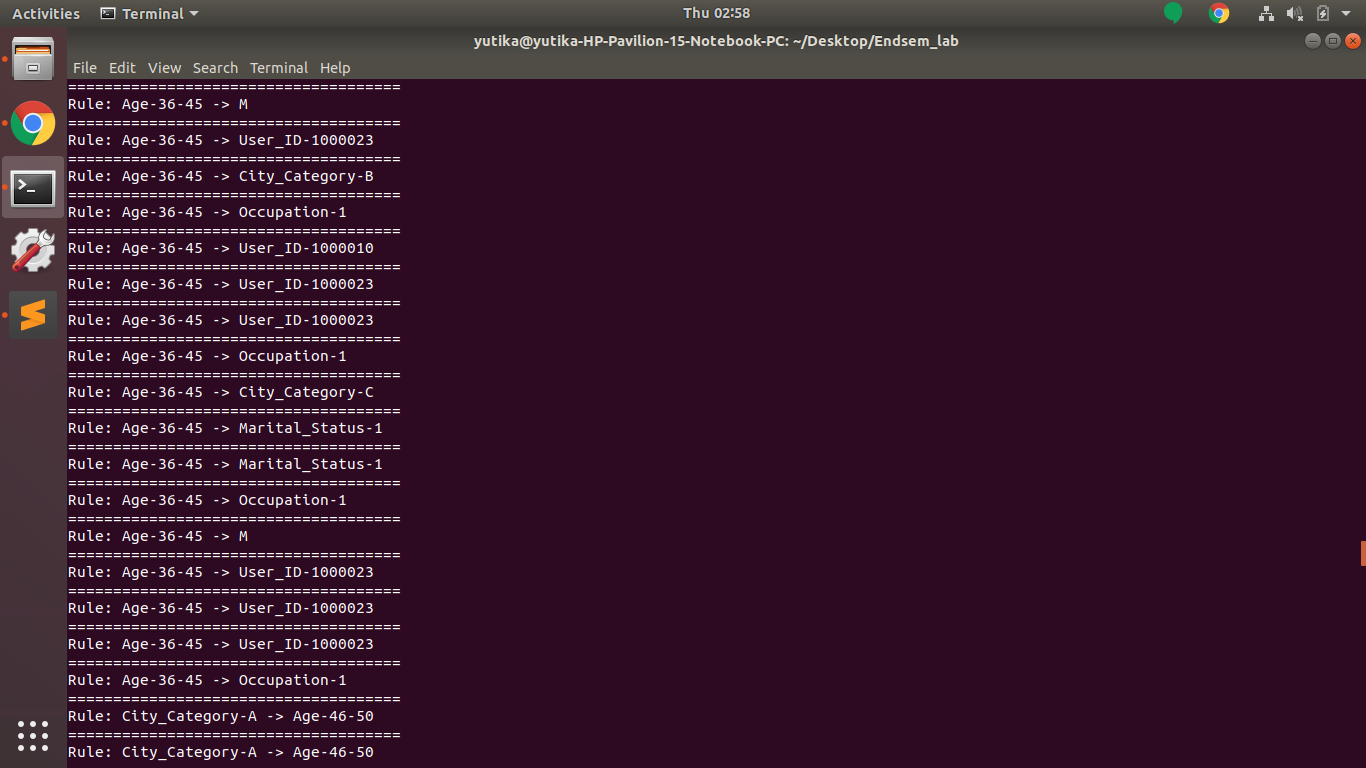
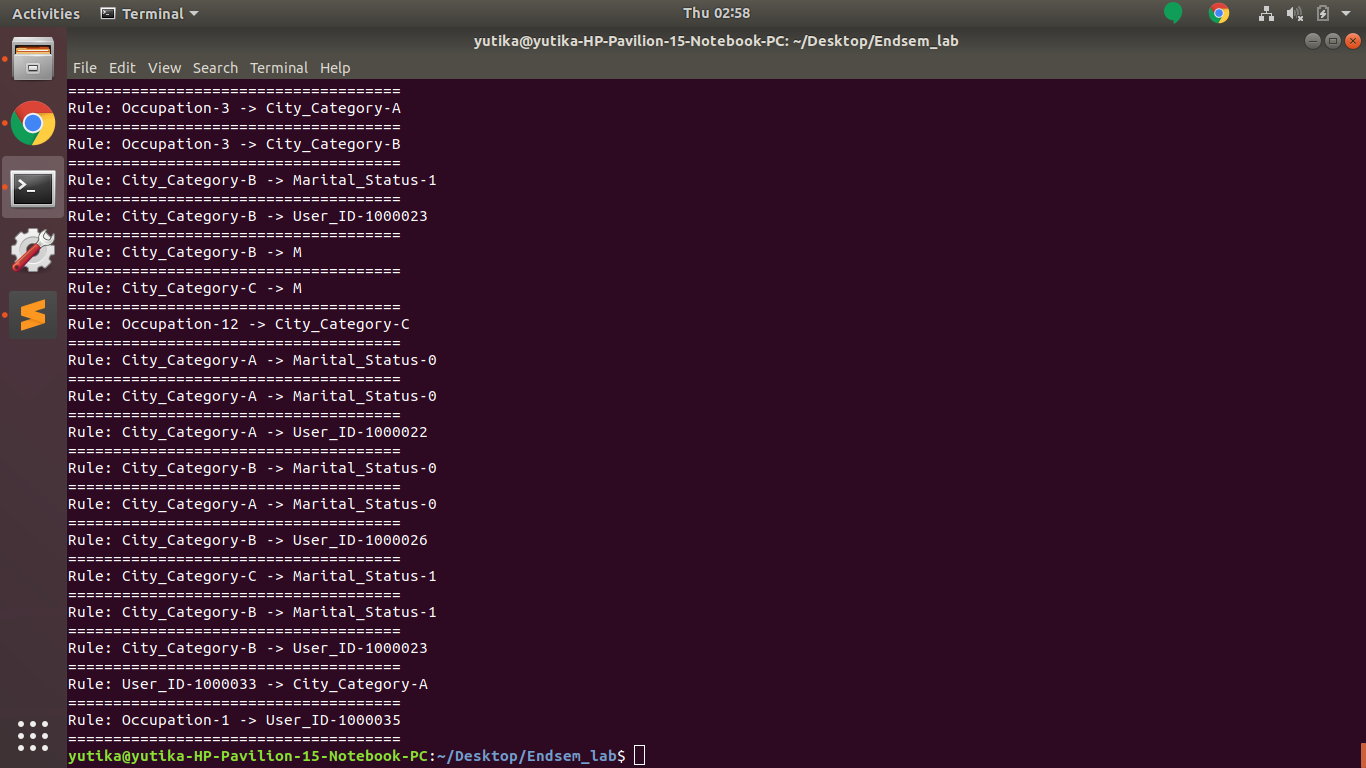
#print("Confidence: " , confidence)

