<u>Machine Learning Project</u>

Comparison of different methods of handling missing values and different classification algorithms

THAPAR INSTITUTE
OF ENGINEERING & TECHNOLOGY
(Deemed to be University)

Submitted to Dr. Sushma Jain

Name - Tanya Srivastava

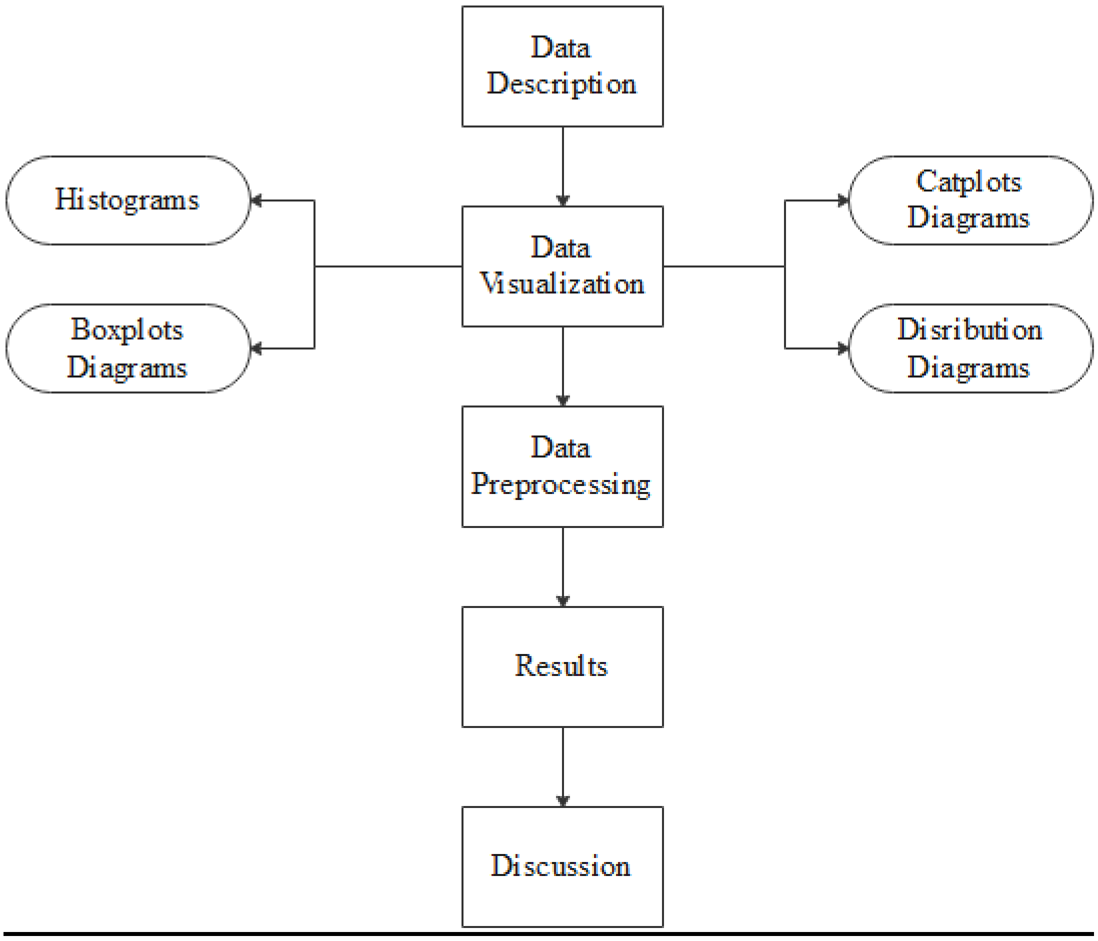
Roll No. - 101918031

Batch - COBS 2

Index

Overview

The flow of the project follows the following diagram. We first describe the dataset then focus the data visualization to get a better sense of the data. Data preprocessing involves various steps like

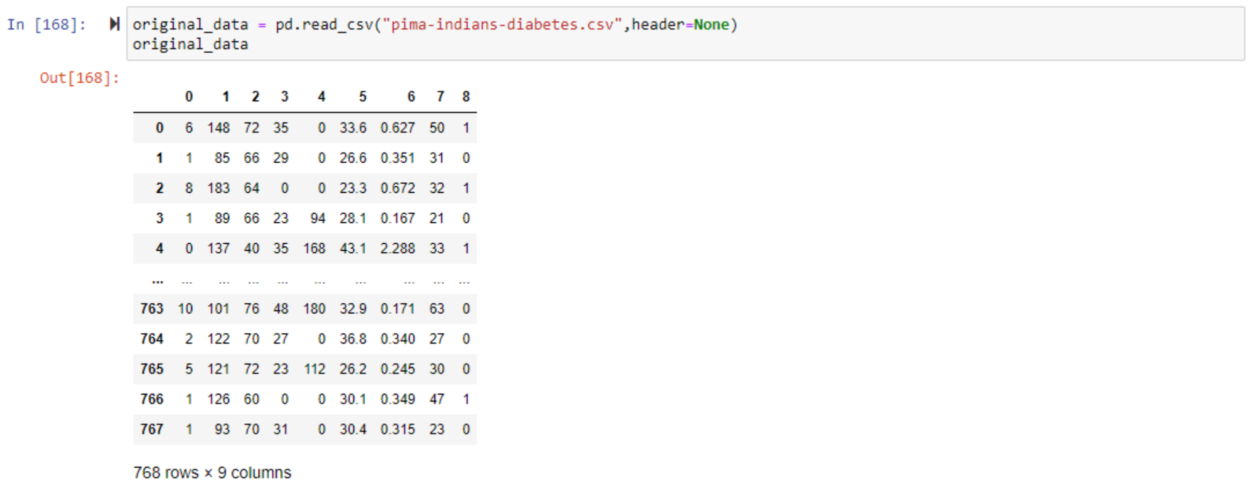


Dataset Description

The table below gives a brief description of the dataset used in this project.

Dataset	pima-indians-diabetes.csv
Number of attributes	8
Number of targets	1
Number of records	768
Type of Target	Categorical
Type of attributes	Numerical
Type of Problem	Classification (Binary)

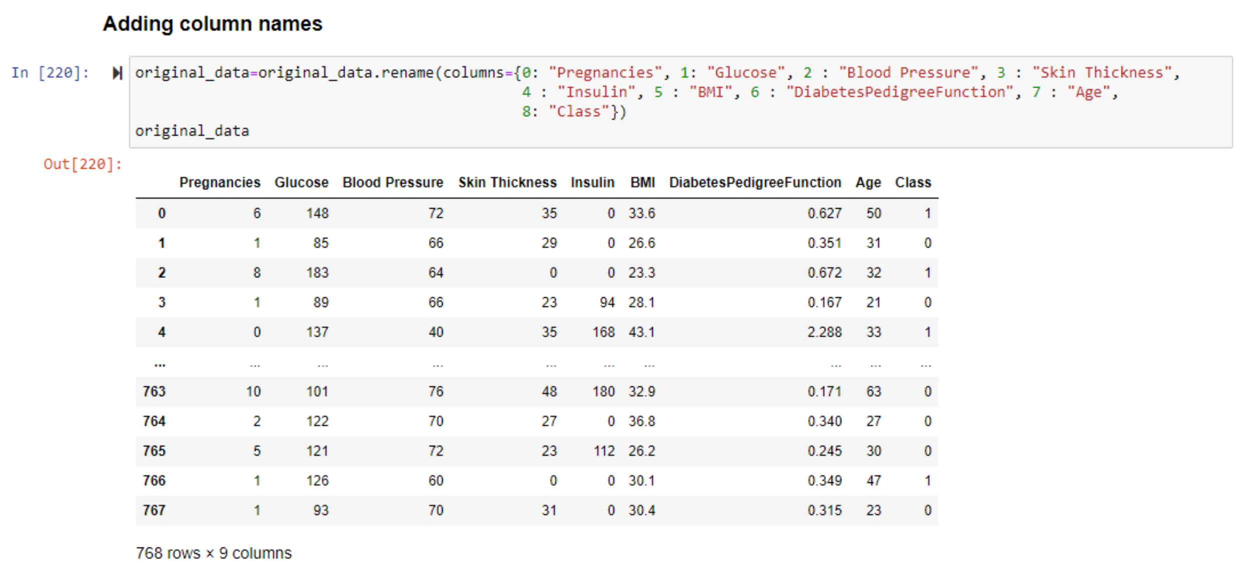
The initial view of the data in a pandas data frame was as follows



```
In [168]:  original_data = pd.read_csv("pima-indians-diabetes.csv",header=None)
           original_data
```

```
Out[168]:
            0    1   2   3    4     5      6   7   8
       0    6  148  72  35    0  33.6  0.627  50   1
       1    1   85  66  29    0  26.6  0.351  31   0
       2    8  183  64   0    0  23.3  0.672  32   1
       3    1   89  66  23   94  28.1  0.167  21   0
       4    0  137  40  35  168  43.1  2.288  33   1
     ...  ...  ... ... ...  ...   ...    ... ... ...
     763   10  101  76  48  180  32.9  0.171  63   0
     764    2  122  70  27    0  36.8  0.340  27   0
     765    5  121  72  23  112  26.2  0.245  30   0
     766    1  126  60   0    0  30.1  0.349  47   1
     767    1   93  70  31    0  30.4  0.315  23   0

    768 rows × 9 columns
```

After labelling the attributes in the data frame, we get the following



Adding column names

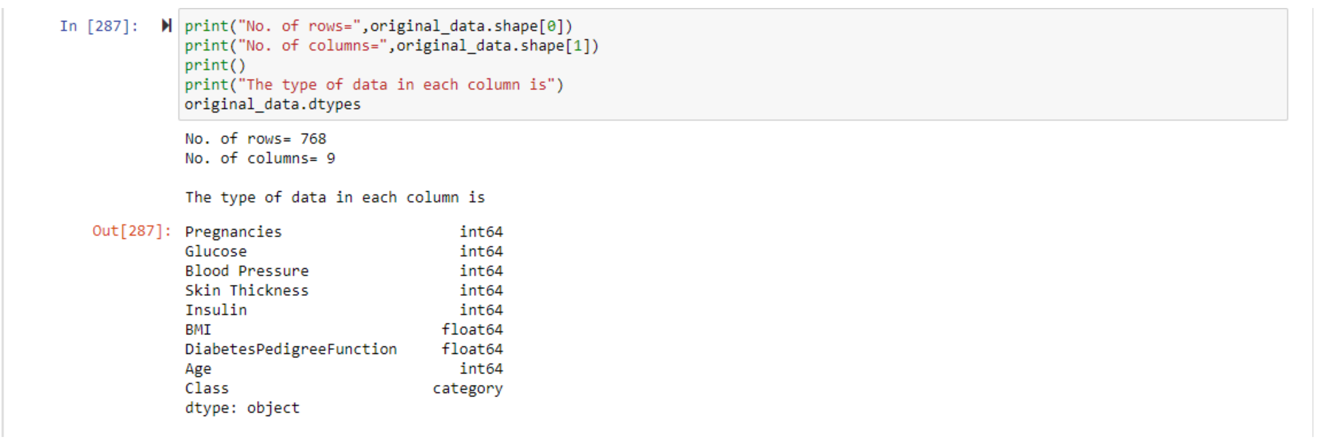
```
In [220]:  original_data=original_data.rename(columns={0: "Pregnancies", 1: "Glucose", 2 : "Blood Pressure", 3 : "Skin Thickness",
                                                    4 : "Insulin", 5 : "BMI", 6 : "DiabetesPedigreeFunction", 7 : "Age",
                                                    8: "Class"})
           original_data
```

Out[220]:

	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Class
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1
...	...	...	...	...	...	...	...	...	...
763	10	101	76	48	180	32.9	0.171	63	0
764	2	122	70	27	0	36.8	0.340	27	0
765	5	121	72	23	112	26.2	0.245	30	0
766	1	126	60	0	0	30.1	0.349	47	1
767	1	93	70	31	0	30.4	0.315	23	0

768 rows × 9 columns

A more detailed analysis of the columns in the dataset (especially the datatype of the attribute and target) is presented as follows



```
In [287]:  print("No. of rows=",original_data.shape[0])
           print("No. of columns=",original_data.shape[1])
           print()
           print("The type of data in each column is")
           original_data.dtypes

           No. of rows= 768
           No. of columns= 9

           The type of data in each column is

Out[287]:  Pregnancies                    int64
           Glucose                        int64
           Blood Pressure                 int64
           Skin Thickness                 int64
           Insulin                        int64
           BMI                          float64
           DiabetesPedigreeFunction     float64
           Age                            int64
           Class                       category
           dtype: object
```

COLUMN NAME	**DESCRIPTION**

Pregnancies	Number of pregnancies
	`count    768.000000`
	`mean       3.845052`
	`std        3.369578`
	`min        0.000000`
	`25%        1.000000`
	`50%        3.000000`
	`75%        6.000000`
	`max       17.000000`
	`Name: Pregnancies, dtype: float64`

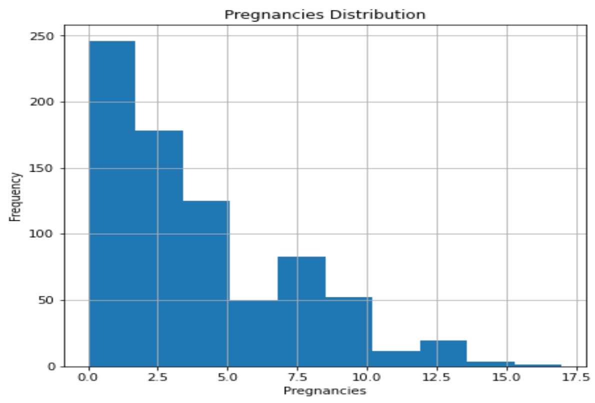
Glucose	Self-explanatory, cannot be zero

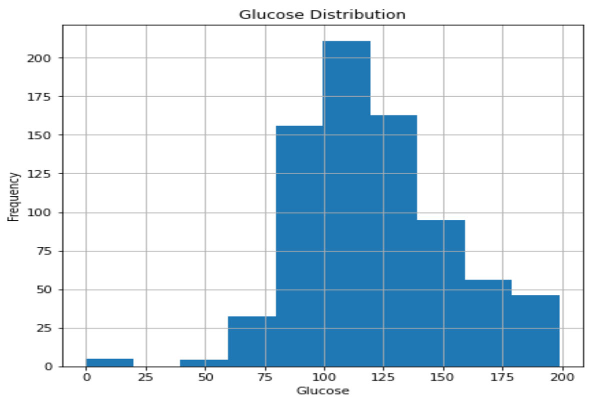
	count 768.000000 mean 120.894531 std 31.972618 min 0.000000 25% 99.000000 50% 117.000000 75% 140.250000 max 199.000000 Name: Glucose, dtype: float64
Blood Pressure	Self-explanatory, cannot be zero count 768.000000 mean 69.105469 std 19.355807 min 0.000000 25% 62.000000 50% 72.000000 75% 80.000000 max 122.000000 Name: Blood Pressure, dtype: float64
Skin Thickness	Self-explanatory, cannot be zero count 768.000000 mean 20.536458 std 15.952218 min 0.000000 25% 0.000000 50% 23.000000 75% 32.000000 max 99.000000 Name: Skin Thickness, dtype: float64
Insulin	Self-explanatory, cannot be zero count 768.000000 mean 79.799479 std 115.244002 min 0.000000 25% 0.000000 50% 30.500000 75% 127.250000 max 846.000000 Name: Insulin, dtype: float64
BMI	Self-explanatory, cannot be zero count 768.000000 mean 31.992578 std 7.884160 min 0.000000 25% 27.300000 50% 32.000000 75% 36.600000 max 67.100000 Name: BMI, dtype: float64
DiabetesPedigreeFunction	A function which scores likelihood of diabetes based on family history

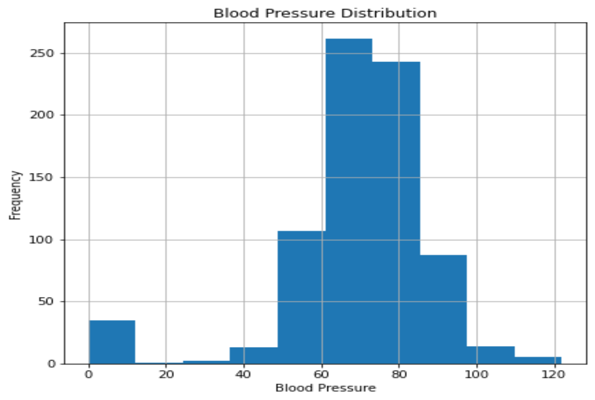
	count 768.000000 mean 0.471876 std 0.331329 min 0.078000 25% 0.243750 50% 0.372500 75% 0.626250 max 2.420000 Name: DiabetesPedigreeFunction, dtype: float64
Age	Self-explanatory count 768.000000 mean 33.240885 std 11.760232 min 21.000000 25% 24.000000 50% 29.000000 75% 41.000000 max 81.000000 Name: Age, dtype: float64
Class	0 – The person doesn’t have diabetes 1 – The person has diabetes Categorical Type

