INTRODUCTION TO DATA MINING

(CS 432)

<PROJECT 1>

Group Members:

Syed Mohib Haider Kazmi

Dyass Khalid Warraich

Moiz Sohail

Course Instructor: Mian Muhammad Awais

Date: (21/04/2019)

Number of Hours Spent: 150 Hours

Appendix:

Column Name	Question	Values
AA3	Zone	1 = NORTH 2 = EAST 3 = WEST 4 = SOUTH
AA5	Town Class	1 = TOWN CLASS 1 2 = TOWN CLASS 2 3 = TOWN CLASS 3 4 = TOWN CLASS 4 5 = TOWN CLASS 5
DG3	What is your Marital Status?	1 = Singe/ Not married 2 = Polygamously married (i.e., has multiple spouses) 3 = Monogamously married 4 = Divorced 5 = Separated 6 = Widow/widower 7 = Living together with my partner but not married 8 = Living together/cohabitating 96 = Other (Specify) 99 = DK
DG5_3	Do you have any of the following type of official identification? Passport	1 = Yes 2 = No
DG5_4	Do you have any of the following type of official identification? Driver's license	1 = Yes 2 = No
DG5_5	Do you have any of the following type of official identification? School-issued ID / College-issued ID	1=Yes 2=No
DG5_8	Do you have any of the following type of official identification? Employee ID (for government/civil servants)	1=Yes 2=No
DG5_9	Do you have any of the following type of official identification? Military ID	1=Yes 2=No
DG5_96	Do you have any of the following type of official identification? Other (Specify)	1=Yes 2=No
DG5_10	Do you have any of the following type of official identification? Bank passbook	1=Yes 2=No
DL5	You have said that these are the ways you got money in the past 12 months. Which of these brought you the most money?	1=Government student scholarship (either directly or through schools) 2=Government pension 3=Other government benefits, such as NREGA, MGNREGA, MahamayaYojana, etc 4=Pension that you receive

		from ex-employer or scheme 5=Money from family/friends / spouse for regular support or emergencies 6=Grow a crop and sell it 7=Rear livestock/poultry/fish/bees and sell it or sell by-products of it 8=Buy/get agricultural products from farmers and process it/change it to another form (e.g., maize to flour) 9=Buy/get agricultural products from farmers/processors and sell it 10=Sell something to farmers for the purpose of farming (e.g., seeds, fertilizers, equipment) 11=Sell something to processors of farming products for the purpose of processing (packaging, machinery, chemicals) 12=Make/manufacture something that is used for farming purposes or processing of farming products 13=Provide a service to farmers or processors of farming products (e.g., renting ploughs, tractors, other equipment) 14=Rent land to farmers for farming purposes 15=Fish farming /Fishing – aquaculture, fishermen 16=Employed on other people's farm 17=Employed to do other people's domestic chores 18=Employed by the government 19=Employed in private sector – office/business/factory 20=Running your own business 21=Subletting of house/rooms 22=Earning money from investments (e.g., shares, stocks)

		23=Aid agency/NGO/government assistance in form of food or grants 96=Other (Specify) 99=DK
DL17	Does the household possess a stove/gas burner?	1=No 2=Yes

Part 1: Data Cleaning and Preprocessing:

Several libraries have been used to convert our data into a better and efficient version to extract meaningful and relevant patterns and information which is prime task of data mining. Below are the libraries used to achieve refined data and help in deducing patterns, scaling and graphing models etc., in further parts:

```
#imports
import numpy as np
import pandas as pd
import math

from sklearn.model_selection import train_test_split
from sklearn import linear_model

from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.metrics import mean_squared_error

from sklearn import linear_model
from sklearn import tree  #sklearn libray for decision trees
```

Figure 1.1

Figure 1.1 shows all the libraries imported for this project. Individual reasoning for use of libraries will follow now:

1. **import *numpy as np:*** Numpy helps in doing mathematical and scientific operations and is used extensively to work with multi-dimensional arrays and matrices. Numpy has not been used directly in the project but aids use of further libraries.
2. ***import pandas as pd:*** Pandas are used with Numpy library for fast numeric array computations. Its main functionality in our project is to allow the use of DataFrame attribute values to represent DataFrame df as a Numpy array.
3. ***From sklearn.model_selection import train_test_split:*** For training a model, we need a dataset for training as well as testing. The dataset used to train is supposed to be different from the one used to test so this library helps splitting the dataset into 2 datasets, one for training and one for testing. It also gives options to set parameters to split into specific size of test dataset and train dataset or random state.
4. ***from sklearn.linear_model import LinearRegression:*** Linear regression is used to fit model and predict outcomes based on training dataset. It gives the options of creating different graphs and plots to aid visualization of data and predictions.
5. ***from sklearn.linear_model import Ridge*** AND ***from sklearn.linear_model import Lasso:*** Ridge and Lasso regressions are supervised learning through linear regression simple techniques to reduce complexity and prevent over-fitting of the model which may result from simple linear regressions. Ridge and Lasso have different cost functions so Lasso regressions not only reduce over-fitting but also help in feature selection while Ridge

regressions shrink the coefficients which helps reduce the model complexity and multi-collinearity.

6. ***from sklearn.metrics import mean_squared_error:*** This helps to calculate mean squared error of yTest dataset easily without having to do heavy computations. This, in turn, helps in calculation of Root Mean Squared Errors using ***maths*** library.
7. ***from sklearn import linear_model*** AND ***from sklearn import tree:*** The following two have been explained in the context of usage in relevant

Input Datasets:

```
df = pd.read_csv("Project.csv",low_memory = False,index_col=0)
```

Figure 1.2

```
df2 = pd.read_excel("Dictionary.xlsx")
```

Figure 1.3

The above figures (figure 1.2 and figure 1.3) were the commands used to load both the dataset and dictionary files into DataFrames, respectively. Panda library was used to load the datasets in two variables df and df2.
A new function was created to get columns of the excel sheet, which is also the dictionary for the project, as defined below:

```
df2['Column Name']
def get_cols_sheet(temp_df,col_name):
    """
        Function to get columns of excel sheet file:Dictionary for our project
    """
    name = []
    for values in temp_df[col_name]:
        name.append(values)
    return name
```

Figure 1.4

The function takes two arguments namely, a temporary dataframe with values, which are appended in a list created later in the function, and a column name. Using col_name (column name), a specific column of dataset is accessed to get values of it which are then appended into a list called names, which is also returned.

The function is called in the manner shown below:

```
names_dict = get_cols_sheet(df2,'Column Name')
```

Figure 1.5

By inputting the ‘Column Name’, columns which are needed are stored in names_dict.

Another function was created which takes a temporary dataframe (temp_df) to read column names of the project file. For all the elements in the temporary dataframe colums are read and appended in a list named “names” and returns the list:

```
def get_column_names(temp_df):
    """
        function to read column names of project file

    """
    names = []
    for ele in temp_df.columns:
        names.append(ele)
    return names
```

Figure 1.6

Now these commands were used to find the number of columns in Project dataset and number of columns in Dictionary dataset. Figure 1.7: project dataset was given as an argument to find the number of columns which turned out to be 1234. Similarly, Figure 1.8 was used to get the number of columns in dictionary dataset, which turned out to be 1105. This means there is a need to remove extra data.

```
In [11]: names = get_column_names(df)

In [12]: print(len(names)) #columns in project are 1235
1234
```

Figure 1.7

```
In [13]: print(len(names_dict))
1105
```

Figure 1.8

To complete the task of data removal, another function was created for the purpose of deleting columns (Figure 1.10). The function, deleteCols, was passed three arguments that are a temporary dataframe, a list containing names of columns that are required, and a list containing all column names. The basic functionality of this function is to search for values in the argument, total – list containing all columns, if the values are not found in list_second, excel dataset columns, and if values are also not found in required, list containing names of columns that are required, and if values do not equal ‘train_id’, the value is deleted. The updated dataframe is then returned.

```
def deleteCols(temp_df,required,total):
    """
        Function to remove extra columns whose values we dont have
        temp_df is our read data frame from project.csv
        required is our read 1st column from dict.csv
        total is our read 1st row from project.csv
    """
    list_second = ['AA3','AA4','AA5','AA6','AA7','AA8','AA14','AA15']# name of cols in second xlsx sheet
    i = 0
    for values in total:
        if values not in list_second:
            if values not in required:
                if values !='train_id':
                    del temp_df[values]
    return temp_df
```

Figure 1.9

```
df = deleteCols(df,names_dict,names)
```

Figure 1.10

The following code is used for further data cleaning. Here we fill the missing values with mean if the data is in numerical form. It is also used to replace values of columns having NA with means of the column.

```
df.fillna(round(df.mean()),inplace = True)
```

Figure 1.11

For removing data which is redundant in a sense that it would not help in our mining task, columns with mean equal to 1 were deleted. This, again, serves the purpose of data preprocessing.

```
for values in df.columns[df.isnull().mean() == 1]:
    del df[values]
```

Figure 1.12

Furthermore, since, the missing columns values can be replaced with mode of the column, I.e., the most occurring digit, we decided to make two different codes for finding our results based on both methods separately, one with mean and one with mode.

```
for column in df.columns:
    df[column].fillna(df[column].mode()[0], inplace=True)
```

Figure 1.13

In continuation, another function named updown was created which takes a temporary dataframe as input and serves the purpose of rounding off the values. If the values in column have unit decimal place greater or equal to 5, the value if ceiled to the next number, else it is floored to previous number. (For example, a value of 2.49 will be rounded off to 2 while value of 2.5 shall be rounded off to 3.)

```
def updown(temp_df):
    """
        Criteria for scaling the value up or down:
            If the value of the column is greater than or equal to x.5 then we will ciel the values
            Otherwise we will floor the values
    """
    i = 0
    for columns in temp_df:
        if i == 1:
            pass
        else:
            for values in df[columns]:
                if type(values) == float:
                    values = round(values)

updown(df)
```

Figure 1.14

Moving forward, Null entries or 'unnamed entries' (in context of this project) were removed. For that built-in function was used which searches columns with value == 'unnamed' and drops them.

```
df.drop(df.columns[df.columns.str.contains('unnamed',case = False)],axis = 1)
```

Figure 1.15

By Mean:

train_id	AA3	AA4	AA5	AA6	AA7	AA14	AA15	DG1	Gender	DG3	...	LN1B	LN2_1	LN2_2	LN2_3	LN2_4	GN1	GN2	GN3	GN4	GN5
0	3	32	3.0	7.0	323011	3854	481	1975	1	3	...	1	1	1	1	1	99.0	99	99	99	99
1	2	26	3.0	8.0	268131	2441	344	1981	1	8	...	3	1	1	3	4	6.0	1	2	2	2
2	1	16	3.0	7.0	167581	754	143	1995	1	3	...	4	1	1	2	2	1.0	2	2	2	2
3	4	44	5.0	7.0	445071	5705	604	1980	1	3	...	2	1	1	4	5	6.0	2	2	99	99
4	4	43	3.0	6.0	436161	5645	592	1958	1	3	...	1	2	4	4	4	6.0	1	1	1	1
5	3	35	3.0	8.0	358081	3319	409	1976	0	3	...	3	1	1	5	5	4.0	2	1	1	1
6	3	35	3.0	7.0	357091	3247	401	1998	0	1	...	3	1	1	1	1	4.0	4	4	4	4
7	4	41	4.0	7.0	414021	4459	535	1991	0	3	...	2	1	1	4	4	3.0	3	3	3	3
8	1	16	2.0	7.0	162011	890	175	1958	0	5	...	1	4	3	1	1	1.0	1	2	2	1
9	3	34	3.0	7.0	347242	3959	398	1984	1	5	...	2	1	1	4	4	3.0	3	3	3	3
10	2	23	3.0	7.0	237111	1544	240	1954	0	3	...	3	1	1	1	1	1.0	1	1	1	1
11	1	31	3.0	6.0	316081	597	118	1970	0	3	...	1	2	1	5	5	1.0	3	3	3	3
12	3	32	3.0	7.0	323011	3854	481	1945	1	3	...	4	1	1	1	1	6.0	99	99	99	99
13	2	26	5.0	7.0	265011	2154	327	1956	0	6	...	3	3	3	4	2	1.0	1	1	1	1
14	1	31	3.0	8.0	318211	488	103	1952	0	3	...	2	1	1	4	3	1.0	1	1	1	1
15	1	31	3.0	7.0	317021	485	102	1968	0	3	...	4	2	2	4	4	3.0	3	3	3	3
16	3	33	3.0	8.0	338041	3444	427	1983	0	3	...	4	1	1	3	3	1.0	1	1	1	1
17	3	33	3.0	7.0	333021	3509	435	1980	1	3	...	2	1	1	1	1	2.0	2	2	2	2
18	3	34	3.0	6.0	346021	3994	501	1961	0	3	...	3	1	1	3	3	1.0	1	1	1	1
19	4	44	2.0	7.0	442011	5841	623	1988	1	3	...	1	4	4	5	5	2.0	1	3	2	3
20	3	35	3.0	8.0	358121	3294	406	1981	0	3	...	2	1	1	1	1	2.0	2	99	99	99
21	1	11	1.0	7.0	111011	99999	93	1981	1	3	...	1	4	4	5	4	6.0	2	2	2	2
22	1	16	3.0	7.0	167381	921	180	1996	1	3	...	3	1	1	1	1	1.0	1	2	2	2
23	4	41	2.0	7.0	412011	5010	547	1971	1	3	...	3	1	1	2	2	1.0	1	1	1	1
24	2	21	3.0	6.0	216091	2017	302	1996	0	1	...	1	3	3	4	4	6.0	4	4	4	4
25	1	16	4.0	7.0	164031	714	134	2000	0	1	...	1	4	4	4	4	4.0	4	4	4	4
26	1	16	5.0	7.0	165121	887	174	1994	0	3	...	1	4	4	1	1	1.0	1	1	1	1
27	1	16	3.0	6.0	166051	875	171	1953	1	3	...	2	2	2	2	2	1.0	2	2	2	3
28	4	42	3.0	6.0	426061	5466	560	1980	1	3	...	4	1	1	4	1	3.0	3	3	3	3
29	3	35	3.0	7.0	357051	3345	412	1955	1	3	...	2	1	1	1	1	6.0	99	99	99	99
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...

18225	4	44	3.0	6.0	446101	5754	611	1967	1	3	...	4	1	1	5	5	3.0	3	3	3	3
18226	4	44	3.0	6.0	446011	5837	623	1991	1	3	...	1	3	3	5	5	2.0	2	2	2	2
18227	4	43	3.0	6.0	436221	5678	598	1964	0	3	...	1	3	4	4	3	1.0	1	1	2	1
18228	1	31	3.0	7.0	317181	517	106	1988	1	6	...	3	2	2	1	1	3.0	3	3	3	3
18229	3	33	3.0	8.0	338291	3564	442	1970	0	3	...	4	1	1	1	1	1.0	1	1	1	1
18230	4	44	3.0	7.0	447091	5764	612	1998	0	1	...	1	4	4	4	4	4.0	4	4	4	4
18231	1	16	3.0	7.0	163061	738	140	1961	0	3	...	2	1	1	4	4	1.0	1	1	1	1
18232	1	16	3.0	7.0	163011	755	143	1986	0	3	...	2	2	2	4	4	2.0	2	2	2	2
18233	1	16	3.0	8.0	168342	888	175	1976	1	3	...	2	4	4	5	4	3.0	3	3	3	3
18234	1	16	3.0	6.0	166241	736	139	1978	0	3	...	2	2	2	5	2	1.0	1	1	1	1
18235	3	32	3.0	7.0	327091	3734	469	1981	1	3	...	3	1	1	1	5	2.0	3	3	3	3
18236	2	26	3.0	7.0	267051	2201	331	1988	1	8	...	3	2	2	3	4	3.0	3	3	3	3
18237	2	26	3.0	6.0	266161	2197	330	1951	1	3	...	4	1	1	1	3	6.0	3	3	3	3
18238	3	32	3.0	8.0	328051	3858	481	1995	0	3	...	1	3	3	4	5	1.0	1	1	1	1
18239	3	32	5.0	7.0	325011	3782	474	1997	0	1	...	1	2	3	4	4	1.0	4	4	1	1
18240	2	26	3.0	6.0	266041	2302	336	1976	1	8	...	2	1	1	4	4	6.0	2	2	2	2
18241	4	43	3.0	7.0	435021	5674	598	1982	0	3	...	1	4	4	4	5	1.0	1	1	1	1
18242	1	16	3.0	7.0	163041	743	141	1952	0	3	...	4	1	1	2	2	1.0	1	1	1	1
18243	3	32	3.0	6.0	326081	3897	486	1999	1	1	...	4	2	2	4	3	4.0	4	4	4	4
18244	1	31	2.0	7.0	312011	667	127	1976	1	3	...	3	1	1	1	1	1.0	1	99	99	99
18245	3	32	3.0	7.0	327101	3904	486	1989	0	3	...	1	1	1	5	3	1.0	3	4	4	4
18246	2	26	3.0	6.0	266221	2191	330	1976	1	3	...	2	3	3	4	4	6.0	3	3	3	3
18247	1	16	3.0	8.0	168111	992	196	1931	0	3	...	2	4	5	5	4	1.0	1	1	1	1
18248	2	26	3.0	8.0	268091	2313	336	1966	1	8	...	2	1	1	3	3	2.0	2	2	2	2
18249	2	25	3.0	8.0	258101	3052	386	1986	1	3	...	1	4	4	5	4	3.0	3	3	3	3
18250	1	16	3.0	7.0	167221	880	173	1965	0	3	...	4	1	1	1	1	2.0	2	2	2	2
18251	3	34	3.0	7.0	343041	4100	511	1992	0	1	...	1	1	5	5	4	99.0	99	99	99	99
18252	3	34	3.0	6.0	346191	4287	530	1980	0	3	...	2	3	3	3	2	1.0	1	1	1	1
18253	1	14	3.0	6.0	146041	361	71	1978	0	3	...	1	4	4	5	5	1.0	1	1	1	2
18254	1	16	3.0	6.0	166161	800	153	1984	1	3	...	2	1	1	1	1	1.0	3	3	3	3

18255 rows × 1045 columns

Figure 1.16

Above is the resultant dataset after cleaning and preprocessing. After applying a few techniques, the column number is now reduced to 1045 columns (previously 1235 columns). The values in the columns no more have decimal parts because of usage of ceil and floor methods, similarly data no more contains NULL values and missing values (replaced with mean value of the respective columns) while columns with mean = 1 were deleted. Now the data is prepared to be used for hypothesis testing, model building and data mining.

By Mode:

train_id	AA3	AA4	AA5	AA6	AA7	AA14	AA15	DG1	Gender	DG3	...	LN1B	LN2_1	LN2_2	LN2_3	LN2_4	GN1	GN2	GN3	GN4	GN5
0	3	32	3.0	6.0	323011	3854	481	1975	1	3	...	1	1	1	1	1	99.0	99	99	99	99
1	2	26	3.0	8.0	268131	2441	344	1981	1	8	...	3	1	1	3	4	1.0	1	2	2	2
2	1	16	3.0	7.0	167581	754	143	1995	1	3	...	4	1	1	2	2	1.0	2	2	2	2
3	4	44	5.0	6.0	445071	5705	604	1980	1	3	...	2	1	1	4	5	1.0	2	2	99	99
4	4	43	3.0	6.0	436161	5645	592	1958	1	3	...	1	2	4	4	4	1.0	1	1	1	1
5	3	35	3.0	8.0	358081	3319	409	1976	0	3	...	3	1	1	5	5	4.0	2	1	1	1
6	3	35	3.0	7.0	357091	3247	401	1998	0	1	...	3	1	1	1	1	4.0	4	4	4	4
7	4	41	4.0	6.0	414021	4459	535	1991	0	3	...	2	1	1	4	4	3.0	3	3	3	3
8	1	16	2.0	6.0	162011	890	175	1958	0	5	...	1	4	3	1	1	1.0	1	2	2	1
9	3	34	3.0	7.0	347242	3959	398	1984	1	5	...	2	1	1	4	4	3.0	3	3	3	3
10	2	23	3.0	7.0	237111	1544	240	1954	0	3	...	3	1	1	1	1	1.0	1	1	1	1
11	1	31	3.0	6.0	316081	597	118	1970	0	3	...	1	2	1	5	5	1.0	3	3	3	3
12	3	32	3.0	6.0	323011	3854	481	1945	1	3	...	4	1	1	1	1	1.0	99	99	99	99
13	2	26	5.0	6.0	265011	2154	327	1956	0	6	...	3	3	3	4	2	1.0	1	1	1	1
14	1	31	3.0	8.0	318211	488	103	1952	0	3	...	2	1	1	4	3	1.0	1	1	1	1
15	1	31	3.0	7.0	317021	485	102	1968	0	3	...	4	2	2	4	4	3.0	3	3	3	3
16	3	33	3.0	8.0	338041	3444	427	1983	0	3	...	4	1	1	3	3	1.0	1	1	1	1
17	3	33	3.0	6.0	333021	3509	435	1980	1	3	...	2	1	1	1	1	2.0	2	2	2	2
18	3	34	3.0	6.0	346021	3994	501	1961	0	3	...	3	1	1	3	3	1.0	1	1	1	1
19	4	44	2.0	6.0	442011	5841	623	1988	1	3	...	1	4	4	5	5	2.0	1	3	2	3
20	3	35	3.0	8.0	358121	3294	406	1981	0	3	...	2	1	1	1	1	2.0	2	99	99	99
21	1	11	1.0	6.0	111011	99999	93	1981	1	3	...	1	4	4	5	4	1.0	2	2	2	2
22	1	16	3.0	7.0	167381	921	180	1996	1	3	...	3	1	1	1	1	1.0	1	2	2	2
23	4	41	2.0	6.0	412011	5010	547	1971	1	3	...	3	1	1	2	2	1.0	1	1	1	1
24	2	21	3.0	6.0	216091	2017	302	1996	0	1	...	1	3	3	4	4	1.0	4	4	4	4
25	1	16	4.0	6.0	164031	714	134	2000	0	1	...	1	4	4	4	4	4.0	4	4	4	4
26	1	16	5.0	6.0	165121	887	174	1994	0	3	...	1	4	4	1	1	1.0	1	1	1	1
27	1	16	3.0	6.0	166051	875	171	1953	1	3	...	2	2	2	2	2	1.0	2	2	2	3
28	4	42	3.0	6.0	426061	5466	560	1980	1	3	...	4	1	1	4	1	3.0	3	3	3	3
29	3	35	3.0	7.0	357051	3345	412	1955	1	3	...	2	1	1	1	1	1.0	99	99	99	99
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
18225	4	44	3.0	6.0	446101	5754	611	1967	1	3	...	4	1	1	5	5	3.0	3	3	3	3
18226	4	44	3.0	6.0	446011	5837	623	1991	1	3	...	1	3	3	5	5	2.0	2	2	2	2
18227	4	43	3.0	6.0	436221	5678	598	1964	0	3	...	1	3	4	4	3	1.0	1	1	2	1
18228	1	31	3.0	7.0	317181	517	106	1988	1	6	...	3	2	2	1	1	3.0	3	3	3	3
18229	3	33	3.0	8.0	338291	3564	442	1970	0	3	...	4	1	1	1	1	1.0	1	1	1	1
18230	4	44	3.0	7.0	447091	5764	612	1998	0	1	...	1	4	4	4	4	4.0	4	4	4	4
18231	1	16	3.0	6.0	163061	738	140	1961	0	3	...	2	1	1	4	4	1.0	1	1	1	1
18232	1	16	3.0	6.0	163011	755	143	1986	0	3	...	2	2	2	4	4	2.0	2	2	2	2
18233	1	16	3.0	8.0	168342	888	175	1976	1	3	...	2	4	4	5	4	3.0	3	3	3	3
18234	1	16	3.0	6.0	166241	736	139	1978	0	3	...	2	2	2	5	2	1.0	1	1	1	1
18235	3	32	3.0	7.0	327091	3734	469	1981	1	3	...	3	1	1	1	5	2.0	3	3	3	3
18236	2	26	3.0	7.0	267051	2201	331	1988	1	8	...	3	2	2	3	4	3.0	3	3	3	3
18237	2	26	3.0	6.0	266161	2197	330	1951	1	3	...	4	1	1	1	3	1.0	3	3	3	3
18238	3	32	3.0	8.0	328051	3858	481	1995	0	3	...	1	3	3	4	5	1.0	1	1	1	1
18239	3	32	5.0	6.0	325011	3782	474	1997	0	1	...	1	2	3	4	4	1.0	4	4	1	1
18240	2	26	3.0	6.0	266041	2302	336	1976	1	8	...	2	1	1	4	4	1.0	2	2	2	2
18241	4	43	3.0	6.0	435021	5674	598	1982	0	3	...	1	4	4	4	5	1.0	1	1	1	1
18242	1	16	3.0	6.0	163041	743	141	1952	0	3	...	4	1	1	2	2	1.0	1	1	1	1
18243	3	32	3.0	6.0	326081	3897	486	1999	1	1	...	4	2	2	4	3	4.0	4	4	4	4
18244	1	31	2.0	6.0	312011	667	127	1976	1	3	...	3	1	1	1	1	1.0	1	99	99	99
18245	3	32	3.0	7.0	327101	3904	486	1989	0	3	...	1	1	1	5	3	1.0	3	4	4	4
18246	2	26	3.0	6.0	266221	2191	330	1976	1	3	...	2	3	3	4	4	1.0	3	3	3	3
18247	1	16	3.0	8.0	168111	992	196	1931	0	3	...	2	4	5	5	4	1.0	1	1	1	1
18248	2	26	3.0	8.0	268091	2313	336	1966	1	8	...	2	1	1	3	3	2.0	2	2	2	2
18249	2	25	3.0	8.0	258101	3052	386	1986	1	3	...	1	4	4	5	4	3.0	3	3	3	3
18250	1	16	3.0	7.0	167221	880	173	1965	0	3	...	4	1	1	1	1	2.0	2	2	2	2
18251	3	34	3.0	6.0	343041	4100	511	1992	0	1	...	1	1	5	5	4	99.0	99	99	99	99
18252	3	34	3.0	6.0	346191	4287	530	1980	0	3	...	2	3	3	3	2	1.0	1	1	1	1
18253	1	14	3.0	6.0	146041	361	71	1978	0	3	...	1	4	4	5	5	1.0	1	1	1	2
18254	1	16	3.0	6.0	166161	800	153	1984	1	3	...	2	1	1	1	1	1.0	3	3	3	3

Figure 1.17

Similar to what we did to get results for figure 1.16, Figure 1.17 shows the resultant dataset, which was generated using mode instead of mean, which we did previously. Carefully looking at the data, we realize that few values are indeed different from the dataset generated using mean technique, so for further tasks, we will use both methods to potray results.

Part 2: Task# 1:

For this part, we need to train and test our dataset i.e., we train our model with some portion of dataset and then test it against our test dataset to find how accurately our model can predict the results. To start off, processed.csv dataset (contains processed data) must be chosen and converted back to csv format from dataframe format. Next, gender column is separated. If the columns of dataframe contains 'Gender' string, it would be dropped. Similarly, 'train_id' is also dropped from the dataset to help in training. Dataset, now, must be split into training and testing datasets, 20 percent was chosen to be the size of datasets.

```
file_name = "processed.csv"
```

Figure 2.1

```
df.to_csv(file_name,encoding='utf-8')
```

Figure 2.2

```
y = df['Gender']
```

Figure 2.3

```
x = df.drop(df.columns[df.columns.str.contains('Gender',case = False)],axis = 1)
```

Figure 2.4

```
x = df.drop(df.columns[df.columns.str.contains('train_id',case = False)],axis = 1)
```

Figure 2.5

```
xTrain, xTest, yTrain, yTest = train_test_split(x, y, test_size = 0.2, random_state = 2)
```

Figure 2.6

A linear regression is to be run to predict gender of the survey respondents. The model is then fit with xTrain and yTrain datasets previously generated (above figure number) using fit function. Accuracy can be calculated using score function, which was then stored in accuracy named variable. It resulted in an accuracy of 100%. Similarly, predictions of gender were made using xTrain data only (using function predict – it results an array with predicted values). As seen in the figure, different predicted values were generated. These were then rounded off to nearest integer values as the values could be either 0 or 1 (given two genders, male and female).

```
linear_model = LinearRegression()
```

```
linear_model.fit(xTrain,yTrain)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

```
accuracy = linear_model.score(xTrain,yTrain)
```

```
accuracy #overfitting
```

```
1.0
```

```
y_predictions = linear_model.predict(xTrain)
```

```
y_predictions
```

```
array([-1.17241389e-11,  1.00000000e+00,  1.00000000e+00, ...,
        1.00000000e+00,  1.00000000e+00, -9.56128961e-12])
```

```
accuracy = linear_model.score(xTest,yTest)
```

```
accuracy #overfiitng
```

```
1.0
```

Figure 2.7

By Mode:

```
linear_model = LinearRegression()
```

```
yTrain
```

```
array([0, 1, 1, ..., 1, 1, 0], dtype=int64)
```

```
linear_model.fit(xTrain,yTrain,)
```

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None,
         normalize=False)
```

```
linear_model.score(xTrain,yTrain)
```

```
1.0
```

```
linear_model.score(xTest,yTest)
```

```
1.0
```

```
accuracy = linear_model.score(xTest,yTest)
```

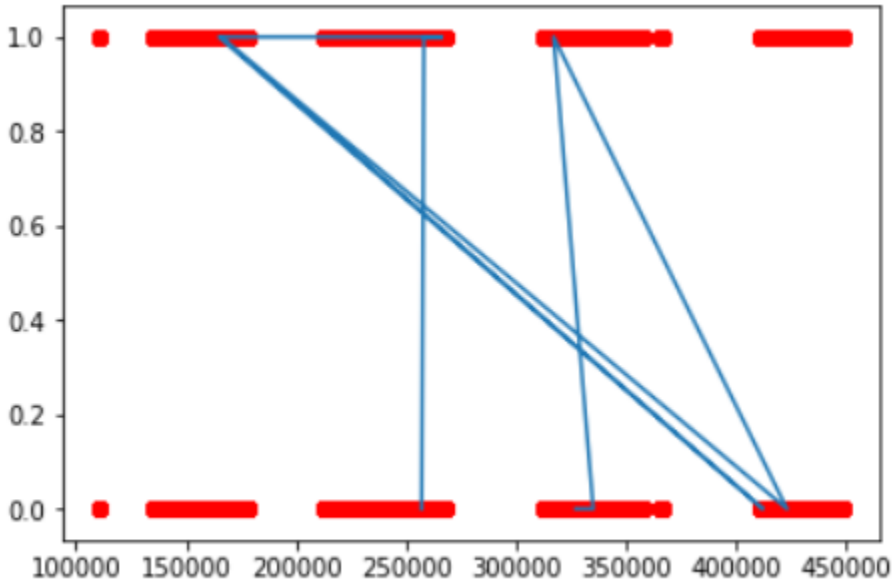
```
accuracy #
```

```
1.0
```

Figure 2.8

For figure 2.8, we use the dataset generated using mode instead of mean. The array generated by this method gives integers while by mean values in the array had to be rounded off as it is possible for mean to have decimal points, however, mode replaces whole values resulting in a rounded array. The results of train and test data were similar, and accuracy was 100 % as well; the exact results obtained from mean method.

Graph:



Graph 1

A scatter plot was generated for the above task. Blue lines show the predicted values of different variables while red patches represent data points for that variable.

Part 2: Task# 2:

To predict the gender of the survey respondent, classifiers were to be made. Using the function, tree.DecisionTreeClassifier, a decision tree class instance (model) is made. The model is then fitted using the xTrain and yTrain datasets generated in previous steps. The model is then trained with xTrain and yTrain datasets and tested against xTest and yTest datasets which were generated previously. The calculated accuracy was 100%.

```
classifier = tree.DecisionTreeClassifier() #Decision tree class instance
```

```
classifier.fit(xTrain,yTrain)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

```
classifier.score(xTrain,yTrain)
```

```
1.0
```

```
classifier.score(xTest,yTest)
```

```
1.0
```

Figure 2.9

By Mode:

```
classifier = tree.DecisionTreeClassifier(criterion='entropy') #Decision tree class instance
```

```
classifier.fit(xTrain,yTrain)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
```

```
classifier.score(xTrain,yTrain)
```

```
1.0
```

```
classifier.score(xTest,yTest)
```

```
1.0
```

Figure 2.10

Using mode technique in this case has no obvious effects as can be observed in the Figure 2.10. The results are similar to those in Figure 2.11 which were generated using mean.

Part 2: Task# 3:

To find interesting patterns in this dataset, association rule mining can be used. Apriori is a technique to identify underlying relations between different items which means this algorithm helps find patterns.

```
def data_generator(filename,dic):
    def data_gen():
        with open(filename) as file:
            names = []
            for x,line in enumerate(file):
                l = line.split(',')
                if x==0:
                    names = l
                else:
                    yield tuple(convertor(i,l,dic,names) for i in range(1,len(l)))

    return data_gen
```

The dataset available is too large to compute quickly so to reduce inefficiency, a data generator was defined which returns a generator instead of a list every time the function is called. This function generates tuples with class names and relevant class values.

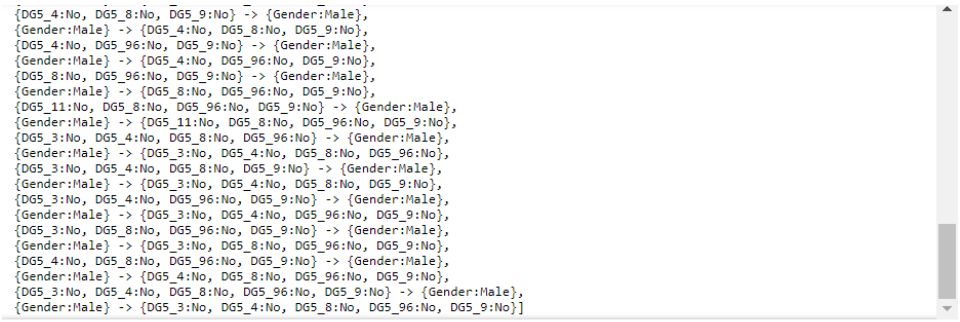
Next, the generated tuples are stored in a variable called transactions which is used by Apriori function to find itemset and rules with minimum support of 0.5 and minimum confidence of 0.3, then the rules are generated whose number turn out to be quite large as can be seen in the figure below:

```
transactions = data_generator(file_name,dictionary)
itemsets, rules = apriori(transactions, min_support = 0.5, min_confidence = 0.3)
rules
```

```
[{DG3:Monogamously married} -> {AA5:Town Class 3},
 {AA5:Town Class 3} -> {DG3:Monogamously married},
 {DG3A:Hinduism} -> {AA5:Town Class 3},
 {AA5:Town Class 3} -> {DG3A:Hinduism},
 {DG5_11:No} -> {AA5:Town Class 3},
 {AA5:Town Class 3} -> {DG5_11:No},
 {DG5_1:Yes} -> {AA5:Town Class 3},
 {AA5:Town Class 3} -> {DG5_1:Yes},
 {DG5_2:No} -> {AA5:Town Class 3},
 {AA5:Town Class 3} -> {DG5_2:No},
 {DG5_3:No} -> {AA5:Town Class 3},
 {AA5:Town Class 3} -> {DG5_3:No},
 {DG5_4:No} -> {AA5:Town Class 3},
 {AA5:Town Class 3} -> {DG5_4:No},
 {DG5_5:No} -> {AA5:Town Class 3},
 {AA5:Town Class 3} -> {DG5_5:No},
 {DG5_6:Yes} -> {AA5:Town Class 3},
 {AA5:Town Class 3} -> {DG5_6:Yes},
 {DG5_7:Yes} -> {AA5:Town Class 3},
 {AA5:Town Class 3} -> {DG5_7:Yes},
```

```
rules = list(filter(lambda x: x.lhs[0] and x.rhs[0] and ("Gender" in x.lhs[0] or "Gender" in x.rhs[0]),rules))
rules
```

```
[{Gender:Male} -> {DG5_11:No},
 {DG5_11:No} -> {Gender:Male},
 {Gender:Male} -> {DG5_3:No},
 {DG5_3:No} -> {Gender:Male},
 {Gender:Male} -> {DG5_4:No},
 {DG5_4:No} -> {Gender:Male},
 {Gender:Male} -> {DG5_8:No},
 {DG5_8:No} -> {Gender:Male},
 {Gender:Male} -> {DG5_96:No},
 {DG5_96:No} -> {Gender:Male},
 {Gender:Male} -> {DG5_9:No},
 {DG5_9:No} -> {Gender:Male},
 {DG5_11:No, DG5_8:No} -> {Gender:Male},
 {Gender:Male} -> {DG5_11:No, DG5_8:No},
 {DG5_11:No, DG5_96:No} -> {Gender:Male},
 {Gender:Male} -> {DG5_11:No, DG5_96:No},
 {DG5_11:No, DG5_9:No} -> {Gender:Male},
 {Gender:Male} -> {DG5_11:No, DG5_9:No},
 {DG5_3:No, DG5_4:No} -> {Gender:Male},
 {Gender:Male} -> {DG5_3:No, DG5_4:No},
```



The above figures show rules filtered by gender Male. The results are very interesting. While Male gender in south east Asian countries are considered to be the prime breadwinner for the family, it can be observed from the rules generated over the dataset that most Male members do not have official Employee IDs. For instance, the following rule:

```
{Gender:Male} -> {DG5_3:No, DG5_4:No, DG5_8:No, DG5_96:No, DG5_9:No}
```

Shows that Gender Male is associated with DG5_3 (Passport identity), DG5_4 (Driver's License), DG5_8 (Employee ID), DG5_96 (Military ID) and DG5_9 (Any Special ID) in a manner that when they all do not exist in case of a particular person, then the gender is considered to be Male. Looking into the definition of these columns names we find out that the person has NO Passport, NO Driver's License, NO Employee ID, NO Military ID and NO special ID document. Various findings can be deduced from these rules: 1) Most of the survey respondents would belong to lower-middle class in terms of income category as they have no passports which are necessary for travelling abroad so we can create a link that since the respondents do not have a passport, there is a high chance that they cannot afford a foreign trip; 2) Most of the survey respondents do not have a Driver's License which means that the respondents do not own a vehicle to drive and use alternate means to commute, again reaffirming the hypothesis that most of the respondents belong to lower-middle class; 3) Most of the respondents did not have an Employee ID which is given to Government and Civil servants which means that individuals with Employee ID (working for Government or are Civil servants) can either be under-represented or would be of Female gender which contrasts starkly from the reality as it was noted that women working in government sector touched 10% only, in 2013 (The Times of India).

Part 2: Task# 4:

To find correlation is this part, function corr is used on dataframe.

```
corr = df.corr()
```

```
corr
```

Figure 2.11

corr now represents the correlation matrix (1045x1045 size) as shown below:

	AA3	AA4	AA5	AA6	AA7	AA14	AA15	DG1	Gender	DG3	...	LN1B	LN2_1	LN2_2	LN2_3	LN2_4	GN1	GN2	GN3	GN4	GN5
AA3	1.000000	0.925447	-0.005902	-0.093804	0.924611	0.117725	0.958523	-0.070281	-0.008179	0.018889	...	-0.079768	0.049245	0.053422	0.060630	0.063183	0.020233	0.048947	0.049809	0.031502	0.031291
AA4	0.925447	1.000000	0.012762	-0.077420	0.999793	0.084883	0.889837	-0.068858	-0.028591	0.031169	...	-0.084329	0.061542	0.063216	0.050255	0.049743	0.013123	0.046940	0.054093	0.043527	0.044589
AA5	-0.005902	0.012762	1.000000	0.012471	0.018902	-0.505351	-0.007074	-0.003111	0.006411	-0.001953	...	0.055592	-0.065424	-0.067240	-0.074488	-0.069536	0.017228	0.022697	0.020120	0.014895	0.015104
AA6	-0.093804	-0.077420	0.012471	1.000000	-0.073705	0.046036	-0.070851	0.016651	-0.012479	0.020498	...	-0.008820	0.011688	0.010624	0.012946	-0.003407	0.012594	0.032619	0.033587	0.039596	0.045022
AA7	0.924611	0.999793	0.018902	-0.073705	1.000000	0.074603	0.889307	-0.069531	-0.028565	0.031779	...	-0.081114	0.057108	0.058756	0.046732	0.046441	0.013191	0.047191	0.054024	0.043489	0.044553
AA14	0.117725	0.084883	-0.505351	0.046036	0.074603	1.000000	0.131857	0.014166	-0.000338	0.004824	...	-0.095498	0.139811	0.142444	0.128753	0.129005	-0.015848	-0.007293	-0.000146	-0.004038	-0.005680
AA15	0.958523	0.889837	-0.007074	-0.070851	0.889307	0.131857	1.000000	-0.067746	-0.000432	0.011747	...	-0.098245	0.050606	0.056100	0.088676	0.101423	0.016100	0.042819	0.041620	0.024165	0.022936
DG1	-0.070281	-0.068858	-0.003111	0.016651	-0.069531	0.014166	-0.067746	1.000000	0.067827	-0.040159	...	-0.203573	0.232229	0.238173	0.186134	0.174095	-0.023267	-0.036997	-0.028059	-0.033726	-0.025619
Gender	-0.008179	-0.028591	0.006411	-0.012479	-0.028565	-0.000338	-0.000432	0.067827	1.000000	-0.008646	...	0.165073	-0.138924	-0.137520	-0.117376	-0.106413	0.047848	0.021997	0.022552	0.019375	0.027117
DG3	0.018889	0.031169	-0.001953	0.020498	0.031779	0.004824	0.011747	-0.040159	-0.008646	1.000000	...	0.048740	-0.043356	-0.042672	-0.068006	-0.057339	0.040091	0.269473	0.246047	0.214297	0.209442
DG3A	0.015543	0.015987	0.019920	0.026478	0.016046	-0.000318	0.015110	0.013312	-0.042679	0.401518	...	0.014612	0.008779	0.005722	-0.028221	-0.031760	0.037241	0.203294	0.186594	0.165179	0.166348
DG4	0.048420	0.048365	-0.010886	0.032032	0.047505	0.033999	0.040403	0.093414	-0.099952	0.473673	...	-0.114338	0.125327	0.125622	0.066324	0.055424	0.039297	0.260545	0.233828	0.202474	0.200310
DG5_1	-0.076690	-0.128986	-0.005729	0.040850	-0.128332	-0.020660	-0.061884	0.057116	0.005085	0.151817	...	0.039034	-0.025765	-0.033673	-0.051594	-0.055826	0.022192	0.094318	0.080326	0.079772	0.078855
DG5_2	-0.017259	0.012977	0.114460	-0.012327	0.017696	-0.189568	-0.049358	0.010985	0.168942	0.007780	...	0.232248	-0.272913	-0.271533	-0.205272	-0.178385	0.032184	0.032251	0.031778	0.026897	0.027852
DG5_3	-0.095358	-0.089541	-0.009101	0.033690	-0.088759	-0.013821	-0.087506	-0.007063	0.092177	0.001713	...	0.095483	-0.053265	-0.048569	0.018519	0.018164	0.005182	0.007117	0.011378	0.013032	0.009998
DG5_4	-0.090789	-0.080796	0.034008	0.029384	-0.078206	-0.056312	-0.086374	-0.006521	0.267392	0.026813	...	0.189721	-0.217015	-0.211693	-0.155886	-0.144250	0.047227	0.052099	0.048255	0.057975	0.056666
DG5_5	-0.095018	-0.083681	0.036679	0.036055	-0.082289	-0.042983	-0.093037	-0.188122	0.071055	0.046286	...	0.188615	-0.221948	-0.217181	-0.128090	-0.130527	0.015289	0.023982	0.018973	0.031348	0.030811
DG5_6	-0.080020	-0.060849	-0.007916	0.025764	-0.060999	-0.013558	-0.082441	0.316647	-0.000799	0.091417	...	-0.062094	0.108776	0.116195	0.031268	0.027899	0.039024	0.085873	0.082777	0.062498	0.070624
DG5_7	-0.197286	-0.225239	0.015650	0.048293	-0.224925	-0.066428	-0.178122	0.117710	-0.014039	0.076727	...	0.007804	0.006876	0.011278	-0.043463	-0.060448	0.021054	0.066821	0.063595	0.054433	0.049514
DG5_8	-0.057252	-0.051730	0.016881	0.008396	-0.050931	-0.030535	-0.060312	0.025699	0.045429	0.008285	...	0.064053	-0.056913	-0.056818	-0.032654	-0.031372	-0.001871	0.001479	0.004429	0.012673	0.014545
DG5_9	-0.021554	-0.023415	-0.008501	0.008864	-0.023240	0.005295	-0.021400	0.010373	0.020041	-0.005701	...	0.009370	-0.000130	0.003937	0.016303	0.016482	0.005244	0.007046	0.008241	0.006106	0.006085
DG5_10	0.032645	0.060365	0.014300	0.015485	0.060514	-0.005961	0.013584	0.096897	0.052481	0.070115	...	0.072459	-0.085317	-0.096525	-0.138699	-0.132257	0.097405	0.146048	0.140282	0.149237	0.144097
DG5_11	-0.091877	-0.076698	-0.010588	-0.010845	-0.076748	-0.004892	-0.085002	0.050586	-0.000525	0.009727	...	-0.008460	-0.014488	-0.025911	-0.035305	-0.042430	0.029474	0.036100	0.029409	0.036763	0.039580
DG5_96	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DG6	0.025704	0.025620	-0.000754	0.026420	0.025932	-0.007148	0.012307	0.042499	-0.014939	0.479761	...	0.012811	-0.014676	-0.012259	-0.037911	-0.036533	0.076597	0.311009	0.286426	0.251959	0.251490
DG8a	0.062854	0.063675	0.010477	-0.027757	0.064010	-0.024270	0.055404	-0.006416	-0.051107	0.260244	...	-0.030658	0.027370	0.021140	-0.018662	-0.009311	0.039201	0.193996	0.176852	0.150377	0.148233
DG8b	-0.020534	-0.028756	0.005102	0.012616	-0.028782	0.002224	-0.014218	0.032801	0.019440	-0.010484	...	-0.008292	-0.000915	-0.000448	0.010176	0.013776	-0.000550	-0.006613	-0.005303	-0.010003	-0.011753
DG8c	-0.013769	-0.021483	0.007020	0.011470	-0.021286	0.008458	-0.006648	0.031191	-0.001314	-0.008665	...	-0.003084	-0.006042	-0.002656	0.008428	0.008612	0.004546	-0.001579	-0.002362	-0.003043	-0.004522
DG9a	0.041644	0.073122	0.011541	0.040599	0.073522	-0.005172	0.032153	-0.014864	-0.032850	0.041697	...	-0.040277	0.052551	0.050124	-0.059190	-0.047580	0.022344	0.045276	0.042611	0.041590	0.041769
DG9b	-0.014564	-0.008970	-0.008613	-0.006219	-0.009112	0.006641	-0.013580	-0.001240	-0.013550	-0.004291	...	-0.006796	-0.001219	-0.002040	-0.000289	-0.000147	-0.002221	-0.001298	-0.002694	-0.001910	-0.001818
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
FB27_9	-0.005475	-0.001515	0.040742	-0.008144	-0.000424	-0.030635	-0.011678	0.003731	0.015223	0.008504	...	0.045638	-0.079127	-0.073057	-0.053937	-0.041141	0.013009	0.014572	0.016747	0.014126	0.016025
FB27_96	0.001486	0.002822	0.001108	0.004338	0.002605	0.004728	0.001312	0.011297	-0.001348	0.000516	...	-0.005041	0.000034	0.000121	0.004941	0.010278	0.005006	0.004679	0.004882	0.005532	0.005469
FB28_1	0.004400	0.003552	0.005447	0.001014	0.003479	-0.004128	0.004070	0.011384	0.000020	-0.002651	...	-0.002529	0.005273	0.013251	0.012299	0.012296	0.000304	-0.001044	-0.000660	-0.000392	-0.000577
FB28_2	0.013003	0.011105	-0.002074	0.002675	0.011193	0.000796	0.014641	0.009082	0.009151	-0.002515	...	-0.006192	-0.000114	0.000322	0.012091	0.008258	-0.003499	-0.005180	-0.004153	-0.006230	-0.006191

FB28_3	-0.031844	-0.035787	-0.004682	0.014059	-0.036008	-0.000265	-0.030311	-0.005967	-0.007803	-0.002686	...	-0.006495	-0.003401	0.012373	0.019251	0.016743	-0.008760	-0.003461	-0.004085	-0.001387	-0.007131
FB28_4	-0.002309	-0.001975	0.012118	-0.001699	-0.002050	-0.000128	-0.002184	-0.006977	-0.006061	-0.000267	...	0.009551	-0.005359	0.000942	0.003954	0.004416	0.001081	-0.001503	-0.001987	-0.002419	-0.001739
FB28_5	-0.000046	0.000653	0.003271	0.005323	0.000834	0.003881	0.003957	0.003965	0.012236	-0.000312	...	0.001771	-0.012933	0.001595	0.010036	0.012416	0.003619	0.003163	0.003690	0.003879	0.003907
FB28_6	-0.001065	0.001962	0.022080	-0.005685	0.002226	-0.018909	0.001356	-0.000906	0.000697	0.001558	...	0.008723	-0.012336	-0.002319	0.003410	0.001974	0.003890	0.003311	0.003302	0.003599	0.003382
FB28_7	0.003050	0.006871	0.015835	-0.000872	0.007281	-0.016284	0.003280	0.013305	0.012196	-0.000255	...	0.004489	-0.003807	-0.006603	-0.006624	-0.009101	0.001152	0.001014	0.000986	-0.006546	-0.006512
FB28_8	-0.013552	-0.008910	0.000187	0.004390	-0.008656	-0.000438	-0.016021	0.002262	0.010204	-0.000404	...	0.001068	0.005045	0.007414	0.001725	-0.000341	0.001169	0.001251	0.000977	-0.012158	0.000924
FB28_9	-0.007770	-0.005924	-0.014095	0.010730	-0.006504	0.021526	0.000188	0.008386	-0.006890	-0.002263	...	-0.017025	0.032692	0.033037	0.020338	-0.000147	-0.009040	-0.006426	-0.007621	-0.009428	-0.009277
FB28_96	-0.009066	-0.009041	-0.000010	-0.008760	-0.009103	-0.000668	-0.006663	0.007710	0.006967	-0.000655	...	-0.004535	-0.006591	-0.006595	-0.007117	-0.005045	0.000167	-0.000374	-0.000042	-0.000048	-0.000047
FB29_1	0.008460	0.008780	-0.002245	-0.033959	0.010158	-0.023343	0.003577	0.017959	0.074136	-0.025209	...	0.046437	-0.082137	-0.069342	-0.050736	-0.036569	0.031063	0.016841	0.018213	0.022620	0.024633
FB29_2	-0.004963	-0.040509	-0.014824	-0.018907	-0.039758	0.002996	-0.006881	-0.003571	0.041202	-0.004032	...	0.050527	-0.079648	-0.066535	-0.042384	-0.024383	0.014140	0.020912	0.020353	0.016651	0.008682
FB29_3	0.020574	-0.005865	-0.003045	-0.017858	-0.005530	-0.006886	0.013891	-0.001744	0.034775	-0.045330	...	0.036940	-0.078462	-0.056360	-0.031674	-0.005043	0.013822	-0.005297	0.002248	0.006684	0.011614
FB29_4	-0.028625	-0.026940	-0.009845	-0.003890	-0.025995	0.003562	-0.027830	-0.006079	0.013612	0.002193	...	0.038286	-0.032410	-0.017424	0.015409	0.017459	0.002088	0.000318	0.005982	0.015394	0.015352
FB29_5	-0.020609	-0.016727	-0.011020	-0.001086	-0.015801	-0.022014	-0.022203	-0.016117	0.026661	-0.084356	...	0.059995	-0.092498	-0.061707	-0.014944	-0.011780	-0.010650	-0.039720	-0.032178	-0.026992	-0.020443
FB29_6	-0.019712	-0.031985	-0.001085	-0.003943	-0.030646	-0.021054	-0.023136	0.001474	0.012190	-0.004062	...	0.037055	-0.055677	-0.041587	-0.010308	-0.005592	0.008996	0.012875	0.008917	0.005310	0.010388
FB29_96	-0.002812	0.005561	-0.006197	-0.003399	0.005975	0.013808	-0.012672	0.017698	0.015466	0.005814	...	0.020917	-0.000265	0.002597	0.009634	0.015567	-0.000401	0.003527	0.001001	0.012100	0.004783
LN1A	-0.063363	-0.069334	0.058524	-0.005764	-0.065895	-0.099664	-0.083393	-0.220182	0.173237	0.075296	...	0.901146	-0.545334	-0.528078	-0.484621	-0.440339	0.063927	0.096611	0.087148	0.088840	0.091784
LN1B	-0.079768	-0.084329	0.055592	-0.008820	-0.081114	-0.095498	-0.098245	-0.203573	0.165073	0.048740	...	1.000000	-0.520494	-0.505620	-0.455148	-0.429517	0.057411	0.077606	0.071396	0.076774	0.078451
LN2_1	0.049245	0.061542	-0.065424	0.011688	0.057108	0.139811	0.050606	0.232229	-0.138924	-0.043356	...	-0.520494	1.000000	0.916975	0.522751	0.475593	-0.062987	-0.086477	-0.078974	-0.092940	-0.094708
LN2_2	0.053422	0.063216	-0.067240	0.010624	0.058756	0.142444	0.056100	0.238173	-0.137520	-0.042672	...	-0.505620	0.916975	1.000000	0.550462	0.499407	-0.060462	-0.081955	-0.072947	-0.089192	-0.091321
LN2_3	0.060630	0.050255	-0.074488	0.012946	0.046732	0.128753	0.088676	0.186134	-0.117376	-0.068006	...	-0.455148	0.522751	0.550462	1.000000	0.866993	-0.087549	-0.118866	-0.098540	-0.100352	-0.096124
LN2_4	0.063183	0.049743	-0.069536	-0.003407	0.046441	0.129005	0.101423	0.174095	-0.106413	-0.057339	...	-0.429517	0.475593	0.499407	0.866993	1.000000	-0.093990	-0.127607	-0.111249	-0.114725	-0.107199
GN1	0.020233	0.013123	0.017228	0.012594	0.013191	-0.015848	0.016100	-0.023267	0.047848	0.040091	...	0.057411	-0.062987	-0.060462	-0.087549	-0.093990	1.000000	0.506319	0.438242	0.384265	0.372087
GN2	0.048947	0.046940	0.022697	0.032619	0.047191	-0.007293	0.042819	-0.036997	0.021997	0.269473	...	0.077606	-0.086477	-0.081955	-0.118866	-0.127607	0.506319	1.000000	0.760556	0.645190	0.626430
GN3	0.049809	0.054093	0.020120	0.033587	0.054024	-0.000146	0.041620	-0.028059	0.022552	0.246047	...	0.071396	-0.078974	-0.072947	-0.098540	-0.111249	0.438242	0.760556	1.000000	0.724044	0.688290
GN4	0.031502	0.043527	0.014895	0.039596	0.043489	-0.004038	0.024165	-0.033726	0.019375	0.214297	...	0.076774	-0.092940	-0.089192	-0.100352	-0.114725	0.384265	0.645190	0.724044	1.000000	0.867174
GN5	0.031291	0.044589	0.015104	0.045022	0.044553	-0.005680	0.022936	-0.025619	0.027117	0.209442	...	0.078451	-0.094708	-0.091321	-0.096124	-0.107199	0.372087	0.626430	0.688290	0.867174	1.000000

Figure 2.12

The correlation matrix is of size 1045x1045, looking at the figure above, interpreting results can be quite difficult so a heatmap was mapped out (using the command sns.heatmap and giving correlation matrix as argument and x and y axis given column values) as given below:

```
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x169a5e5ce80>
```

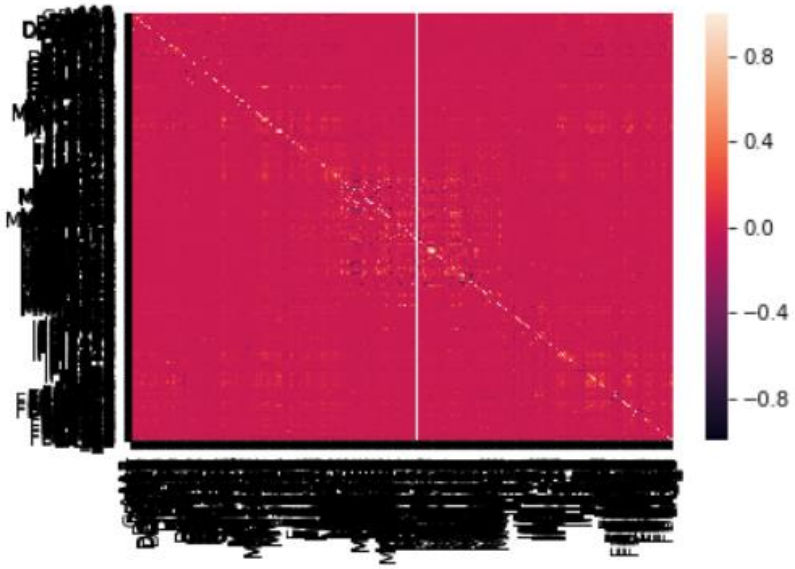


Figure 2.13

By Mode:

	AA3	AA4	AA5	AA6	AA7	AA14	AA15	DG1	Gender	DG3	...	LN1B	LN2_1	LN2_2	LN2_3	LN2_4	GN1	GN2	GN3	GN4	GN5
AA3	1.000000	0.925447	-0.005902	-0.114678	0.924611	0.117725	0.958523	-0.070281	-0.008179	0.018889	...	-0.079768	0.049245	0.053422	0.060630	0.063183	0.022775	0.048947	0.049809	0.031502	0.031291
AA4	0.925447	1.000000	0.012762	-0.090386	0.999793	0.084883	0.889837	-0.068858	-0.028591	0.031169	...	-0.084329	0.061542	0.063216	0.050255	0.049743	0.015705	0.046940	0.054093	0.043527	0.044589
AA5	-0.005902	0.012762	1.000000	-0.042032	0.018902	-0.505351	-0.007074	-0.003111	0.006411	-0.001953	...	0.055592	-0.065424	-0.067240	-0.074488	-0.069536	0.019871	0.022697	0.020120	0.014895	0.015104
AA6	-0.114678	-0.090386	-0.042032	1.000000	-0.076125	-0.184570	-0.090639	-0.008716	-0.007769	0.034782	...	0.085558	-0.121544	-0.123206	-0.089088	-0.093811	0.010549	0.024849	0.015629	0.022122	0.025948
AA7	0.924611	0.999793	0.018902	-0.076125	1.000000	0.074603	0.889307	-0.069531	-0.028565	0.031779	...	-0.081114	0.057108	0.058756	0.046732	0.046441	0.015951	0.047191	0.054024	0.043489	0.044553
AA14	0.117725	0.084883	-0.505351	-0.184570	0.074603	1.000000	0.131857	0.014166	-0.000338	0.004824	...	-0.095498	0.139811	0.142444	0.128753	0.129005	-0.025010	-0.007293	-0.000146	-0.004038	-0.005680
AA15	0.958523	0.889837	-0.007074	-0.090639	0.889307	0.131857	1.000000	-0.067746	-0.000432	0.011747	...	-0.098245	0.050606	0.056100	0.088676	0.101423	0.017429	0.042819	0.041620	0.024165	0.022936
DG1	-0.070281	-0.068858	-0.003111	-0.008716	-0.069531	0.014166	-0.067746	1.000000	0.067827	-0.040159	...	-0.203573	0.232229	0.238173	0.186134	0.174095	-0.037318	-0.036997	-0.028059	-0.033726	-0.025619
Gender	-0.008179	-0.028591	0.006411	-0.007769	-0.028565	-0.000338	-0.000432	0.067827	1.000000	-0.008646	...	0.165073	-0.138924	-0.137520	-0.117376	-0.106413	0.016402	0.021997	0.022552	0.019375	0.027117
DG3	0.018889	0.031169	-0.001953	0.034782	0.031779	0.004824	0.011747	-0.040159	-0.008646	1.000000	...	0.048740	-0.043356	-0.042672	-0.068006	-0.057339	0.025482	0.269473	0.246047	0.214297	0.209442
DG3A	0.015543	0.015987	0.019920	0.011398	0.016046	-0.000318	0.015110	0.013312	-0.042679	0.401518	...	0.014612	0.008779	0.005722	-0.028221	-0.031760	0.027762	0.203294	0.186594	0.165179	0.166348
DG4	0.048420	0.048365	-0.010886	-0.004931	0.047505	0.033999	0.040403	0.093414	-0.099952	0.473673	...	-0.114338	0.125327	0.125622	0.066324	0.055424	0.023768	0.260545	0.233828	0.202474	0.200310
DG5_1	-0.076690	-0.128986	-0.005729	0.057415	-0.128332	-0.020660	-0.061884	0.057116	0.005085	0.151817	...	0.039034	-0.025765	-0.033673	-0.051594	-0.055826	0.012237	0.094318	0.080326	0.079772	0.078855
DG5_2	-0.017259	0.012977	0.114460	0.119702	0.017696	-0.189568	-0.049358	0.010985	0.168942	0.007780	...	0.232248	-0.272913	-0.271533	-0.205272	-0.178385	0.024467	0.032251	0.031778	0.026897	0.027852
DG5_3	-0.095358	-0.089541	-0.009101	0.054601	-0.088759	-0.013821	-0.087506	-0.007063	0.092177	0.001713	...	0.095483	-0.053265	-0.048569	0.018519	0.018164	0.000303	0.007117	0.011378	0.013032	0.009998
DG5_4	-0.090789	-0.080796	0.034008	0.097529	-0.078206	-0.056312	-0.086374	-0.006521	0.267392	0.026813	...	0.189721	-0.217015	-0.211693	-0.155886	-0.144250	0.032238	0.052099	0.048255	0.057975	0.056666
DG5_5	-0.095018	-0.083681	0.036679	0.065088	-0.082289	-0.042983	-0.093037	-0.188122	0.071055	0.046286	...	0.188615	-0.221948	-0.217181	-0.128090	-0.130527	0.015859	0.023982	0.018973	0.031348	0.030811
DG5_6	-0.080020	-0.060849	-0.007916	0.015105	-0.060999	-0.013558	-0.082441	0.316647	-0.000799	0.091417	...	-0.062094	0.108776	0.116195	0.031268	0.027899	0.026259	0.085873	0.082777	0.062498	0.070624
DG5_7	-0.197286	-0.225239	0.015650	0.045663	-0.224925	-0.066428	-0.178122	0.117710	-0.014039	0.076727	...	0.007804	0.006876	0.011278	-0.043463	-0.060448	0.016160	0.066821	0.063595	0.054433	0.049514
DG5_8	-0.057252	-0.051730	0.016881	0.028843	-0.050931	-0.030535	-0.060312	0.025699	0.045429	0.008285	...	0.064053	-0.056913	-0.056818	-0.032654	-0.031372	-0.004845	0.001479	0.004429	0.012673	0.014545
DG5_9	-0.021554	-0.023415	-0.008501	0.014292	-0.023240	0.005295	-0.021400	0.010373	0.020041	-0.005701	...	0.009370	-0.000130	0.003937	0.016303	0.016482	0.002750	0.007046	0.008241	0.006106	0.006085
DG5_10	0.032645	0.060365	0.014300	0.010899	0.060514	-0.005961	0.013584	0.096897	0.052481	0.070115	...	0.072459	-0.085317	-0.096525	-0.138699	-0.132257	0.078103	0.146048	0.140282	0.149237	0.144097
DG5_11	-0.091877	-0.076698	-0.010588	-0.004692	-0.076748	-0.004892	-0.085002	0.050586	-0.000525	0.009727	...	-0.008460	-0.014488	-0.025911	-0.035305	-0.042430	0.021658	0.036100	0.029409	0.036763	0.039580
DG5_96	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
DG6	0.025704	0.025620	-0.000754	0.028759	0.025932	-0.007148	0.012307	0.042499	-0.014939	0.479761	...	0.012811	-0.014676	-0.012259	-0.037911	-0.036533	0.058279	0.311009	0.286426	0.251959	0.251490
DG8a	0.062854	0.083675	0.010477	-0.012732	0.084010	-0.024270	0.055404	-0.006416	-0.051107	0.260244	...	-0.030658	0.027370	0.021140	-0.018662	-0.009311	0.023815	0.193996	0.176852	0.150377	0.148233
DG8b	-0.020534	-0.028756	0.005102	0.007073	-0.028782	0.002224	-0.014218	0.032801	0.019440	-0.010484	...	-0.008292	-0.000915	-0.000448	0.010176	0.013776	-0.001948	-0.006613	-0.005303	-0.010003	-0.011753
DG8c	-0.013769	-0.021483	0.007020	0.012644	-0.021286	0.008458	-0.006648	0.031191	-0.001314	-0.008695	...	-0.003084	-0.006042	-0.002656	0.008428	0.008612	0.004228	-0.001579	-0.002362	-0.003043	-0.004522
DG9a	0.040373	0.071437	0.011433	0.033620	0.071830	-0.004788	0.031023	-0.014753	-0.032036	0.036886	...	-0.039859	0.052181	0.049888	-0.058632	-0.047200	0.013728	0.041763	0.039403	0.038828	0.039061
DG9b	-0.014564	-0.008970	-0.008613	-0.006059	-0.009112	0.006641	-0.013580	-0.001240	-0.013550	-0.004291	...	-0.006796	-0.001219	-0.002040	-0.000289	-0.000147	-0.001755	-0.001298	-0.002694	-0.001910	-0.001818
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
FB27_9	-0.005475	-0.001515	0.040742	0.021759	-0.000424	-0.030635	-0.011678	0.003731	0.015223	0.008504	...	0.045638	-0.079127	-0.073057	-0.053937	-0.041141	0.010721	0.014572	0.016747	0.014126	0.016025
FB27_96	0.001486	0.002822	0.001108	-0.004423	0.002605	0.004728	0.001312	0.011297	-0.001348	0.000516	...	-0.005041	0.000034	0.000121	0.004941	0.010278	0.003638	0.004679	0.004882	0.005532	0.005469
FB28_1	0.004400	0.003552	0.005447	0.000028	0.003479	-0.004128	0.004070	0.011384	0.000020	-0.002651	...	-0.002529	0.005273	0.013251	0.012299	0.012296	-0.001003	-0.001044	-0.000660	-0.000392	-0.000577
FB28_2	0.013003	0.011105	-0.002074	0.005176	0.011193	0.000796	0.014641	0.009082	0.009151	-0.002515	...	-0.006192	-0.000114	0.000322	0.012091	0.008258	-0.002481	-0.005180	-0.004153	-0.006230	-0.006191

																	0.012091	0.008258	-0.002481	-0.005180	-0.004153	-0.006230	-0.006191
FB28_3	-0.020181	-0.017218	-0.000335	0.031924	-0.016437	0.011816	-0.025050	0.002142	0.044970	0.014306	...	0.055744	-0.090221	-0.057412	-0.031787	-0.032115	0.013639	0.021630	0.022034	0.025907	0.025866
FB28_4	0.000680	0.000994	0.012951	-0.001263	0.001014	-0.002028	0.001782	-0.008402	-0.010151	-0.001050	...	0.006397	-0.002313	0.001994	0.004271	0.004350	-0.002060	-0.002060	-0.002241	-0.002760	-0.002507
FB28_5	0.018385	0.018028	0.011891	-0.013513	0.017779	0.008238	0.021144	0.009228	-0.007769	0.012908	...	-0.009612	0.022059	0.026080	0.011883	0.018190	-0.004611	-0.006132	0.000294	-0.007322	-0.007418
FB28_6	0.003741	0.005354	0.020861	-0.008731	0.005161	0.003987	0.006544	0.005558	0.009757	-0.002440	...	-0.005971	0.020440	0.016421	0.004286	0.001907	-0.001690	-0.003121	-0.003677	-0.004548	-0.004941
FB28_7	0.003050	0.006871	0.015835	0.008307	0.007281	-0.016284	0.003280	0.013305	0.012196	-0.000255	...	0.004489	-0.003807	-0.006603	-0.006624	-0.009101	0.000874	0.001014	0.000986	-0.006546	-0.006512
FB28_8	-0.013552	-0.008910	0.000187	0.011440	-0.008656	-0.000438	-0.016021	0.002262	0.010204	-0.000404	...	0.001068	0.005045	0.007414	0.001725	-0.000341	0.000936	0.001251	0.000977	-0.012158	0.000924
FB28_9	-0.007770	-0.005924	-0.014095	-0.014320	-0.006504	0.021526	0.000188	0.008386	-0.009890	-0.002263	...	-0.017025	0.032692	0.033037	0.020338	-0.000147	-0.007837	-0.006426	-0.007621	-0.009428	-0.009277
FB28_96	-0.008953	-0.009418	-0.000414	-0.005441	-0.009396	-0.002349	-0.006660	0.003019	0.006956	-0.000796	...	-0.002362	-0.006126	-0.006161	-0.008408	-0.008440	-0.001166	-0.002059	-0.001825	-0.002068	-0.002045
FB29_1	0.008460	0.008780	-0.002245	0.020903	0.010158	-0.023343	0.003577	0.017959	0.074136	-0.025209	...	0.046437	-0.082137	-0.069342	-0.050736	-0.036569	0.023316	0.016841	0.018213	0.022620	0.024633
FB29_2	-0.004963	-0.040509	-0.014824	0.016742	-0.039758	0.002996	-0.006881	-0.003571	0.041202	-0.004032	...	0.050527	-0.079648	-0.066535	-0.042384	-0.024383	0.009964	0.020912	0.020353	0.016651	0.008682
FB29_3	0.020574	-0.005865	-0.003045	0.000236	-0.005530	-0.006886	0.013891	-0.001744	0.034775	-0.045330	...	0.036940	-0.078462	-0.056360	-0.031674	-0.005043	0.010752	-0.005297	0.002248	0.006684	0.011614
FB29_4	-0.028625	-0.026940	-0.009645	0.031551	-0.025995	0.003562	-0.027830	-0.006079	0.013612	0.002193	...	0.038286	-0.032410	-0.017424	0.015409	0.017459	0.000310	0.000318	0.005982	0.015394	0.015352
FB29_5	-0.020609	-0.016727	-0.011020	0.032591	-0.015801	-0.022014	-0.022203	-0.016117	0.026661	-0.084356	...	0.059995	-0.092498	-0.061707	-0.014944	-0.011780	-0.010133	-0.039720	-0.032178	-0.026992	-0.020443
FB29_6	-0.019712	-0.031985	-0.001085	0.042690	-0.030646	-0.021054	-0.023136	0.001474	0.012190	-0.004062	...	0.037055	-0.055677	-0.041587	-0.010308	-0.005592	0.006483	0.012875	0.008917	0.005310	0.010388
FB29_96	-0.002812	0.005561	-0.006197	0.012572	0.005975	0.013808	-0.012672	0.017698	0.015466	0.005814	...	0.020917	-0.000265	0.002597	0.009634	0.015567	-0.002245	0.003527	0.001001	0.012100	0.004783
LN1A	-0.063363	-0.069334	0.058524	0.093484	-0.065895	-0.099664	-0.083393	-0.220182	0.173237	0.075296	...	0.901146	-0.545334	-0.528078	-0.484621	-0.440339	0.067601	0.096611	0.087148	0.088840	0.091784
LN1B	-0.079768	-0.084329	0.055592	0.085558	-0.081114	-0.095498	-0.098245	-0.203573	0.165073	0.048740	...	1.000000	-0.520494	-0.505620	-0.455148	-0.429517	0.062061	0.077606	0.071396	0.076774	0.078451
LN2_1	0.049245	0.061542	-0.065424	-0.121544	0.057108	0.139811	0.050606	0.232229	-0.138924	-0.043356	...	-0.520494	1.000000	0.916975	0.522751	0.475593	-0.064971	-0.086477	-0.078974	-0.092940	-0.094708
LN2_2	0.053422	0.063216	-0.067240	-0.123206	0.058756	0.142444	0.056100	0.238173	-0.137520	-0.042672	...	-0.505620	0.916975	1.000000	0.550462	0.499407	-0.063364	-0.081955	-0.072947	-0.089192	-0.091321
LN2_3	0.060630	0.050255	-0.074488	-0.089088	0.046732	0.128753	0.088676	0.186134	-0.117376	-0.068006	...	-0.455148	0.522751	0.550462	1.000000	0.866993	-0.085549	-0.118866	-0.098540	-0.100352	-0.096124
LN2_4	0.063183	0.049743	-0.069536	-0.093811	0.046441	0.129005	0.101423	0.174095	-0.106413	-0.057339	...	-0.429517	0.475593	0.499407	0.866993	1.000000	-0.093998	-0.127607	-0.111249	-0.114725	-0.107199
GN1	0.022775	0.015705	0.019871	0.010549	0.015951	-0.025010	0.017429	-0.037318	0.016402	0.025482	...	0.062061	-0.064971	-0.063364	-0.085549	-0.093998	1.000000	0.482804	0.418875	0.366945	0.354274
GN2	0.048947	0.046940	0.022697	0.024849	0.047191	-0.007293	0.042819	-0.036997	0.021997	0.269473	...	0.077606	-0.086477	-0.081955	-0.118866	-0.127607	0.482804	1.000000	0.760556	0.645190	0.626430
GN3	0.049809	0.054093	0.020120	0.015629	0.054024	-0.000146	0.041620	-0.028059	0.022552	0.246047	...	0.071396	-0.078974	-0.072947	-0.098540	-0.111249	0.418875	0.760556	1.000000	0.724044	0.668290
GN4	0.031502	0.043527	0.014895	0.022122	0.043489	-0.004038	0.024165	-0.033726	0.019375	0.214297	...	0.076774	-0.092940	-0.089192	-0.100352	-0.114725	0.366945	0.645190	0.724044	1.000000	0.867174
GN5	0.031291	0.044589	0.015104	0.025948	0.044553	-0.005680	0.022936	-0.025619	0.027117	0.209442	...	0.078451	-0.094708	-0.091321	-0.096124	-0.107199	0.354274	0.626430	0.668290	0.867174	1.000000

Figure 2.14

```
sns.heatmap(corr,
            xticklabels=corr.columns.values,
            yticklabels=corr.columns.values)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x1e617870748>
```

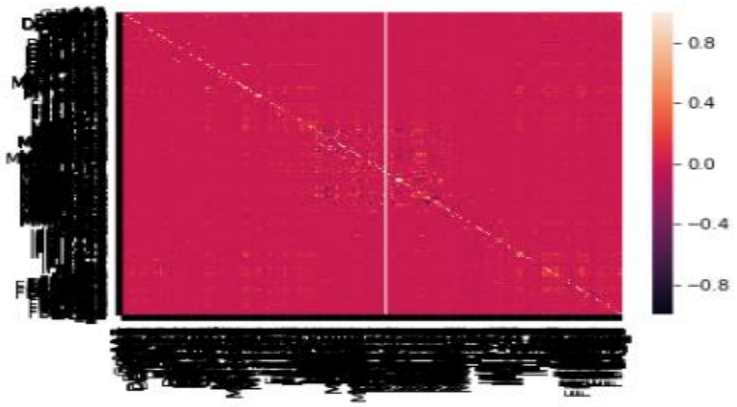


Figure 2.15

As we observe from the dataset, a few values are different when computed using mode rather than mean. However, the difference is so minute that it cannot be observed for which another heatmap was generated (I.e., figure 2.15) which cannot be differentiated from the one generated using mean method (I.e., figure 2.16).

Part 2: Task# 5:

As indicated by previous results of our models, they were overfitted, hence, nulling out the effective use of predictive techniques. To overcome it, we use L1 Regularization method i.e., use of Lasso, as we have huge number of features. Lasso(L1) shrinks the less important feature's coefficient to zero thus, removing some features altogether. This serves the purpose as feature selection reduces the number of total features.

```
linear_model2 = linear_model.Lasso(alpha=0.1)
```

Figure 2.16

Now the model is fitted with xTrain and yTrain datasets which were previously generated. Alpha's value is set at 0.1 which is the universal standard value.

```
linear_model2.fit(xTrain,yTrain)
```

```
Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
```

Figure 2.17

The accuracy of prediction is scored using xTrain and yTrain datasets and then by the testing datasets xTest and yTest. The accuracy pf predictions calculated is 82.98% (approximately) on training datasets and 82.75% (approximately). The previous for both training and test datasets was 100%, clearly the model is not over-fitted as accuracy for Training dataset is larger than the one of Testing dataset, depicting useful information and concluding accurate calculations.

```
accuracy = linear_model2.score(xTrain,yTrain)
```

```
accuracy
```

```
0.8297743823044037
```

Figure 2.18

```
accuracy = linear_model2.score(xTest,yTest)
```

```
accuracy
```

```
0.8274708121983319
```

Figure 2.19

By Mode:

```
linear_model2 = linear_model.Lasso(alpha=0.01)  #L1 Regularization
#Lasso(L1) shrinks the less important feature's coefficient to zero thus, removing some feature altogether.
#So, this works well for feature selection in case we have a huge number of features.
```

```
linear_model2.fit(xTrain,yTrain)
```

```
Lasso(alpha=0.01, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state=None,
   selection='cyclic', tol=0.0001, warm_start=False)
```

```
accuracy = linear_model2.score(xTrain,yTrain)#clearly this model is not overfitting.
```

```
accuracy
```

```
0.9981756210143089
```

```
accuracy = linear_model2.score(xTest,yTest) #This alpha is slightly reduced due to model being overfitting here
```

```
accuracy
```

```
0.998129820876364
```

Figure 2.20

Figure 2.20 shows the L1 regularization using mode method instead of mean (as done in previous step). For this model, Alpha value was chosen at 0.01 because when Alpha was chosen to be 0.1, in order to make them comparable, it did not converge due to which different values of Alpha were tried and tested. As Lasso tends to shrink the coefficients of less important features to zero, thus it removes some of the features. The accuracy obtained from this model is 99.81% (approximately) which is more than 17 % compared to the results obtained in previous step which means that using mode technique is much more effective for L1 regularization rather than using mean technique. The model is not over-fitted as accuracy for Training dataset is larger than the one of Testing dataset

Part 2: Task# 6:

Another regularization technique, Ridge, will be used to perform model testing and get much accurate results compared to previous methods like Lasso. Lasso regressions reduce over-fitting and emphasize on feature selection while Ridge regressions shrink the coefficients which helps reduce the model complexity and multi-collinearity, thus making more accurate predictions. To use Ridge, we called the function Ridge using our linear model and fitted it with xTrain and yTrain

datasets, which were generated in previous tasks. Alpha used for this model was 0.1 which is universal standard value.

```
linear_model3 = linear_model.Ridge(alpha=0.1)
```

Figure 2.21

```
linear_model3.fit(xTrain,yTrain)
```

```
Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

```
accuracy = linear_model3.score(xTrain,yTrain)
```

```
accuracy
```

```
0.9999999983324558
```

```
accuracy = linear_model3.score(xTest,yTest)
```

```
accuracy
```

```
0.9999999974364979
```

Figure 2.22

The accuracy of prediction on training and testing dataset have negligible difference, both being 99.99% (approximately) accurate. Model solved with Ridge is much more accurate compared to Lasso because of its general purpose of reducing complexity and multi-collinearity in the model. The model is not over-fitted as accuracy for Training dataset is larger than the one of Testing dataset.

By Mode:

```
linear_model3 = linear_model.Ridge(alpha=0.1) #L2 Regularization
```

```
linear_model3.fit(xTrain,yTrain)
```

```
Ridge(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

```
accuracy = linear_model3.score(xTrain,yTrain)
```

```
accuracy
```

```
0.99999999832826
```

```
accuracy = linear_model3.score(xTest,yTest)
```

```
accuracy
```

```
0.9999999939605437
```

Figure 2.23

Using mode technique to run Ridge model, we get results very close to what we got using mean method as can be observed by Figure 2.22 and 2.23. This model gives better accuracy with mean method; however, the difference is negligible. The model is not over-fitted as accuracy for Training dataset is larger than the one of Testing dataset.

Part 3: BONUS TASK# 1:

For this part, a logistic regression model was implemented. Liblinear algorithm was used over 10000 iterations (default value is 1000) to initialize the logistic model.

```
logistic_model = LogisticRegression(solver='liblinear',max_iter=10000)
```

Figure 3.1

The logistic model is fitted with xTrain and yTrain datasets generated in the previous tasks (using previously generated data). The model is then checked for accuracy which turns out to 98.42 % (approximately). Then, the model is used to predict yTest given the argument xTest values. Mean squared error term is calculated using yTest dataset and y_predictions (predicted values of y generated in previous step) and further used to calculate Root Mean Squared Error (RMSE). RMSE calculates the standard deviation, i.e., how spread out, the residuals are (Residuals are a measure of how far from the regression line data points are). Lastly, the model predictions are scored for accuracy which turn out to be 97.95% (approximately) accurate. The model is not over-fitted as accuracy for Training dataset is larger than the one of Testing dataset.

```
logistic_model.fit(xTrain,yTrain)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=10000, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='liblinear',
          tol=0.0001, verbose=0, warm_start=False)
```

```
accuracy = logistic_model.score(xTrain,yTrain)
```

```
accuracy
```

```
0.9841824157764996
```

```
y_predictions = logistic_model.predict(xTest)
mse = mean_squared_error(yTest, y_predictions)
rmse = math.sqrt(mse)

print('RMSE: {}'.format(rmse))
```

```
RMSE: 0.14332591242820383
```

```
accuracy = logistic_model.score(xTest,yTest)
```

```
accuracy
```

```
0.9794576828266228
```

Figure 3.2

By Mode:

```
logistic_model = LogisticRegression(solver='liblinear',max_iter=1000)
#solver is the algorithm
#max_iter is the number of iterations the default is 1000
```

```
clf = logistic_model.fit(xTrain, yTrain);
```

```
accuracy = logistic_model.score(xTrain,yTrain)
```

```
accuracy
```

```
0.9847986852917009
```

```
y_predictions = logistic_model.predict(xTest)
mse = mean_squared_error(yTest, y_predictions)
rmse = math.sqrt(mse)

print('RMSE: {}'.format(rmse))
```

```
RMSE: 0.1480263658438443
```

```
accuracy = logistic_model.score(xTest,yTest)
```

```
accuracy
```

```
0.9780881950150644
```

Figure 3.3

Using mode technique to run logistic regression, we find out that root squared mean error term is larger compared to that calculated using mean method. This means that the accuracy calculated using mode technique would be lesser than that of mean method, the difference, though not large, can be observed from figures 3.2 and 3.3. The model is not over-fitted as accuracy for Training dataset is larger than the one of Testing dataset.

Part 3: BONUS TASK# 2:

For this task, K means clustering unsupervised learning algorithm was used. A cluster refers to a collection of data points aggregated together because of certain similarities. We define a target number *k*, which refers to the number of centroids we need in the dataset. A centroid is the imaginary or real location representing the center of the cluster. The k-means score is an indication of how far the points are from the centroids. We run the below shown commands:

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=0).fit(xTrain)

kmeans.labels_

array([0, 0, 0, ..., 1, 0, 1])

kmeans.score(xTrain,yTrain)

-39296221347868.33
```

Figure 3.3

In scikit learn, more the score is closer to 0, the better it is. Bad scores return a large negative number, whereas good scores return close to zero. Generally, we do not take the absolute value of the output from the score's method for better visualization. Since a very large negative value is estimated, our data set is not fit for this technique.

By Mode:

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, random_state=0).fit(xTrain)

kmeans.labels_

array([0, 0, 0, ..., 1, 0, 1])

kmeans.score(xTrain,yTrain)

-39296292487635.02

kmeans1 = KMeans(n_clusters=3, random_state=0).fit(xTrain)

kmeans1.score(xTrain,yTrain)

-22088057945246.14

kmeans2 = KMeans(n_clusters=4, random_state=0).fit(xTrain)

kmeans2.score(xTrain,yTrain)

-9799069939612.398
```

Figure 3.4

Finding k mean clustering using mode technique gives much worse results compared to that of mean method. As known to us before, larger negative numbers indicate bad scores and as we get closer to 0, our scores are referred to be good. Since, it is not the case in this scenario as seen in figure 3.4 where scores are much worse compared to those in figure 3.3, dataset generated using mode will certainly be not fit for this technique.

Part 3: BONUS TASK# 3:

For this part, we use NearestCentroid. It is a K nearest neighbours is an unsupervised learning algorithm. This algorithm initializes cluster centroids and then calculates Euclidean distance. The model is then fitted with xTrain and yTrain datasets. The model assigns values based on Euclidean distance from either of the clusters (male and female is this context). The commands are shown below:

```
from sklearn.neighbors.nearest_centroid import NearestCentroid
```

```
knn = NearestCentroid()s
```

```
knn.fit(xTrain,yTrain)
```

```
NearestCentroid(metric='euclidean', shrink_threshold=None)
```

```
knn.score(xTest,yTest)
```

```
0.5321829635716242
```

Figure 3.5

The accuracy of this model is 53.22% (approximately) which is quite low thus, we can conclude K nearest neighbor is a weak classifier as shown from the knn.score on test data.

By Mode:

```
from sklearn.neighbors.nearest_centroid import NearestCentroid
from sklearn.neighbors import kneighbors_graph
```

```
knn = NearestCentroid()
```

```
knn.fit(xTrain,yTrain)
```

```
NearestCentroid(metric='euclidean', shrink_threshold=None)
```

```
knn.score(xTest,yTest)
```

```
0.5321829635716242
```

```
graph = kneighbors_graph(xTrain, 2, mode='connectivity', include_self=True)
```

```
graph
```

```
<14604x14604 sparse matrix of type '<class 'numpy.float64'>'
        with 29208 stored elements in Compressed Sparse Row format>
```

Figure 3.6

As we know that K nearest neighbors is processed using the Euclidean distance so we would expect the results to be pretty similar because using either mean or mode would not create a difference in the distance from the centroids. This can be observed in figure 3.6 that results are same as those in figure 3.5 with accuracy of both being 53.22%. This reaffirms the fact that solving using mean or mode for K nearest neighbors results in no difference.

Part 3: BONUS TASK# 4:

For feature selection, in this part, we used chi-squared method. Firstly, gender variable was stored in y variable from the dataset. Then, a variable x was generated which stores all columns except for the gender one. Chi-Squared test is then used for feature selection, in this case we choose top 1 feature so to that k=1 was chosen. The result was stored in x_new which is an array of values shown below:

```
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2

y = df['Gender']

x = df.drop(df.columns[df.columns.str.contains('Gender',case = False)],axis = 1)

x_new = SelectKBest(chi2, k=1).fit_transform(x, y)

y = y.values

x_new

array([[323011.],
       [268131.],
       [167581.],
       ...,
       [346191.],
       [146041.],
       [166161.]])
```

Figure 3.7

By Mode:

```
y = df['Gender']

x = df.drop(df.columns[df.columns.str.contains('Gender',case = False)],axis = 1)

x_new = SelectKBest(chi2, k=5).fit_transform(x, y)

y = y.values

x_new

array([[323011.],
       [268131.],
       [167581.],
       ...,
       [346191.],
       [146041.],
       [166161.]])
```

Figure 3.8

Similar to what we did in previous step, we do using mode technique rather than mean. We used top 5 features so k was set equal to 5. However, the resultant array was similar to that generated by mean method.

Part 3: BONUS TASK# 5:

For this part, we use the array x_new, generated in previous task. We also use keras for modelling in this task. It allows creation of layer-by-layer model for most problems. It is a method of making neural network for prediction. We used 10 epochs to improve the accuracy of the learning of our model. The result turns out to be approximately 53.71% after every epoch. It is known that increasing the number of epochs generally leads to increase in accuracy but what we can infer after looking into the detailed result is that increasing the number of epochs would not have much effect on the prediction accuracy as it doesn't improve over every iteration so this accuracy must be the best achieved from this method, in this scenario. The expression "dropout" alludes to dropping out units in a neural system. More actually, at each preparation arrange, hubs are either dropped out of the net with likelihood 1-p or kept with likelihood p, so a decreased system is left; approaching and friendly edges to a dropped-out hub are additionally evacuated. The motivation to add dropout is to abstain from overfitting.

Observation about dropout:

Dropout powers a neural system to adapt progressively hearty highlights that are valuable related to a wide range of irregular subsets of different neurons.

Dropout generally copies the quantity of cycles required for the system to combine. In any case, preparing for every age is left.

With H concealed units, every one of which can be dropped, we have 2^H conceivable models. In testing stage, the whole system is considered, and every actuation is diminished by a factor of p.

```
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.layers import Dropout
```

```
model = Sequential([
    Dense(30, input_shape=(1,)),
    Activation('relu'),
    Dropout(0.2),
    Dense(1),
    Activation('softmax'),
])
```

```
x_new[94]
```

```
array([256061.])
```

```
y[94]
```

```
1
```

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
model.fit(x_new,y, epochs=10, batch_size=32)
#Acc
```

```
Epoch 1/10
18255/18255 [==============================] - 1s 62us/step - loss: 7.3795 - acc: 0.5371
Epoch 2/10
18255/18255 [==============================] - 1s 31us/step - loss: 7.3795 - acc: 0.5371
Epoch 3/10
18255/18255 [==============================] - 1s 30us/step - loss: 7.3795 - acc: 0.5371
Epoch 4/10
18255/18255 [==============================] - 1s 30us/step - loss: 7.3795 - acc: 0.5371
Epoch 5/10
18255/18255 [==============================] - 1s 30us/step - loss: 7.3795 - acc: 0.5371
Epoch 6/10
18255/18255 [==============================] - 1s 30us/step - loss: 7.3795 - acc: 0.5371
Epoch 7/10
18255/18255 [==============================] - 1s 30us/step - loss: 7.3795 - acc: 0.5371
Epoch 8/10
18255/18255 [==============================] - 1s 31us/step - loss: 7.3795 - acc: 0.5371
Epoch 9/10
18255/18255 [==============================] - 1s 30us/step - loss: 7.3795 - acc: 0.5371
Epoch 10/10
18255/18255 [==============================] - 1s 30us/step - loss: 7.3795 - acc: 0.5371

<keras.callbacks.History at 0x2dd8803e860>
```

Figure 3.9

By Mode:

```
model.compile(optimizer='adam',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

