A PROJECT REPORT

ON

DIABETES PREDICTION MODEL USING MACHINE LEARNING

Submitted in partial fulfilment of the requirement for the award of

Diploma in Computer Engineering

By

M. Abhinav Reddy : 18047-CM-028

K. Vinay Kumar : 18047-CM-026

Y. Yashwanth : 18047-CM-056

Thakur Harsh Raj Singh : 18047-CM-049

Under the guidance of

Mr. B. VAJRAIAH,
M.Tech., MISTE
Senior Lecturer in Computer Engineering

COMPUTER ENGINEERING SECTION
S.G.M. GOVT. POLYTECHNIC,
ABDULLAPURMET. R.R. DIST.

S.G.M. GOVT. POLYTECHNIC,
ABDULLAPURMET, R.R. DIST.

COMPUTER ENGINEERING SECTION

C E R T I F I C A T E

This is to certify that the project report entitled "Diabetes prediction model using machine learning" submitted by the team members in partial fulfilment for the award of Diploma in Computer Engineering by the State Board of Technical Education & Training, T.S., during academic year 2020-2021.

The results presented in this project have been verified and found satisfactory.

Project Guide	Head of Section
B. VAJRAIAH, M.Tech., MISTE Senior Lecturer, Computer Engineering	SUSHEEL KUMAR JOSHI,M.Tech Head of Section, Computer Engineering

External Examiner Principal

ACKNOWLEDGEMENT

We wish to record our deep sense of gratitude to our respected project guide Mr. B. Vajraiah, Senior Lecturer in Computer Engineering for the invaluable guidance, kind encouragement, personal affection and for having spared valuable time at every stage of preparation of this project.

We express our sincere thanks to Sri Ch.V. Krishna Rao, Principal, for providing us the conductive environment for carrying through our academic schedules and project with ease.

We are grateful to Sri Susheel Kumar Joshi, Head of Computer Engineering Section for providing right suggestions at every phase of our project.

Finally, we thank our staff members Mrs. G. Sree Madhuri, Lecturer in Computer Engineering, Mrs. P. Alekya Lecturer in Computer Engineering and Mrs. K. Sunitha, Lecturer in Computer Engineering for their co-operation and generous hospitality in completing this project.

Above all, we are indebted to our parents, who have brought us upto this stage with their care, love and affection

- M. Abhinav Reddy
- K. Vinay Kumar
- Y. Yashwanth
- Thakur Harsh Raj Singh

INDEX

INTRODUCTION

DIABETES PREDICTION MODEL

AIM

The aim of this project is to take an input of every feature described, from the user and use those values to predict whether the user/person whose details have been entered is diabetic or not.

MOTIVATION

People having diabetes have high risk of diseases like heart disease, kidney disease, stroke, eye problem, nerve damage, etc. Current practice in hospital is to collect required information for diabetes diagnosis through various tests and appropriate treatment is provided based on diagnosis. Big Data Analytics plays an significant role in healthcare.

Diabetes Mellitus is among critical diseases and lots of people are suffering from this disease. Age, obesity, lack of exercise, hereditary diabetes, living style, bad diet, high blood pressure, etc. can cause Diabetes Mellitus. People having diabetes have high risk of diseases like heart disease, kidney disease, stroke, eye problem, nerve damage, etc. Current practice in hospital is to collect required information for diabetes diagnosis through various tests and appropriate treatment is provided based on diagnosis. Big Data Analytics plays an significant role in healthcare industries. Healthcare industries have large volume databases. Using big data analytics one can study huge datasets and find hidden information, hidden patterns to discover knowledge from the data and predict outcomes accordingly. In existing method, the classification and prediction accuracy is not so high. In this paper, we have proposed a diabetes prediction model for better classification of diabetes which includes few external factors responsible for diabetes along with regular factors like Glucose, BMI, Age, Insulin, etc. Classification accuracy is boosted with new dataset compared to existing dataset. Further with imposed a pipeline model for diabetes prediction intended towards improving the accuracy of classification.

OVERVIEW

The project is based on a machine learning model i.e., Decision tree classifier, where a set of values are taken from the user to predict whether he/she is diabetic. The model then displays the output in either one of the following forms:

1) 0 (Zero) – The user/person whose details have been entered are not diabetic.

2) 1 (one) – The user/person whose details have been entered are diabetic.

The machine learning algorithm used in this project is Decision Tree Classifier which is a tree-based model.

The features that the user have to enter in order to predict are:

- **Number of Pregnancies**
- **BMI**
- **Diabetic Pedigree Function**
- **Glucose**
- **Blood Pressure**
- **Skin Thickness**
- **Age**

NUMBER OF PREGNANCIES

The user has to enter the number of pregnancies in the past. Zero has to be entered for Male patients

BMI

Body Mass Index (BMI) is a person's weight in kilograms divided by the square of height in meters. A high BMI can indicate high body fatness. BMI screens for weight categories that may lead to health problems, but it does not diagnose the body fatness or health of an individual.

GLUCOSE

A blood sugar level less than 140 mg/dL (7.8 mmol/L) is normal. A reading of more than 200 mg/dL (11.1 mmol/L) after two hours indicates diabetes. A reading between 140 and 199 mg/dL (7.8 mmol/L and 11.0 mmol/L) indicates prediabetes.

BLOOD PRESSURE

A normal blood pressure level is less than 120/80 mmHg. No matter your age, you can take steps each day to keep your blood pressure in a healthy range.

DIABETIC PEDIGREE FUNCTION

Diabetes pedigree function is a function which scores likelihood of diabetes based on family history

SKIN THICKNESS

Skin is the largest organ of the body. It has an area of 2 square metres (22 square feet) in adults, and weighs about 5 kilograms. The thickness of skin varies from 0.5mm thick on the eyelids to 4.0mm thick on the heels of your feet.

AGE

Age refers to the value: current year – year of birth

LIBRARIES

The following are the main libraries used in this project:

1) NumPy
2) Pandas
3) Matplotlib
4) Seaborn
5) Scikit-Learn
6) Pickle-mixin
7) Flask

Other libraries used in the project are listed below:

argon2-cffi==20.1.0
async-generator==1.10
attrs==20.3.0
backcall @ file:///home/ktietz/src/ci/backcall_1611930011877/work
bleach==3.3.0
certifi==2020.12.5
cffi==1.14.5
click==8.0.1
colorama @ file:///tmp/build/80754af9/colorama_1607707115595/work
cycler==0.10.0
dataclasses==0.8
decorator @ file:///home/ktietz/src/ci/decorator_1611930055503/work
defusedxml==0.7.1
entrypoints==0.3
Flask==2.0.1
imbalanced-learn==0.8.0
imblearn==0.0
importlib-metadata==3.10.0
ipykernel @ file:///C:/ci/ipykernel_1596206775157/work/dist/ipykernel-5.3.4-py3-none-any.whl
ipython @ file:///C:/ci/ipython_1593446240034/work
ipython-genutils @ file:///tmp/build/80754af9/ipython_genutils_1606773439826/work
ipywidgets==7.6.3
itsdangerous==2.0.1
jedi==0.17.0
Jinja2==3.0.1
joblib==1.0.1
jsonschema==3.2.0
jupyter==1.0.0
jupyter-client @ file:///tmp/build/80754af9/jupyter_client_1601311786391/work
jupyter-console==6.4.0
jupyter-core @ file:///C:/ci/jupyter_core_1612213536848/work
jupyterlab-pygments==0.1.2
jupyterlab-widgets==1.0.0
kiwisolver==1.3.1
MarkupSafe==2.0.1
matplotlib==3.3.4
mistune==0.8.4
nbclient==0.5.3
nbconvert==6.0.7
nbformat==5.1.2

nest-asyncio==1.5.1
notebook==6.3.0
numpy==1.19.5
packaging==20.9
pandas==1.1.5
pandocfilters==1.4.3
parso @ file:///tmp/build/80754af9/parso_1607623074025/work
pickle-mixin==1.0.2
pickleshare @ file:///tmp/build/80754af9/pickleshare_1606932040724/workNote: you may need to restart the ke
rnel to use updated packages.

Pillow==8.1.2
prometheus-client==0.9.0
prompt-toolkit @ file:///tmp/build/80754af9/prompt-toolkit_1616415428029/work
pycparser==2.20
Pygments @ file:///tmp/build/80754af9/pygments_1615143339740/work
pyparsing==2.4.7
pyrsistent==0.17.3
python-dateutil @ file:///home/ktietz/src/ci/python-dateutil_1611928101742/work
pytz==2021.1
pywin32==227
pywinpty==0.5.7
pyzmq==20.0.0
qtconsole==5.0.3
QtPy==1.9.0
scikit-learn==0.24.2
scipy==1.5.4
seaborn==0.11.1
Send2Trash==1.5.0
six @ file:///C:/ci/six_1605187303045/work
terminado==0.9.4
testpath==0.4.4
threadpoolctl==2.1.0
tornado @ file:///C:/ci/tornado_1606942379977/work
traitlets==4.3.3
typing-extensions==3.7.4.3
wcwidth @ file:///tmp/build/80754af9/wcwidth_1593447189090/work
webencodings==0.5.1
Werkzeug==2.0.1
widgetsnbextension==3.5.1

DATA PREPARATION

The data is used in this project is taken from Kaggle and the name of the dataset is diabetes prediction dataset. It consists of 9 features and 767 columns in total.

A glimpse of the data set is given below:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
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

The data is imported into the python notebook by the following statement using pandas library:

df (variable) = pd.read_csv(“filename.csv”)

where,

df – variable

pd – pandas object

csv – file type

filename – It is the name of the file

NOTE: If python notebook directory and the datafile directory is not same, then we will have to mention the path of the file which contains data in the python notebook.

PANDAS LIBRARY

For those of you who are getting started with Machine learning, just like me, would have come across Pandas, *the data analytics library*. In the rush to understand the gimmicks of ML, we often fail to notice the importance of this library. But soon you will hit a roadblock where you would need to play with your data, clean and perform data transformations before feeding it into your ML model.

Why do we need this blog when there are already a lot of documentation and tutorials? Pandas, unlike most python libraries, has a steep learning curve. The reason is that you need to understand your data well in order to apply the functions appropriately. Learning Pandas syntactically is not going to get you anywhere. Another problem with Pandas is that there is that there is more than one way to do things. Also, when I started with Pandas it's extensive and elaborate documentation was overwhelming. I checked out the cheat sheets and that scared me even more.

In this blog, I am going to take you through Pandas functionalities by cracking specific use cases that you would need to achieve with a given data.

Setup and Installation

Before we move on with the code for understanding the features of Pandas, let's get Pandas installed in your system. I advise you to create a virtual environment and install Pandas inside the virtualenv.

Create virtualenv

virtualenv -p python3 venv
source venv/bin/activate

Install Pandas

pip install pandas

Jupyter Notebook

If you are learning Pandas, I would advise you to dive in and use a jupyter notebook for the same. The visualization of data in jupyter notebooks makes it easier to understand what is going on at each step.

pip install jupyter
jupyter notebook

Jupyter by default runs in your system-wide installation of python.

Sample Data

We created a simple purchase order data. It comprises of sales data of each salesperson of a company over countries and their branches at different regions in each country,

Country	Region	Requester	Date of Purchase	Total	Quantity
India	North	John	2/7/2018 0:00:00	100,000.00	567
US	North	Bill	12/18/2017 0:00:00	120,000.00	3000
UK	North	Thomas	3/15/2016 0:00:00	140,000.00	345
Australia	East	John	1/18/2012 0:00:00	160,000.00	1000
Africa	East	Bill	12/6/2014 0:00:00	180,000.00	123
Singapore	East	Thomas	4/18/2018 0:00:00	200,000.00	1000
Mylasia	West	John	2/28/2010 0:00:00	1,000,000.00	7890
India	West	Bill	6/27/2016 0:00:00	240,000.00	200
US	West	Thomas	8/9/2011 0:00:00	26,000,000.00	1000
UK	North	John	5/14/2010 0:00:00	100,000.00	1000
Australia	North	Bill	3/13/2011 0:00:00	120,000.00	567
Africa	North	Thomas	1/31/2015 0:00:00	140,000.00	1000
Singapore	East	John	4/14/2013 0:00:00	160,000.00	892
Mylasia	East	Bill	2/8/2016 0:00:00	180,000.00	444
India	East	Thomas	10/13/2018 0:00:00	200,000.00	90
US	West	John	9/12/2011 0:00:00	220,000.00	90
UK	West	Bill	11/14/2017 0:00:00	240,000.00	90
Australia	West	Thomas	8/19/2011 0:00:00	260,000.00	90
Africa	North	John	4/22/2013 0:00:00	140,000.00	85
Singapore	North	Bill	3/12/2016 0:00:00	150,000.00	85

Load data into Pandas

With Pandas, we can load data from different sources. Few of them are loading from CSV or a remote URL or from a database. The loaded data is stored in a Pandas data structure called DataFrame. DataFrame's are usually refered by the variable name df . So, anytime you see df from here on you should be associating it with Dataframe.

From CSV File

```
import pandas
df = pandas.read_csv("path_to_csv")
```

From Remote URL

You can pass a remote URL to the CSV file in `read_csv`.

```
import pandas
df = pandas.read_csv("remote/url/path/pointing/to/csv")
```

From DB

In order to read from Database, read the data from DB into a python `list` and use `DataFrame()` to create one

```
db = # Create DB connection object
cur = db.cursor()
cur.execute("SELECT * FROM <TABLE>")
df = pd.DataFrame(cur.fetchall())
```

Each of the above snippets reads data from a source and loads it into Pandas' internal data structure called `DataFrame`

Understanding Data

Now that we have the Dataframe ready let's go through it and understand what's inside it

```
# 1. shows you a gist of the data
df.head()# 2. Some statistical information about your data
df.describe()# 3. List of columns headers
df.columns.values
```

A gist of the data

```
In [6]: df.head()
```

Out[6]:

	Country	Region	Requester	Date of Purchase	Total	Quantity
0	India	North	John	9/16/2016 0:00:00	100000	567
1	US	North	Bill	10/19/2018 0:00:00	120000	3000
2	UK	North	Thomas	6/10/2014 0:00:00	140000	345
3	Australia	East	John	11/23/2010 0:00:00	160000	1000
4	Africa	East	Bill	2/17/2010 0:00:00	180000	123

Know your columns

```
In [7]: df.columns.values
```

```
Out[7]: array(['Country', 'Region', 'Requester', 'Date of Purchase', 'Total',
       'Quantity'], dtype=object)
```

```
In [8]: df.describe()
```

Out[8]:

	Total	Quantity
count	2.000000e+01	20.000000
mean	1.502500e+06	977.900000
std	5.769280e+06	1761.923497
min	1.000000e+05	85.000000
25%	1.400000e+05	90.000000
50%	1.700000e+05	505.500000
75%	2.250000e+05	1000.000000
max	2.600000e+07	7890.000000

Pick & Choose your Data

Now that we have loaded our data into a DataFrame and understood its structure, let's pick and choose and perform visualizations on the data. When it comes to selecting your data, you can do it with both Indexesor based on certain conditions. In this section, let's go through each one of these methods.

Indexes

Indexes are labels used to refer to your data. These labels are usually your column headers. For eg., Country, Region, Quantity Etc.,

Selecting Columns

1. Create a list of columns to be selected

columns_to_be_selected = ["Total", "Quantity", "Country"]# 2. Use it as an index to the DataFrame

df[columns_to_be_selected]# 3. Using loc method

df.loc[columns_to_be_selected]

	Total	Quantity	Country
0	100000	567	India
1	120000	3000	US
2	140000	345	UK
3	160000	1000	Australia
4	180000	123	Africa
5	200000	1000	Singapore
6	1000000	7890	Mylasia
7	240000	200	India
8	26000000	1000	US
9	100000	1000	UK
10	120000	567	Australia

Selecting Rows

Unlike the columns, our current DataFrame does not have a label which we can use to refer the row data. But like arrays, DataFrame provides numerical indexing(0, 1, 2…) by default.

 # 1. using numerical indexes - iloc

df.iloc[0:3, :]**# 2. using labels as index - loc**

row_index_to_select = [0, 1, 4, 5]

df.loc[row_index_to_select]

```
In [38]: df.loc[[0, 1, 4, 5]]
```

Out[38]:

	Country	Region	Requester	Date of Purchase	Total	Quantity
0	India	North	John	2016-09-16	100000	567
1	US	North	Bill	2018-10-19	120000	3000
4	Africa	East	Bill	2010-02-17	180000	123
5	Singapore	East	Thomas	2017-08-14	200000	1000

NUMPY

1. Numpy Definition: a Python library used for scientific computing. It can also be used as an efficient multi-dimensional container for data.

2. Importing Numpy:

```
In [1]: import numpy
In [2]: import numpy as np
```

In the first line, we directly call on the library by referencing to it as " numpy. " For example, creating an array: `numpy.array([1, 2, 3])` .

In the second line, we make reference to the library as "np" instead of "numpy." For example, creating an array: `np.array([1, 2, 3])` . In this tutorial, we will be using this method.

3. Numpy vs. Lists: Numpy and Lists are similar to each other in the sense that they can both store data, be indexed and be iterated. However, Numpy uses less memory, is faster, and more convenient than Lists. Also, we cannot perform calculations (add, subtract, multiply, divide and exponentiation) on Python Lists but we can on Numpy Arrays.

VISUALIZATION LIBRARIES

MATPLOTLIB

Matplotlib is a cross-platform, data visualization and graphical plotting library for Python and its numerical extension NumPy. As such, it offers a viable opensource alternative to MATLAB. Developers can also use matplotlib's APIs (Application Programming Interfaces) to embed plots in GUI applications.

A Python matplotlib script is structured so that a few lines of code are all that is required in most instances to generate a visual data plot. The matplotlib scripting layer overlays two APIs:

The pyplot API is a hierarchy of Python code objects topped by *matplotlib.pyplot*

An OO (Object-Oriented) API collection of objects that can be assembled with greater flexibility than pyplot. This API provides direct access to Matplotlib's backend layers.

Matplotlib and Pyplot in Python

The pyplot API has a convenient MATLAB-style stateful interface. In fact, matplotlib was originally written as an open source alternative for MATLAB. The OO API and its interface is more customizable and powerful than pyplot, but considered more difficult to use. As a result, the pyplot interface is more commonly used, and is referred to by default in this article.

Understanding matplotlib's pyplot API is key to understanding how to work with plots:

matplotlib.pyplot.figure: Figure is the top-level container. It includes everything visualized in a plot including one or more *Axes*.

matplotlib.pyplot.axes: Axes contain most of the elements in a plot: *Axis, Tick, Line2D, Text,* etc., and sets the coordinates. It is the area in which data is plotted. Axes include the X-Axis, Y-Axis, and possibly a Z-Axis, as well.

Installing Matplotlib

Matplotlib and its dependencies can be downloaded as a binary (pre-compiled) package from the Python Package Index (PyPI), and installed with the following command:

python -m pip install matplotlib

Matplotlib is also available as uncompiled source files. Compiling from source will require your local system to have the appropriate compiler for your OS, all dependencies, setup scripts, configuration files, and patches available. This can result in a fairly complex installation. Alternatively, consider using the ActiveState Platform to automatically build matplotlib from source and package it for your OS.

SEABORN

Seaborn is a data visualization library built on top of matplotlib and closely integrated with pandas data structures in Python. Visualization is the central part of Seaborn which helps in exploration and understanding of data.

Seaborn offers the following functionalities:

1. Dataset oriented API to determine the relationship between variables.
2. Automatic estimation and plotting of linear regression plots.
3. It supports high-level abstractions for multi-plot grids.
4. Visualizing univariate and bivariate distribution.

These are only some of the functionalities offered by Seaborn, there are many more of them, and we can explore all of them here.

To initialize the Seaborn library, the command used is:

```
import seaborn as sns
```

Using Seaborn we can plot wide varieties of plots like:

1. Distribution Plots
2. Pie Chart & Bar Chart
3. Scatter Plots
4. Pair Plots
5. Heat maps


Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age
Outcome

MACHINE LEARNING ALGORITHM

DECISION TREE CLASSIFIER

What are they?

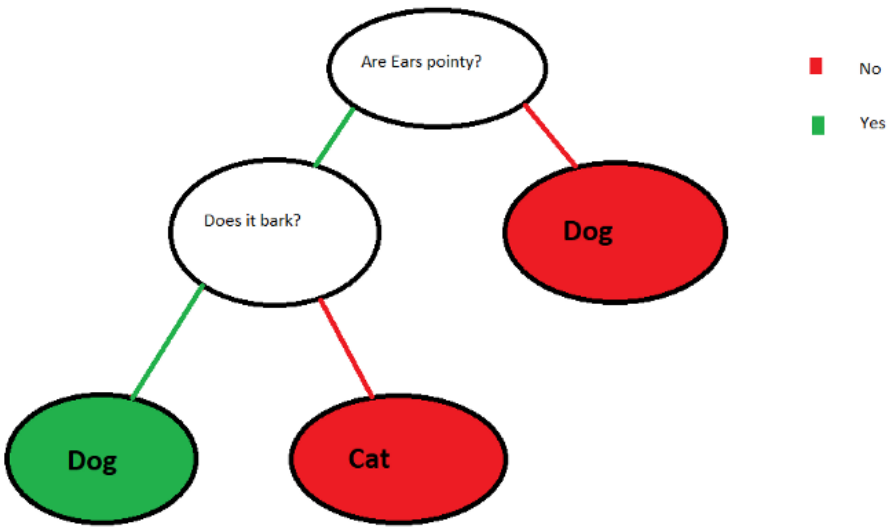
Decision trees are a tree algorithm that split the data based on certain decisions. Look at the image below of a very simple decision tree. We want to decide if an animal is a cat or a dog based on 2 questions.

Are the ears pointy?

Does the animal bark?

We can answer each question and depending on the answer, we can classify the animal as either a dog or a cat. The red lines represent the answer “NO” and the green line, “YES”.

This way the decision process can be laid out like a tree. The question nodes are called **root nodes** and the answer/terminal nodes are called **leaf nodes**.



Simple Decision Tree. Image by the author.

Which division comes first?

When we have a table of data with multiple columns, Decision trees split the result based on columns. For the above DecisionTree, if we had a table like shown below, how do we decide which split comes first? Do we split first based on pointy ears or barking?

	are ears points?	does it bark?	label
0	Yes	No	Cat
1	No	Yes	Dog
2	No	Yes	Dog
3	No	Yes	Dog
4	Yes	No	Cat
5	Yes	Yes	Dog
6	No	Yes	Dog
7	Yes	Yes	Dog
8	Yes	No	Cat
9	No	No	Cat

Cat or dog table. Image by the author.

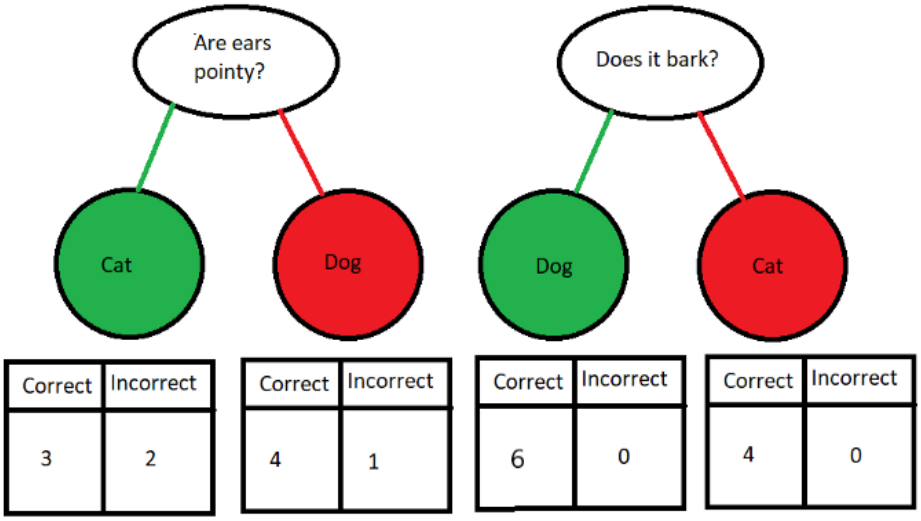
We can use certain metrics to decide which column is the best classifier. There are 3 popular metrics that are used.

Gini Impurity.

Information Gain.

Variance reduction.

Gini impurity as far as I have seen is the most popular, so let's take a look at it.
Classifying based on a single variable, we have the following **stumps** (single node trees).



Decisions and correctness. Image by the author.

Look at the above decisions and the number of correct and incorrect predictions they made. At each leaf node, the Gini impurity is calculated as below:

Gini at leaf = 1 -("yes" probability)² -("No" probability)²

So Gini for the first Node's leaves will be:
Leaf 1: $1 - (3/5)^2 - (2/5)^2 = 0.48$
Leaf 2: $1 - (4/5)^2 - (1/5)^2 = 0.32$

The Gini impurity for the entire node is the weighted sum of the Gini at each leaf node. So the Gini for the first node is:
$(0.48)*(5/10) + (0.32)*(5/10) = 0.4$

The weights for the sum, in this case, is the same as both classes contain 5 elements each. This is not usually the case and that is when the weighted sum of Gini is important.

Similarly, the Gini impurity for the second node will be: 0

As the Gini Impurity is lower for the second node, that should be our first split. This way, we decide the order in which features are used to split data.

How to split based on continuous variables?

How do we split continuous variables or multicategory variables? Continuous variables are a little trickier, the key idea is to arrange the continuous variable in order, and at each data point, take the average between the previous and the current data point and split the data based on

the average, and calculate the Gini impurity. After going through the entire data, select the split with least Gini Impurity.

Example: Consider a list of heights and a classification as tall or short. At each data point (height), we split based on average of the current and previous data. Say the average at step i is 150 cm, then we assume all data less than 150 cms is short and greater than 150cm is tall. Then we calculate the gini impurity based on how many correct and incorrect predictions we made. Then we choose another split and check the gini impurity of that split, till we find the split with lowest impurity.

A similar strategy is used for categorical variables with more than 2 classes.

*If this explanation is a little confusing, or you need further clarification, watch this amazing video by **Statquest** on youtube.*

Decision Trees for Regression

So far we have only discussed decision trees for classification, but they can also be used for regression. The process for selecting decision-splits and important features is very similar to that of classification trees. The difference is how the impurity is calculated and how the output is predicted.

Consider a tree with 1 node (also called stump) and 2 leaf nodes. After the data is split, all the values of each of the leaf nodes are averaged, and that average is the prediction for that leaf node.

The difference between the prediction of each data point and the actual value is called the **residual**. The sum of squares of all the residuals after the split is the impurity metric. The split with the lowest impurity or the split with the lowest sum of squared residuals is chosen.

Random Forests

The problem with Decision trees is that they overfit the data. They learn to split the training data to lower the metric but end up doing so in such a way that it overfits the data and the model does poorly on unseen data.

There are 2 main ideas to fix the overfitting of Decision Trees.

Bootstrapping

Ensembling

Bootstrapping

Bootstrapping can be thought of as a sampling method. The main idea is to randomly select a subset of the data, on which the decision tree is trained. This makes the Decision tree overfit less (because it sees lesser data), however, it does nothing to increase the validation accuracy. This is only part of the solution.

Ensembling

We reduced overfitting by bootstrapping, and now our tree fits only on a subset of data, but how does this help?

This is where the second part of Random Forests comes in. The idea of random forests is to make the decision tree a weaker predictor but to have many predictors and consider a central tendency (mean-average or mode-highest votes) of the set of predictors.

Random forests combine bootstrapping and ensembling. The data is randomly sampled, and a decision tree is trained on each of the sets of randomly sampled data. This way there are many decision trees each trained on a different random subset of the data.

As no tree has seen the entire data, they cannot overfit the data, and as there are a lot of trees, the errors in prediction are driven lower by averaging the outputs of all the trees.

This way with ensembling and bootstrapping, random forests achieve far superior performance as compared to any single Decision tree.

Gradient boosted Decision Trees

Gradient boosting has a similar approach to random forests, in that it chooses weaker predictors, but multiple in number to both avoid overfitting and achieve top tier performance. Gradient boosted trees can be used for both classification and regression.

Mechanism

Consider a regression problem. Gradient boosted trees have the following approach.

Step one is to always predict the average of all the target values. This is a baseline prediction of the target value. Once we have a baseline, each target variable is subtracted by the baseline prediction, this gives the residuals for step one.

residuals = target — prediction

Step two is to fit a decision tree for the residuals, where the output of each leaf node is the average of residuals in the leaf node. Now to predict the target, we scale the output by a value called learning rate (lr) and add this scaled value to the previous prediction. The sum is the new prediction, and we can repeat the above steps by calculating a new set of residuals.

$Pred_i = Pred_{(i-1)} + lr * output$

*Where **output** is the **average of residuals** in the leaf node*

The residual is analogous to the derivative of the loss function in neural networks. As the residual is the difference between the target and the prediction, if the residual is positive it means our prediction was lower than the target, and adding a scaled residual to our prediction will make it closer to the correct target. Similar idea for negative residuals.

When multiple steps of gradient boosted trees are stacked, the prediction will be remarkably close to the prediction, and the model will perform very well on unseen data as well.

MATHEMATICAL CONCEPTS USED IN THE PROJECT

CORRELATION

Correlation (to be exact Correlation in Statistic) is a measure of a mutual relationship between two variables whether they are causal or not. This degree of measurement could be measured on any kind of data type (Continous and Continous, Categorical and Categorical, Continous and Categorical).

Reasons Behind Correlation:

It may happen because of several reasons like:

1. Mutual dependence Between the variables: Both the variables may be mutually influencing each other so that neither can be designated as the cause and the other the effect.

When two variables(X and Y) affect each other mutually, we cannot say X is the cause or Y is the cause.

For Example, The price of a commodity is affected by demand and supply.

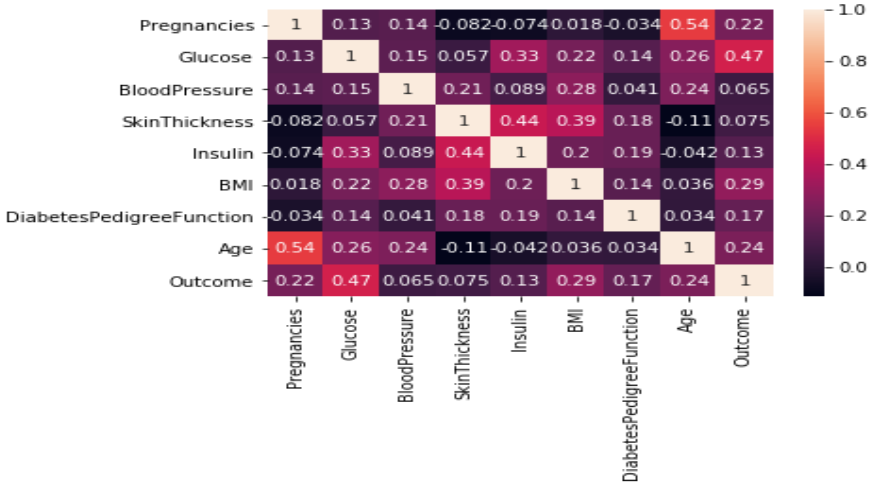
2. Due to pure chance: In a small sample, X and Y are highly correlated but in the universe X and Y are not correlated.

For Example, Correlation between income and weight of a person. This may be due to:

– Sampling fluctuations

– Bias of investigator in selecting the sample.

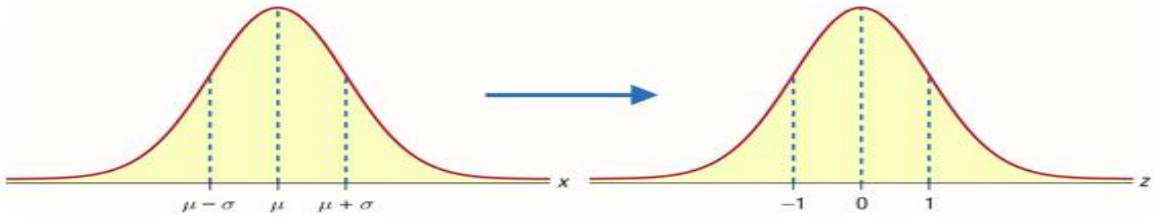
Such a relation is called a non-sense or spurious relation.



NORMAL DISTRIBUTION

Standard Normal Distribution is a special case of Normal Distribution when $\mu = 0$ and $\sigma = 1$. For any Normal distribution, we can convert it into Standard Normal distribution using the formula:

$$z = \frac{x - \mu}{\sigma}$$



To understand the importance of converting Normal Distribution into Standard Normal Distribution, let's suppose there are two students: *Ross and Rachel.* Ross scored 65 in the exam of paleontology and Rachel scored 80 in the fashion designing exam.

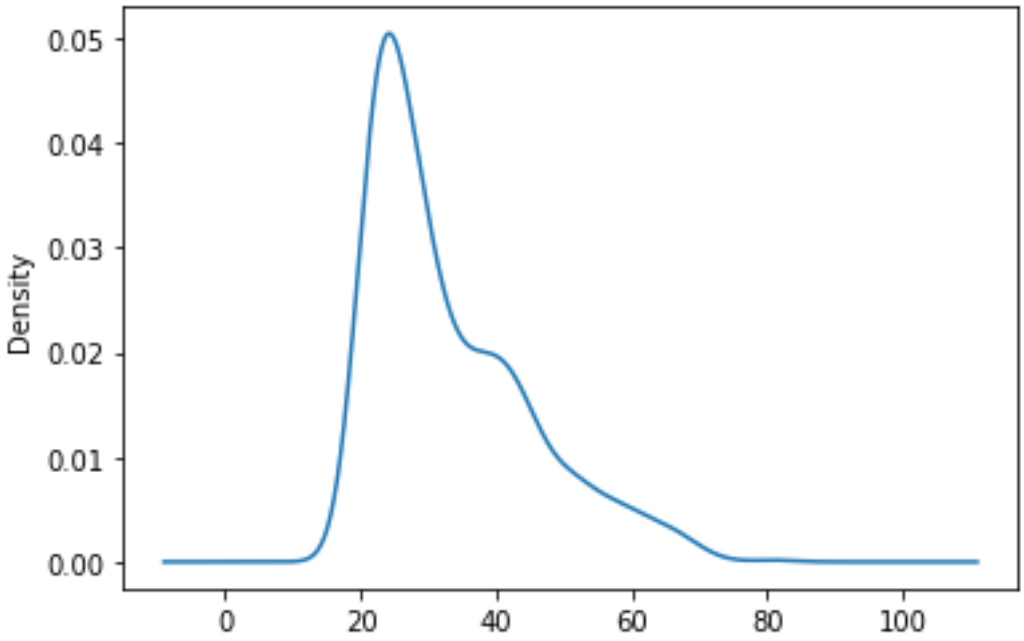
Can we conclude that Rachel scored better than Ross?

No**,** because the way people performed in paleontology may be different from the way people performed in fashion designing. The variability may not be the same here.

