**DATA MINING AND WAREHOUSING**

Group members :

1) Gibraan Jafar 421037 BE – A div

2) Divyank Shah 422048 BE – B div

3) Ankit Sonje 422060 BE – B div

4) Sneha Ukarande 422070 BE – B div

Project Title :

Predicting diabetes mellitus using classification algorithms after relevant data mining.

Project Definition :

Consider a labeled dataset belonging to an application domain. Apply suitable data

preprocessing steps such as handling of null values, data reduction, discretization. For

prediction of class labels of given data instances, build classifier models using different

techniques (minimum 3), analyze the confusion matrix and compare these models. Also apply cross validation while preparing the training and testing datasets.

Performing data analysis of the Pima Indian diabetes dataset by using various data mining techniques followed by fitting various classification machine learning models on the same to predict whether a person suffers from Type2 of diabetes mellitus or not .

Learning Objectives :

The purpose of this mini project is to understand how to derive meaning insights from existing data-sets . Also it is the aim to understand how to create datasets from data pools by handling null values , performing data reduction and discretization which can be used to build regression or classifier models .

Software Requirements :

1) Anaconda IDE

2) Editor ( Spyder or Jupyter Notebook )

3) Python ( preferably 3.6 )

4) Packages

i) Numpy

ii) Pandas

iii) Matplotlib

iv) Seaborn

v) Sklearn

Hardware Requirements :

8 GB RAM , Intel i5 processor , Nvidia 2 GB GPU , reliable and fast internet connection , 1TB HDD , Lenovo Ideapad 310

Learning Objectives :

We are going to understand fundamentals of Data Mining , to identify the appropriateness and need of mining the data . To learn the preprocessing, mining and post processing of the data . To understand various methods, techniques and algorithms in data mining .

Outcomes :

After this mini project , you will be able to

i) Apply basic , intermediate and advanced techniques to mine the data.

ii) Analyze the output generated by the process of data mining .

Iii) Explore the hidden patterns in the data.

iv) Optimize the mining process by choosing best data mining technique

v) Will be able to appropriate basic machine learning models for target purpose

What is Data Mining ?

**Data mining** is the process of discovering patterns in large data sets involving methods at the intersection of [machine learning](https://en.wikipedia.org/wiki/Machine_learning), statistics, and database systems .Data mining is the process of analyzing hidden patterns of data according to different perspectives for categorization into useful information, which is collected and assembled in common areas, such as data warehouses, for efficient analysis, data mining algorithms, facilitating business decision making and other information requirements to ultimately cut costs and increase revenue.

Data mining is also known as data discovery and knowledge discovery.

The major steps involved in a data mining process are:

* Extract, transform and load data into a data warehouse
* Store and manage data in a multidimensional databases
* Provide data access to business analysts using application software
* Present analyzed data in easily understandable forms, such as graphs .

Some of the tools used :

1) Numpy (Numerical Python) is a linear algebra library in Python. It is a very important library on which almost every data science or machine learning Python packages such as SciPy (Scientific Python), Mat−plotlib (plotting library), Scikit-learn, etc depends on to a reasonable extent.

NumPy is very useful for performing mathematical and logical operations on Arrays. It provides an abundance of useful features for operations on n-arrays and matrices in Python.

2) **Pandas** : Pandas is an easy to use and a very powerful library for data analysis. Like NumPy, it vectorises most of the basic operations that can be parallely computed even on a CPU, resulting in faster computation. Pandas is built on top of the **NumPy** package, meaning a lot of the structure of NumPy is used or replicated in Pandas. Data in pandas is often used to feed statistical analysis in **SciPy**, plotting functions from **Matplotlib**, and machine learning algorithms in **Scikit-learn**.

3) **Matplotlib** is a plotting library in Python programming language . It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits . Matplotlib is designed to be as usable as MATLAB, with the ability to use Python, and the advantage of being free and open-source. Matplotlib is a Python 2D plotting library which produces publication quality figures in a variety of hardcopy formats and interactive environments across platforms. Matplotlib can be used in Python scripts, the Python and IPython shells, the Jupyter notebook, web application servers, and four graphical user interface toolkits.

