



4222-SURYA GROUP OF INSTITUTIONS
VIKRAVANDI-605652



NAAN MUDHALVAN PROJECT

SUBJECT CODE –SB3001

COURSE NAME – EXPERIENCE BASED PRACTICAL LEARNING

AI BASED DIABETES PREDICTION SYSTEM

PREPARED BY

S.DINESH

REG.NO:422221106006

ECE DEPARTMENT

AI-based-diabetes-prediction-system:



INTRODUCTION:

Welcome to the future of healthcare with our AI-based Diabetes Prediction System. Harnessing the power of artificial intelligence, our innovative solution is designed to revolutionize the way we approach diabetes detection and prevention. By analyzing vast amounts of data and employing advanced machine learning algorithms, our system provides accurate predictions, enabling early intervention and personalized care. Stay ahead of diabetes with our cutting-edge technology, leading the way towards a healthier tomorrow.

AI MODELS USED:

- ❖ Logistic Regression
- ❖ Decision tress
- ❖ Random forest

STEPS TAKEN:

DATA

COLLECTION:

	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

When encountered with a data set, first we should analyze and “**get to know**” the data set. This step is necessary to familiarize with the data, to gain some understanding of the potential features and to see if data cleaning is needed. First, we will import the necessary libraries and import our data set. We can observe the mentioned columns in the data set.

```
%matplotlib inline
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
diabetes = pd.read_csv('datasets/diabetes.csv')
diabetes.columns
```

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

We can examine the data set using the pandas’ **head()** method.

```
diabetes.head()
```

```
print("Diabetes data set dimensions : {}".format(diabetes.shape))
```

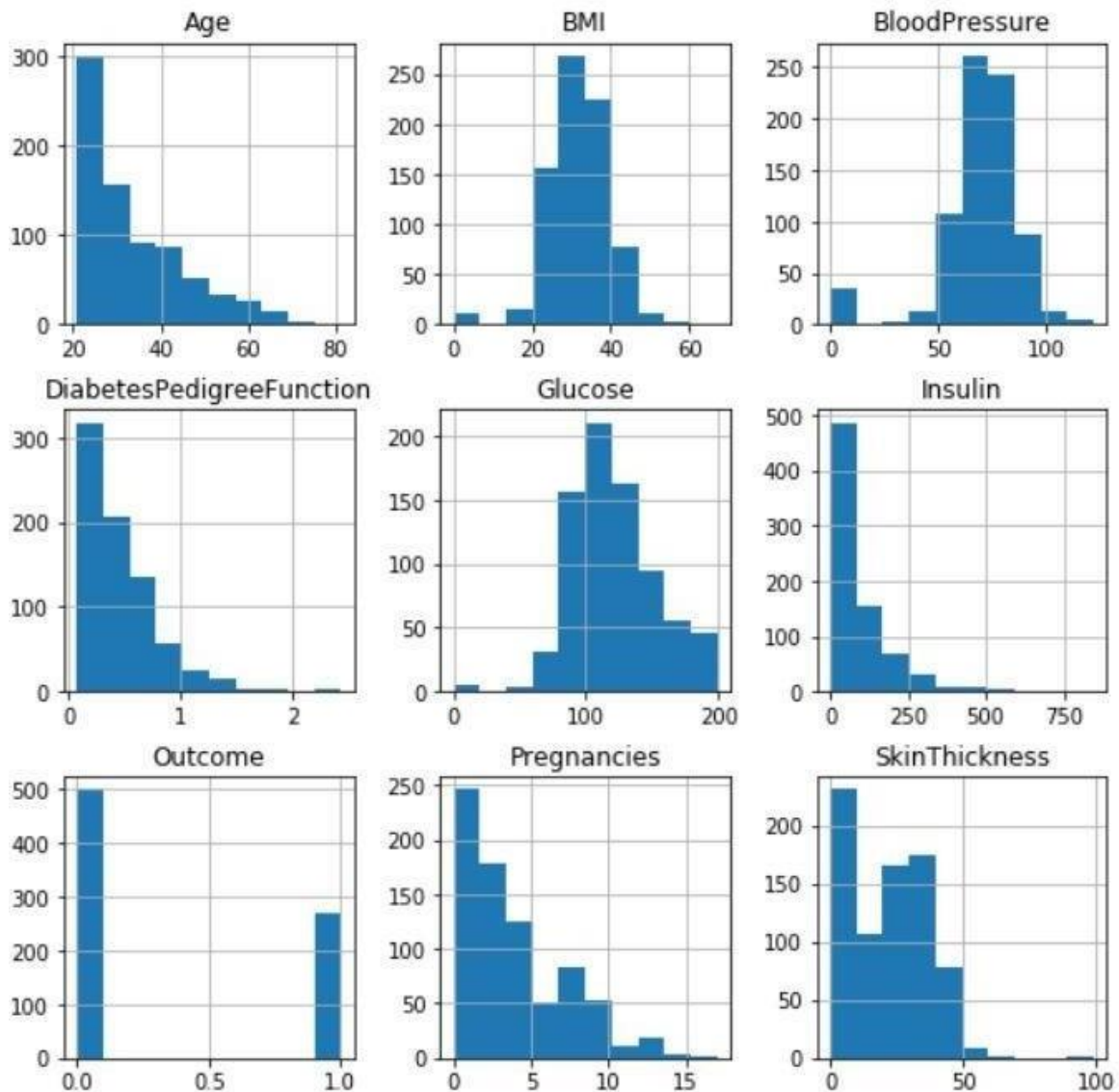
```
Diabetes data set dimensions : (768, 9) .
```

We can observe that the data set contain 768 rows and 9 columns. ‘*Outcome*’ is the column which we are going to predict, which says if the patient is diabetic or not. 1 means the person is diabetic and 0 means a person is not. We can identify that out of the 768 persons, 500 are labeled as 0 (non-diabetic) and 268 as 1 (diabetic).

```
diabetes.groupby('Outcome').size()
```

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

Visualization of data is an imperative aspect of data science. It helps to understand data and also to explain the data to another person. Python has several interesting visualization libraries such as Matplotlib, Seaborn, etc. visualization which is built on top of matplotlib, to find the data distribution of the features.



```
diabetes.groupby('Outcome').hist(figsize=(9, 9))
```

DATA PREPROCESSING:

The next phase of the machine learning work flow is data cleaning. Considered to be one of the crucial steps of the workflow, because it can make or break the model. There is a saying in machine learning **“Better data beats fancier algorithms”**, which suggests better data gives you better resulting models.

There are several factors to consider in the data cleaning process

1. Duplicate or irrelevant observations

2. Bad labeling of data, same category occurring multiple times
3. Missing or null data points.
4. Unexpected outliers.

```
diabetes.isnull().su
```

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

```
m()
```

```
diabetes.isna().su
```

```
m()
```

Blood pressure: By observing the data we can see that there are 0 values for blood pressure. And it is evident that the readings of the data set seem wrong because a living person cannot have a diastolic blood pressure of zero. By observing the data we can see 35 counts where the value is 0.

```
print("Total : ", diabetes[diabetes.BloodPressure == 0].shape[0])Total :
35print(diabetes[diabetes.BloodPressure == 0].groupby('Outcome')['Age'].count())Outcome
0    19
1    16
Name: Age, dtype: int64
```

Plasma glucose levels: Even after fasting glucose levels would not be as low as zero. Therefore zero is an invalid reading. By observing the data we can see 5 counts where the value is 0.

```
print("Total : ", diabetes[diabetes.Glucose == 0].shape[0])Total : 5print(diabetes[diabetes.Glucose
== 0].groupby('Outcome')['Age'].count())Total : 5
Outcome
0    3
1    2
Name: Age, dtype: int64
```

Skin Fold Thickness: For normal people, skin fold thickness can't be less than 10 mm better

yet zero. Total count where value is 0: 227.

```
print("Total : ", diabetes[diabetes.SkinThickness == 0].shape[0])Total :  
227print(diabetes[diabetes.SkinThickness == 0].groupby('Outcome')['Age'].count())Outcome  
0    139
```

BMI: Should not be 0 or close to zero unless the person is really underweight which could be life-threatening.

```
print("Total : ", diabetes[diabetes.BMI == 0].shape[0])Total : 11print(diabetes[diabetes.BMI ==  
0].groupby('Outcome')['Age'].count())Outcome  
0     9  
1     2  
Name: Age, dtype: int64
```

Insulin: In a rare situation a person can have zero insulin but by observing the data, we can find that there is a total of 374 counts.

```
print("Total : ", diabetes[diabetes.Insulin == 0].shape[0])Total : 374print(diabetes[diabetes.Insulin  
== 0].groupby('Outcome')['Age'].count())Outcome  
0    236  
1    138  
Name: Age, dtype: int64
```

Here are several ways to handle invalid data values :

1. Ignore/remove these cases: This is not actually possible in most cases because that would mean losing valuable information. And in this case “skin thickness” and “insulin” columns mean to have a lot of invalid points. But it might work for “BMI”, “glucose ”and “blood pressure” data points.
2. Put average/mean values: This might work for some data sets, but in our case putting a mean value to the blood pressure column would send a wrong signal to the model.
3. Avoid using features: It is possible to not use the features with a lot of invalid values for the model. This may work for “skin thickness” but it's hard to predict that.
4. We will remove the rows which the “BloodPressure”, “BMI” and “Glucose” are zero.

```
5. diabetes_mod = diabetes[(diabetes.BloodPressure != 0) & (diabetes.BMI != 0) &
    (diabetes.Glucose != 0)]print(diabetes_mod.shape)(724, 9)
```

MODEL SELECTION:

Feature Selection is the process of transforming the gathered data into features that better represent the problem that we are trying to solve to the model, to improve its performance and accuracy. Feature selection creates more input features from the existing features and also combines several features to produce more intuitive features to feed to the model.

```
feature_names = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI',
'DiabetesPedigreeFunction', 'Age']X = diabetes_mod[feature_names]
y = diabetes_mod.
```

MODEL SELECTION:

Model selection or algorithm selection phase is the most exciting and the heart of machine learning. It is the phase where we select the model which performs best for the data set at hand

.First, we will be calculating the “**Classification Accuracy (Testing Accuracy)**” of a given set of classification models with their default parameters to determine which model performs better with the diabetes data set. We will import the necessary libraries for the notebook. We import 7 classifiers namely **K-Nearest Neighbours** , **Support Vector Classifier**, **Logistic Regression**, **Gaussian Naive Bayes**, **Random Forest**, and **Gradient Boost** to be contenders for the best classifier.

```

from sklearn.neighbors import
KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.linear_model import
LogisticRegression
from sklearn.tree import
DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import

from sklearn import models = []
models.append(('KNN', KNeighborsClassifier()))
models.append(('SVC', SVC()))
models.append(('LR', LogisticRegression()))
models.append(('DT', DecisionTreeClassifier()))
models.append(('GNB', GaussianNB()))
models.append(('RF', RandomForestClassifier()))
models.append(('GB', GradientBoostingClassifier()))
from sklearn.ensemble
import GradientBoostingClassifier

```

EVALUATION:

It is a general practice to avoid training and testing on the same data. The reasons are that the goal of the model is to predict **out-of-sample data**, and the model could be overly complex leading to **overfitting**. To avoid the aforementioned problems, there are two precautions.

1. Train/Test Split
2. K-Fold Cross-Validation

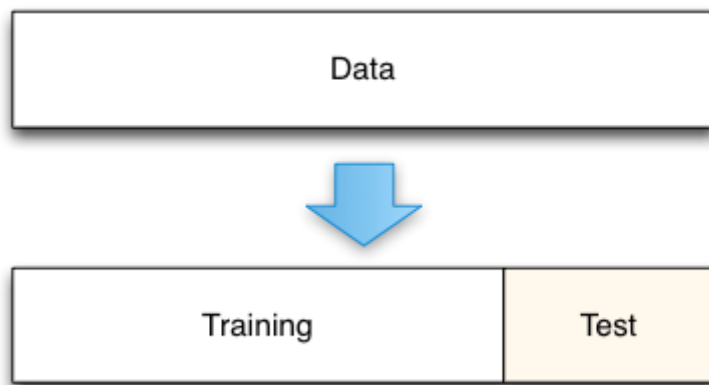
We will import “*train_test_split*” for train/test split and “*cross_val_score*” for k-fold cross-validation. “*accuracy_score*” is to evaluate the accuracy of the model in the train/test split method.

TRAIN/TEST SPLIT:

This method split the data set into two portions: a **training set** and a **testing set**. The **training set** is used to train the model. And the **testing set** is used to test the model, and evaluate the accuracy.

