REPORT
CAPSTONE PROJECT
HEALTHCARE

SUBMITTED BY:

HARSH RAJ SINGH

Problem Statement

NIDDK (National Institute of Diabetes and Digestive and Kidney Diseases) research creates knowledge about and treatments for the most chronic, costly, and consequential diseases.

Objective

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.

Week-1: Data Exploration

```
In [3]: df.shape
Out[3]: (768, 9)

In [4]: df.columns
Out[4]: Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
               'BMI', 'DiabetesPedigreeFunction', 'Age', 'Outcome'],
              dtype='object')

In [5]: df.dtypes
Out[5]: Pregnancies                   int64
        Glucose                       int64
        BloodPressure                 int64
        SkinThickness                 int64
        Insulin                       int64
        BMI                         float64
        DiabetesPedigreeFunction    float64
        Age                           int64
        Outcome                       int64
        dtype: object
```

There are 8 columns out of which there are 6 integer type columns and 2 float type

```
In [6]: df.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
```

There are total 768 entries in the dataframe and there are no missing values

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

Out[8]:

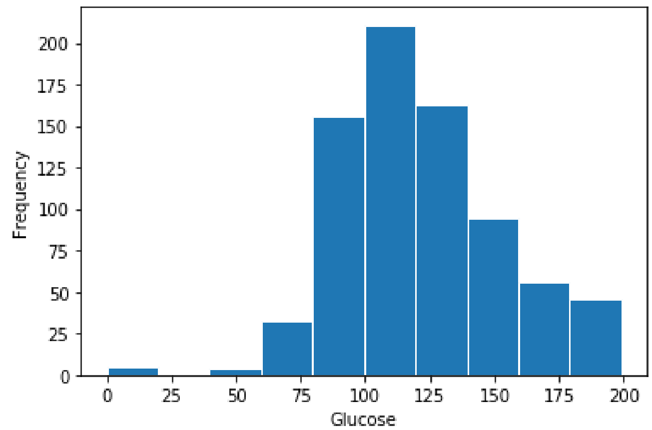
	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	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
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	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

```
In [9]: #checking for null values
        df.isnull().any()

Out[9]: Pregnancies                 False
        Glucose                     False
        BloodPressure               False
        SkinThickness               False
        Insulin                     False
        BMI                         False
        DiabetesPedigreeFunction    False
        Age                         False
        Outcome                     False
        dtype: bool
```

The statistical summary for each column is presented above. And again it is checked that there are no missing values in the dataframe.

Checking data distribution

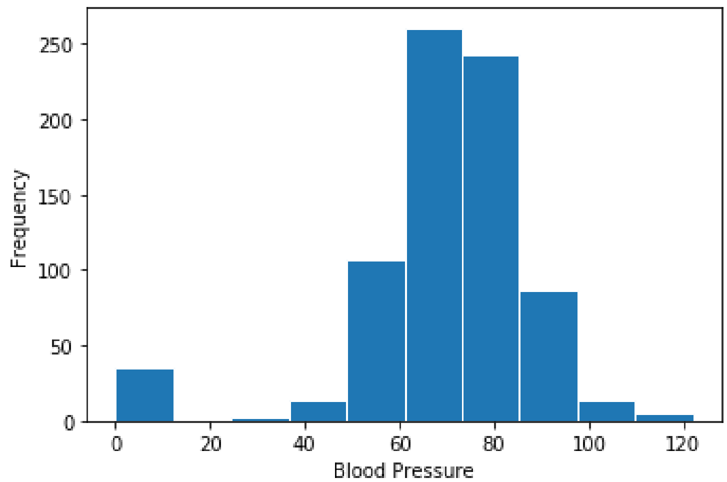
```
In [11]: #Checking the data distribution of Glucose
         plt.hist(df['Glucose'],ec='white')
         plt.xlabel("Glucose")
         plt.ylabel("Frequency")
         plt.show()
```




```
In [12]: df['Glucose'].value_counts()

Out[12]: 100    17
         99     17
         129    14
         125    14
         111    14
                ..
         177     1
         172     1
         169     1
         160     1
         199     1
         Name: Glucose, Length: 136, dtype: int64
```

The value of Glucose levels is mostly concentrated between 75 and 150, with a few value counts lying outside this range and may possibly be outliers. The value counts of glucose levels is also shown in the table.

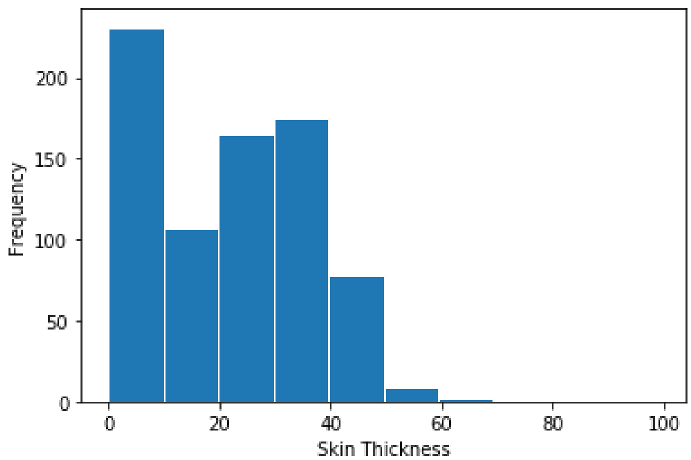
```
In [13]: #Checking the data distribution of Blood Pressure
         plt.hist(df['BloodPressure'],ec='white')
         plt.xlabel("Blood Pressure")
         plt.ylabel("Frequency")
         plt.show()
```




```
In [14]: df['BloodPressure'].value_counts().head(10)

Out[14]: 70    57
         74    52
         68    45
         78    45
         72    44
         64    43
         80    40
         76    39
         60    37
         0     35
         Name: BloodPressure, dtype: int64
```

The blood pressure values are mostly concentrated around 50 and 100 mm Hg, and there are a few blood pressure values that are around zero and appears to be faulty or erroneous. The count of each blood pressure value is shown in the picture above.

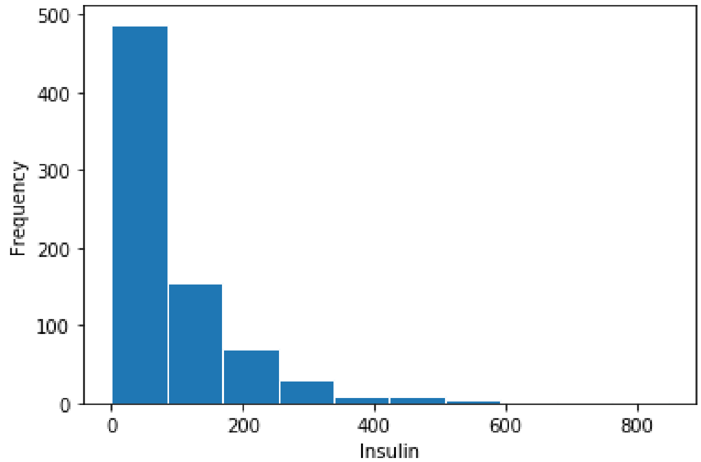
```
In [15]: #Checking the data distribution of Skin Thickness
plt.hist(df['SkinThickness'],ec='white')
plt.xlabel("Skin Thickness")
plt.ylabel("Frequency")
plt.show()
```




```
In [16]: df['SkinThickness'].value_counts().head(10)

Out[16]: 0     227
32     31
30     27
27     23
23     22
33     20
18     20
28     20
31     19
39     18
Name: SkinThickness, dtype: int64
```

The skin thickness is mostly concentrated around 10 and 50 mm. There are a large number of values around zero which appear to be faulty and misguiding. From the value counts it is visible that 227 values of skin thickness are 0 which is not possible in real life and is wrongly recorded.

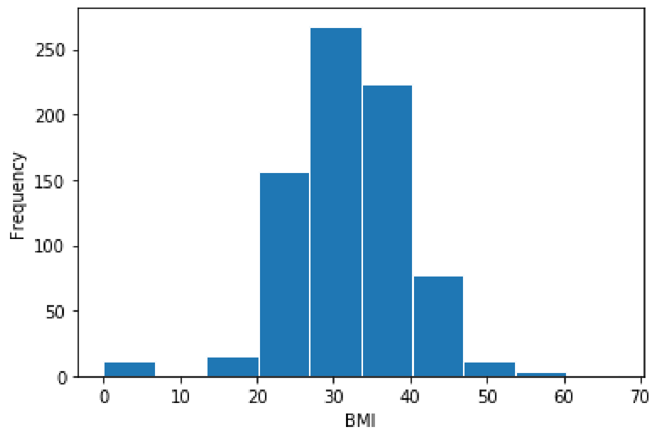
```
In [17]: #Checking the data distribution of Insulin
         plt.hist(df['Insulin'],ec='white')
         plt.xlabel("Insulin")
         plt.ylabel("Frequency")
         plt.show()
```




```
In [18]: df['Insulin'].value_counts().head(10)

Out[18]: 0      374
         105     11
         140      9
         130      9
         120      8
         100      7
         94       7
         180      7
         110      6
         115      6
         Name: Insulin, dtype: int64
```

The insulin level is mostly between 100 and 300. There are a large number of zero insulin values(374) which are probably misguiding values and should be removed.

```
In [19]: #Checking the data distribution of BMI
plt.hist(df['BMI'],ec='white')
plt.xlabel("BMI")
plt.ylabel("Frequency")
plt.show()
```




```
In [20]: df['BMI'].value_counts().head(10)

Out[20]: 32.0    13
31.6    12
31.2    12
0.0     11
33.3    10
32.4    10
32.8     9
30.8     9
32.9     9
30.1     9
Name: BMI, dtype: int64
```

The body mass index values are almost normally distributed with values mostly between 20 and 45, with a few outliers. The value counts table is depicted above. The zero BMI does not signify anything and possibly denotes error in recording data.

Data Cleaning

```
In [21]: #Determining the number of zero values in these columns
         print("Glucose",df[df['Glucose']==0].Glucose.value_counts().values)
         print("Blood Pressure",df[df['BloodPressure']==0].BloodPressure.value_counts().values)
         print("Skin Thickness",df[df['SkinThickness']==0].SkinThickness.value_counts().values)
         print("Insulin",df[df['Insulin']==0].Insulin.value_counts().values)
         print("BMI",df[df['BMI']==0].BMI.value_counts().values)

         Glucose [5]
         Blood Pressure [35]
         Skin Thickness [227]
         Insulin [374]
         BMI [11]
```

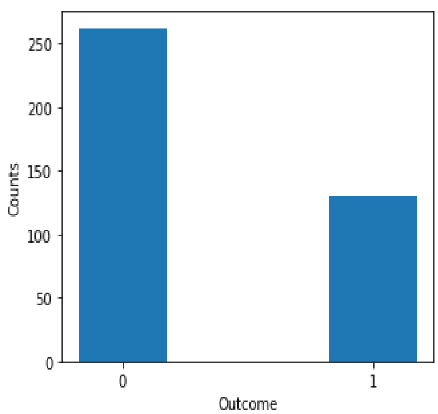
The number of zero values in Glucose, Blood Pressure, Skin Thickness, Insulin and BMI values are depicted above. These values are non-realistic and insignificant, and should therefore be removed for correct analysis and data modelling.

```
In [22]: #Dropping the zero blood pressure values
         df=df.drop(df[df['BloodPressure']==0].index)
         df.shape

Out[22]: (733, 9)

In [23]: #Dropping the zero glucose values
         df=df.drop(df[df['Glucose']==0].index)
         df.shape

Out[23]: (728, 9)

In [24]: #Dropping the zero skin thickness values
         df=df.drop(df[df['SkinThickness']==0].index)
         df.shape

Out[24]: (534, 9)

In [25]: #Dropping the zero insulin values
         df=df.drop(df[df['Insulin']==0].index)
         df.shape

Out[25]: (393, 9)

In [26]: #Dropping the zero BMI values
         df=df.drop(df[df['BMI']==0].index)
         df.shape

Out[26]: (392, 9)
```

Week-2: Data Exploration

Checking the data balance

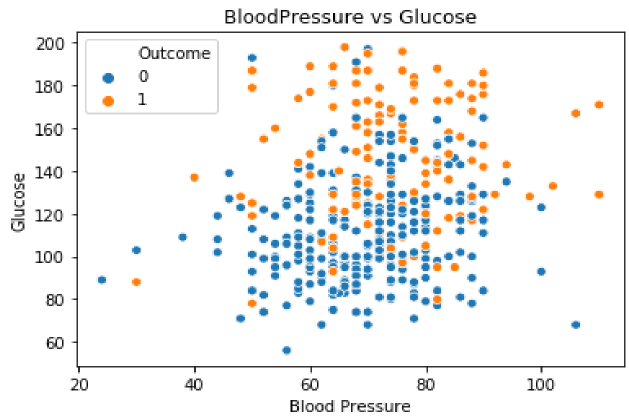
```
In [27]: #checking data balance
         df['Outcome'].value_counts()

Out[27]: 0    262
         1    130
         Name: Outcome, dtype: int64

In [28]: #plotting data outcome counts
         plt.figure(figsize=(5,4))
         A=plt.bar(df['Outcome'].value_counts().index,height=df['Outcome'].value_counts().values,width=0.35,align='center')
         plt.xticks(df['Outcome'].value_counts().index)
         plt.xlabel('Outcome')
         plt.ylabel('Counts')
         plt.show()
```



From the above analysis it can be seen that from the dataset, 262 of the total patients analyzed do not have diabetes and rest do not have diabetes. The data is slightly biased towards negative possibility of having diabetes, but is not highly biased and is good enough for performing the analysis and building a machine learning model.

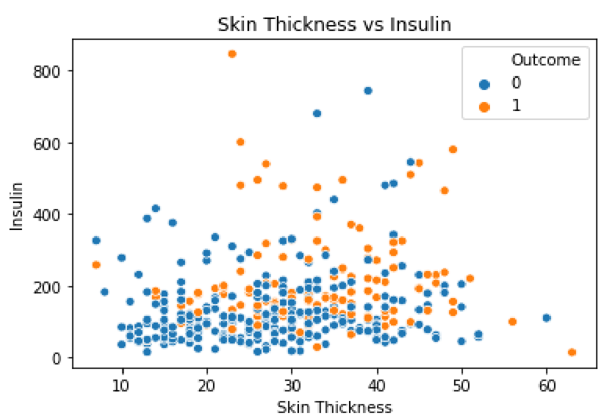
Analyzing relation between variables by scatter chart

```
In [29]: #scatter plots
         plt.figure(figsize=(6,4))
         sns.scatterplot(x= "BloodPressure" ,y= "Glucose",hue="Outcome",data=df)
         plt.xlabel('Blood Pressure')
         plt.ylabel('Glucose')
         plt.title('BloodPressure vs Glucose')
         plt.legend(loc='best')
         plt.show()
```



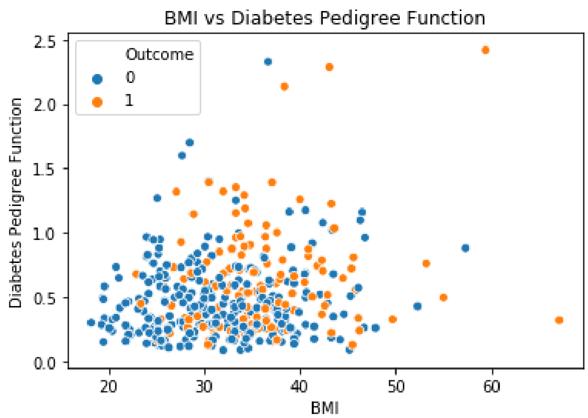
The datapoints appear to be sparsely distributed with a few outliers. The points are almost distinguishable and are separated out for patients with/without diabetes. High glucose level patients seem to have higher chance of getting diabetes than with lower levels. But patients with both lower and higher blood pressures seem to be affected by diabetes.

```
In [30]: plt.figure(figsize=(6,4))
         sns.scatterplot(x= "SkinThickness" ,y= "Insulin",hue="Outcome",data=df)
         plt.xlabel('Skin Thickness')
         plt.ylabel('Insulin')
         plt.title('Skin Thickness vs Insulin')
         plt.legend(loc='best')
         plt.show()
```



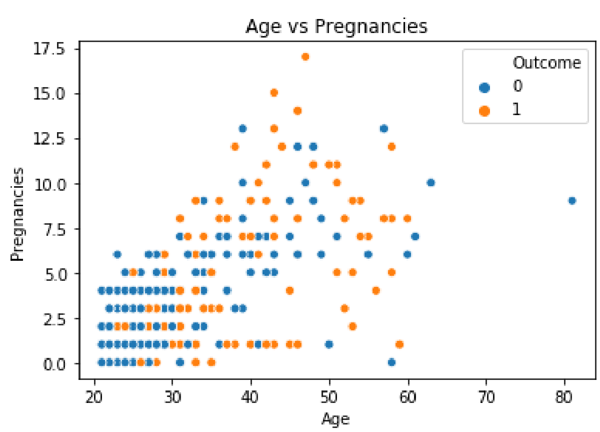
The datapoints appear to be concentrated and mixed for both labels with a few outliers, and are thus not clearly distinguishable. For higher insulin levels, the outcome is mostly positive for a patient having diabetes.

In [31]:
```
plt.figure(figsize=(6,4))
sns.scatterplot(x= "BMI" ,y= "DiabetesPedigreeFunction",hue="Outcome",data=df)
plt.xlabel('BMI')
plt.ylabel('Diabetes Pedigree Function')
plt.title('BMI vs Diabetes Pedigree Function')
plt.legend(loc='best')
plt.show()
```



From the scatter plot, it can be seen that patients with higher BMI and higher Diabetes Pedigree Function are most likely to have diabetes as compared to the lower levels. The data points for the two possible outcome labels are well separated and can be distinguished.

In [32]:
```
plt.figure(figsize=(6,4))
sns.scatterplot(x= "Age" ,y= "Pregnancies",hue="Outcome",data=df)
plt.xlabel('Age')
plt.ylabel('Pregnancies')
plt.title('Age vs Pregnancies')
plt.legend(loc='best')
plt.show()
```



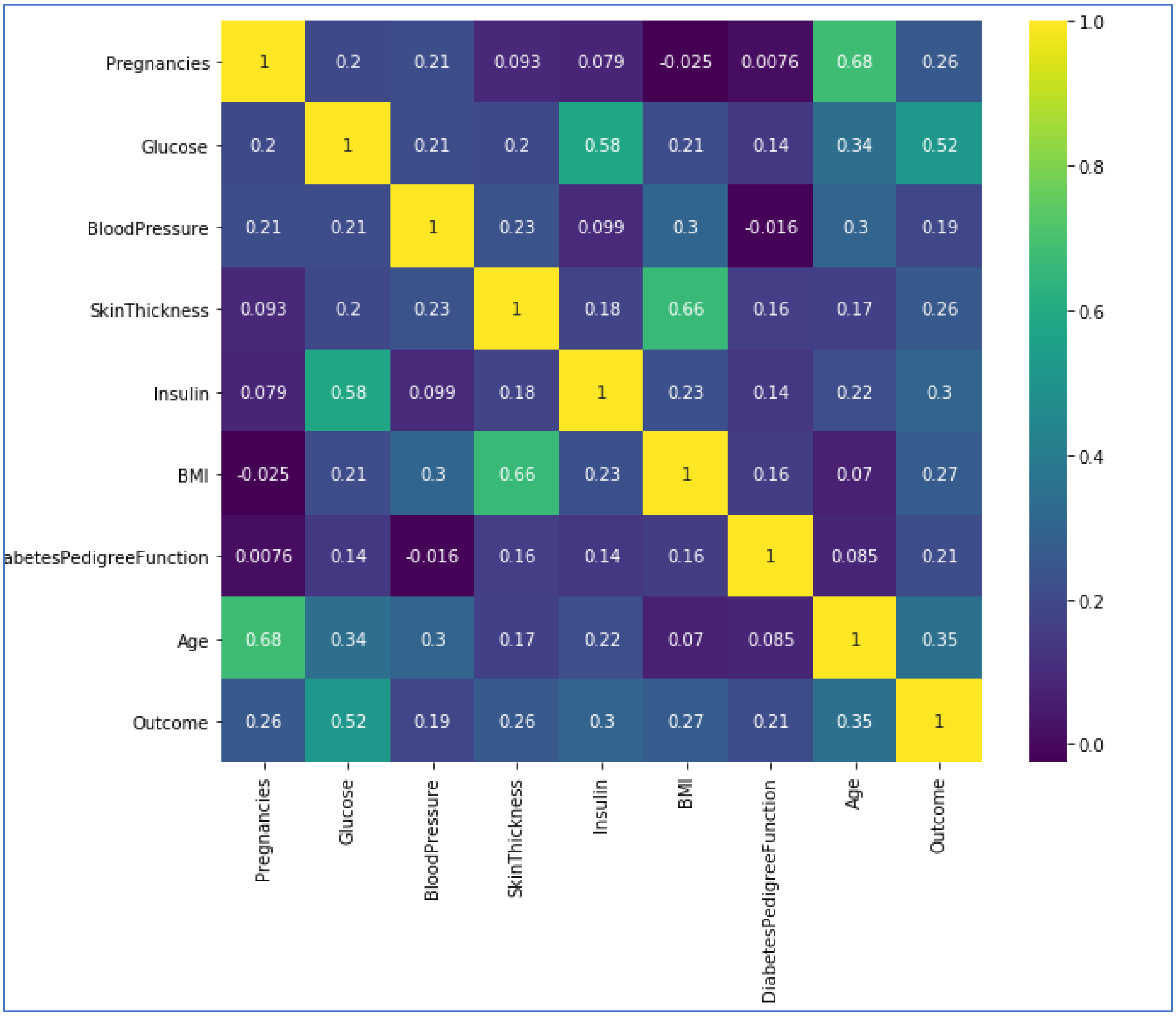
Patients with higher age and higher number of pregnancies are most likely to be diagnosed with diabetes as compared to the lower numbers,

Correlation Analysis

In [33]:
```
#correlation matrix
df.corr()
```

Out[33]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
Pregnancies	1.000000	0.198291	0.213355	0.093209	0.078984	-0.025347	0.007562	0.679608	0.256566
Glucose	0.198291	1.000000	0.210027	0.198856	0.581223	0.209516	0.140180	0.343641	0.515703
BloodPressure	0.213355	0.210027	1.000000	0.232571	0.098512	0.304403	-0.015971	0.300039	0.192673
SkinThickness	0.093209	0.198856	0.232571	1.000000	0.182199	0.664355	0.160499	0.167761	0.255936
Insulin	0.078984	0.581223	0.098512	0.182199	1.000000	0.226397	0.135906	0.217082	0.301429
BMI	-0.025347	0.209516	0.304403	0.664355	0.226397	1.000000	0.158771	0.069814	0.270118
DiabetesPedigreeFunction	0.007562	0.140180	-0.015971	0.160499	0.135906	0.158771	1.000000	0.085029	0.209330
Age	0.679608	0.343641	0.300039	0.167761	0.217082	0.069814	0.085029	1.000000	0.350804
Outcome	0.256566	0.515703	0.192673	0.255936	0.301429	0.270118	0.209330	0.350804	1.000000



It can be seen that glucose, age and insulin are most strongly correlated with outcome. Skin thickness and BMI are somewhat moderately correlated with the outcome, and the skin thickness, diabetes pedigree function and pregnancies are somewhat lesser correlated with the outcome. In terms of correlation among features, insulin is strongly correlated with glucose, BMI is strongly correlated with skin thickness and pregnancies is strongly correlated with age.

Week-3: Data Modelling

```
In [35]: #Creating feature matrix and outcome vectos
         X=df.iloc[:,[0,1,2,3,4,5,6,7]].values
         Y=df['Outcome']
         print(X.shape,Y.shape)

         (392, 8) (392,)

In [36]: #Splitting the data into training and testing data sets
         from sklearn.model_selection import train_test_split
         x_tr,x_ts,y_tr,y_ts=train_test_split(X,Y,test_size=0.2,random_state=50)

In [37]: print(x_tr.shape,x_ts.shape,y_tr.shape,y_ts.shape)

         (313, 8) (79, 8) (313,) (79,)
```

All the features, i.e., Glucose, Blood Pressure, Skin Thickness, BMI, Insulin, Age and Pregnancies are considered to be a part of the feature matrix X for modelling. The outcome is the target vector Y. The training and test dataset are split in the ratio of 80:20. 80% of the data is used for training the model and 20% is used for testing. The shape of the two datasets after train test split is shown in the figure.

The performance of 5 classifiers, namely Logistic Regression, Decision Tree, Random Forest, SVM and kNN is analyzed for accurate prediction of diabetes and the comparison report is later presented.

Logistic Regression

```
In [38]: #Logistic Regression
         from sklearn.linear_model import LogisticRegression
         log_reg=LogisticRegression(max_iter=1000)
         log_reg.fit(x_tr,y_tr)
         y_pr1=log_reg.predict(x_ts)

In [39]: #importing evaluation matrices
         from sklearn.metrics import accuracy_score
         from sklearn.metrics import classification_report
         from sklearn.metrics import confusion_matrix
         from sklearn.metrics import roc_auc_score

In [40]: print(confusion_matrix(y_ts,y_pr1))

         [[50  6]
          [ 7 16]]

In [41]: print("Accuracy Score:%.2f"%(accuracy_score(y_ts,y_pr1)*100),"%")
         print("AUC:%.2f"%(roc_auc_score(y_ts, y_pr1)))

         Accuracy Score:83.54 %
         AUC:0.79

In [42]: print(classification_report(y_pr1,y_ts))

                       precision    recall  f1-score   support

                    0       0.89      0.88      0.88        57
                    1       0.70      0.73      0.71        22

             accuracy                           0.84        79
            macro avg       0.79      0.80      0.80        79
         weighted avg       0.84      0.84      0.84        79
```

Applying Logistic Regression classifier, the accuracy score of 83.54% is obtained. The confusion matrix seems to be balanced. The AUC(ROC) score of 0.79 is obtained. The other parameters obtained on the classification report can be seen above.

Decision Tree

```
In [43]: #Decision Tree
         from sklearn.tree import DecisionTreeClassifier
         dec_tr = DecisionTreeClassifier(max_depth=4)
         dec_tr.fit(x_tr,y_tr)
         y_pr2=dec_tr.predict(x_ts)

In [44]: print(confusion_matrix(y_ts,y_pr2))

         [[50  6]
          [ 8 15]]

In [45]: print("Accuracy Score:%.2f"%(accuracy_score(y_ts,y_pr2)*100),"%")
         print("AUC:%.2f"%(roc_auc_score(y_ts, y_pr2)))

         Accuracy Score:82.28 %
         AUC:0.77

In [46]: print(classification_report(y_pr2,y_ts))

                       precision    recall  f1-score   support

                    0       0.89      0.86      0.88        58
                    1       0.65      0.71      0.68        21

             accuracy                           0.82        79
            macro avg       0.77      0.79      0.78        79
         weighted avg       0.83      0.82      0.83        79
```

Applying Decision Tree classifier, the accuracy score of 82.28% is obtained. Max Depth of the tree was tuned and chosen to be 4. The confusion matrix seems to be balanced. The AUC(ROC) score of 0.77 is obtained. The other parameters obtained on the classification report can be seen above.

Random Forest

```
In [47]: #Random Forest
         from sklearn.ensemble import RandomForestClassifier
         rf=RandomForestClassifier(n_estimators=20)
         rf.fit(x_tr,y_tr)
         y_pr3=rf.predict(x_ts)

In [48]: print(confusion_matrix(y_ts,y_pr3))

         [[51  5]
          [ 9 14]]

In [49]: print("Accuracy Score:%.2f"%(accuracy_score(y_ts,y_pr3)*100),"%")
         print("AUC:%.2f"%(roc_auc_score(y_ts, y_pr3)))

         Accuracy Score:82.28 %
         AUC:0.76

In [50]: print(classification_report(y_pr3,y_ts))

                       precision    recall  f1-score   support

                    0       0.91      0.85      0.88        60
                    1       0.61      0.74      0.67        19

             accuracy                           0.82        79
            macro avg       0.76      0.79      0.77        79
         weighted avg       0.84      0.82      0.83        79
```

Applying Random Forest classifier, the accuracy score of 82.28% is obtained. Number of estimators for random forest were tuned and selected to be 20.The confusion matrix seems to be balanced. The AUC(ROC) score of 0.76 is obtained. The other parameters obtained on the classification report can be seen above.

Support Vector Machine

```
In [51]: #SVM
         from sklearn.svm import SVC
         svm=SVC(kernel='linear',gamma='auto')
         svm.fit(x_tr,y_tr)
         y_pr4=svm.predict(x_ts)

In [52]: print(confusion_matrix(y_ts,y_pr4))

         [[50  6]
          [ 7 16]]

In [53]: print("Accuracy Score:%.2f"%(accuracy_score(y_ts,y_pr4)*100),"%")
         print("AUC:%.2f"%roc_auc_score(y_ts, y_pr4))

         Accuracy Score:83.54 %
         AUC:0.79

In [54]: print(classification_report(y_pr4,y_ts))

                       precision    recall  f1-score   support

                    0       0.89      0.88      0.88        57
                    1       0.70      0.73      0.71        22

             accuracy                           0.84        79
            macro avg       0.79      0.80      0.80        79
         weighted avg       0.84      0.84      0.84        79
```

Applying SVM, the accuracy score of 83.54% is obtained. The kernel was chosen to be linear and gamma was chosen to be auto. The confusion matrix seems to be balanced. The AUC(ROC) score of 0.79 is obtained. The other parameters obtained on the classification report can be seen above.

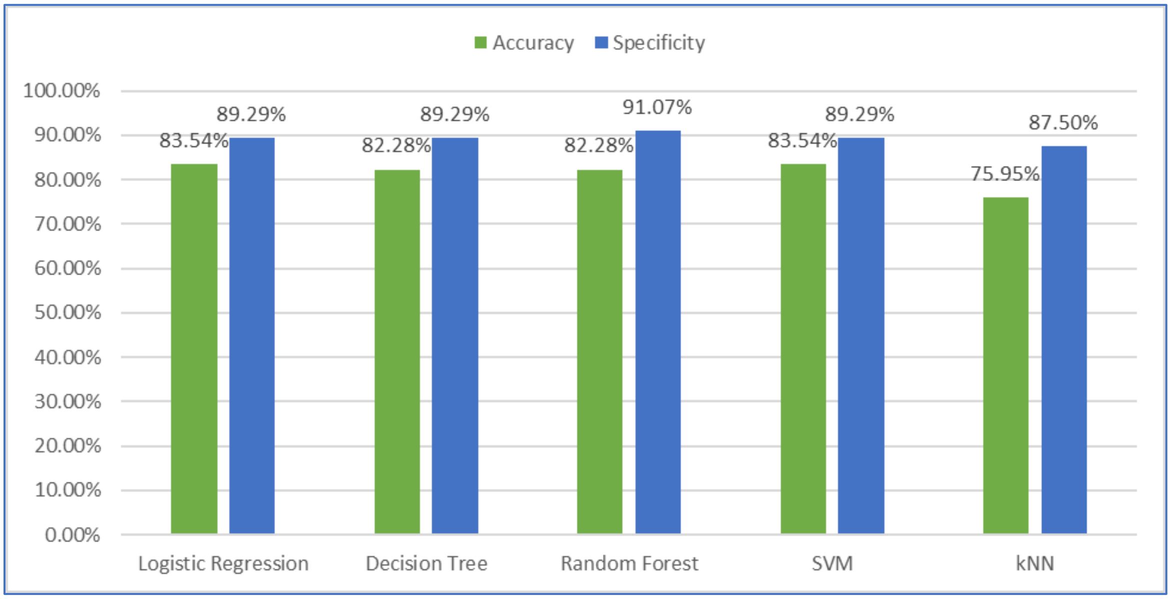
k-Nearest Neighbors

```
In [55]: #kNN
         from sklearn.neighbors import KNeighborsClassifier
         knn=KNeighborsClassifier(n_neighbors=12,p=1)
         knn.fit(x_tr,y_tr)
         y_pr5=knn.predict(x_ts)

In [56]: print(confusion_matrix(y_ts,y_pr5))

         [[49  7]
          [12 11]]

In [57]: print("Accuracy Score:%.2f"%(accuracy_score(y_ts,y_pr5)*100),"%")
         print("AUC:%.2f"%roc_auc_score(y_ts, y_pr5))

         Accuracy Score:75.95 %
         AUC:0.68

In [58]: print(classification_report(y_pr5,y_ts))

                       precision    recall  f1-score   support

                    0       0.88      0.80      0.84        61
                    1       0.48      0.61      0.54        18

             accuracy                           0.76        79
            macro avg       0.68      0.71      0.69        79
         weighted avg       0.78      0.76      0.77        79
```

Applying kNN classifier, the accuracy score of 75.95% is obtained. The number of neighbors was selected and tuned as 12. The confusion matrix seems to be balanced. The AUC(ROC) score of 0.68 is obtained. The other parameters obtained on the classification report can be seen above.

