```
### Description

### Attributes
- A: Number of pragnancies
- B: Concentration of plasma glucose in a 2 hour oral glucose tolerance test
- C: Measured in mmHg
- D: Measured in mm
- E: Insulin concentration in serum in 2-hour period. Measured in (mu U/ml)
- F: Weight in kg/height in (m^2)
- G: Function that assigns probability of someone getting diabetes
- H: Age
- Target: Value of 0 or 1 correspond to no diabetes and diabetes
```

In [50]:

```python
import numpy as np     # for performance on computation
import pandas as pd    #for reading data into Data Frame


from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split   # for train and test rand
from sklearn.preprocessing import StandardScaler #standardizes a feature by subt
# Suppress warnings
import warnings
warnings.filterwarnings('ignore')

# Visualization
import matplotlib.pyplot as plt     # for visualization

import seaborn as sns     # for visualization
plt.style.use('seaborn-whitegrid')
```

In [44]:

```python
# checking present working directory
pwd
```

Out[44]: 'C:\\Users\\varun\\EXAM 1 AAML 6 26 2020'

In [51]:

```python
#Loading the train dataset
diabet_data = pd.read_csv('train.csv')

#Print the first 5 rows of the dataframe.
diabet_data.head()
```

Out[51]:

	A	B	C	D	E	F	G	H	Target
0	5	122	86	NaN	NaN	34.7	0.290	33	0
1	2	175	88	NaN	NaN	22.9	0.326	22	0
2	4	129	86	2.0	27.0	35.1	0.231	23	0
3	12	92	62	7.0	258.0	27.6	0.926	44	1
4	3	102	44	2.0	94.0	3.8	0.400	26	0

In [52]:
```
#rename columns in Train Data a more metadata friendly
diabetes_data = diabet_data.rename(columns = {'A':'Pregnancies','B':'Glucose','C
                                              'F':'BMI',
                                             'G':'PROBABLITY','H':'Age'})

#Print the first 5 rows of the dataframe with proper meta data column names.
diabetes_data.head()
```

Out[52]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	PROBABLITY	Age	Target
0	5	122	86	NaN	NaN	34.7	0.290	33	0
1	2	175	88	NaN	NaN	22.9	0.326	22	0
2	4	129	86	2.0	27.0	35.1	0.231	23	0
3	12	92	62	7.0	258.0	27.6	0.926	44	1
4	3	102	44	2.0	94.0	3.8	0.400	26	0

In [53]:
```
#Loading the test dataset
test_data_orig = pd.read_csv('test.csv')


#Print the first 5 rows of the dataframe.
test_data_orig.head()
```

Out[53]:

	A	B	C	D	E	F	G	H
0	6	148	72	35.0	NaN	33.6	0.627	50
1	1	85	66	29.0	NaN	26.6	0.351	31
2	5	116	74	NaN	NaN	25.6	0.201	30
3	4	110	92	NaN	NaN	37.6	0.191	30
4	5	166	72	19.0	175.0	25.8	0.587	51

In [160]:
```
# Renaming columns of test dataset to make it consistent with training dataset lo

test_data = test_data_orig.rename(columns = {'A':'Pregnancies','B':'Glucose','C'
                                              'F':'BMI',
                                             'G':'PROBABLITY','H':'Age'})
#Print the first 5 rows of the dataframe.
test_data.head()
```

Out[160]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	PROBABLITY	Age
0	6	148	72	35.0	NaN	33.6	0.627	50
1	1	85	66	29.0	NaN	26.6	0.351	31
2	5	116	74	NaN	NaN	25.6	0.201	30
3	4	110	92	NaN	NaN	37.6	0.191	30
4	5	166	72	19.0	175.0	25.8	0.587	51

In [55]:
```
test_data.columns     # Noticing Target variable already dropped from test data, 
                      # for further testing
```

Out[55]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin',
       'BMI', 'PROBABLITY', 'Age'],
      dtype='object')
```

In [56]:
```
test_data.shape    # checking the size of test data using shape function
```

Out[56]:
```
(268, 8)
```

Exploratory Data analysis

In [57]:
```
# Getting information about the total number of records, data types,columns, NON
diabetes_data.info(verbose=True)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
Pregnancies      500 non-null int64
Glucose          500 non-null int64
BloodPressure    500 non-null int64
SkinThickness    360 non-null float64
Insulin          253 non-null float64
BMI              492 non-null float64
PROBABLITY       500 non-null float64
Age              500 non-null int64
Target           500 non-null int64
dtypes: float64(4), int64(5)
memory usage: 35.2 KB
```

All attributes in the diabetest dataset are either Integer or Float.

Target is binary classification 0 (No Diabetes) or 1 (Diabetes).

So the problem we are trying to predict is a supervissed classification problem, since target is binary - we can finalize ML model with one of the following classification algorithms

Logistic regression, KNN, Decission Tree, Random Forest and SVM

In [58]:
```
#Total number of training records using shape function
# len(diabetes_data)  we can sue lenth method for total no. of records
print("Total number of records: ", diabetes_data.shape[0])
print ("No. of Columns including Target", diabetes_data.shape[1] )
```

```
Total number of records:  500
No. of Columns including Target 9
```

In [59]:
```
# describe method for generating statistics to summarize the central tendency, d
#excluding NaN values. Using transpose on describe method for changing its defau
#Note: It excudes Object type from stats
diabetes_data.describe().T
```

Out[59]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	500.0	3.876000	3.394653	0.000	1.000	3.000	6.00000	17.00
Glucose	500.0	121.470000	32.738735	0.000	99.000	116.000	143.00000	199.00
BloodPressure	500.0	68.666000	20.288067	0.000	62.000	70.000	80.00000	122.00
SkinThickness	360.0	26.308333	13.120056	1.000	18.000	27.000	35.00000	99.00
Insulin	253.0	106.332016	122.448436	1.000	21.000	67.000	145.00000	846.00
BMI	492.0	29.932724	10.624439	2.100	25.375	32.000	36.02500	59.40
PROBABLITY	500.0	0.472286	0.341394	0.078	0.240	0.378	0.61225	2.42
Age	500.0	33.270000	11.890663	21.000	24.000	29.000	41.00000	72.00
Target	500.0	0.324000	0.468469	0.000	0.000	0.000	1.00000	1.00

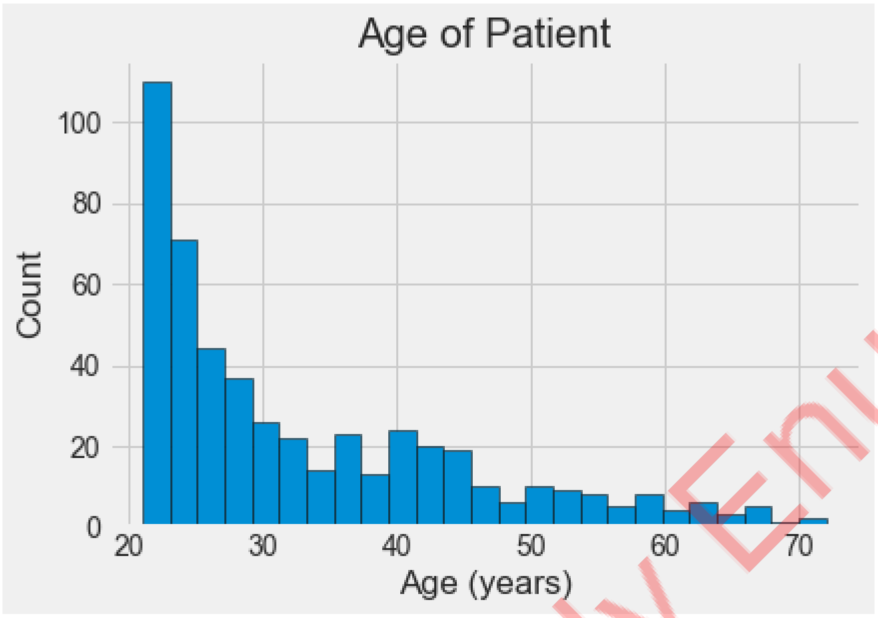
In [60]:
```
# getting stats on categorical variable using describe function with include=['O
#diabetes_data.describe(include=['O'])  #No objects(categorical) found in this d
```

In [61]:
```
#CHECKING GIVEN TEST DATA TO MAKE SURE TRAIN AND TEST HAS SAME NUMBER OF PREDICT
diabetes_data.columns.values,test_data.columns.values
# quick look to check target variable dropped or not to make sure no data leakage
```

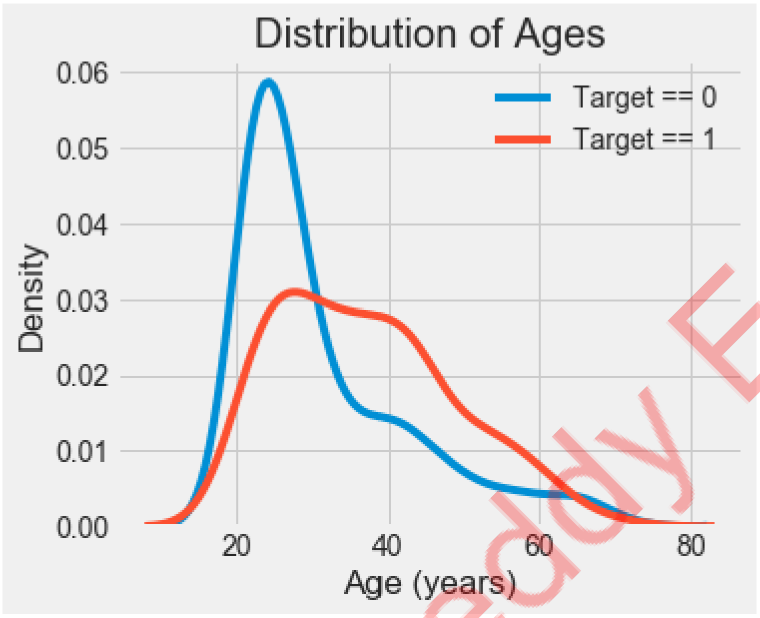
Out[61]:
```
(array(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
        'Insulin', 'BMI', 'PROBABLITY', 'Age', 'Target'], dtype=object),
 array(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
        'Insulin', 'BMI', 'PROBABLITY', 'Age'], dtype=object))
```

In [62]:
```
# Plot the distribution of ages in years
```

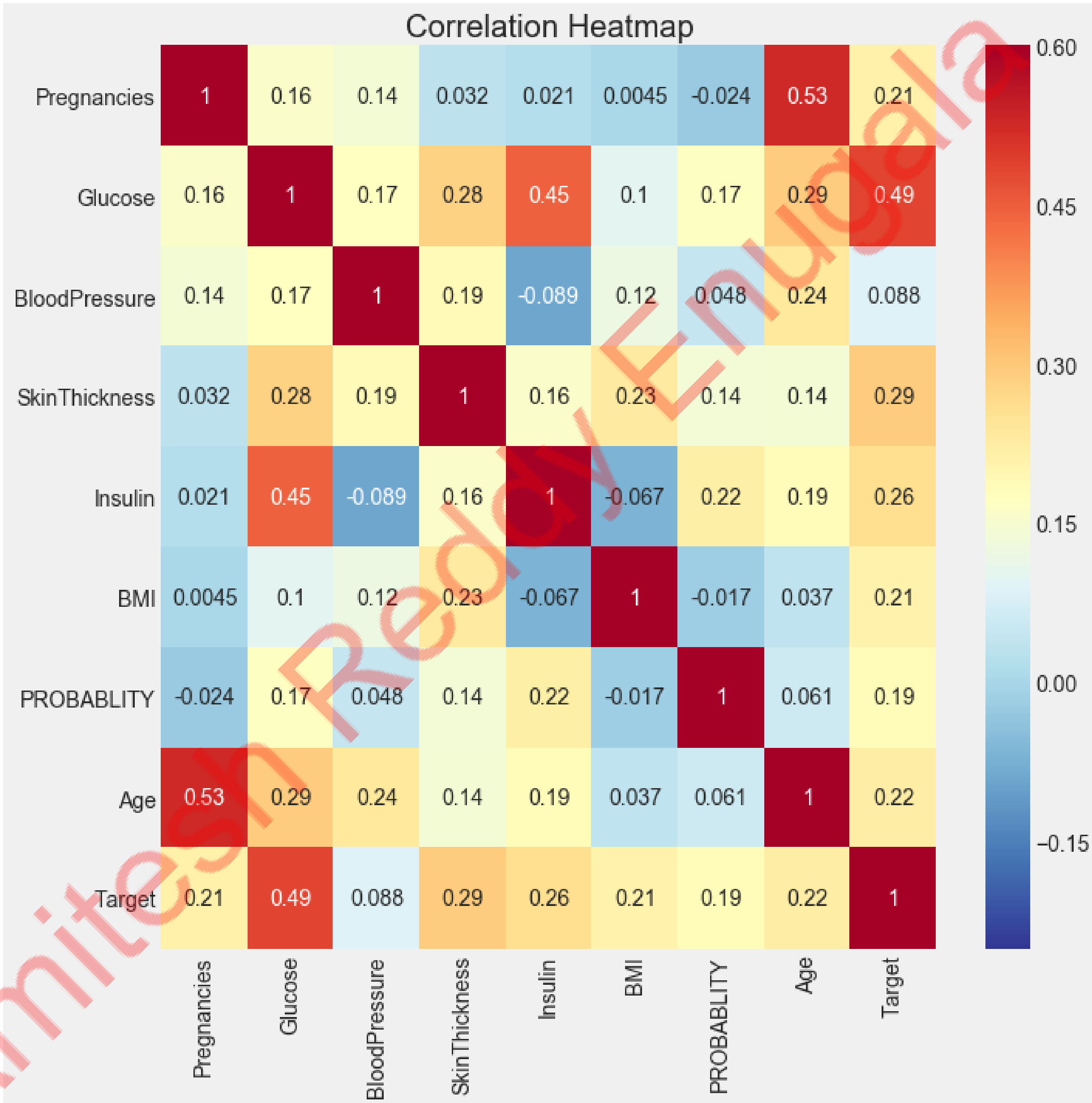
In [63]:
```
# Set the style of plots
plt.style.use('fivethirtyeight')

# Plot the distribution of ages in years
plt.hist(diabetes_data['Age'], edgecolor = 'k', bins = 25)
plt.title('Age of Patient'); plt.xlabel('Age (years)'); plt.ylabel('Count');
```



In [64]:
```
# Checking Age corelation against Target (No Diabetes/ Diabetes)
plt.figure(figsize = (5, 4))

# KDE plot for the persons showing No Diabetes
sns.kdeplot(diabetes_data.loc[diabetes_data['Target'] == 0, 'Age'], label = 'Tar

# KDE plot for the Patients has Diabetes
sns.kdeplot(diabetes_data.loc[diabetes_data['Target'] == 1, 'Age'], label = 'Tar

# Labeling of plot
plt.xlabel('Age (years)'); plt.ylabel('Density'); plt.title('Distribution of Age
```



In [65]:
```
# Heatmap of correlations
ext_data = diabetes_data[['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickn
        'Insulin', 'BMI', 'PROBABLITY', 'Age', 'Target']]
ext_data_corrs = ext_data.corr()
ext_data_corrs

plt.figure(figsize = (10, 10))
sns.heatmap(ext_data_corrs, cmap = plt.cm.RdYlBu_r, vmin = -0.25, annot = True, 
plt.title('Correlation Heatmap');
```



In [66]:
```
# Test dataset info given for exam
test_data.info()
                    # values across the predictors.
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 268 entries, 0 to 267
Data columns (total 8 columns):
Pregnancies      268 non-null int64
Glucose          268 non-null int64
BloodPressure    268 non-null int64
SkinThickness    181 non-null float64
Insulin          141 non-null float64
BMI              265 non-null float64
PROBABLITY       268 non-null float64
Age              268 non-null int64
dtypes: float64(4), int64(4)
memory usage: 16.8 KB
```

In [67]:
```
len(test_data)
```

Out[67]:
```
268
```

In [68]:
```
# We notice test dataset also has lot of missing values for Insulin and SkinThick
```

Deep Analysis on Training Dataset

In [69]:
```
# Data exploration on Training Dataset with Transpose function

diabetes_data.describe().T

#check minimum  value of below listed columns be zero (0) and high values to chec
# to high value for not available number (NAN) since all predictors in the train
```

Out[69]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	500.0	3.876000	3.394653	0.000	1.000	3.000	6.00000	17.00
Glucose	500.0	121.470000	32.738735	0.000	99.000	116.000	143.00000	199.00
BloodPressure	500.0	68.666000	20.288067	0.000	62.000	70.000	80.00000	122.00
SkinThickness	360.0	26.308333	13.120056	1.000	18.000	27.000	35.00000	99.00
Insulin	253.0	106.332016	122.448436	1.000	21.000	67.000	145.00000	846.00
BMI	492.0	29.932724	10.624439	2.100	25.375	32.000	36.02500	59.40
PROBABLITY	500.0	0.472286	0.341394	0.078	0.240	0.378	0.61225	2.42
Age	500.0	33.270000	11.890663	21.000	24.000	29.000	41.00000	72.00
Target	500.0	0.324000	0.468469	0.000	0.000	0.000	1.00000	1.00

In [70]:
```
# From above descriptive statistics, for the columns Glucose and BloodPressure, c
# does not make sense and  thus indicates zeros in these two columns represent mi

print('The min value of Glucose predictor: ', diabetes_data.Glucose.min())
print('The min value of BloodPressure predictor: ', diabetes_data.BloodPressure.m

#Following columns or variables have an invalid zero value as it doesn't make sen
# having Glucose and BloodPressue values as zero, so we need to make then as Not

#Glucose
#BloodPressure
```

```
The min value of Glucose predictor:  0
The min value of BloodPressure predictor:  0
```

DATA IMPUTATION:

It is better to replace zeros with np.NaN since after that counting them would be easier and then impute missing values with suitable values

In [91]:
```
Glucose_zero_count = (diabetes_data['Glucose'] == 0).sum()

print("The number records with  Glucose as Zero: " , Glucose_zero_count)

print("The median for Glucose from training dataset: ",diabetes_data.Glucose.med
print("The mean for Glucose from training dataset: ",diabetes_data.Glucose.mean(
```

```
The number records with  Glucose as Zero:  3
The median for Glucose from training dataset:  116.0
The mean for Glucose from training dataset:  121.47
```

In [92]:
```
BloodPressure_zero_count = (diabetes_data['BloodPressure'] == 0).sum()

print("The number records with  BloodPressure as Zero: " , BloodPressure_zero_co

print("The median for BloodPressure from training dataset: ",diabetes_data.Blood
print("The mean for BloodPressure from training dataset: ",diabetes_data.BloodPr
```

```
The number records with  BloodPressure as Zero:  26
The median for BloodPressure from training dataset:  70.0
The mean for BloodPressure from training dataset:  68.666
```

In [93]:
```
#The copy () method in Python returns a copy of the Set.
#We can copy a set to another set using the = operator, however copying a set us
diabetes_data_set_impute = diabetes_data.copy(deep = True)    # using deepcopy f
```

In [94]:
```
diabetes_data_set_impute[['Glucose','BloodPressure']] = diabetes_data_set_impute
```

In [95]:
```
diabetes_data_set_impute.info()   #Showing now latest counts after imputation
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
Pregnancies      500 non-null int64
Glucose          497 non-null float64
BloodPressure    474 non-null float64
SkinThickness    360 non-null float64
Insulin          253 non-null float64
BMI              492 non-null float64
PROBABLITY       500 non-null float64
Age              500 non-null int64
Target           500 non-null int64
dtypes: float64(6), int64(3)
memory usage: 35.2 KB
```

In [96]:
```
# Now we can see from above results, the Glucose has three and BloodPressure clo
```

In [97]:
```
## showing the latest count of Nans
print(diabetes_data_set_impute.isnull().sum())
```

```
Pregnancies        0
Glucose            3
BloodPressure     26
SkinThickness    140
Insulin          247
BMI                8
PROBABLITY         0
Age                0
Target             0
dtype: int64
```

We can see that columns Gloucose,BloodPressure and BMI have just a few NaN (missing) values, whereas columns SkinThickness and Insulin columns are showing a lot more missing values, nearly half of the total training dataset (500 total)

We need to use different data imputing methods for addressing “missing value” for different columns for making sure that there are still a sufficient number of records left to train a predictive model.

1) Check under each column percentage of missing values, if few impute with mean, median,Forward/Backward Propoagation etc

2) How much percentage of total number of columns missing across each record, if more than three predictors, drop those records since in this case we have total eight predictors

Impute Missing Values

In [98]:
```
#Use aggregate function for replacing missing values for the columns Glucose, Blo
# few missing values
diabetes_data_set_impute['Glucose'].fillna(diabetes_data_set_impute['Glucose'].me
diabetes_data_set_impute['BloodPressure'].fillna(diabetes_data_set_impute['BloodP
diabetes_data_set_impute['BMI'].fillna(diabetes_data_set_impute['BMI'].median(),i
```

In [99]:
```
diabetes_data_set_impute.describe().T
# Notice for 'Glucose'and 'BloodPressure' min values are not showing zeros since
# also stats changed now for these and for BMI
```

Out[99]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	500.0	3.876000	3.394653	0.000	1.00	3.000	6.00000	17.00
Glucose	500.0	122.166000	31.349834	57.000	99.00	116.000	143.00000	199.00
BloodPressure	500.0	72.410000	12.347746	30.000	64.00	72.000	80.00000	122.00
SkinThickness	360.0	26.308333	13.120056	1.000	18.00	27.000	35.00000	99.00
Insulin	253.0	106.332016	122.448436	1.000	21.00	67.000	145.00000	846.00
BMI	500.0	29.965800	10.542127	2.100	25.55	32.000	35.90000	59.40
PROBABLITY	500.0	0.472286	0.341394	0.078	0.24	0.378	0.61225	2.42
Age	500.0	33.270000	11.890663	21.000	24.00	29.000	41.00000	72.00
Target	500.0	0.324000	0.468469	0.000	0.00	0.000	1.00000	1.00

In [100]:
```
diabetes_data_set_impute.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 500 entries, 0 to 499
Data columns (total 9 columns):
Pregnancies      500 non-null int64
Glucose          500 non-null float64
BloodPressure    500 non-null float64
SkinThickness    360 non-null float64
Insulin          253 non-null float64
BMI              500 non-null float64
PROBABLITY       500 non-null float64
Age              500 non-null int64
Target           500 non-null int64
dtypes: float64(6), int64(3)
memory usage: 35.2 KB
```

In [101]:
```
# TWO MORE Columns Needs to be imputed - SkinThickness and Insulin.

## showing the count of Nans
print(diabetes_data_set_impute.isnull().sum())
```

```
Pregnancies        0
Glucose            0
BloodPressure      0
SkinThickness    140
Insulin          247
BMI                0
PROBABLITY         0
Age                0
Target             0
dtype: int64
```

In [102]:
```
#Percentage of missing values in each column
print((diabetes_data_set_impute.isnull().sum() / diabetes_data_set_impute.shape[(

# We can use mean method to achieve same with the following way
# print((diabetes_data_set.isnull().mean()) * 100.00)
```

```
Pregnancies       0.0
Glucose           0.0
BloodPressure     0.0
SkinThickness    28.0
Insulin          49.4
BMI               0.0
PROBABLITY        0.0
Age               0.0
Target            0.0
dtype: float64
```

In [103]:
```
# as Insulin has nearly 50% of missing values we can drop it
diabetes_data_set_imputed= diabetes_data_set_impute.drop(columns=['Insulin'])
```

In [104]:
```
diabetes_data_set_imputed.columns    # Insulin Column should not be there becaus(
```

Out[104]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'BMI',
       'PROBABLITY', 'Age', 'Target'],
      dtype='object')
```

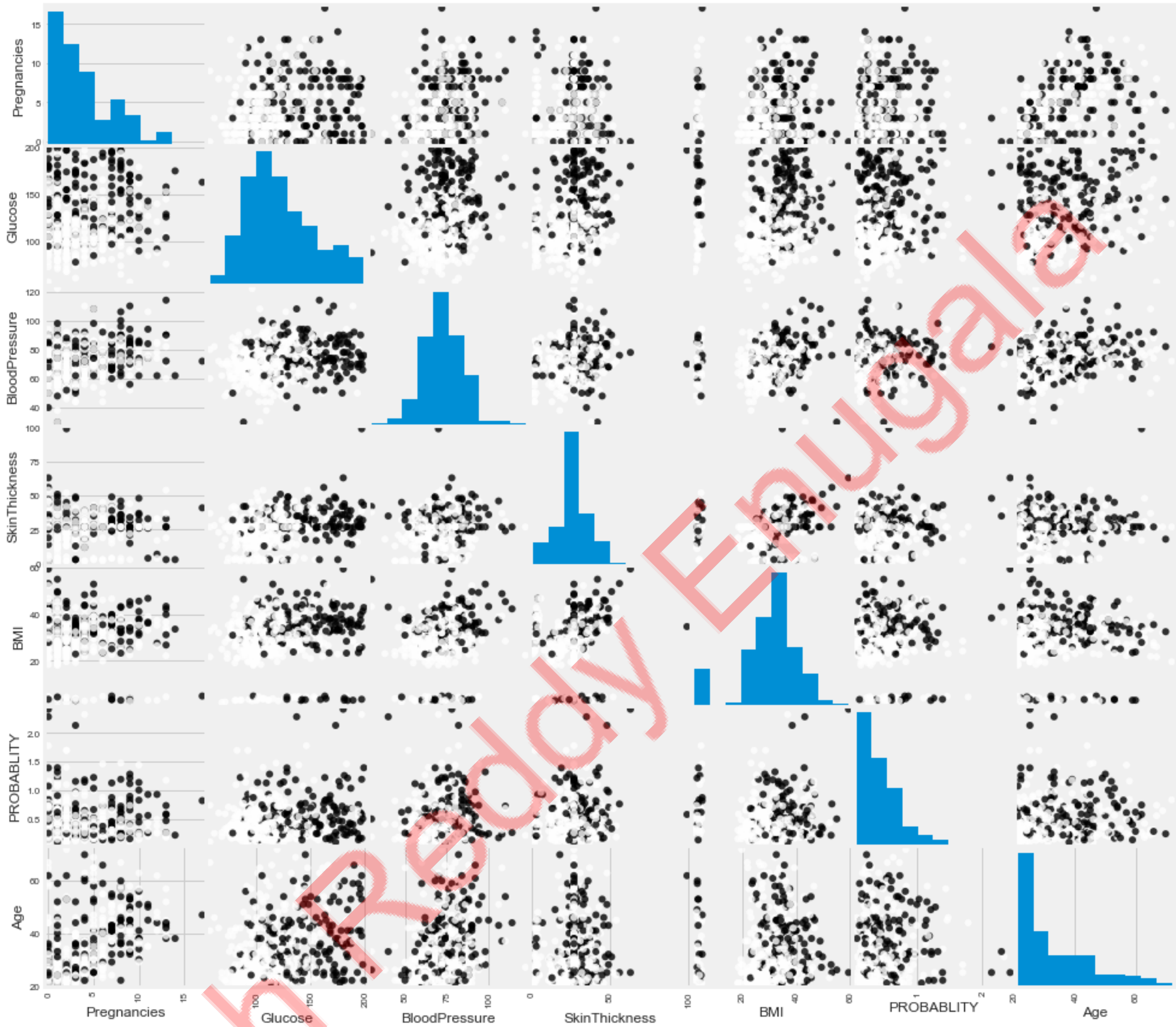
In [132]:
```
diabetes_data_set_imputed['SkinThickness'].fillna(diabetes_data_set_imputed['Skin
diabetes_data_set_imputed.shape
```

Out[132]:
```
(500, 8)
```

In [111]:
```
#aaaaaa
%matplotlib inline
from pandas.plotting import scatter_matrix

X1 = diabetes_data_set_imputed[['Pregnancies', 'Glucose', 'BloodPressure', 'Skin
y1 = diabetes_data_set_imputed.Target

attributes = ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'BMI',

scatter_matrix(X1[attributes], figsize = (15,15), c = y1, alpha = 0.8, marker =
```

Out[111]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF464E240>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF266D518>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF10C1358>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF0DF28D0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF16C9550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF27EA0F0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF12F7E80>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF273D630>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF273D7F0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF24C0358>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF251E898>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF26C6A20>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF262EEF0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF16C49E8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF271E898>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF48DC5F8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF4706390>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF45A96D8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF4499550>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF1F90D68>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF1FB15F8>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF24B8710>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF1AA7898>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF1379DD8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF40D27B8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF420E438>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF1751668>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF172B860>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF1DDA128>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF4215D30>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF1743B00>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF44B1908>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF24CD400>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF0D97978>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF2475E48>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF272CEF0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF4220D68>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF195BA90>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF26B1470>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF1DA2E10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF1F7C7F0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF41181D0>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF1FCEB70>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF44714E0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF1336EF0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF44DB8D0>,
```

```
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF44F72B0>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF4508C50>,
       <matplotlib.axes._subplots.AxesSubplot object at 0x0000011BF43FA630>]],
      dtype=object)
```



In [112]:
```
# Before proceeding for Model selection , re-checking test data what we have
test_data.describe().T
```

Out[112]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	268.0	3.787313	3.327829	0.000	1.00000	3.0000	6.0000	15.000
Glucose	268.0	119.820896	30.522872	0.000	100.00000	119.0000	137.2500	196.000
BloodPressure	268.0	69.925373	17.491197	0.000	64.00000	72.0000	78.0000	110.000
SkinThickness	181.0	25.016575	12.651515	1.000	16.00000	26.0000	35.0000	52.000
Insulin	141.0	104.453901	106.517234	1.000	19.00000	72.0000	168.0000	545.000
BMI	265.0	29.741132	10.818601	2.000	25.50000	31.6000	36.1000	67.100
PROBABLITY	268.0	0.471112	0.312305	0.084	0.25125	0.3615	0.6435	1.893
Age	268.0	33.186567	11.534782	21.000	24.00000	29.5000	39.2500	81.000

In [161]:
```
## showing the count of Nans
print(test_data.isnull().sum())
```

```
Pregnancies        0
Glucose            0
BloodPressure      0
SkinThickness     87
Insulin          127
BMI                3
PROBABLITY         0
Age                0
dtype: int64
```

In [162]:
```
Glucose_zero_count = (test_data['Glucose'] == 0).sum()

print("The number records with  Glucose as Zero: " , Glucose_zero_count)

print("The median for Glucose from training dataset: ",test_data.Glucose.median(
print("The mean for Glucose from training dataset: ",test_data.Glucose.mean())
```

```
The number records with  Glucose as Zero:  2
The median for Glucose from training dataset:  119.0
The mean for Glucose from training dataset:  119.82089552238806
```

In [163]:
```
BloodPressure_zero_count = (test_data['BloodPressure'] == 0).sum()

print("The number records with  BloodPressure as Zero: " , BloodPressure_zero_co

print("The median for BloodPressure from training dataset: ",test_data.BloodPres
print("The mean for BloodPressure from training dataset: ",test_data.BloodPressu
```

```
The number records with  BloodPressure as Zero:  9
The median for BloodPressure from training dataset:  72.0
The mean for BloodPressure from training dataset:  69.92537313432835
```

In [164]:
```
#Use aggregate function for replacing missing values for the columns Glucose, Bl
# few missing values
test_data['Glucose'].fillna(test_data['Glucose'].median(),inplace=True)
test_data['BloodPressure'].fillna(test_data['BloodPressure'].median(),inplace=Tr
test_data['BMI'].fillna(test_data['BMI'].median(),inplace=True)
test_data['SkinThickness'].fillna(test_data['SkinThickness'].median(),inplace=Tr
```

In [165]:
```
# as Insulin has nearly 50% of missing values we can drop it
test_data= test_data.drop(columns=['Insulin'])

test_data.columns    # Insulin Column should not be there because of DROP in pre
```

Out[165]:
```
Index(['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'BMI',
       'PROBABLITY', 'Age'],
      dtype='object')
```

Since we don't have labels known for the test_data for the final model performance measure with confusion matrix, we need to split the given training dataset (diabetes_data_set) into train and test with 80% and 20% respectively.

In [134]:
```
#y = train['Target']
#X = train.drop(['Target'], axis = 1)

y = diabetes_data_set_imputed['Target']
X = diabetes_data_set_imputed.drop(['Target'], axis = 1)
```

In [135]:
```
from sklearn.model_selection import train_test_split

# Partitioning Train dataset  randomly into train and test set using a 80/20 spl
X_train_orig,X_test_orig,y_train,y_test=train_test_split(X,y,test_size=0.2,rando
```

LOGISTIC REGRESSION

In [136]:
```
# Scaling with MinMax TRANSFORMATION
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train_orig)
X_test = scaler.transform(X_test_orig)
```

In [137]:
```
X.head()   # Checking Before scaling versus after scaling
```

Out[137]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	BMI	PROBABLITY	Age
0	5	122.0	86.0	27.0	34.7	0.290	33
1	2	175.0	88.0	27.0	22.9	0.326	22
2	4	129.0	86.0	2.0	35.1	0.231	23
3	12	92.0	62.0	7.0	27.6	0.926	44
4	3	102.0	44.0	2.0	3.8	0.400	26

In [138]:
```
X_train    #after applying min-max scalar the training distribution scaled down l
```

Out[138]:
```
array([[0.11764706, 0.36619718, 0.67391304, ..., 0.70701754, 0.32749787,
        0.64705882],
       [0.76470588, 0.67605634, 0.63043478, ..., 0.03859649, 0.46797609,
        0.35294118],
       [0.05882353, 0.42253521, 0.32608696, ..., 0.55087719, 0.16567037,
        0.11764706],
       ...,
       [0.76470588, 0.50704225, 0.45652174, ..., 0.65789474, 0.20964987,
        0.45098039],
       [0.29411765, 0.71126761, 0.43478261, ..., 0.48070175, 0.05508113,
        0.82352941],
       [0.        , 0.19014085, 0.36956522, ..., 0.58596491, 0.19940222,
        0.        ]])
```

In [139]:
```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)

#Predict the response for test dataset  from the scaled train and test
y_pred_logi_scale = logreg.predict(X_test)
print(confusion_matrix(y_test, y_pred_logi_scale))
print(classification_report(y_test, y_pred_logi_scale))
```

```
[[62  5]
 [19 14]]
              precision    recall  f1-score   support

           0       0.77      0.93      0.84        67
           1       0.74      0.42      0.54        33

    accuracy                           0.76       100
   macro avg       0.75      0.67      0.69       100
weighted avg       0.76      0.76      0.74       100
```

In [141]:
```
#Testing Logic Regression without Scaling , resulting accuracy score reduced comp

logreg = LogisticRegression()
logreg.fit(X_train_orig, y_train)   # Predict the response for test dataset using
                                    # without using scaler

y_pred_logi = logreg.predict(X_test)

print(confusion_matrix(y_test, y_pred_logi))
print(classification_report(y_test, y_pred_logi))
```

```
[[67  0]
 [33  0]]
              precision    recall  f1-score   support

           0       0.67      1.00      0.80        67
           1       0.00      0.00      0.00        33

    accuracy                           0.67       100
   macro avg       0.34      0.50      0.40       100
weighted avg       0.45      0.67      0.54       100
```

DECISION TREE AND RANDOM FOREST

DECISION TREE

In [142]:
```
#parameter selection for Decision Tree and Random Forest
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier

#Parameter tuning using GridSearch

def dt_param_selection(X_train_orig, y_train, nfolds):
    opt_tree = DecisionTreeClassifier(random_state=0)
    param_DT = {"max_depth": range(1,120),
           "min_samples_split": range(2,3,4),
           "max_leaf_nodes": range(2,5)}
    grid_tree = GridSearchCV(opt_tree,param_DT,cv=nfolds)
    grid_tree.fit(X_train_orig,y_train)
    grid_tree.best_params_
    return grid_tree.best_params_

dt_param_selection(X_train_orig, y_train, 5)
```

Out[142]: {'max_depth': 1, 'max_leaf_nodes': 2, 'min_samples_split': 2}

In [179]:
```
# Using GridSearch Parameter tuning recommendation

tree = DecisionTreeClassifier(max_depth=1, max_leaf_nodes=2, min_samples_split=2

# Train Decision Tree Classifer
tree.fit(X_train_orig,y_train)

y_pred_decision_tree = tree.predict(X_test_orig)

print("Accuracy on training set with Decission Tree: {:.4f}".format(clf.score(X_
print("Accuracy on test set with Decission Tree: {:.4f}".format(clf.score(X_test_

print(confusion_matrix(y_test, y_pred_decision_tree))
print(classification_report(y_test, y_pred_decision_tree))
```

```
Accuracy on training set with Decission Tree: 0.3225
Accuracy on test set with Decission Tree: 0.3300
[[58  9]
 [18 15]]
              precision    recall  f1-score   support

           0       0.76      0.87      0.81        67
           1       0.62      0.45      0.53        33

    accuracy                           0.73       100
   macro avg       0.69      0.66      0.67       100
weighted avg       0.72      0.73      0.72       100
```

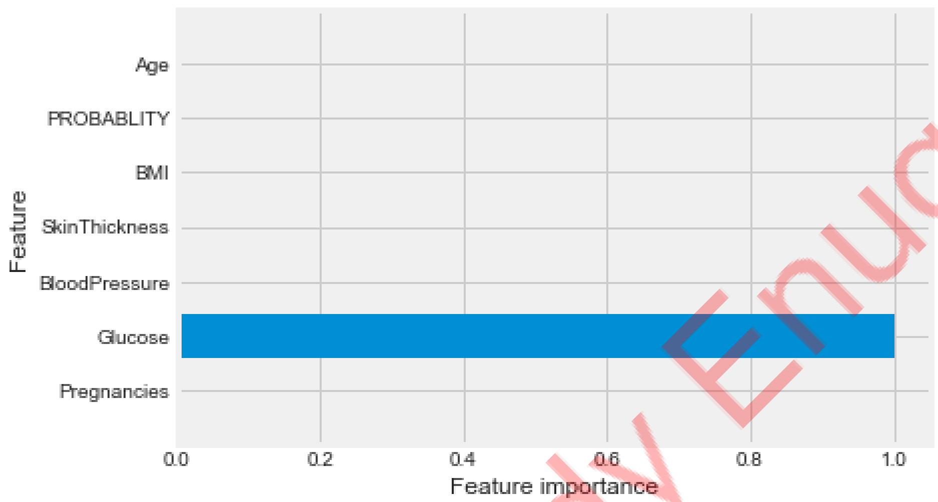
In [144]:
```
print("Feature importances:")
print(clf.feature_importances_)

# The importance of a feature is computed as the (normalized)
# total reduction of the criterion brought by that feature.
# It is also known as the Gini importance.
```

```
Feature importances:
[0. 1. 0. 0. 0. 0. 0.]
```

In [145]:
```
X_train_orig.shape[1]
```

Out[145]:
```
7
```

In [146]:
```
# Feature Importance
def plot_feature_importances(model):
    n_features = X_train_orig.shape[1]
    plt.barh(np.arange(n_features), model.feature_importances_, align='center')
    plt.yticks(np.arange(n_features), X_test_orig.columns)
    plt.xlabel("Feature importance")
    plt.ylabel("Feature")
    plt.ylim(-1, n_features)

plot_feature_importances(clf)
```



RANDOM FOREST

In [147]:

```
# Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

features = list(X_train_orig.columns)

random_forest = RandomForestClassifier(max_depth=1, max_leaf_nodes=2, min_sample
random_forest.fit(X_train_orig, y_train)



y_pred_random_forest = random_forest.predict(X_test_orig)



print("Accuracy score with Random Forest Algorithm: {:.4f}".format(accuracy_scor

print(confusion_matrix(y_test, y_pred_random_forest))
print(classification_report(y_test, y_pred_random_forest))
```

```
Accuracy score with Random Forest Algorithm: 0.7000
[[67  0]
 [30  3]]
              precision    recall  f1-score   support

           0       0.69      1.00      0.82        67
           1       1.00      0.09      0.17        33

    accuracy                           0.70       100
   macro avg       0.85      0.55      0.49       100
weighted avg       0.79      0.70      0.60       100
```

In [148]:

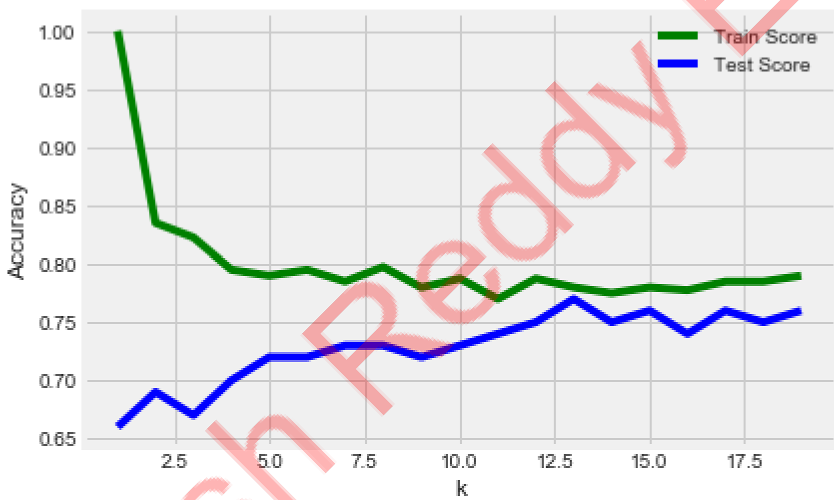
```
# Scaling with MinMax TRANSFORMATION

from sklearn.metrics import classification_report, confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import MinMaxScaler


scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train_orig)
X_test = scaler.transform(X_test_orig)
```

KNN

In [149]:
```
from sklearn.neighbors import KNeighborsClassifier

train_score_array = []
test_score_array = []

for k in range(1,20):
    knn = KNeighborsClassifier(k)
    knn.fit(X_train, y_train)
    train_score_array.append(knn.score(X_train, y_train))
    test_score_array.append(knn.score(X_test, y_test))
```

In [150]:
```
x_axis = range(1,20)
%matplotlib inline
plt.plot(x_axis, train_score_array, label = 'Train Score', c = 'g')
plt.plot(x_axis, test_score_array, label = 'Test Score', c='b')
plt.xlabel('k')
plt.ylabel('Accuracy')
plt.legend()
```

Out[150]: <matplotlib.legend.Legend at 0x11bf3ccb550>



In [151]:
```
#choosing k = 8 as best value
knn = KNeighborsClassifier(19)
knn.fit(X_train, y_train)


print('Train score: {:.4f}'.format(knn.score(X_train, y_train)))
print('Train score: {:.4f}'.format(knn.score(X_test, y_test)))
```

```
Train score: 0.7900
Train score: 0.7600
```

In [152]:
```
y_pred_knn = knn.predict(X_test)


print("Accuracy score with KNN Algorithm: {:.4f}".format(accuracy_score(y_test, y

print(confusion_matrix(y_test, y_pred_knn))
print(classification_report(y_test, y_pred_knn))
```

```
Accuracy score with KNN Algorithm: 0.7600
[[63  4]
 [20 13]]
              precision    recall  f1-score   support

           0       0.76      0.94      0.84        67
           1       0.76      0.39      0.52        33

    accuracy                           0.76       100
   macro avg       0.76      0.67      0.68       100
weighted avg       0.76      0.76      0.73       100
```

SVM

In [153]:
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC


#Create a dictionary called param_grid and fill out some parameters for kernels,
param_grid = {'C': [0.1,1, 10, 100], 'gamma': [1,0.1,0.01,0.001],'kernel': ['lin

#Create a GridSearchCV object and fit it to the training data
grid = GridSearchCV(SVC(),param_grid,refit=True,verbose=2)
grid.fit(X_train,y_train)
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
[CV] C=0.1, gamma=1, kernel=linear ...................................
[CV] ................... C=0.1, gamma=1, kernel=linear, total=   0.0s
[CV] C=0.1, gamma=1, kernel=linear ...................................
[CV] ................... C=0.1, gamma=1, kernel=linear, total=   0.0s
[CV] C=0.1, gamma=1, kernel=linear ...................................
[CV] ................... C=0.1, gamma=1, kernel=linear, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=linear .................................
[CV] ................. C=0.1, gamma=0.1, kernel=linear, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=linear .................................
[CV] ................. C=0.1, gamma=0.1, kernel=linear, total=   0.0s
[CV] C=0.1, gamma=0.1, kernel=linear .................................
[CV] ................. C=0.1, gamma=0.1, kernel=linear, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=linear ................................
[CV] ................ C=0.1, gamma=0.01, kernel=linear, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=linear ................................
[CV] ................ C=0.1, gamma=0.01, kernel=linear, total=   0.0s
[CV] C=0.1, gamma=0.01, kernel=linear ................................
[CV] ................ C=0.1, gamma=0.01, kernel=linear, total=   0.0s
```

In [154]:
```
#Find the optimal parameters
print(grid.best_estimator_)
```

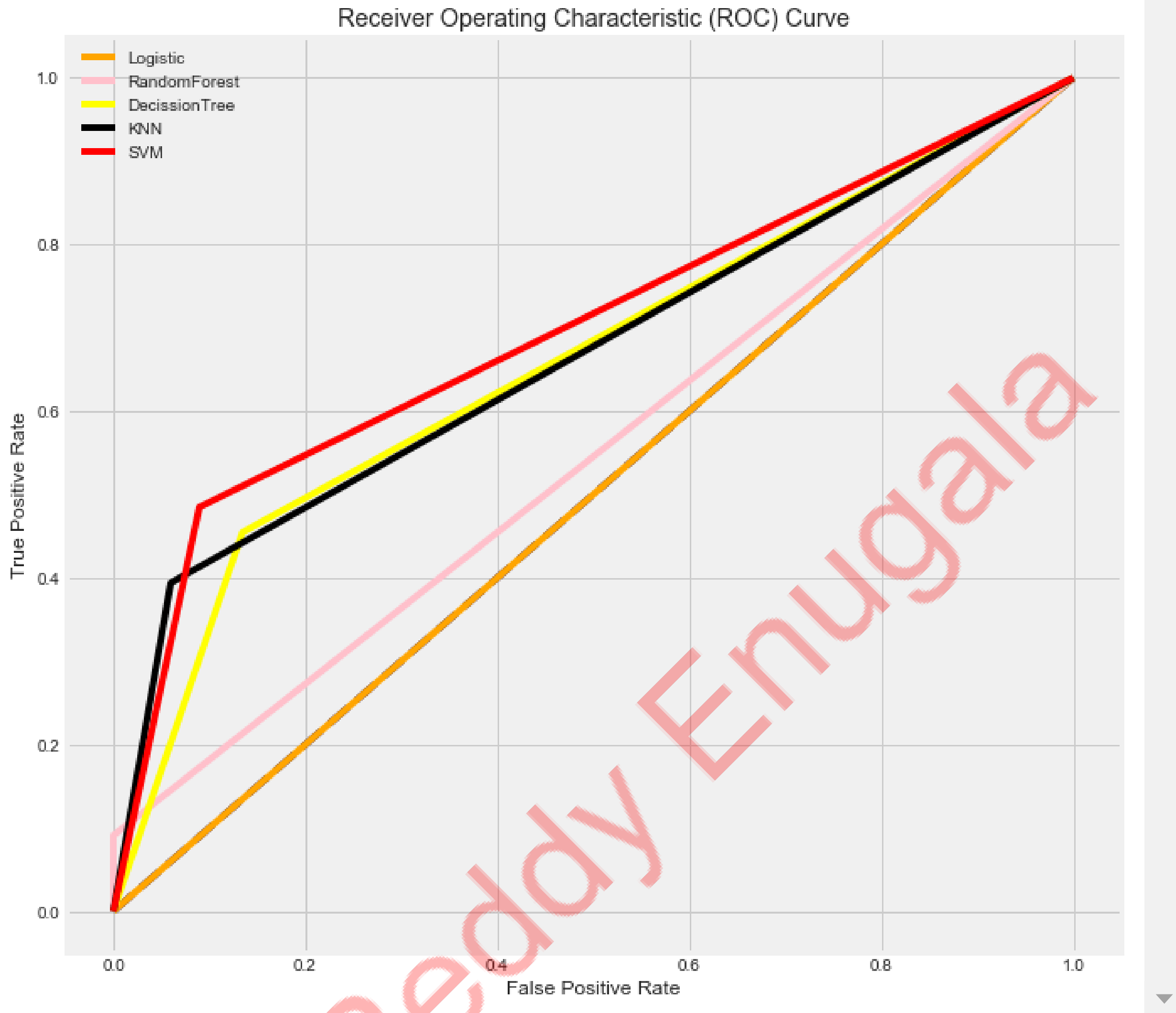
```
SVC(C=10, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=1, kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [172]:
```
#Create a svm Classifier
clf = SVC(C=10, gamma=1, kernel='linear') # Linear Kernel

#Train the model using the training sets
clf.fit(X_train, y_train)

#Predict the response for test dataset
y_pred_svm = clf.predict(X_test)

print("Accuracy score with SVM Algorithm: {:.4f}".format(accuracy_score(y_test, y

print(confusion_matrix(y_test, y_pred_svm))
print(classification_report(y_test, y_pred_svm))
```

```
Accuracy score with SVM Algorithm: 0.7700
[[61  6]
 [17 16]]
              precision    recall  f1-score   support

           0       0.78      0.91      0.84        67
           1       0.73      0.48      0.58        33

    accuracy                           0.77       100
   macro avg       0.75      0.70      0.71       100
weighted avg       0.76      0.77      0.76       100
```

In [180]:

```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

plt.figure(figsize = (10,10))
plt.plot([0, 1], [0, 1], color='darkblue', linestyle='--')

plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic (ROC) Curve')

fpr, tpr, thresholds = roc_curve(y_test, y_pred_logi)
plt.plot(fpr, tpr, color='orange', label='Logistic')


fpr1, tpr1, thresholds = roc_curve(y_test, y_pred_random_forest)
plt.plot(fpr1, tpr1, color='pink', label='RandomForest')

fpr3, tpr3, thresholds = roc_curve(y_test, y_pred_decision_tree)
plt.plot(fpr3, tpr3, color='yellow', label='DecissionTree')

fpr4, tpr4, thresholds = roc_curve(y_test, y_pred_knn)
plt.plot(fpr4, tpr4, color='black', label='KNN')

fpr5, tpr5, thresholds = roc_curve(y_test, y_pred_svm)
plt.plot(fpr5, tpr5, color='red', label='SVM')

plt.legend()
plt.show()
```



In [181]:
```
from sklearn.metrics import roc_auc_score

print(roc_auc_score(y_test,y_pred_logi))
print(roc_auc_score(y_test,y_pred_random_forest))
print(roc_auc_score(y_test,y_pred_decision_tree))
print(roc_auc_score(y_test,y_pred_knn))
print(roc_auc_score(y_test,y_pred_svm))
```

```
0.5
0.5454545454545454
0.6601085481682497
0.6671189507010403
0.6976481230212573
```

We can observe that roc_auc_score for svm is highest hence we predict test dataset with svm

In [175]:
```
#using minmaxscaler() to convert test_data
scaler = MinMaxScaler()
test_data_scale = scaler.fit_transform(test_data)
```

In [184]:
```
#using svm ML algorithm to predict Test values
predict = clf.predict(test_data_scale)
np.unique(predict, return_counts=True)
```

Out[184]:
```
(array([0, 1], dtype=int64), array([153, 115], dtype=int64))
```

In [190]:
```
test_data = test_data.rename(columns = {'Pregnancies':'A','Glucose':'B','BloodPr
                                                'BMI':'F','PROBABLITY':'G','Age'
test_data['Target'] = predict
```

In [193]:
```
test_data.to_csv("predictions.csv", index=None)
```

In [ ]: