Individual Assignment

-Aditya Vardhan bayya

-SE22UCSE015

OBJECTIVE:

The objective of this project is to develop a machine learning model which predicts the presence of the disease using the provided dataset. Model will be assessed based on ROC-AUC performance on unseen data.

DATASETS:

Training Dataset- contains target variable

Test Dataset-target variable is not present

CHALLENGES ENCOUNTERED:

All the predictions came 0 for the entire test set, this occurred because training on train set didn't occur properly.

When I used Xgboost roc value didn't come more than 0.5.

I had to choose other model , RANDOM FOREST gave the best ROC value as it hyperparameter can be changed according to our data needed.

Test roc was 0.95, which came out to be a good model when I used RANDOM FOREST.

STEPS INVOLVED:

STEP-1: DATA PREPROCESSING

Downloading the training data set

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
```

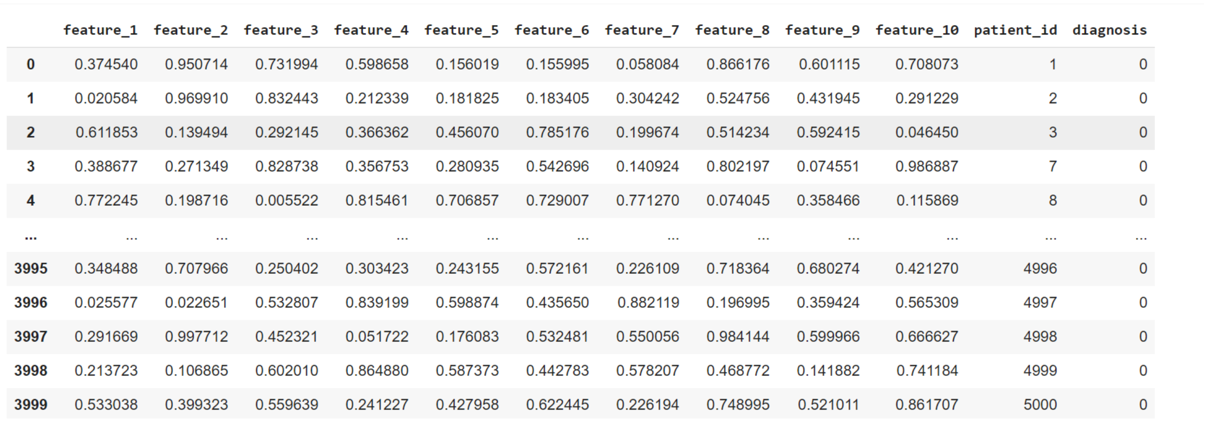
1) LOADING THE TRAINING DATASET

Using various python libraries for importing the data

We use the following code for representing the training data in machine learning

```python
#loading the training dataset
data=pd.read_csv('drive/MyDrive/Disease_train.csv')
data
```

Output of the dataset with minimization set is below:



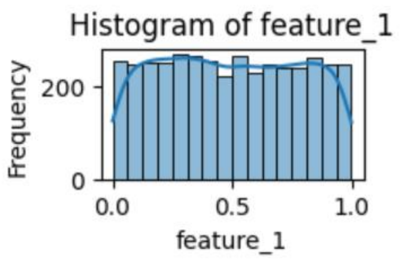
	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10	patient_id	diagnosis
0	0.374540	0.950714	0.731994	0.598658	0.156019	0.155995	0.058084	0.866176	0.601115	0.708073	1	0
1	0.020584	0.969910	0.832443	0.212339	0.181825	0.183405	0.304242	0.524756	0.431945	0.291229	2	0
2	0.611853	0.139494	0.292145	0.366362	0.456070	0.785176	0.199674	0.514234	0.592415	0.046450	3	0
3	0.388677	0.271349	0.828738	0.356753	0.280935	0.542696	0.140924	0.802197	0.074551	0.986887	7	0
4	0.772245	0.198716	0.005522	0.815461	0.706857	0.729007	0.771270	0.074045	0.358466	0.115869	8	0
...	...	...	...	...	...	...	...	...	...	...	...	...
3995	0.348488	0.707966	0.250402	0.303423	0.243155	0.572161	0.226109	0.718364	0.680274	0.421270	4996	0
3996	0.025577	0.022651	0.532807	0.839199	0.598874	0.435650	0.882119	0.196995	0.359424	0.565309	4997	0
3997	0.291669	0.997712	0.452321	0.051722	0.176083	0.532481	0.550056	0.984144	0.599966	0.666627	4998	0
3998	0.213723	0.106865	0.602010	0.864880	0.587373	0.442783	0.578207	0.468772	0.141882	0.741184	4999	0
3999	0.533038	0.399323	0.559639	0.241227	0.427958	0.622445	0.226194	0.748995	0.521011	0.861707	5000	0

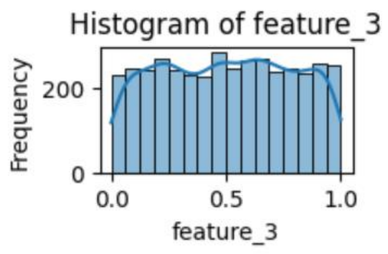
Plotting histogram for all columns

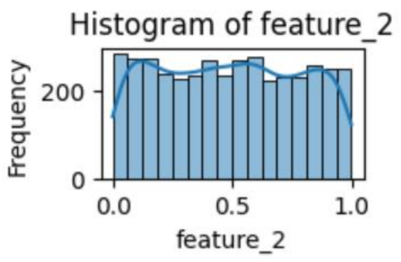
```python
df = pd.DataFrame(data)

# Plot histograms for each column (excluding 'patient_id')
columns = df.columns.drop('patient_id')
for column in columns:
    plt.figure(figsize=(2, 1))
    sns.histplot(df[column], kde=True)
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.show()
```

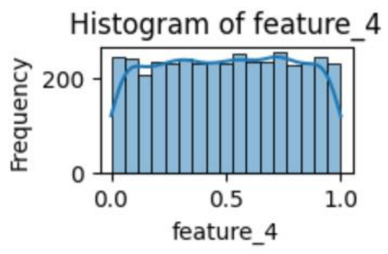
Plt.show() provides graphs for all the features in the dataset

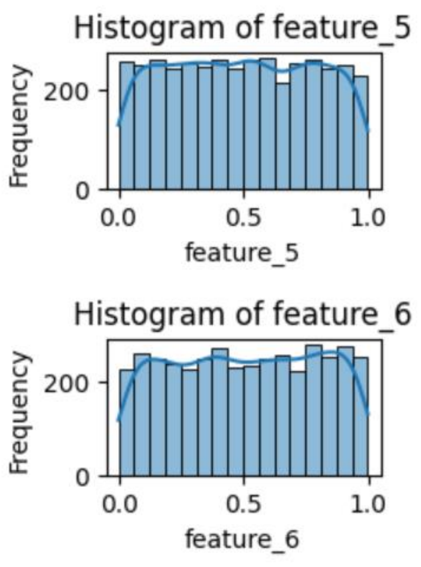
The following are the histograms of the given dataset

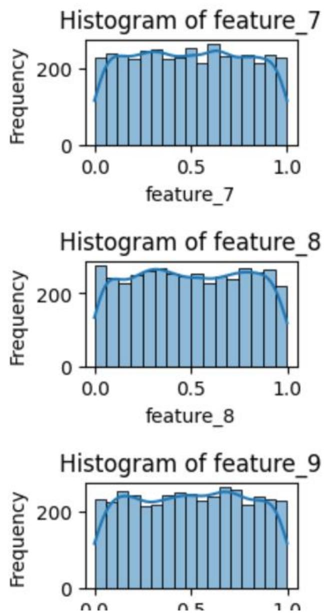


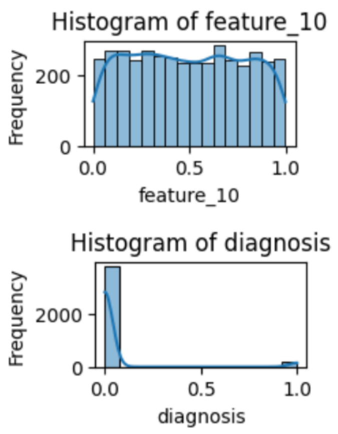








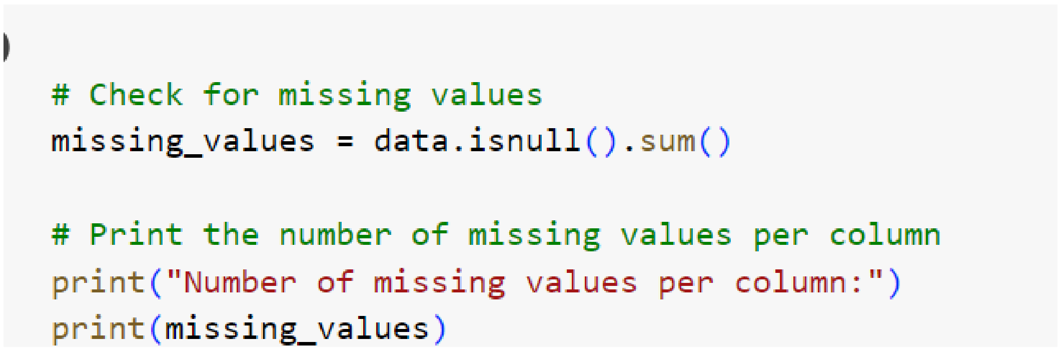




2) HANDLING THE MISSING VALUES

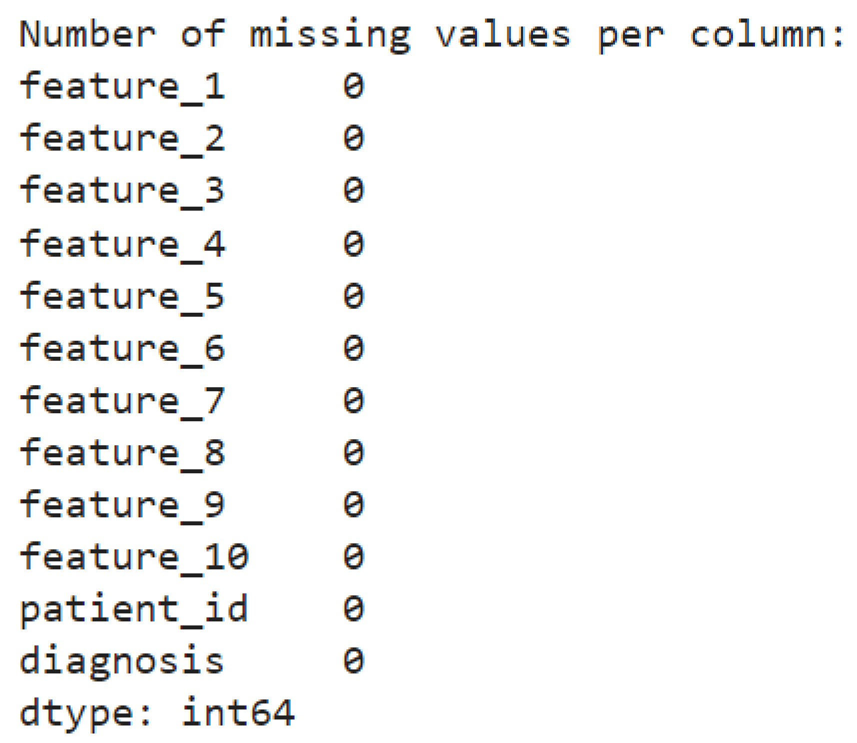
Checking if any missing values are present in the data

Using the below code for checking the missing values in the set



```
# Check for missing values
missing_values = data.isnull().sum()