Week-4: Data Modelling

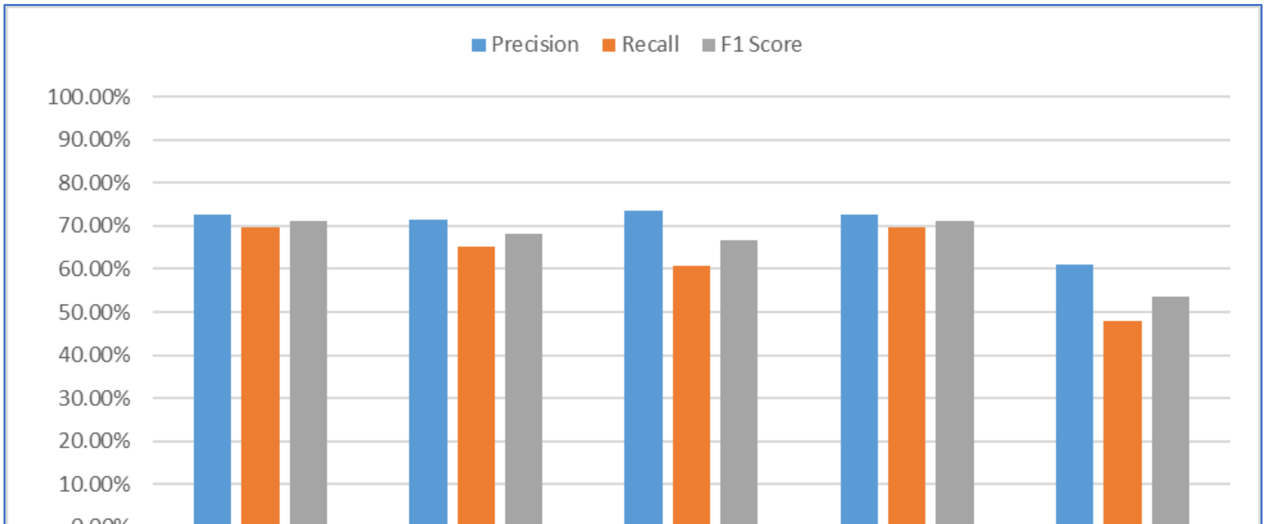
Modelling Performance Evaluation and Analysis summary



Comments:

For all the models the specificity was observed to be around 90%. Highest specificity was observed for Random Forest classifier(91%) and the lowest specificity was observed for kNN (87%)

All the classifiers exhibited prediction accuracy score of around 82-83% except kNN, for which accuracy score was the least and was around 76%.



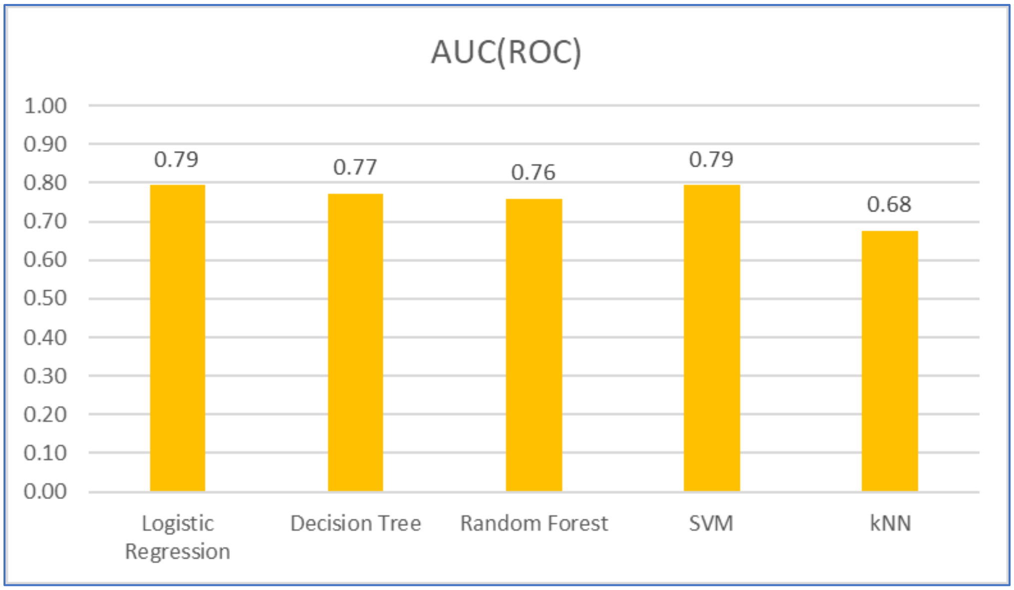
	Logistic Regression	Decision Tree	Random Forest	SVM	kNN
Precision	72.73%	71.43%	73.68%	72.73%	61.11%
Recall	69.57%	65.22%	60.87%	69.57%	47.83%
F1 Score	71.11%	68.18%	66.67%	71.11%	53.66%

Comments:

The precision scores for all the classifiers was observed to be around 71-74% except kNN. Highest precision was observed for Random Forest and lowest precision was observed for kNN.

The recall or sensitivity score was observed to be highest for Logistic Regression and SVM classifiers(70%) and least for kNN(48%).

Similar trend is observed in case of F1 score and F1 score was highest for SVM and Logistic Regression (71%) and lowest for kNN(54%)

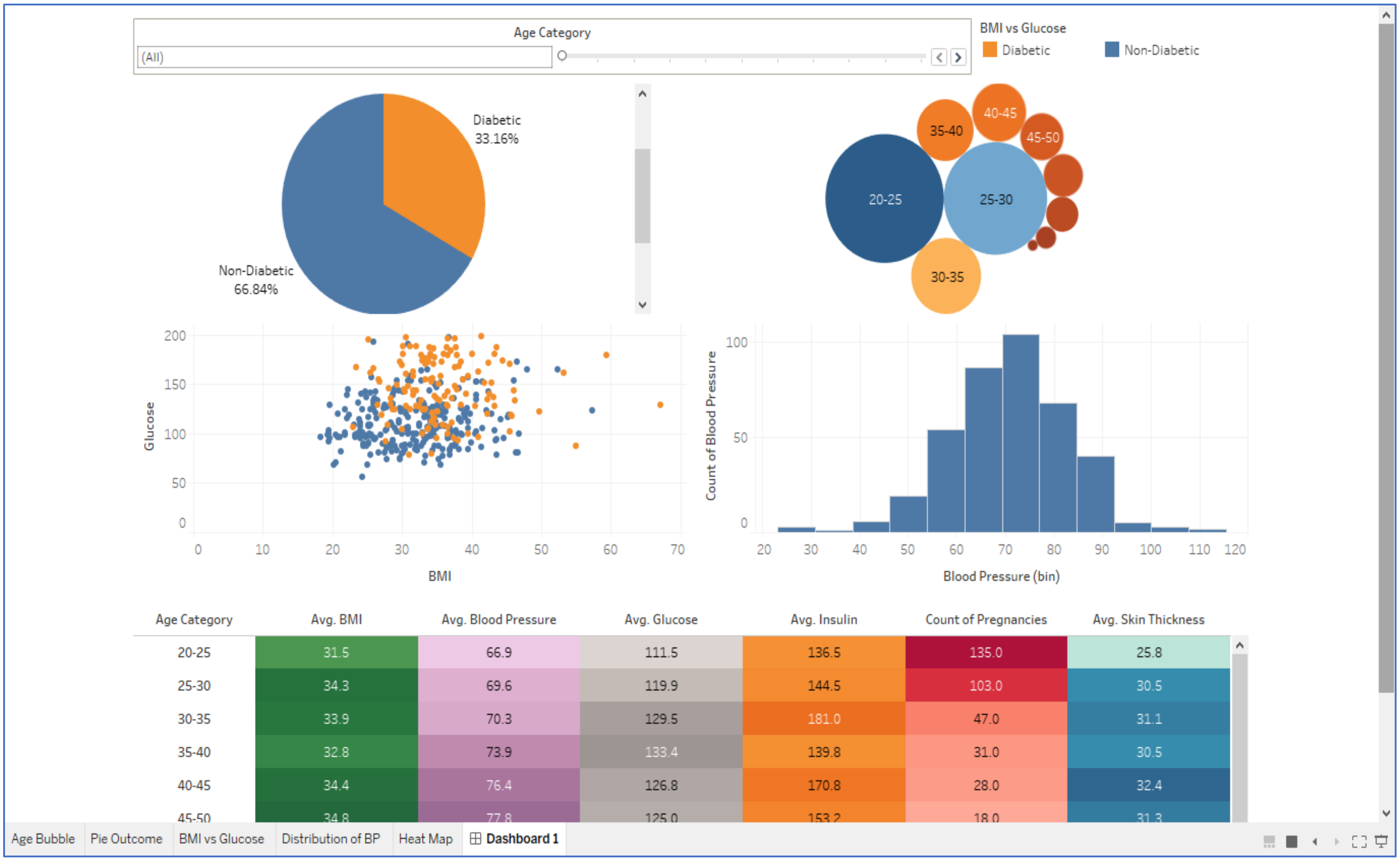


SVM and Logistic Regression classifiers exhibited highest AUC of around 0.79 and kNN exhibited least AUC of around 0.68.

Summary of best performers in terms of all performance evaluation parameters

Evaluation Parameter	**Best Performer**	**Value**
Accuracy Score	Logistic Regression/SVM	83.54%
Specificity	Random Forest	91.07%
Precision	Random Forest	73.68%
Recall	Logistic Regression/SVM	69.57%
F1 Score	Logistic Regression/SVM	71.11%
AUC (ROC)	Logistic Regression/SVM	0.79

Data Reporting in Tableau



Age Category	Avg. BMI	Avg. Blood Pressure	Avg. Glucose	Avg. Insulin	Count of Pregnancies	Avg. Skin Thickness
20-25	31.5	66.9	111.5	136.5	135.0	25.8
25-30	34.3	69.6	119.9	144.5	103.0	30.5
30-35	33.9	70.3	129.5	181.0	47.0	31.1
35-40	32.8	73.9	133.4	139.8	31.0	30.5
40-45	34.4	76.4	126.8	170.8	28.0	32.4
45-50	34.8	77.8	125.0	153.2	18.0	31.3

Conclusion

- Data Exploration tasks were performed and the distribution of data for variables was analyzed. Data cleaning was performed to prepare the data for modelling.
- The relationship between variables was studied by means of scatter plot and heat map was analyzed for correlation analysis.
- For prediction of diabetes, 5 models were built namely, Logistic Regression, Decision Tree, Random Forest, SVM and kNN, and their performances were analyzed in terms of performance evaluation parameters.
- From the performance of the classifiers, SVM/Logistic Regression classification technique is recommended for prediction of diabetes.
- kNN was the worst performer among all the classifiers analyzed and is not recommended for prediction of diabetes.
- Data reporting task was performed in Tableau and the image of the dashboard is attached in the document.