4) **Seaborn** is a Python data visualization library based on matplotlib. It provides a high-level interface for drawing attractive and informative statistical graphics.

I] **Logistic Regression** is a statistical model that in its basic form uses a logistic function ( example : a sigmoid function ) to model a binary dependent variable , although many more complex extensions exist . In regression analysis , logistic regression is estimation the parameters of a logistic model. Logistic Regression is used in various fields , including machine learning , mostly medical fields and social sciences eg : Trauma nd Injury Severity Score (TRISS)

II] **Naive Bayes** classifier is a probabilistic machine learning model that’s used for classification task. The crux of the classifier is based on the Bayes theorem.

P(A/B) = P(B/A) x P(A) / P(B)

These are a family of simple probabilistic classifiers based on applying Bayes Theorem with string ( naive independence assumptions between the features .They are among the simplest bayesian network models . Types :

i) Gaussian Naive bayes

ii) Multinomial Naive bayes

iii) Bernoulli Naive bayes

III] **Decision Trees :**

A decision tree is a decision support tool that uses a tree-like model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm that only contains conditional control statements. Decision Tree consists of :

1. Nodes : Test for the value of a certain attribute.
2. Edges/ Branch : Correspond to the outcome of a test and connect to the next node or leaf.
3. Leaf nodes : Terminal nodes that predict the outcome (represent class labels or class distribution).

Types of Decision trees :

i) Classification

ii) Regression

IV] **Support Vector Machine :**

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples. In two dimentional space this hyperplane is a line dividing a plane in two parts where in each class lay in either side **.**

V] **K Nearest Neighbour**

K Nearest Neighbour is a non-parametric method used for classification and regression . In both cases the input consists of the k closest training samples in the feature space .

In K-KNN classification , the output is a class membership . An object is classified by a plurality vote its neighbours m with the object being assigned to the class most common among its k nearest neighbors .

In K-NN Regression , th output is the property value for the object . This value is the average of the values of k nearest neighbors .

K-NN is a type of instance based learning or lazy learning , where the funciton is only approximated locally and all computation is deferred until classification .

Sample Code :

# to supress warnings

import warnings

warnings.filterwarnings('ignore')

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

%matplotlib inline

import seaborn as sns

from sklearn.preprocessing import StandardScaler

# for confusion matrix

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import accuracy\_score

from sklearn.metrics import classification\_report

**# Importing the data**

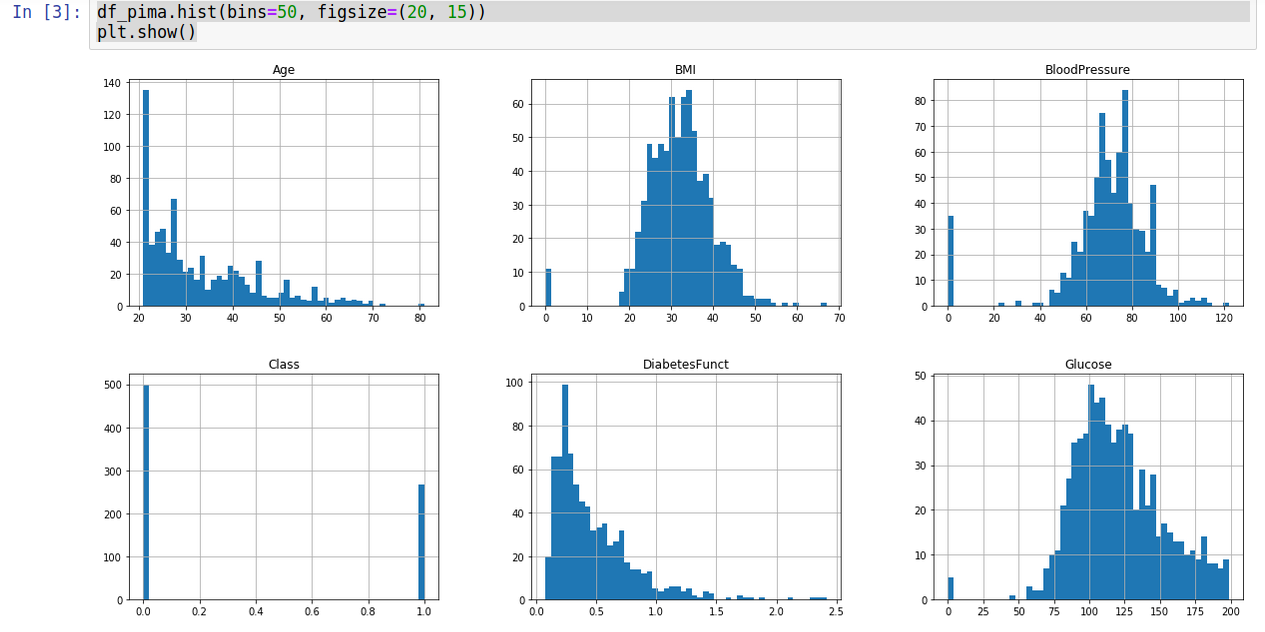
df\_pima=pd.read\_csv('./Datasets/PimaIndiansDiabetes.csv')