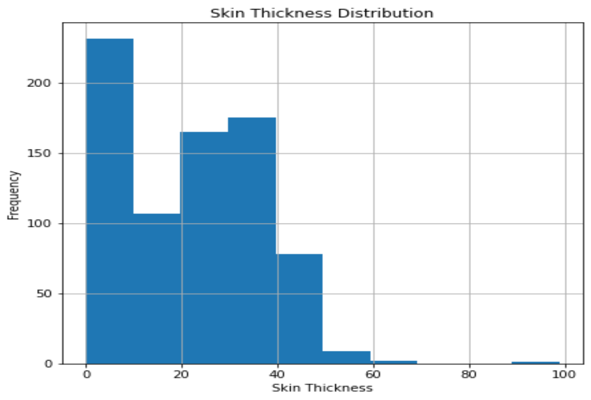
Data Visualization

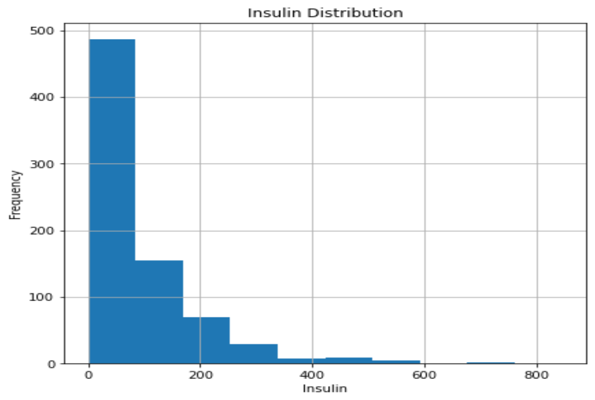
Histograms

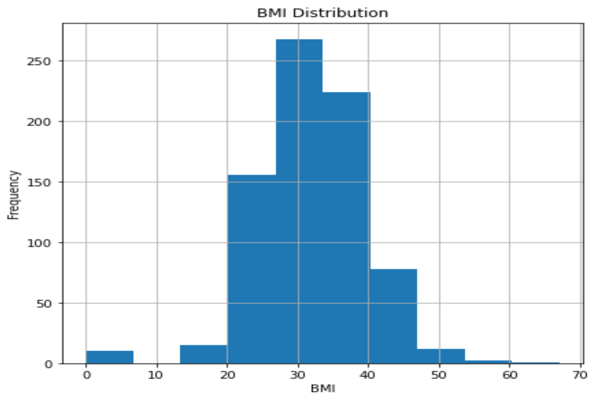


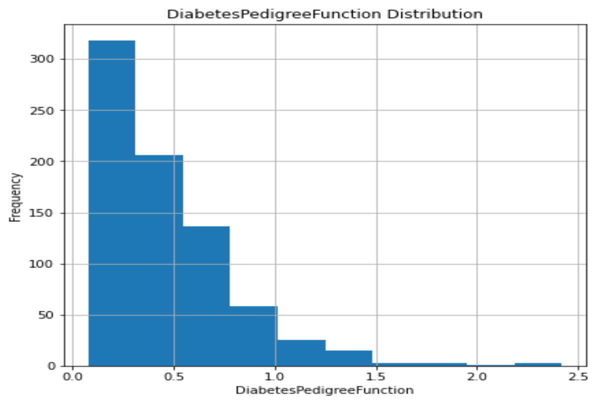


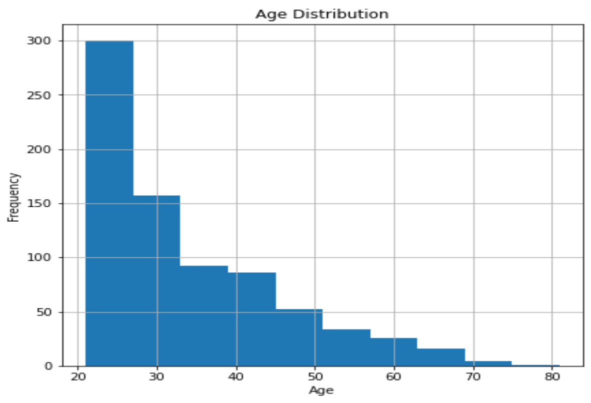




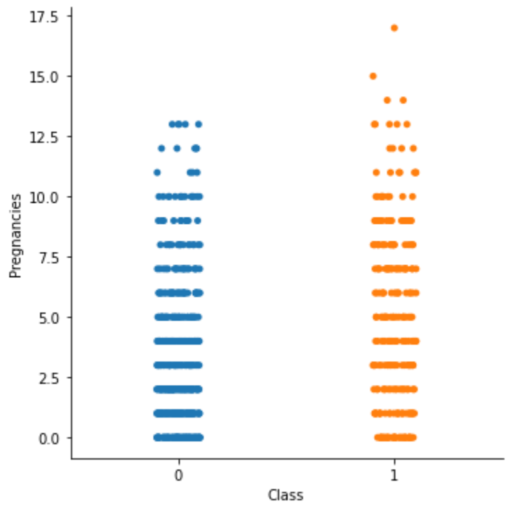


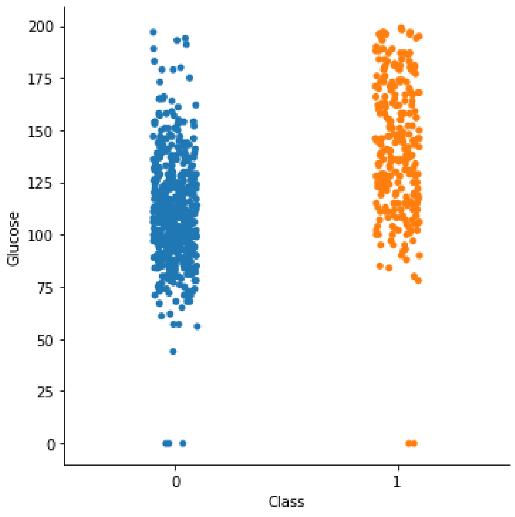


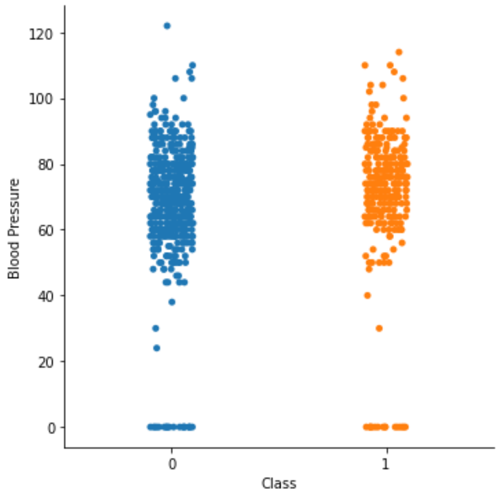


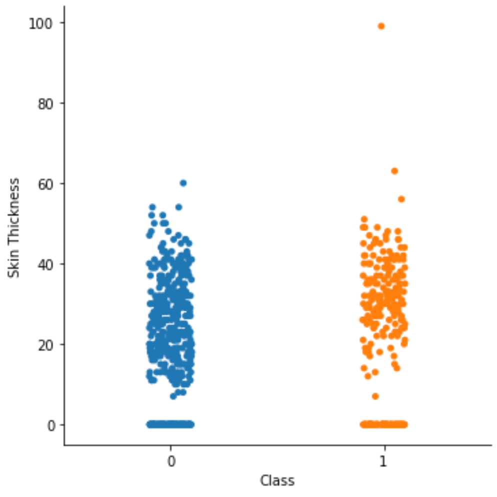


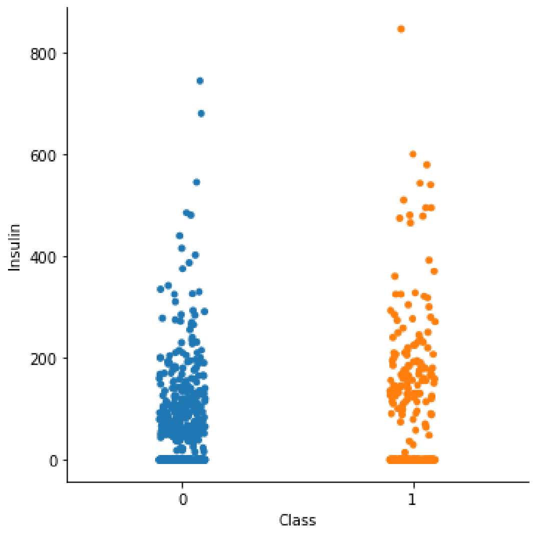
a. **Catplots**

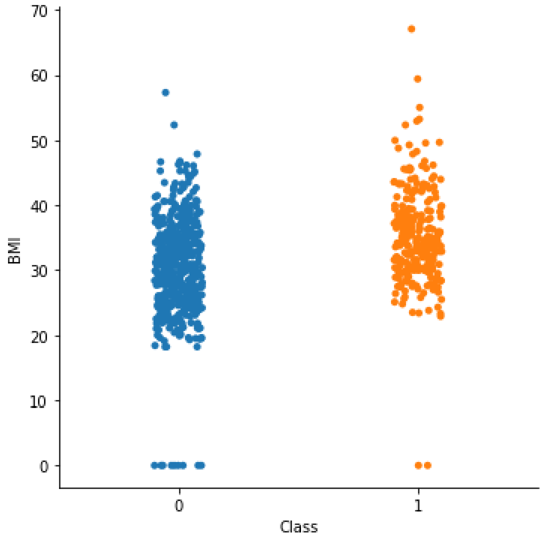

Pregnancies
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
1
Class

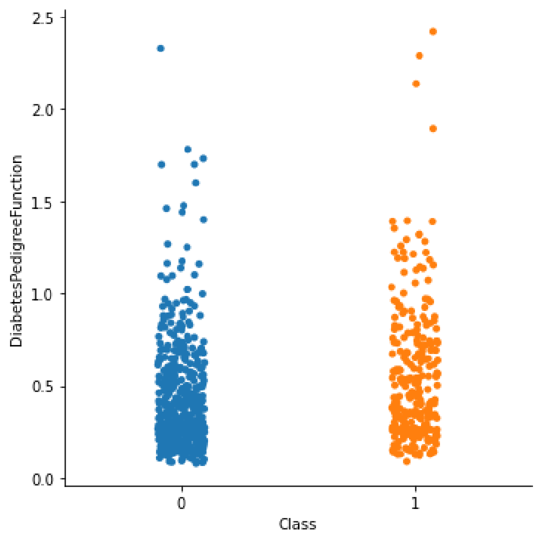

Glucose
200
175
150
125
100
75
50
25
0
0
1
Class

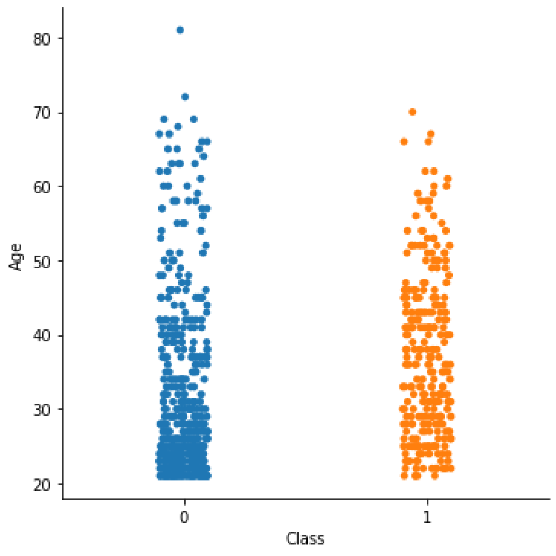

Blood Pressure
120
100
80
60
40
20
0
0
1
Class


Skin Thickness
100
80
60
40
20
0
0
1
Class

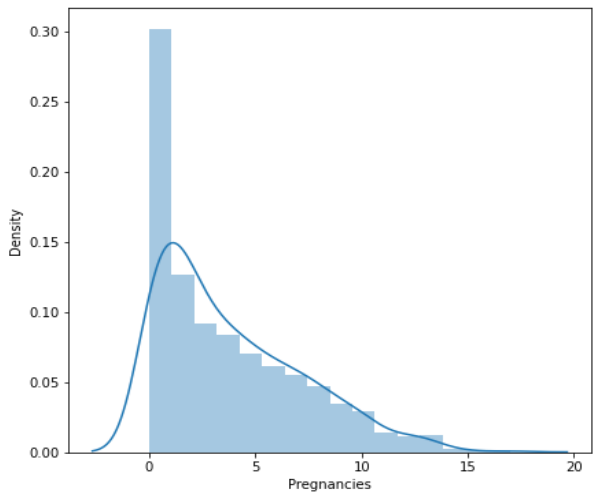

Insulin
800
600
400
200
0
0
1
Class

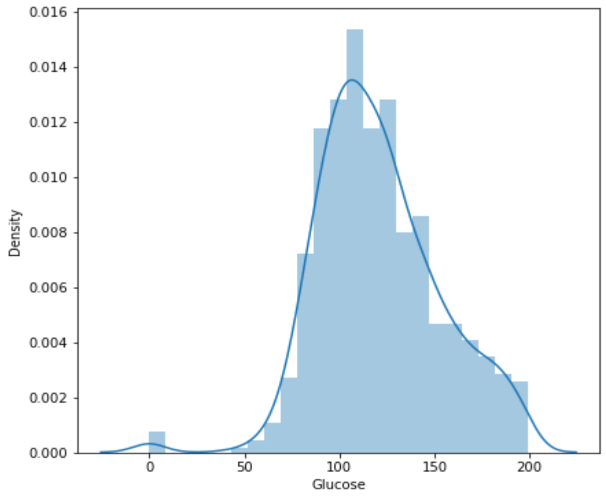

BMI
70
60
50
40
30
20
10
0
0
1
Class

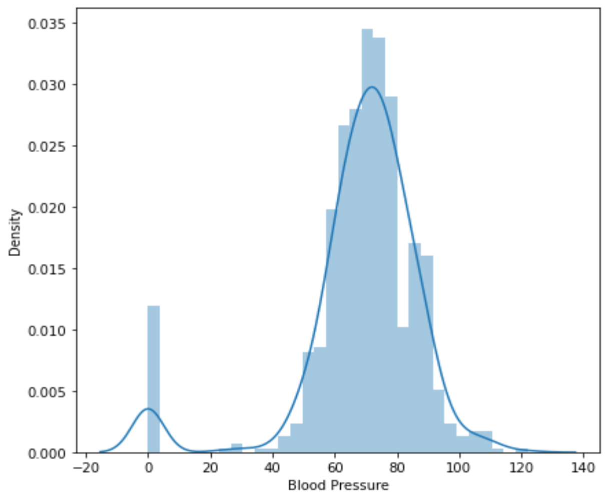


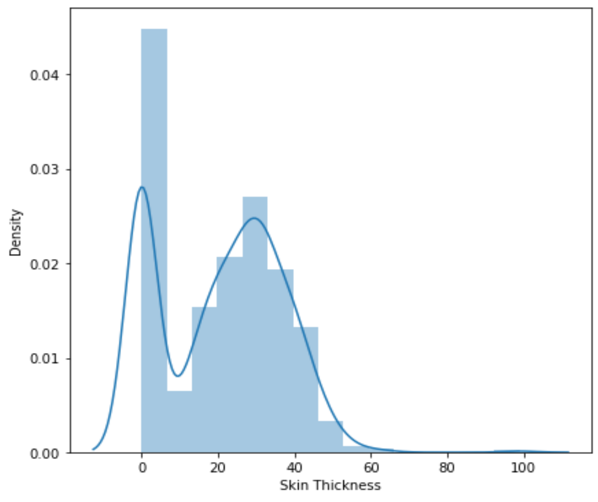


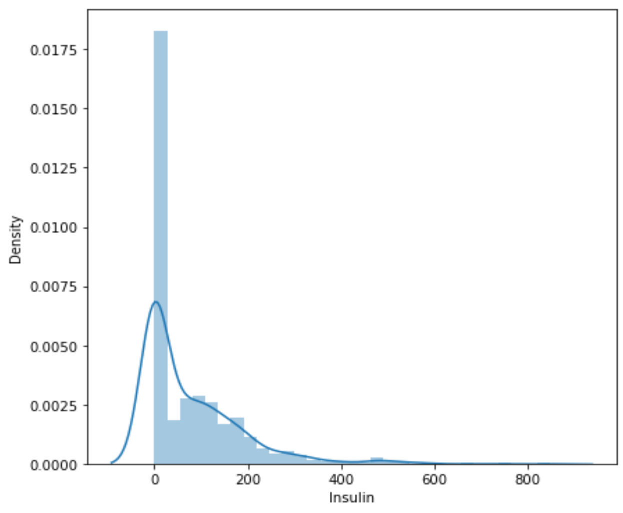
Data Distribution

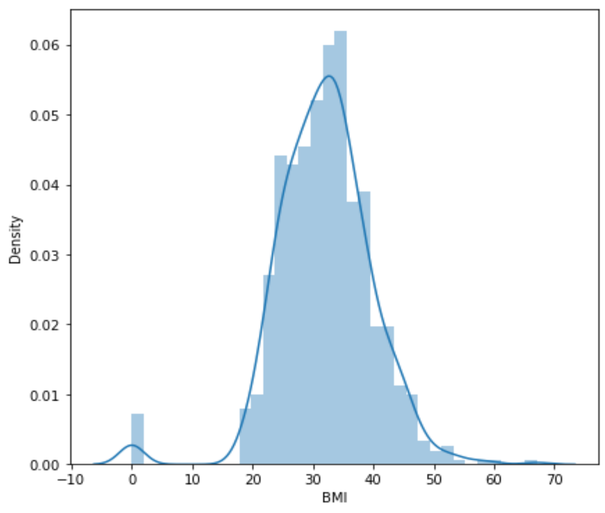


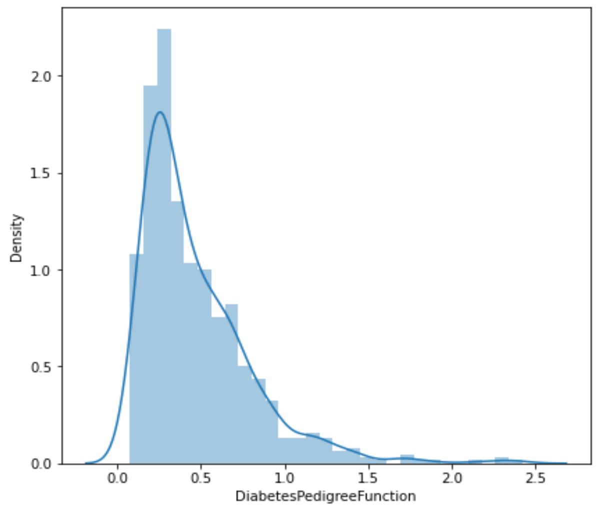


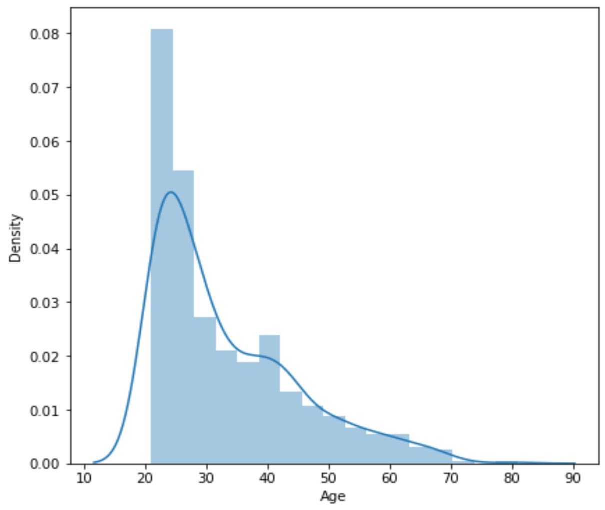




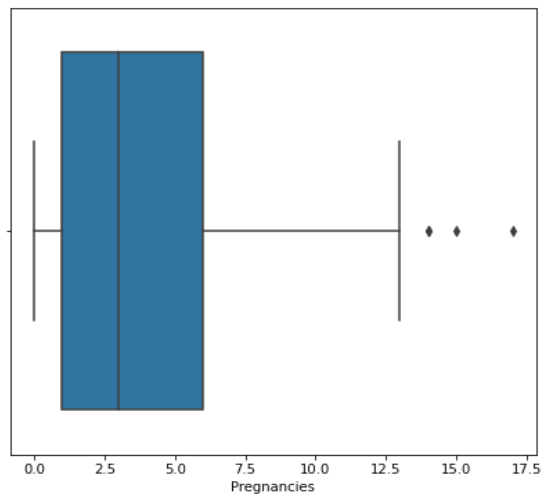

Density
Insulin
0.0000
0.0025
0.0050
0.0075
0.0100
0.0125
0.0150
0.0175
0
200
400
600
800

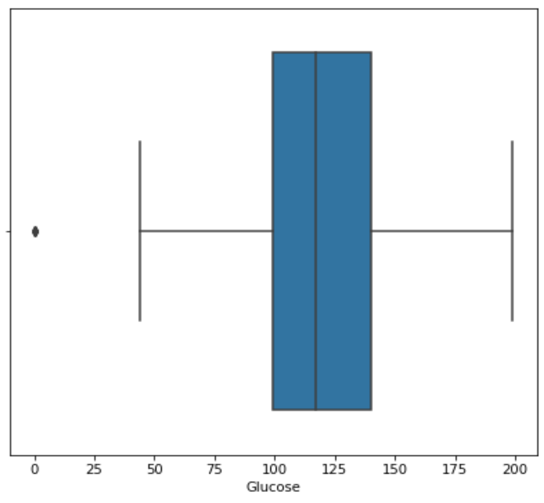

Density
BMI
0.00
0.01
0.02
0.03
0.04
0.05
0.06
−10
0
10
20
30
40
50
60
70

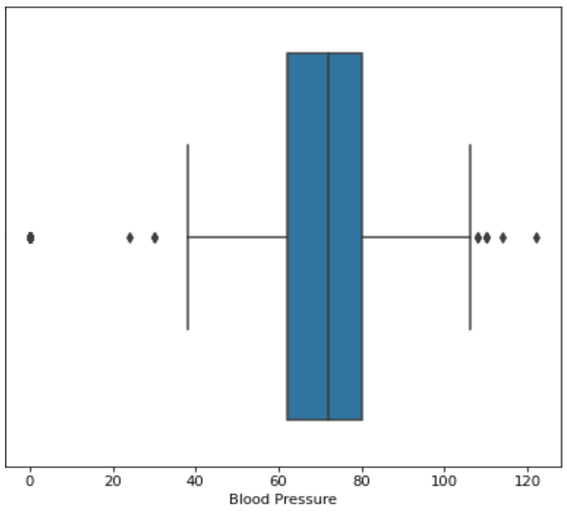

Density
DiabetesPedigreeFunction
0.0
0.5
1.0
1.5
2.0
2.5

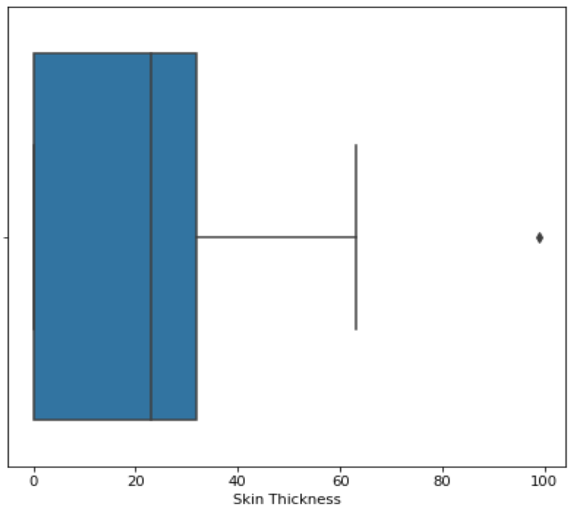

Density
Age
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
10
20
30
40
50
60
70
80
90

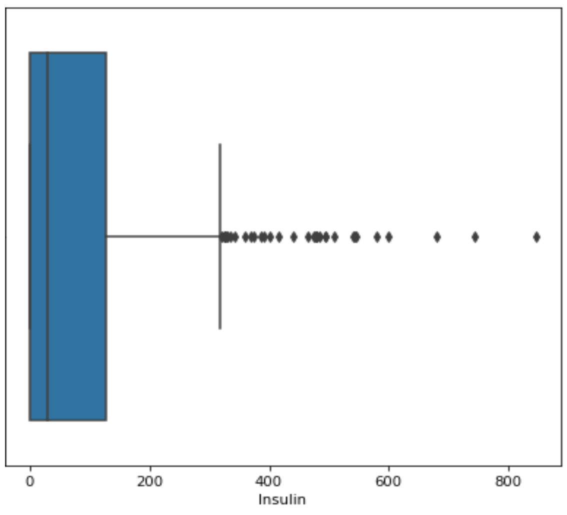
b. Boxplots

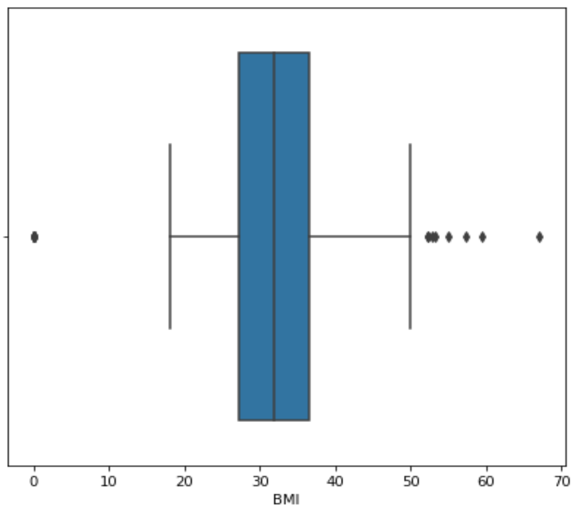

Pregnancies
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

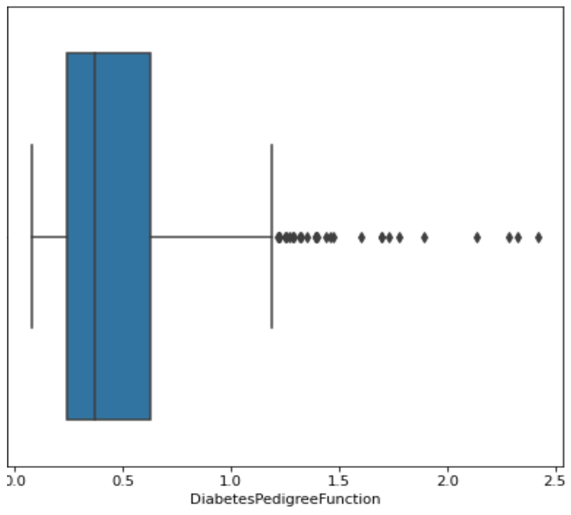

Glucose
0
25
50
75
100
125
150
175
200

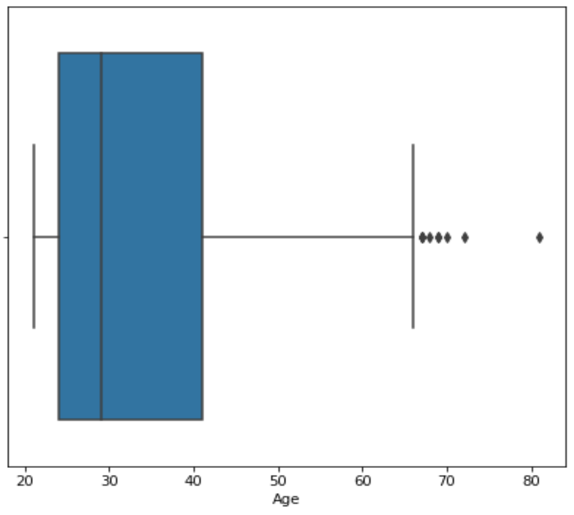

0
20
40
60
80
100
120
Blood Pressure


0
20
40
60
80
100
Skin Thickness


0
200
400
600
800
Insulin

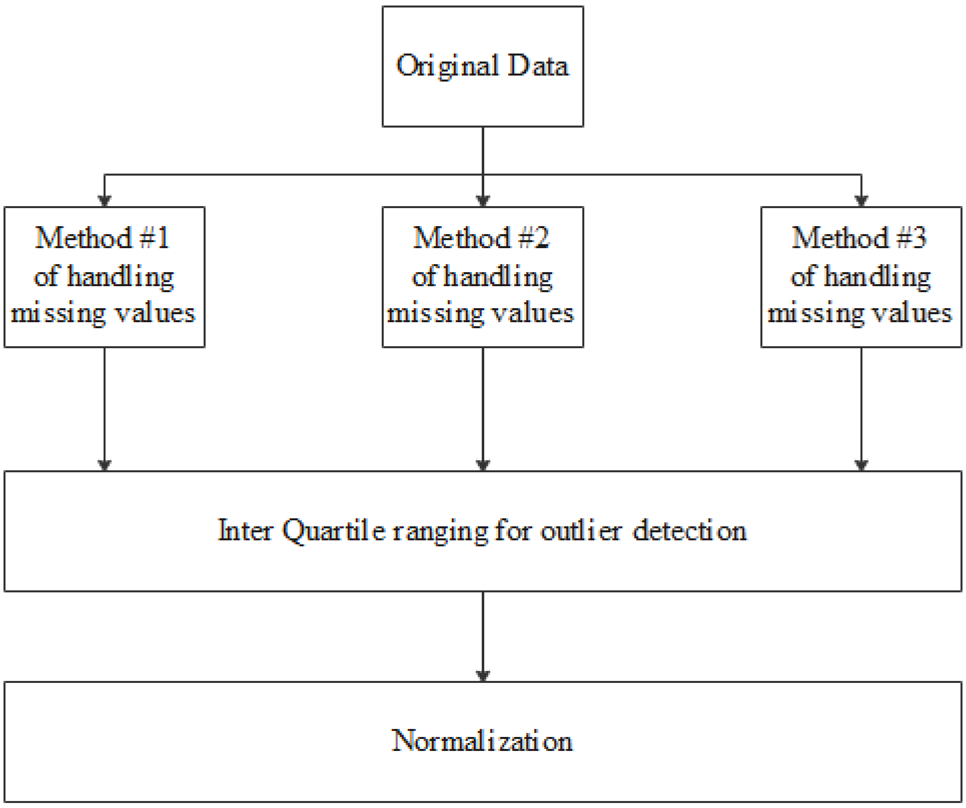

0
10
20
30
40
50
60
70
BMI


0.0
0.5
1.0
1.5
2.0
2.5
DiabetesPedigreeFunction


20
30
40
50
60
70
80
Age

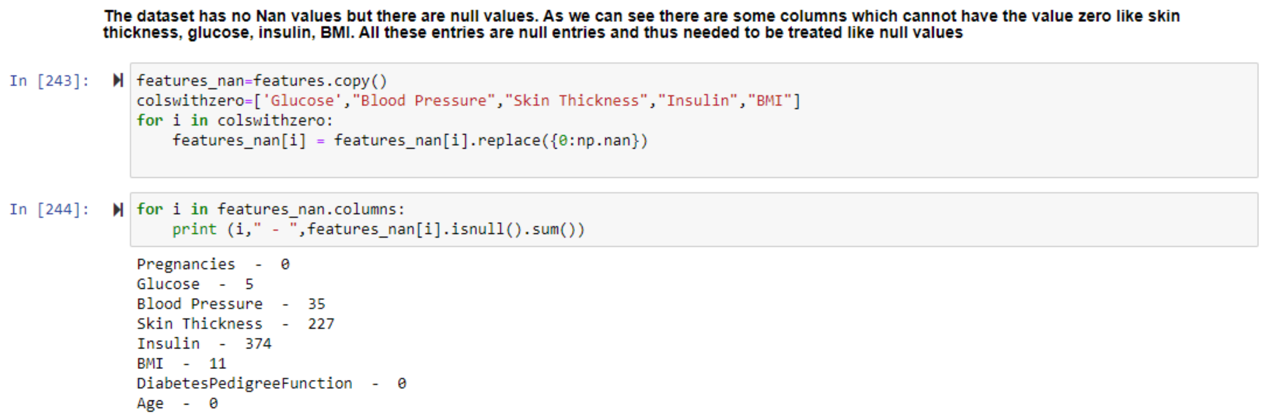
Data Pre-processing

The following diagram showcases the pre-processing pipeline used for this project.


Original Data
Method #1 of handling missing values
Method #2 of handling missing values
Method #3 of handling missing values
Inter Quartile ranging for outlier detection
Normalization

Handling null values

From the data visualization stage, we see many records having zero values in Glucose, Blood Pressure, Skin Thickness, Insulin and BMI which is not possible thus, these are null values. So, to make the handling of these missing values comparatively easier, we convert the zero values to null values. Then we see the number of missing values in each attribute.


The dataset has no Nan values but there are null values. As we can see there are some columns which cannot have the value zero like skin thickness, glucose, insulin, BMI. All these entries are null entries and thus needed to be treated like null values

```
In [243]:  features_nan=features.copy()
           colswithzero=['Glucose',"Blood Pressure","Skin Thickness","Insulin","BMI"]
           for i in colswithzero:
               features_nan[i] = features_nan[i].replace({0:np.nan})
```

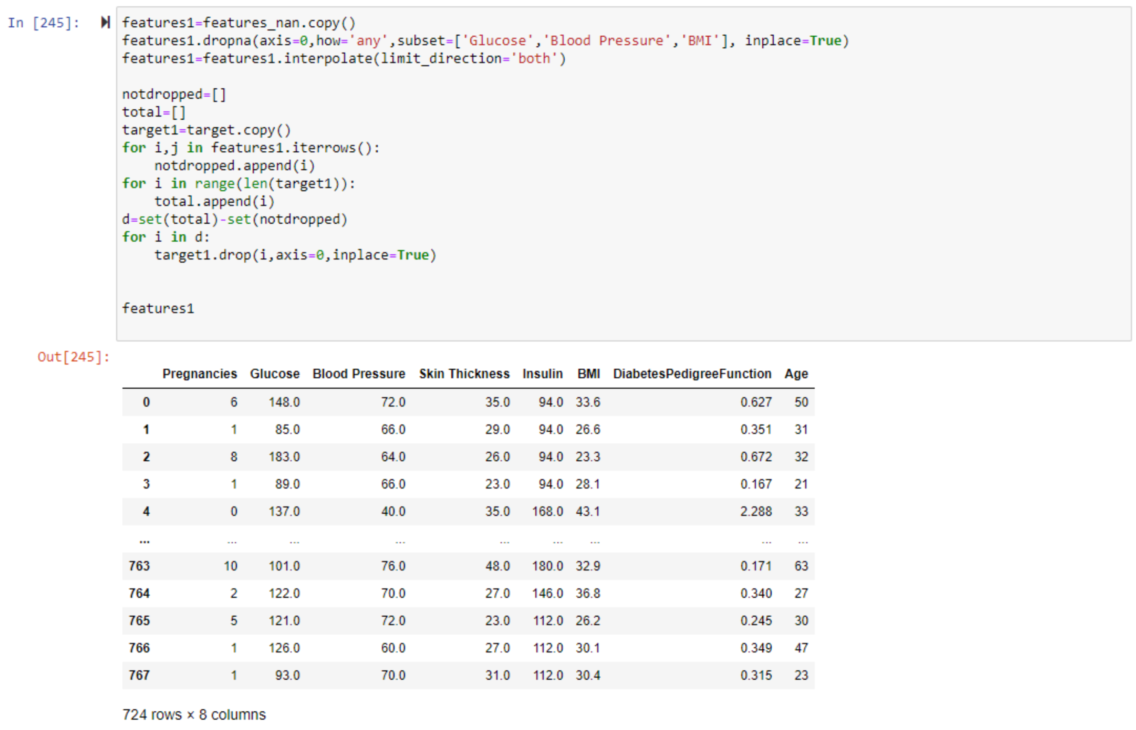
```
In [244]:  for i in features_nan.columns:
               print (i," - ",features_nan[i].isnull().sum())

           Pregnancies  -  0
           Glucose  -  5
           Blood Pressure  -  35
           Skin Thickness  -  227
           Insulin  -  374
           BMI  -  11
           DiabetesPedigreeFunction  -  0
           Age  -  0
```

The number of missing values are-

COLUMN NAME	Number of missing values
Pregnancies	0
Glucose	5
Blood Pressure	35
Skin Thickness	227
Insulin	374
BMI	11
DiabetesPedigreeFunction	0
Age	0

For the sake of this project, we handle the missing values in 3 different ways to compare later. The three approaches are described below

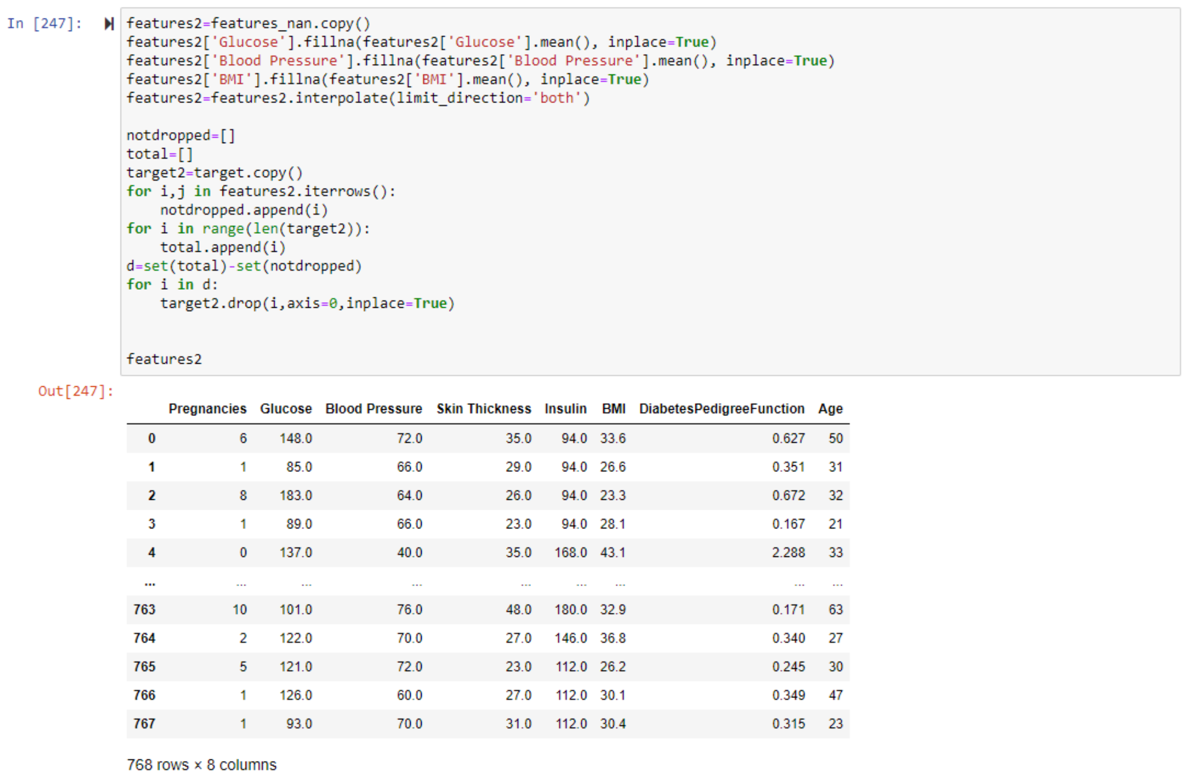
1. We drop all records with missing Glucose, Blood Pressure and BMI and then we use interpolate() function for substituting the missing values for Skin Thickness and Insulin. After this, we are left with 724 records (initially there were 768 records). The code and the output are showcased in the following image.



```
In [245]:
features1=features_nan.copy()
features1.dropna(axis=0,how='any',subset=['Glucose','Blood Pressure','BMI'], inplace=True)
features1=features1.interpolate(limit_direction='both')

notdropped=[]
total=[]
target1=target.copy()
for i,j in features1.iterrows():
    notdropped.append(i)
for i in range(len(target1)):
    total.append(i)
d=set(total)-set(notdropped)
for i in d:
    target1.drop(i,axis=0,inplace=True)


features1
```

```
Out[245]:
     Pregnancies  Glucose  Blood Pressure  Skin Thickness  Insulin   BMI  DiabetesPedigreeFunction  Age
  0            6    148.0            72.0            35.0     94.0  33.6                     0.627   50
  1            1     85.0            66.0            29.0     94.0  26.6                     0.351   31
  2            8    183.0            64.0            26.0     94.0  23.3                     0.672   32
  3            1     89.0            66.0            23.0     94.0  28.1                     0.167   21
  4            0    137.0            40.0            35.0    168.0  43.1                     2.288   33
...          ...      ...             ...             ...      ...   ...                       ...  ...
763           10    101.0            76.0            48.0    180.0  32.9                     0.171   63
764            2    122.0            70.0            27.0    146.0  36.8                     0.340   27
765            5    121.0            72.0            23.0    112.0  26.2                     0.245   30
766            1    126.0            60.0            27.0    112.0  30.1                     0.349   47
767            1     93.0            70.0            31.0    112.0  30.4                     0.315   23

724 rows × 8 columns
```

2. In the second approach, we substitute the missing values of glucose, Blood Pressure and BMI with their mean values and then interpolate the missing values for Skin Thickness and Insulin. Here, no records are dropped. The code and the output are showcased in the following image.



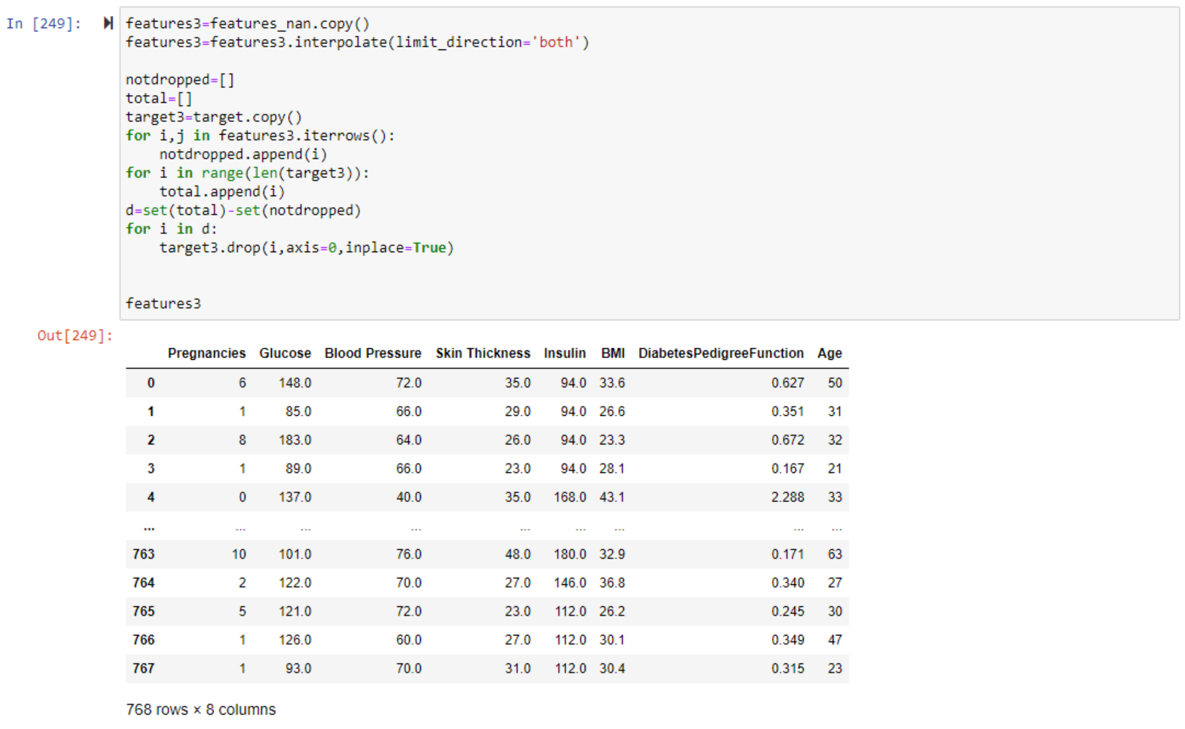
In [247]:
```
features2=features_nan.copy()
features2['Glucose'].fillna(features2['Glucose'].mean(), inplace=True)
features2['Blood Pressure'].fillna(features2['Blood Pressure'].mean(), inplace=True)
features2['BMI'].fillna(features2['BMI'].mean(), inplace=True)
features2=features2.interpolate(limit_direction='both')

notdropped=[]
total=[]
target2=target.copy()
for i,j in features2.iterrows():
    notdropped.append(i)
for i in range(len(target2)):
    total.append(i)
d=set(total)-set(notdropped)
for i in d:
    target2.drop(i,axis=0,inplace=True)


features2
```

Out[247]:

	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.0	94.0	33.6	0.627	50
1	1	85.0	66.0	29.0	94.0	26.6	0.351	31
2	8	183.0	64.0	26.0	94.0	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101.0	76.0	48.0	180.0	32.9	0.171	63
764	2	122.0	70.0	27.0	146.0	36.8	0.340	27
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30
766	1	126.0	60.0	27.0	112.0	30.1	0.349	47
767	1	93.0	70.0	31.0	112.0	30.4	0.315	23

768 rows × 8 columns

3. In the third approach, we interpolate all the values. Here, no records are dropped. The code and the output are showcased in the following image.



In [249]:
```
features3=features_nan.copy()
features3=features3.interpolate(limit_direction='both')

notdropped=[]
total=[]
target3=target.copy()
for i,j in features3.iterrows():
    notdropped.append(i)
for i in range(len(target3)):
    total.append(i)
d=set(total)-set(notdropped)
for i in d:
    target3.drop(i,axis=0,inplace=True)


features3
```

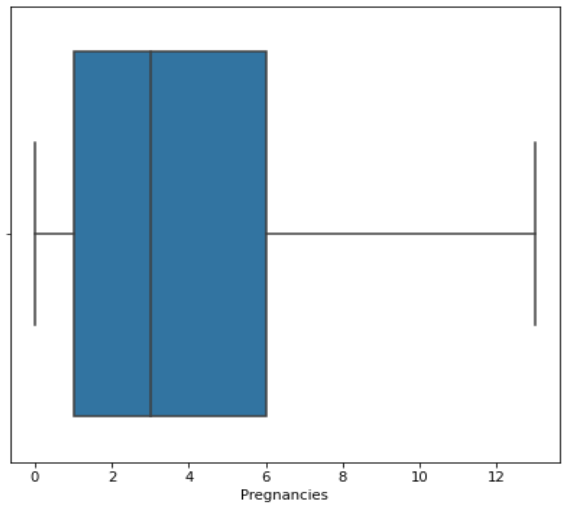
Out[249]:

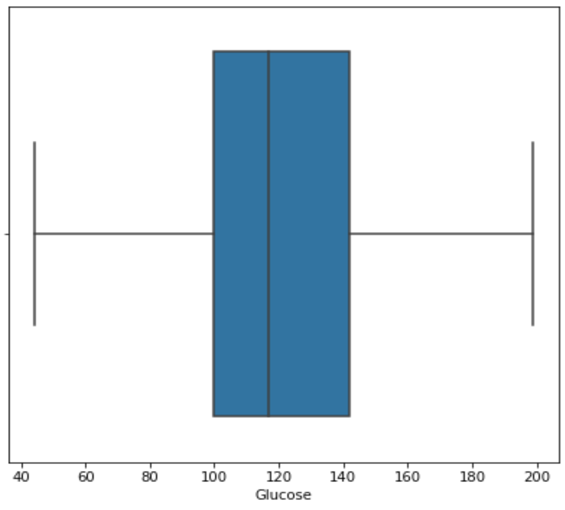
	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148.0	72.0	35.0	94.0	33.6	0.627	50
1	1	85.0	66.0	29.0	94.0	26.6	0.351	31
2	8	183.0	64.0	26.0	94.0	23.3	0.672	32
3	1	89.0	66.0	23.0	94.0	28.1	0.167	21
4	0	137.0	40.0	35.0	168.0	43.1	2.288	33
...	...	...	...	...	...	...	...	...
763	10	101.0	76.0	48.0	180.0	32.9	0.171	63
764	2	122.0	70.0	27.0	146.0	36.8	0.340	27
765	5	121.0	72.0	23.0	112.0	26.2	0.245	30
766	1	126.0	60.0	27.0	112.0	30.1	0.349	47
767	1	93.0	70.0	31.0	112.0	30.4	0.315	23

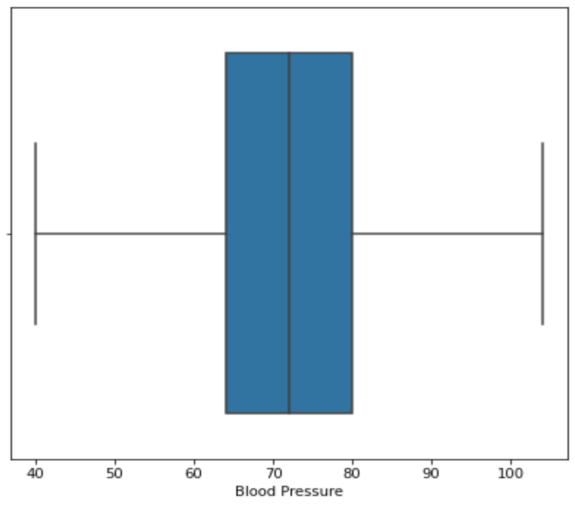
768 rows × 8 columns

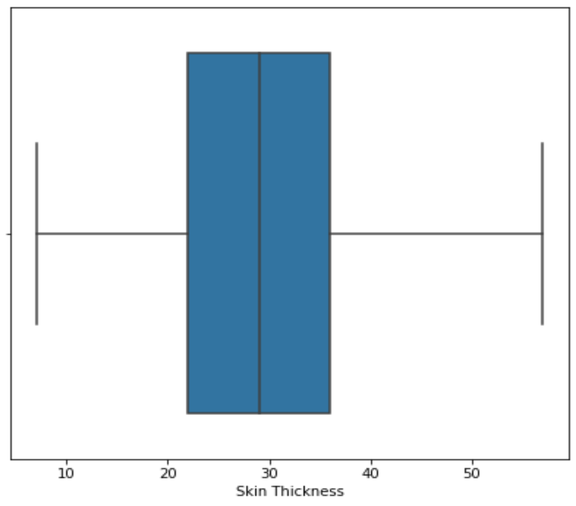
Outlier detection and removal

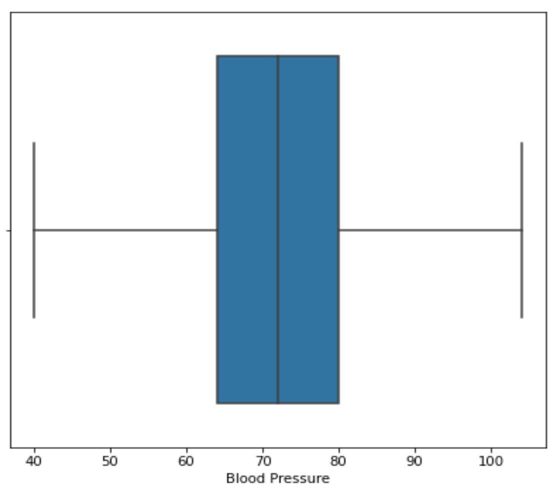
To remove the outliers, we use Interquartile Range (IQR) proximity rule. Let the 25 percentile and 75 percentile be represented by q25 and q75 and iqr=q75 – q25. The data points that fall below q25 – 1.5*iqr or above q75 + 1.5*iqr are classified as outliers. After the outlier removal, the boxplots of the data were the following (only the boxplot of the 1st approach).

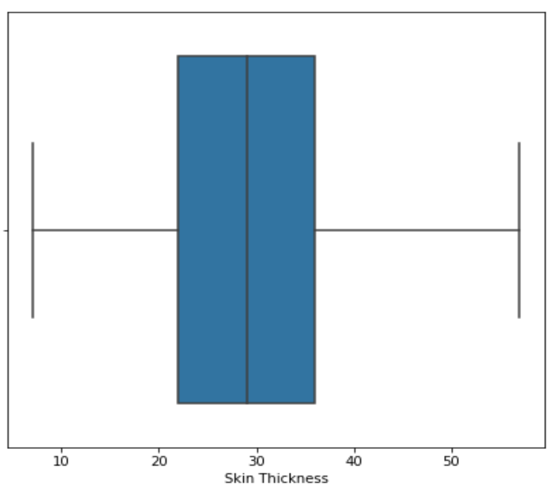


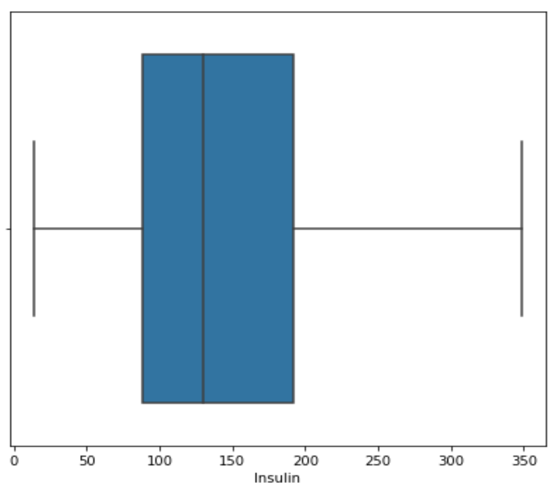


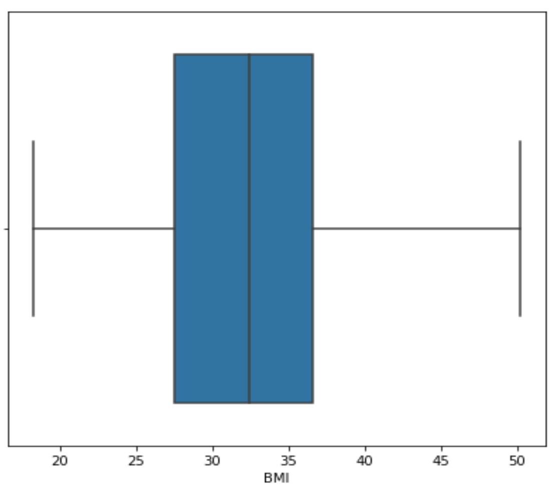


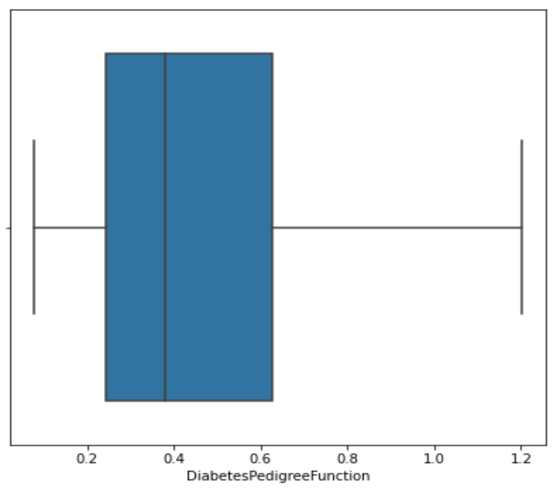


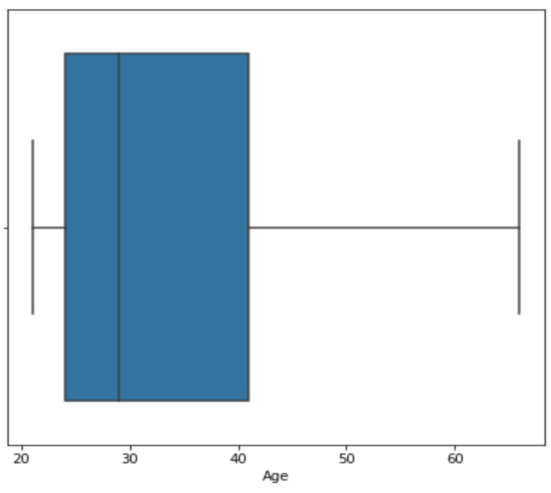








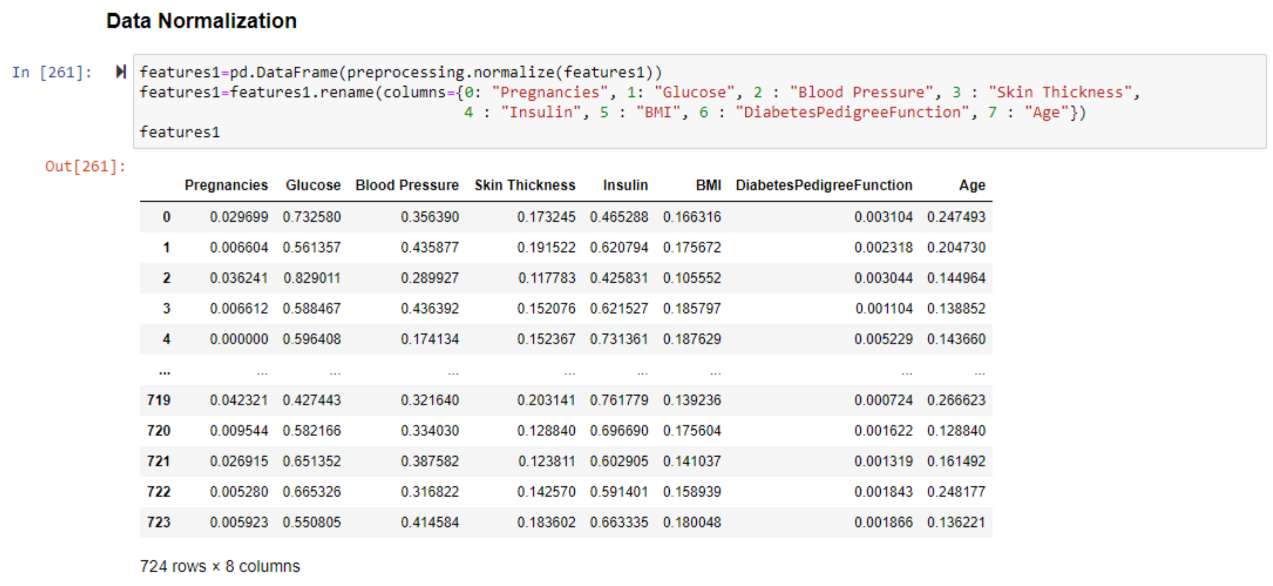




As we can see, there are no outliers now.

Data Normalization

We normalized the 3 data frames (the data frames that were made after applying 3 different approaches to handle missing data). The normalization of the 1st data frame is shown here.



Data Normalization

```
In [261]:  features1=pd.DataFrame(preprocessing.normalize(features1))
           features1=features1.rename(columns={0: "Pregnancies", 1: "Glucose", 2 : "Blood Pressure", 3 : "Skin Thickness",
                                               4 : "Insulin", 5 : "BMI", 6 : "DiabetesPedigreeFunction", 7 : "Age"})
           features1
Out[261]:
```

	Pregnancies	Glucose	Blood Pressure	Skin Thickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	0.029699	0.732580	0.356390	0.173245	0.465288	0.166316	0.003104	0.247493
1	0.006604	0.561357	0.435877	0.191522	0.620794	0.175672	0.002318	0.204730
2	0.036241	0.829011	0.289927	0.117783	0.425831	0.105552	0.003044	0.144964
3	0.006612	0.588467	0.436392	0.152076	0.621527	0.185797	0.001104	0.138852
4	0.000000	0.596408	0.174134	0.152367	0.731361	0.187629	0.005229	0.143660
...	...	...	...	...	...	...	...	...
719	0.042321	0.427443	0.321640	0.203141	0.761779	0.139236	0.000724	0.266623
720	0.009544	0.582166	0.334030	0.128840	0.696690	0.175604	0.001622	0.128840
721	0.026915	0.651352	0.387582	0.123811	0.602905	0.141037	0.001319	0.161492
722	0.005280	0.665326	0.316822	0.142570	0.591401	0.158939	0.001843	0.248177
723	0.005923	0.550805	0.414584	0.183602	0.663335	0.180048	0.001866	0.136221

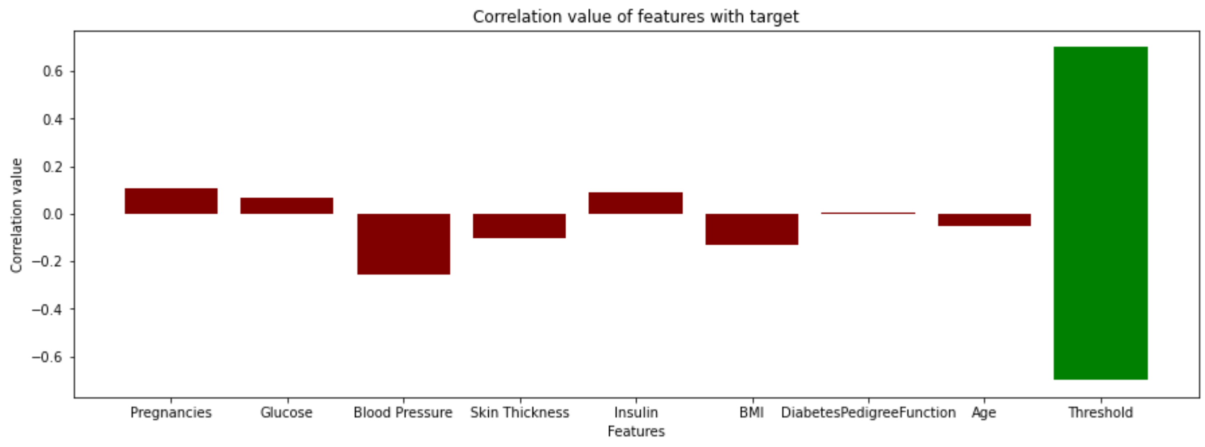
724 rows × 8 columns

Feature Selection

We carry on the feature selection by

1. Finding the correlation of the target with all the attributes. If any attribute has an absolute value of correlation of more than (or equal to) 0.7, we drop that attribute. A visualization of the correlation is as follows (only shown for the first dataset).

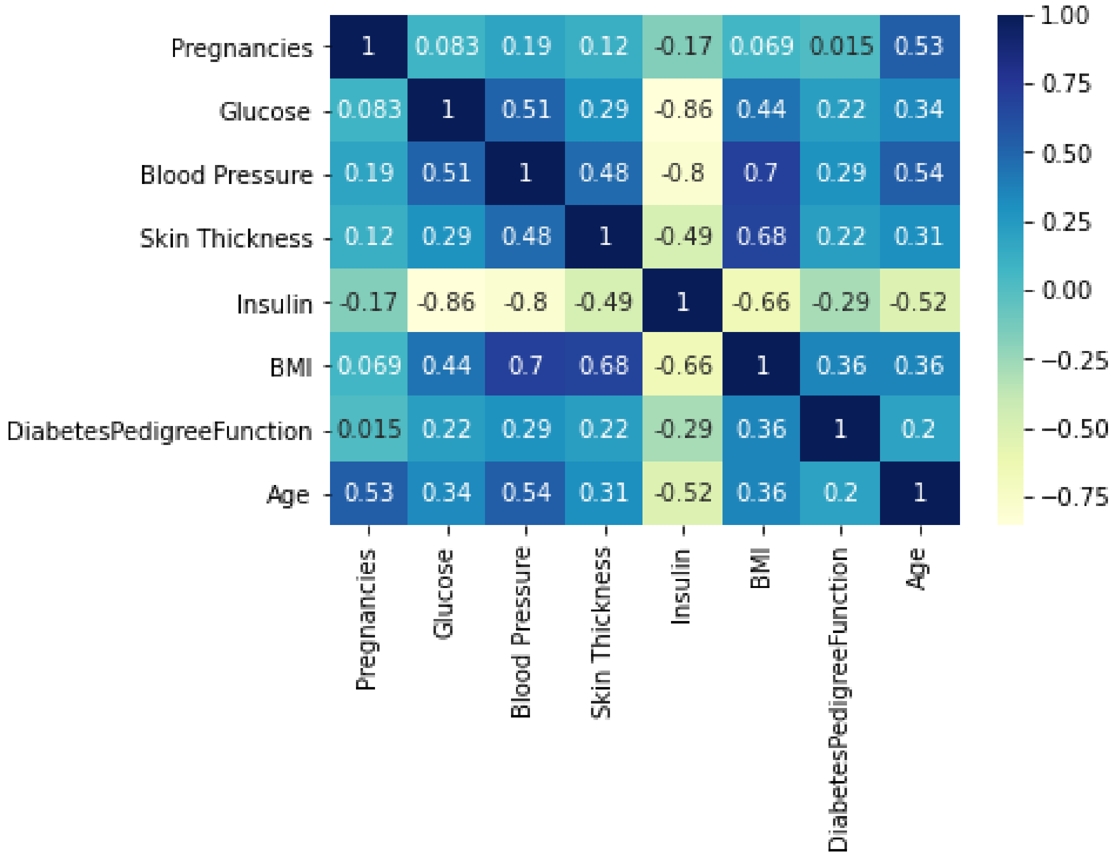
   1st data frame



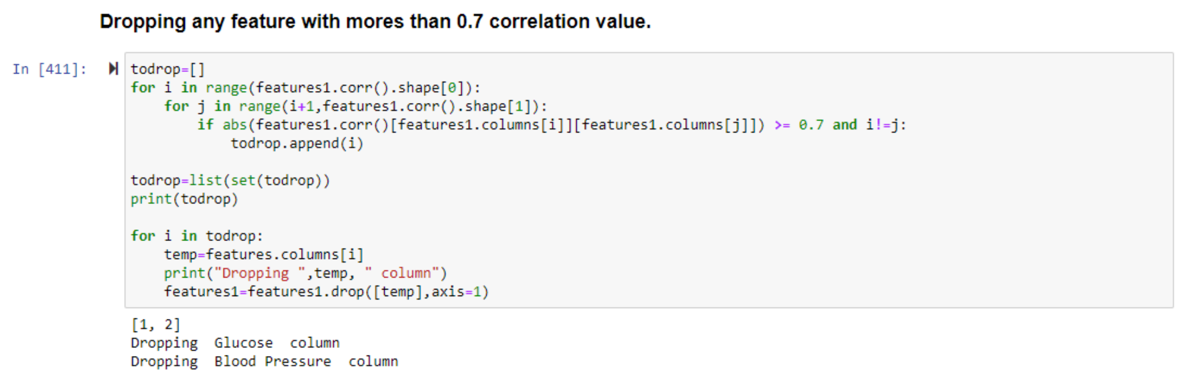
   As none of the features were strongly correlated to the target values, none of them were dropped.

2. The second step was to find the correlation of features with each other. If any feature was strongly correlated with another feature (correlation > 0.7), it was dropped. A

heatmap for the attribute’s correlation with each other for the 1st data frame id shown below.



As we can see from the heatmap, Glucose, Blood pressure and Insulin have high absolute values of correlation within each other.



Dropping any feature with mores than 0.7 correlation value.

```
In [411]:
todrop=[]
for i in range(features1.corr().shape[0]):
    for j in range(i+1,features1.corr().shape[1]):
        if abs(features1.corr()[features1.columns[i]][features1.columns[j]]) >= 0.7 and i!=j:
            todrop.append(i)

todrop=list(set(todrop))
print(todrop)

for i in todrop:
    temp=features.columns[i]
    print("Dropping ",temp, " column")
    features1=features1.drop([temp],axis=1)
```

```
[1, 2]
Dropping  Glucose  column
Dropping  Blood Pressure  column
```

Classification Models

Since we have 3 different pre-processed data frames, we separate them into training and testing datasets. We take 75% of the data in each of the data frames as training data and the remaining as the testing data.

Here is the list of all the classification models compared

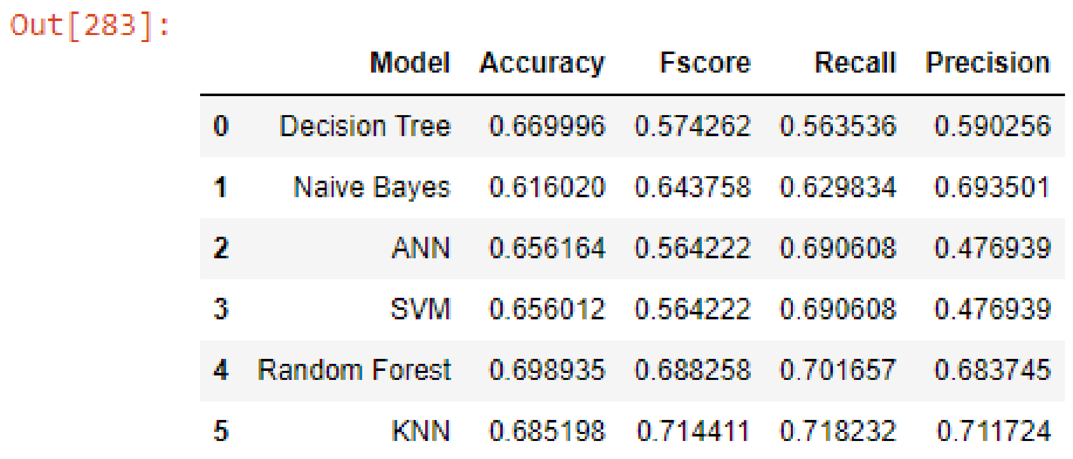
1. Decision Tree
2. Naive Bayes
3. ANN
4. SVM
5. Random Forest
6. KNN

We will be comparing these models on these parameters

1. Accuracy (k fold cross validation, k=10)
2. Fscore
3. Recall
4. Precision.

Results

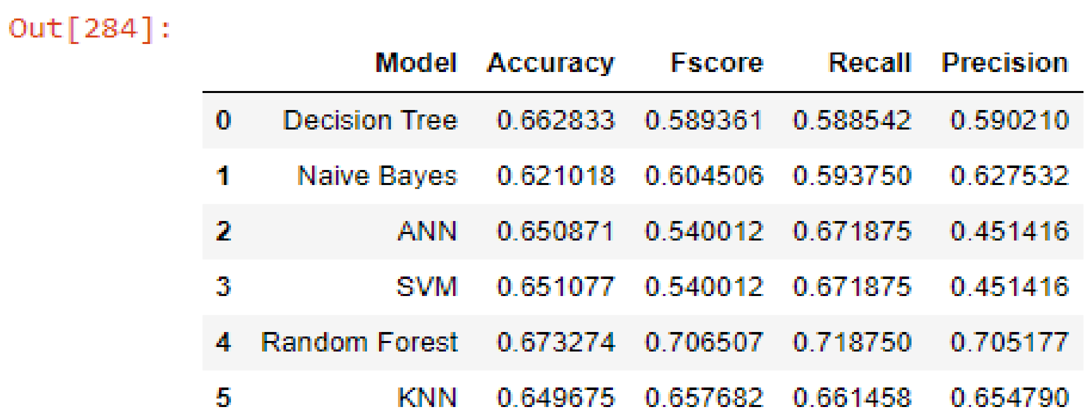
Data frame #1



Out[283]:

	Model	Accuracy	Fscore	Recall	Precision
0	Decision Tree	0.669996	0.574262	0.563536	0.590256
1	Naive Bayes	0.616020	0.643758	0.629834	0.693501
2	ANN	0.656164	0.564222	0.690608	0.476939
3	SVM	0.656012	0.564222	0.690608	0.476939
4	Random Forest	0.698935	0.688258	0.701657	0.683745
5	KNN	0.685198	0.714411	0.718232	0.711724

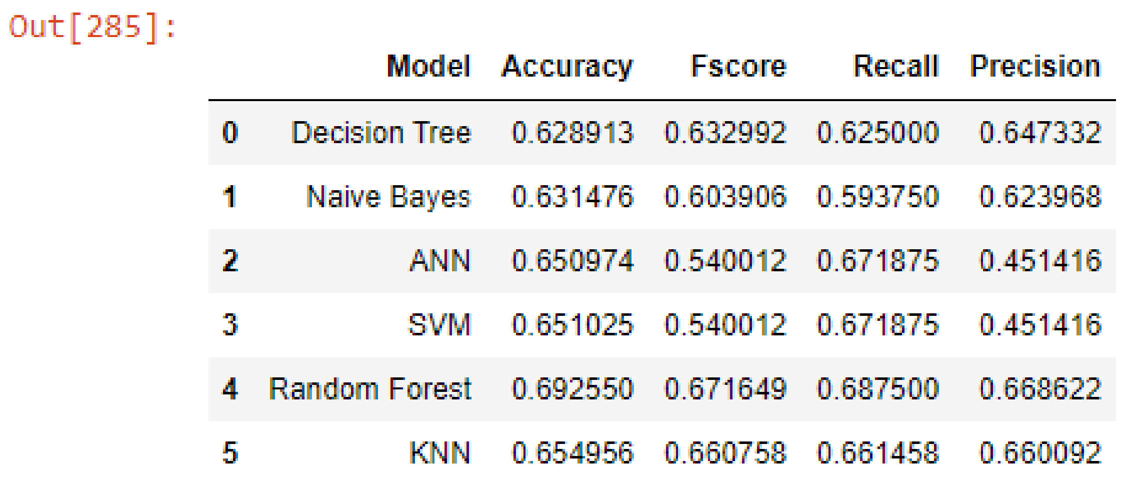
Data frame #2



Out[284]:

	Model	Accuracy	Fscore	Recall	Precision
0	Decision Tree	0.662833	0.589361	0.588542	0.590210
1	Naive Bayes	0.621018	0.604506	0.593750	0.627532
2	ANN	0.650871	0.540012	0.671875	0.451416
3	SVM	0.651077	0.540012	0.671875	0.451416
4	Random Forest	0.673274	0.706507	0.718750	0.705177
5	KNN	0.649675	0.657682	0.661458	0.654790

Data frame #3



Out[285]:

	Model	Accuracy	Fscore	Recall	Precision
0	Decision Tree	0.628913	0.632992	0.625000	0.647332
1	Naive Bayes	0.631476	0.603906	0.593750	0.623968
2	ANN	0.650974	0.540012	0.671875	0.451416
3	SVM	0.651025	0.540012	0.671875	0.451416
4	Random Forest	0.692550	0.671649	0.687500	0.668622
5	KNN	0.654956	0.660758	0.661458	0.660092

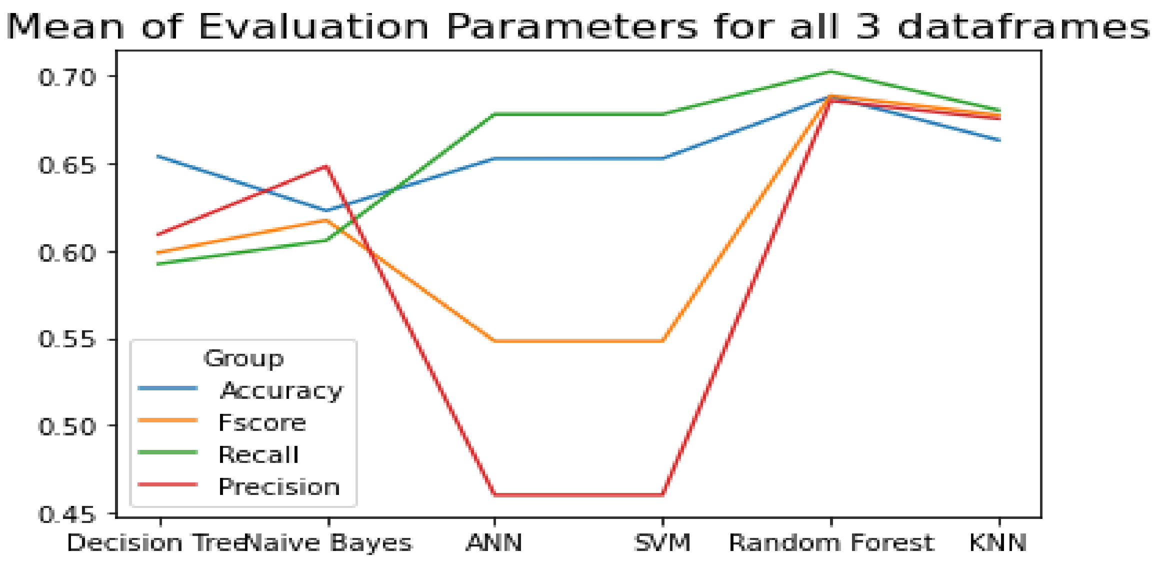
Evaluation of results

To properly evaluate the results, we use a number of graphs showcasing the accuracy of the models with the performance of the different preprocessed data frames. With these graphs, we answer these two fundamental questions

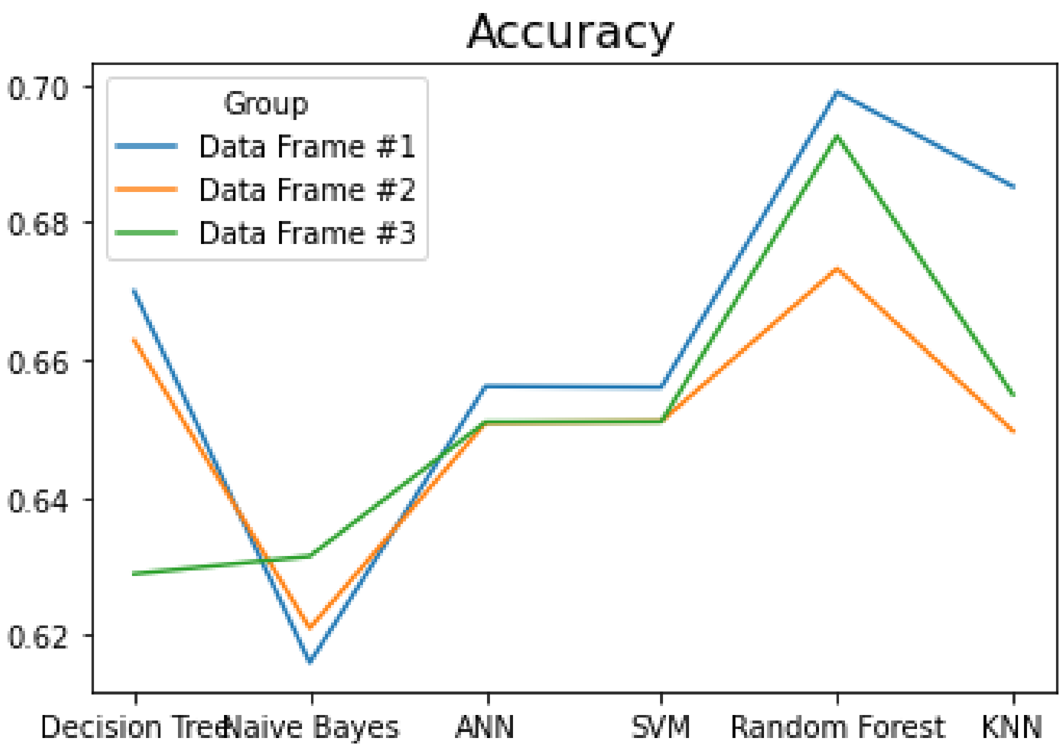
1. Which model performs the best across all data frames?
2. Which data frame performs the best across all models?

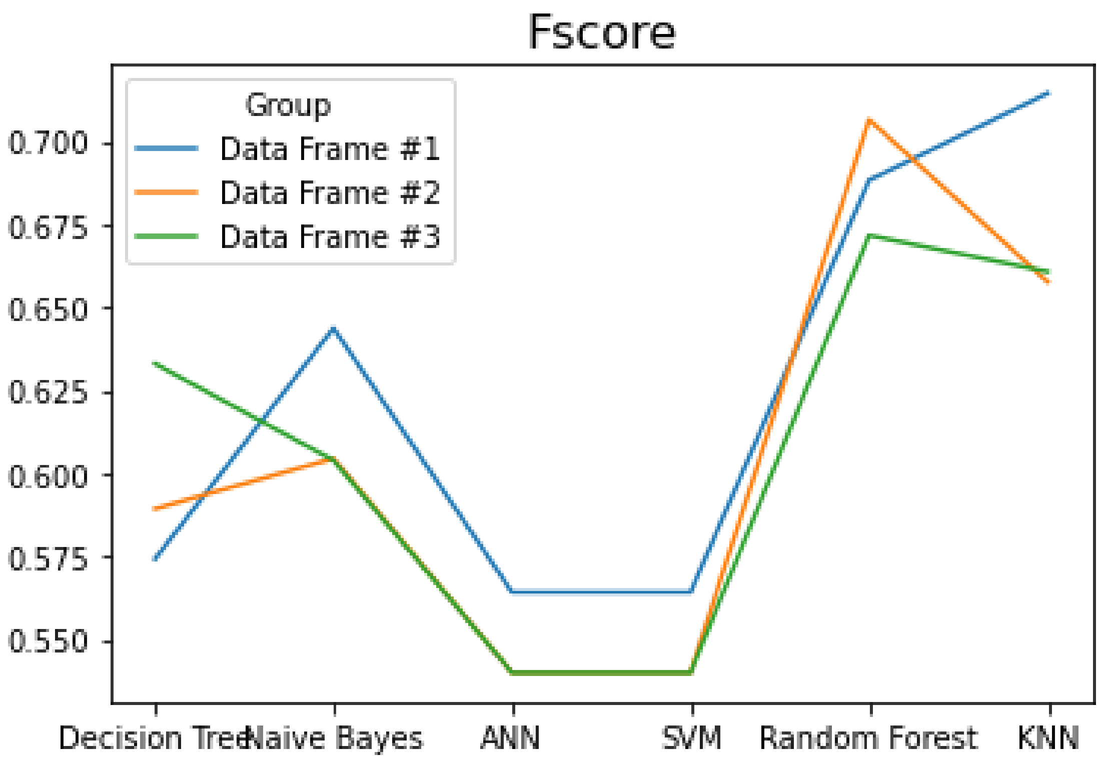
To answer the first question, we compare the models on the 4 evaluation parameters which are Accuracy, Fscore, Recall and Precision. When we compare the mean values (of the 3 data frames) of these 4 evaluation parameters, we begin to see a clear winner.

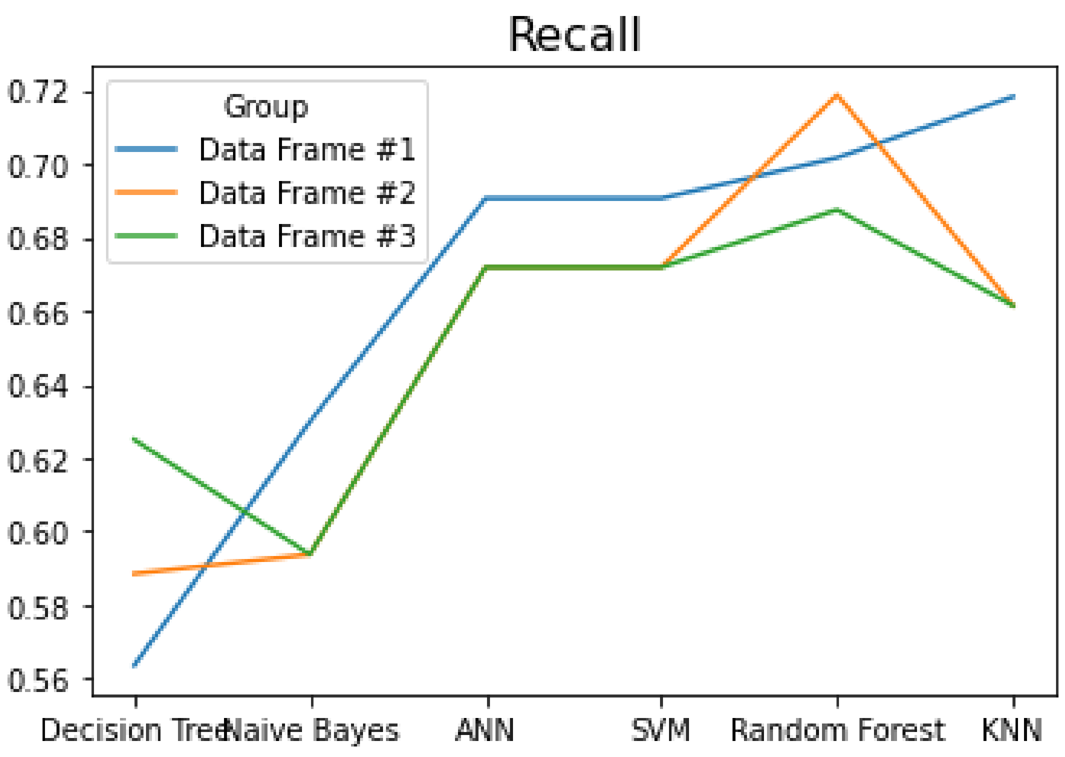
Random Forest is the most robust out of the 5 models we compared and gives a high value in all the 4 parameters across all the dataframes. Since we are evaluating a model which predicts the existence of diabetes or not, we choose **recall to be more important than precision** as getting false negatives is more costly than getting false positives. As we can see, random forest gives the highest recall value as well as the highest accuracy value.

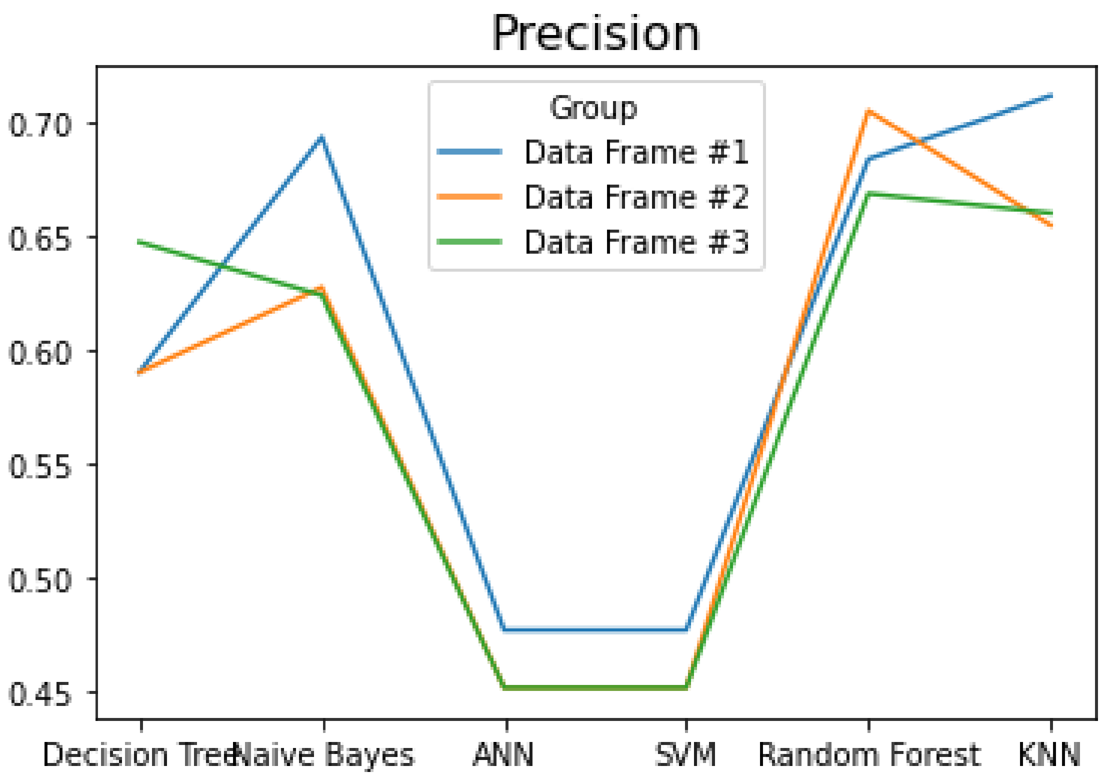


Now, we want to answer the second question. Which data frame performs the best across all models? As we have mentioned earlier, recall and accuracy are the most important metric for evaluating this problem. From the graphs showcased below, we see that data frame #1 has the highest accuracy but data frame #2 has the highest Fscore, Recall and Precision but the lowest accuracy. As accuracy is a more holistic measure of the effectiveness of the model, dataframe #1 and its associated missing value handling method is the best. **Method #1** involved dropping all records with missing Glucose, Blood Pressure and BMI and then using the interpolate() function for substituting the missing values for Skin Thickness and Insulin.


Accuracy
Group
Data Frame #1
Data Frame #2
Data Frame #3
0.70
0.68
0.66
0.64
0.62
Decision Tree
Naive Bayes
ANN
SVM
Random Forest
KNN


Fscore
Group
Data Frame #1
Data Frame #2
Data Frame #3
0.700
0.675
0.650
0.625
0.600
0.575
0.550
Decision Tree
Naive Bayes
ANN
SVM
Random Forest
KNN


Recall
Group
Data Frame #1
Data Frame #2
Data Frame #3
0.72
0.70
0.68
0.66
0.64
0.62
0.60
0.58
0.56
Decision Tree
Naive Bayes
ANN
SVM
Random Forest
KNN


Precision
Group
Data Frame #1
Data Frame #2
Data Frame #3
0.70
0.65
0.60
0.55
0.50
0.45
Decision Tree
Naive Bayes
ANN
SVM
Random Forest
KNN

Code

```
#!/usr/bin/env python
# coding: utf-8

# ## Importing all needed modules and libraries
# ---

# In[373]:



import numpy as np
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
import seaborn as sb
import matplotlib.pyplot as mp
from mpl_toolkits.mplot3d import Axes3D
import warnings
warnings.filterwarnings('ignore')
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from numpy import mean
from numpy import std
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score


# ## Data Description
# ---


# In[374]:



original_data = pd.read_csv("pima-indians-diabetes.csv",header=None)
original_data



# ### Adding column names


# In[375]:



original_data=original_data.rename(columns={0: "Pregnancies", 1: "Glucose", 2 : "Blood Pressure",
3 : "Skin Thickness",
                                    4 : "Insulin", 5 : "BMI", 6 : "DiabetesPedigreeFunction", 7 : "Age",
                                    8: "Class"})
original_data



# In[376]:



original_data = original_data.astype({"Class":'category'})
```

```
# In[377]:


print("No. of rows=",original_data.shape[0])
print("No. of columns=",original_data.shape[1])
print()
print("The type of data in each column is")
original_data.dtypes


# In[378]:


for i in original_data.columns:
    print(original_data[i].describe())
    print()


# ### Here we see that we have 8 attributes (all numerical type) and 1 target attribute. Now we shall see how many classes is in the target column.

# In[379]:


set(list(original_data.iloc[:,-1]))


# ### As we see, this will be a binary classification problem.

# # Data Visualizing
# ---

# ### Histograms
# In[380]:



fig=mp.figure(figsize=(15, 30))
for i in range(8):
    ax=fig.add_subplot(4,2,i+1)
    original_data[original_data.columns[i]].hist(bins=10,ax=ax)
    ax.set_xlabel(original_data.columns[i])
    ax.set_ylabel("Frequency")
    ax.set_title(original_data.columns[i] + " Distribution")



# ### Plots

# In[381]:



ax=sb.catplot(x="Class", y=original_data.columns[0], data=original_data)
ax=sb.catplot(x="Class", y=original_data.columns[1], data=original_data)
ax=sb.catplot(x="Class", y=original_data.columns[2], data=original_data)
ax=sb.catplot(x="Class", y=original_data.columns[3], data=original_data)
ax=sb.catplot(x="Class", y=original_data.columns[4], data=original_data)
ax=sb.catplot(x="Class", y=original_data.columns[5], data=original_data)
ax=sb.catplot(x="Class", y=original_data.columns[6], data=original_data)
ax=sb.catplot(x="Class", y=original_data.columns[7], data=original_data)



# ### As we can see, there are many outliers in the data. (BMI, SkinThickness, Insulin and glucose cannot be zero). Thus outlier detection would have to be done.

# ### We seperate out the features and the target

# In[382]:
```

```
features=original_data.iloc[:,:-1]
target=original_data.iloc[:,-1]


# In[383]:


fig=mp.figure(figsize=(15, 30))
for i in range(8):
    ax=fig.add_subplot(4,2,i+1)
    sb.distplot(features[features.columns[i]])


# In[384]:


fig=mp.figure(figsize=(15, 30))
for i in range(8):
    ax=fig.add_subplot(4,2,i+1)
    sb.boxplot(features[features.columns[i]])


# ## Data Preprocessing
# ---

# ### We check null entries

# #### The dataset has no Nan values but there are null values. As we can see there are some columns which cannot have the value zero like skin thickness, glucose, insulin, BMI. All these entries are null entries and thus needed to be treated like null values

# In[385]:
```

```
features_nan=features.copy()
colswithzero=['Glucose',"Blood Pressure","Skin Thickness","Insulin","BMI"]
for i in colswithzero:
    features_nan[i] = features_nan[i].replace({0:np.nan})


# In[386]:


for i in features_nan.columns:
    print (i," - ",features_nan[i].isnull().sum())


# #### To handle null values, we will take 3 aproaches

# ### 1st Approach
# #### Drop all records with missing Glucose, Blood Pressure and BMI and using interpolate for Skin Thickenss and Insulin

# In[387]:


features1=features_nan.copy()
features1.dropna(axis=0,how='any',subset=['Glucose','Blood Pressure','BMI'], inplace=True)
features1=features1.interpolate(limit_direction='both')

notdropped=[]
total=[]
target1=target.copy()
for i,j in features1.iterrows():
    notdropped.append(i)
for i in range(len(target1)):
    total.append(i)
```

```
d=set(total)-set(notdropped)
for i in d:
    target1.drop(i,axis=0,inplace=True)



features1



# In[388]:



for i in features1.columns:
    print (i," - ",features1[i].isnull().sum())



# ### 2nd Approach
# #### Replace missing values for Glucose, Blood Pressure and BMI with the mean and interpolate values for Insulin and Skin Thickness.


# In[389]:



features2=features_nan.copy()
features2['Glucose'].fillna(features2['Glucose'].mean(), inplace=True)
features2['Blood Pressure'].fillna(features2['Blood Pressure'].mean(), inplace=True)
features2['BMI'].fillna(features2['BMI'].mean(), inplace=True)
features2=features2.interpolate(limit_direction='both')

notdropped=[]
total=[]
target2=target.copy()
for i,j in features2.iterrows():
    notdropped.append(i)
for i in range(len(target2)):
    total.append(i)
d=set(total)-set(notdropped)
for i in d:
    target2.drop(i,axis=0,inplace=True)


features2


# In[390]:


for i in features2.columns:
    print (i," - ",features2[i].isnull().sum())


# ###  3rd Approach
# #### Interpolate all values

# In[391]:


features3=features_nan.copy()
features3=features3.interpolate(limit_direction='both')

notdropped=[]
total=[]
target3=target.copy()
for i,j in features3.iterrows():
    notdropped.append(i)
for i in range(len(target3)):
    total.append(i)
d=set(total)-set(notdropped)
for i in d:
    target3.drop(i,axis=0,inplace=True)


features3


# In[392]:


for i in features3.columns:
    print (i," - ",features3[i].isnull().sum())


# #### Our data now has no null values.

# ### Outlier detection and removal
# https://www.analyticsvidhya.com/blog/2021/05/feature-engineering-how-to-detect-and-remove-outliers-with-python-code/
# https://www.asc.ohio-state.edu/goel.1//STATLEARN/PROJECTS/Presentations/Diabetes_PimaIndians.pdf

# In[393]:


fig=mp.figure(figsize=(15, 30))
for i in range(8):
    ax=fig.add_subplot(4,2,i+1)
    sb.distplot(features1[features1.columns[i]])


# In[394]:


fig=mp.figure(figsize=(15, 30))
```

```
for i in range(8):
    ax=fig.add_subplot(4,2,i+1)
    sb.distplot(features2[features2.columns[i]])


# In[395]:


fig=mp.figure(figsize=(15, 30))
for i in range(8):
    ax=fig.add_subplot(4,2,i+1)
    sb.distplot(features3[features3.columns[i]])


# In[396]:


fig=mp.figure(figsize=(15, 30))
for i in range(8):
    ax=fig.add_subplot(4,2,i+1)
    sb.boxplot(features1[features1.columns[i]])


# In[397]:


fig=mp.figure(figsize=(15, 30))
for i in range(8):
    ax=fig.add_subplot(4,2,i+1)
    sb.boxplot(features2[features2.columns[i]])


# In[398]:
```

```
fig=mp.figure(figsize=(15, 30))
for i in range(8):
    ax=fig.add_subplot(4,2,i+1)
    sb.boxplot(features3[features3.columns[i]])


# #### To remove outliers, we will employ IQR filtering.
#

# In[399]:


for i in features1.columns:
    q25=features1[i].quantile(0.25)
    q75=features1[i].quantile(0.75)
    iqr=q75-q25
    high=q75+1.5*iqr
    low=q25-1.5*iqr
    for j,k in features1.iterrows():
        if features1[i][j]>high:
            features1.at[j,i]=high
        if features1[i][j]<low:
            features1.at[j,i]=low

for i in features2.columns:
    q25=features2[i].quantile(0.25)
    q75=features2[i].quantile(0.75)
    iqr=q75-q25
    high=q75+1.5*iqr
    low=q25-1.5*iqr
    for j,k in features2.iterrows():
        if features2[i][j]>high:
            features2.at[j,i]=high
        if features2[i][j]<low:
            features2.at[j,i]=low

print()

for i in features3.columns:
    q25=features3[i].quantile(0.25)
    q75=features3[i].quantile(0.75)
    iqr=q75-q25
    high=q75+1.5*iqr
    low=q25-1.5*iqr
    for j,k in features3.iterrows():
        if features3[i][j]>high:
            features3.at[j,i]=high
        if features3[i][j]<low:
            features3.at[j,i]=low


# In[400]:


fig=mp.figure(figsize=(15, 30))
for i in range(8):
    ax=fig.add_subplot(4,2,i+1)
    sb.boxplot(features1[features1.columns[i]])


# In[401]:


fig=mp.figure(figsize=(15, 30))
for i in range(8):
    ax=fig.add_subplot(4,2,i+1)
    sb.boxplot(features2[features2.columns[i]])
```

```
# In[402]:



fig=mp.figure(figsize=(15, 30))
for i in range(8):
    ax=fig.add_subplot(4,2,i+1)
    sb.boxplot(features3[features3.columns[i]])



# ### Data Normalization
#


# In[403]:



features1=pd.DataFrame(preprocessing.normalize(features1))
features1=features1.rename(columns={0: "Pregnancies", 1: "Glucose", 2 : "Blood Pressure", 3 : "Skin Thickness",
                              4 : "Insulin", 5 : "BMI", 6 : "DiabetesPedigreeFunction", 7 : "Age"})
features1



# In[404]:



features2=pd.DataFrame(preprocessing.normalize(features2))
features2=features2.rename(columns={0: "Pregnancies", 1: "Glucose", 2 : "Blood Pressure", 3 : "Skin Thickness", 4 : "Insulin", 5 : "BMI", 6 : "DiabetesPedigreeFunction", 7 : "Age"})
features2



# In[405]:
```

```
features3=pd.DataFrame(preprocessing.normalize(features3))

features3=features3.rename(columns={0: "Pregnancies", 1: "Glucose", 2 : "Blood Pressure", 3 : "Skin Thickness", 4 : "Insulin", 5 : "BMI", 6 : "DiabetesPedigreeFunction", 7 : "Age"})

features3



# ### We find correlation if target with all the features

#


# In[406]:



fig=mp.figure(figsize=(15, 5))

features1.corrwith(target)

ax=fig.add_subplot(1,1,1)

mp.bar(features.columns,features1.corrwith(target), color ='maroon')

mp.bar("Threshold",0.7, color ='green',)

mp.bar("Threshold",-0.7, color ='green',)

mp.xlabel("Features")

mp.ylabel("Correlation value")

mp.title("Correlation value of features with target")

mp.show()




fig=mp.figure(figsize=(15, 5))

features2.corrwith(target)

ax=fig.add_subplot(1,1,1)

mp.bar(features.columns,features2.corrwith(target), color ='maroon')

mp.bar("Threshold",0.7, color ='green',)

mp.bar("Threshold",-0.7, color ='green',)

mp.xlabel("Features")
mp.ylabel("Correlation value")
mp.title("Correlation value of features with target")
mp.show()

fig=mp.figure(figsize=(15, 5))
features3.corrwith(target)
ax=fig.add_subplot(1,1,1)
mp.bar(features.columns,features3.corrwith(target), color ='maroon')
mp.bar("Threshold",0.7, color ='green',)
mp.bar("Threshold",-0.7, color ='green',)
mp.xlabel("Features")
mp.ylabel("Correlation value")
mp.title("Correlation value of features with target")
mp.show()



# #### Dropping any feature with mores than 0.7 correlation value.

# In[407]:


for i in range(len(features1.corrwith(target))):
    if abs(features1.corrwith(target)[i])>=0.7:
        features1.drop(columns=[i])



# In[408]:


for i in range(len(features2.corrwith(target))):
    if abs(features2.corrwith(target)[i])>=0.7:
        features2.drop(columns=[i])
```

```
# In[409]:


for i in range(len(features3.corrwith(target))):
    if abs(features3.corrwith(target)[i])>=0.7:
        features3.drop(columns=[i])


# ### None of the features were dropped

# ### We find correlation within features

# In[410]:


dataplot = sb.heatmap(features1.corr(), cmap="YlGnBu", annot=True)
mp.show()

dataplot = sb.heatmap(features2.corr(), cmap="YlGnBu", annot=True)
mp.show()

dataplot = sb.heatmap(features3.corr(), cmap="YlGnBu", annot=True)
mp.show()


# ### Dropping any feature with mores than 0.7 correlation value.

# In[411]:
```

```
todrop=[]
for i in range(features1.corr().shape[0]):
    for j in range(i+1,features1.corr().shape[1]):
        if abs(features1.corr()[features1.columns[i]][features1.columns[j]]) >= 0.7 and i!=j:
            todrop.append(i)

todrop=list(set(todrop))
print(todrop)

for i in todrop:
    temp=features.columns[i]
    print("Dropping ",temp, " column")
    features1=features1.drop([temp],axis=1)



# In[372]:


features1


# In[270]:



todrop=[]
for i in range(features2.corr().shape[0]):
    for j in range(i+1,features2.corr().shape[1]):
        if abs(features2.corr()[features2.columns[i]][features2.columns[j]]) >= 0.7 and i!=j:
            todrop.append(i)

todrop=list(set(todrop))

for i in todrop:
    temp=features2.columns[i]
    todrop.remove(i)
    print("Dropping ",temp, " column")
    features2=features2.drop([temp],axis=1)


# In[271]:


todrop=[]
for i in range(features3.corr().shape[0]):
    for j in range(i+1,features3.corr().shape[1]):
        if abs(features3.corr()[features3.columns[i]][features3.columns[j]]) >= 0.7 and i!=j:
            todrop.append(i)

todrop=list(set(todrop))

for i in todrop:
    temp=features3.columns[i]
    print("Dropping ",temp, " column")
    features3=features3.drop([temp],axis=1)


# In[272]:


features3


# ## Classification Models

# In[273]:


x1train,x1test,y1train,y1test=train_test_split(features1, target1,random_state=88)
```

```
x2train,x2test,y2train,y2test=train_test_split(features2, target2,random_state=88)
x3train,x3test,y3train,y3test=train_test_split(features3, target3,random_state=88)


# In[274]:


tab1= [[0 for i in range(5)] for j in range(6)]
tab2= [[0 for i in range(5)] for j in range(6)]
tab3= [[0 for i in range(5)] for j in range(6)]

tabs=['tab1','tab2','tab3']
features=['features1','features2','features3']
targets=['target1','target2','target3']
xtrain=['x1train','x2train','x3train']
ytrain=['y1train','y2train','y3train']
xtest=['x1test','x2test','x3test']
ytest=['y1test','y2test','y3test']

models=['Decision Tree','Naive Bayes','ANN','SVM','Random Forest','KNN']

for i in tabs:
    for j in range(len(models)):
        eval(i)[j][0]=models[j]


# ## Decision Tree

# In[275]:


for i in range(3):
    cv = KFold(n_splits=10, shuffle=True)
    DT=DecisionTreeClassifier(random_state=9)
    scores = cross_val_score(DT, eval(features[i]), eval(targets[i]), scoring='accuracy', cv=cv, n_jobs=-
1)

    DT.fit(eval(xtrain[i]),eval(ytrain[i]))

    ypred=DT.predict(eval(xtest[i]))

    temp=[scores.mean(),f1_score(eval(ytest[i]), ypred,average='weighted'),recall_score(eval(ytest[i]),
ypred,average='weighted'),precision_score(eval(ytest[i]), ypred,average='weighted')]

    eval(tabs[i])[0][1:5]=temp



# ## Naive Bayes


# In[276]:




for i in range(3):

    cv = KFold(n_splits=10, shuffle=True)

    NB=GaussianNB()

    scores = cross_val_score(NB, eval(features[i]), eval(targets[i]), scoring='accuracy', cv=cv, n_jobs=-
1)

    NB.fit(eval(xtrain[i]),eval(ytrain[i]))

    ypred=NB.predict(eval(xtest[i]))

    temp=[scores.mean(),f1_score(eval(ytest[i]), ypred,average='weighted'),recall_score(eval(ytest[i]),
ypred,average='weighted'),precision_score(eval(ytest[i]), ypred,average='weighted')]

    eval(tabs[i])[1][1:5]=temp



# ## ANN


# In[277]:




for i in range(3):

    cv = KFold(n_splits=10, shuffle=True)


ANN=MLPClassifier(hidden_layer_sizes=(2,3),random_state=5,verbose=False,learning_rate_init=0.0
1)
```

```
    scores = cross_val_score(ANN, eval(features[i]), eval(targets[i]), scoring='accuracy', cv=cv, 
n_jobs=-1)
    ANN.fit(eval(xtrain[i]),eval(ytrain[i]))
    ypred=ANN.predict(eval(xtest[i]))
    temp=[scores.mean(),f1_score(eval(ytest[i]), ypred,average='weighted'),recall_score(eval(ytest[i]), 
ypred,average='weighted'),precision_score(eval(ytest[i]), ypred,average='weighted')]
    eval(tabs[i])[2][1:5]=temp



# ## SVM


# In[278]:



for i in range(3):
    cv = KFold(n_splits=10, shuffle=True)
    svc=SVC(kernel='linear')
    svc.fit(x1train,y1train)
    scores = cross_val_score(svc, eval(features[i]), eval(targets[i]), scoring='accuracy', cv=cv, n_jobs=-
1)
    svc.fit(eval(xtrain[i]),eval(ytrain[i]))
    ypred=svc.predict(eval(xtest[i]))
    temp=[scores.mean(),f1_score(eval(ytest[i]), ypred,average='weighted'),recall_score(eval(ytest[i]), 
ypred,average='weighted'),precision_score(eval(ytest[i]), ypred,average='weighted')]
    eval(tabs[i])[3][1:5]=temp



# ## Random Forest


# In[279]:



for i in range(3):
    cv = KFold(n_splits=10, shuffle=True)
    rf=RandomForestClassifier(random_state=0)
    scores = cross_val_score(rf, eval(features[i]), eval(targets[i]), scoring='accuracy', cv=cv, n_jobs=-
1)
    rf.fit(eval(xtrain[i]),eval(ytrain[i]))
    ypred=rf.predict(eval(xtest[i]))
    temp=[scores.mean(),f1_score(eval(ytest[i]), ypred,average='weighted'),recall_score(eval(ytest[i]),
ypred,average='weighted'),precision_score(eval(ytest[i]), ypred,average='weighted')]
    eval(tabs[i])[4][1:5]=temp



# In[280]:


len(ypred)


# ## KNN

# In[281]:


for i in range(3):
    cv = KFold(n_splits=10, shuffle=True)
    knn = KNeighborsClassifier(n_neighbors=3)
    knn.fit(x1train,y1train)
    scores = cross_val_score(knn, eval(features[i]), eval(targets[i]), scoring='accuracy', cv=cv,
n_jobs=-1)
    knn.fit(eval(xtrain[i]),eval(ytrain[i]))
    ypred=knn.predict(eval(xtest[i]))
    temp=[scores.mean(),f1_score(eval(ytest[i]), ypred,average='weighted'),recall_score(eval(ytest[i]),
ypred,average='weighted'),precision_score(eval(ytest[i]), ypred,average='weighted')]
    eval(tabs[i])[5][1:5]=temp


# ## Evaluation
# In[282]:


tab1=pd.DataFrame(tab1)
tab2=pd.DataFrame(tab2)
tab3=pd.DataFrame(tab3)

for i in tabs:
    eval(i).rename(columns={0: "Model", 1: "Accuracy", 2 : "Fscore", 3 : "Recall", 4 : "Precision"}, 
inplace=True)


# In[283]:


tab1


# In[284]:


tab2


# In[285]:


tab3


# In[456]:


mean_Accuracy=(tab1['Accuracy']+tab2["Accuracy"]+tab3["Accuracy"])/3
```

```
mean_Fscore=(tab1['Fscore']+tab2["Fscore"]+tab3["Fscore"])/3
mean_Recall=(tab1['Recall']+tab2["Recall"]+tab3["Recall"])/3
mean_Precision=(tab1['Precision']+tab2["Precision"]+tab3["Precision"])/3


# In[459]:


mp.plot(tab1['Model'],mean_Accuracy, label='Accuracy')
mp.plot(tab1['Model'],mean_Fscore, label='Fscore')
mp.plot(tab1['Model'],mean_Recall, label='Recall')
mp.plot(tab1['Model'],mean_Precision, label='Precision')
mp.legend(title='Group')
mp.title('Mean of Evaluation Parameters for all 3 dataframes', fontsize=16)
mp.figure(figsize=(200,7))
mp.show()


# In[ ]:




# In[424]:


tab1.plot(x ='Model', y='Accuracy', kind = 'line', figsize=(10,5),title="DATA FRAME #1")
tab1.plot(x ='Model', y='Fscore', kind = 'line', figsize=(10,5),title="DATA FRAME #1")
tab1.plot(x ='Model', y='Recall', kind = 'line', figsize=(10,5),title="DATA FRAME #1")
tab1.plot(x ='Model', y='Precision', kind = 'line', figsize=(10,5),title="DATA FRAME #1")


# In[425]:
```

```
tab2.plot(x ='Model', y='Accuracy', kind = 'line', figsize=(10,5),title="DATA FRAME #2")
tab2.plot(x ='Model', y='Fscore', kind = 'line', figsize=(10,5),title="DATA FRAME #2")
tab2.plot(x ='Model', y='Recall', kind = 'line', figsize=(10,5),title="DATA FRAME #2")
tab2.plot(x ='Model', y='Precision', kind = 'line', figsize=(10,5),title="DATA FRAME #2")


# In[426]:


tab3.plot(x ='Model', y='Accuracy', kind = 'line', figsize=(10,5),title="DATA FRAME #3")
tab3.plot(x ='Model', y='Fscore', kind = 'line', figsize=(10,5),title="DATA FRAME #3")
tab3.plot(x ='Model', y='Recall', kind = 'line', figsize=(10,5),title="DATA FRAME #3")
tab3.plot(x ='Model', y='Precision', kind = 'line', figsize=(10,5),title="DATA FRAME #3")


# In[449]:


mp.plot(tab1['Model'],tab1['Accuracy'], label='Data Frame #1')
mp.plot(tab2['Accuracy'], label='Data Frame #2')
mp.plot(tab3['Accuracy'],label='Data Frame #3')
mp.legend(title='Group')
mp.title('Accuracy', fontsize=16)
mp.figure(figsize=(200,7))
mp.show()


# In[450]:


mp.plot(tab1['Model'],tab1['Fscore'], label='Data Frame #1')
mp.plot(tab2['Fscore'], label='Data Frame #2')
mp.plot(tab3['Fscore'],label='Data Frame #3')
mp.legend(title='Group')
mp.title('Fscore', fontsize=16)
mp.figure(figsize=(200,7))
mp.show()
# In[451]:
mp.plot(tab1['Model'],tab1['Recall'], label='Data Frame #1')
mp.plot(tab2['Recall'], label='Data Frame #2')
mp.plot(tab3['Recall'],label='Data Frame #3')
mp.legend(title='Group')
mp.title('Recall', fontsize=16)
mp.figure(figsize=(200,7))
mp.show()
# In[452]:
mp.plot(tab1['Model'],tab1['Precision'], label='Data Frame #1')
mp.plot(tab2['Precision'], label='Data Frame #2')
mp.plot(tab3['Precision'],label='Data Frame #3')
mp.legend(title='Group')
mp.title('Precision', fontsize=16)
mp.figure(figsize=(200,7))
mp.show()
```