# Print the number of missing values per column
print("Number of missing values per column:")
print(missing_values)
```

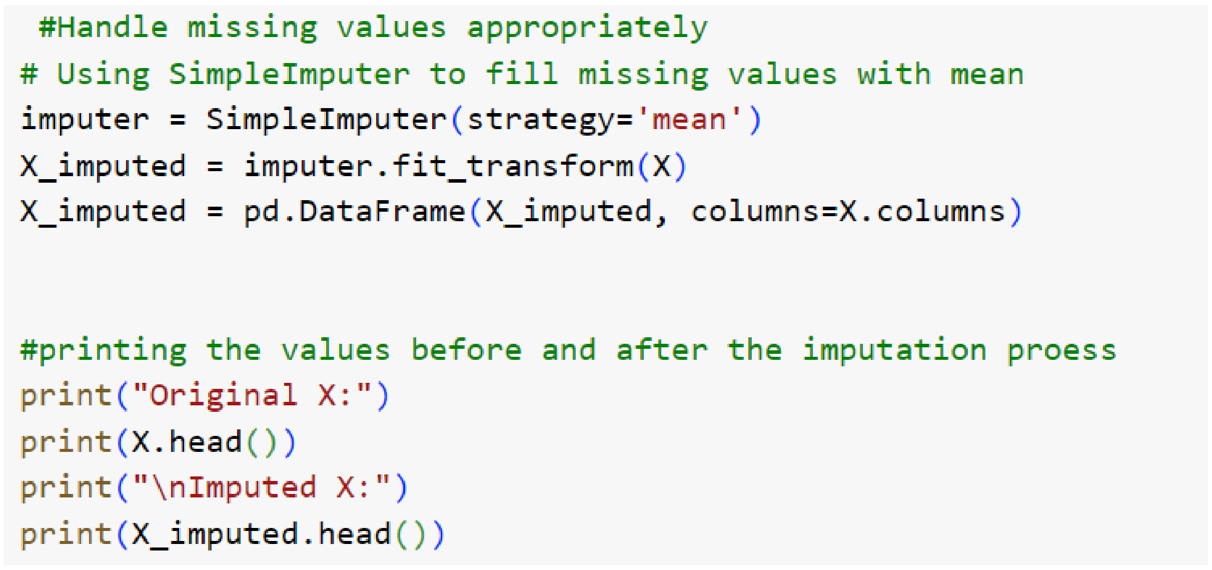
OUTPUT OF THE MISSING VALUES IS:



```
Number of missing values per column:
feature_1     0
feature_2     0
feature_3     0
feature_4     0
feature_5     0
feature_6     0
feature_7     0
feature_8     0
feature_9     0
feature_10    0
patient_id    0
diagnosis     0
dtype: int64
```

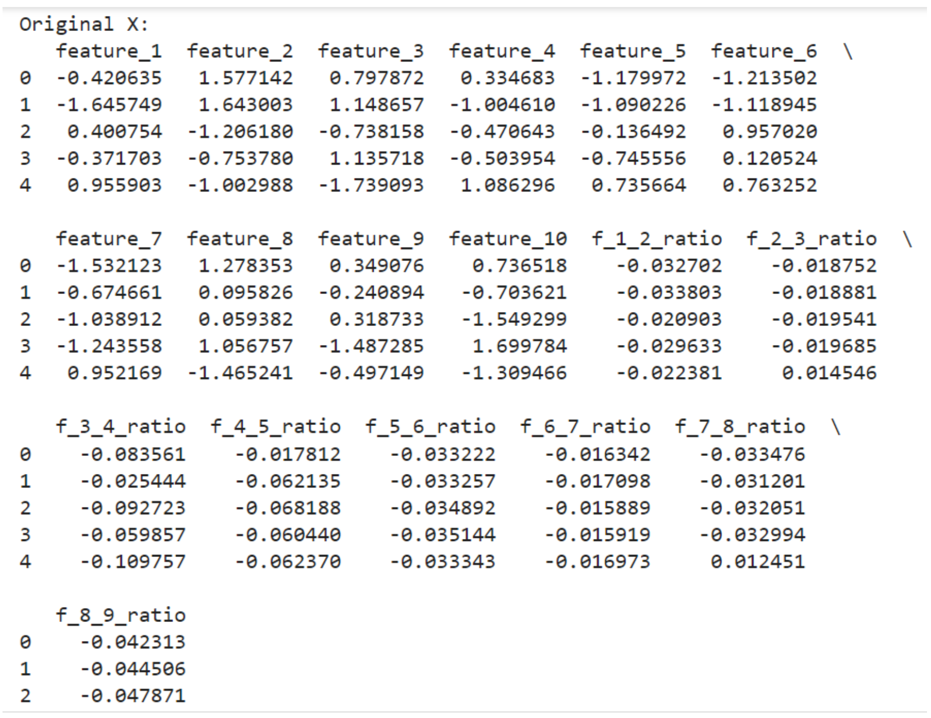
These are all the missing values in the data. There is no need to handle the missing values, but if we perform handling on this data, the data will be repeated as it is.

Using the following snippet we can handle the missing values if present



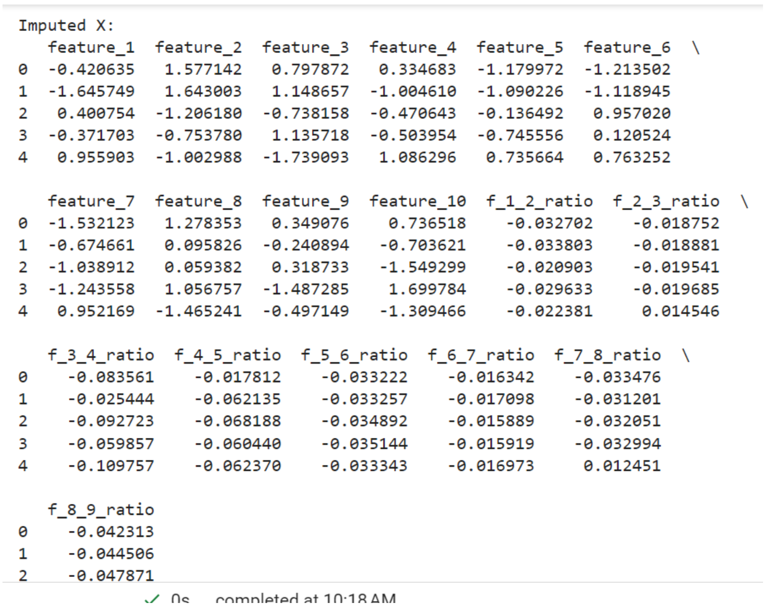
```
 #Handle missing values appropriately
# Using SimpleImputer to fill missing values with mean
imputer = SimpleImputer(strategy='mean')
X_imputed = imputer.fit_transform(X)
X_imputed = pd.DataFrame(X_imputed, columns=X.columns)


#printing the values before and after the imputation proess
print("Original X:")
print(X.head())
print("\nImputed X:")
print(X_imputed.head())
```

Output before and after imputation



```
Original X:
   feature_1  feature_2  feature_3  feature_4  feature_5  feature_6  \
0  -0.420635   1.577142   0.797872   0.334683  -1.179972  -1.213502
1  -1.645749   1.643003   1.148657  -1.004610  -1.090226  -1.118945
2   0.400754  -1.206180  -0.738158  -0.470643  -0.136492   0.957020
3  -0.371703  -0.753780   1.135718  -0.503954  -0.745556   0.120524
4   0.955903  -1.002988  -1.739093   1.086296   0.735664   0.763252

   feature_7  feature_8  feature_9  feature_10  f_1_2_ratio  f_2_3_ratio  \
0  -1.532123   1.278353   0.349076    0.736518    -0.032702    -0.018752
1  -0.674661   0.095826  -0.240894   -0.703621    -0.033803    -0.018881
2  -1.038912   0.059382   0.318733   -1.549299    -0.020903    -0.019541
3  -1.243558   1.056757  -1.487285    1.699784    -0.029633    -0.019685
4   0.952169  -1.465241  -0.497149   -1.309466    -0.022381     0.014546

   f_3_4_ratio  f_4_5_ratio  f_5_6_ratio  f_6_7_ratio  f_7_8_ratio  \
0    -0.083561    -0.017812    -0.033222    -0.016342    -0.033476
1    -0.025444    -0.062135    -0.033257    -0.017098    -0.031201
2    -0.092723    -0.068188    -0.034892    -0.015889    -0.032051
3    -0.059857    -0.060440    -0.035144    -0.015919    -0.032994
4    -0.109757    -0.062370    -0.033343    -0.016973     0.012451

   f_8_9_ratio
0    -0.042313
1    -0.044506
2    -0.047871
```

Values after imputation is performed:



```
Imputed X:
   feature_1  feature_2  feature_3  feature_4  feature_5  feature_6  \
0  -0.420635   1.577142   0.797872   0.334683  -1.179972  -1.213502
1  -1.645749   1.643003   1.148657  -1.004610  -1.090226  -1.118945
2   0.400754  -1.206180  -0.738158  -0.470643  -0.136492   0.957020
3  -0.371703  -0.753780   1.135718  -0.503954  -0.745556   0.120524
4   0.955903  -1.002988  -1.739093   1.086296   0.735664   0.763252

   feature_7  feature_8  feature_9  feature_10  f_1_2_ratio  f_2_3_ratio  \
0  -1.532123   1.278353   0.349076    0.736518    -0.032702    -0.018752
1  -0.674661   0.095826  -0.240894   -0.703621    -0.033803    -0.018881
2  -1.038912   0.059382   0.318733   -1.549299    -0.020903    -0.019541
3  -1.243558   1.056757  -1.487285    1.699784    -0.029633    -0.019685
4   0.952169  -1.465241  -0.497149   -1.309466    -0.022381     0.014546

   f_3_4_ratio  f_4_5_ratio  f_5_6_ratio  f_6_7_ratio  f_7_8_ratio  \
0    -0.083561    -0.017812    -0.033222    -0.016342    -0.033476
1    -0.025444    -0.062135    -0.033257    -0.017098    -0.031201
2    -0.092723    -0.068188    -0.034892    -0.015889    -0.032051
3    -0.059857    -0.060440    -0.035144    -0.015919    -0.032994
4    -0.109757    -0.062370    -0.033343    -0.016973     0.012451