```
history = model.fit(xTest, yTest,epochs=10)
#Acc
```

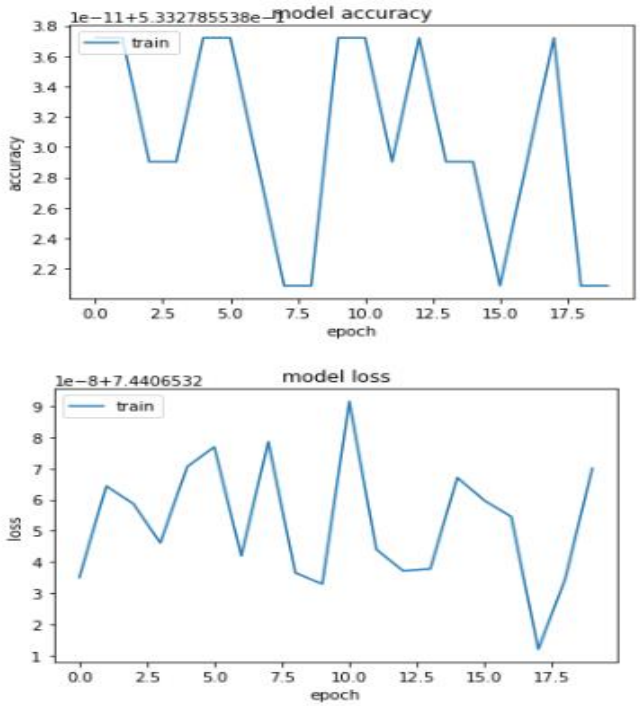
```
Epoch 1/10
3651/3651 [==============================] - 4s 1ms/step - loss: 7.4407 - acc: 0.5333
Epoch 2/10
3651/3651 [==============================] - 2s 462us/step - loss: 7.4407 - acc: 0.5333
Epoch 3/10
3651/3651 [==============================] - 2s 470us/step - loss: 7.4407 - acc: 0.5333
Epoch 4/10
3651/3651 [==============================] - 2s 490us/step - loss: 7.4407 - acc: 0.5333
Epoch 5/10
3651/3651 [==============================] - 2s 510us/step - loss: 7.4407 - acc: 0.5333
Epoch 6/10
3651/3651 [==============================] - 2s 464us/step - loss: 7.4407 - acc: 0.5333
Epoch 7/10
3651/3651 [==============================] - 2s 477us/step - loss: 7.4407 - acc: 0.5333
Epoch 8/10
3651/3651 [==============================] - 2s 485us/step - loss: 7.4407 - acc: 0.5333
Epoch 9/10
3651/3651 [==============================] - 2s 475us/step - loss: 7.4407 - acc: 0.5333
Epoch 10/10
3651/3651 [==============================] - 2s 466us/step - loss: 7.4407 - acc: 0.5333
```

v

Figure 3.10

For this part, we use mode instead of mean to build a neural network. The results are not much different but as it can be observed from figures 3.9 and 3.10, mode results into an increase in loss and thus accuracy although the number of epochs remain the same.

Graphs:



Graph 2

The above generated graphs show the accuracy and loss levels over different number of epochs. The graph span over 20 epochs and as it can observed at 10 number of epochs, the loss and accuracy are the highest. The number of epochs chosen for this task was also 10 because the combination of the accuracy and loss at these many epochs generated better results. As the number of epochs tend to increase towards 20, the accuracy of train dataset tends to decrease to lowest point while the loss starts to increase which means that moving towards higher number of epochs will only lead to decreased accuracy rates.

Part 3: BONUS TASK #6:

For this part, we use Random forest classifier. It creates a set of decision trees from randomly selected subset of training set. It then aggregates the votes from different decision trees to decide the final class of the test object. The xTrain and xTest datasets generated in previous tasks was scaled using sc.fit_transform and sc.transform commands. Classifier is then run with parameters n_estimator = 1045 (as we had 1045 columns) and random_state = 0. Classifier is, then, fitted with xTrain and yTrain datasets previously generated. Train accuracy was 100% while Test accuracy was 99.34%. Since, the difference is not too much, we can say that it was not overfitted.

```
from sklearn.ensemble import RandomForestClassifier
```

```
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score
```

```
#first we need to do scaling
sc = StandardScaler()
xTrain = sc.fit_transform(xTrain)
xTest = sc.transform(xTest)
```

```
classifier = RandomForestClassifier(n_estimators=1045, random_state=0)
classifier.fit(xTrain, yTrain)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1045, n_jobs=None,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

```
classifier.score(xTrain,yTrain)
```

```
1.0
```

```
classifier.score(xTest,yTest)
```

```
0.9934264585045193
```

Figure 3.11

By Mode:

```
sc = StandardScaler()
xTrain = sc.fit_transform(xTrain)
xTest = sc.transform(xTest)
```

```
classifier = RandomForestClassifier(n_estimators=1045, random_state=0)
classifier.fit(xTrain, yTrain)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=1045, n_jobs=None,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

```
classifier.score(xTrain,yTrain)
```

```
1.0
```

```
classifier.score(xTest,yTest)
```

```
0.9928786633798959
```

Figure 3.12

For this part, we used mode instead of mean to process Random Forest classifier. The classifier had same settings as in the previous step (n_estimators = 1045, the number of columns, and random_state = 0 (default)). The accuracy using training datasets i.e., xTrain and yTrain was 100% while for test dataset, it was 99.29% (approximately) which compared to what was calculated by mean method is 0.05% lower which is not a huge difference.

Part 3: BONUS TASK #7:

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of particular feature in a class is unrelated to the presence of any other feature. The model was trained using the training datasets generated previously ie., xTrain and yTrain with an accuracy of 59.52% (approximately) while accuracy at testing was 57.46% (approximately). Figure 3.13 depicts the results generated. Guassian is used in classification and assumes that feature follow a normal distribution due to which the accuracy is not at par with other methods. It can be observed from the figure that 5912 points were mislabeled out of 14604 which makes up 40.5% (approximately) of the dataset.

```
gaussian_naive_bayes = GaussianNB() #This calculates posterior probability
```

```
gaussian_naive_bayes.fit(xTrain,yTrain)
```

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

```
y_pred = gaussian_naive_bayes.predict(xTrain)
```

```
gaussian_naive_bayes.score(xTrain,yTrain)
```

```
0.5951794029033142
```

```
gaussian_naive_bayes.score(xTest,yTest)
```

```
0.574637085729937
```

```
gaussian_naive_bayes.predict_proba(xTest)
```

```
array([[1.02454496e-01, 8.97545504e-01],
       [5.56238557e-16, 1.00000000e+00],
       [9.65703045e-04, 9.99034297e-01],
       ...,
       [1.00000000e+00, 5.03557848e-17],
       [5.95387393e-05, 9.99940461e-01],
       [8.70790208e-03, 9.91292098e-01]])
```

```
print("Number of mislabeled points out of a total %d points : %d"
      % (xTrain.shape[0],(yTrain != y_pred).sum()))
```

```
Number of mislabeled points out of a total 14604 points : 5912
```

Figure 3.13

Part 3: BONUS TASK #8:

```
pca = PCA(n_components=2)
```

```
pca.fit(x)
```

```
PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)
```

```
pca.explained_variance_ratio_
```

```
array([0.95215806, 0.04783313])
```

```
pca.explained_variance_
```

```
array([9.63168513e+09, 4.83862596e+08])
```

```
pca.get_covariance
```

```
<bound method _BasePCA.get_covariance of PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)>
```

```
pca.mean_
```

```
array([ 2.37145987, 28.55831279,  3.04300192, ...,  7.4219118 ,
        8.92588332,  8.81736511])
```

```
pca.score_samples(xTrain)
```

```
array([-3461.55721058, -3873.00588152, -3676.76662074, ...,
       -3636.46207583, -3635.4668378 , -4511.30928424])
```

```
pca.singular_values_
```

```
array([13259592.01099049,  2971940.07771592])
```

```
pca.explained_variance_ratio_
```

```
array([0.95215806, 0.04783313])
```

```
pca.svd_solver
```

```
'auto'
```

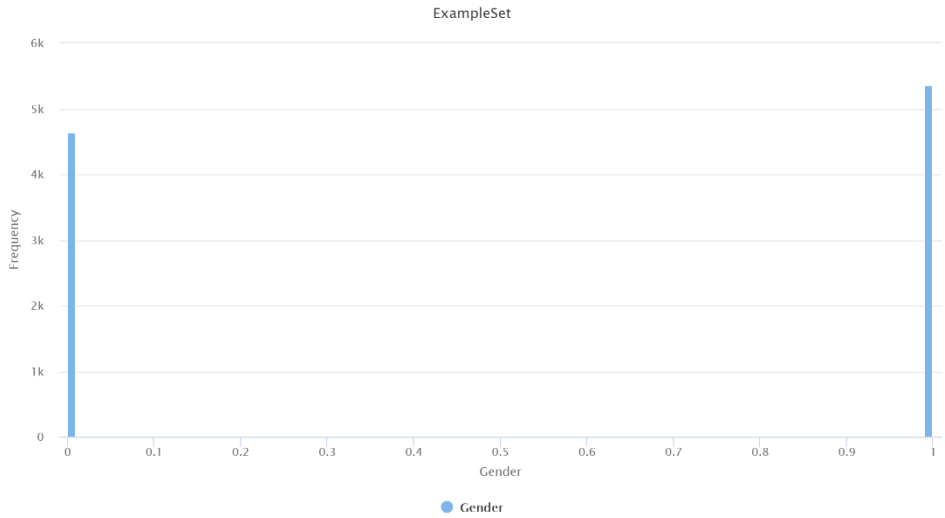
```
pca.get_precision
```

```
<bound method _BasePCA.get_precision of PCA(copy=True, iterated_power='auto', n_components=2, random_state=None,
  svd_solver='auto', tol=0.0, whiten=False)>
```

Figure 3.14

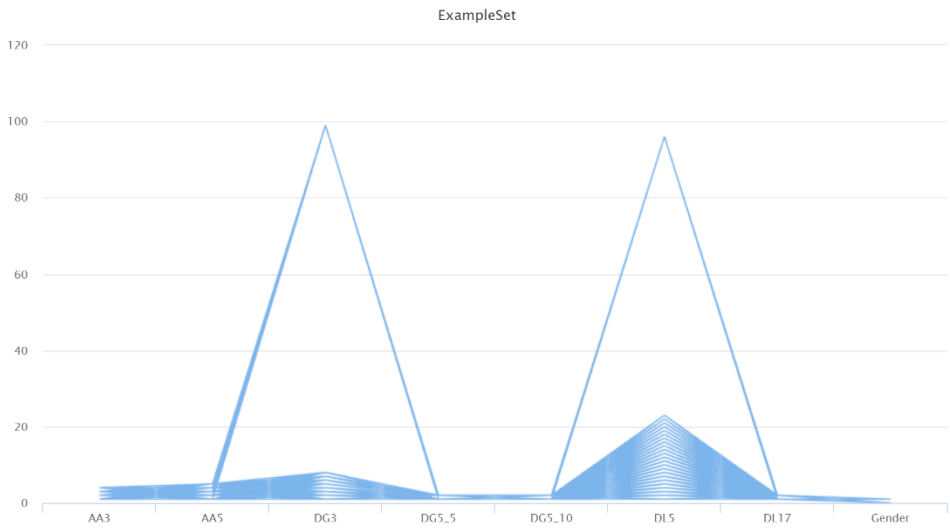
Principal Component Analysis is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called principal components.

ADDITIONAL GRAPHS:



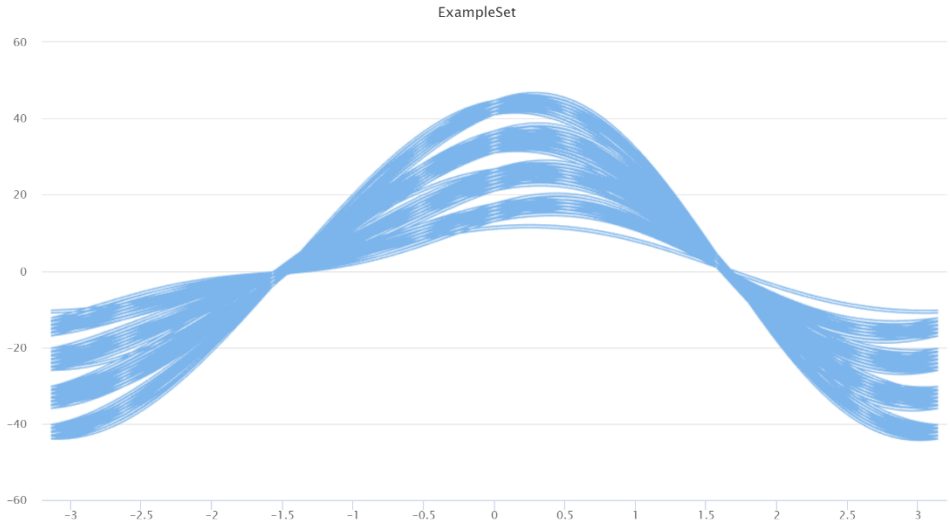
Graph 3

Histogram of gender was generated as shown in graph 3. 0 was assigned to Male gender and1 to Female gender. It can be observed from the graph that Female gender had more representation in the dataset compared to that of Male gender. Female representation is approximately 5.4k while Male is around 4.8k which means that there were 600 more Females giving the survey.



Graph 4

Graph 4 shows parallel coordinates graph generated over the variables AA3 (Zone), AA5 (Town Class), DG3 (Martial Status), DG5_5 (School/College ID), DG5_10 (Bank Passbook), DL5 (Means of Income), DL17 (Possess a stove burner) and Gender. Parallel coordinates graphing is a common way of visualizing high-dimensional geometry and analyzing multivariate data. It can be observed from the plot that when Zone increases i.e., from North to South, the Town class also increases from Class 1 to Class 5 which would mean that South is considered to be better than Class 1, hence North better than South. Similarly, value of Marital status also increases which means that individuals are more likely to be settled i.e., to be married or living with partners with/without being married, a shift from being single, widowed or divorced. The individuals are also more likely to have School/College IDs which means that as Town class increases, likely hood of getting advanced education increases (2 = No, 1 = Yes). Furthermore, individuals have Bank passbooks and are more likely to have white-collar jobs or run their own businesses. An intriguing discovery is that as class is increasing, chances of possessing a stove burner decrease which means that such individuals prefer take-outs or dining out rather than cooking themselves. This behavior can be considered strange that given higher levels of education and better jobs, such individuals should be aware of healthy diet, however, given the fact that it is a developing country, lesser campaigns on awareness of healthy diet can be given the credit for this. Below is another representation of the aforementioned model.



Graph 5

Graph 5 depicts Andrew curve generated over classes AA3, AA5, DG3, DG5_5 DG5_10, DL5, DL17 and Gender. Andrew curve is also a way of visualizing high-dimensional data and a smoothened version of parallel coordinates plot (plotted in Graph 4).

Works Cited

Nagarajan, Rema. “The Times of India”. *Women account for just 22% of workforce in India.* The Times of India. 29 Nov 2013. Web. 20 April 2019.

<https://timesofindia.indiatimes.com/india/Women-account-for-just-22-of-workforce-in-India/articleshow/26548372.cms>