*Pros : But, train/test split is still useful because of its **flexibility and speed***

*Cons : Provides a **high-variance estimate** of out-of-sample accuracy*



```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify =  
diabetes_mod.Outcome, random_state=0)
```

Then we fit each model in a loop and calculate the accuracy of the respective model using the “*accuracy_score*”.

```
names = []  
scores = []  
for name, model in models:  
    model.fit(X_train, y_train)  
    y_pred = model.predict(X_test)  
    scores.append(accuracy_score(y_test, y_pred))  
    names.append(name)  
tr_split = pd.DataFrame({'Name': names,  
                          'Score': scores})  
print(tr_split)
```

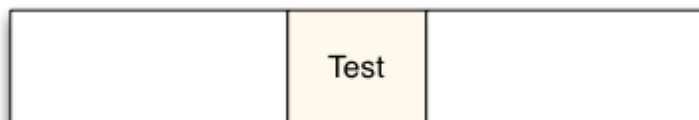
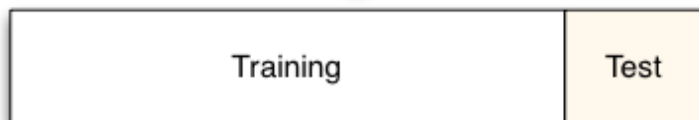
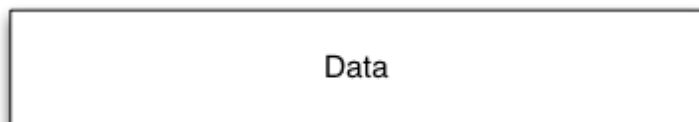
	Name	Score
0	KNN	0.711521
1	SVC	0.656075
2	LR	0.776440
3	DT	0.681327
4	GNB	0.755681
5	RF	0.739165
6	GB	0.765442

K-FOLD CROSS VALIDATION:

```
names = []
scores = []
for name, model in models:

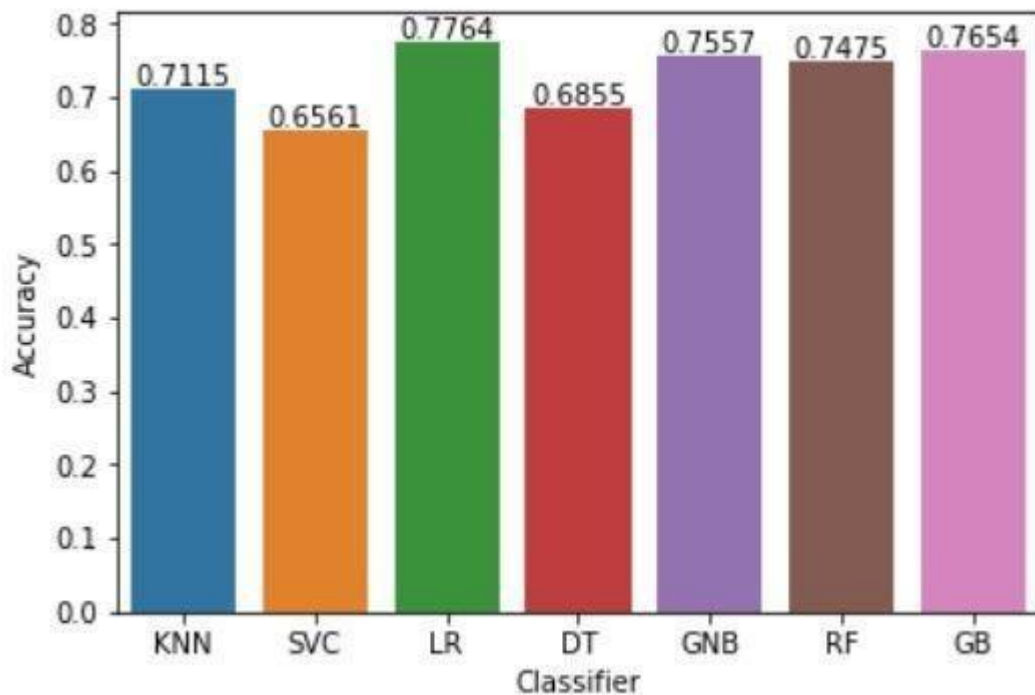
    kfold = KFold(n_splits=10, random_state=10)
    score = cross_val_score(model, X, y, cv=kfold,
scoring='accuracy').mean()

    names.append(name)
    scores.append(score)
kf_cross_val = pd.DataFrame({'Name':
names, 'Score': scores})
print(kf_cross_val)
```

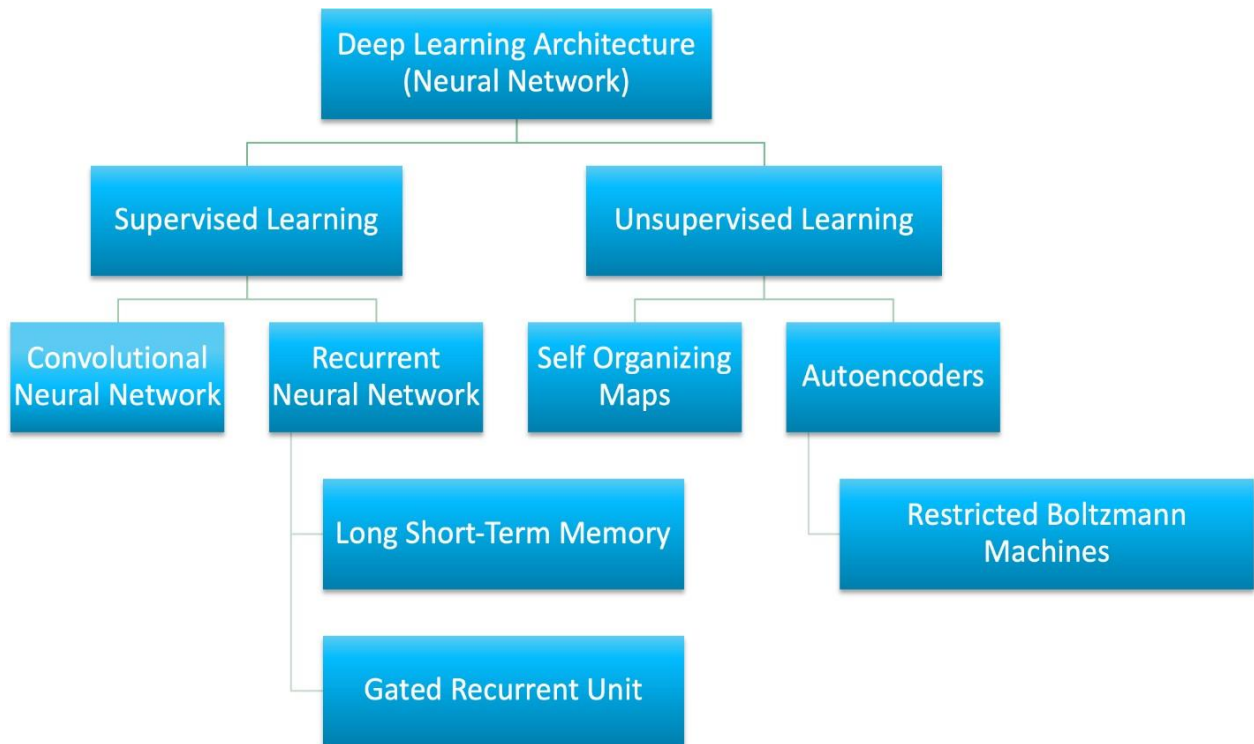


	Name	Score
0	KNN	0.711521
1	SVC	0.656075
2	LR	0.776440
3	DT	0.685494
4	GNB	0.755681
5	RF	0.747519
6	GB	0.765442

```
axis = sns.barplot(x = 'Name', y = 'Score', data = kf_cross_val)
axis.set(xlabel='Classifier', ylabel='Accuracy')
for p in axis.patches:
    height = p.get_height()
    axis.text(p.get_x() + p.get_width()/2, height + 0.005,
' {:.14f}'.format(height), ha="center")
plt.show()
```



DEEP LEARNING ARCHITECTURE:



Importing libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Importing dataset

```
dataset = pd.read_csv('../input/diabetes-data-set/diabetes.csv')
```

Viewing the dataset, its dimensions, features and statistical Summary

```
dataset.head()
```

	Pregnancies	Glucose	BP	Skin thickness	Insulin	BMI	Diabetes pedigree function	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

```
dataset.shape
(768, 9)
dataset.info()
```

```
<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  DiabetesPedigree
    Function              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
```

```
dataset.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.2500	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.0000	122.00

SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10

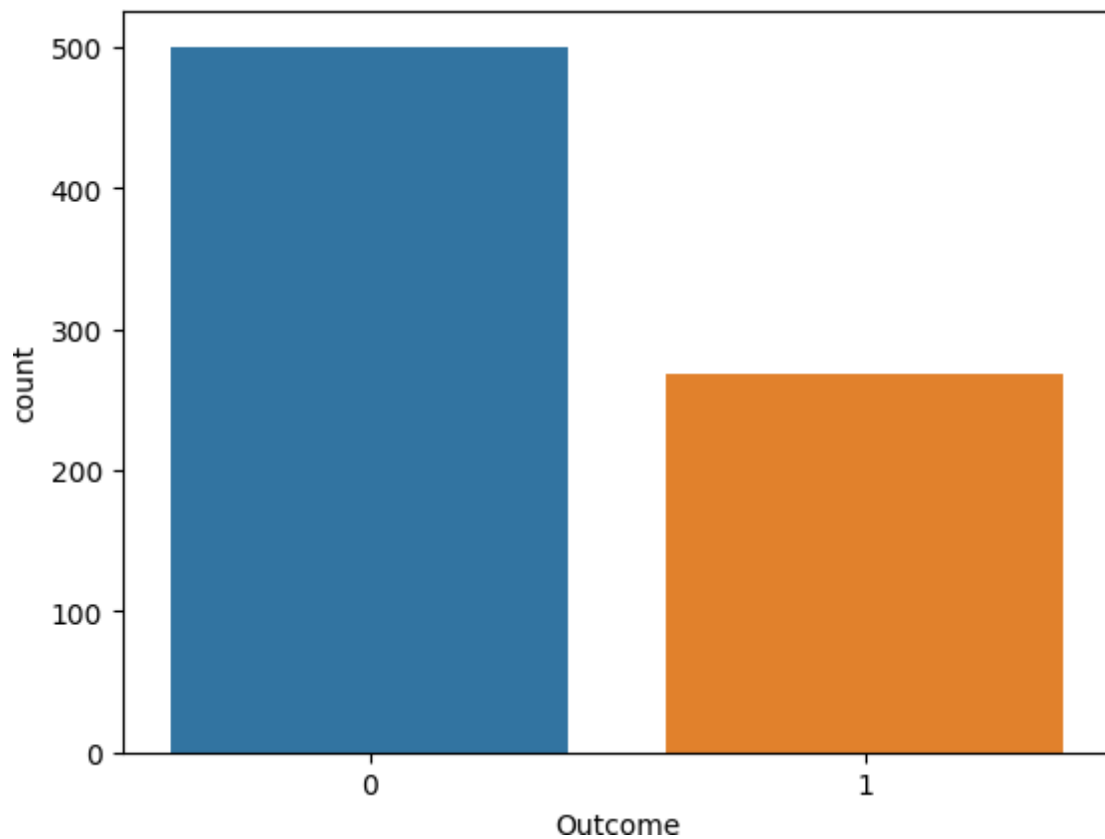
count	mean	std	min	25%	50%	75%	max	
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.0000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

Detecting null values

```
dataset.isnull().sum()
Pregnancies      0
Glucose           0
BloodPressure     0
SkinThickness     0
Insulin           0
BMI               0
DiabetesPedigreeFunction  0
Age              0
Outcome           0
dtype: int64
```

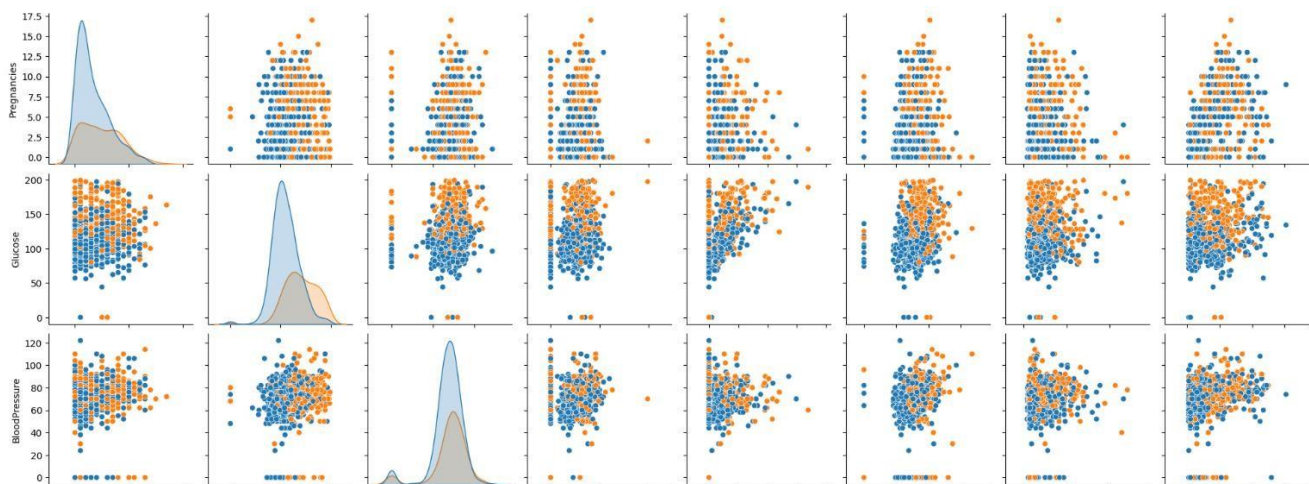
Data Visualization

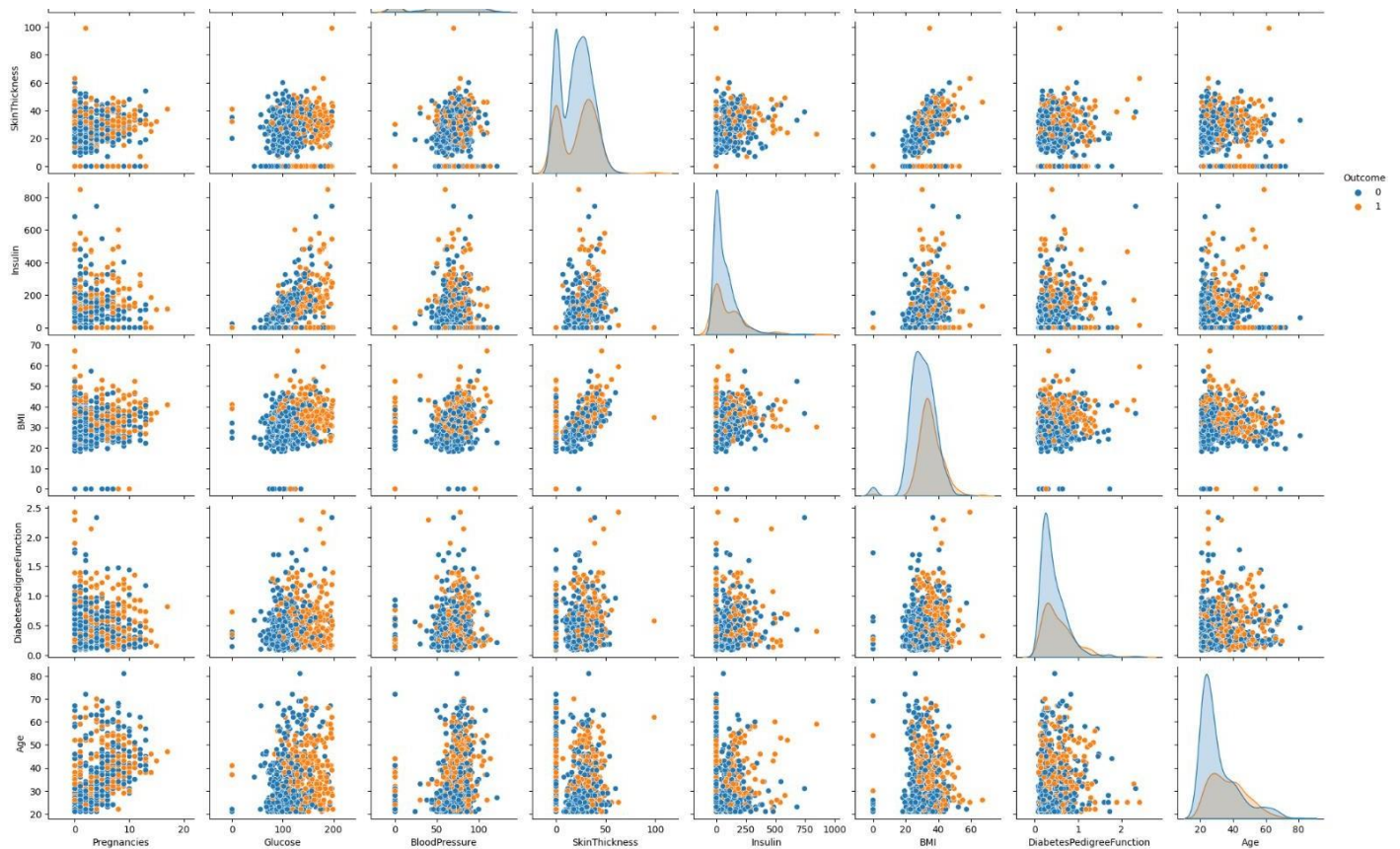
```
sns.countplot(x = 'Outcome', data = dataset)
<Axes: xlabel='Outcome', ylabel='count'>
```



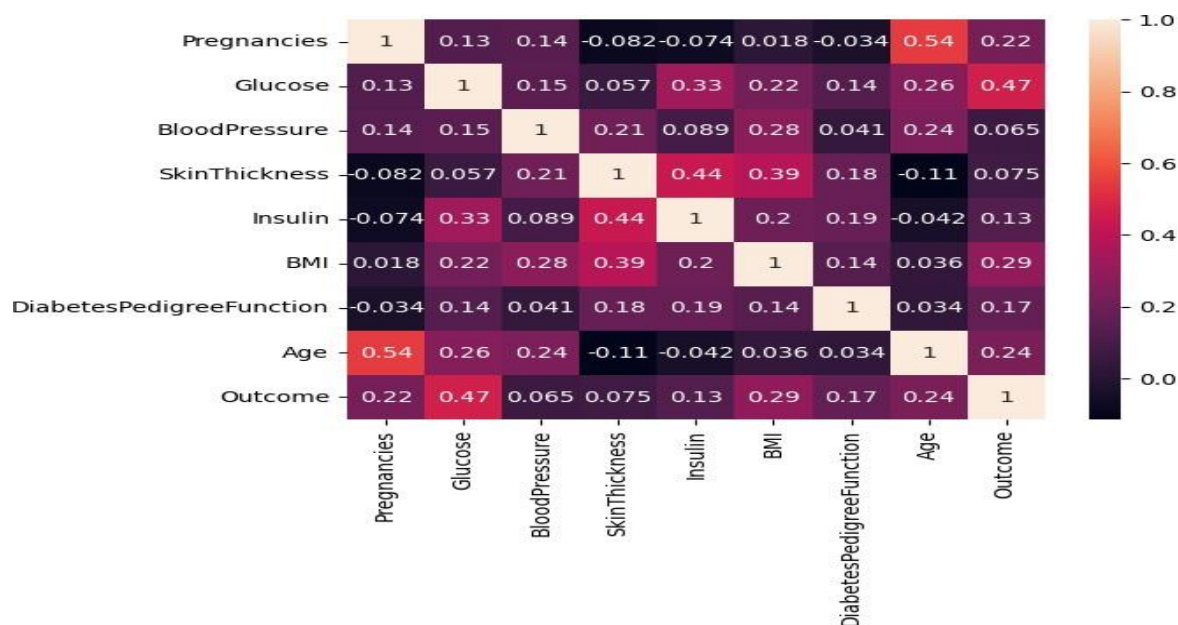
```
# Pairplot
sns.pairplot(data = dataset, hue = 'Outcome')
plt.show()
```

/opt/conda/lib/python3.10/site-packages/seaborn/axisgrid.py:118: UserWarning: The figure layout has changed to tight
 self._figure.tight_layout(*args, **kwargs)





```
# Heatmap
sns.heatmap(dataset.corr(), annot = True)
plt.show()
```



Processing the Data

Replacing zero values with NaN

```
dataset_new = dataset
dataset_new[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]] =
dataset_new[["Glucose", "BloodPressure", "SkinThickness", "Insulin", "BMI"]].replace(0,
np.NaN)
```



```
linkcode
# Count of NaN
dataset_new.isnull().sum()
```

```
Pregnancies      0
Glucose           5
BloodPressure     35
SkinThickness    227
Insulin          374
BMI              11
DiabetesPedigree
e
Function          0
Age              0
Outcome          0
dtype: int64
```

```
# Replacing NaN with mean values
```

```
dataset_new["Glucose"].fillna(dataset_new["Glucose"].mean(), inplace = True)
dataset_new["BloodPressure"].fillna(dataset_new["BloodPressure"].mean(), inplace = True)
dataset_new["SkinThickness"].fillna(dataset_new["SkinThickness"].mean(), inplace = True)
dataset_new["Insulin"].fillna(dataset_new["Insulin"].mean(), inplace = True)
dataset_new["BMI"].fillna(dataset_new["BMI"].mean(), inplace = True)
```

```
dataset_new.isnull().sum()
```

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

Logistic Regression

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

```
# Splitting X and Y
```

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = 0.20, random_state = 42, stratify = dataset_new['Outcome'])
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, Y_train)
y_predict = model.predict(X_test)
```

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train, Y_train)
```

```
y_predict = model.predict(X_test)
```

/opt/conda/lib/python3.10/site-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data
as shown in: [https://scikit-](https://scikit-learn.org/stable/modules/preprocessing.html)

[learn.org/stable/modules/preprocessing.html](https://scikit-learn.org/stable/modules/preprocessing.html)

Please also refer to the documentation for alternative solver

options: [https://scikit-](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

[learn.org/stable/modules/linear_model.html#logistic-regression](https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression)

n_iter_i = _check_optimize_result(

```
y_predict
```

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

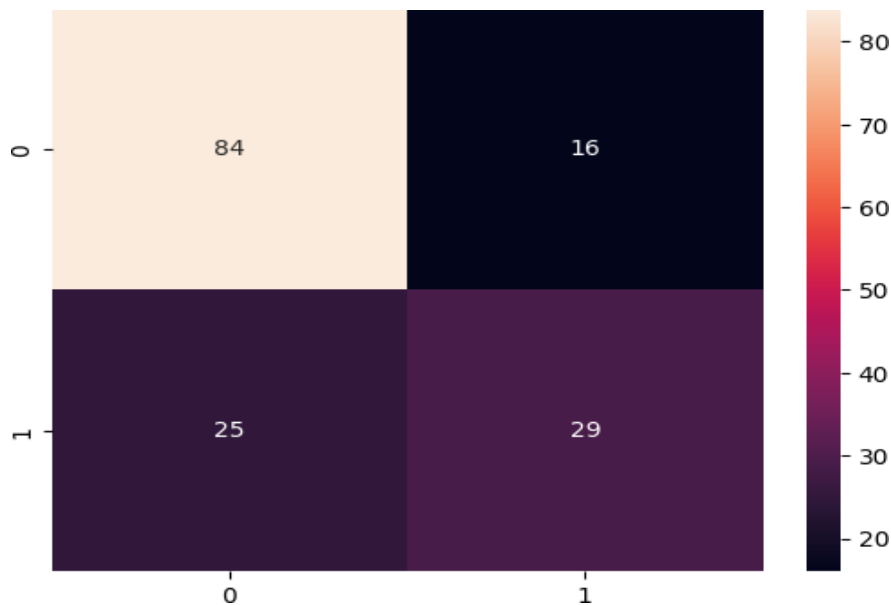
```
# Confusion matrix
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, y_predict)
cm
```

```
array([[84, 16],
       [25, 29]])
```

```
# Heatmap of Confusion matrix
```

```
sns.heatmap(pd.DataFrame(cm), annot=True)
```



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

```
accuracy
```

```
0.7337662337662337
```

```
y_predict =
```

```
model.predict([[1,148,72,35,79.799,33.6,0.627,50]])
```

```
print(y_predict)
```

```
if
```

```
    y_predict==1
```

```
    :
```

```
        print("Diabet
```

```
ic")
```

```
else:
```

```
    print("Non
```

```
Diabetic")[1]
```

```
Diabetic
```

```
/opt/conda/lib/python3.10/site-packages/sklearn/base.py:439: UserWarning: X does not have  
valid feature names, but LogisticRegression was fitted with feature names  
warnings.warn(
```

Why do we need Data Preprocessing?

A real-world data generally contains noises, missing values, and maybe in an unusable format which cannot be directly used for machine learning models. Data preprocessing is required tasks for cleaning the data and making it suitable for a machine learning model which also increases the accuracy and efficiency of a machine learning model.

It involves below steps:

- **Getting the dataset**
 - **Importing libraries**
 - **Importing datasets**
 - **Finding Missing Data**
 - **Encoding Categorical Data**
 - **Splitting dataset into training and test set**
 - **Feature scaling**
-

1) **Get the Dataset**

To create a machine learning model, the first thing we required is a dataset as a machine learning model completely works on data. The collected data for a particular problem in a proper format is known as the **dataset**.

Dataset may be of different formats for different purposes, such as, if we want to create a machine learning model for business purpose, then dataset will be different with the dataset required for a liver patient. So each dataset is different from another dataset. To use the dataset in our code, we usually put it into a **CSV file**. However, sometimes, we may also need to use an HTML or xlsx file.

What is a CSV File?

CSV stands for "**Comma-Separated Values**" files; it is a file format which allows us to save the tabular data, such as spreadsheets. It is useful for huge datasets and can use these datasets in programs.

Here we will use a demo dataset for data preprocessing, and for practice, it can be downloaded from [here](#), for real-world problems, we can download datasets online from various sources such as

We can also create our dataset by gathering data using various API with Python and put that data into a

2) Importing Libraries

In order to perform data preprocessing using Python, we need to import some predefined Python libraries. These libraries are used to perform some specific jobs. There are three specific libraries that we will use for data preprocessing, which are:

Numpy: numpy Python library is used for including any type of mathematical operation in the code. It is the fundamental package for scientific calculation in Python. It also supports to add large, multidimensional arrays and matrices. So, in Python, we can import it as:

1. import numpy as nm

Here we have used **nm**, which is a short name for Numpy, and it will be used in the whole program.

Matplotlib: The second library is **matplotlib**, which is a Python 2D plotting library, and with this library, we need to import a sub-library **pyplot**. This library is used to plot any type of charts in Python for the code. It will be imported as below:

1. import matplotlib.pyplot as mpt

Here we have used mpt as a short name for this library.

Pandas: The last library is the Pandas library, which is one of the most famous Python libraries and used for importing and managing the datasets. It is an open-source data manipulation and analysis library. It will be imported as below:

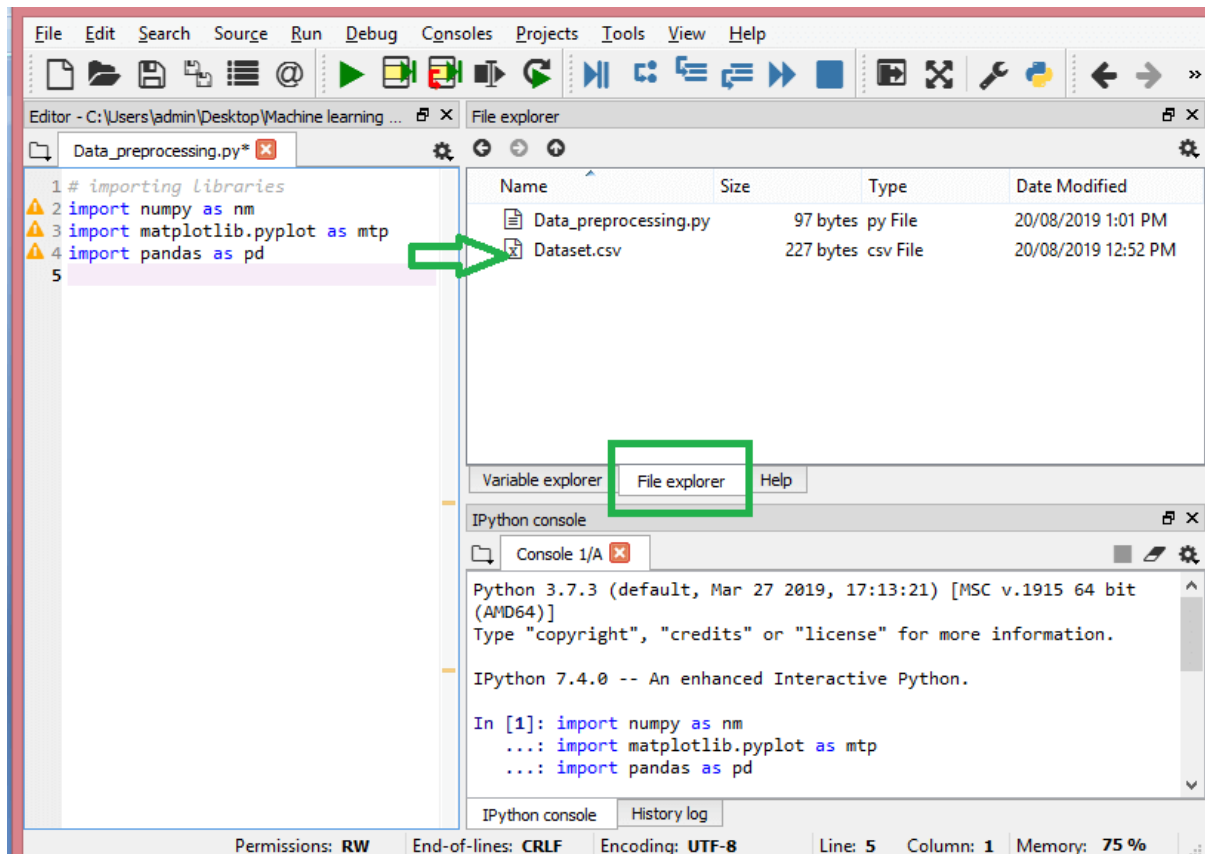
Here, we have used pd as a short name for this library. Consider the below image:

3) Importing the Datasets

Now we need to import the datasets which we have collected for our machine learning project. But before importing a dataset, we need to set the current directory as a working directory. To set a working directory in Spyder IDE, we need to follow the below steps:

1. Save your Python file in the directory which contains dataset.
2. Go to File explorer option in Spyder IDE, and select the required directory.
3. Click on F5 button or run option to execute the file

Here, in the below image, we can see the Python file along with required dataset. Now, the current folder is set as a working directory.



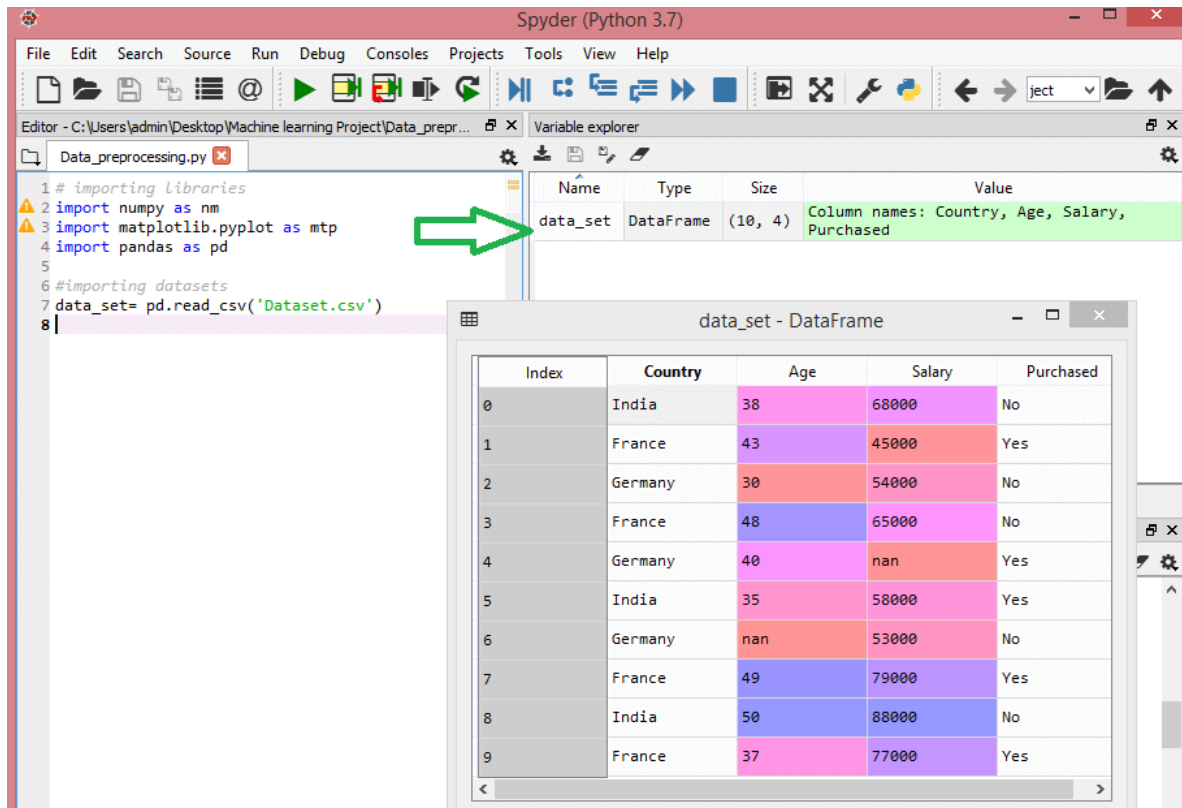
read_csv() function:

Now to import the dataset, we will use read_csv() function of pandas library, which is used to read a csv file and performs various operations on it. Using this function, we can read a csv file locally as well as through an URL.

We can use read_csv function as below:

1. `data_set = pd.read_csv('Dataset.csv')`

Here, **data_set** is a name of the variable to store our dataset, and inside the function, we have passed the name of our dataset. Once we execute the above line of code, it will successfully import the dataset in our code. We can also check the imported dataset by clicking on the section **variable explorer**, and then double click on **data_set**. Consider the below image:



As in the above image, indexing is started from 0, which is the default indexing in Python. We can also change the format of our dataset by clicking on the format option.

Extracting dependent and independent variables:

In machine learning, it is important to distinguish the matrix of features (independent variables) and dependent variables from dataset. In our dataset, there are three independent variables that are **Country**, **Age**, and **Salary**, and one is a dependent variable which is **Purchased**.

Extracting independent variable:

To extract an independent variable, we will use **iloc[]** method of Pandas library. It is used to extract the required rows and columns from the dataset.

1. `x= data_set.iloc[:, :-1].values`

Here we have taken all the rows with the last column only. It will give the array of dependent variables.

By executing the above code, we will get output as:

Output: In the above code, the first colon(:) is used to take all the rows, and the second colon(:) is for all the columns. Here we have used :-1, because we don't want to take the last column as it contains the dependent variable. So by doing this, we will get the matrix of features.

By executing the above code, we will get output as:

1. [['India' 38.0 68000.0]
2. ['France' 43.0 45000.0]
3. ['Germany' 30.0 54000.0]
4. ['France' 48.0 65000.0]
5. ['Germany' 40.0 nan]
6. ['India' 35.0 58000.0]
7. ['Germany' nan 53000.0]
8. ['France' 49.0 79000.0]
9. ['India' 50.0 88000.0]
10. ['France' 37.0 77000.0]]

As we can see in the above output, there are only three variables.

Extracting dependent variable:

To extract dependent variables, again, we will use Pandas .iloc[] method.

1. `y= data_set.iloc[:,3].values`

```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'], dtype=object)
```

4) Handling Missing data:

The next step of data preprocessing is to handle missing data in the datasets. If our dataset contains some missing data, then it may create a huge problem for our machine learning model. Hence it is necessary to handle missing values present in the dataset.

Ways to handle missing data:

There are mainly two ways to handle missing data, which are:

By deleting the particular row: The first way is used to commonly deal with null values. In this way, we just delete the specific row or column which consists of null values. But this way is not so efficient and removing data may lead to loss of information which will not give the accurate output.

By calculating the mean: In this way, we will calculate the mean of that column or row which contains any missing value and will put it on the place of missing value. This strategy is useful for the features which have numeric data such as age, salary, year, etc. Here, we will use this approach.

To handle missing values, we will use **Scikit-learn** library in our code, which contains various libraries for building machine learning models. Here we will use **Imputer** class of **sklearn.preprocessing** library. Below is the code for i

Program:

1. #handling missing data (Replacing missing data with the mean value)
2. from sklearn.preprocessing import Imputer
3. `imputer= Imputer(missing_values='NaN', strategy='mean', axis = 0)`
4. #Fitting imputer object to the independent variables x.
5. `imputerimputer= imputer.fit(x[:, 1:3])`
6. #Replacing missing data with the calculated mean value
7. `x[:, 1:3]= imputer.transform(x[:, 1:3])`

Output:

```
array([[ 'India', 38.0, 68000.0],
```

```
['France', 43.0, 45000.0],  
['Germany', 30.0, 54000.0],  
['France', 48.0, 65000.0],  
['Germany', 40.0, 65222.22222222222],  
['India', 35.0, 58000.0],  
['Germany', 41.111111111111114, 53000.0],  
['France', 49.0, 79000.0],  
['India', 50.0, 88000.0],  
['France', 37.0, 77000.0]], dtype=object
```

As we can see in the above output, the missing values have been replaced with the means of rest column values.

5) Encoding Categorical data:

Categorical data is data which has some categories such as, in our dataset; there are two categorical variable, **Country**, and **Purchased**.

Since machine learning model completely works on mathematics and numbers, but if our dataset would have a categorical variable, then it may create trouble while building the model. So it is necessary to encode these categorical variables into numbers.

For Country variable:

Firstly, we will convert the country variables into categorical data. So to do this, we will use **LabelEncoder()** class from **preprocessing** library.

Program:

1. #Categorical data
2. #for Country Variable
3. from sklearn.preprocessing import LabelEncoder
4. **label_encoder_x= LabelEncoder()**
5. **x[:, 0]= label_encoder_x.fit_transform(x[:, 0])**

Output:

```
Out[15]:  
array([[2, 38.0, 68000.0],
```

```
[0, 43.0, 45000.0],  
[1, 30.0, 54000.0],  
[0, 48.0, 65000.0],  
[1, 40.0, 65222.22222222222],  
[2, 35.0, 58000.0],  
[1, 41.111111111111114, 53000.0],  
[0, 49.0, 79000.0],  
[2, 50.0, 88000.0],  
[0, 37.0, 77000.0]], dtype=object)
```

Explanation:

In above code, we have imported **LabelEncoder** class of **sklearn library**. This class has successfully encoded the variables into digits.

But in our case, there are three country variables, and as we can see in the above output, these variables are encoded into 0, 1, and 2. By these values, the machine learning model may assume that there is some correlation between these variables which will produce the wrong output. So to remove this issue, we will use **dummy encoding**.

Dummy Variables:

Dummy variables are those variables which have values 0 or 1. The 1 value gives the presence of that variable in a particular column, and rest variables become 0. With dummy encoding, we will have a number of columns equal to the number of categories.

In our dataset, we have 3 categories so it will produce three columns having 0 and 1 values. For Dummy Encoding, we will use **OneHotEncoder** class of **preprocessing** library.

Program:

1. #for Country Variable

2. `from sklearn.preprocessing import LabelEncoder, OneHotEncoder`
3. `label_encoder_x= LabelEncoder()`
4. `x[:, 0]= label_encoder_x.fit_transform(x[:, 0])`
5. `#Encoding for dummy variables`
6. `onehot_encoder= OneHotEncoder(categorical_features= [0])`
7. `x= onehot_encoder.fit_transform(x).toarray()`

Output:

```
array([[0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.80000000e+01,
        6.80000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.30000000e+01,
        4.50000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 3.00000000e+01,
        5.40000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.80000000e+01,
        6.50000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 4.00000000e+01,
        6.52222222e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 3.50000000e+01,
        5.80000000e+04],
       [0.00000000e+00, 1.00000000e+00, 0.00000000e+00, 4.11111111e+01,
        5.30000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 4.90000000e+01,
        7.90000000e+04],
       [0.00000000e+00, 0.00000000e+00, 1.00000000e+00, 5.00000000e+01,
        8.80000000e+04],
       [1.00000000e+00, 0.00000000e+00, 0.00000000e+00, 3.70000000e+01,
        7.70000000e+04]])
```

As we can see in the above output, all the variables are encoded into numbers 0 and 1 and divided into three columns.

It can be seen more clearly in the variables explorer section, by clicking on x option as:

	0	1	2	3	4
0	0	0	1	38	68000
1	1	0	0	43	45000
2	0	1	0	30	54000
3	1	0	0	48	65000
4	0	1	0	40	65222.2
5	0	0	1	35	58000
6	0	1	0	41.1111	53000
7	1	0	0	49	79000
8	0	0	1	50	88000
9	1	0	0	37	77000

For Purchased Variable:

Program:

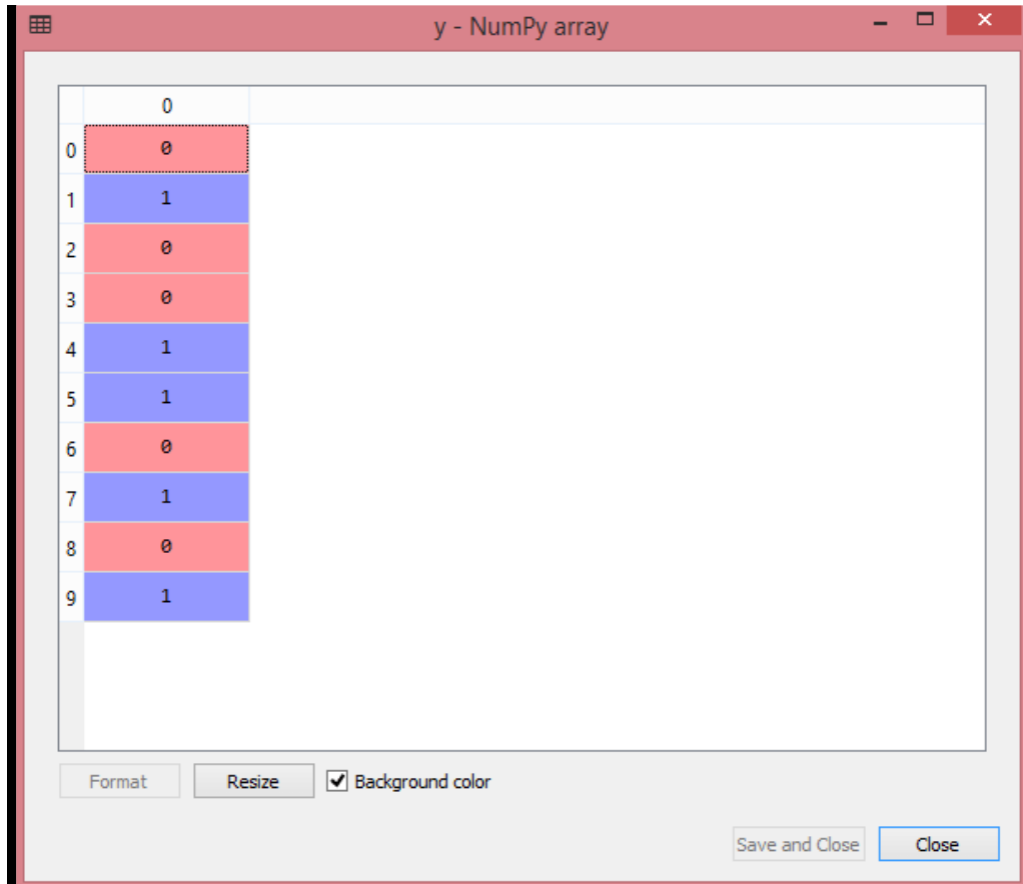
1. `labelencoder_y= LabelEncoder()`
2. `y= labelencoder_y.fit_transform(y)`

For the second categorical variable, we will only use labelencoder object of **LabelEncoder** class. Here we are not using **OneHotEncoder** class because the purchased variable has only two categories yes or no, and which are automatically encoded into 0 and 1.

Output:

```
Out[17]: array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

It can also be seen as:



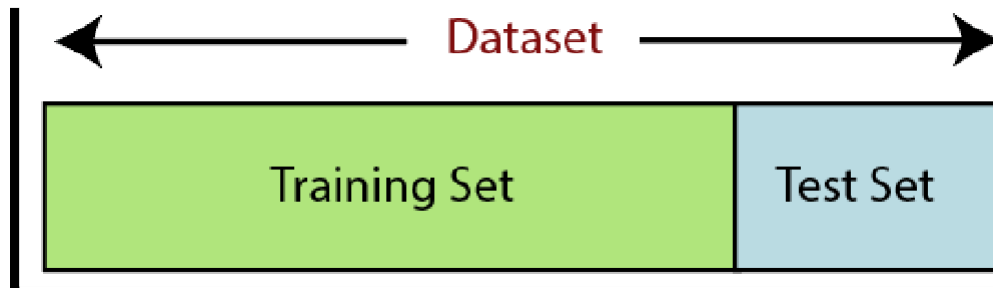
	0
0	0
1	1
2	0
3	0
4	1
5	1
6	0
7	1
8	0
9	1

6) Splitting the Dataset into the Training set and Test set

In machine learning data preprocessing, we divide our dataset into a training set and test set. This is one of the crucial steps of data preprocessing as by doing this, we can enhance the performance of our machine learning model.

Suppose, if we have given training to our machine learning model by a dataset and we test it by a completely different dataset. Then, it will create difficulties for our model to understand the correlations between the models.

If we train our model very well and its training accuracy is also very high, but we provide a new dataset to it, then it will decrease the performance. So we always try to make a machine learning model which performs well with the training set and also with the test dataset. Here, we can define these datasets as:



Training Set: A subset of dataset to train the machine learning model, and we already know the output.

Test set: A subset of dataset to test the machine learning model, and by using the test set, model predicts the output.

For splitting the dataset, we will use the below lines of code:

Program:

1. `from sklearn.model_selection import train_test_split`
2. `x_train, x_test, y_train, y_test= train_test_split(x, y, test_size= 0.2, random_state= 0)`

Explanation:

- In the above code, the first line is used for splitting arrays of the dataset into random train and test subsets.
- In the second line, we have used four variables for our output that are
 - **x_train:** features for the training data
 - **x_test:** features for testing data
 - **y_train:** Dependent variables for training data
 - **y_test:** Independent variable for testing data
- In **train_test_split() function**, we have passed four parameters in which first two are for arrays of data, and **test_size** is for specifying the size of the test set. The test_size maybe .5, .3, or .2, which tells the dividing ratio of training and testing sets.

- The last parameter **random_state** is used to set a seed for a random generator so that you always get the same result, and the most used value for this is 42.

Output:

By executing the above code, we will get 4 different variables, which can be seen under the variable explorer section.

Variable explorer			
Name	Type	Size	Value
data_set	DataFrame	(10, 4)	Column names: Country, Age, Salary, Purchased
x	float64	(10, 5)	[[0.0e+00 0.0e+00 1.0e+00 3.8e+01 6.8e+04] [1.0e+00 0.0e+00 0.0e+00 4 ...
x_test	float64	(2, 5)	[[0.0e+00 1.0e+00 0.0e+00 3.0e+01 5.4e+04] [0.0e+00 0.0e+00 1.0e+00 5 ...
x_train	float64	(8, 5)	[[0.00000000e+00 1.00000000e+00 0.00000000e+00 4.00000000e+01 6.5222 ...
y	int32	(10,)	[0 1 0 0 1 1 0 1 0 1]
y_test	int32	(2,)	[0 0]
y_train	int32	(8,)	[1 1 1 0 1 0 0 1]

As we can see in the above image, the x and y variables are divided into 4 different variables with corresponding values.

7) Feature Scaling

Feature scaling is the final step of data preprocessing in machine learning. It is a technique to standardize the independent variables of the dataset in a specific range. In feature scaling, we put our variables in the same range and in the same scale so that no any variable dominate the other variable.

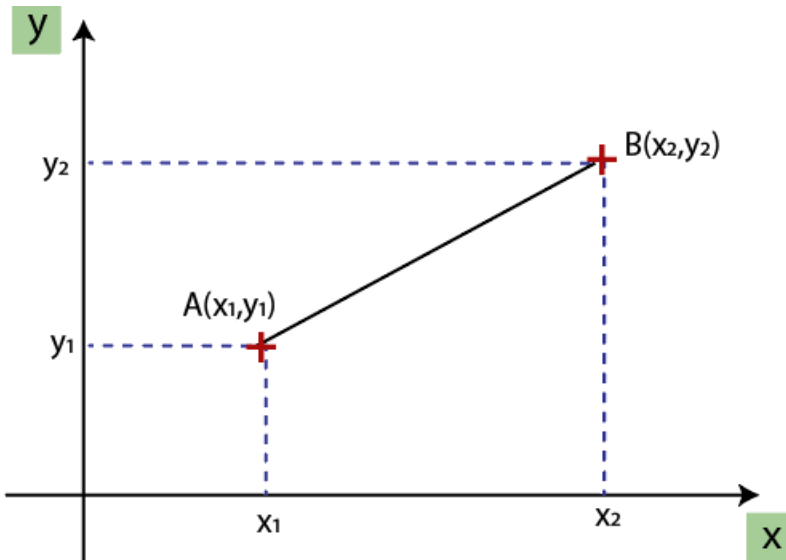
Consider the below dataset:

Index	Country	Age	Salary	Purchased
0	India	38	68000	No
1	France	43	45000	Yes
2	Germany	30	54000	No
3	France	48	65000	No
4	Germany	40	nan	Yes
5	India	35	58000	Yes
6	Germany	nan	53000	No
7	France	49	79000	Yes
8	India	50	88000	No
9	France	37	77000	Yes

Format Resize ☒ Background color ☒ Column min/max Save and Close Close

As we can see, the age and salary column values are not on the same scale. A machine learning model is based on **Euclidean distance**, and if we do not scale the variable, then it will cause some issue in our machine learning model.

Euclidean distance is given as:



Euclidean Distance Between A and B = $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

If we compute any two values from age and salary, then salary values will dominate the age values, and it will produce an incorrect result. So to remove this issue, we need to perform feature scaling for machine learning.

There are two ways to perform feature scaling in machine learning:

Standardization

$$X' = \frac{x - \text{mean}(x)}{a}$$

Diagram illustrating the Standardization formula:

- new value** points to X' .
- original value** points to x .
- mean** points to $\text{mean}(x)$.
- Standard deviation** points to a .

Normalization

$$X' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Diagram illustrating the Normalization formula:

- new value** points to X' .
- original value** points to x .

Here, we will use the standardization method for our dataset.

For feature scaling, we will import *StandardScaler* class of *sklearn.preprocessing* library as:

1. `from sklearn.preprocessing import StandardScaler`

Now, we will create the object of **StandardScaler** class for independent variables or features. And then we will fit and transform the training dataset.

Program:

1. `st_x= StandardScaler()`
2. `x_train= st_x.fit_transform(x_train)`

For test dataset, we will directly apply **transform()** function instead of **fit_transform()** because it is already done in training set.

1. `x_test= st_x.transform(x_test)`

Output:

By executing the above lines of code, we will get the scaled values for x_train and x_test as:

X_TRAIN:

x_train - NumPy array					
	0	1	2	3	4
0	-1	1.73205	-0.57735	-0.294607	0.133962
1	1	-0.57735	-0.57735	-0.930959	1.22627
2	1	-0.57735	-0.57735	0.341745	-1.7415
3	-1	1.73205	-0.57735	-0.0589215	-0.999562
4	1	-0.57735	-0.57735	1.61445	1.41175
5	1	-0.57735	-0.57735	1.40233	0.113352
6	-1	-0.57735	1.73205	-0.718842	0.391581
7	-1	-0.57735	1.73205	-1.35519	-0.535848

Format Resize ☒ Background color

Save and Close Close

x_test:

X_TEST:

	0	1	2	3	4
0	-1	1.73205	-0.57735	-0.294607	0.133962
1	1	-0.57735	-0.57735	-0.930959	1.22627
2	1	-0.57735	-0.57735	0.341745	-1.7415
3	-1	1.73205	-0.57735	-0.0589215	-0.999562
4	1	-0.57735	-0.57735	1.61445	1.41175
5	1	-0.57735	-0.57735	1.40233	0.113352
6	-1	-0.57735	1.73205	-0.718842	0.391581
7	-1	-0.57735	1.73205	-1.35519	-0.535848

As we can see in the above output, all the variables are scaled between values -1 to 1.

IMPORT LIBRARIES:

NumPy is a very popular python library for large multi-dimensional array and matrix processing, with the help of a large collection of high-level mathematical functions. It is very useful for fundamental scientific computations in Machine Learning. It is particularly useful for linear algebra, Fourier transform, and random number capabilities. High-end libraries like TensorFlow uses NumPy internally for manipulation of Tensors.

SciPy is a very popular library among Machine Learning enthusiasts as it contains different modules for optimization, linear algebra, integration and statistics. There is a difference between the SciPy library and the SciPy stack. The SciPy is one of the core packages that make up the SciPy stack. SciPy is also very useful for image manipulation.

Scikit-learn is one of the most popular ML libraries for classical ML algorithms. It is built on top of two basic Python libraries, viz., NumPy and SciPy. Scikit-learn supports most of the supervised and unsupervised learning algorithms. Scikit-learn can also be used for data-mining and data-analysis, which makes it a great tool who is starting out with ML.

TensorFlow is a very popular open-source library for high performance numerical computation developed by the Google Brain team in Google. As the

name suggests, Tensorflow is a framework that involves defining and running computations involving tensors. It can train and run deep neural networks that can be used to develop several AI applications. TensorFlow is widely used in the field of deep learning research and application.

Keras is a very popular Machine Learning library for Python. It is a high-level neural networks API capable of running on top of TensorFlow, CNTK, or Theano. It can run seamlessly on both CPU and GPU. Keras makes it really for ML beginners to build and design a Neural Network. One of the best thing about Keras is that it allows for easy and fast prototyping.

PyTorch is a popular open-source Machine Learning library for Python based on Torch, which is an open-source Machine Learning library that is implemented in C with a wrapper in Lua. It has an extensive choice of tools and libraries that support Computer Vision, Natural Language Processing(NLP), and many more ML programs. It allows developers to perform computations on Tensors with GPU acceleration and also helps in creating computational graphs.

Pandas is a popular Python library for data analysis. It is not directly related to Machine Learning. As we know that the dataset must be prepared before training. In this case, Pandas comes handy as it was developed specifically for data extraction and preparation. It provides high-level data structures and wide variety tools for data analysis. It provides many inbuilt methods for grouping, combining and filtering data.

Matplotlib is a very popular Python library for data visualization. Like Pandas, it is not directly related to Machine Learning. It particularly comes in handy when a programmer wants to visualize the patterns in the data. It is a 2D plotting library used for creating 2D graphs and plots. A module named pyplot makes it easy for programmers for plotting as it provides features to control line styles, font properties, formatting axes, etc. It provides various kinds of graphs and plots for data visualization, viz., histogram, error charts, bar charts, etc.,

Import Dependencies

```
%matplotlib inline

# Start Python Imports
import math, time, datetime

import random as rd

# Data Manipulation
import numpy as np
import pandas as pd

# Visualization
import matplotlib.pyplot as plt
import missingno as msno
import seaborn as sns
plt.style.use('seaborn-whitegrid')

# Preprocessing
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, label_binarize

# Machine learning
import catboost
from sklearn.model_selection import train_test_split
from sklearn import model_selection, tree, preprocessing, metrics, linear_model
from sklearn.svm import LinearSVC
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LinearRegression, LogisticRegression, SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from catboost import CatBoostClassifier, Pool, cv

# Let's be rebels and ignore warnings for now
import warnings
warnings.filterwarnings('ignore')
```

```
/tmp/ipykernel_20/160898924.py:16: MatplotlibDeprecationWarning: The
seaborn
styles shipped by Matplotlib are deprecated since 3.6, as they no longer corres
pond to the styles shipped by seaborn. However, they will remain available as
'seaborn-v0_8-<style>'. Alternatively, directly use the seaborn API instead.
plt.style.use('seaborn-whitegrid')
```

HELPING FUNCTIONS:

Activation Function: Activation functions introduce non-linearity into the neural networks, allowing them to learn complex patterns. Common activation functions include Sigmoid, Tanh, ReLU (Rectified Linear Unit), and Softmax.

Loss Functions (Cost Functions): Loss functions measure the difference between the predicted values and the actual values (labels) in the training data. They are used to train machine learning models by minimizing the error. Common loss functions include Mean Squared Error (MSE) for regression problems and Cross-Entropy Loss for classification problems.

Optimization Algorithms: Optimization algorithms are used to minimize the loss function during the training of machine learning models. Gradient Descent, Stochastic Gradient Descent (SGD), and variants like Adam and RMSprop are commonly used optimization algorithms.

Regularization Techniques: Regularization methods are used to prevent overfitting, which occurs when a model performs well on the training data but poorly on unseen data. L1 and L2 regularization add penalty terms to the loss function, encouraging the model to use simpler (smoother) solutions. Dropout is another technique where randomly selected neurons are ignored during training to prevent overfitting.

Data Preprocessing: Data preprocessing techniques include data cleaning, feature scaling, and feature engineering. Data is often normalized or standardized to ensure that all features have the same scale. Categorical variables are encoded into numerical values through techniques like one-hot encoding.

Cross-Validation: Cross-validation is a technique used to assess the performance and generalizability of a machine learning model. It involves dividing the dataset into multiple subsets (folds) and training the model on different subsets while evaluating it on the remaining data.

Evaluation Metrics: Evaluation metrics are used to measure the performance of machine learning models. Common evaluation metrics include accuracy, precision, recall, F1-score, and area under the ROC curve (AUC-ROC) for classification problems. For regression problems, metrics like Mean Absolute Error (MAE), Mean Squared Error (MSE), and R-squared are used.

Feature Selection: Feature selection techniques help in selecting the most relevant features for the model, improving its performance and reducing overfitting. Common methods include Recursive Feature Elimination (RFE) and feature importance from tree-based models.

Ensemble Methods: Ensemble methods combine predictions from multiple machine learning models to improve overall performance. Bagging, Boosting, and Stacking are popular ensemble techniques.

These helping functions and techniques are fundamental to the practice of machine learning and are applied across various types of machine learning algorithms and models.

```
def systematic_sample(df,
    size):length = len(df)
    interval = length //
    size
    rd.seed(None)
    first = rd.randint(0, interval)
    indexes = np.arange(first, length, step =
    interval)
    return df.iloc[indexes]

def missing_values_table(df):
    # Total missing values
    mis_val = df.isnull().sum()

    # Percentage of missing values
    mis_val_percent = 100 * df.isnull().sum() / len(df)

    # Make a table with the results
    mis_val_table = pd.concat([mis_val, mis_val_percent], axis=1)

    # Rename the columns
    mis_val_table_ren_columns =
    mis_val_table.rename( columns = {0 : 'Missing
    Values', 1 : '% of Total Values'})

    # Sort the table by percentage of missing descending
    mis_val_table_ren_columns = mis_val_table_ren_columns[
    mis_val_table_ren_columns.iloc[:,1] != 0].sort_values(
    '% of Total Values', ascending=False).round(1)

    # Print some summary information
    print ("Your selected dataframe has " + str(df.shape[1]) + "
    columns.\n""There are " +
    str(mis_val_table_ren_columns.shape[0]) +
    " columns that have missing values.")

    # Return the dataframe with missing information
    return
mis_val_table_ren_columns

def fill_na(df):
    for col in df.columns:
        if df[col].isnull().any():
            if df[col].dtypes in ["float", "int"]:
                df[col].fillna(df[col].mean(), inplace=True)
            else:
                df[col].fillna(df[col].mode()[0], inplace=True)
```

```
def plot_count_dist(data, bin_df, label_column, target_column, figsize=(20, 5), use_bin_df=False):
```

Function to plot counts and distributions of a label variable and target variable side by side.

::param_data:: = target dataframe

::param_bin_df:: = binned dataframe for countplot

::param_label_column:: = binary labelled column

```

::param_target_column:: = column you want to view counts and distributions
::param_figsize:: = size of figure (width, height)
::param_use_bin_df:: = whether or not to use the bin_df, default False
"""
if use_bin_df:
    fig = plt.figure(figsize=figsize)
    plt.subplot(1, 2, 1)
    sns.countplot(y=target_column, data=bin_df);
    plt.subplot(1, 2, 2)
    sns.distplot(data.loc[data[label_column] == 1][target_column],
                  kde_kws={"label": "Survived"});
    sns.distplot(data.loc[data[label_column] == 0][target_column],
                  kde_kws={"label": "Did not survive"});
else:
    fig = plt.figure(figsize=figsize)
    plt.subplot(1, 2, 1)
    sns.countplot(y=target_column, data=data);
    plt.subplot(1, 2, 2)
    sns.distplot(data.loc[data[label_column] == 1][target_column],
                  kde_kws={"label": "Survived"});
    sns.distplot(data.loc[data[label_column] == 0][target_column],
                  kde_kws={"label": "Did not survive"});

# Function that runs the requested algorithm and returns the accuracy metrics
def fit_ml_algo(algo, X_train, y_train, cv):

    # One Pass
    model = algo.fit(X_train, y_train)
    acc = round(model.score(X_train, y_train) * 100, 2)

    # Cross Validation
    train_pred = model_selection.cross_val_predict(algo,
                                                    X_train,
                                                    y_train,
                                                    cv=cv,
                                                    n_jobs = -
                                                    1)

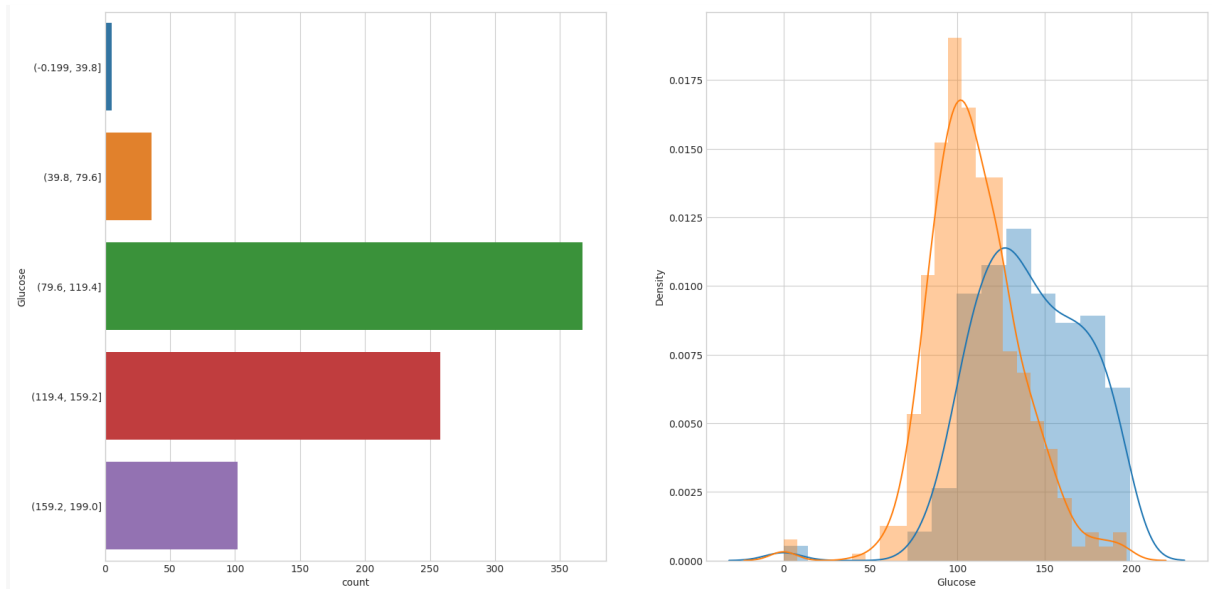
    # Cross-validation accuracy metric
    acc_cv = round(metrics.accuracy_score(y_train, train_pred) * 100, 2)

    return train_pred, acc,
    acc_cv

# Feature Importance
def feature_importance(model, data):
    """
    Function to show which features are most important in the model.
    ::param_model:: Which model to use?
    ::param_data:: What data to
    use? """
    fea_imp = pd.DataFrame({'imp': model.feature_importances_, 'col':
data.columns}) fea_imp = fea_imp.sort_values(['imp', 'col'],
ascending=[True, False]).iloc[-30:]

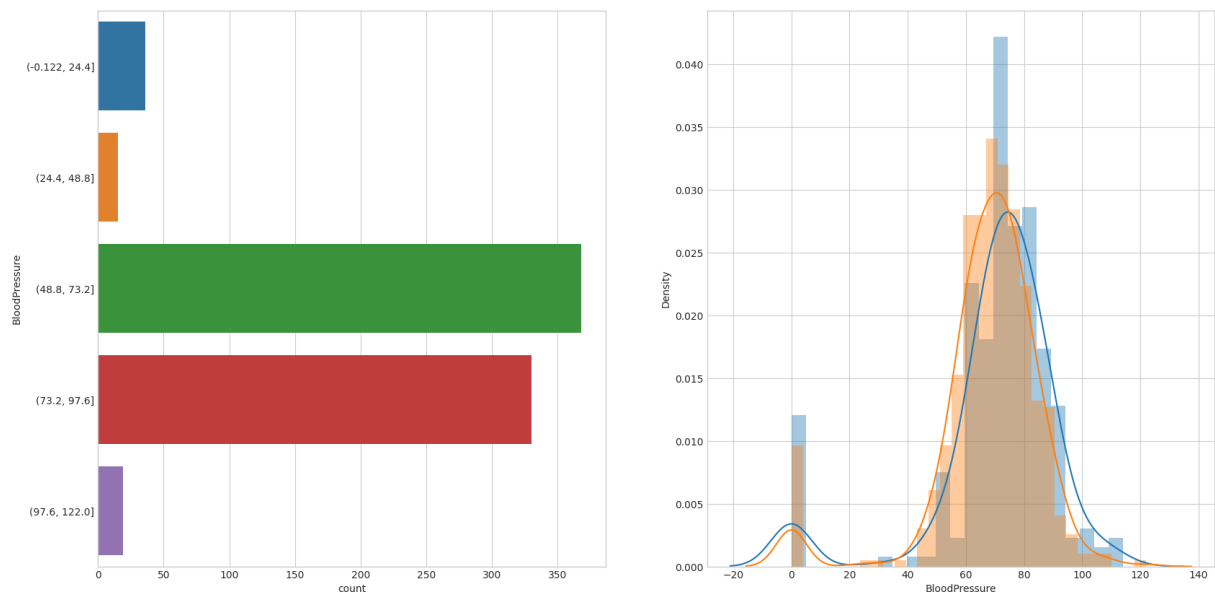
```

```
return fea_imp
#plt.savefig('catboost_feature_importance.png')
```



Visualise the Fare bin counts as well as the Fare distribution versus Survived.

```
plot_count_dist(data=df,
                bin_df=df_dis,
                label_column='Outcome',
                target_column='BloodPressure',
                figsize=(20,10),
                use_bin_df=True)
```

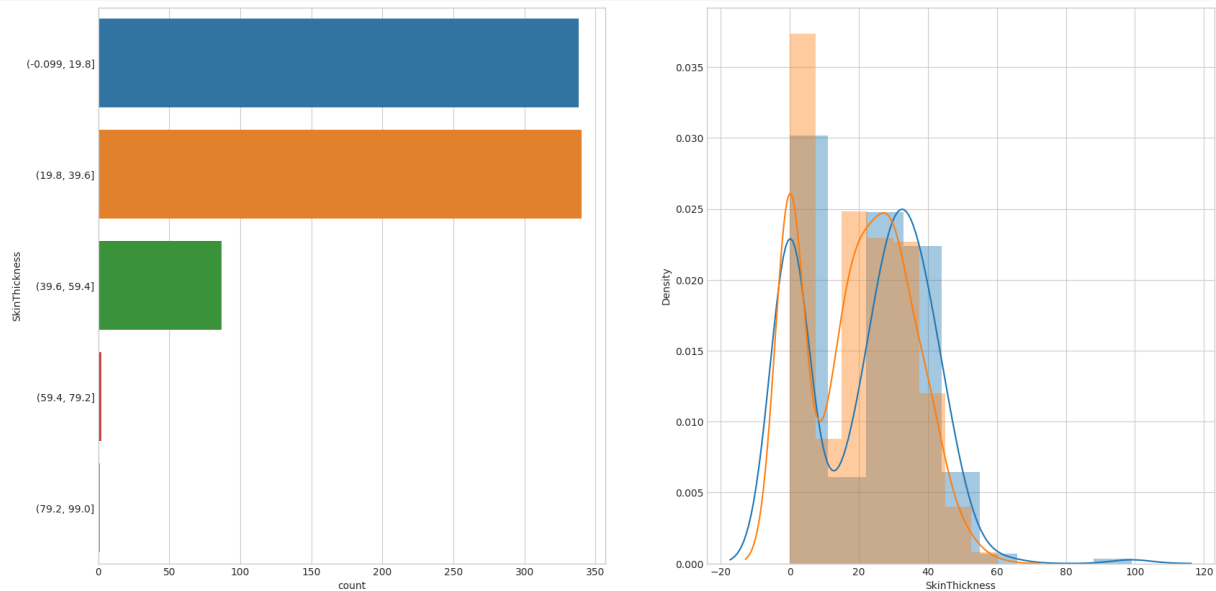


Visualise the Fare bin counts as well as the Fare distribution versus Survived.

```
plot_count_dist(data=df,
```

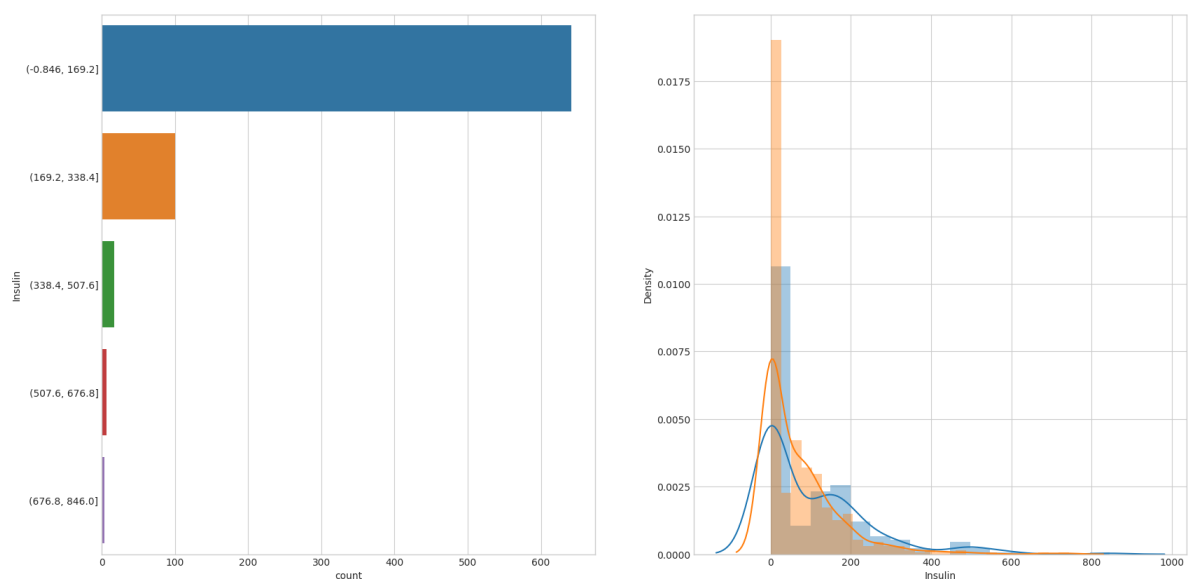
```
bin_df=df_dis,
label_column='Outcome',
target_column='SkinThickness',
```

```
figsize=(20,10),
use_bin_df=True)
```



Visualise the Fare bin counts as well as the Fare distribution versus Survived.

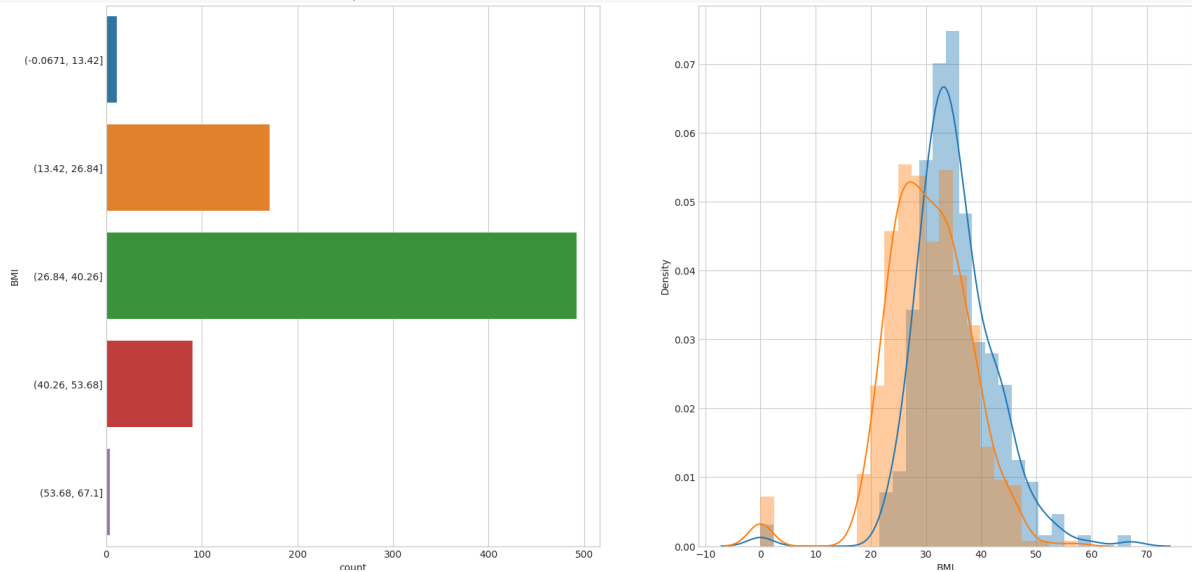
```
plot_count_dist(data=df,
bin_df=df_dis,
label_column='Outcome',
target_column='Insulin',
figsize=(20,10),
use_bin_df=True)
```



Visualise the Fare bin counts as well as the Fare distribution versus Survived.

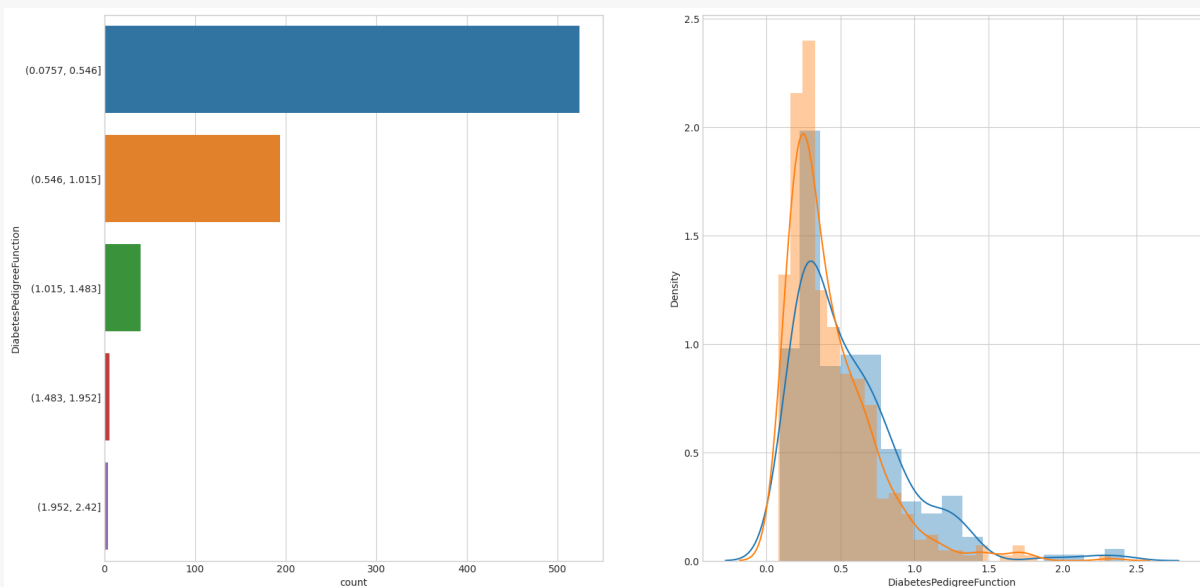
```
plot_count_dist(data=df,
bin_df=df_dis,
label_column='Outcome',
```

```
target_column='BMI',
figsize=(20,10),
use_bin_df=True)
```



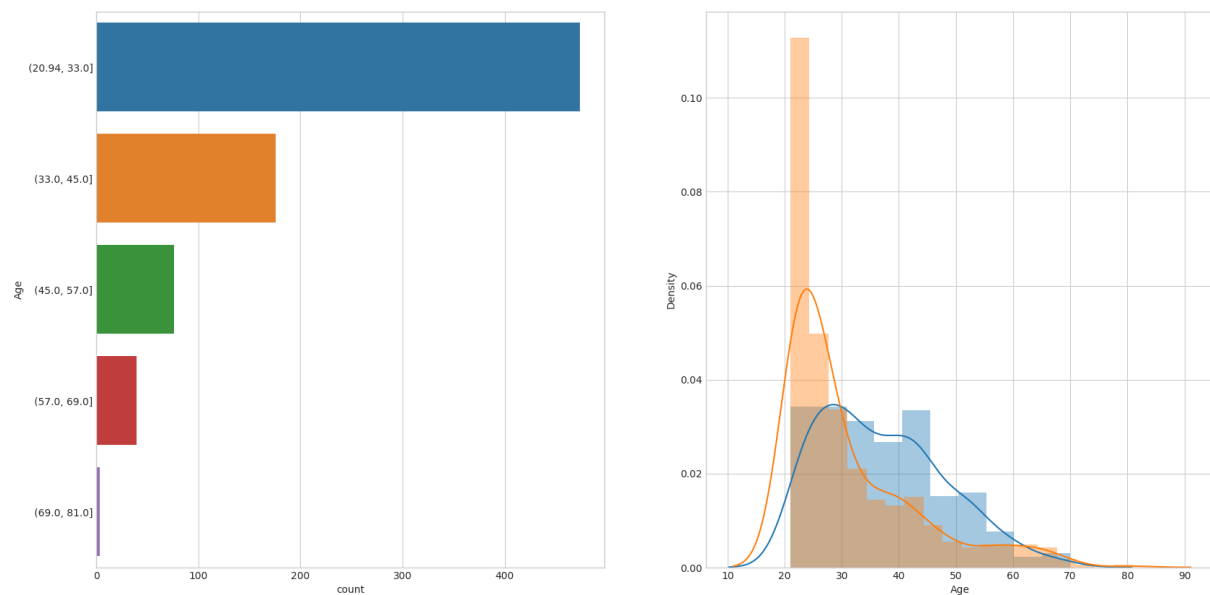
Visualise the Fare bin counts as well as the Fare distribution versus Survived.

```
plot_count_dist(data=df,
bin_df=df_dis,
label_column='Outcome',
target_column='DiabetesPedigreeFunction',
figsize=(20,10),
use_bin_df=True)
```



Visualise the Fare bin counts as well as the Fare distribution versus Survived.

```
plot_count_dist(data=df,
bin_df=df_dis,
label_column='Outcome',
target_column='Age',
figsize=(20,10),
use_bin_df=True)
```



431	0	F	F	F	T	F	F	F	F	F	.	F	T	F	F	F	F	T	F	F	F
		a	a	a	r	a	a	a	a	a	.	a	r	a	a	a	a	r	a	a	a
		s	s	s	e	s	s	s	s	s	.	s	e	s	s	s	s	e	s	s	s
584	1	F	F	F	F	F	F	F	F	T	.	F	T	F	F	F	F	F	T	F	F
		a	a	a	a	a	a	a	a	r	.	a	r	a	a	a	a	a	r	a	a
		s	s	s	s	s	s	s	s	e	.	s	e	s	s	s	s	s	e	s	s
737	0	F	F	F	F	F	F	F	F	T	.	F	T	F	F	F	F	T	F	F	F
		a	a	a	a	a	a	a	a	r	.	a	r	a	a	a	a	r	a	a	a
		s	s	s	s	s	s	s	s	e	.	s	e	s	s	s	s	e	s	s	s

```
df_con_enc = pd.get_dummies(df_con,
columns=one_hot_cols)df_con_enc.head()
```

MACHINE LEARNING:

Machine learning is a branch of artificial intelligence (AI) and computer science which focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy. Machine learning is an important component of the growing field of data science. Through the use of statistical methods, algorithms are trained to make classifications or predictions, and to uncover key insights in data mining projects. These insights subsequently drive decision making within applications and businesses, ideally impacting key growth

metrics. As big data continues to expand and grow, the market demand for data scientists will increase. They will be required to help identify the most relevant business questions and the data to answer them. Machine learning algorithms are typically created using frameworks that accelerate solution development, such as TensorFlow and PyTorch.

1. **A Decision Process:** In general, machine learning algorithms are used to make a prediction or classification. Based on some input data, which can be labeled or unlabeled, your algorithm will produce an estimate about a pattern in the data.
2. **An Error Function:** An error function evaluates the prediction of the model. If there are known examples, an error function can make a comparison to assess the accuracy of the model.
3. **A Model Optimization Process:** If the model can fit better to the data points in the training set, then weights are adjusted to reduce the discrepancy between the known example and the model estimate. The algorithm will repeat this “evaluate and optimize” process, updating weights autonomously until a threshold of accuracy has been met.

COMMON MACHINE LEARNING ALGORITHMS:

- **Neural networks:** Neural networks simulate the way the human brain works, with a huge number of linked processing nodes. Neural networks are good at recognizing patterns and play an important role in applications including natural language translation, image recognition, speech recognition, and image creation.
- **Linear regression:** This algorithm is used to predict numerical values, based on a linear relationship between different values. For example, the technique could be used to predict house prices based on historical data for the area.
- **Logistic regression:** This supervised learning algorithm makes predictions for categorical response variables, such as “yes/no” answers to questions. It can be used for applications such as classifying spam and quality control on a production line.

- **Clustering:** Using unsupervised learning, clustering algorithms can identify patterns in data so that it can be grouped. Computers can help data scientists by identifying differences between data items that humans have overlooked.
- **Decision trees:** Decision trees can be used for both predicting numerical values (regression) and classifying data into categories. Decision trees use a branching sequence of linked decisions that can be represented with a tree diagram. One of the advantages of decision trees is that they are easy to validate and audit, unlike the black box of the neural network.
- **Random forests:** In a random forest, the machine learning algorithm predicts a value or category by combining the results from a number of decision trees.

```
# Select the dataframe we want to use first for predictions
selected_df = df_con_enc
selected_df.head()

# Split the dataframe into data and labels
X_train = selected_df.drop('Outcome', axis=1) # data
y_train = selected_df.Outcome # labels

# Shape of the data (without labels)
X_train.shape

(768, 1254)
linkcode
X_train.head()

# Shape of the labels
y_train.shape
(768,)

# Logistic Regression
start_time = time.time()
train_pred_log, acc_log, acc_cv_log = fit_ml_algo(LogisticRegression(),
                                                    X_train,
                                                    n,
                                                    y_train,
                                                    10)

log_time = (time.time() - start_time)
print("Accuracy: %s" % acc_log)
print("Accuracy CV 10-Fold: %s" % acc_cv_log)
print("Running Time: %s" % datetime.timedelta(seconds=log_time))
```

Accuracy: 96.09

Accuracy CV 10-Fold:

67.45 Running Time:

0:00:02.451250# k-

Nearest Neighbours

start_time = time.time()

```
train_pred_knn, acc_knn, acc_cv_knn = fit_ml_algo(KNeighborsClassifier(),
                                                    X_train,
                                                    n,
                                                    y_train,
                                                    10)
```

knn_time = (time.time() - start_time)

print("Accuracy: %s" % acc_knn)

print("Accuracy CV 10-Fold: %s" %

acc_cv_knn)

print("Running Time: %s" % datetime.timedelta(seconds=knn_time))

Accuracy: 75.26

Accuracy CV 10-Fold:

66.8 Running Time:

0:00:00.358875

Gaussian Naive Bayes

start_time = time.time()

```
train_pred_gaussian, acc_gaussian, acc_cv_gaussian =
fit_ml_algo(GaussianNB(),
```

```
            X_train
```

```
            n,
```

```
            y_train
```

```
            n,10)
```

gaussian_time = (time.time() - start_time)

print("Accuracy: %s" % acc_gaussian)

print("Accuracy CV 10-Fold: %s" %

acc_cv_gaussian)

```

print("Running Time: %s" %
datetime.timedelta(seconds=gaussian_time))Accuracy: 94.79
Accuracy CV 10-Fold: 59.9
Running Time: 0:00:00.219412
# Linear SVC
start_time = time.time()
train_pred_svc, acc_linear_svc, acc_cv_linear_svc = fit_ml_algo(LinearSVC(),
                                                                X_train,
                                                                n,
                                                                y_train,
                                                                10)
linear_svc_time = (time.time() - start_time)
print("Accuracy: %s" % acc_linear_svc)
print("Accuracy CV 10-Fold: %s" %
acc_cv_linear_svc)
print("Running Time: %s" % datetime.timedelta(seconds=linear_svc_time))
Accuracy: 100.0
Accuracy CV 10-Fold:
65.23 Running Time:
0:00:00.268296#
Stochastic Gradient
Descent start_time =
time.time()
train_pred_sgd, acc_sgd, acc_cv_sgd =
fit_ml_algo(SGDClassifier(),
X_train,
y_train,
n,10)
sgd_time = (time.time() - start_time)
print("Accuracy: %s" % acc_sgd)
print("Accuracy CV 10-Fold: %s" %
acc_cv_sgd)
print("Running Time: %s" % datetime.timedelta(seconds=sgd_time))
Accuracy: 100.0
Accuracy CV 10-Fold:
63.93 Running Time:
0:00:00.437200# Decision
Tree Classifier start_time
= time.time()
train_pred_dt, acc_dt, acc_cv_dt = fit_ml_algo(DecisionTreeClassifier(),
                                                                X_train,
                                                                n,
                                                                y_train,
                                                                ),
)
dt_time = (time.time() - start_time)
print("Accuracy: %s" % acc_dt)

```

```
print("Accuracy CV 10-Fold: %s" %  
acc_cv_dt)  
print("Running Time: %s" % datetime.timedelta(seconds=dt_time))  
Accuracy: 100.0  
Accuracy CV 10-Fold:  
61.85 Running Time:  
0:00:00.596504# Gradient  
Boosting Trees start_time  
= time.time()  
train_pred_gbt, acc_gbt, acc_cv_gbt = fit_ml_algo(GradientBoostingClassifier(),  
X_train y_train,10)
```

```
gbt_time = (time.time() - start_time)  
print("Accuracy: %s" % acc_gbt)  
print("Accuracy CV 10-Fold: %s" %  
acc_cv_gbt)  
print("Running Time: %s" % datetime.timedelta(seconds=gbt_time))  
Accuracy: 81.9  
Accuracy CV 10-Fold:  
63.54 Running Time:  
0:00:06.448709
```

CATBOOST ALGORITHM:

CatBoost is a supervised machine learning method that is used by the [Train Using AutoML](#) tool and uses decision trees for classification and regression. As its name suggests, CatBoost has two main features, it works with categorical data (the Cat) and it uses gradient boosting (the Boost). Gradient boosting is a process in which many decision trees are constructed iteratively. Each subsequent tree improves the result of the previous tree, leading to better results. CatBoost improves on the original gradient boost method for a faster implementation.

CatBoost overcomes a limitation of other decision tree-based methods in which, typically, the data must be pre-processed to convert categorical string variables to numerical values, one-hot encodings, and so on. This method can directly consume a combination of categorical and non-categorical explanatory variables without preprocessing. It preprocesses as part of the algorithm. CatBoost uses a method called ordered encoding to encode categorical features. Ordered encoding considers the target statistics from all the rows prior to a data point to calculate a value to replace the categorical feature. Another unique characteristic of CatBoost is that it uses symmetric trees. This means that at every depth level, all the decision nodes use the same split condition.

CatBoost can also be faster than other methods such as [XGBoost](#). It retains certain features—such as cross-validation, regularization, and missing value support—from the prior algorithms. This method performs well with both small data and large data.

```
systematic sample(X train, 5)
```

[illegible]

P r e g n a n c i e s _0	P r e g n a n c i e s _1	P r e g n a n c i e s _2	P r e g n a n c i e s _3	P r e g n a n c i e s _4	P r e g n a n c i e s _5	P r e g n a n c i e s _6	P r e g n a n c i e s _7	P r e g n a n c i e s _8	P r e g n a n c i e s _9	· · ·	A g e - 6 3	A g e - 6 4	A g e - 6 5	A g e - 6 6	A g e - 6 7	A g e - 6 8	A g e - 6 9	A g e - 7 0	A g e - 7 2	A g e - 8 1
19 5	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	T r u e	F a l s e	F a l s e	F a l s e	F a l s e	· · ·	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e
34 8	F a l s e	F a l s e	F a l s e	T r u e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	· · ·	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e
50 1	F a l s e	F a l s e	F a l s e	T r u e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	· · ·	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e
65 4	F a l s e	T r u e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	· · ·	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e	F a l s e

systematic_sample(y_train, 5)

149 0

302 0

455 1

608 0

761 1

Nam Outcome, dtype:
e: int64

Define the categorical features for the CatBoost

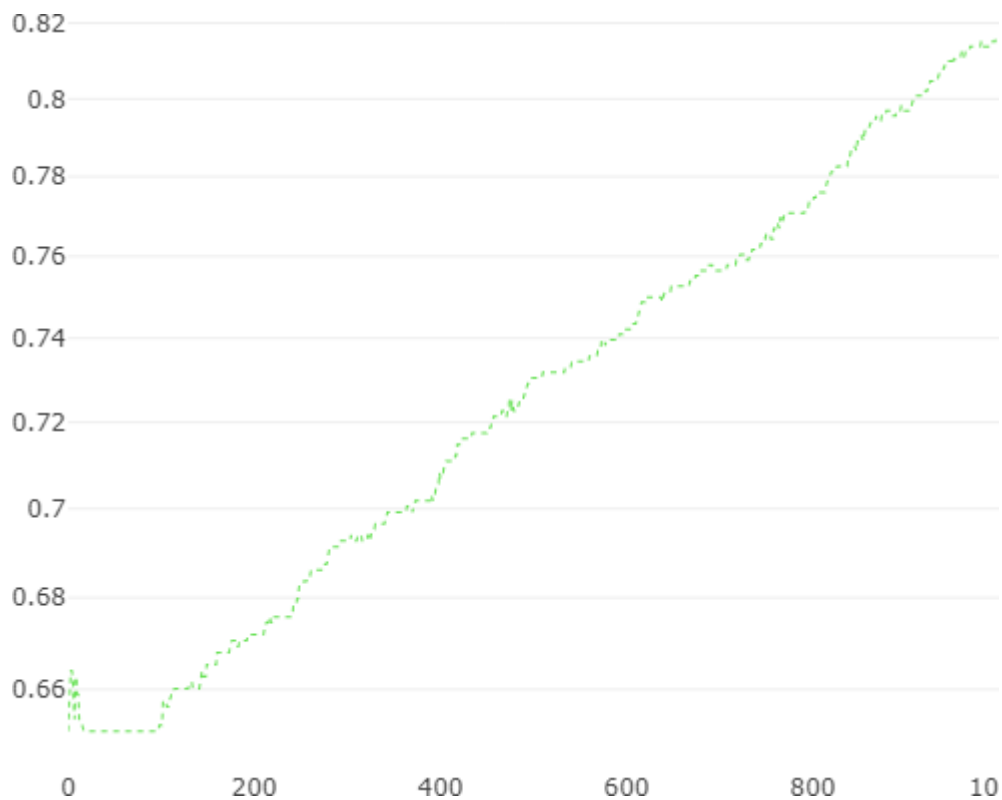
```

catboost_model = CatBoostClassifier(iterations=1000,
                                    custom_loss=['Accuracy
'],
                                    loss_function='Logloss')

# Fit CatBoost model
catboost_model.fit(train_pool,
                    plot=True)

# CatBoost accuracy
acc_catboost = round(catboost_model.score(X_train, y_train) * 100, 2)

```



469:	learn: 0.5395230	total: 2.34s	remaining: 2.64s
470:	learn: 0.5394029	total: 2.34s	remaining: 2.63s
471:	learn: 0.5392298	total: 2.35s	remaining: 2.63s
472:	learn: 0.5390315	total: 2.36s	remaining: 2.63s
473:	learn: 0.5388890	total: 2.37s	remaining: 2.63s
474:	learn: 0.5387269	total: 2.37s	remaining: 2.62s
475:	learn: 0.5384723	total: 2.38s	remaining: 2.62s
476:	learn: 0.5383353	total: 2.38s	remaining: 2.62s
477:	learn: 0.5382355	total: 2.39s	remaining: 2.61s
478:	learn: 0.5380677	total: 2.4s	remaining: 2.61s
479:	learn: 0.5378452	total: 2.4s	remaining: 2.61s
480:	learn: 0.5376842	total: 2.41s	remaining: 2.6s
481:	learn: 0.5375600	total: 2.42s	remaining: 2.6s
482:	learn: 0.5374369	total: 2.42s	remaining: 2.6s


```

487:    learn: 0.5367199          total: 2.46s    remaining: 2.59s
488:    learn: 0.5365787          total: 2.47s    remaining: 2.58s
489:    learn: 0.5364203          total: 2.48s    remaining: 2.58s
490:    learn: 0.5362019          total: 2.48s    remaining: 2.57s
491:    learn: 0.5361098          total: 2.49s    remaining: 2.57s
492:    learn: 0.5358471          total: 2.49s    remaining: 2.56s
493:    learn: 0.5357295          total: 2.5s     remaining: 2.56s
494:    learn: 0.5355541          total: 2.5s     remaining: 2.55s
495:    learn: 0.5353691          total: 2.51s    remaining: 2.55s
496:    learn: 0.5352261          total: 2.51s    remaining: 2.54s
497:    learn: 0.5351270          total: 2.52s    remaining: 2.54s
498:    learn: 0.5350399          total: 2.52s    remaining: 2.53s
499:    learn: 0.5349238          total: 2.53s    remaining: 2.53s
500:    learn: 0.5348211          total: 2.53s    remaining: 2.52s
501:    learn: 0.5347660          total: 2.54s    remaining: 2.52s
502:    learn: 0.5346753          total: 2.54s    remaining: 2.51s
503:    learn: 0.5344722          total: 2.55s    remaining: 2.51s
504:    learn: 0.5341997          total: 2.56s    remaining: 2.5s
505:    learn: 0.5340434          total: 2.56s    remaining: 2.5s

print("---CatBoost Metrics---")
print("Accuracy:
{}".format(acc_catboost))
print("Accuracy cross-validation 10-Fold: {}".format(acc_cv_catboost))
print("Running Time:
{}".format(datetime.timedelta(seconds=catboost_time)))

---CatBoost
Metrics---
Accuracy: 81.64
Accuracy cross-validation 10-Fold:
66.54Running Time:
0:00:59.138368
linkcode
models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression', 'Naive
              Bayes', 'Stochastic Gradient Decent', 'Linear
              SVC', 'Decision Tree', 'Gradient Boosting
              Trees', 'CatBoost'],
    'Score': [
        acc_knn,
        acc_log,
        acc_gaussian
        , acc_sgd,
        acc_linear_s
        vc, acc_dt,
        acc_gbt,

        acc_catboost
    ])
print("---Regular Accuracy Scores---")
models.sort_values(by='Score', ascending=False)

```

Model	Score	
3	Stochastic Gradient Decent	100.00
4	Linear SVC	100.00
5	Decision Tree	100.00
1	Logistic Regression	96.09
2	Naive Bayes	94.79
6	Gradient Boosting Trees	81.90
7	CatBoost	81.64
0	KNN	75.26

```

cv_models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression', 'Naive Bayes',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree', 'Gradient Boosting Trees',
              'CatBoost'],
    'Score': [
        acc_cv_knn,
        acc_cv_log,
        acc_cv_gaussian,
        acc_cv_sgd,
        acc_cv_linear_svc,
        acc_cv_dt,
        acc_cv_gbt,
        acc_cv_catboost
    ]
})
print('---Cross-validation Accuracy Scores---')

```

```
cv_models.sort_values(by='Score', ascending=False)
```

---Cross-validation Accuracy Scores---

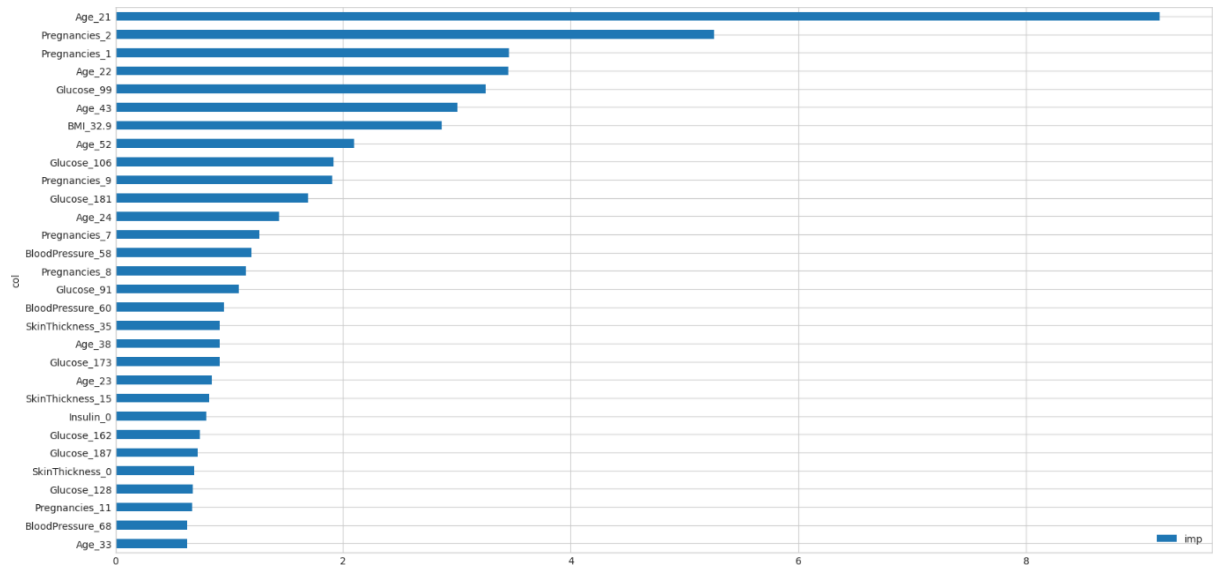
Model	Score	
1	Logistic Regression	67.45
0	KNN	66.80
7	CatBoost	66.54
4	Linear SVC	65.23
3	Stochastic Gradient Decent	63.93
6	Gradient Boosting Trees	63.54
5	Decision Tree	61.85

```
# Plot the feature importance scores
```

```
feature_importance(catboost_model, X_train)
```

--	--	--

imp	col	
1214	0.628039	Age_33
173	0.628329	BloodPressure_68



CONCLUSION:

We have gone through many more different kind of model and we analyse that the top most efficient algorithm among them till now is the random forest one, that is producing best output and result from all the tried model.