   f_8_9_ratio
0    -0.042313
1    -0.044506
2    -0.047871
```

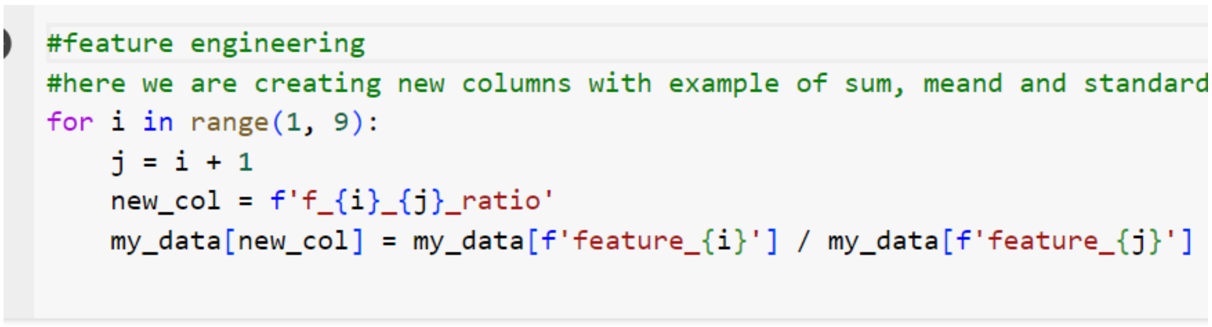
✓ 0s completed at 10:18 AM

We can observe that values before and after imputation are same and there is no change.

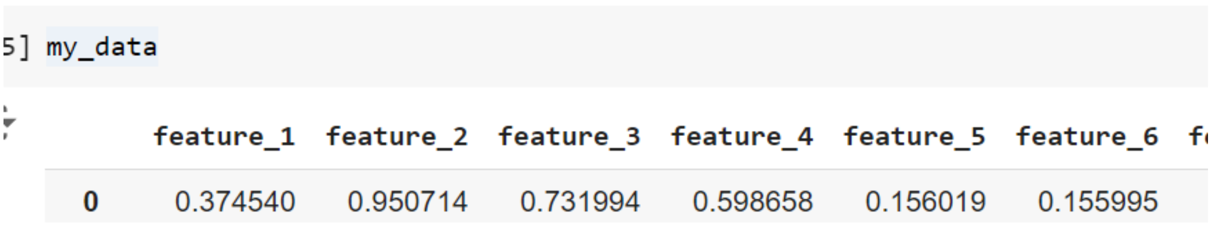
FEATURE ENGINEERING

Then we have scaled the features and also performed feature engineering.

Taking the first 9 features and doing scaling of the data



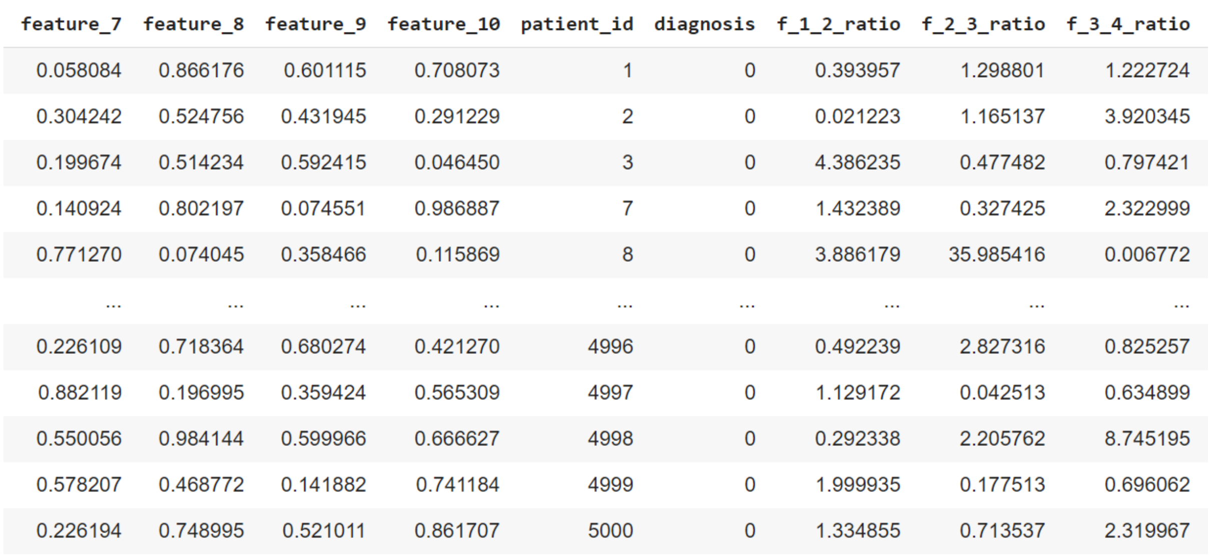
```
#feature engineering
#here we are creating new columns with example of sum, meand and standard
for i in range(1, 9):
    j = i + 1
    new_col = f'f_{i}_{j}_ratio'
    my_data[new_col] = my_data[f'feature_{i}'] / my_data[f'feature_{j}']
```



```
5] my_data
```

```
     feature_1  feature_2  feature_3  feature_4  feature_5  feature_6  fe
0    0.374540   0.950714   0.731994   0.598658   0.156019   0.155995
```

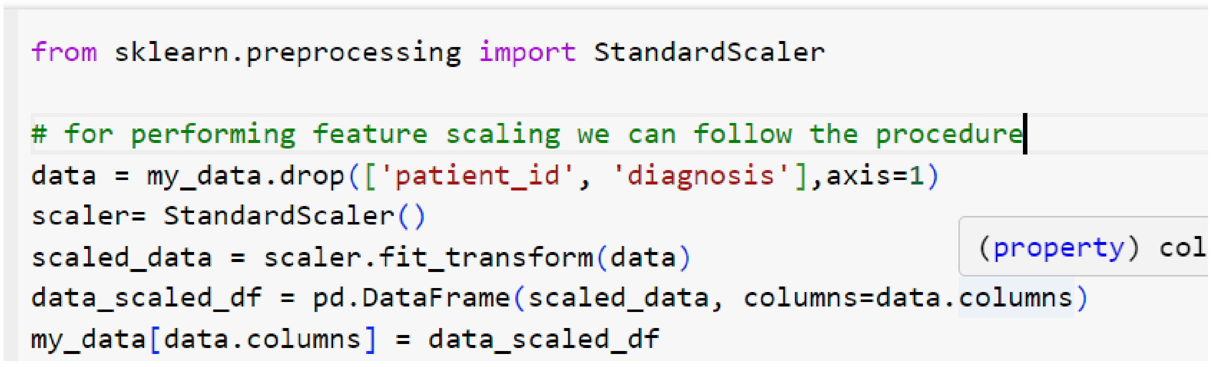
After using the above code we got the data of feature engineering results



feature_7	feature_8	feature_9	feature_10	patient_id	diagnosis	f_1_2_ratio	f_2_3_ratio	f_3_4_ratio
0.058084	0.866176	0.601115	0.708073	1	0	0.393957	1.298801	1.222724
0.304242	0.524756	0.431945	0.291229	2	0	0.021223	1.165137	3.920345
0.199674	0.514234	0.592415	0.046450	3	0	4.386235	0.477482	0.797421
0.140924	0.802197	0.074551	0.986887	7	0	1.432389	0.327425	2.322999
0.771270	0.074045	0.358466	0.115869	8	0	3.886179	35.985416	0.006772
...	...	...	...	...	...	...	...	...
0.226109	0.718364	0.680274	0.421270	4996	0	0.492239	2.827316	0.825257
0.882119	0.196995	0.359424	0.565309	4997	0	1.129172	0.042513	0.634899
0.550056	0.984144	0.599966	0.666627	4998	0	0.292338	2.205762	8.745195
0.578207	0.468772	0.141882	0.741184	4999	0	1.999935	0.177513	0.696062
0.226194	0.748995	0.521011	0.861707	5000	0	1.334855	0.713537	2.319967

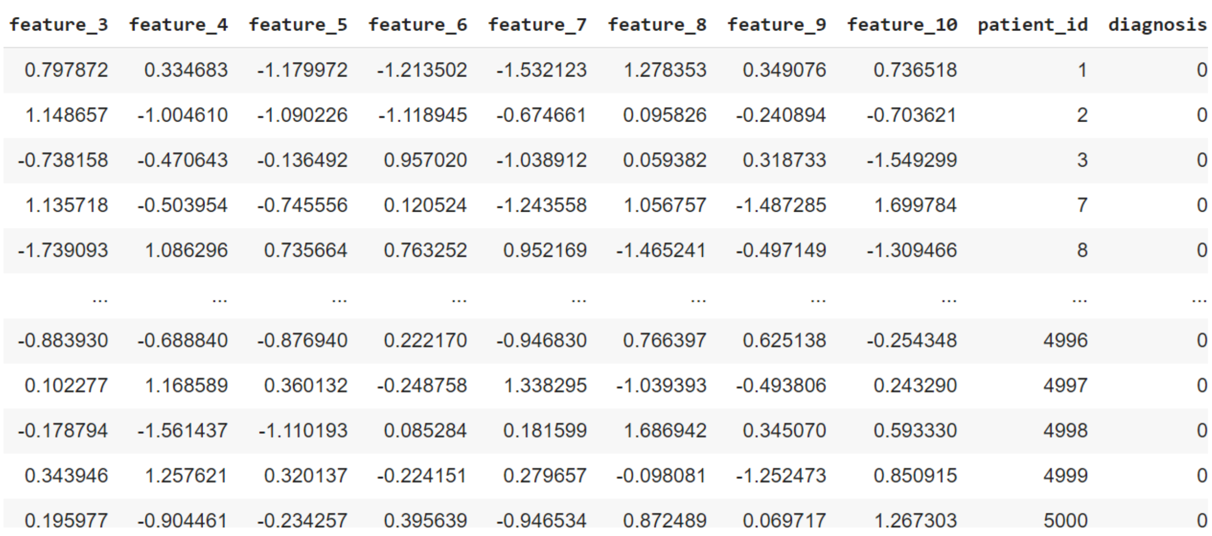
FEATURE SCALING

Using this code and scaling the data and after scaling again putting back into the required dataframe



```
from sklearn.preprocessing import StandardScaler

# for performing feature scaling we can follow the procedure
data = my_data.drop(['patient_id', 'diagnosis'],axis=1)
scaler= StandardScaler()
scaled_data = scaler.fit_transform(data)
data_scaled_df = pd.DataFrame(scaled_data, columns=data.columns)
my_data[data.columns] = data_scaled_df
```

Output of the scaled data set is above:

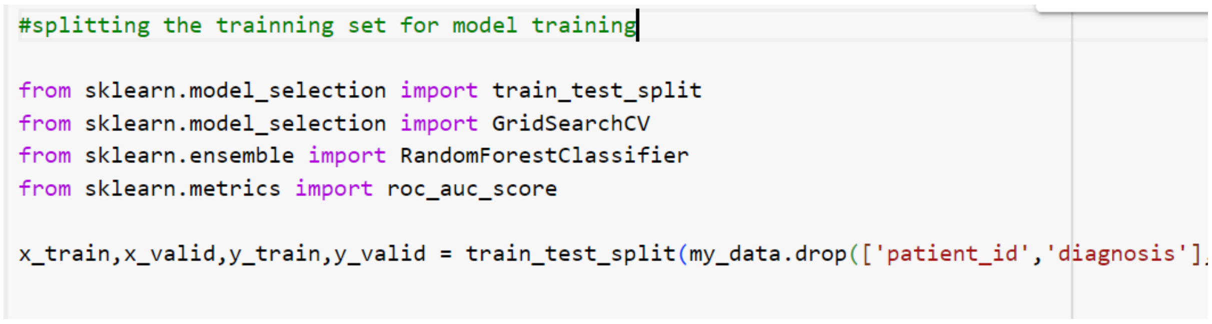


	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	feature_10	patient_id	diagnosis
	0.797872	0.334683	-1.179972	-1.213502	-1.532123	1.278353	0.349076	0.736518	1	0
	1.148657	-1.004610	-1.090226	-1.118945	-0.674661	0.095826	-0.240894	-0.703621	2	0
	-0.738158	-0.470643	-0.136492	0.957020	-1.038912	0.059382	0.318733	-1.549299	3	0
	1.135718	-0.503954	-0.745556	0.120524	-1.243558	1.056757	-1.487285	1.699784	7	0
	-1.739093	1.086296	0.735664	0.763252	0.952169	-1.465241	-0.497149	-1.309466	8	0
	...	...	...	...	...	...	...	...	...	...
	-0.883930	-0.688840	-0.876940	0.222170	-0.946830	0.766397	0.625138	-0.254348	4996	0
	0.102277	1.168589	0.360132	-0.248758	1.338295	-1.039393	-0.493806	0.243290	4997	0
	-0.178794	-1.561437	-1.110193	0.085284	0.181599	1.686942	0.345070	0.593330	4998	0
	0.343946	1.257621	0.320137	-0.224151	0.279657	-0.098081	-1.252473	0.850915	4999	0
	0.195977	-0.904461	-0.234257	0.395639	-0.946534	0.872489	0.069717	1.267303	5000	0

400 rows * 20 columns

STEP-2: MODEL TRAINING

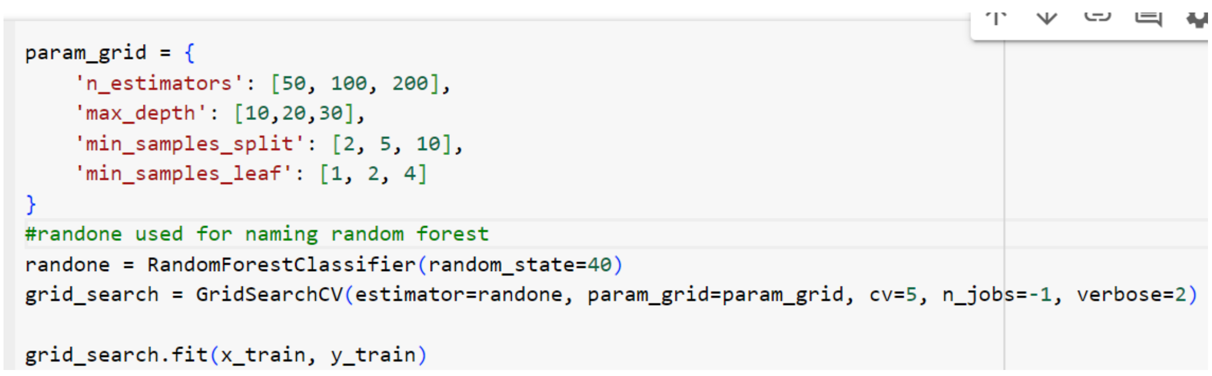
Splitting the data set in training dataset into validation subsets



```
#splitting the trainning set for model training

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import roc_auc_score

x_train,x_valid,y_train,y_valid = train_test_split(my_data.drop(['patient_id','diagnosis']
```

I have used random forest method for splitting the data and also for model training

Generating the best hyperparameters such that toc score is above 0.8 that depicts that model is perfect.



```
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [10,20,30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
#randone used for naming random forest
randone = RandomForestClassifier(random_state=40)
grid_search = GridSearchCV(estimator=randone, param_grid=param_grid, cv=5, n_jobs=-1, verbose=2)

grid_search.fit(x_train, y_train)
```

Checking of best parameters and also knowing roc score of the data

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE

# Best parameters and best score
print("Best Parameters:", grid_search.best_params_)
print("Best ROC-AUC Score:", grid_search.best_score_)

# Evaluate on validation set
val_y_pred = grid_search.predict_proba(x_valid)[:, 1]
roc = roc_auc_score(y_valid, val_y_pred)
```

Smote is used incase of there is oversampling it will correct the data and make correct prediction and it is very crucial step.

Output of best parameters and also roc score

```
Best Parameters: {'max_depth': 10, 'min_samples_leaf': 2, 'min_samples_split': 2, 'n_estimators': 50}
Best ROC-AUC Score: 0.9521875
```

I have splitted the data into train and test and now with the help of random forest will train and test the model and also predict roc on test data with the parameters generated in the above section

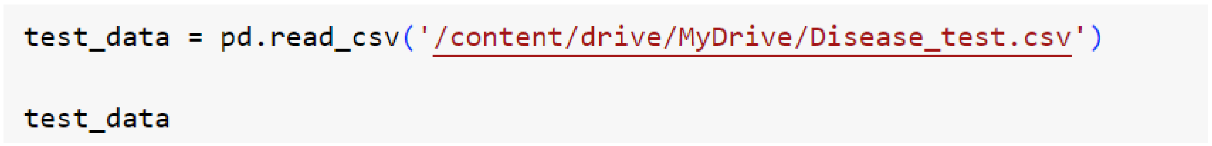
```
from sklearn.ensemble import RandomForestClassifier

n_estimators = grid_search.best_params_['n_estimators']

rfc = RandomForestClassifier(random_state=40, class_weight='balanced',
                             n_estimators=n_estimators, max_depth=20,
                             min_samples_leaf=2,
                             min_samples_split=grid_search.best_params_

rfc.fit(x_train, y_train)

# Predict class labels for the test set
y_test_pred = rfc.predict(x_test)

# Compute ROC-AUC score on the test set (if needed)
roc_test = roc_auc_score(y_test, y_test_pred)
print(f'ROC on test : {roc_test}')
```

After giving all the parameters the roc on test came out to be 0.9534

Roc above 0.8 says that model is performing good on test data where we have splitted the data into train and test.

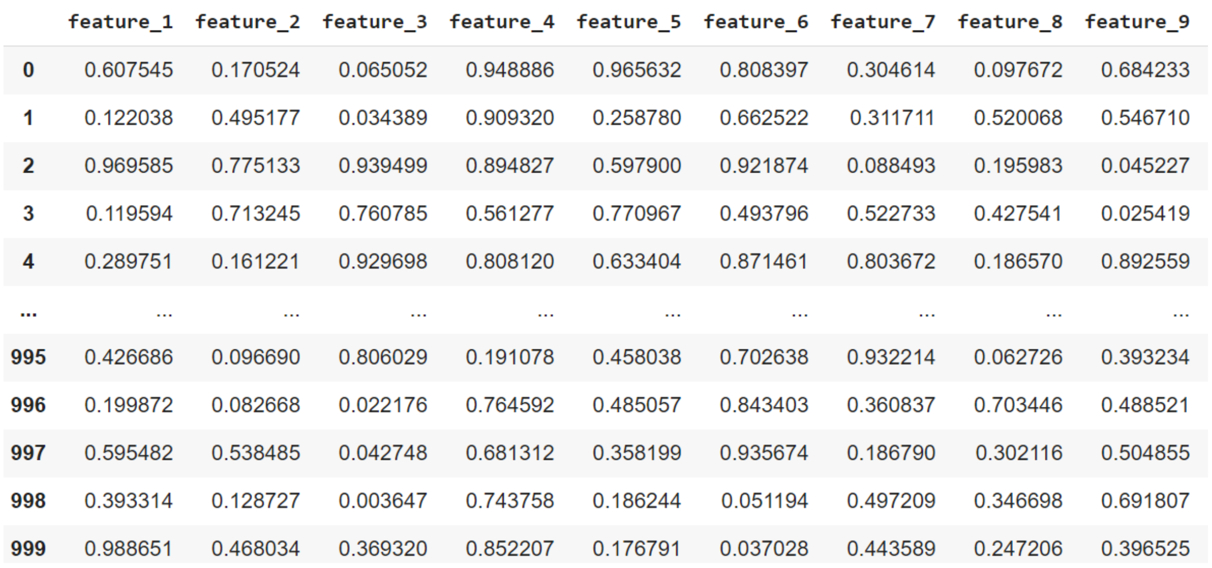
STEP-3: PREDICTION

Loding the test data set into the google collab



```
test_data = pd.read_csv('/content/drive/MyDrive/Disease_test.csv')

test_data
```

Output of the test_data is



	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9
0	0.607545	0.170524	0.065052	0.948886	0.965632	0.808397	0.304614	0.097672	0.684233
1	0.122038	0.495177	0.034389	0.909320	0.258780	0.662522	0.311711	0.520068	0.546710
2	0.969585	0.775133	0.939499	0.894827	0.597900	0.921874	0.088493	0.195983	0.045227
3	0.119594	0.713245	0.760785	0.561277	0.770967	0.493796	0.522733	0.427541	0.025419
4	0.289751	0.161221	0.929698	0.808120	0.633404	0.871461	0.803672	0.186570	0.892559
...	...	...	...	...	...	...	...	...	...
995	0.426686	0.096690	0.806029	0.191078	0.458038	0.702638	0.932214	0.062726	0.393234
996	0.199872	0.082668	0.022176	0.764592	0.485057	0.843403	0.360837	0.703446	0.488521
997	0.595482	0.538485	0.042748	0.681312	0.358199	0.935674	0.186790	0.302116	0.504855
998	0.393314	0.128727	0.003647	0.743758	0.186244	0.051194	0.497209	0.346698	0.691807
999	0.988651	0.468034	0.369320	0.852207	0.176791	0.037028	0.443589	0.247206	0.396525

Test data doesn’t have diagnosis information as we have to predict on test data

Columns in training and testing data should be same hence we again perform feature scaling and make the columns same

```
for i in range(1, 9):
    j = i + 1
    new_col = f'f_{i}_{j}_ratio'
    test_data[new_col] = test_data[f'feature_{i}'] / test_data[f'feature_{j}']
```

Generating a new test data after adding features

```
test_data_new = test_data.drop(columns=['patient_id'],axis=1)

test_data_new
```

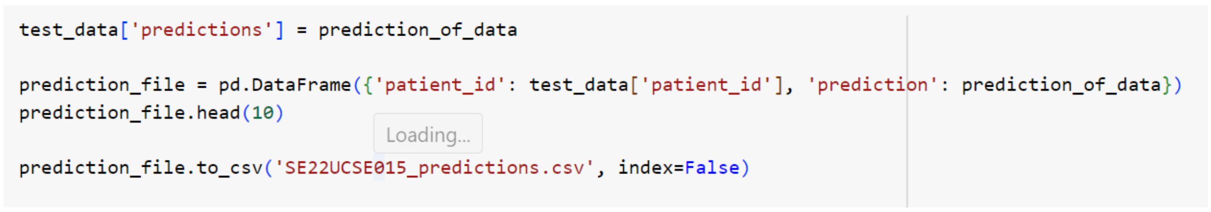
Output after adding columns

	feature_1	feature_2	feature_3	feature_4	feature_5	feature_6	feature_7	feature_8	feature_9	f
0	0.607545	0.170524	0.065052	0.948886	0.965632	0.808397	0.304614	0.097672	0.684233	
1	0.122038	0.495177	0.034389	0.909320	0.258780	0.662522	0.311711	0.520068	0.546710	
2	0.969585	0.775133	0.939499	0.894827	0.597900	0.921874	0.088493	0.195983	0.045227	
3	0.119594	0.713245	0.760785	0.561277	0.770967	0.493796	0.522733	0.427541	0.025419	
4	0.289751	0.161221	0.929698	0.808120	0.633404	0.871461	0.803672	0.186570	0.892559	
...	...	...	...	...	...	...	...	...	...	
995	0.426686	0.096690	0.806029	0.191078	0.458038	0.702638	0.932214	0.062726	0.393234	
996	0.199872	0.082668	0.022176	0.764592	0.485057	0.843403	0.360837	0.703446	0.488521	
997	0.595482	0.538485	0.042748	0.681312	0.358199	0.935674	0.186790	0.302116	0.504855	
998	0.393314	0.128727	0.003647	0.743758	0.186244	0.051194	0.497209	0.346698	0.691807	
999	0.988651	0.468034	0.369320	0.852207	0.176791	0.037028	0.443589	0.247206	0.396525	

```
X_test_scaled = scaler.transform(test_data_new)
X_scaled_data = pd.DataFrame(X_test_scaled, columns=test_data_new.columns)

prediction_of_data = rfc.predict(X_scaled_data)
prediction_of_data
```

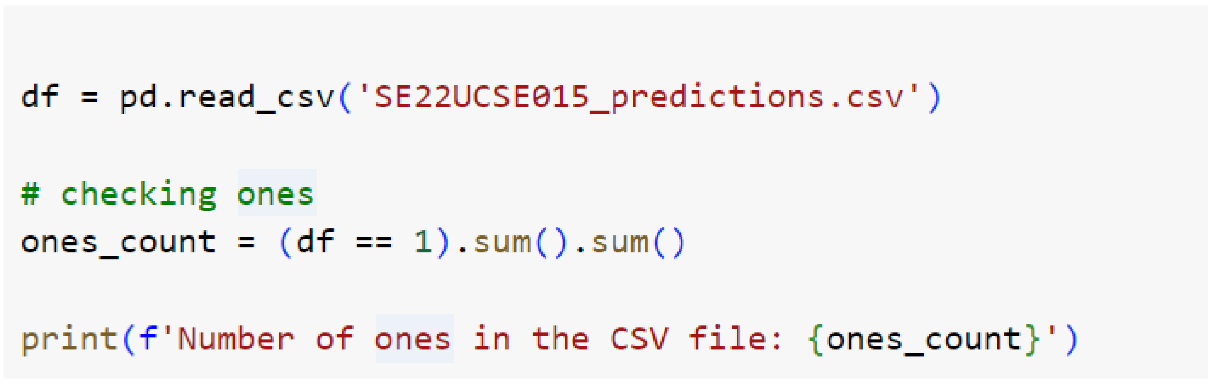
Adding the scaling features to the test data as we have to predict the values on the new test data

PREDICTING VALUES IN TEST DATA



```
test_data['predictions'] = prediction_of_data

prediction_file = pd.DataFrame({'patient_id': test_data['patient_id'], 'prediction': prediction_of_data})
prediction_file.head(10)

prediction_file.to_csv('SE22UCSE015_predictions.csv', index=False)
```

Naming the columns in csv file as patiend_id and prediction



```
df = pd.read_csv('SE22UCSE015_predictions.csv')

# checking ones
ones_count = (df == 1).sum().sum()

print(f'Number of ones in the CSV file: {ones_count}')
```

Checking the number of ones in csv file in order to know whether the data predicted is nearly correct or no

File of csv will be saved in designated site and we can check the test prediction.