#print("Lift: " ,lift)

#print("Leverage: " , leverage)

print("=====================================")

* Output:



6. Develop a parallelized application using Map Reduce framework to check if a input matrix is symmetric or not( transpose of the matrix is equal to itself).

* How to run the program:
  + python matrix\_transpose.py input.txt
  + python matrix\_transpose.py input1.txt
* Input:
  + Input1.txt:

0,0,1

0,1,1

0,2,0

1,0,0

1,1,1

1,2,0

2,0,0

2,1,0

2,2,1

* + Input.txt:

0,0,1

0,1,0

0,2,0

1,0,0

1,1,1

1,2,0

2,0,0

2,1,0

2,2,1

* Implementation: input1.txt gives output as False and input.txt gives output as True. input.txt has matrix which is symmetric whereas input1.txt has asymmetric matrix. Transpose is done to check whether the original matrix is same as the matrix transpose. Implemented on Spark platform.
* Code:

from \_\_future\_\_ import print\_function

import sys

from operator import add

from pyspark.sql import SparkSession

from numpy import genfromtxt

if \_\_name\_\_ == "\_\_main\_\_":

if len(sys.argv) != 2:

print("Usage: matrix\_symmetric <input file>", file=sys.stderr)

sys.exit(-1)

spark = SparkSession\

.builder\

.appName("Python\_matrix\_symmetric")\

.getOrCreate()

def f(x):

if x[1]>x[0]:

x[0],x[1] = x[1],x[0]

return tuple(x)

def y(a,b):

if a[0]==b[0]:

a[1]+=1

b[1]+=1

return(a,b)

def z(a):

if a[1]!=2:

return(1)

else:

return(0)

lines = spark.read.text(sys.argv[1]).rdd\

.map(lambda r: r[0].split(','))\

.map(lambda x: [int(y) for y in x])\

.filter(lambda x: x[0]!=x[1])\

.map(f)\

.map(lambda x: (x,1))\

.reduceByKey(add)

count = lines.map(z).reduce(add)

print(not count)

spark.stop()

* Output:

