# dfen4g4jm

February 28, 2023

# 1 Diabetes Classification-KNN-detailed

## 1.1 Import the required libraries

- mtend - is an ML library.
- mpy - is used to perform mathamatical operations.
- ndas - is used in data managment operations.
- tplotlib - is used to plot graphs.
- seaborn - is used to visualize random distributions.

```
[1]: from mlxtend.plotting import plot_decision_regions
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     sns.set()
     import warnings
     warnings.filterwarnings('ignore')
     %matplotlib inline
     #plt.style.use('ggplot')
     #ggplot is R based visualisation package that provides better graphics with␣
      ↪higher level of abstraction
```

## 1.2 Import and view data.

- pd.read_csv - is used to load the Data set.
- db.head() - is used to view top 5 values.

```
[2]: diabetes_data = pd.read_csv('C:/Users/suman/Desktop/DS learn/Project/Pima␣
      ↪Indians Diabetes/diabetes.csv')
     diabetes_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
```

### 1.3 Perform EDA and statistical analysis.

- EDA = Exploratory Data analysis.
- It is a method used to analyze and summarize data sets.
- db.info() = It gives the info of data types of each feature.
- In Data science variable/feature = columns and record/observation/trial = rows.
- db.discribe() = It gives the statistical data of the data set.
- It gives Count, Mean, STD, min, max, 25th percentile, 50th percentile and 75th percentile.

[3]:
```
diabetes_data.info(verbose=True)
```

```
<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
```

[4]:
```
diabetes_data.describe()
```

[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
```

## 1.4 Data cleaning

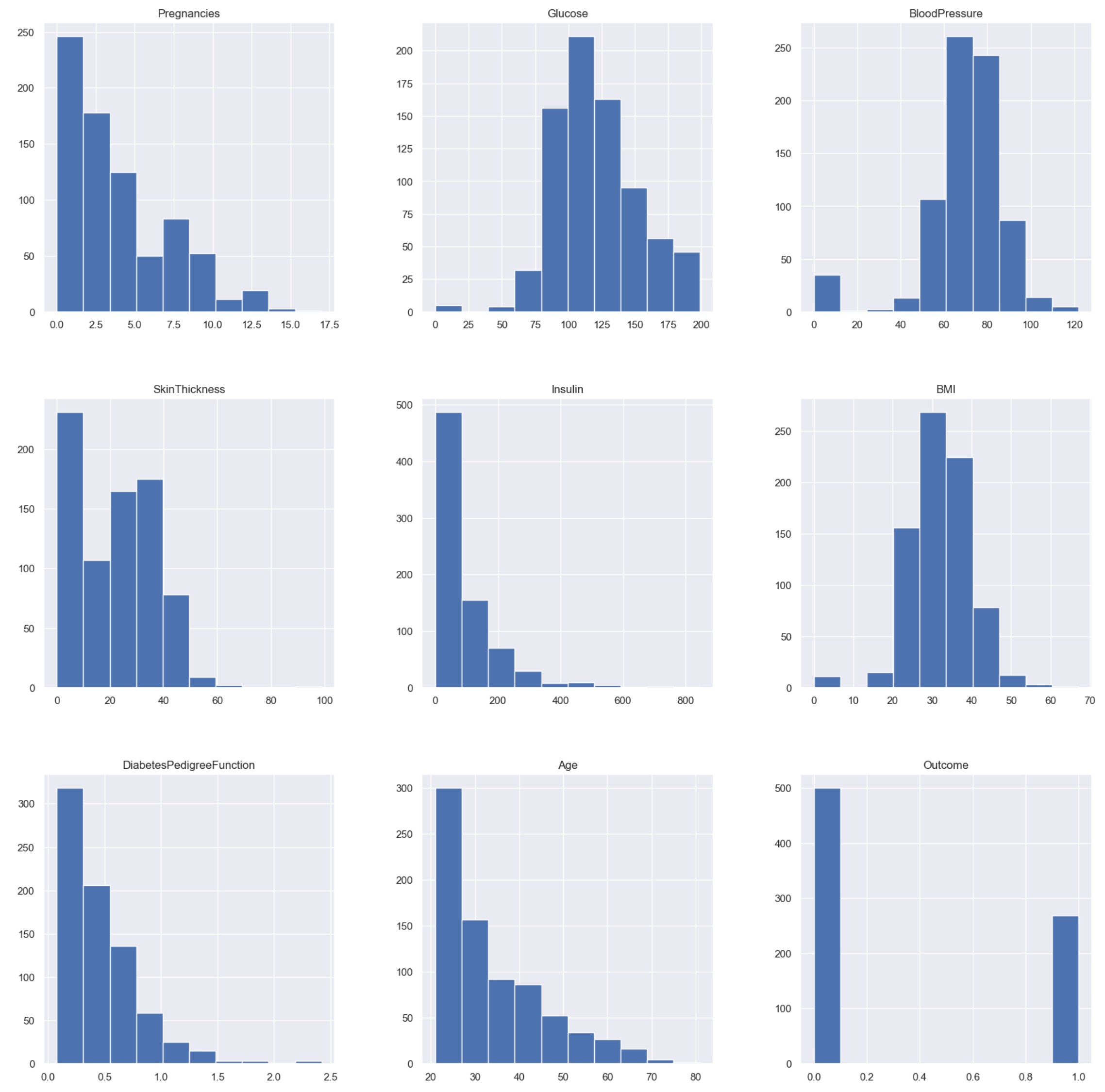
- Following columns or variables have an invalid zero value:
- Glucose
- BloodPressure
- SkinThickness
- Insulin
- BMI
- On these columns, a value of zero does not make sense and thus indicates missing value.
- It is better to replace zeros with nan since after that counting them would be easier and zeros need to be replaced with suitable values.
- The copy() method returns a copy of the specified list / dataset.
- replace() replaces the specified value with other value.
- isnull().sum() gives the total null values.

[5]:
```
diabetes_data_copy = diabetes_data.copy(deep = True)
diabetes_data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']]␣
  ↪=␣
  ↪diabetes_data_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].
  ↪replace(0,np.NaN)

print(diabetes_data_copy.isnull().sum())
```

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

To fill these Nan values the data distribution needs to be understood. db.hist() = gives a Histogram for each feature. - It is used to summarize discrete or continuous data that are measured on an interval scale. It is often used to illustrate the major features of the distribution of the data in a convenient form.

[6]:
```
p = diabetes_data.hist(figsize = (20,20))
```

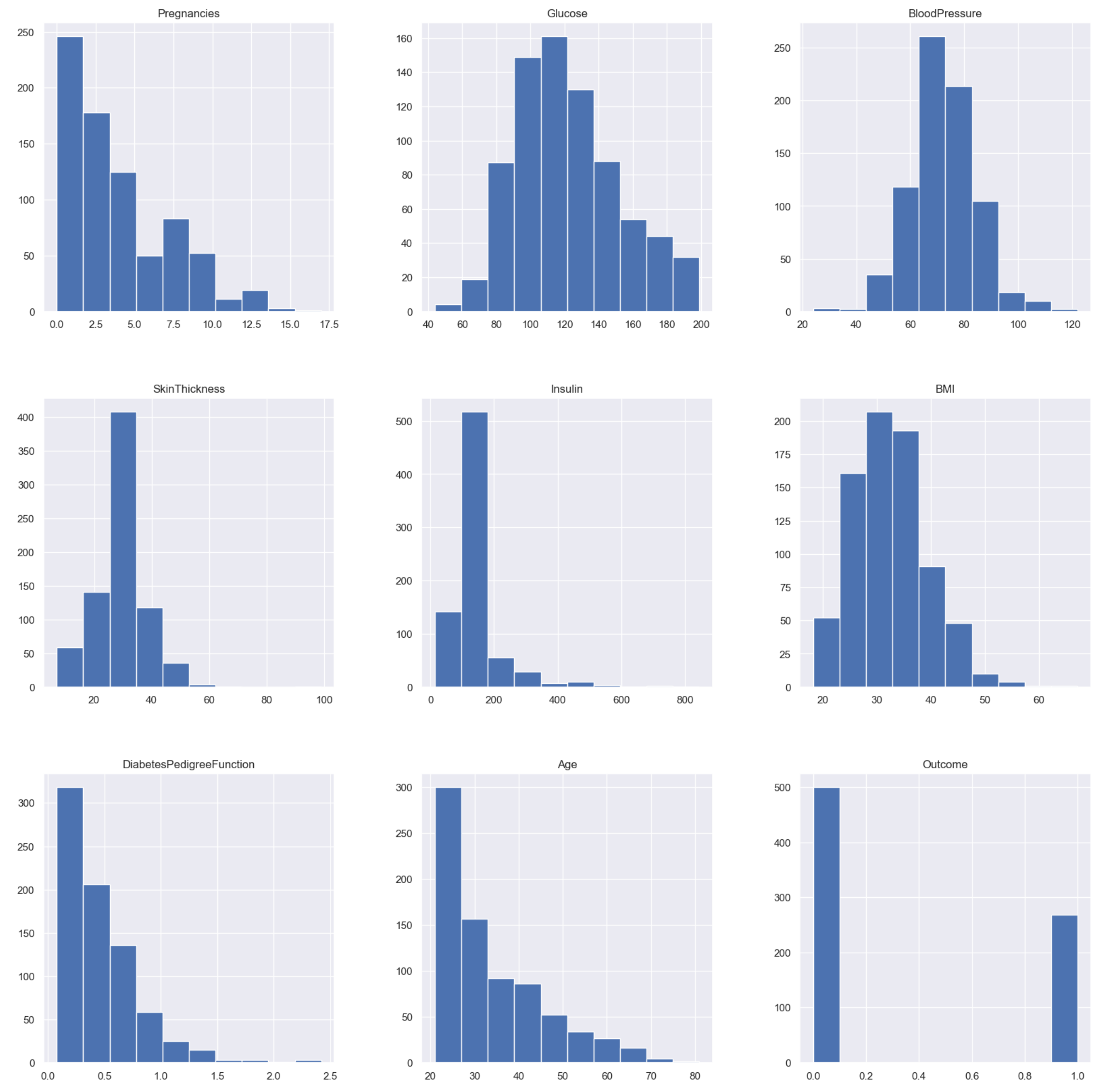


### 1.5 Aiming to impute nan values for the features in accordance with their distribution and plotting after replacement.

[7]:
```
diabetes_data_copy['Glucose'].fillna(diabetes_data_copy['Glucose'].mean(),␣
  ↪inplace = True)
diabetes_data_copy['BloodPressure'].fillna(diabetes_data_copy['BloodPressure'].
  ↪mean(), inplace = True)
diabetes_data_copy['SkinThickness'].fillna(diabetes_data_copy['SkinThickness'].
  ↪median(), inplace = True)
```

```
diabetes_data_copy['Insulin'].fillna(diabetes_data_copy['Insulin'].median(),␣
  ↪inplace = True)
diabetes_data_copy['BMI'].fillna(diabetes_data_copy['BMI'].median(), inplace =␣
  ↪True)
```

[8]:
```
p = diabetes_data_copy.hist(figsize = (20,20))
```



## 1.6 Skewness

- A **left-skewed distribution** has a long left tail. Left-skewed distributions are also called negatively-skewed distributions. That's because there is a long tail in the negative direction on the number line. The mean is also to the left of the peak.
- A **right-skewed distribution** has a long right tail. Right-skewed distributions are also

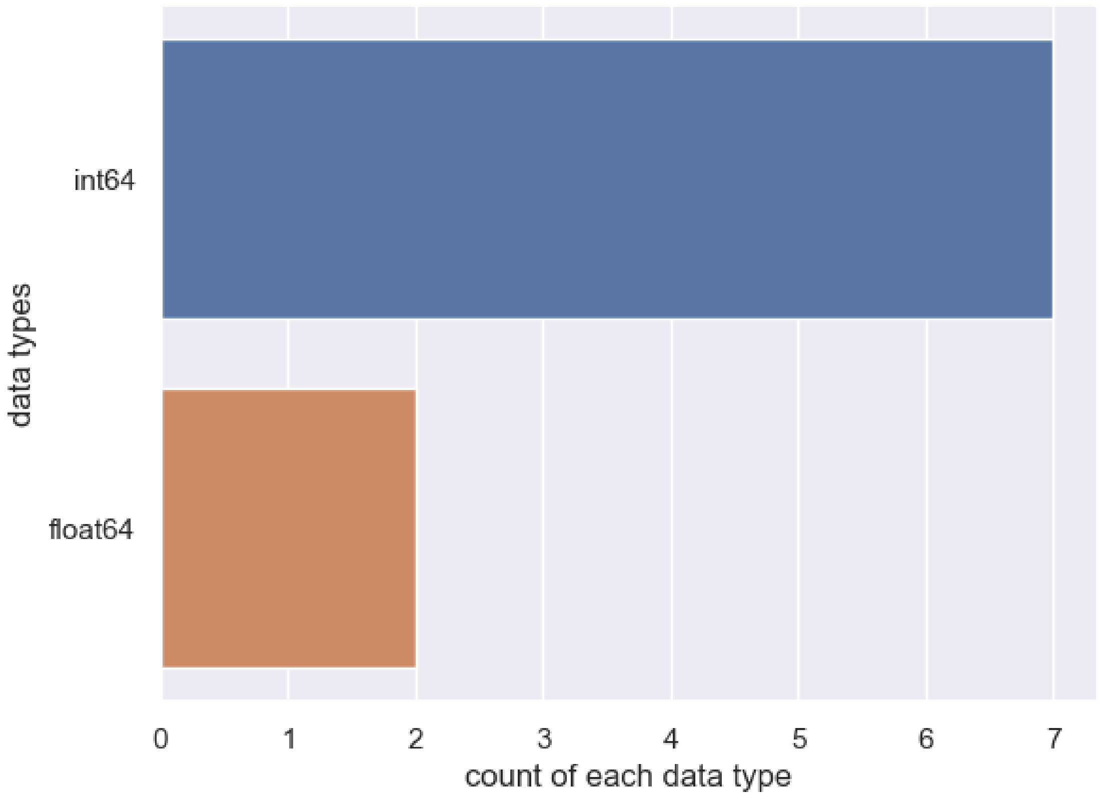
called positive-skew distributions. That's because there is a long tail in the positive direction on the number line. The mean is also to the right of the peak.

Observing the shape of the data.

```
[9]: diabetes_data.shape
```

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

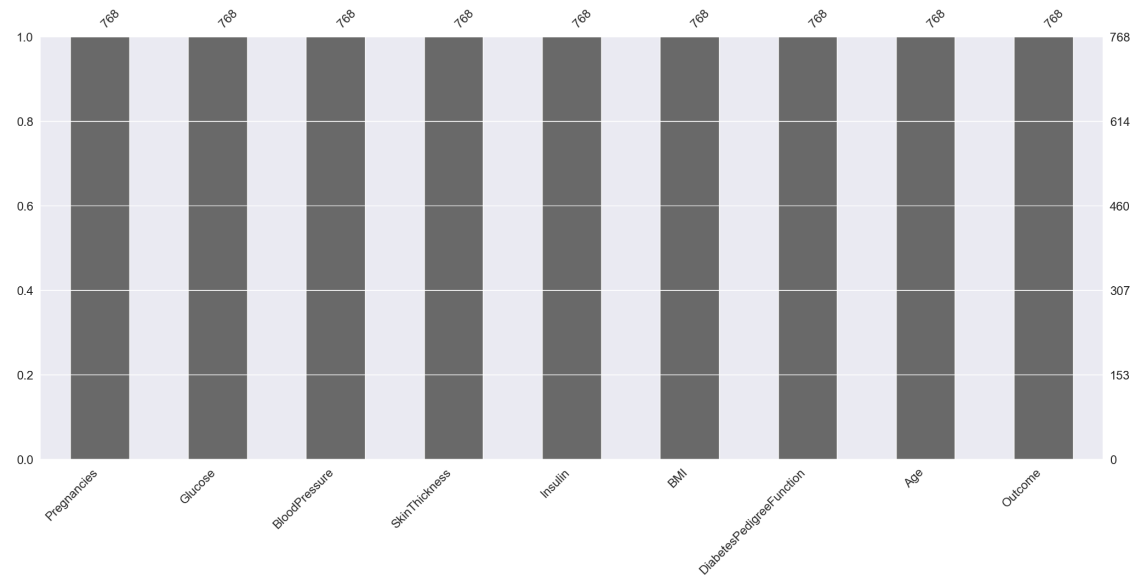
```
[10]: sns.countplot(y=diabetes_data.dtypes ,data=diabetes_data)
      plt.xlabel("count of each data type")
      plt.ylabel("data types")
      plt.show()
```



## 1.7 Null count analysis

missingno library is used to count the null vlaues.

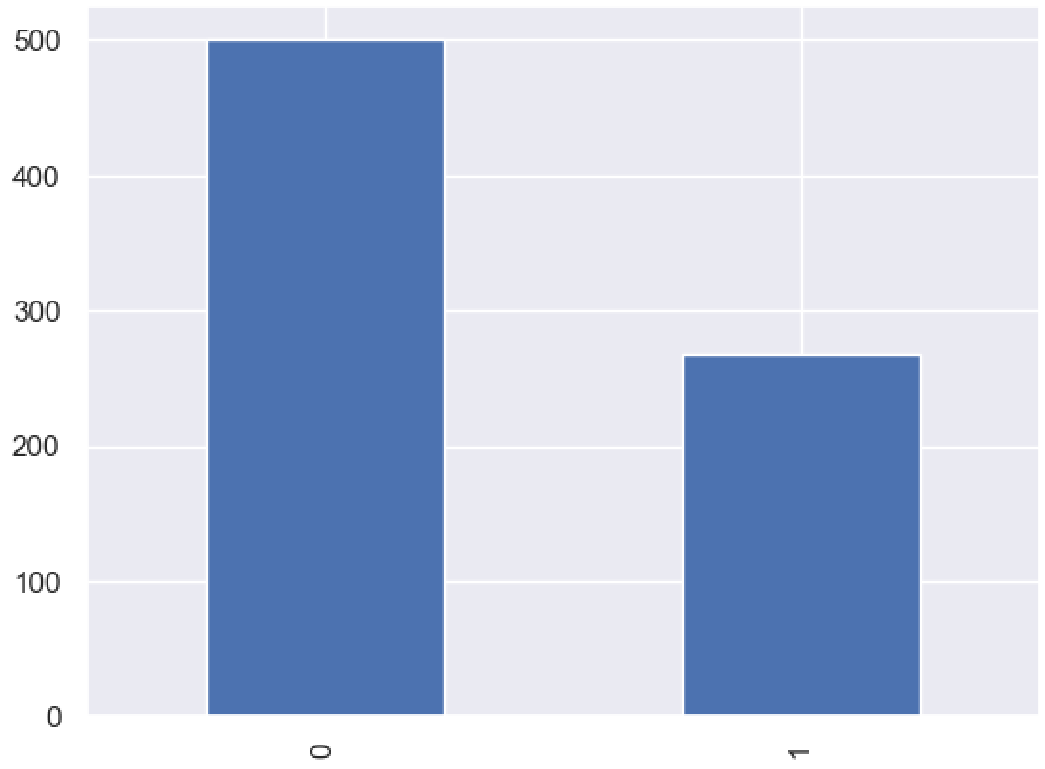
```
[11]: import missingno as msno
      p=msno.bar(diabetes_data)
```



Checking the balance of the data by plotting the count of outcomes by their value

[12]:
```
print(diabetes_data.Outcome.value_counts())
p=diabetes_data.Outcome.value_counts().plot(kind="bar")
```

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

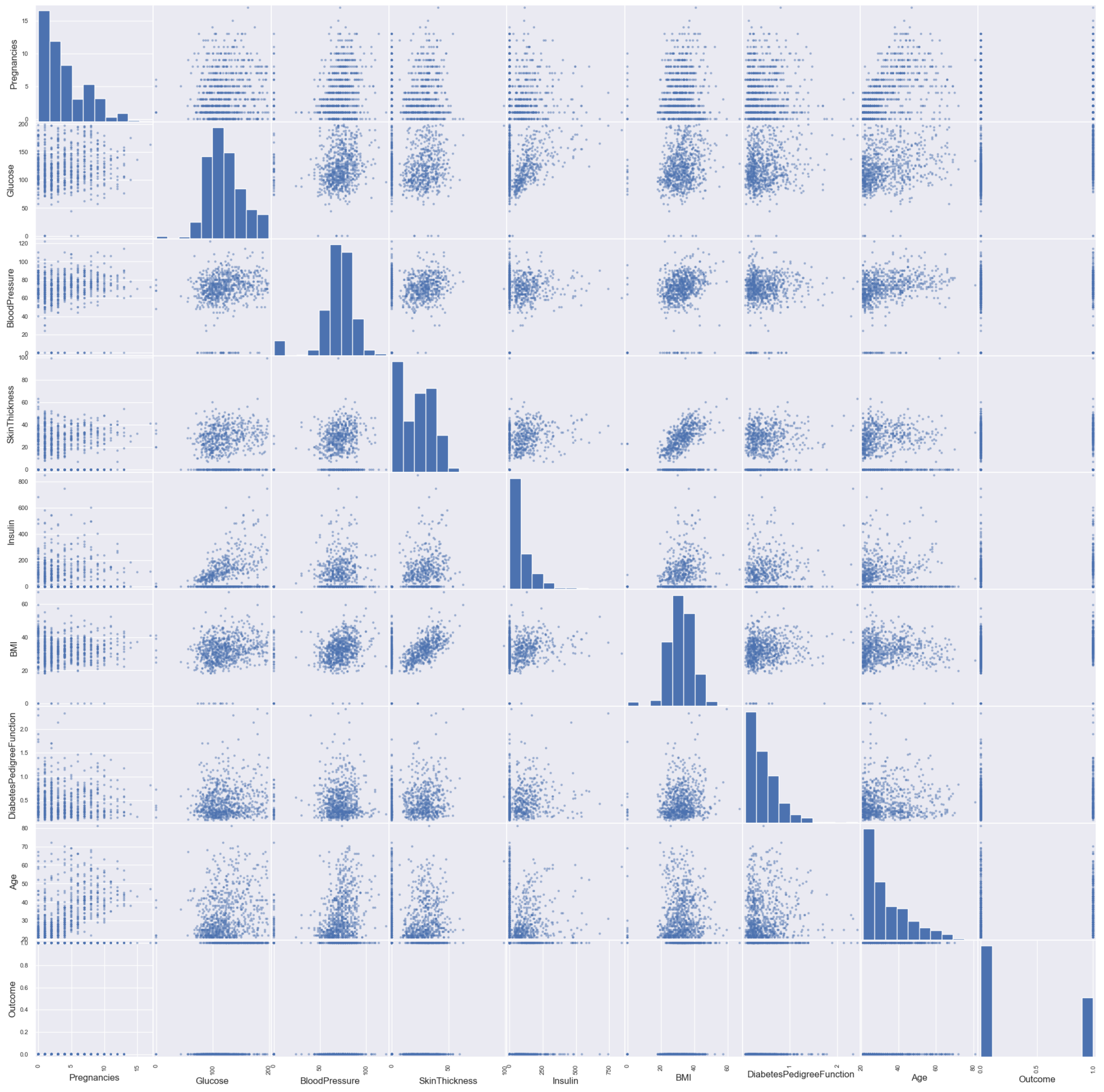


The above graph shows that the data is biased towards datapoints having outcome value as 0 where it means that diabetes was not present actually. The number of non-diabetics is almost twice the number of diabetic patients.

Scatter plot of uncleaned data.

```
[13]: pd.plotting.scatter_matrix(diabetes_data,figsize=(25, 25))
```

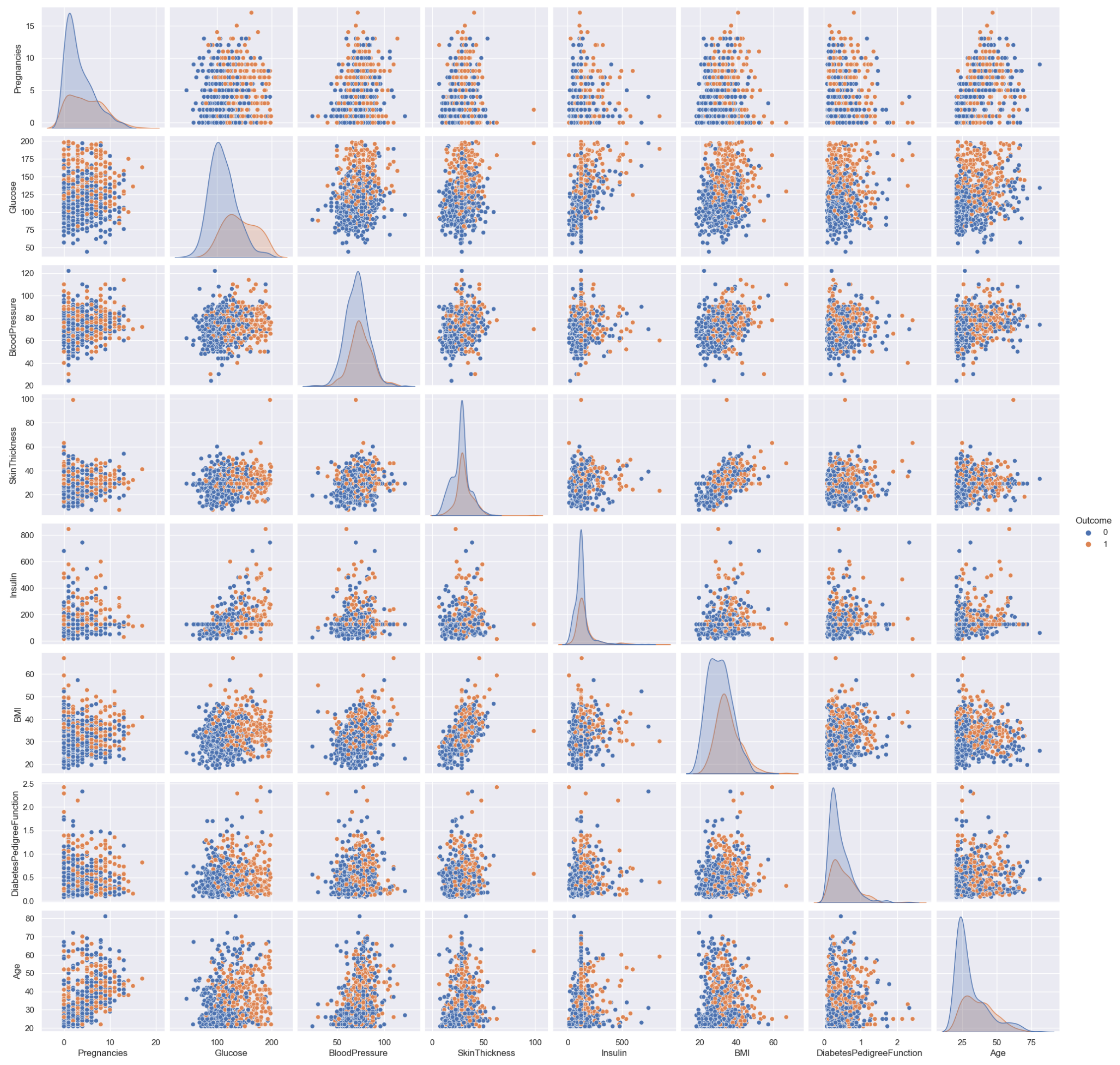
```
[13]: array([[<AxesSubplot:xlabel='Pregnancies', ylabel='Pregnancies'>,
              <AxesSubplot:xlabel='Glucose', ylabel='Pregnancies'>,
              <AxesSubplot:xlabel='BloodPressure', ylabel='Pregnancies'>,
              <AxesSubplot:xlabel='SkinThickness', ylabel='Pregnancies'>,
              <AxesSubplot:xlabel='Insulin', ylabel='Pregnancies'>,
              <AxesSubplot:xlabel='BMI', ylabel='Pregnancies'>,
              <AxesSubplot:xlabel='DiabetesPedigreeFunction', ylabel='Pregnancies'>,
              <AxesSubplot:xlabel='Age', ylabel='Pregnancies'>,
              <AxesSubplot:xlabel='Outcome', ylabel='Pregnancies'>],
             [<AxesSubplot:xlabel='Pregnancies', ylabel='Glucose'>,
              <AxesSubplot:xlabel='Glucose', ylabel='Glucose'>,
              <AxesSubplot:xlabel='BloodPressure', ylabel='Glucose'>,
              <AxesSubplot:xlabel='SkinThickness', ylabel='Glucose'>,
              <AxesSubplot:xlabel='Insulin', ylabel='Glucose'>,
              <AxesSubplot:xlabel='BMI', ylabel='Glucose'>,
              <AxesSubplot:xlabel='DiabetesPedigreeFunction', ylabel='Glucose'>,
              <AxesSubplot:xlabel='Age', ylabel='Glucose'>,
              <AxesSubplot:xlabel='Outcome', ylabel='Glucose'>],
             [<AxesSubplot:xlabel='Pregnancies', ylabel='BloodPressure'>,
              <AxesSubplot:xlabel='Glucose', ylabel='BloodPressure'>,
              <AxesSubplot:xlabel='BloodPressure', ylabel='BloodPressure'>,
              <AxesSubplot:xlabel='SkinThickness', ylabel='BloodPressure'>,
              <AxesSubplot:xlabel='Insulin', ylabel='BloodPressure'>,
              <AxesSubplot:xlabel='BMI', ylabel='BloodPressure'>,
              <AxesSubplot:xlabel='DiabetesPedigreeFunction', ylabel='BloodPressure'>,
              <AxesSubplot:xlabel='Age', ylabel='BloodPressure'>,
              <AxesSubplot:xlabel='Outcome', ylabel='BloodPressure'>],
             [<AxesSubplot:xlabel='Pregnancies', ylabel='SkinThickness'>,
              <AxesSubplot:xlabel='Glucose', ylabel='SkinThickness'>,
              <AxesSubplot:xlabel='BloodPressure', ylabel='SkinThickness'>,
              <AxesSubplot:xlabel='SkinThickness', ylabel='SkinThickness'>,
              <AxesSubplot:xlabel='Insulin', ylabel='SkinThickness'>,
              <AxesSubplot:xlabel='BMI', ylabel='SkinThickness'>,
              <AxesSubplot:xlabel='DiabetesPedigreeFunction', ylabel='SkinThickness'>,
              <AxesSubplot:xlabel='Age', ylabel='SkinThickness'>,
              <AxesSubplot:xlabel='Outcome', ylabel='SkinThickness'>],
             [<AxesSubplot:xlabel='Pregnancies', ylabel='Insulin'>,
              <AxesSubplot:xlabel='Glucose', ylabel='Insulin'>,
              <AxesSubplot:xlabel='BloodPressure', ylabel='Insulin'>,
              <AxesSubplot:xlabel='SkinThickness', ylabel='Insulin'>,
              <AxesSubplot:xlabel='Insulin', ylabel='Insulin'>,
              <AxesSubplot:xlabel='BMI', ylabel='Insulin'>,
              <AxesSubplot:xlabel='DiabetesPedigreeFunction', ylabel='Insulin'>,
              <AxesSubplot:xlabel='Age', ylabel='Insulin'>,
              <AxesSubplot:xlabel='Outcome', ylabel='Insulin'>],
             [<AxesSubplot:xlabel='Pregnancies', ylabel='BMI'>,
              <AxesSubplot:xlabel='Glucose', ylabel='BMI'>,
              <AxesSubplot:xlabel='BloodPressure', ylabel='BMI'>,
              <AxesSubplot:xlabel='SkinThickness', ylabel='BMI'>,
              <AxesSubplot:xlabel='Insulin', ylabel='BMI'>,
              <AxesSubplot:xlabel='BMI', ylabel='BMI'>,
              <AxesSubplot:xlabel='DiabetesPedigreeFunction', ylabel='BMI'>,
              <AxesSubplot:xlabel='Age', ylabel='BMI'>,
              <AxesSubplot:xlabel='Outcome', ylabel='BMI'>],
             [<AxesSubplot:xlabel='Pregnancies', ylabel='DiabetesPedigreeFunction'>,
              <AxesSubplot:xlabel='Glucose', ylabel='DiabetesPedigreeFunction'>,
              <AxesSubplot:xlabel='BloodPressure', ylabel='DiabetesPedigreeFunction'>,
              <AxesSubplot:xlabel='SkinThickness', ylabel='DiabetesPedigreeFunction'>,
              <AxesSubplot:xlabel='Insulin', ylabel='DiabetesPedigreeFunction'>,
              <AxesSubplot:xlabel='BMI', ylabel='DiabetesPedigreeFunction'>,
              <AxesSubplot:xlabel='DiabetesPedigreeFunction',
      ylabel='DiabetesPedigreeFunction'>,
              <AxesSubplot:xlabel='Age', ylabel='DiabetesPedigreeFunction'>,
              <AxesSubplot:xlabel='Outcome', ylabel='DiabetesPedigreeFunction'>],
             [<AxesSubplot:xlabel='Pregnancies', ylabel='Age'>,
              <AxesSubplot:xlabel='Glucose', ylabel='Age'>,
              <AxesSubplot:xlabel='BloodPressure', ylabel='Age'>,
              <AxesSubplot:xlabel='SkinThickness', ylabel='Age'>,
              <AxesSubplot:xlabel='Insulin', ylabel='Age'>,
              <AxesSubplot:xlabel='BMI', ylabel='Age'>,
              <AxesSubplot:xlabel='DiabetesPedigreeFunction', ylabel='Age'>,
              <AxesSubplot:xlabel='Age', ylabel='Age'>,
              <AxesSubplot:xlabel='Outcome', ylabel='Age'>],
             [<AxesSubplot:xlabel='Pregnancies', ylabel='Outcome'>,
              <AxesSubplot:xlabel='Glucose', ylabel='Outcome'>,
              <AxesSubplot:xlabel='BloodPressure', ylabel='Outcome'>,
              <AxesSubplot:xlabel='SkinThickness', ylabel='Outcome'>,
              <AxesSubplot:xlabel='Insulin', ylabel='Outcome'>,
              <AxesSubplot:xlabel='BMI', ylabel='Outcome'>,
              <AxesSubplot:xlabel='DiabetesPedigreeFunction', ylabel='Outcome'>,
              <AxesSubplot:xlabel='Age', ylabel='Outcome'>,
              <AxesSubplot:xlabel='Outcome', ylabel='Outcome'>]], dtype=object)
```



The pairs plot builds on two basic figures, the histogram and the scatter plot. The histogram on the diagonal allows us to see the distribution of a single variable while the scatter plots on the upper and lower triangles show the relationship (or lack thereof) between two variables.

Pair plot for clean data

[14]:
```
p=sns.pairplot(diabetes_data_copy, hue = 'Outcome')
```

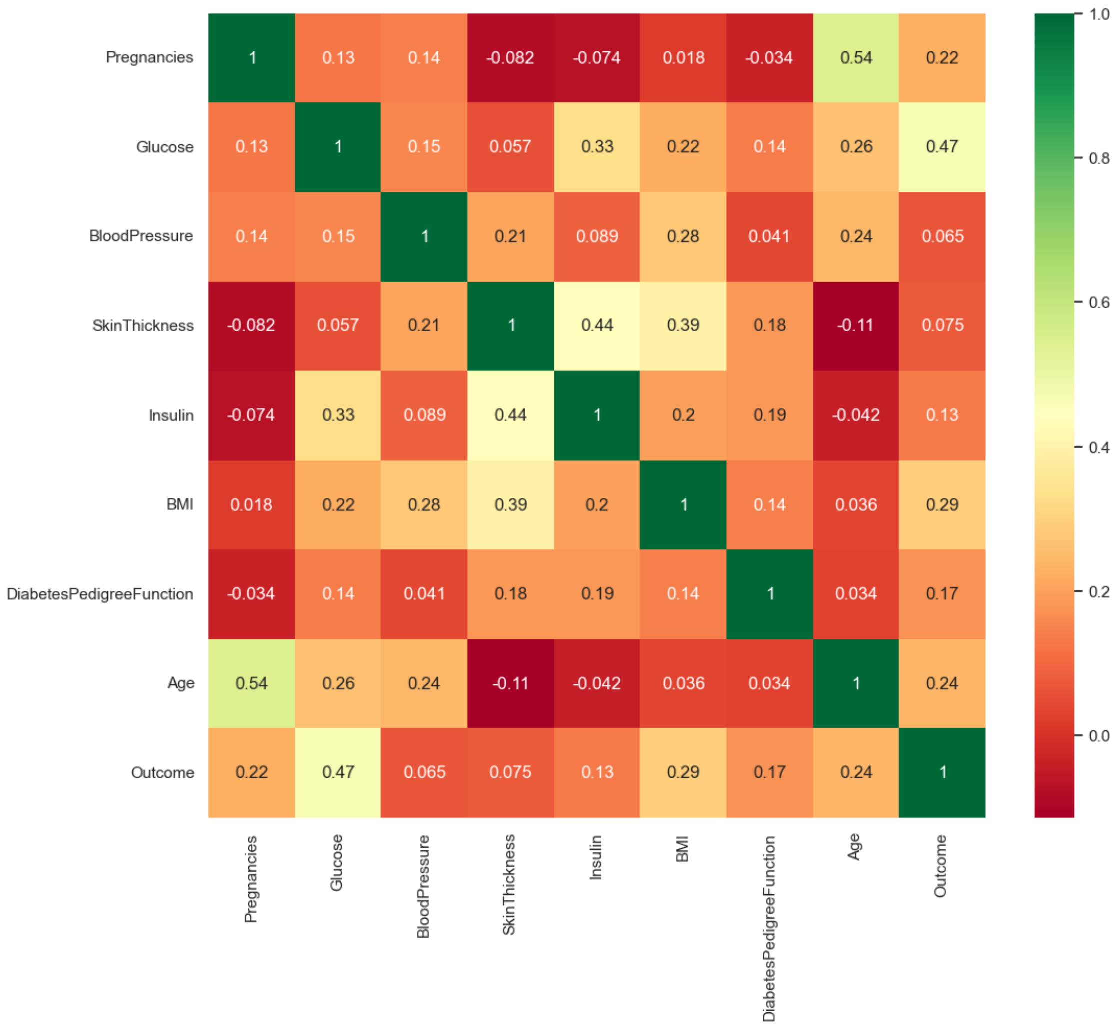


Pearson's Correlation Coefficient: helps you find out the relationship between two quantities. It gives you the measure of the strength of association between two variables. The value of Pearson's Correlation Coefficient can be between -1 to +1. 1 means that they are highly correlated, 0 means no correlation and -1 means negetively correlated. i.e 1 = directly proportional / 0 = no correlation / -1 = inversely proportional.

A heat map is a two-dimensional representation of information with the help of colors. Heat maps can help the user visualize simple or complex information

Heat map for unclean data

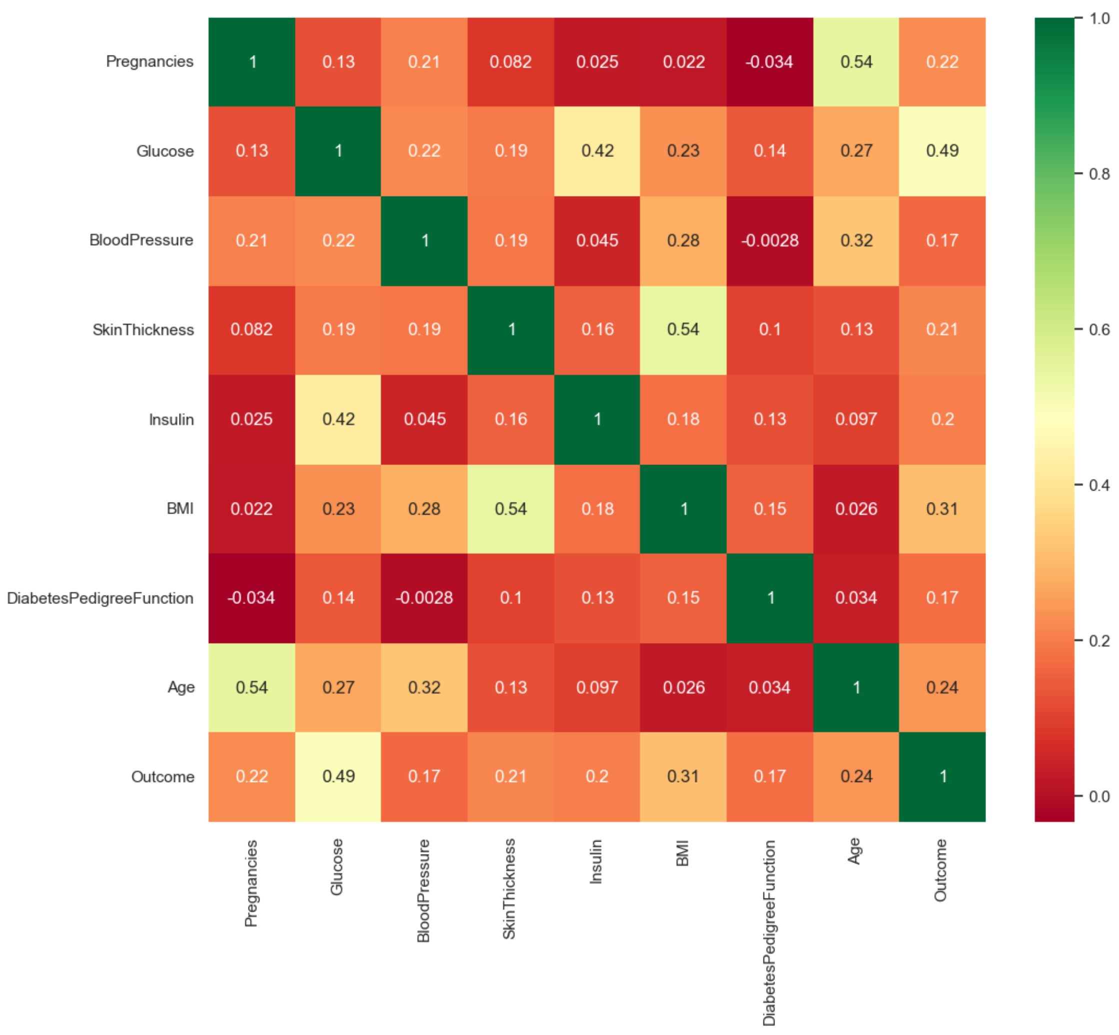
[15]:
```
plt.figure(figsize=(12,10))
p=sns.heatmap(diabetes_data.corr(), annot=True,cmap ='RdYlGn')
```



| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| Pregnancies | 1 | 0.13 | 0.14 | -0.082 | -0.074 | 0.018 | -0.034 | 0.54 | 0.22 |
| Glucose | 0.13 | 1 | 0.15 | 0.057 | 0.33 | 0.22 | 0.14 | 0.26 | 0.47 |
| BloodPressure | 0.14 | 0.15 | 1 | 0.21 | 0.089 | 0.28 | 0.041 | 0.24 | 0.065 |
| SkinThickness | -0.082 | 0.057 | 0.21 | 1 | 0.44 | 0.39 | 0.18 | -0.11 | 0.075 |
| Insulin | -0.074 | 0.33 | 0.089 | 0.44 | 1 | 0.2 | 0.19 | -0.042 | 0.13 |
| BMI | 0.018 | 0.22 | 0.28 | 0.39 | 0.2 | 1 | 0.14 | 0.036 | 0.29 |
| DiabetesPedigreeFunction | -0.034 | 0.14 | 0.041 | 0.18 | 0.19 | 0.14 | 1 | 0.034 | 0.17 |
| Age | 0.54 | 0.26 | 0.24 | -0.11 | -0.042 | 0.036 | 0.034 | 1 | 0.24 |
| Outcome | 0.22 | 0.47 | 0.065 | 0.075 | 0.13 | 0.29 | 0.17 | 0.24 | 1 |

Heat map for clean data.

[16]:
```
plt.figure(figsize=(12,10))
p=sns.heatmap(diabetes_data_copy.corr(), annot=True,cmap ='RdYlGn')
```



| | Pregnancies | Glucose | BloodPressure | SkinThickness | Insulin | BMI | DiabetesPedigreeFunction | Age | Outcome |
|---|---|---|---|---|---|---|---|---|---|
| Pregnancies | 1 | 0.13 | 0.21 | 0.082 | 0.025 | 0.022 | -0.034 | 0.54 | 0.22 |
| Glucose | 0.13 | 1 | 0.22 | 0.19 | 0.42 | 0.23 | 0.14 | 0.27 | 0.49 |
| BloodPressure | 0.21 | 0.22 | 1 | 0.19 | 0.045 | 0.28 | -0.0028 | 0.32 | 0.17 |
| SkinThickness | 0.082 | 0.19 | 0.19 | 1 | 0.16 | 0.54 | 0.1 | 0.13 | 0.21 |
| Insulin | 0.025 | 0.42 | 0.045 | 0.16 | 1 | 0.18 | 0.13 | 0.097 | 0.2 |
| BMI | 0.022 | 0.23 | 0.28 | 0.54 | 0.18 | 1 | 0.15 | 0.026 | 0.31 |
| DiabetesPedigreeFunction | -0.034 | 0.14 | -0.0028 | 0.1 | 0.13 | 0.15 | 1 | 0.034 | 0.17 |
| Age | 0.54 | 0.27 | 0.32 | 0.13 | 0.097 | 0.026 | 0.034 | 1 | 0.24 |
| Outcome | 0.22 | 0.49 | 0.17 | 0.21 | 0.2 | 0.31 | 0.17 | 0.24 | 1 |

## 1.8 Scalling Data

So if the data in any conditions has data points far from each other, scaling is a technique to make them closer to each other or in simpler words, we can say that the scaling is used for making data points generalized so that the distance between them will be lower. Scalling is always advisable to bring all the features to the same scale for applying distance based algorithms like KNN.

[17]:
```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X =  pd.DataFrame(sc_X.fit_transform(diabetes_data_copy.drop(["Outcome"],axis =␣
 ↪1),),
        columns=['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',␣
 ↪'Insulin',
       'BMI', 'DiabetesPedigreeFunction', 'Age'])
```

[18]:
```
X.head()
```

[18]:
```
   Pregnancies   Glucose  BloodPressure  SkinThickness   Insulin       BMI  \
0     0.639947  0.865108      -0.033518       0.670643 -0.181541  0.166619
1    -0.844885 -1.206162      -0.529859      -0.012301 -0.181541 -0.852200
2     1.233880  2.015813      -0.695306      -0.012301 -0.181541 -1.332500
3    -0.844885 -1.074652      -0.529859      -0.695245 -0.540642 -0.633881
4    -1.141852  0.503458      -2.680669       0.670643  0.316566  1.549303

   DiabetesPedigreeFunction       Age
0                  0.468492  1.425995
1                 -0.365061 -0.190672
2                  0.604397 -0.105584
3                 -0.920763 -1.041549
4                  5.484909 -0.020496
```

[19]:
```
y = diabetes_data_copy.Outcome
```

## 1.9 Test Train Split and Cross Validation methods

- Train Test Split : To have unknown datapoints to test the data rather than testing with the same points with which the model was trained. This helps capture the model performance much better.
- Cross Validation: When model is split into training and testing it can be possible that specific type of data point may go entirely into either training or testing portion. This would lead the model to perform poorly. Hence over-fitting and underfitting problems can be well avoided with cross validation techniques
- About Stratify : Stratify parameter makes a split so that the proportion of values in the sample produced will be the same as the proportion of values provided to parameter stratify.
- For example, if variable y is a binary categorical variable with values 0 and 1 and there are 25% of zeros and 75% of ones, stratify=y will make sure that your random split has 25% of 0's and 75% of 1's.

[20]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=1/
 ↪3,random_state=42, stratify=y)
```

[21]:
```
from sklearn.neighbors import KNeighborsClassifier


test_scores = []
train_scores = []

for i in range(1,15):

    knn = KNeighborsClassifier(i)
    knn.fit(X_train,y_train)

    train_scores.append(knn.score(X_train,y_train))
    test_scores.append(knn.score(X_test,y_test))
```

Here the score represents the accuracy of the model and k value is the number of nearest values taken into concideration to get the accurate value. For ex, k = 11 means the model needs 11 nearest values to get the highest accuracy.

[22]:
```
max_train_score = max(train_scores)
train_scores_ind = [i for i, v in enumerate(train_scores) if v ==␣
 ↪max_train_score]
print('Max train score {} % and k = {}'.
 ↪format(max_train_score*100,list(map(lambda x: x+1, train_scores_ind))))
```

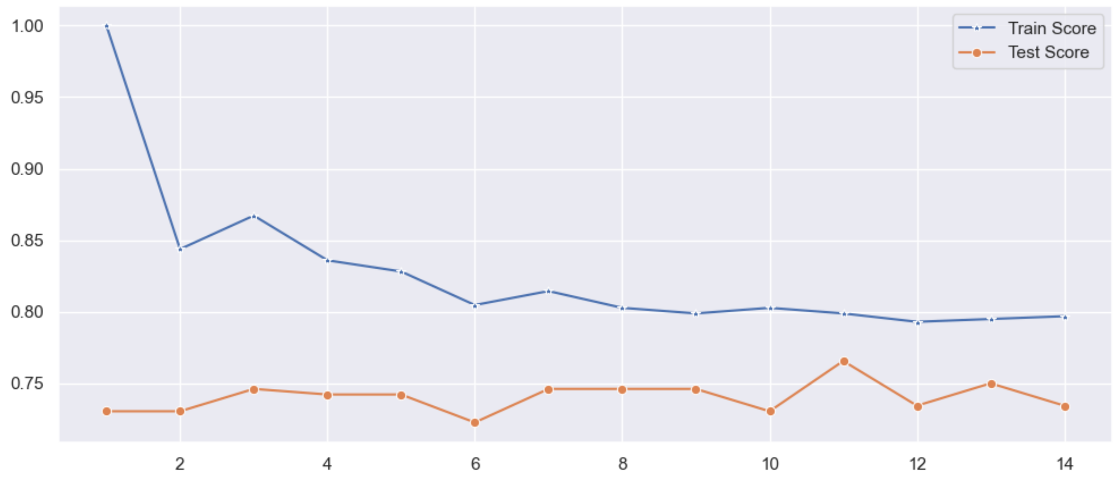
```
Max train score 100.0 % and k = [1]
```

[23]:
```
max_test_score = max(test_scores)
test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_score]
print('Max test score {} % and k = {}'.
 ↪format(max_test_score*100,list(map(lambda x: x+1, test_scores_ind))))
```

```
Max test score 76.5625 % and k = [11]
```

### 1.10 Result Visualisation

[24]:
```
plt.figure(figsize=(12,5))
p = sns.lineplot(range(1,15),train_scores,marker='*',label='Train Score')
p = sns.lineplot(range(1,15),test_scores,marker='o',label='Test Score')
```

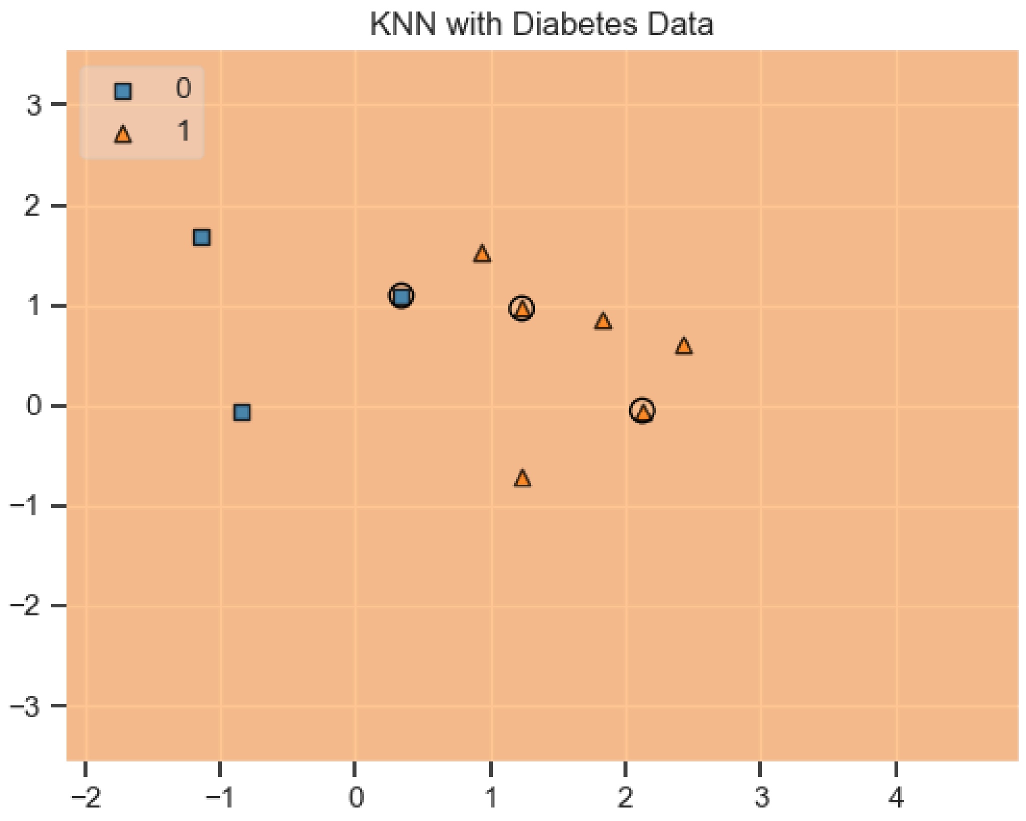


The best result is captured at k = 11 hence 11 is used for the final model

[25]:
```
knn = KNeighborsClassifier(11)

knn.fit(X_train,y_train)
knn.score(X_test,y_test)
```

[25]:
```
0.765625
```

[26]:
```
value = 20000
width = 20000
plot_decision_regions(X.values, y.values, clf=knn, legend=2,
                      filler_feature_values={2: value, 3: value, 4: value, 5:␣
 ↪value, 6: value, 7: value},
                      filler_feature_ranges={2: width, 3: width, 4: width, 5:␣
 ↪width, 6: width, 7: width},
                      X_highlight=X_test.values)
plt.title('KNN with Diabetes Data')
plt.show()
```



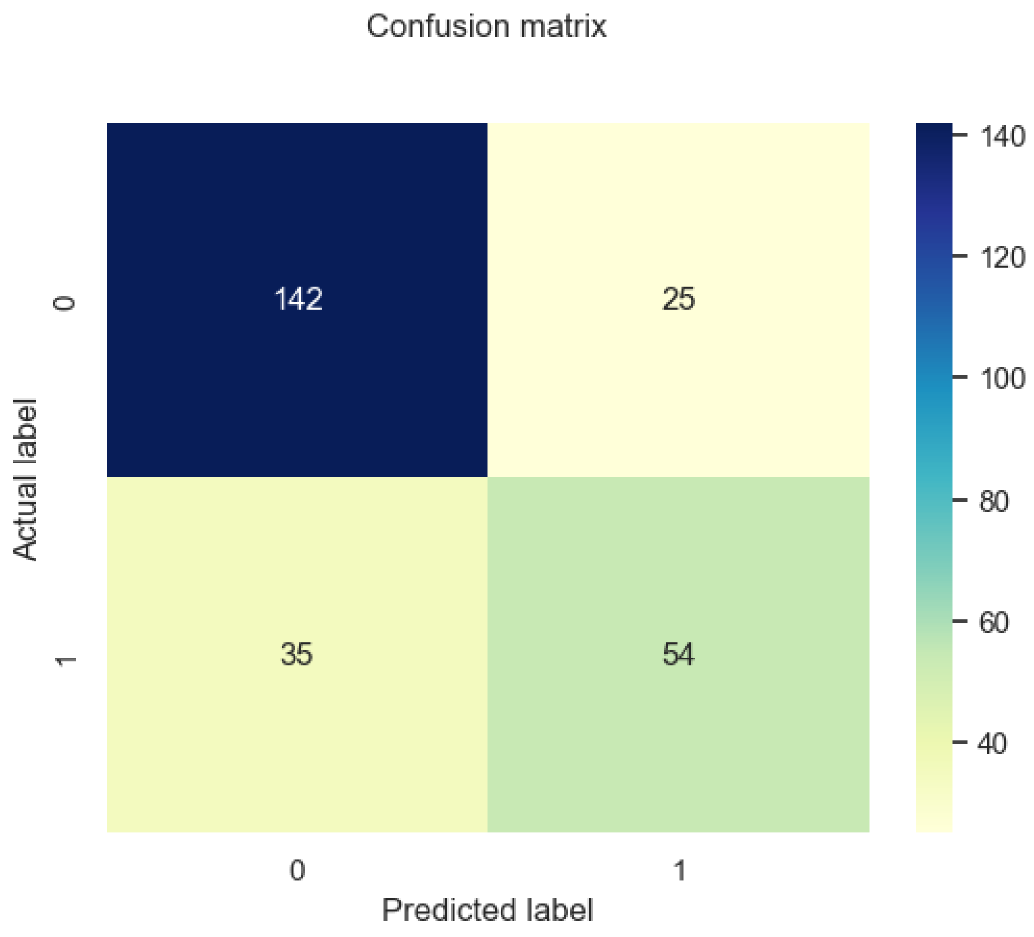
[27]:
```
from sklearn.metrics import confusion_matrix
y_pred = knn.predict(X_test)
confusion_matrix(y_test,y_pred)
```

```
pd.crosstab(y_test, y_pred, rownames=['True'], colnames=['Predicted'],␣
  ↪margins=True)
```

```
[27]: Predicted    0   1  All
      True
      0          142  25  167
      1           35  54   89
      All        177  79  256
```

```
[28]: y_pred = knn.predict(X_test)
      from sklearn import metrics
      cnf_matrix = metrics.confusion_matrix(y_test, y_pred)
      p = sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
      plt.title('Confusion matrix', y=1.1)
      plt.ylabel('Actual label')
      plt.xlabel('Predicted label')
```

```
[28]: Text(0.5, 20.049999999999997, 'Predicted label')
```



## 1.11 Classification Report

### 1.11.1 Precision Score

- TP – True Positives
- FP – False Positives
- Precision – Accuracy of positive predictions.
- Precision = TP/(TP + FP)

### 1.11.2 Recall Score

- Recall(sensitivity or true positive rate): Fraction of positives that were correctly identified.
- Recall = TP/(TP+FN)

### 1.11.3 F1 Score

- F1 Score (aka F-Score or F-Measure) – A helpful metric for comparing two classifiers.
- F1 Score takes into account precision and the recall.
- It is created by finding the the harmonic mean of precision and recall.
- F1 = 2 x (precision x recall)/(precision + recall)

- **Precision** - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passengers that labeled as survived, how many actually survived? High precision relates to the low false positive rate. We have got 0.788 precision which is pretty good.

- Precision = TP/TP+FP

- **Recall** (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes. The question recall answers is: Of all the passengers that truly survived, how many did we label? A recall greater than 0.5 is good.

- Recall = TP/TP+FN

- **F1 score** - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it's better to look at both Precision and Recall.

- F1 Score = 2(Recall Precision) / (Recall + Precision)

[29]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

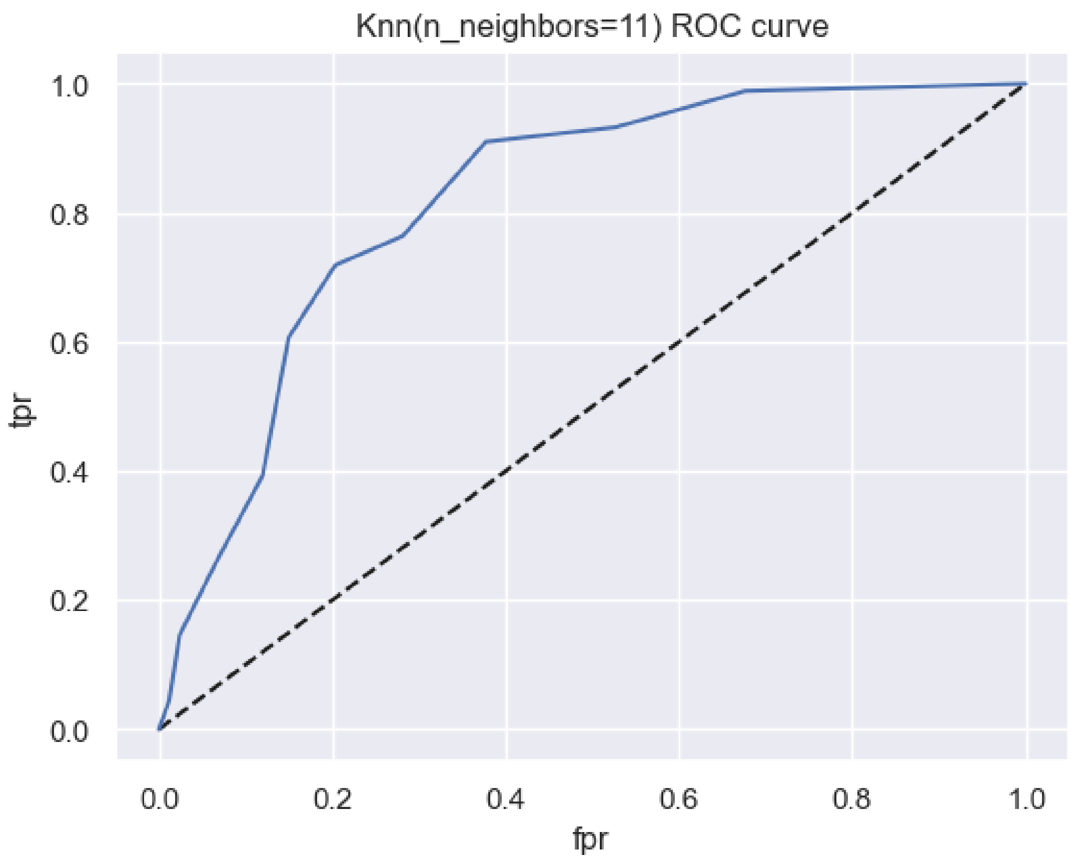
```
              precision    recall  f1-score   support

           0       0.80      0.85      0.83       167
           1       0.68      0.61      0.64        89

    accuracy                           0.77       256
   macro avg       0.74      0.73      0.73       256
```

```
weighted avg       0.76      0.77      0.76       256
```

### 1.12 ROC - AUC

ROC (Receiver Operating Characteristic) Curve tells us about how good the model can distinguish between two things (e.g If a patient has a disease or no). Better models can accurately distinguish between the two. Whereas, a poor model will have difficulties in distinguishing between the two

[30]:
```
from sklearn.metrics import roc_curve
y_pred_proba = knn.predict_proba(X_test)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
plt.plot([0,1],[0,1],'k--')
plt.plot(fpr,tpr, label='Knn')
plt.xlabel('fpr')
plt.ylabel('tpr')
plt.title('Knn(n_neighbors=11) ROC curve')
plt.show()
```



```
[31]: from sklearn.metrics import roc_auc_score
      roc_auc_score(y_test,y_pred_proba)
```

```
[31]: 0.8193500639171096
```

## 1.13 Hyper Parameter optimization

- Grid search is an approach to hyperparameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.

Let's consider the following example:

- Suppose, a machine learning model X takes hyperparameters a1, a2 and a3. In grid searching, you first define the range of values for each of the hyperparameters a1, a2 and a3. You can think of this as an array of values for each of the hyperparameters. Now the grid search technique will construct many versions of X with all the possible combinations of hyperparameter (a1, a2 and a3) values that you defined in the first place. This range of hyperparameter values is referred to as the grid.
- Suppose, you defined the grid as: a1 = [0,1,2,3,4,5] a2 = [10,20,30,40,5,60] a3 = [105,105,110,115,120,125]

Note that, the array of values of that you are defining for the hyperparameters has to be legitimate in a sense that you cannot supply Floating type values to the array if the hyperparameter only takes Integer values.

Now, grid search will begin its process of constructing several versions of X with the grid that you just defined.

It will start with the combination of [0,10,105], and it will end with [5,60,125]. It will go through all the intermediate combinations between these two which makes grid search computationally very expensive.

```
[32]: from sklearn.model_selection import GridSearchCV
      param_grid = {'n_neighbors':np.arange(1,50)}
      knn = KNeighborsClassifier()
      knn_cv= GridSearchCV(knn,param_grid,cv=5)
      knn_cv.fit(X,y)
      print("Best Score:" + str(knn_cv.best_score_))
      print("Best Parameters: " + str(knn_cv.best_params_))
```

```
Best Score:0.7721840251252015
Best Parameters: {'n_neighbors': 25}
```

- The **param_grid** variable specifies a dictionary of hyperparameters and their corresponding values that should be explored during the grid search. In this case, **n_neighbors** is the only hyperparameter being tuned, and it is set to take on integer values between 1 and 49 inclusive.
- The **GridSearchCV** function then creates a **knn_cv** object that performs a **5-fold cross-validation grid** search over the hyperparameters specified in **param_grid** using the KNeighborsClassifier model.

- The fit method is then called on the **knn_cv** object to perform the grid search and select the best hyperparameters based on the highest cross-validation score.
- Finally, the best_score_ attribute of the **knn_cv** object is used to print the best cross-validation score achieved during the grid search, and the **best_params_ attribute** is used to print the hyperparameters that resulted in the best score.
- Note that X and y in the **knn_cv.fit(X,y)** line represent the feature matrix and target vector, respectively, of the training data used to fit the model.