df\_pima.head(10)

**# Data Mining and Analysis**

df\_pima.hist(bins=50, figsize=(20, 15))

plt.show()



**# Data Pre processing**

# Determining outliers

df\_pima.isna().sum()

**# Replacing 0s and outliers with NaNs**

df\_pima['Glucose'] = df\_pima['Glucose'].replace(0, np.nan)

df\_pima['BloodPressure'] = df\_pima['BloodPressure'].replace(0, np.nan)

df\_pima['SkinThickness'] = df\_pima['SkinThickness'].replace(0, np.nan)

df\_pima['Insulin'] = df\_pima['Insulin'].replace(0, np.nan)

df\_pima['BMI'] = df\_pima['BMI'].replace(0, np.nan)

df\_pima['DiabetesFunct'] = df\_pima['DiabetesFunct'].replace(0, np.nan) # useless cuz DiabetesFuct=0 for no instance

df\_pima['Age'] = df\_pima['Age'].replace(0, np.nan)# useless cuz Age=0 for no instance

total = df\_pima.isnull().sum().sort\_values(ascending=False)

percent = (df\_pima.isnull().sum()/df\_pima.isnull().count()).sort\_values(ascending=False)

missing\_data = pd.concat([total, percent], axis=1, keys=['Total', 'Percent'])

f, ax = plt.subplots(figsize=(10, 5))

plt.xticks(rotation='90')

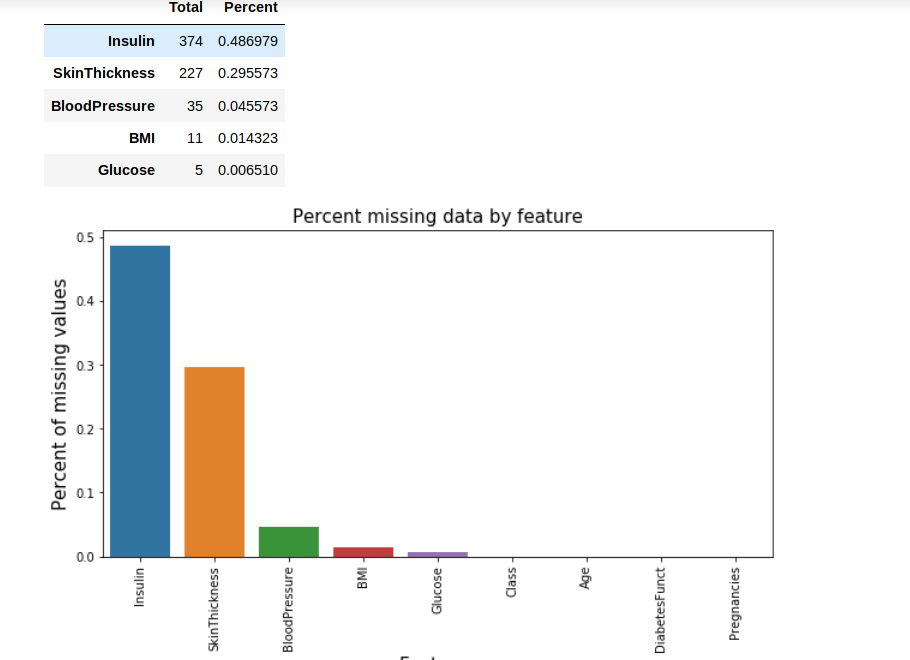
sns.barplot(x=missing\_data.index, y=missing\_data['Percent'])