Suppose we write the following code :

```
df['Age'].plot(kind='kde')
```

OUTPUT;



DEPLOYEMENT OF MODEL IN LOCAL ENVIRONMENT

FLASK

Flask is a web application framework written in Python. It has multiple modules that make it easier for a web developer to write applications without having to worry about the details like protocol management, thread management, etc.

Flask gives is a variety of choices for developing web applications and it gives us the necessary tools and libraries that allow us to build a web application.

Installing Flask is simple and straightforward. Here, I am assuming you already have Python 3 and pip installed. To install Flask, you need to run the following command:

```
sudo apt-get install python3-flask
```

That's it! You're all set to dive into the problem statement take one step closer to deploying your machine learning model.

PICKLE FILE

It is used for serializing and de-serializing a Python object structure. Any object in python can be pickled so that it can be saved on disk. What pickle does is that it "serialises" the object first before writing it to file. Pickling is a way to convert a python object (list, dict,

etc.) into a character stream. The idea is that this character stream contains all the information necessary to reconstruct the object in another python script. So lets continue:

First of all you have to import it through this command:

```
import pickle
```

pickle has two main methods. The first one is `dump`, which dumps an object to a file object and the second one is `load`, which loads an object from a file object.

HTML

HTML was first created by Tim Berners-Lee, Robert Cailliau, and others starting in 1989. It stands for Hyper Text Markup Language.

Hypertext means that the document contains links that allow the reader to jump to other places in the document or to another document altogether. The latest version is known as HTML5.

A Markup Language is a way that computers speak to each other to control how text is processed and presented. To do this HTML uses two things: tags and attributes.

THE PROJECT CODE

THE NEXT PAGES WILL CONTAIN THE CODE TO PREPARE A MACHINE LEARNING MODEL BASED ON DECISION TREE CLASSIFIER TO PREDICT WHETHER A PATIENT IS DIABETIC OR NOT.

IT WILL INCLUDE ALL THE CONCEPTS MENTIONED ABOVE. THE CODE IS WRITTEN IN A JUPYTER NOTEBOOK.

WE CREATED A VIRTUAL ENVIRONMENT AND IT IS FOR THIS REASON YOU WILL BE SEEING **PIP INSTALL** LIBRARIES FOR EVERY LIBRARY IMPORTED.

In [1]:

```
pip install numpy
```

```
Requirement already satisfied: numpy in c:\users\raabh\anaconda3\envs\diabet
es\lib\site-packages (1.19.5)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\raabh\anaconda3\envs\diabe
tes\lib\site-packages (1.1.5)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\raabh\anac
onda3\envs\diabetes\lib\site-packages (from pandas) (2.8.1)
Requirement already satisfied: pytz>=2017.2 in c:\users\raabh\anaconda3\envs
\diabetes\lib\site-packages (from pandas) (2021.1)
Requirement already satisfied: numpy>=1.15.4 in c:\users\raabh\anaconda3\env
s\diabetes\lib\site-packages (from pandas) (1.19.5)
Requirement already satisfied: six>=1.5 in c:\users\raabh\anaconda3\envs\dia
betes\lib\site-packages (from python-dateutil>=2.7.3->pandas) (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

In [2]:

```
import numpy as np
import pandas as pd
```

In [3]:

```
df = pd.read_csv("Diabetes.csv")
```

In [ ]:

```
df.head()
```

In [5]:

```
df.tail()
```

Out[5]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunct
763	10	101	76	48	180	32.9	0.
764	2	122	70	27	0	36.8	0.
765	5	121	72	23	112	26.2	0.
766	1	126	60	0	0	30.1	0.
767	1	93	70	31	0	30.4	0.

In [11]:

```
df.isnull().sum()
```

Out[11]:

```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

In [19]:

```
df['Pregnancies'].isna().sum()
```

Out[19]:

```
0
```

In [20]:

```
df.columns
```

Out[20]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n',
       'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
      dtype='object')
```

In [22]:

```
for cols in df.columns:
    if df[cols].isna().any() == True:
        print(cols, "have missing values", df[cols].isna().sum(), "number of missing va

# We can observe that there are no missing values in the diabetes datset
```

In [25]:

```
df.shape
```

Out[25]:

```
(768, 9)
```

In [26]:

```
df.info()

# We can observe that there are 768 columns and 9 rows (8 featues and 1 label)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [27]:

```
df.describe()
```

Out[27]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabete
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

In [7]:

```
pip install seaborn
```

```
Requirement already satisfied: seaborn in c:\users\raabh\anaconda3\envs\diab
etes\lib\site-packages (0.11.1)
Requirement already satisfied: pandas>=0.23 in c:\users\raabh\anaconda3\envs
\diabetes\lib\site-packages (from seaborn) (1.1.5)
Requirement already satisfied: numpy>=1.15 in c:\users\raabh\anaconda3\envs
\diabetes\lib\site-packages (from seaborn) (1.19.5)
Requirement already satisfied: matplotlib>=2.2 in c:\users\raabh\anaconda3\e
nvs\diabetes\lib\site-packages (from seaborn) (3.3.4)
Requirement already satisfied: scipy>=1.0 in c:\users\raabh\anaconda3\envs\d
iabetes\lib\site-packages (from seaborn) (1.5.4)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\raabh\anacon
da3\envs\diabetes\lib\site-packages (from matplotlib>=2.2->seaborn) (2.8.1)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\raabh\anaconda3
\envs\diabetes\lib\site-packages (from matplotlib>=2.2->seaborn) (1.3.1)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
c:\users\raabh\anaconda3\envs\diabetes\lib\site-packages (from matplotlib>=
2.2->seaborn) (2.4.7)
Requirement already satisfied: pillow>=6.2.0 in c:\users\raabh\anaconda3\env
s\diabetes\lib\site-packages (from matplotlib>=2.2->seaborn) (8.1.2)
Requirement already satisfied: cycler>=0.10 in c:\users\raabh\anaconda3\envs
\diabetes\lib\site-packages (from matplotlib>=2.2->seaborn) (0.10.0)
Requirement already satisfied: six in c:\users\raabh\anaconda3\envs\diabetes
\lib\site-packages (from cycler>=0.10->matplotlib>=2.2->seaborn) (1.15.0)
Requirement already satisfied: pytz>=2017.2 in c:\users\raabh\anaconda3\envs
\diabetes\lib\site-packages (from pandas>=0.23->seaborn) (2021.1)
Note: you may need to restart the kernel to use updated packages.
```

In [8]:

```
pip install matplotlib
```

```
Requirement already satisfied: matplotlib in c:\users\raabh\anaconda3\envs\d
iabetes\lib\site-packages (3.3.4)
Requirement already satisfied: cycler>=0.10 in c:\users\raabh\anaconda3\envs
\diabetes\lib\site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in
c:\users\raabh\anaconda3\envs\diabetes\lib\site-packages (from matplotlib)
(2.4.7)
Requirement already satisfied: python-dateutil>=2.1 in c:\users\raabh\anacon
da3\envs\diabetes\lib\site-packages (from matplotlib) (2.8.1)
Requirement already satisfied: numpy>=1.15 in c:\users\raabh\anaconda3\envs
\diabetes\lib\site-packages (from matplotlib) (1.19.5)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\raabh\anaconda3
\envs\diabetes\lib\site-packages (from matplotlib) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\raabh\anaconda3\env
s\diabetes\lib\site-packages (from matplotlib) (8.1.2)
Requirement already satisfied: six in c:\users\raabh\anaconda3\envs\diabetes
\lib\site-packages (from cycler>=0.10->matplotlib) (1.15.0)
Note: you may need to restart the kernel to use updated packages.
```

In [28]:

```
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
```

In [29]:

```
df.corr()

# Correlation matrix allows us to view the dependencies of one feature on another. Corr
```

Out[29]:

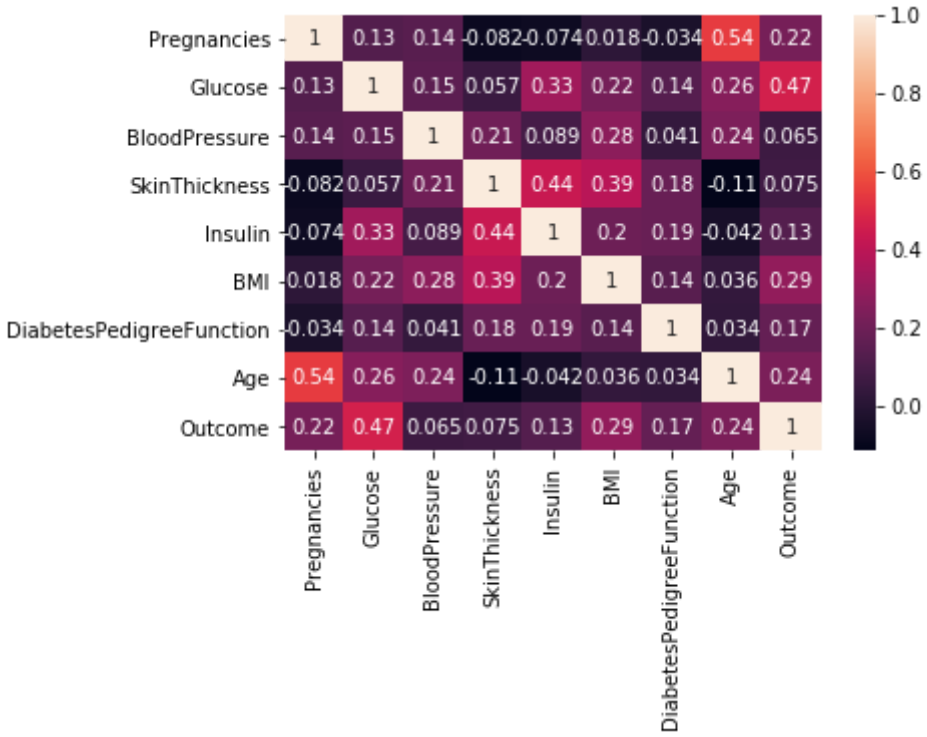
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.0
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.2
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.2
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.3
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.1
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.0
DiabetesPedigreeFunction	-0.033523	0.137337	0.041265	0.183928	0.185071	0.1
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.0
Outcome	0.221898	0.466581	0.065068	0.074752	0.130548	0.2

In [34]:

```
1 sns.heatmap(df.corr(),annot=True)
```

Out[34]:

<matplotlib.axes._subplots.AxesSubplot at 0x2072fae0b88>

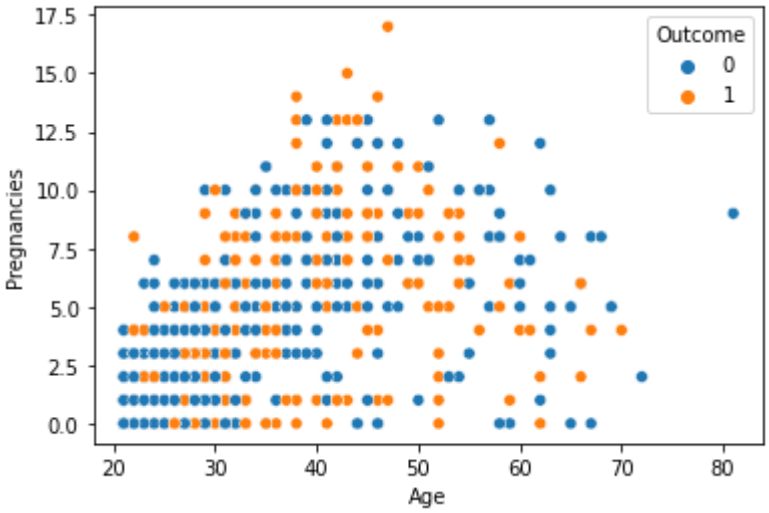


In [10]:

```
1 sns.scatterplot(x=df['Age'],y=df['Pregnancies'],hue=df['Outcome'],data=df)
2
3 # We are plotting a scatter plot to see the realtion between these two features as thei
4
5 # I am unable to observe any specific realtion between Age and Pregnancies. So, we'll j
```

Out[10]:

<AxesSubplot:xlabel='Age', ylabel='Pregnancies'>

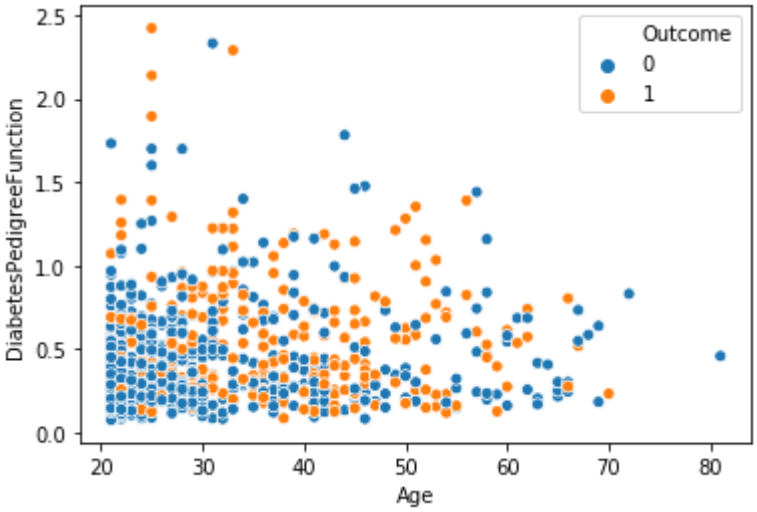


In [37]:

```
1 sns.scatterplot(x=df['Age'],y=df['DiabetesPedigreeFunction'],hue=df['Outcome'],data=df)
```

Out[37]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x2072fad4e48>
```



In [38]:

```
1 df['Age'].shape
```

Out[38]:

```
(768,)
```

In [39]:

```
1 df['Pregnancies'].shape
```

Out[39]:

```
(768,)
```

In [40]:

```
1 df.head()
```

Out[40]:

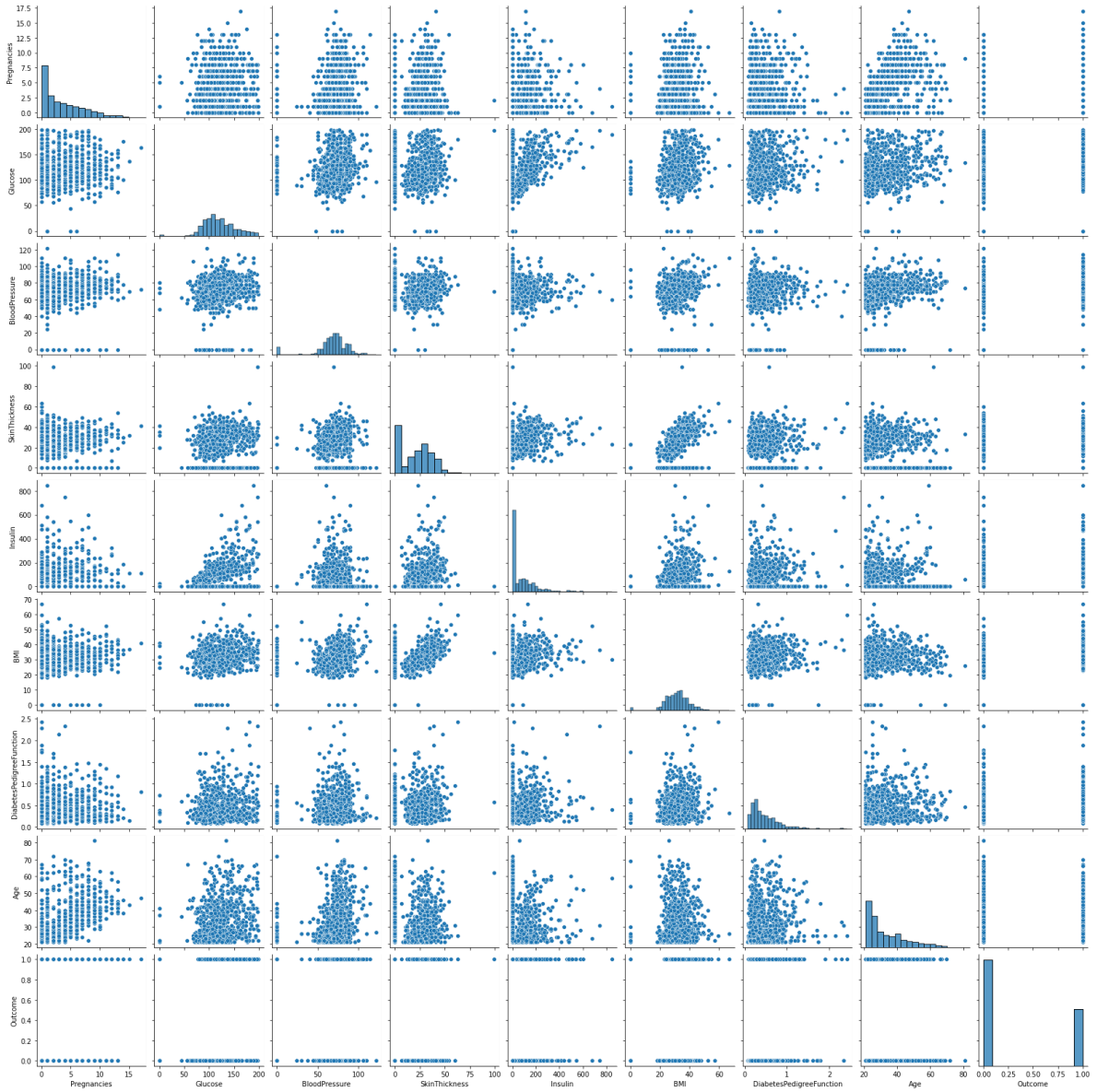
	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

In [17]:

```
1 sns.pairplot(df)
```

Out[17]:

<seaborn.axisgrid.PairGrid at 0x13a37398080>



In [15]:

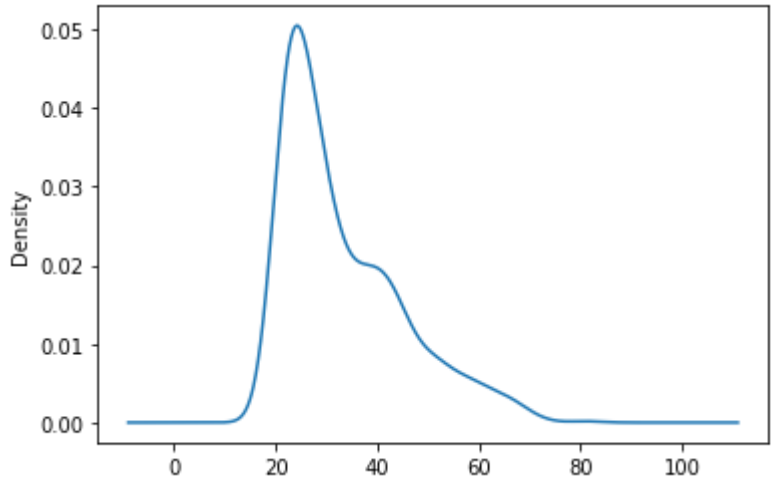
```
1 X = df.drop('Outcome',axis=1)
2 y = df['Outcome']
```

In [16]:

```
1 df['Age'].plot(kind='kde')
```

Out[16]:

```
<AxesSubplot:ylabel='Density'>
```



In [17]:

```
1 df['Age'].value_counts().head(10)
```

Out[17]:

```
22    72
21    63
25    48
24    46
23    38
28    35
26    33
27    32
29    29
31    24
Name: Age, dtype: int64
```

In [18]:

```
y.value_counts()

# We can observe that there are more 0's than 1's. This leads to an imbalance dataset c
```

Out[18]:

```
0    500
1    268
Name: Outcome, dtype: int64
```

In [19]:

```
pip install imblearn
```

```
Requirement already satisfied: imblearn in c:\users\raabh\anaconda3\envs\dia
betes\lib\site-packages (0.0)
Requirement already satisfied: imbalanced-learn in c:\users\raabh\anaconda3
\envs\diabetes\lib\site-packages (from imblearn) (0.8.0)
Requirement already satisfied: numpy>=1.13.3 in c:\users\raabh\anaconda3\env
s\diabetes\lib\site-packages (from imbalanced-learn->imblearn) (1.19.5)
Requirement already satisfied: joblib>=0.11 in c:\users\raabh\anaconda3\envs
\diabetes\lib\site-packages (from imbalanced-learn->imblearn) (1.0.1)
Requirement already satisfied: scipy>=0.19.1 in c:\users\raabh\anaconda3\env
s\diabetes\lib\site-packages (from imbalanced-learn->imblearn) (1.5.4)
Requirement already satisfied: scikit-learn>=0.24 in c:\users\raabh\anaconda
3\envs\diabetes\lib\site-packages (from imbalanced-learn->imblearn) (0.24.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\raabh\anacon
da3\envs\diabetes\lib\site-packages (from scikit-learn>=0.24->imbalanced-lea
rn->imblearn) (2.1.0)
Note: you may need to restart the kernel to use updated packages.
```

In [20]:

```
from imblearn.combine import SMOTETomek

# Library for implementing oversampling
```

In [21]:

```
smk = SMOTETomek(random_state=11)
```

In [22]:

```
X_res,y_res = smk.fit_resample(X,y)
```

In [23]:

```
print(X_res.shape,y_res.shape)
```

```
(952, 8) (952,)
```

In [24]:

```
1 y_res.value_counts()
```

Out[24]:

```
1    476
0    476
Name: Outcome, dtype: int64
```

In [25]:

```
1 X_res['Age'].value_counts()
```

Out[25]:

```
22    76
21    63
25    58
24    50
28    45
23    43
29    42
26    42
27    36
41    31
31    30
30    28
33    28
42    27
32    25
34    23
36    21
37    19
46    19
43    18
45    17
39    17
38    17
40    15
35    15
50    14
51    13
44    13
52    11
53    10
47     9
54     9
58     8
48     8
57     6
56     6
60     5
62     5
49     4
63     4
66     4
55     3
59     3
61     2
65     2
67     2
69     2
72     1
64     1
68     1
81     1
Name: Age, dtype: int64
```

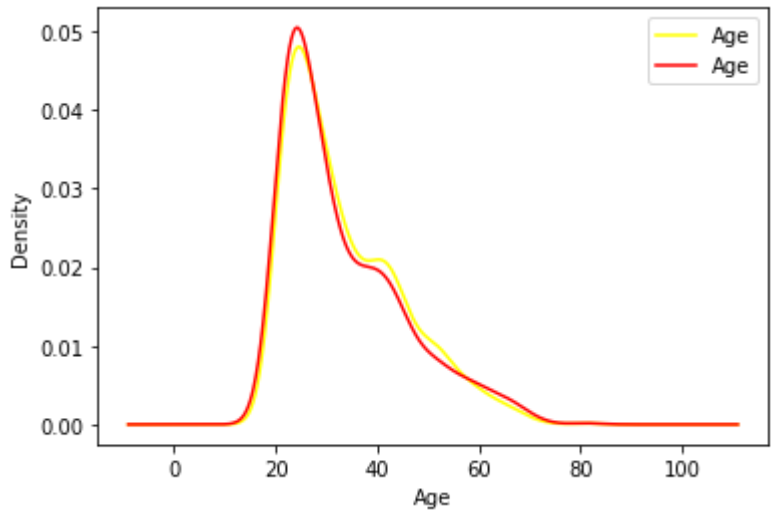
In [26]:

```
X_res['Age'].plot(kind="kde",color='yellow')
df['Age'].plot(kind='kde',color='red')
plt.legend()
plt.xlabel("Age")

# We can clearly notice that there are no differences between
```

Out[26]:

```
Text(0.5, 0, 'Age')
```

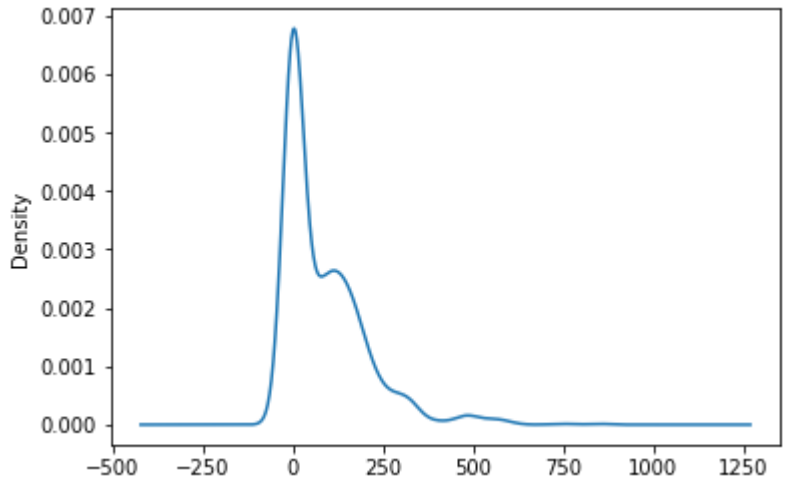


In [27]:

```
X_res['Insulin'].plot(kind='kde')
```

Out[27]:

```
<AxesSubplot:ylabel='Density'>
```



In [28]:

```
X_res.shape
```

Out[28]:

```
(952, 8)
```

In [29]:

```
1 y_res.shape
```

Out[29]:

```
(952,)
```

In [30]:

```
1 X_res.head()
```

Out[30]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

In [31]:

```
1 y_res.head()
```

Out[31]:

```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

In [32]:

```
1 from sklearn.preprocessing import StandardScaler
```

In [33]:

```
1 scaled_features = StandardScaler()
```

In [34]:

```
1 scale_X = scaled_features.fit_transform(X_res)
```

In [35]:

```
1 final_X = pd.DataFrame(scale_X,columns=X_res.columns)
```

In [36]:

```
final_X.head()
```

Out[36]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigre
0	0.609835	0.684394	0.124995	0.840259	-0.745850	0.113056	
1	-0.870438	-1.258118	-0.180906	0.468287	-0.745850	-0.813464	
2	1.201944	1.763568	-0.282873	-1.329580	-0.745850	-1.250252	
3	-0.870438	-1.134784	-0.180906	0.096314	0.070259	-0.614924	
4	-1.166492	0.345225	-1.506477	0.840259	0.712727	1.370476	

MACHINE LEARNING ALGORITHM

In [37]:

```
# Decision Tree algorithm will be applie to the dataset in the notebook
```

In [38]:

```
X_res.shape
# FEATURES SHAPE
```

Out[38]:

```
(952, 8)
```

In [39]:

```
y_res.shape
# LABELS
```

Out[39]:

```
(952,)
```

In [40]:

```
from sklearn.model_selection import train_test_split
```

In [41]:

```
train_X,test_X,train_y,test_y = train_test_split(X_res,y_res,test_size=0.25)
```

In [42]:

```
from sklearn.tree import DecisionTreeClassifier
```

In [43]:

```
dt_c = DecisionTreeClassifier()
```

In [44]:

```
from sklearn.metrics import accuracy_score
```

In [45]:

```
dt_c.fit(train_X,train_y)
```

Out[45]:

```
DecisionTreeClassifier()
```

In [46]:

```
accuracy_score = accuracy_score(test_y,dt_c.predict(test_X))
```

In [47]:

```
accuracy_score
```

Out[47]:

```
0.7815126050420168
```

In [48]:

```
pip install pickle-mixin
```

```
Requirement already satisfied: pickle-mixin in c:\users\raabh\anaconda3\envs
\diabetes\lib\site-packages (1.0.2)Note: you may need to restart the kernel
to use updated packages.
```

In [49]:

```
# Saving the model to the disk

import pickle
pickle.dump(dt_c, open('model.pkl','wb'))
```

In [50]:

```
#Loading the model

model = pickle.load(open('model.pkl','rb'))
```

In [51]:

```
1 X_res.columns
```

Out[51]:

```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insuli
n',
       'BMI', 'DiabetesPedigreeFunction', 'Age'],
      dtype='object')
```

In [52]:

```
1 pip install scikit-learn
```

```
Requirement already satisfied: scikit-learn in c:\users\raabh\anaconda3\envs
\diabetes\lib\site-packages (0.24.2)
Requirement already satisfied: numpy>=1.13.3 in c:\users\raabh\anaconda3\env
s\diabetes\lib\site-packages (from scikit-learn) (1.19.5)
Requirement already satisfied: joblib>=0.11 in c:\users\raabh\anaconda3\envs
\diabetes\lib\site-packages (from scikit-learn) (1.0.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\raabh\anacon
da3\envs\diabetes\lib\site-packages (from scikit-learn) (2.1.0)
Requirement already satisfied: scipy>=0.19.1 in c:\users\raabh\anaconda3\env
s\diabetes\lib\site-packages (from scikit-learn) (1.5.4)
Note: you may need to restart the kernel to use updated packages.
```

In [53]:

```
1 pip install --user flask
```

```
Requirement already satisfied: flask in c:\users\raabh\appdata\roaming\pytho
n\python36\site-packages (2.0.1)
Requirement already satisfied: Jinja2>=3.0 in c:\users\raabh\appdata\roaming
\python\python36\site-packages (from flask) (3.0.1)
Requirement already satisfied: Werkzeug>=2.0 in c:\users\raabh\appdata\roami
ng\python\python36\site-packages (from flask) (2.0.1)
Requirement already satisfied: itsdangerous>=2.0 in c:\users\raabh\appdata\r
oaming\python\python36\site-packages (from flask) (2.0.1)
Requirement already satisfied: click>=7.1.2 in c:\users\raabh\appdata\roamin
g\python\python36\site-packages (from flask) (8.0.1)
Requirement already satisfied: colorama in c:\users\raabh\anaconda3\envs\dia
betes\lib\site-packages (from click>=7.1.2->flask) (0.4.4)
Requirement already satisfied: importlib-metadata in c:\users\raabh\anaconda
3\envs\diabetes\lib\site-packages (from click>=7.1.2->flask) (3.10.0)
Requirement already satisfied: MarkupSafe>=2.0 in c:\users\raabh\anaconda3\e
nvs\diabetes\lib\site-packages (from Jinja2>=3.0->flask) (2.0.1)
Requirement already satisfied: dataclasses in c:\users\raabh\appdata\roaming
\python\python36\site-packages (from Werkzeug>=2.0->flask) (0.8)
Requirement already satisfied: zipp>=0.5 in c:\users\raabh\anaconda3\envs\di
abetes\lib\site-packages (from importlib-metadata->click>=7.1.2->flask) (3.
4.1)
Requirement already satisfied: typing-extensions>=3.6.4 in c:\users\raabh\an
aconda3\envs\diabetes\lib\site-packages (from importlib-metadata->click>=7.
1.2->flask) (3.7.4.3)
Note: you may need to restart the kernel to use updated packages.
```

In [58]:

```
pip freeze --local > requirement.txt
```

```
Note: you may need to restart the kernel to use updated packages.
```

In [ ]:

```

```

CODE FOR CREATING AN APPLICATION

```
import numpy as np
import pickle
from flask import Flask, request, render_template

app = Flask(__name__, template_folder='./templates')
app.debug=True
model = pickle.load(open('model.pkl','rb'))

@app.route('/')
def home():
    return render_template('index.html')

@app.route('/predict', methods=['POST'])
def predict():
    features = [int(x) for x in request.form.values()]
    final_features = [np.array(features)]
    prediction = model.predict(final_features)

    return render_template('index.html', prediction_text="Your output is
{}".format(prediction))

if __name__ == "__main__":
    app.run(use_reloader=False,debug=True)
```

HTML CODE FOR CREATING THE FRONT END AND TAKING INPUT FROM THE USER

```
<!DOCTYPE html>
<html>
<head>
	<meta charset="UTF-8">
	<title>ML API</title>
</head>
<body>
	<div class="login">
		<h1>DIABETES PREDICTION MODEL</h1>

		<form action="{{url_for('predict')}}"method="POST">
			<input type="text" name="Pregnancies" placeholder="Number of Pregnancies" required="required" />
			<input type="text" name="Glucose" placeholder="Glucose" required="required" />
			<input type="text" name="BloodPressure" placeholder="BloodPressure" required="required" />
			<input type="text" name="SkinThickness" placeholder="SkinThickness" required="required" />
			<input type="text" name="Insulin" placeholder="Insulin" required="required" />
			<input type="text" name="BMI" placeholder="BMI" required="required" />
			<input type="text" name="DiabetesPedigreeFunction" placeholder="DiabetesPedigreeFunction" required="required" />
			<input type="text" name="Age" placeholder="Age" required="required" />
```

```
                    <button type="submit" class="btn btn-primary btn-block btn-large">predict</button>
                </form>


        <br>
        <br>


        {{prediction_text}}
        </div>


</body>
</html>
```

EXECUTION

In order to execute the code written in the local environment. We need a total of three sections:

1. The model preparation code which trains the model
2. The application to execute
3. The HTML code for the front end and to request values from the user

Firstly go to the local environment and change the current directory to the path where all the files mentioned above are present.

Then, after the directory is set, type the following code in anaconda prompt:

```
Python app.py
```

This will execute and give us an output which looks like this:

```
* Serving Flask app "app" (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Now, copy the local environment link shown in the output above and this should execute and give you a GUI that would be able to take input of all the features and predict the output based on the machine learning model you trained i.e., Decision Tree Classifier.

The output can be seen in the next page along with the values entered and prediction.

DIABETES PREDICTION MODEL

Number of Pregnancies	Glucose	BloodPressure
SkinThickness	Insulin	BMI
DiabetesPedigreeFunction	Age	predict

Your output is [0]

CONCLUSION

To conclude, the project is very useful in analysing the trends of patients suffering from diabetes and to make necessary prediction that will be helpful for people in foreseeing future to predict diabetic conditions.

There are some limitations to the project. They can be observed as follows:

1. In the feature **"number of pregnancies"**, if twins/ triplets are born, then we still do not have a concrete answer to how that could affect the patient's diabetic record.

2. The project includes features that are complex and hard to calculate for a normal person.

3. Any machine learning model will generate a prediction with greater accuracy, if more data is fed to it. So, in order to collect data, the patients need to be willingly give permission to release their data and considering present world's scenario, it will be difficult for people to give their personal data.

4. The model only accepts inputs in the form of integers as of now. This is done so, to reduce the complexity of the model.

REFERENCES

1. https://www.analyticsvidhya.com/blog/2021/02/machine-learning-101-decision-tree-algorithm-for-classification/

2. https://www.analyticsvidhya.com/blog/2020/04/statistics-data-science-normal-distribution/#:~:text=According%20to%20the%20Empirical%20Rule,standard%20deviations%20of%20the%20mean

3. https://matplotlib.org/

4. https://towardsdatascience.com/seaborn-python-8563c3d0ad41#:~:text=Seaborn%20is%20a%20library%20in%20Python%20predominantly%20used%20for%20making%20statistical%20graphics.&text=Seaborn%20is%20a%20data%20visualization,exploration%20and%20understanding%20of%20data.

5. https://towardsdatascience.com/model-deployment-using-flask-c5dcbb6499c9

6. https://www.w3schools.com/html/

7. https://flatironschool.com/blog/how-much-math-do-you-need-to-become-a-data-scientist

8. https://medium.com/s/story/essential-math-for-data-science-why-and-how-e88271367fbd

9. https://www.kaggle.com/antaresnyc/human-metagenomics/tasks?taskId=2965

10. https://www.geeksforgeeks.org/understanding-python-pickling-example/