Untitled

July 13, 2021

1 Project 2 : Healthcare

- DESCRIPTION

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.
- 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.

Dataset Description

- The datasets consists of several medical predictor variables and one target variable (Outcome). Predictor variables includes the number of pregnancies the patient has had, their BMI, insulin level, age, and more.

|Variables||Description| |---------||-----------| |Pregnancies||Number of times pregnant **Numeric**| |Glucose||Plasma glucose concentration in an oral glucose tolerance test **Numeric**| |BloodPressure||Diastolic blood pressure (mm Hg) **Numeric**| |SkinThickness||Triceps skinfold thickness (mm) **Numeric**| |Insulin||Two hour serum insulin **Numeric**| |BMI||Body Mass Index **Numeric**| |DiabetesPedigreeFunction||Diabetes pedigree function **Numeric**| |Age||Age in years **Numeric**| |Outcome||Class variable (either 0 or 1). 268 of 768 values are 1, and the others are 0|

1.0.1 Week 1 Task

- 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: 1)Glucose 2)BloodPressure 3)SkinThickness 4)Insulin 5)BMI
- Visually explore these variables using histograms. Treat the missing values accordingly.
- 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.

[1]:
```
# Importing necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
healthcare_data = pd.read_csv("health care diabetes.csv")
healthcare_data.head()
```

[2]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35        0  33.6
1            1       85             66             29        0  26.6
2            8      183             64              0        0  23.3
3            1       89             66             23       94  28.1
4            0      137             40             35      168  43.1

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
```

[3]:
```
healthcare_data.count() # Total 768 rows are present in the dataset.
healthcare_data.shape   # (768, 9) shape of the data.
```

[3]:
```
(768, 9)
```

[4]:
```
healthcare_data.describe() # description of the all features in dataset.
```

[4]:
```
       Pregnancies     Glucose  BloodPressure  SkinThickness     Insulin  \
count   768.000000  768.000000     768.000000     768.000000  768.000000
mean      3.845052  120.894531      69.105469      20.536458   79.799479
std       3.369578   31.972618      19.355807      15.952218  115.244002
min       0.000000    0.000000       0.000000       0.000000    0.000000
25%       1.000000   99.000000      62.000000       0.000000    0.000000
50%       3.000000  117.000000      72.000000      23.000000   30.500000
75%       6.000000  140.250000      80.000000      32.000000  127.250000
max      17.000000  199.000000     122.000000      99.000000  846.000000

              BMI  DiabetesPedigreeFunction         Age     Outcome
count  768.000000                768.000000  768.000000  768.000000
mean    31.992578                  0.471876   33.240885    0.348958
std      7.884160                  0.331329   11.760232    0.476951
min      0.000000                  0.078000   21.000000    0.000000
25%     27.300000                  0.243750   24.000000    0.000000
50%     32.000000                  0.372500   29.000000    0.000000
```

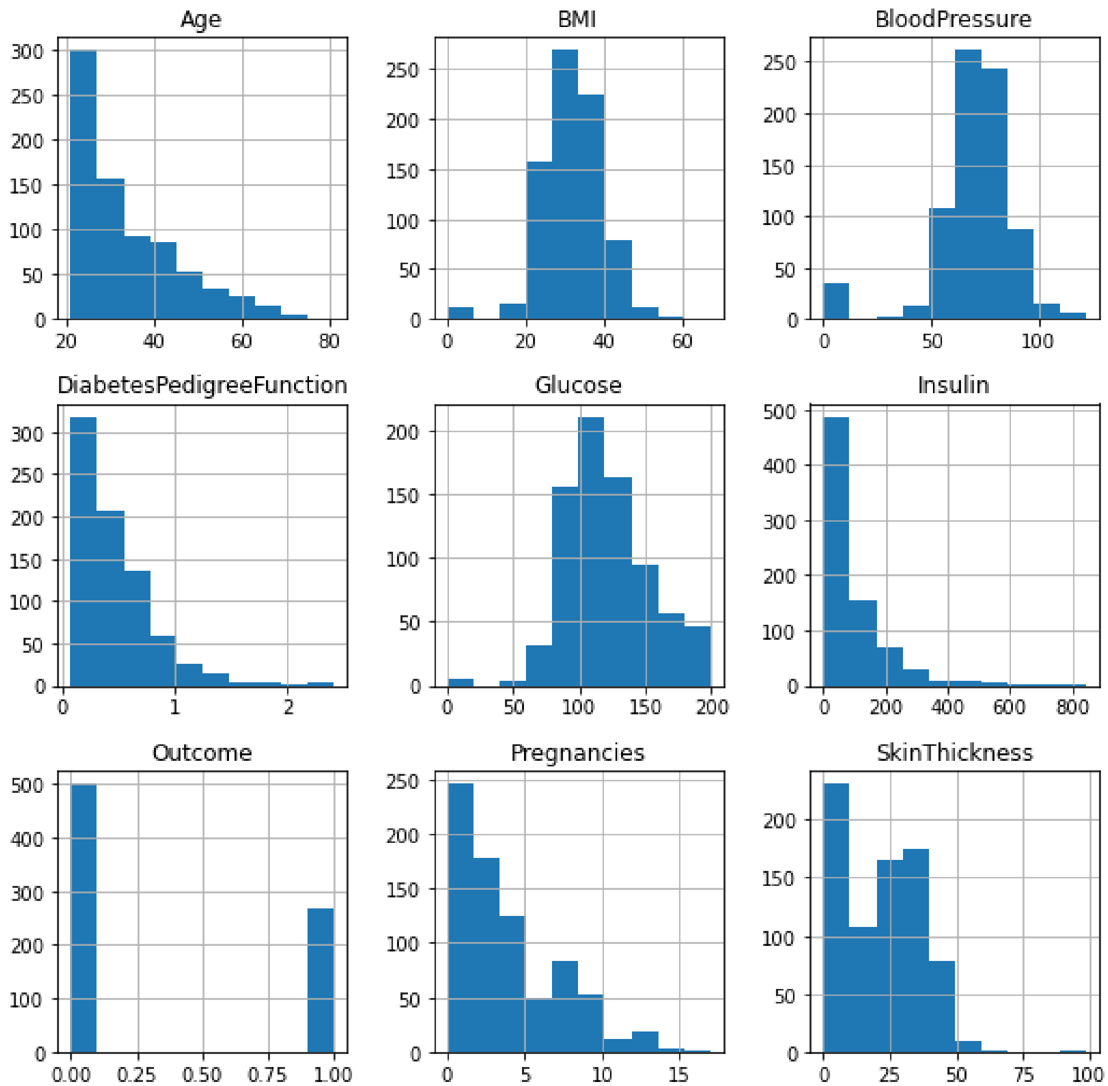
```
75%    36.600000                    0.626250   41.000000    1.000000
max    67.100000                    2.420000   81.000000    1.000000
```

[5]:
```
healthcare_data.dtypes
```

[5]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

[6]:
```
healthcare_data.hist(figsize=(10,10))
```

[6]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000002D8F867F6A0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002D8F8D5AB50>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002D8F8D89FA0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000002D8F8DC3460>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002D8F8DEE8B0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002D8F8E19C40>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000002D8F8E19D30>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002D8F8E53220>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000002D8F8EADA30>]],
      dtype=object)
```



[7]:
```
healthcare_data.isnull().sum()
```

[7]:
```
Pregnancies                 0
Glucose                     0
BloodPressure               0
SkinThickness               0
Insulin                     0
BMI                         0
DiabetesPedigreeFunction    0
Age                         0
Outcome                     0
dtype: int64
```

Note: 0 value present in Gluscose,bloodpressure,skinthickness,insulin,BMI doesn't make sense. We will replace this 0 with median of the particular column.

[8]:
```
# Missing values treatment
healthcare_data['Glucose']=healthcare_data['Glucose'].
 ↪replace(0,healthcare_data['Glucose'].median())
healthcare_data['BloodPressure']=healthcare_data['BloodPressure'].
 ↪replace(0,healthcare_data['BloodPressure'].median())
healthcare_data['SkinThickness']=healthcare_data['SkinThickness'].
 ↪replace(0,healthcare_data['SkinThickness'].median())
healthcare_data['Insulin']=healthcare_data['Insulin'].
 ↪replace(0,healthcare_data['Insulin'].median())
healthcare_data['BMI']=healthcare_data['BMI'].replace(0,healthcare_data['BMI'].
 ↪median())
```

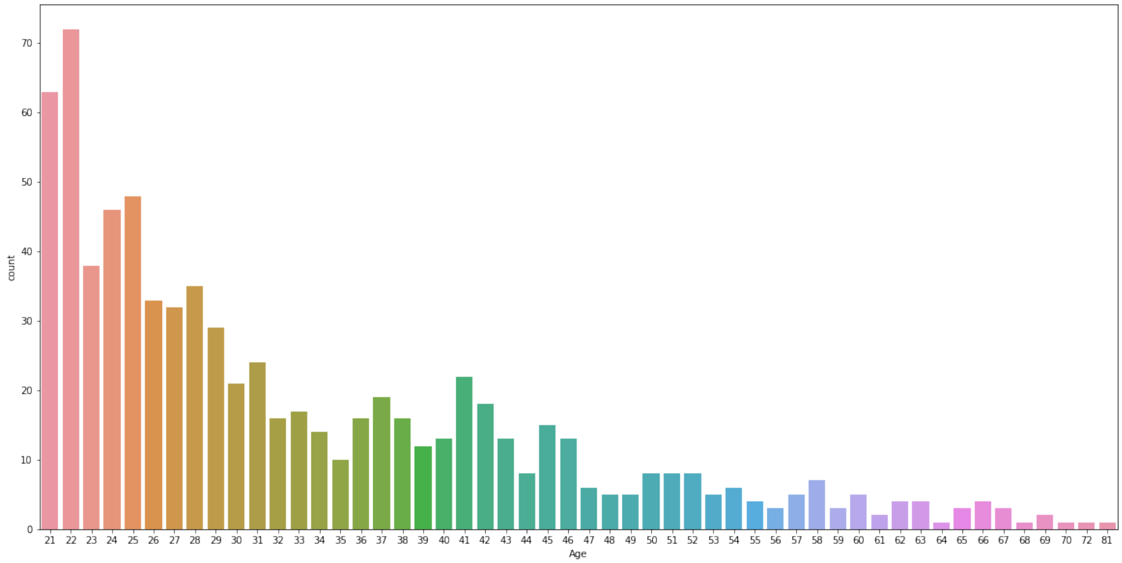
[9]:
```
healthcare_data.head(10) # missing value treatment done successfully.
```

[9]:
```
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  \
0            6      148             72             35     30.5  33.6
1            1       85             66             29     30.5  26.6
2            8      183             64             23     30.5  23.3
3            1       89             66             23     94.0  28.1
4            0      137             40             35    168.0  43.1
5            5      116             74             23     30.5  25.6
6            3       78             50             32     88.0  31.0
7           10      115             72             23     30.5  35.3
8            2      197             70             45    543.0  30.5
9            8      125             96             23     30.5  32.0

   DiabetesPedigreeFunction  Age  Outcome
0                     0.627   50        1
1                     0.351   31        0
2                     0.672   32        1
3                     0.167   21        0
4                     2.288   33        1
5                     0.201   30        0
6                     0.248   26        1
7                     0.134   29        0
8                     0.158   53        1
9                     0.232   54        1
```

[10]:
```
#Create a count (frequency) plot describing the data types and the count of␣
 ↪variables.
plt.figure(figsize=(20,10))
sns.countplot(x='Age',data=healthcare_data)
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2d8f9087eb0>
```



1.0.2 Project Task: Week 2

Data Exploration:

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.
2. Create scatter charts between the pair of variables to understand the relationships. Describe your findings.
3. Perform correlation analysis. Visually explore it using a heat map.

[11]:
```
healthcare_data.dtypes
```

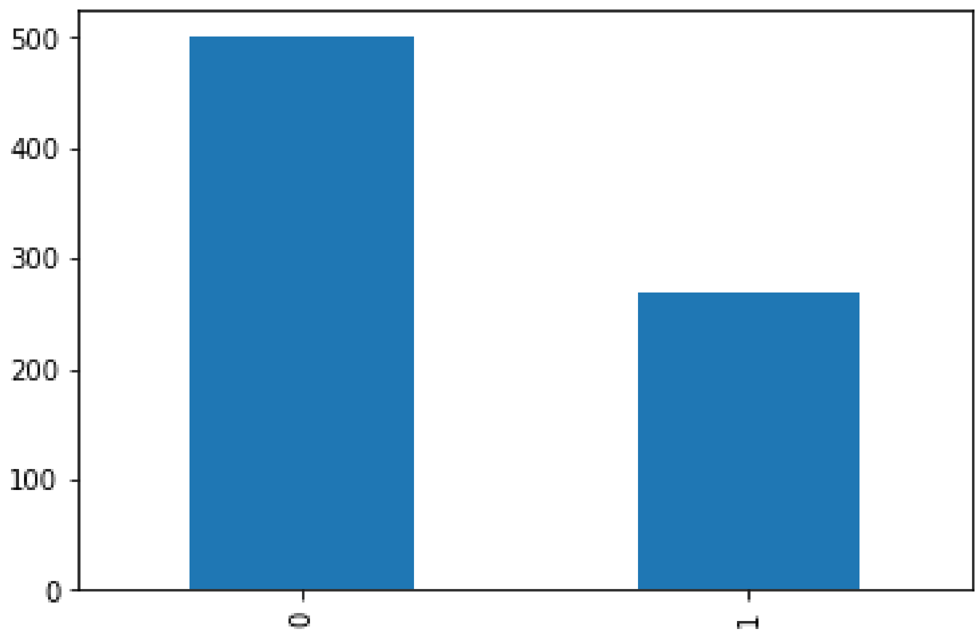
[11]:
```
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                     float64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
```

observation :Here the **Outcome** is numeric we will convert this into categorical.

[12]:
```
healthcare_data['Outcome']=pd.Categorical(healthcare_data['Outcome'])
```

[13]:
```
# Check the balance of the data by plotting the count of outcomes by their␣
↪value. Describe your findings
print(healthcare_data['Outcome'].value_counts())
healthcare_data['Outcome'].value_counts().plot.bar();
```

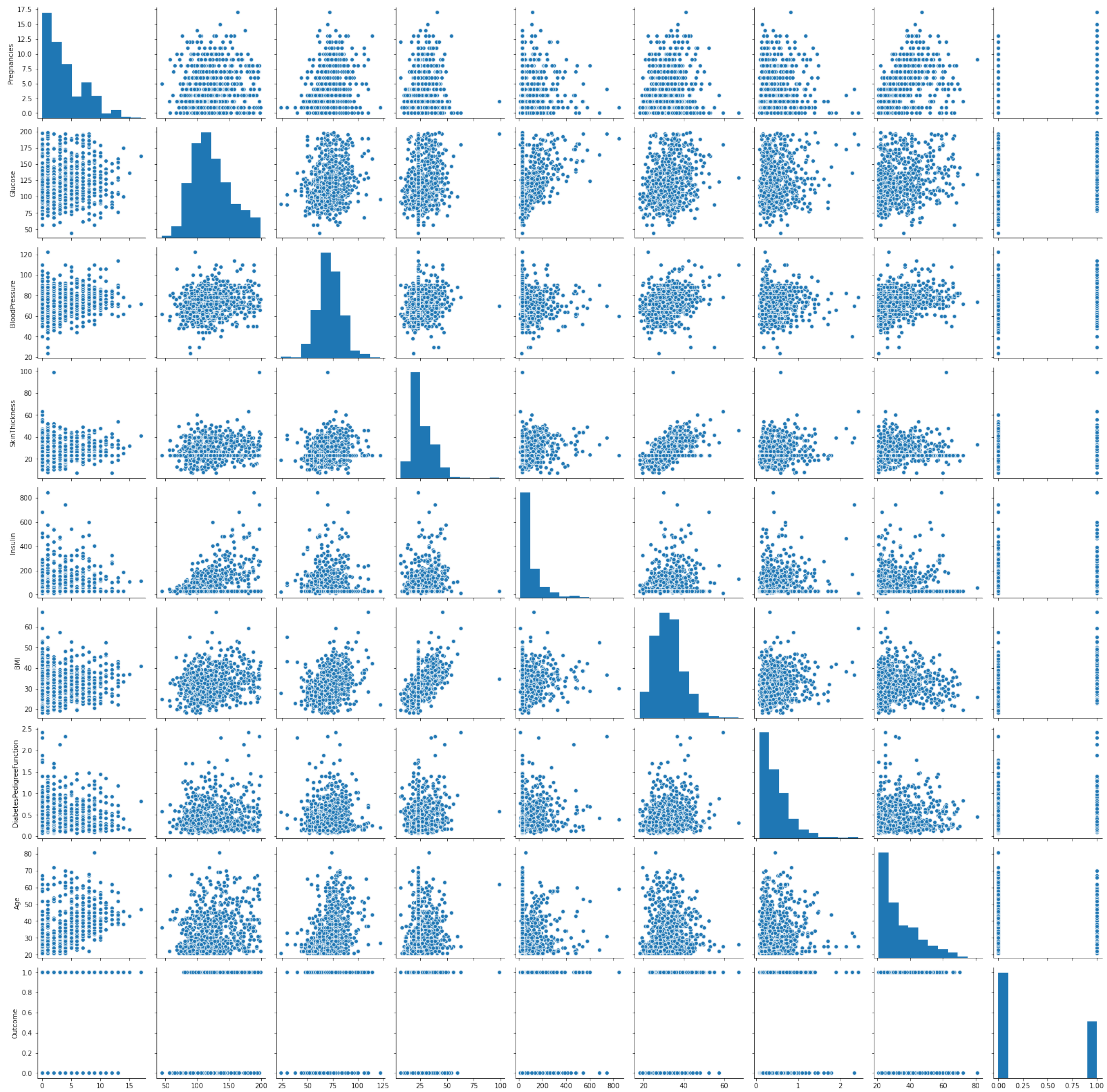
```
0    500
1    268
Name: Outcome, dtype: int64
```



Observation : The data is balanced with 500 zeros and 268 ones.

[14]:
```
# Create scatter charts between the pair of variables to understand the␣
↪relationships. Describe your findings.
sns.pairplot(healthcare_data,palette='hue')
```

[14]: <seaborn.axisgrid.PairGrid at 0x2d8f9625f40>



[15]:
```
#Perform correlation analysis. Visually explore it using a heat map.
healthcare_data.corr()
```

[15]:

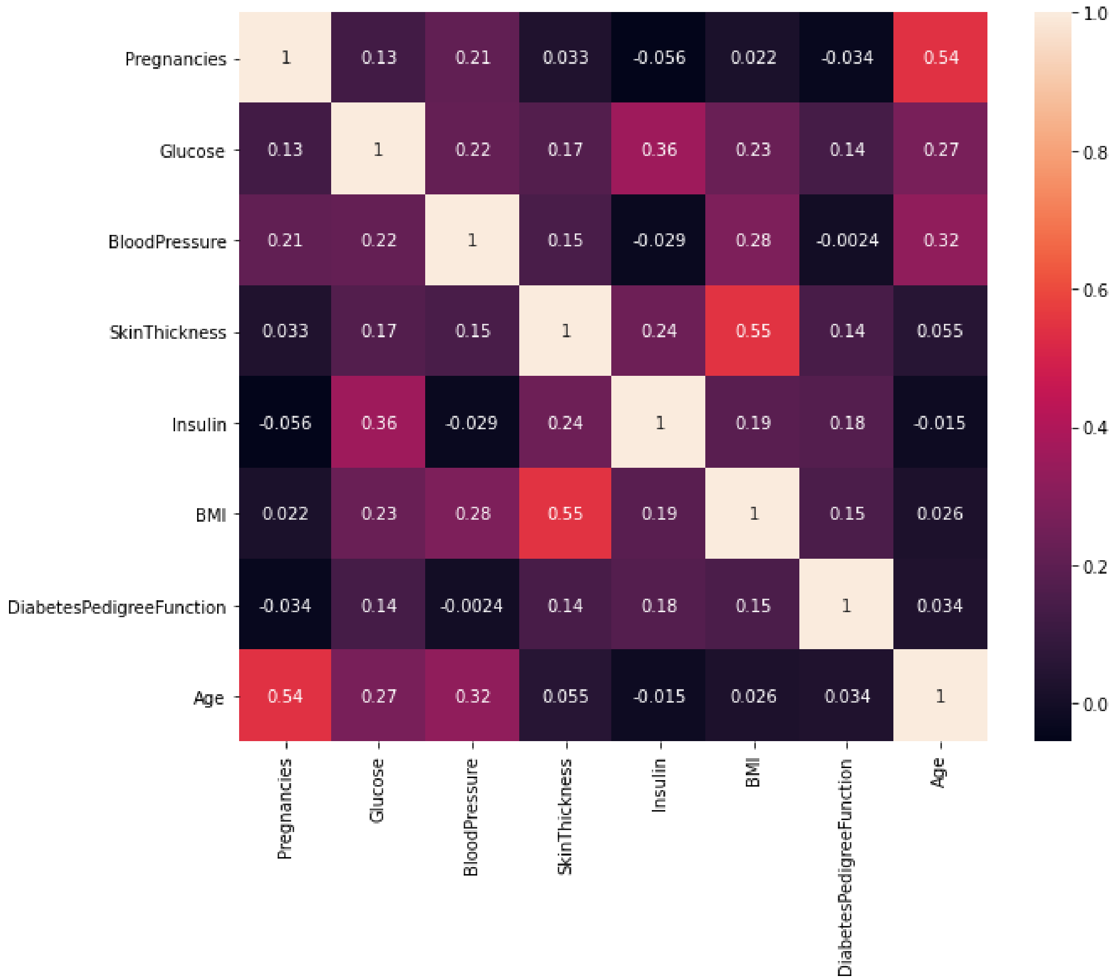
	Pregnancies	Glucose	BloodPressure	SkinThickness \
Pregnancies	1.000000	0.128213	0.208615	0.032568
Glucose	0.128213	1.000000	0.218937	0.172143
BloodPressure	0.208615	0.218937	1.000000	0.147809
SkinThickness	0.032568	0.172143	0.147809	1.000000
Insulin	-0.055697	0.357573	-0.028721	0.238188
BMI	0.021546	0.231400	0.281132	0.546951
DiabetesPedigreeFunction	-0.033523	0.137327	-0.002378	0.142977
Age	0.544341	0.266909	0.324915	0.054514

```
                            Insulin       BMI  DiabetesPedigreeFunction  \
Pregnancies               -0.055697  0.021546                 -0.033523
Glucose                    0.357573  0.231400                  0.137327
BloodPressure             -0.028721  0.281132                 -0.002378
SkinThickness              0.238188  0.546951                  0.142977
Insulin                    1.000000  0.189022                  0.178029
BMI                        0.189022  1.000000                  0.153506
DiabetesPedigreeFunction   0.178029  0.153506                  1.000000
Age                       -0.015413  0.025744                  0.033561

                               Age
Pregnancies               0.544341
Glucose                   0.266909
BloodPressure             0.324915
SkinThickness             0.054514
Insulin                  -0.015413
BMI                       0.025744
DiabetesPedigreeFunction  0.033561
Age                       1.000000
```

[16]:
```
plt.figure(figsize=(10,8))
sns.heatmap(healthcare_data.corr(),annot=True)
```

[16]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x2d8fb01d1c0>
```



Observation : There is no strong linear relation in between the feature variables.

1.0.3 Project Task: Week 3

Data Modeling:

1. Devise strategies for model building. It is important to decide the right validation framework. Express your thought process.
2. Apply an appropriate classification algorithm to build a model. Compare various models with the results from KNN algorithm.
3. Data Modeling: Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

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

- First we will split the data using train test split. Here validation framework is train test split because the Target column data is balanced. I prefer k-fold cross validation technique when there is highly bias data. If I use train test split in this case then there will be chances of missing the data points while training the model.
- will create the model using different classification algorithm
- compare accuracy of every model.
- will go for higher accuracy appropriate model.

[17]:
```
# divide the data
X=healthcare_data.drop('Outcome',axis=1)
y=healthcare_data['Outcome']
```

[18]:
```
# train test split
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.
↪2,random_state=21)
```

[19]:
```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)
```

```
(614, 8)
(154, 8)
(614,)
(154,)
```

[20]:
```
#Apply an appropriate classification algorithm to build a model. Compare␣
↪various models with the results from KNN algorithm.
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors=4)
knn_model.fit(X_train,y_train)
```

[20]:
```
KNeighborsClassifier(n_neighbors=4)
```

[21]:
```
# prediction from training and the testing data to know model accuracy
knn_y_pred_train= knn_model.predict(X_train)
knn_y_pred_test = knn_model.predict(X_test)
```

[22]:
```
# Check whether the model is overfitted,underfitted or it is appropriate
from sklearn.metrics import accuracy_score,classification_report
print('The accuracy of the training model is :
↪',accuracy_score(y_train,knn_y_pred_train))
print('The accuracy of the testing model is :
↪',accuracy_score(y_test,knn_y_pred_test))
```

```
The accuracy of the training model is : 0.8143322475570033
The accuracy of the testing model is : 0.6688311688311688
```

[23]:
```
print(classification_report(y_test,knn_y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.66      0.93      0.77        94
           1       0.70      0.27      0.39        60

    accuracy                           0.67       154
   macro avg       0.68      0.60      0.58       154
weighted avg       0.68      0.67      0.62       154
```

1.0.4 Using Logistic Regression

[24]:
```
from sklearn.linear_model import LogisticRegression
log_model = LogisticRegression()
log_model.fit(X_train,y_train)
```

```
C:\Users\nisarg\anaconda3\lib\site-
packages\sklearn\linear_model\_logistic.py:762: 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-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[24]:
```
LogisticRegression()
```

[25]:
```
# prediction from training and the testing data to know model accuracy
log_y_pred_train= log_model.predict(X_train)
log_y_pred_test = log_model.predict(X_test)
```

[26]:
```
print('The accuracy of the training model is :
 ↪',accuracy_score(y_train,log_y_pred_train))
print('The accuracy of the testing model is :
 ↪',accuracy_score(y_test,log_y_pred_test))
```

```
The accuracy of the training model is : 0.7915309446254072
The accuracy of the testing model is : 0.7402597402597403
```

[27]:
```
print(classification_report(y_test,log_y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.73      0.90      0.81        94
           1       0.76      0.48      0.59        60

    accuracy                           0.74       154
   macro avg       0.75      0.69      0.70       154
weighted avg       0.74      0.74      0.72       154
```

Observation : Training accuracy of knn is higher than logistic but testing accuracy of logistic is greater than knn.

1.0.5 Using Decision Tree

[28]:
```
from sklearn.tree import DecisionTreeClassifier
dec_model = DecisionTreeClassifier()
dec_model.fit(X_train,y_train)
```

[28]:
```
DecisionTreeClassifier()
```

[29]:
```
# prediction from training and the testing data to know model accuracy
dec_y_pred_train= dec_model.predict(X_train)
dec_y_pred_test = dec_model.predict(X_test)
```

[30]:
```
print('The accuracy of the training model is :
 ↪',accuracy_score(y_train,dec_y_pred_train))
print('The accuracy of the testing model is :
 ↪',accuracy_score(y_test,dec_y_pred_test))
```

```
The accuracy of the training model is : 1.0
The accuracy of the testing model is : 0.7012987012987013
```

[31]:
```
print(classification_report(y_test,dec_y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.74      0.79      0.76        94
           1       0.63      0.57      0.60        60

    accuracy                           0.70       154
   macro avg       0.68      0.68      0.68       154
weighted avg       0.70      0.70      0.70       154
```

Observation : Here both training and testing accuracy is greator than knn.

1.0.6 Using Random forest

[32]:
```
from sklearn.ensemble import RandomForestClassifier
ref_model = RandomForestClassifier(n_estimators=150)
ref_model.fit(X_train,y_train)
```

[32]:
```
RandomForestClassifier(n_estimators=150)
```

[33]:
```
# prediction from training and the testing data to know model accuracy
ref_y_pred_train= ref_model.predict(X_train)
ref_y_pred_test = ref_model.predict(X_test)
```

[34]:
```
print('The accuracy of the training model is :
 ↪',accuracy_score(y_train,ref_y_pred_train))
print('The accuracy of the testing model is :
 ↪',accuracy_score(y_test,ref_y_pred_test))
```

```
The accuracy of the training model is : 1.0
The accuracy of the testing model is : 0.7402597402597403
```

[35]:
```
print(classification_report(y_test,ref_y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.73      0.91      0.81        94
           1       0.78      0.47      0.58        60

    accuracy                           0.74       154
   macro avg       0.75      0.69      0.70       154
weighted avg       0.75      0.74      0.72       154
```

Observation : Here than both the accuracy is higher than knn and decision tree.

1.1 Hyperparameter tunning

[36]:
```
from sklearn.model_selection import RandomizedSearchCV
Ran_ref_model = RandomForestClassifier()
```

[37]:
```
n_estimators= [int(x) for x in np.linspace(100,1000,10)]
criterion=["gini", "entropy"]
max_depth=[4,5,6]
min_samples_split=[2,3,4]
min_samples_leaf=[1,2,3]
```

[38]:
```
param = {'n_estimators':n_estimators,'criterion':criterion,'max_depth':
↪max_depth,'min_samples_split':min_samples_split,'min_samples_leaf':
↪min_samples_leaf}
```

[39]:
```
ran_ref_final_model =␣
↪RandomizedSearchCV(Ran_ref_model,param_distributions=param,n_iter=50,cv=4,n_jobs=-1,verbose
ran_ref_final_model.fit(X_train,y_train)
```

```
Fitting 4 folds for each of 50 candidates, totalling 200 fits

[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   18.7s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed:  1.5min finished
```

[39]:
```
RandomizedSearchCV(cv=4, estimator=RandomForestClassifier(), n_iter=50,
                   n_jobs=-1,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': [4, 5, 6],
                                        'min_samples_leaf': [1, 2, 3],
                                        'min_samples_split': [2, 3, 4],
                                        'n_estimators': [100, 200, 300, 400,
                                                         500, 600, 700, 800,
                                                         900, 1000]},
                   verbose=1)
```

[40]:
```
ran_ref_final_model.best_estimator_
```

[40]:
```
RandomForestClassifier(max_depth=6, min_samples_leaf=3, min_samples_split=3,
                       n_estimators=500)
```

[41]:
```
# prediction from training and the testing data to know model accuracy
ran_ref_y_pred_train= ran_ref_final_model.predict(X_train)
ran_ref_y_pred_test = ran_ref_final_model.predict(X_test)
```

[42]:
```
print('The accuracy of the training model is :
↪',accuracy_score(y_train,ran_ref_y_pred_train))
print('The accuracy of the testing model is :
↪',accuracy_score(y_test,ran_ref_y_pred_test))
```

```
The accuracy of the training model is : 0.8583061889250815
The accuracy of the testing model is : 0.7402597402597403
```

[43]:
```
print(classification_report(y_test,ran_ref_y_pred_test))
```

```
              precision    recall  f1-score   support

           0       0.73      0.91      0.81        94
```

```
           1       0.78      0.47      0.58        60

    accuracy                           0.74       154
   macro avg       0.75      0.69      0.70       154
weighted avg       0.75      0.74      0.72       154
```

Observation : Using the random Forest Classifier with hyper paramter tunning we are getting the appropriate model where accuracy of both train and test is higher.

Data Modeling: Create a classification report by analyzing sensitivity, specificity, AUC (ROC curve), etc. Please be descriptive to explain what values of these parameter you have used.

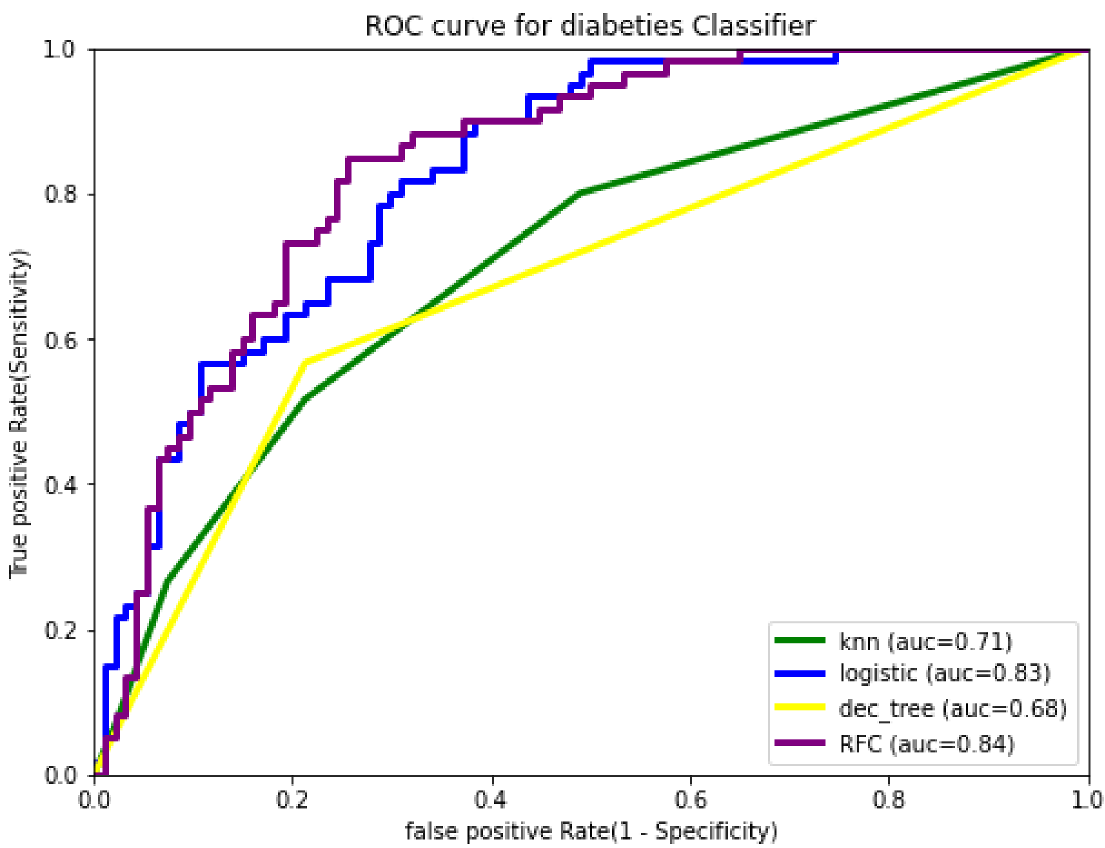
[44]:
```
from sklearn.metrics import roc_curve,roc_auc_score
# Taking prediction probability from every model to calculate fpr,tpr,threshold␣
 ↪values to draw roc curve
y_knn_pred = knn_model.predict_proba(X_test)[:,1]
y_log_pred = log_model.predict_proba(X_test)[:,1]
y_dec_pred = dec_model.predict_proba(X_test)[:,1]
y_ranfor_pred = ran_ref_final_model.predict_proba(X_test)[:,1]
```

[45]:
```
# calculation of fpr(False positive rate), tpr(True positive rate) values for␣
 ↪every model
fpr_knn,tpr_knn,threshold_knn =␣
 ↪roc_curve(y_test,y_knn_pred,drop_intermediate=False)
fpr_log,tpr_log,threshold_log =␣
 ↪roc_curve(y_test,y_log_pred,drop_intermediate=False)
fpr_dec,tpr_dec,threshold_dec =␣
 ↪roc_curve(y_test,y_dec_pred,drop_intermediate=False)
fpr_ran,tpr_ran,threshold_ran =␣
 ↪roc_curve(y_test,y_ranfor_pred,drop_intermediate=False)
```

[46]:
```
# Calculating AUC values for every model
auc_knn = roc_auc_score(y_test,y_knn_pred)
auc_log = roc_auc_score(y_test,y_log_pred)
auc_dec = roc_auc_score(y_test,y_dec_pred)
auc_ran = roc_auc_score(y_test,y_ranfor_pred)
```

[48]:
```
# Plotting roc curve to check which model is gives higher␣
 ↪accuracy,recall,precision
plt.figure(figsize=(8,6))
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.0])
plt.title("ROC curve for diabeties Classifier")
plt.xlabel('false positive Rate(1 - Specificity)')
plt.ylabel("True positive Rate(Sensitivity)")
```

```
plt.plot(fpr_knn,tpr_knn,color='green',lw=3,label='knn (auc=%0.2f)' %auc_knn)
plt.plot(fpr_log,tpr_log,color='blue',lw=3,label='logistic (auc=%0.2f)'%auc_log)
plt.plot(fpr_dec,tpr_dec,color='yellow',lw=3,label='dec_tree (auc=%0.2f)'␣
 ↪%auc_dec)
plt.plot(fpr_ran,tpr_ran,color='purple',lw=3,label='RFC (auc=%0.2f)'%auc_ran)
plt.legend()
plt.show()
```



Observation : from the above roc_auc curve we can say that for the above problem Logistic regression and random forest model gives us better output with auc score of 0.83 and 0.84 respectively

1.1.1 Project Task: Week 4

Data Reporting:

2. Create a dashboard in tableau by choosing appropriate chart types and metrics useful for the business. The dashboard must entail the following:

a. Pie chart to describe the diabetic or non-diabetic population

b. Scatter charts between relevant variables to analyze the relationships

c. Histogram or frequency charts to analyze the distribution of the data

d. Heatmap of correlation analysis among the relevant variables

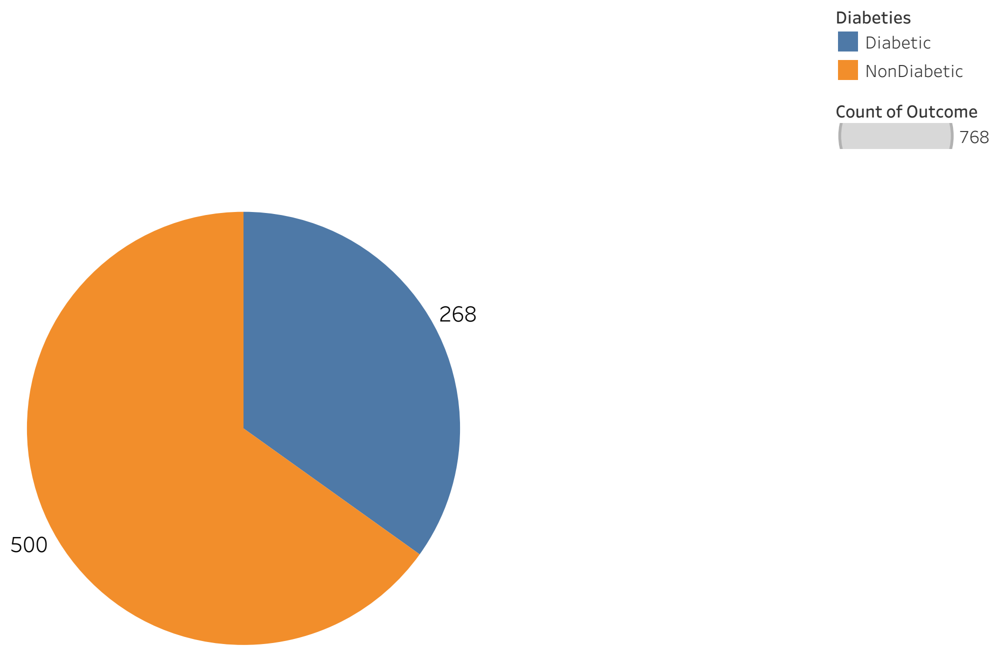
e. Create bins of these age values: 20-25, 25-30, 30-35, etc. Analyze different variables for these age brackets using a bubble chart.

[41]:
```
healthcare_data.to_csv(r'C:\Users\nisarg\Desktop\Data Science\Capstone␣
↪Projects\Project 2\Healthcare - Diabetes\tableaufile.csv',index=False)
```

Observation : we have converted the file to csv for visualization purpose

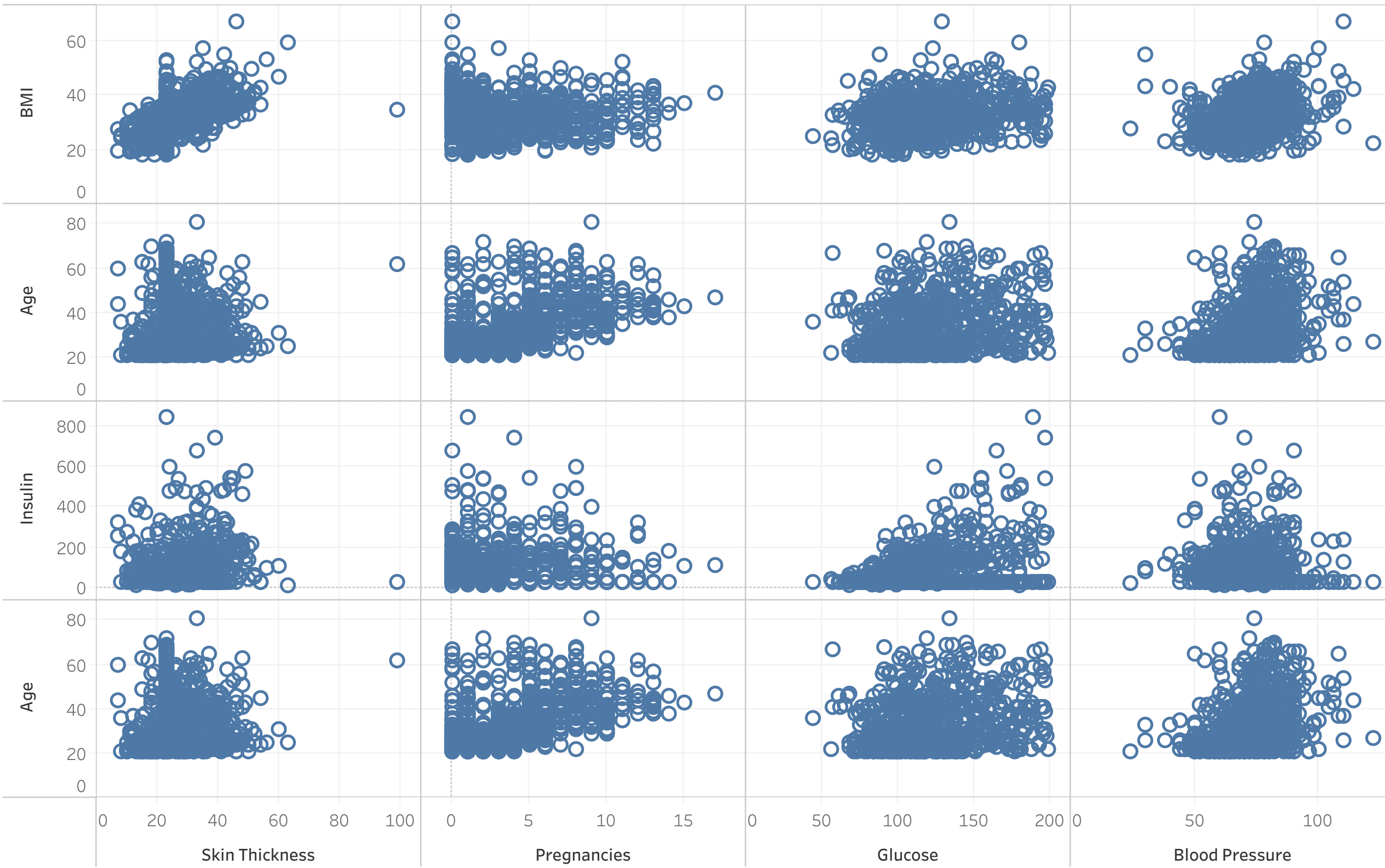
Analysis of Diabetic Patients

Pie chart to describe the diabetic or non-di..	Scatter charts between relevant var..	Histogram plot to see the distribution of th..	Operation on age column by creating th..



Analysis of Diabetic Patients

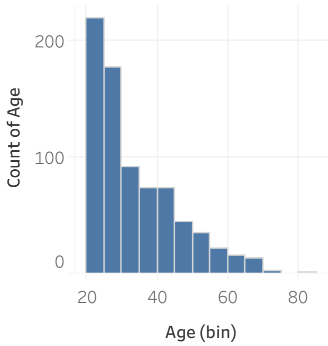
Pie chart to describe the diabetic or non-di..	Scatter charts between relevant var..	Histogram plot to see the distribution of th..	Operation on age column by creating th..



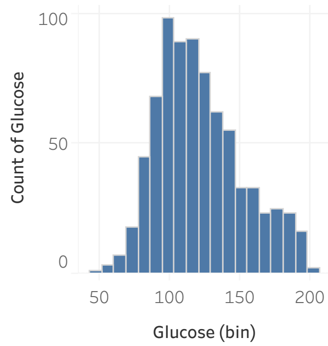
Analysis of Diabetic Patients

Pie chart to describe the diabetic or non-di..	Scatter charts between relevant var..	Histogram plot to see the distribution of th..	Operation on age column by creating th..

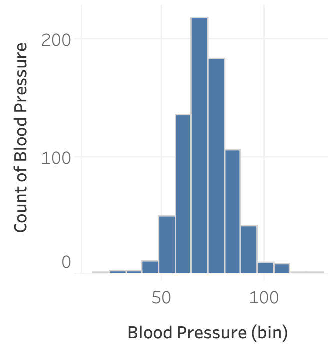
Age



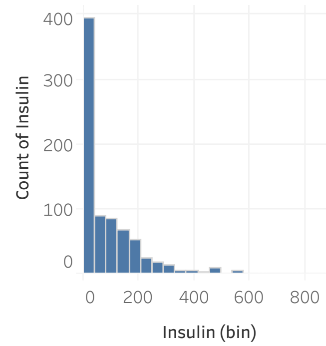
Glucose



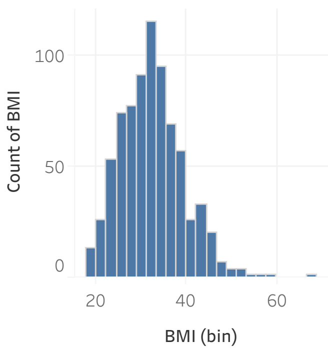
BloodPressure



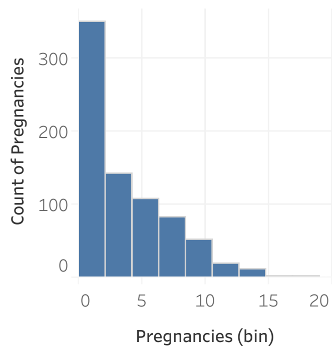
Insulin



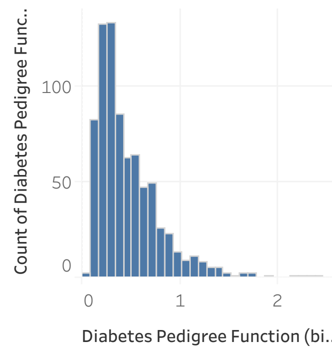
BMI



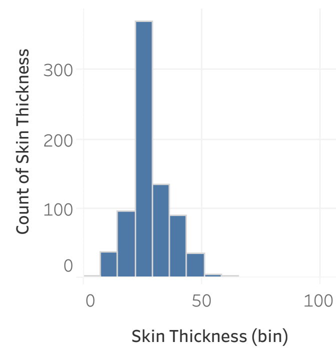
Pregnancies



DPF



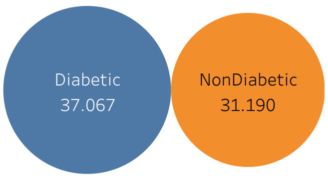
Skin Thickness



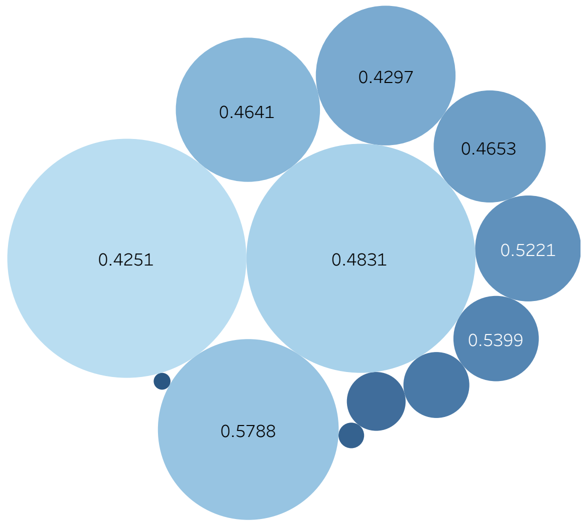
Analysis of Diabetic Patients

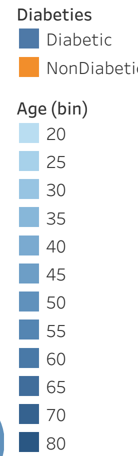
Pie chart to describe the diabetic or non-di..	Scatter charts between relevant var..	Histogram plot to see the distribution of th..	Operation on age column by creating th..

Avg age of Diabetic Patient



avg DPF for age group





Pregnancies with age groups