plt.xlabel('Features', fontsize=15)

plt.ylabel('Percent of missing values', fontsize=15)

plt.title('Percent missing data by feature', fontsize=15)

missing\_data.head()



**# Replacing NaNs with mean values**

df\_pima['Glucose'].fillna(df\_pima['Glucose'].mean(), inplace=True)

df\_pima['BloodPressure'].fillna(df\_pima['BloodPressure'].mean(), inplace=True)

df\_pima['SkinThickness'].fillna(df\_pima['SkinThickness'].mean(), inplace=True)

df\_pima['Insulin'].fillna(df\_pima['Insulin'].mean(), inplace=True)

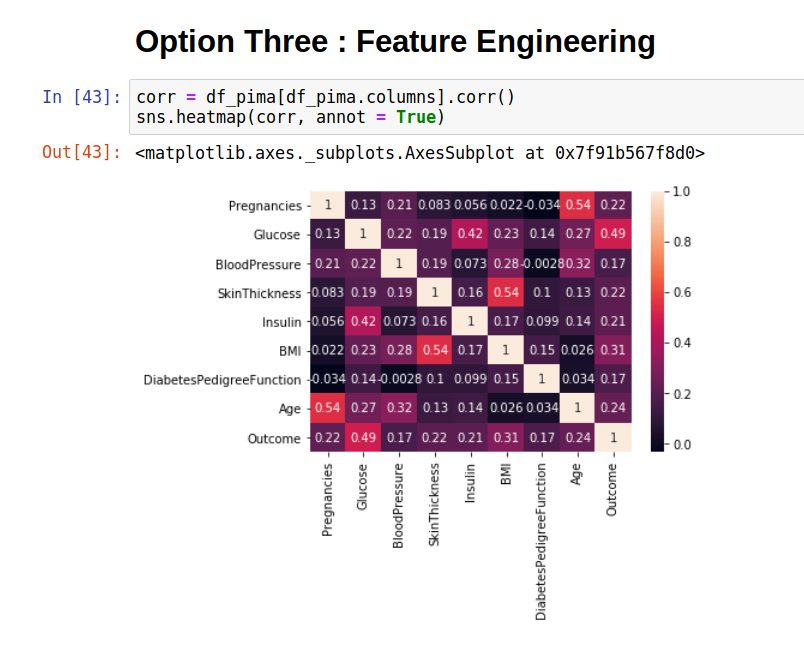
df\_pima['BMI'].fillna(df\_pima['BMI'].mean(), inplace=True)



**# Determining correlation between features**

corr = df\_pima[df\_pima.columns].corr()

sns.heatmap(corr, annot = True)



**# Feature selection**

# No need to eliminate any features cuz correlation between any of them is not too much

PearsonThreshold = 0.9 #change the Pearson Coefficient threshold as required

columns = np.full((corr.shape[0],), True, dtype=bool)

for i in range(corr.shape[0]):

for j in range(i+1, corr.shape[0]):

if corr.iloc[i,j] >= PearsonThreshold:

if columns[j]:

columns[j] = False

selected\_columns = df\_pima.columns[columns]

df\_pima = df\_pima[selected\_columns]

**# Selecting Features**

X = pd.DataFrame(data = df\_pima, columns = ["Pregnancies","Glucose","SkinThickness","BMI","Age","Insulin","DiabetesFunct"])

y = pd.DataFrame(data = df\_pima, columns = ["Class"])

**# Feature Scaling**

scaler = StandardScaler()

X = scaler.fit\_transform(X)

**# Logistic Regression**

from sklearn.linear\_model import LogisticRegression

model = LogisticRegression(C=0.1,penalty='l2')

model.fit(X\_train,y\_train)

acc\_log\_reg = model.score(X\_test,y\_test) \*100

print(acc\_log\_reg)

actual = y\_test

predict = model.predict(X\_test)

result = confusion\_matrix(actual , predict)

print('confusion matrix => \n',result)

print('\nAccuracy score => ',accuracy\_score(actual , predict))

print('Classification report => \n',classification\_report(actual,predict))

**# Naive Bayes**

from sklearn.naive\_bayes import GaussianNB

NBmodel = GaussianNB()

NBmodel.fit(X\_train , y\_train)

acc\_NB = NBmodel.score(X\_test,y\_test) \*100

print(acc\_NB)

actual\_naive = y\_test

predict\_naive = NBmodel.predict(X\_test)

result\_naive = confusion\_matrix(actual\_naive , predict\_naive)

print('confusion matrix => \n',result\_naive)

print('\nAccuracy score => ',accuracy\_score(actual\_naive , predict\_naive))

print('Classification report => \n',classification\_report(actual\_naive,predict\_naive))

**# Decision Trees**

from sklearn.tree import DecisionTreeClassifier

DTmodel = DecisionTreeClassifier(criterion='gini')

DTmodel.fit(X\_train, y\_train)

acc\_DT = DTmodel.score(X\_test,y\_test) \*100

print(acc\_DT)

actual\_DT = y\_test

predict\_DT = DTmodel.predict(X\_test)

result\_DT = confusion\_matrix(actual\_DT , predict\_DT)

print('confusion matrix => \n',result\_DT)

print('\nAccuracy score => ',accuracy\_score(actual\_DT , predict\_DT))

print('Classification report => \n',classification\_report(actual\_DT,predict\_DT))

**# Support Vector machine**

from sklearn.svm import SVC

SVMmodel = SVC(C=0.3,kernel='rbf',gamma=0.43)

SVMmodel.fit(X\_train , y\_train)

acc\_SVM = SVMmodel.score(X\_test,y\_test) \*100

print(acc\_SVM)

actual\_svm = y\_test

predict\_svm = SVMmodel.predict(X\_test)

result\_svm = confusion\_matrix(actual\_svm , predict\_svm)

print('confusion matrix => \n',result\_svm)

print('\nAccuracy score => ',accuracy\_score(actual\_svm , predict\_svm))

print('Classification report => \n',classification\_report(actual\_svm,predict\_svm))

**# K Nearest Neighbours**

from sklearn.neighbors import KNeighborsClassifier

KNNmodel = KNeighborsClassifier(n\_neighbors=13,)

KNNmodel.fit(X\_train , y\_train)

acc\_KNN = KNNmodel.score(X\_test,y\_test) \*100

print(acc\_KNN)

actual\_knn = y\_test

predict\_knn = KNNmodel.predict(X\_test)

result\_knn = confusion\_matrix(actual\_knn , predict\_knn)

print('confusion matrix => \n',result\_knn)

print('\nAccuracy score => ',accuracy\_score(actual\_knn , predict\_knn))

print('Classification report => \n',classification\_report(actual\_knn,predict\_knn))

**# Which model is the best ?**

results = pd.DataFrame({

'Model': ['Logistic Regression', 'Naive Bayes','Decision Tree' ,

'Support Vector Machine', 'K Nearest Neighbour'],

'Score': [acc\_log\_reg, acc\_NB,

acc\_DT, acc\_SVM , acc\_KNN]})

result\_df = results.sort\_values(by='Score', ascending=False)

result\_df = result\_df.set\_index('Score')

result\_df.head(9)

