

# CAPSTONE PROJECT

HEALTHCARE

# DESCRIPTION

- NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases.
- The dataset used in this project is originally from NIDDK. The objective is to predict whether or not a patient has diabetes, based on certain diagnostic measurements included in the dataset.
- Build a model to accurately predict whether the patients in the dataset have diabetes or not.

# PROJECT TASK

WEEK 1

# DATA EXPLORATION(TASK 1)

- Perform descriptive analysis. Understand the variables and their corresponding values. On the columns below, a value of zero does not make sense and thus indicates missing value:
- Glucose
- Blood Pressure
- Skin Thickness
- Insulin
- BMI

```
[1]: # Capstone Project - Healthcare
import pandas as pd
import numpy as np

h_data = pd.read_csv("health care diabetes.csv")

[2]: # Project Task: Week 1

[3]: # checking the shape of dataset
h_data.shape

[4]: (768, 9)

[5]: # Viewing the main heading of the dataset
h_data.head()

[5]: 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

[6]: # Checking whether data contains any null values
h_data.isnull().any()

[6]: Pregnancies      False
Glucose          False
BloodPressure    False
SkinThickness    False
Insulin          False
BMI              False
DiabetesPedigreeFunction False
Age              False
Outcome          False
dtype: bool

[7]: # No null values in the dataset

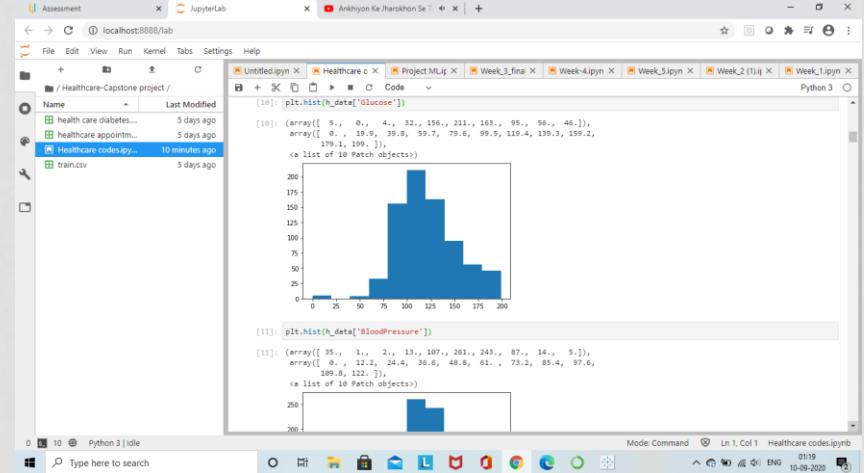
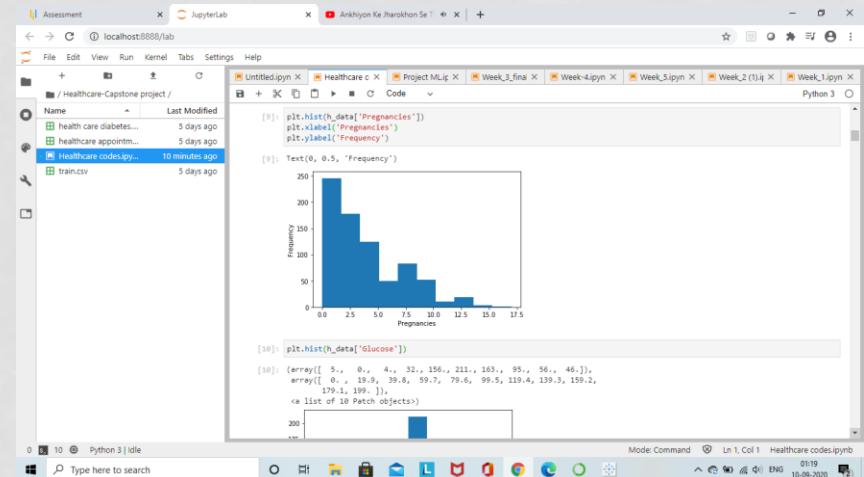
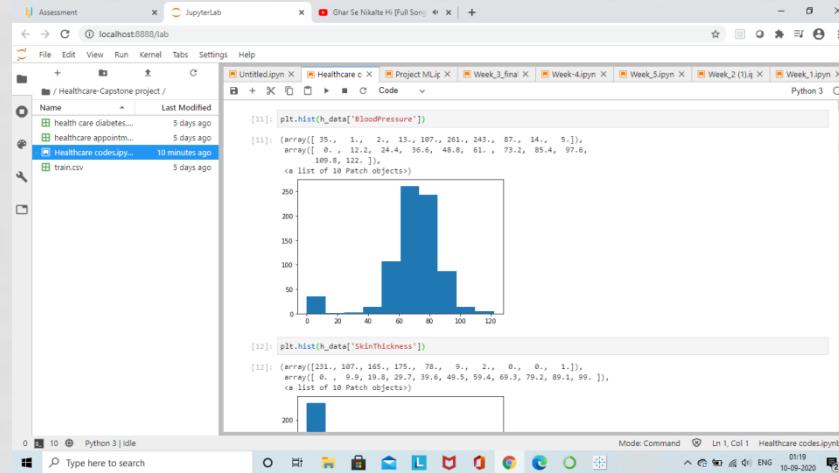
[8]: # Exploring the variables using histograms
# Importing required libraries
#%matplotlib inline
import matplotlib.pyplot as plt
from matplotlib import style

[9]: plt.hist(h_data['Pregnancies'])
plt.xlabel('Pregnancies')
plt.ylabel('Frequency')

[9]: Text(0, 85, 'Frequency')
```

# TASK 2

- Visually explore these variables using histograms. Treat the missing values accordingly.



# TASK 2

JupyterLab | localhost:8888/lab | Ghar Se Nikalte Hi [Full Song] | +

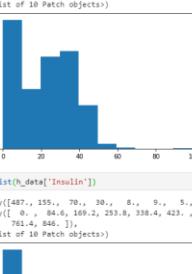
File Edit View Run Kernel Tabs Settings Help

/ Healthcare-Capstone project /

- Name Last Modified
- health care diabetes... 5 days ago
- healthcare appointm... 5 days ago
- Healthcare codes.ipynb** 10 minutes ago
- train.csv 5 days ago

```
[12]: plt.hist(h_data['SkinThickness'])
```

```
[12]: (array([231., 107., 165., 175., 78., 9., 2., 0., 0., 1.]), array([ 0.,  9.9, 19.8, 29.7, 39.6, 49.5, 59.4, 69.3, 79.2, 89.1, 99.]), <list of 10 Patch objects>)
```



```
[13]: plt.hist(h_data['Insulin'])
```

```
[13]: (array([487., 155., 70., 30., 8., 9., 5., 1., 2., 1.]), array([ 0.,  84.6, 169.2, 253.8, 338.4, 423., 507.6, 592.2, 676.8, 761.4, 846.]), <list of 10 Patch objects>)
```



Python 3 | idle Mode Command Ln 1, Col 1 Healthcare codes.ipynb

Type here to search

10 0 ENG 0110 10-09-2020

JupyterLab | localhost:8888/lab | Ghar Se Nikalte Hi [Full Song] | +

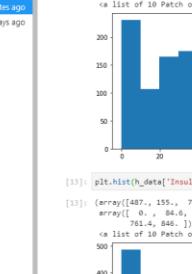
File Edit View Run Kernel Tabs Settings Help

/ Healthcare-Capstone project /

- Name Last Modified
- health care diabetes... 5 days ago
- healthcare appointm... 5 days ago
- Healthcare codes.ipynb** 10 minutes ago
- train.csv 5 days ago

```
[12]: plt.hist(h_data['SkinThickness'])
```

```
[12]: (array([231., 107., 165., 175., 78., 9., 2., 0., 0., 1.]), array([ 0.,  9.9, 19.8, 29.7, 39.6, 49.5, 59.4, 69.3, 79.2, 89.1, 99.]), <list of 10 Patch objects>)
```



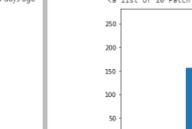
```
[13]: plt.hist(h_data['Insulin'])
```

```
[13]: (array([487., 155., 70., 30., 8., 9., 5., 1., 2., 1.]), array([ 0.,  84.6, 169.2, 253.8, 338.4, 423., 507.6, 592.2, 676.8, 761.4, 846.]), <list of 10 Patch objects>)
```



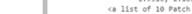
```
[14]: plt.hist(h_data['BMI'])
```

```
[14]: (array([ 0.,  0.,  15., 156., 268., 224., 78., 12., 3., 1.]), array([ 0.,  6.72, 13.42, 20.13, 26.84, 33.59, 40.26, 46.97, 53.68, 60.39, 67.12]), <list of 10 Patch objects>)
```



```
[15]: plt.hist(h_data['DiabetesPedigreeFunction'])
```

```
[15]: (array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.]), array([ 0.078, 0.122, 0.546, 0.7008, 1.0248, 1.249, 1.4932, 1.7274, 1.9516, 2.1859, 2.42]), <list of 10 Patch objects>)
```



Python 3 | idle Mode Command Ln 1, Col 1 Healthcare codes.ipynb

Type here to search

10 0 ENG 0119 10-09-2020

# TASK 2

Three screenshots of JupyterLab environments showing data analysis and visualization for a healthcare dataset.

**Screenshot 1:** Displays histograms for 'DiabetesPedigreeFunction', 'Age', and 'Outcome'.

```
[15]: plt.hist(h_data['DiabetesPedigreeFunction'])  
[16]: plt.hist(h_data['Age'])  
[17]: plt.hist(h_data['Outcome'])
```

**Screenshot 2:** Displays histograms for 'Age' and 'Outcome'.

```
[18]: plt.hist(h_data['Age'])  
[19]: plt.hist(h_data['Outcome'])
```

**Screenshot 3:** Displays histograms for 'Outcome' and shows the distribution of variables for integer or float values.

```
[20]: plt.hist(h_data['Outcome'])  
[21]: # Checking the variable for its integer or float values  
h_data.info()
```

The screenshots show the following distributions:

- DiabetesPedigreeFunction:** A right-skewed histogram with a peak around 0.8-1.0.
- Age:** A right-skewed histogram with a peak around 25-30.
- Outcome:** A bimodal histogram with peaks at 0 and 1.
- Age (Screenshot 2):** A right-skewed histogram with a peak around 25-30.
- Outcome (Screenshot 2):** A bimodal histogram with peaks at 0 and 1.
- Outcome (Screenshot 3):** A bimodal histogram with peaks at 0 and 1.
- Variables (Screenshot 3):** The following table shows the distribution of variables:

Variable	Non-null Count	Data Type
Pregnancies	768	int64
Glucose	768	int64
BloodPressure	768	int64
SkinThickness	768	int64
Insulin	768	int64

# TASK 3

- There are integer and float data type variables in this dataset. Create a count (frequency) plot describing the data types and the count of variables.

The image shows two side-by-side Jupyter Notebook interfaces. Both notebooks are titled 'Assessment' and are connected to the same kernel ('localhost:8888/lab'). The top notebook, 'Healthcare codes.ipynb', displays code and its output for checking variable types and value counts. The bottom notebook, also 'Healthcare codes.ipynb', shows similar code for variables like 'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', and 'Insulin'. Both notebooks are running Python 3.

**Top Notebook (Healthcare codes.ipynb):**

```
[18]: # Checking the variable for its integer or float values
h_data.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   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

[19]: h_data['Pregnancies'].value_counts().head(5)

[19]: 1    135
0    111
2    105
3     78
4     68
Name: Pregnancies, dtype: int64

[20]: h_data['Glucose'].value_counts().head(5)

[20]: 100    17
99     17
129    14
123    14
111    14
Name: Glucose, dtype: int64

[21]: h_data['BloodPressure'].value_counts().head(5)

[21]: 70    57
74    52
66    49
78    46
72    44
Name: BloodPressure, dtype: int64

[22]: h_data['SkinThickness'].value_counts().head(5)

[22]: 0    227
32    31
30    27
27    23
23    22
Name: SkinThickness, dtype: int64

[23]: h_data['Insulin'].value_counts().head(5)

[23]: 0    374
```

**Bottom Notebook (Healthcare codes.ipynb):**

```
[18]: # Checking the variable for its integer or float values
h_data.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   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

[19]: h_data['Pregnancies'].value_counts().head(5)

[19]: 1    135
0    111
2    105
3     78
4     68
Name: Pregnancies, dtype: int64

[20]: h_data['Glucose'].value_counts().head(5)

[20]: 100    17
99     17
129    14
123    14
111    14
Name: Glucose, dtype: int64

[21]: h_data['BloodPressure'].value_counts().head(5)

[21]: 70    57
74    52
66    49
78    46
72    44
Name: BloodPressure, dtype: int64

[22]: h_data['SkinThickness'].value_counts().head(5)

[22]: 0    227
32    31
30    27
27    23
23    22
Name: SkinThickness, dtype: int64

[23]: h_data['Insulin'].value_counts().head(5)

[23]: 0    374
```

# TASK 3

- There is no null values in the dataset.
- There are two float variables and rest are integer variables including Target variable.

```
[24]: h_data['BMI'].value_counts().head(5)
[24]: 32.0    13
      31.6    12
      31.2    12
      0.0     11
      33.3    10
Name: BMI, dtype: int64

[25]: h_data['DiabetesPedigreeFunction'].value_counts().head(5)
[25]: 0.254    6
      0.256    6
      0.259    5
      0.238    5
      0.207    5
Name: DiabetesPedigreeFunction, dtype: int64

[26]: h_data['Age'].value_counts().head(5)
[26]: 22     72
      21     63
      25     48
      24     46
      23     38
Name: Age, dtype: int64
```

# PROJECT TASK

WEEK 2

# TASK 1

- Check the balance of the data by plotting the count of outcomes by their value. Describe your findings and plan future course of action.

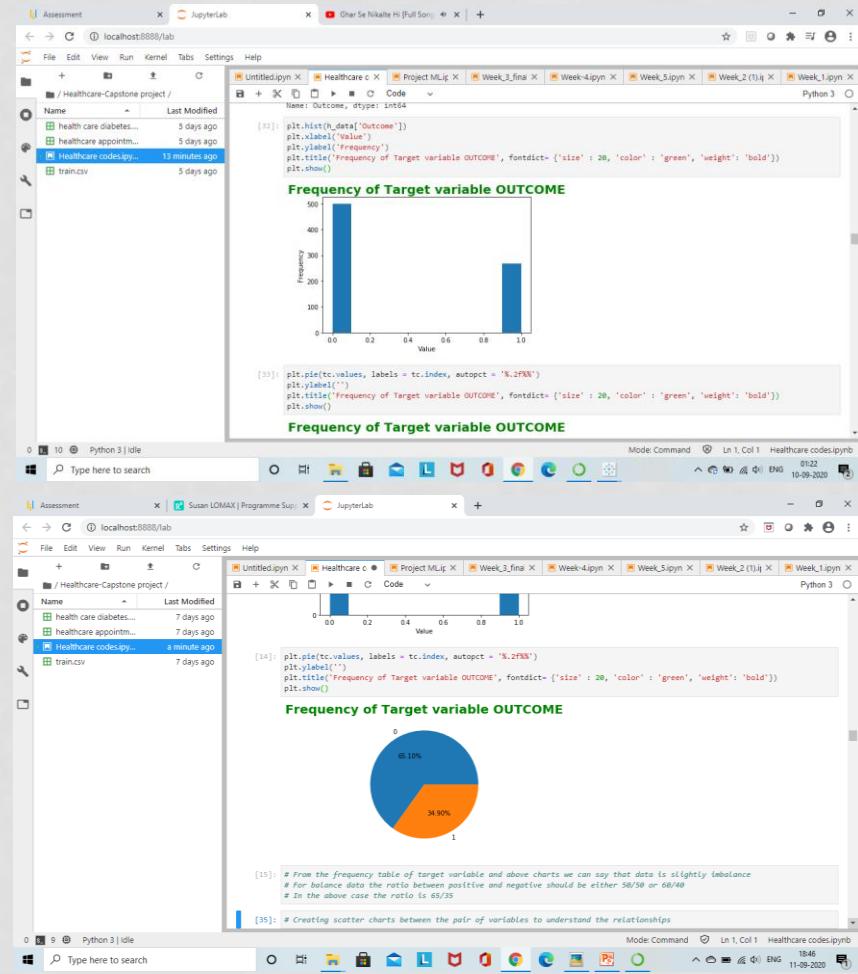
```
[27]: # Target variable  
h_data['Outcome'].value_counts()  
[27]: 0    500  
1     268  
Name: Outcome, dtype: int64  
  
[28]: # Project Task: Week 2  
  
[29]: h_data.describe().transpose()  
[29]:  
count      mean       std      min      25%      50%      75%      max  
Pregnancies 768.0  3.845052  3.349578  0.000  1.00000  3.0000  6.00000  17.00  
Glucose    768.0 120.894531 91.972618 0.000 99.00000 117.0000 140.25000 199.00  
BloodPressure 768.0 69.105469 19.355807 0.000 62.00000 72.0000 80.00000 122.00  
SkinThickness 768.0 20.536458 15.932218 0.000 0.00000 23.0000 32.00000 99.00  
Insulin    768.0 75.794979 115.244002 0.000 0.00000 30.5000 127.25000 846.00  
BMI        768.0 31.992578 7.864160 0.000 27.30000 32.0000 36.60000 67.10  
DiabetesPedigreeFunction 768.0 0.471876 0.311329 0.070 0.24375 0.3725 0.62625 2.42  
Age        768.0 33.240885 11.760232 21.000 24.00000 29.0000 41.00000 81.00  
Outcome   768.0 0.346958 0.476951 0.000 0.00000 0.0000 1.00000 1.00  
  
[30]: # Checking the balance of the data by plotting the count of outcome by its value
```

```
[30]:  
Mode: Command Ln 1, Col 1 Healthcare codes.ipynb  
  
[31]: # Target variable  
tc = h_data['Outcome'].value_counts()  
tc  
[31]: 0    500  
1     268  
Name: Outcome, dtype: int64  
  
[32]: plt.hist(h_data['Outcome'])  
plt.xlabel('Value')  
plt.ylabel('Frequency')  
plt.title('Frequency of Target variable OUTCOME', fontdict= {'size': 20, 'color': 'green', 'weight': 'bold'})  
plt.show()
```

Frequency of Target variable OUTCOME

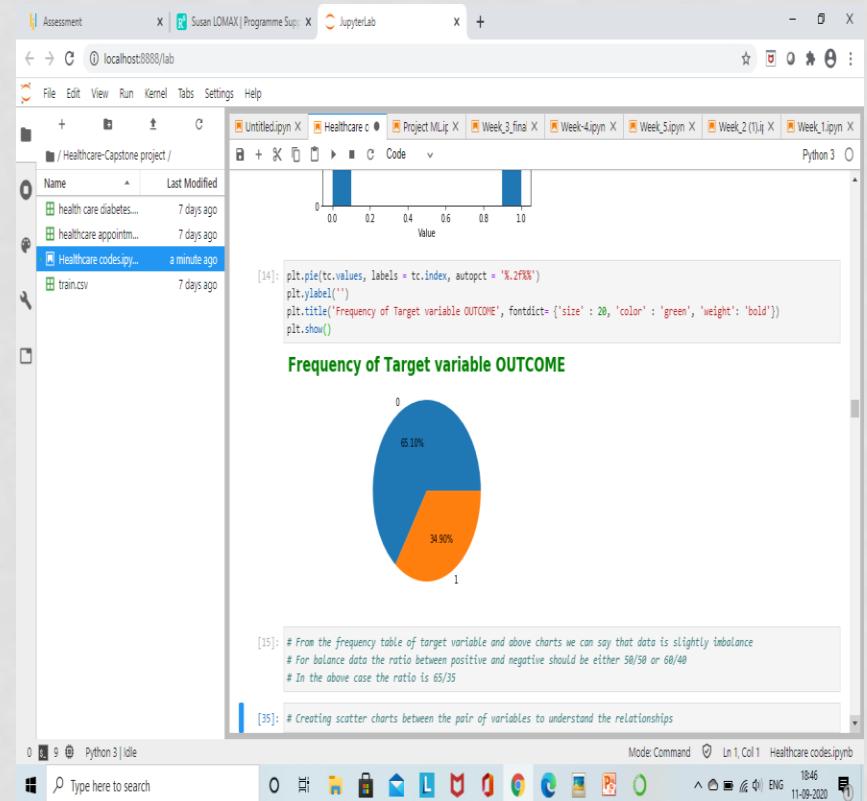
# TASK 1

- Out of 768 people, without diabetes people are 500 and with diabetes people are 268.
- From the pie chart it is clear that without diabetes population is 65% and with diabetic population is 35% approx.



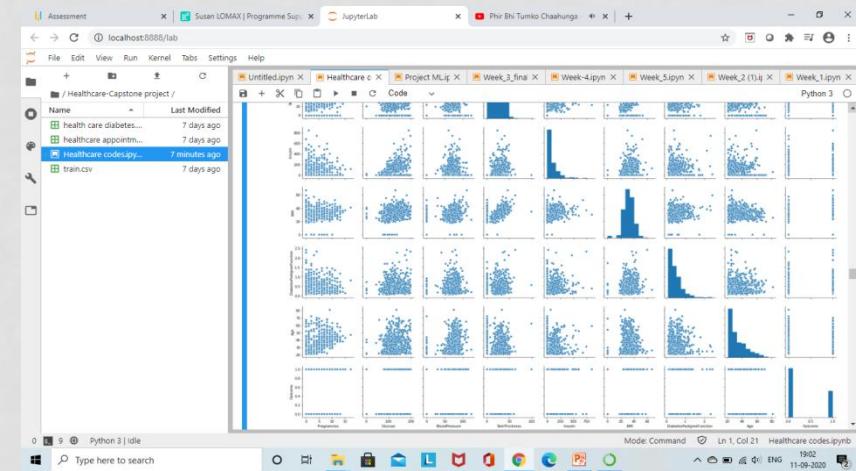
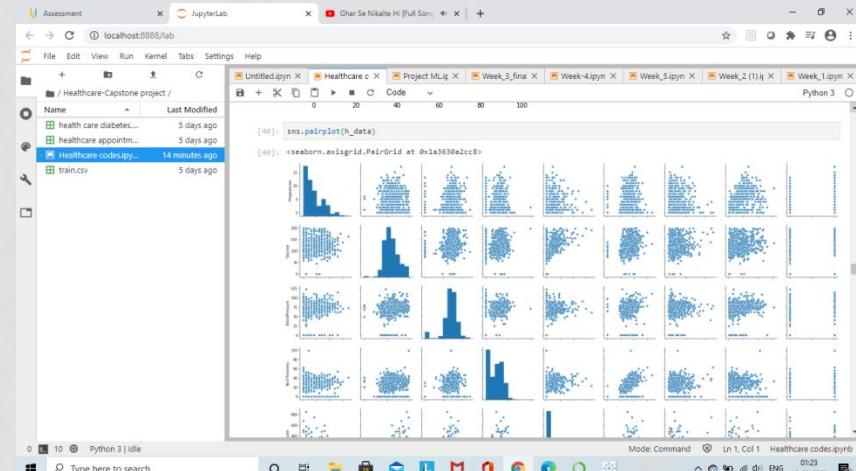
# TASK 1

- As we can see in this pie chart, data is slightly imbalanced, as perfectly balanced data has 50/50 or 60/40 distribution. Here, the ratio in data is 65/35.



# TASK 2

- Create scatter charts between the pair of variables to understand the relationships.
- Describe your findings.



# TASK 2

- There is a strong positive correlation between Age and Blood Pressure.
- There is no correlation between Glucose and Skin Thickness.

The screenshot shows a Jupyter Notebook interface with several tabs at the top: Assessment, JupyterLab, and a video player tab. The main area displays a Python script for performing correlation analysis and selecting the top 6 best features from a dataset. The code includes imports for pandas, NumPy, and scikit-learn's SelectKBest and chi2 modules. It loads a dataset named 'h\_data' and applies the SelectKBest class to extract the top 6 features based on chi-squared test scores. The resulting DataFrame 'featureScores' is then concatenated with the columns DataFrame 'dfcolumns' and sorted by score.

```
[41]: # From the above scatter pair plot we can see that:  
# There is strong positive correlation between Age and BloodPressure  
# There is no correlation between Glucose and SkinThickness  
# There is positive correlation between BMI and SkinThickness  
  
[42]: # Performing correlation analysis  
# Univariate analysis  
  
[43]: # Importing required libraries  
from sklearn.feature_selection import SelectKBest  
from sklearn.feature_selection import chi2  
  
[44]: x = h_data.iloc[:,0:8]  
y = h_data.iloc[:,1]  
  
[45]: # Applying SelectKBest class to extract top 6 best features  
bestfeatures = SelectKBest(score_func=chi2, k=6)  
fit = bestfeatures.fit(x,y)  
  
[46]: dfscores = pd.DataFrame(fit.scores_ )  
dfcolumns = pd.DataFrame(x.columns)  
  
[47]: # Concat two DataFrames for better visualization  
featureScores = pd.concat([dfcolumns, fscores], axis=1)  
featureScores.columns = ['col', 'Score']  
  
[48]: featureScores
```

# TASK 3

- Perform correlation analysis. Visually explore it using a heat map.

JupyterLab - localhost:8888/lab

```
[50]: # The correlation of each feature in the dataset
corrmat = h_data.corr()
top_corr_features = corrmat.index
corrmat
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.129459	0.141332	-0.081672	-0.073535	0.017683	-0.035323	0.044341	0.221886
Glucose	0.129459	1.000000	0.152590	0.057328	0.031187	0.221071	0.137337	0.036514	0.466581
BloodPressure	0.141332	0.152590	1.000000	0.207371	0.088893	0.281805	0.041265	0.299528	0.065068
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.406783	0.392373	0.183928	-0.139971	0.074751
Insulin	-0.073535	0.031187	0.088893	0.436700	1.000000	0.197859	0.180571	-0.042160	0.130548
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	0.140647	0.036242	0.292695
DiabetesPedigreeFunction	-0.033523	0.137337	0.041365	0.183928	0.165071	0.140647	1.000000	0.033561	0.173844
Age	0.544341	0.265314	0.239528	-0.138970	-0.042160	0.036042	0.033561	1.000000	0.238356
Outcome	0.221896	0.466581	0.065068	0.074752	0.130548	0.292695	0.173844	0.238356	1.000000

```
[51]: # Creating heatmap to visually explore the correlation of each feature in dataset
sns.heatmap(h_data[top_corr_features].corr(), annot=True, cmap='GnBu')
```

```
[52]: cm=plt.subplots.AxesSubplot at 0x1a507c7d9c8:
```

JupyterLab - localhost:8888/lab

```
[41]: # From the above scatter pair plot we can see that:
# There is strong positive correlation between Age and BloodPressure
# There is no correlation between Glucose and SkinThickness
# There is positive correlation between BMI and SkinThickness
```

```
[42]: # Performing correlation analysis
# Univariate analysis
```

```
[43]: # Importing required libraries
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import Chi2
```

```
[44]: x = h_data.iloc[:,0:8]
y = h_data.iloc[:,1]
```

```
[45]: # Applying SelectKBest class to extract top 6 best features
bestfeatures = SelectKBest(score_func=chi2, k=6)
fit = bestfeatures.fit(x,y)
```

```
[46]: dfScores = pd.DataFrame(fit.scores_)
dfColumns = pd.DataFrame(x.columns)
```

```
[47]: # Concat two DataFrames for better visualization
featureScores = pd.concat([dfColumns, dfScores], axis=1)
featureScores.columns = ['col', 'Score']
```

```
[48]: featureScores
```

```
[49]: col Score
```

JupyterLab - localhost:8888/lab

```
[49]: FeatureScores.columns = ['col','Score']
```

col	Score
0	111.519691
1	1411.887041
2	17.053737
3	53.108040
4	2175.565273
5	127.669343
6	5.392682
7	181.303689

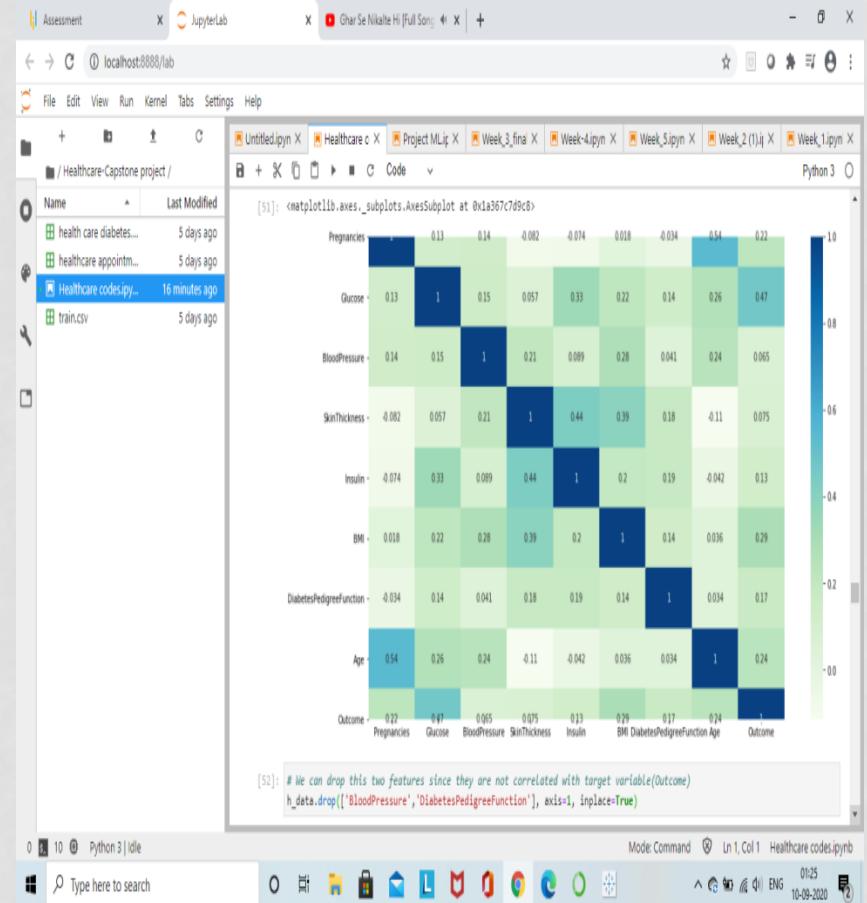
```
[50]: # Print the 6 best features
print(featureScores.nlargest(6,'Score'))
```

col	Score
4	2175.565273
1	1411.887041
2	17.053737
5	127.669343
0	111.519691
3	53.108040

```
[51]: # The correlation of each feature in the dataset
```

# TASK 3

- Strong correlation between independent features which are Insulin and Glucose, BMI and Skin Thickness, Age and Pregnancies.
- Skin Thickness and Insulin has almost no correlation with Pregnancies.



# TASK 3

- Outcome has the strong positive correlation with Glucose.
- Outcome has almost no correlation with Diabetes Pedigree Function.

The screenshot shows a Jupyter Notebook interface with several code cells and a file browser.

**File Browser:** Shows files in the "Healthcare-Capstone project/" directory:

- healthcare diabetes...
- healthcare appointm...
- Healthcare codes.py... (selected)
- transcsv

**Code Cells:**

- [52]:  
```# We can drop this two features since they are not correlated with target variable(Outcome)  
h\_data.drop(['BloodPressure', 'DiabetesPedigreeFunction'], axis=1, inplace=True)```
- [53]:  
```h\_data.head()```
- [53]:  

|   | Pregnancies | Glucose | SkinThickness | Insulin | BMI  | Age | Outcome |
|---|-------------|---------|---------------|---------|------|-----|---------|
| 0 | 6           | 148     | 35            | 0       | 33.6 | 50  | 1       |
| 1 | 1           | 85      | 29            | 0       | 26.6 | 31  | 0       |
| 2 | 8           | 183     | 0             | 0       | 23.3 | 32  | 1       |
| 3 | 1           | 89      | 23            | 94      | 28.1 | 21  | 0       |
| 4 | 0           | 137     | 35            | 168     | 43.1 | 33  | 1       |
- [54]:  
```# Project Task: Week 3```
- [55]:  
```# Strategies for model building-  
# The Target variable(dependent variable) that is 'Outcome' is binary that is it has only two values 0 or 1  
# As dependent variable(Target variable) is binary therefore Logistic regression is the appropriate regression analysis```
- [56]:  
```# Importing required class  
from sklearn.linear\_model import LogisticRegression```
- [57]:  
```# Split the dataset in features and target variable  
x = h\_data.drop(["Outcome"], axis=1)  
y = h\_data["Outcome"]```

**Bottom Bar:** Shows the Python 3 idle environment, search bar, taskbar icons, and system status.

# PROJECT TASK

WEEK 3

# DATA MODELING (TASK 1)

- Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.

```
[52]: # We can drop this two features since they are not correlated with target variable(Outcome)
h_data.drop(['BloodPressure', 'DiabetesPedigreeFunction'], axis=1, inplace=True)

[53]: h_data.head()

[54]: Pregnancies Glucose SkinThickness Insulin BMI Age Outcome
0 0 6 148 35 0 33.6 50 1
1 1 85 29 0 20.0 31 0
2 8 183 0 0 33.3 32 1
3 1 89 23 94 28.1 21 0
4 0 137 35 168 43.1 33 1

[55]: # Project Task: Week 3

[56]: # Strategies for model building-
# The Target variable(dependent variable) that is 'Outcome' is binary that is it has only two values 0 or 1
# As dependent variable(target variable) is binary therefore Logistic regression is the appropriate regression analysis

[57]: # Importing required class
from sklearn.linear_model import LogisticRegression

[58]: # Split the dataset in features and target variable
x = h_data.drop(["Outcome"], axis=1)
y = h_data["Outcome"]

[59]: # Importing required libraries to split x and y into training and testing set.
from sklearn.model_selection import train_test_split

[60]: # We are splitting data in 75% of the data for model training and 25% for model testing
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=0)

[61]: # Creating the model
logreg = LogisticRegression()

# fitting the model with train data
logreg.fit(x_train, y_train)

# fitting the model with predictor test data(x_test)
y_pred = logreg.predict(x_test)

C:\ProgramData\Anaconda\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
FutureWarning)

[62]: # Evaluating the model using confusion matrix

[63]: # Reporting the metrics for creating confusion matrix
from sklearn import metrics

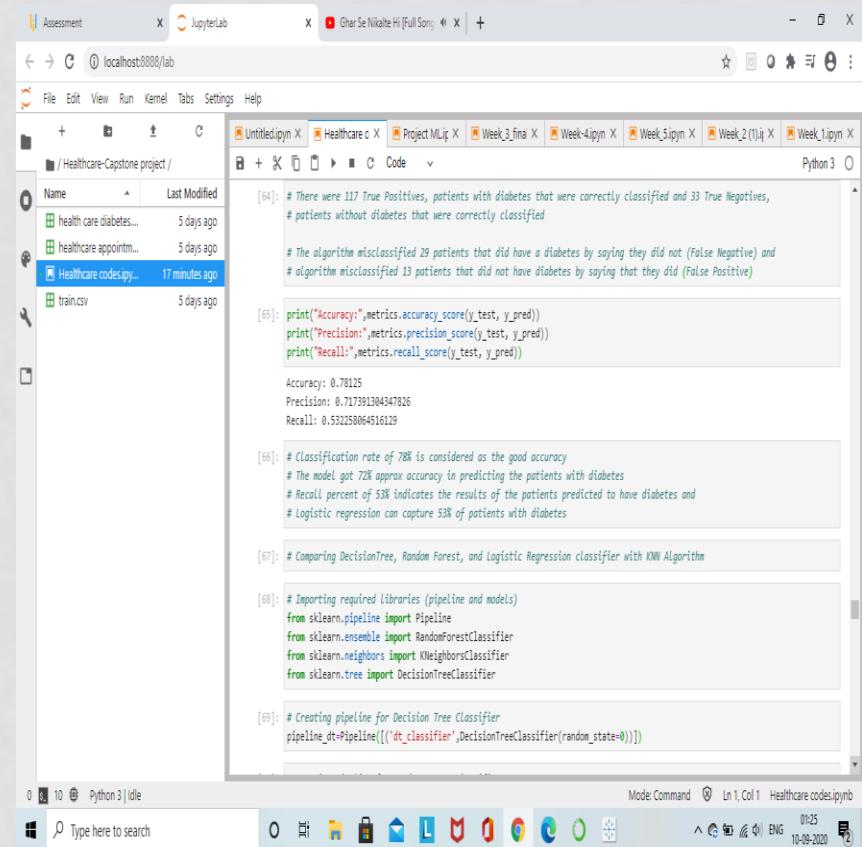
[64]: cf_matrix = metrics.confusion_matrix(y_test, y_pred)
cf_matrix

[65]: array([[117, 13],
       [29, 33]], dtype=int64)

[66]: # There were 117 True Positives, patients with diabetes that were correctly classified and 33 True Negatives.
```

# TASK 1

- Strategies for model building-
- The Target variable(dependent variable) that is 'Outcome' is binary that is it has only two values 0 or 1
- As dependent variable(Target variable) is binary therefore Logistic regression is the appropriate regression analysis



The screenshot shows a Jupyter Notebook interface with several tabs at the top: Assessment, JupyterLab, and Ghar Se Nikalte Hain [Full Song]. The main area displays a Python script for a healthcare project. The code includes comments explaining classification metrics and a pipeline creation section. The output pane shows the results of the code execution.

```
# There were 117 True Positives, patients with diabetes that were correctly classified and 33 True Negatives, # patients without diabetes that were correctly classified
# The algorithm misclassified 29 patients that did have a diabetes by saying they did not (False Negative) and # algorithm misclassified 13 patients that did not have diabetes by saying that they did (False Positive)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))

Accuracy: 0.78125
Precision: 0.71791304347826
Recall: 0.532258064516129

# Classification rate of 78% is considered as the good accuracy
# The model got 72% approx accuracy in predicting the patients with diabetes
# Recall percent of 53% indicates the results of the patients predicted to have diabetes and
# Logistic regression can capture 53% of patients with diabetes

# Comparing DecisionTree, Random Forest, and Logistic Regression classifier with KNN Algorithm

# Importing required Libraries (pipeline and models)
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

# Creating pipeline for Decision Tree Classifier
pipeline_dt=Pipeline([('dt_classifier',DecisionTreeClassifier(random_state=0))])
```

# TASK 2

- Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.

```

[64]: # There were 117 True Positives, patients with diabetes that were correctly classified and 33 True Negatives, patients without diabetes that were correctly classified
# The algorithm misclassified 29 patients that did have a diabetes by saying they did not (False Negative) and 53 patients that did not have diabetes by saying that they did (False Positive)

[65]: print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
print("Precision:",metrics.precision_score(y_test, y_pred))
print("Recall:",metrics.recall_score(y_test, y_pred))

Accuracy: 0.78125
Precision: 0.7179139434726
Recall: 0.53250844616129

[66]: # Classification rate of 78% is considered as the good accuracy
# The model got 78% accuracy in predicting the patients with diabetes
# 53% percent of 53K indicates the majority of the patients predicted to have diabetes and logistic regression can capture 53% of patients with diabetes

[67]: # Comparing DecisionTree, Random Forest, and Logistic Regression classifier with KNN Algorithm

[68]: # Importing required libraries (pipeline and models)
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier

[69]: # Creating pipeline for Decision Tree Classifier
pipeline_dt=Pipeline([('dt_classifier',DecisionTreeClassifier(random_state=0))])

```

```

[70]: # Creating pipeline for Random Forest Classifier
pipeline_rf=Pipeline([('rf_classifier',RandomForestClassifier())])

[71]: # Creating Pipeline for Kneighbors Classifier
pipeline_knn=Pipeline([('kn_classifier',KNeighborsClassifier())])

[72]: # Creating Pipeline for Logistic Regression
pipeline_lr=Pipeline([('lr_classifier',LogisticRegression())])

[73]: # Making the list of all pipelines
pipelines = [pipeline_dt,pipeline_rf,pipeline_knn,pipeline_lr]

[74]: best_accuracy=0.0
best_classifier=0
best_pipeline=""

[75]: # Dictionary of pipelines and classifier type for ease of reference
pipe_dict = {0: 'Decision Tree', 1: 'RandomForest', 2: 'KNeighbors', 3:'Logistic Regression'}

# Fitting the pipelines
for pipe in pipelines:
    pipe.fit(x_train, y_train)

C:\ProgramData\Anaconda3\lib\site-packages\sklearn\ensemble\forest.py:245: FutureWarning: The default value of n_estimators will change from 10 in version 0.20 to 100 in 0.22.
    "10 in version 0.20, 100 in 0.22.", FutureWarning)
C:\ProgramData\Anaconda3\lib\site-packages\sklearn\linear_model\logistic.py:432: FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a solver to silence this warning.
    FutureWarning)

```

```

[76]: for i,model in enumerate(pipelines):
    print(f"\n{i}: Test Accuracy: {format(pipe_dict[i],model.score(x_test,y_test))}")
Decision Tree Test Accuracy: 0.7042366666666666
RandomForest Test Accuracy: 0.7684166666666666
KNeighbors Test Accuracy: 0.7708333333333334
Logistic Regression Test Accuracy: 0.78125

[77]: for i,model in enumerate(pipelines):
    if model.score(x_test,y_test)>best_accuracy:
        best_accuracy=model.score(x_test,y_test)
        best_pipeline=model
        best_classifier=i
        print(f"Classifier with best accuracy(): {format(pipe_dict[best_classifier])}")
Classifier with best accuracy:Logistic Regression

[78]: # Logistic Regression is the best classifier compared with other algorithms, with accuracy of 78%, which is good

[79]: # Creating Confusion Matrix for each Algorithm to compare each classifier

[80]: # Decision Tree Confusion matrix
x_pred = pipeline_dt.predict(x_test)
dt_cf_matrix=metrics.confusion_matrix(y_test,y_pred)
dt_cf_matrix

[80]: array([[100, 25],
       [21, 41]], dtype=int64)

[81]: # Random Forest Confusion matrix
y_pred2 = pipeline_rf.predict(x_test)

```

# TASK 2

- Decision Tree correctly classified 105 patients with diabetes and 41 patients without diabetes
- Random Forest correctly classified 115 patients with diabetes and 31 patients without diabetes
- KNN correctly classified 111 patients with diabetes and 37 patients without diabetes
- Logistic Regression correctly classified 117 patients with diabetes and 33 patients without diabetes

The screenshot shows a Jupyter Notebook interface with several code cells and files listed in the sidebar.

**Code Cells:**

- [80]: # Decision Tree Confusion matrix  
y\_pred1 = pipeline\_dt.predict(x\_test)  
dt\_cf\_matrix=metrics.confusion\_matrix(y\_test,y\_pred1)  
dt\_cf\_matrix
- [80]: array([[105, 25],  
 [21, 41]], dtype=int64)
- [81]: # Random Forest Confusion matrix  
y\_pred2 = pipeline\_rf.predict(x\_test)  
rf\_cf\_matrix=metrics.confusion\_matrix(y\_test,y\_pred2)  
rf\_cf\_matrix
- [81]: array([[115, 15],  
 [31, 31]], dtype=int64)
- [83]: # KNeighbours Confusion matrix  
y\_pred3= pipeline\_knn.predict(x\_test)  
knn\_cf\_matrix=metrics.confusion\_matrix(y\_test,y\_pred3)  
knn\_cf\_matrix
- [83]: array([[111, 19],  
 [25, 37]], dtype=int64)
- [84]: # Logistic Regression Confusion matrix  
y\_pred4=pipeline\_lr.predict(x\_test)  
lr\_cf\_matrix=metrics.confusion\_matrix(y\_test,y\_pred4)  
lr\_cf\_matrix
- [84]: array([[117, 13],  
 [29, 33]], dtype=int64)

**File List:**

- Untitled.ipynb
- Healthcare codes.ipynb
- Project ML.ipynb
- Week\_3\_final.ipynb
- Week\_4.ipynb
- Week\_5.ipynb
- Week\_2.ipynb
- Week\_1.ipynb

# TASK 2

- KNN and Logistic Regression are better performers as compared to Decision Tree and Random Forest

The screenshot shows a Jupyter Notebook interface with multiple tabs open. The current tab is titled 'Healthcare codes.ipynb'. The code cell contains the following Python script:

```
# Logistic Regression Confusion matrix
y_pred4= pipeline_lr.predict(x_test)
lr_cf_matrix=metrics.confusion_matrix(y_test,y_pred4)
lr_cf_matrix

[84]: array([[117,  13],
           [ 29,  33]], dtype=int64)

[85]: # Decision Tree correctly classified 105 patients with diabetes and 41 patients without diabetes
# Random Forest correctly classified 115 patients with diabetes and 31 patients without diabetes
# KNN correctly classified 111 patients with diabetes and 37 patients without diabetes
# Logistic Regression correctly classified 117 patients with diabetes and 33 patients without diabetes

[86]: # KNN and Logistic Regression are better performers as compared to Decision Tree and Random Forest

[87]: # Project Task: Week 4

[88]: # Creating a classification report by analyzing sensitivity, specificity, AUC (ROC curve)

[89]: # Sensitivity identifies what percentage of patients with diabetes were correctly identified.
# Specificity identifies what percentage of patients without diabetes were correctly identified.

[90]: # As Logistic Regression and KNN are better performers therefore we are analyzing sensitivity, specificity, AUC (ROC curve)
# on these two classifiers

[91]: # For KNeighbors
sensitivity, specificity= [111/(111+25), 37/(37+19)]
print("sensitivity:", sensitivity)
print("specificity:", specificity)
```

# PROJECT TASK

WEEK 4

# DATA MODELING (TASK 1)

- Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve). Please be descriptive to explain values of the parameter you have used.

The screenshot shows a JupyterLab environment with multiple tabs open. The active tab is 'Healthcare codes.ipynb'. The code in the notebook is as follows:

```
[84]: # Logistic Regression Confusion matrix  
y_pred4= pipeline_lr1.predict(x_test)  
lr_cm4=metrics.confusion_matrix(y_test,y_pred4)  
lr_cm4  
  
[85]: array([[117, 13],  
       [29, 33]], dtype=int64)  
  
[86]: # Decision Tree correctly classified 105 patients with diabetes and 41 patients without diabetes  
# Random Forest correctly classified 105 patients with diabetes and 31 patients without diabetes  
# KNN correctly classified 101 patients with diabetes and 37 patients without diabetes  
# Logistic Regression correctly classified 117 patients with diabetes and 33 patients without diabetes  
  
[88]: # KNN and Logistic Regression are better performers as compared to Decision Tree and Random Forest  
  
[89]: # Project Task: Week 4  
  
[90]: # Creating a classification report by analyzing sensitivity, specificity, AUC (ROC curve)  
  
[91]: # Sensitivity identifies what percentage of patients with diabetes were correctly identified.  
# Specificity identifies what percentage of patients without diabetes were correctly identified.  
  
[92]: # As Logistic Regression and KNN are better performers therefore we are analyzing sensitivity, specificity, AUC (ROC curve)  
# on these two classifiers  
  
[93]: # For KNN algorithm  
sensitivity, specificity = [111/(111+25), 37/(37+19)]  
print("sensitivity-", sensitivity)  
print("specificity-", specificity)
```

Assessment JupyterLab Tk Mulaqat Zaroori Hai Sami +

File Edit View Run Kernel Tabs Settings Help

Healthcare-Capstone project /

Name Last Modified

- health care diabetes... 5 days ago
- healthcare appointment... 5 days ago
- Healthcare codes.ipynb 18 minutes ago
- train.csv 5 days ago

Untitled.ipynb | Healthcare c... | Project MLIG... | Week\_3\_final | Week-4.ipynb | Week\_5.ipynb | Week\_2 (1).ipynb | Week\_1.ipynb | Python 3

```
[91]: # For kNeighbors
sensitivity, specificity = [11/(11+25), 37/(37+19)]
print("sensitivity:", sensitivity)
print("specificity:", specificity)

sensitivity: 0.461674470588255
specificity: 0.48071420571424897

[92]: # For Logistic Regression
sensitivity, specificity = [18/(19+29), 33/(33+13)]
print("sensitivity:", sensitivity)
print("specificity:", specificity)

sensitivity: 0.8016889830130886
specificity: 0.717391304347826

[93]: # kNeighbours has sensitivity approx equivalent to Logistic Regression
# Logistic Regression has better specificity as compared to kNeighbours
# Therefore I prefer Logistic regression over kNeighbours as it has both sensitivity and specificity better than other one

[94]: # AUC (ROC curve) for logistic Regression
```

```
[95]: # Importing required Libraries
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
```

```
[96]: # We use roc_curve() to get the threshold, TPR and FPR
fpr, tpr, thresholds = roc_curve(y_test, pipeline_lr.predict_proba(x_test)[:,1])
```

```
[97]: fpr
```

```
[98]: tpr
```

```
[99]: roc_auc_score(tpr, fpr)
```

Mode: Command L, C1 Col 1 Healthcare codes.ipynb

Type here to search

Windows Taskbar icons: File Explorer, Edge, Mail, Photos, OneDrive, Task View, Start, Task Manager, Control Panel, Settings, Power, Network, System, Help & Support.

# TASK 1

- As Logistic Regression and KNN are better performers therefore we are analyzing sensitivity, specificity, AUC (ROC curve) on these two classifiers.

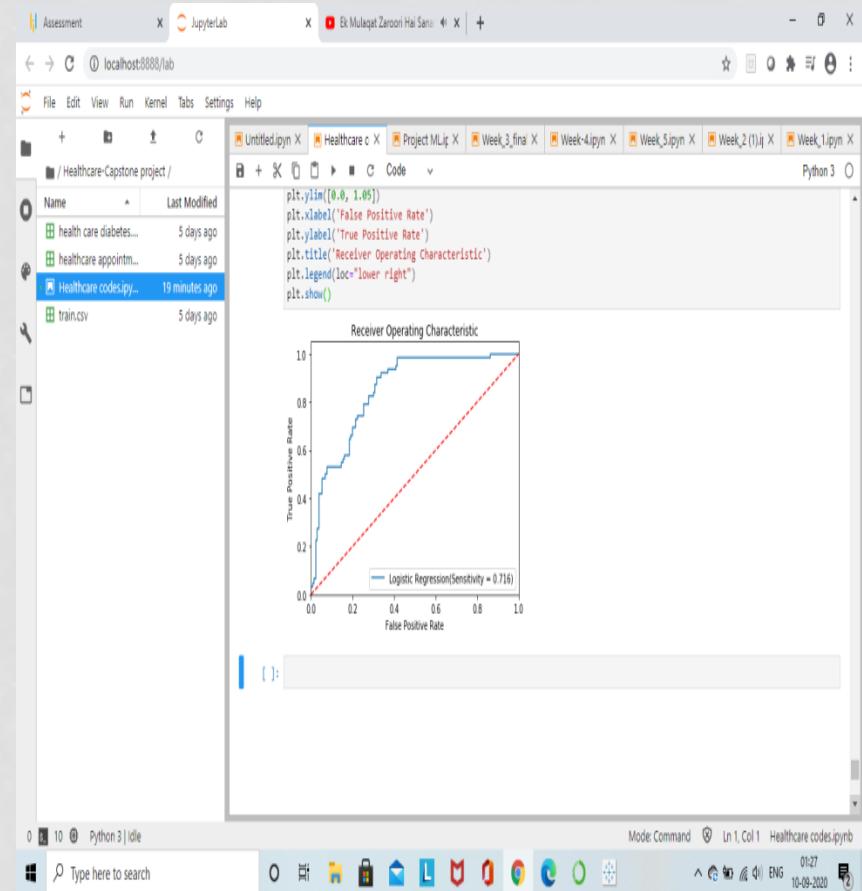
The screenshot shows a Jupyter Notebook interface with several tabs at the top: Assessment, JupyterLab, and a video player showing 'Ek Mulaqat Zaroori Hai Sanchit'. The main area displays a code cell with the following content:

```
[100]: thresholds  
[100]: thresholds  
[100]: array([1.8487957 , 0.8487957 , 0.82151504, 0.81274305, 0.88870539,  
[100]: 0.79198927, 0.73169448, 0.78249244, 0.72259105, 0.71332306,  
[100]: 0.68877731, 0.68112674, 0.61625498, 0.60006157, 0.57045116,  
[100]: 0.55306454, 0.54100147, 0.53972548, 0.53087008, 0.41600897,  
[100]: 0.42808979, 0.42632315, 0.42409418, 0.41759778, 0.40930264,  
[100]: 0.39583261, 0.38580697, 0.38228719, 0.38063133, 0.37801889,  
[100]: 0.37470043, 0.36936023, 0.36702678, 0.35913308, 0.35811472,  
[100]: 0.34813872, 0.34265791, 0.32990758, 0.32528292, 0.31992497,  
[100]: 0.3179205 , 0.31564505, 0.30676044, 0.30584095, 0.30992447,  
[100]: 0.27934072, 0.27180857, 0.27530081 , 0.27471287, 0.26072979,  
[100]: 0.25835169, 0.25095367, 0.24846087, 0.12778914, 0.12627306,  
[100]: 0.0138982 ])  
  
[101]: # For AUC we use roc_auc_score() function for ROC  
lr_roc_auc = roc_auc_score(y_test, pipeline_lr.predict(x_test))  
  
[102]: #Plot the ROC Curve  
plt.figure()  
plt.plot(fpr, tpr, label='Logistic Regression[Sensitivity = %0.3f]' % lr_roc_auc)  
plt.plot([0,1], [0,1], 'r--')  
plt.xlim([0.0, 1.0])  
plt.ylim([0.0, 1.05])  
plt.xlabel('False Positive Rate')  
plt.ylabel('True Positive Rate')  
plt.title('Receiver Operating Characteristic')  
plt.legend(loc="lower right")  
plt.show()
```

The bottom status bar shows: Mode: Command Ln 1, Col 1 Healthcare codes.ipynb, Type here to search, and a taskbar with various icons.

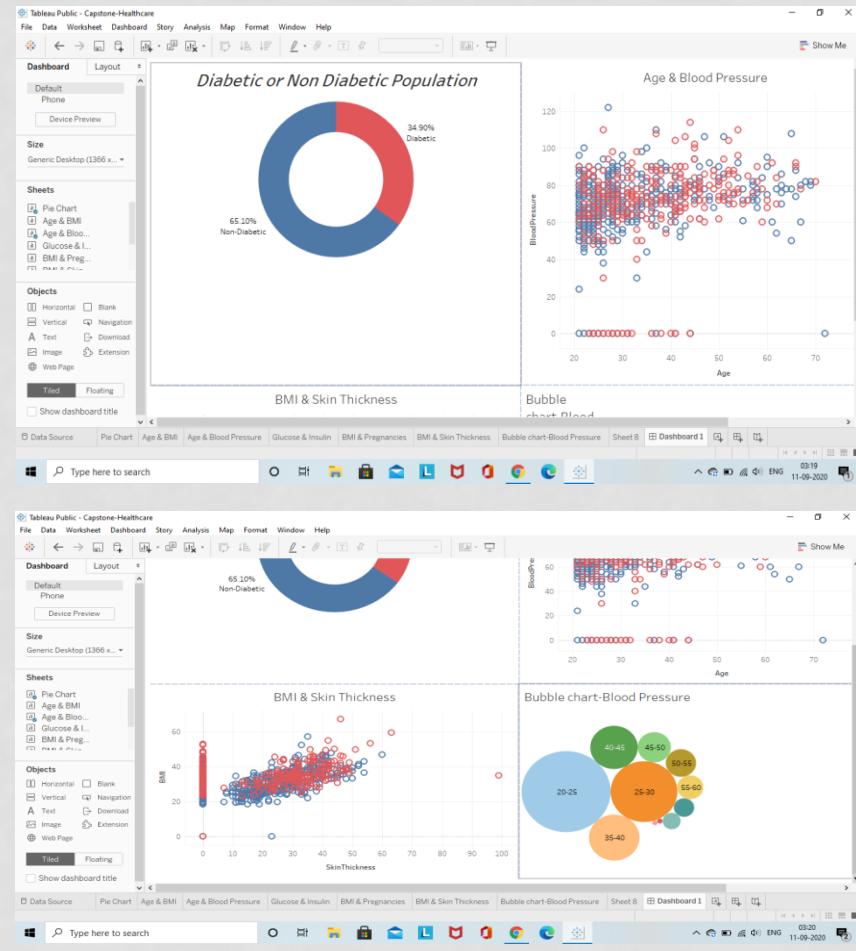
# TASK 1

- KNeighbours has sensitivity approx equivalent to Logistic Regression
- Logistic Regression has better specificity as compared to KNeighbours
- Therefore i prefer Logistic regression over KNeighbours as it has both sensitivity and specificity better than other one



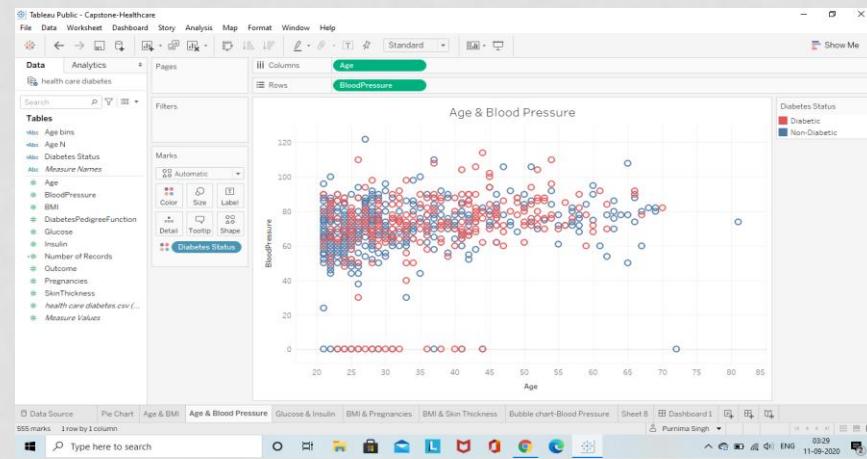
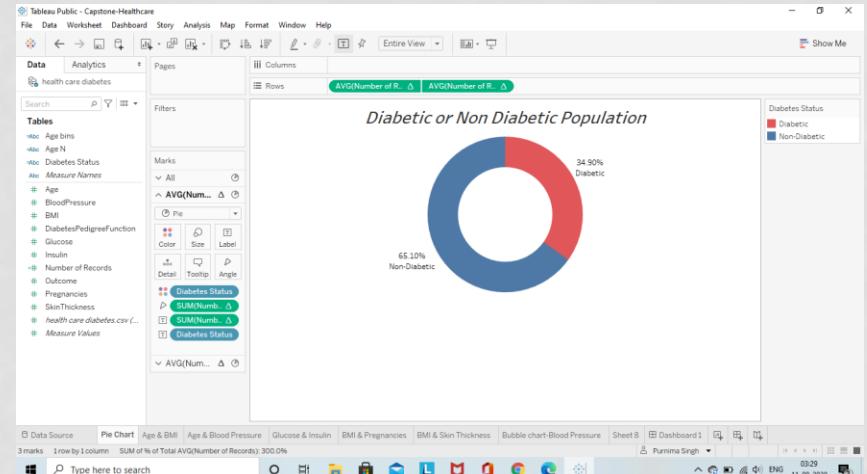
# DATA REPORTING (TASK 2)

- Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:
- Pie chart to describe the diabetic or non-diabetic population.
- Scatter charts between relevant variables to analyze the relationships



# TASK 2

- Histogram or frequency charts to analyze the distribution of the data.
- Heatmap of correlation analysis among the relevant variables.
- Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables for these age brackets using a bubble chart.



# TASK 2

