Diabetes Prediction (SVM)

In [1]:

```
# Import required libs
import seaborn as sb
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats as sts
%matplotlib inline
```

In [2]:

```
# Read diabetes data-set
aimd = pd.read_csv("Diabetes.csv")
```

In [3]:

```
# Check some information related to the imported data-set
aimd.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                                    Non-Null Count  Dty
pe
---  ------                                    --------------  ---
--
 0   No. of times pregnant                     768 non-null    int
64
 1   Plasma glucose concentration              768 non-null    int
64
 2   Diastolic blood pressure (mm Hg)          768 non-null    int
64
 3   Triceps skin fold thickness (mm)          768 non-null    int
64
 4   2-Hour serum insulin (mu U/ml)            768 non-null    int
64
 5   Body mass index (weight in kg/(height in m)^2)  768 non-null    flo
at64
 6   Diabetes pedigree function                768 non-null    flo
at64
 7   Age (years)                               768 non-null    int
64
 8   Class variable (0 or 1)                   768 non-null    int
64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

In [ ]:

Exploring the data-set (EDA) with plots and stats

In [4]:

```
# Check coloumn set
list(aimd.columns)
```

Out[4]:

```
['No. of times pregnant',
 'Plasma glucose concentration',
 'Diastolic blood pressure (mm Hg)',
 'Triceps skin fold thickness (mm)',
 '2-Hour serum insulin (mu U/ml)',
 'Body mass index (weight in kg/(height in m)^2)',
 'Diabetes pedigree function',
 'Age (years)',
 'Class variable (0 or 1)']
```

In [5]:

```
# Rename columns to short values
aimd=aimd.rename(columns={'No. of times pregnant':'Pregnancies', 'Plasma glucose concen
tration':'Glucose',
                          'Diastolic blood pressure (mm Hg)':'BloodPressure','Triceps s
kin fold thickness (mm)':'SkinThickness',
                          '2-Hour serum insulin (mu U/ml)':'Insulin', 'Body mass index
 (weight in kg/(height in m)^2)':'BMI',
                          'Diabetes pedigree function':'Pedigree', 'Age (years)':'Age',
'Class variable (0 or 1)':'Outcome'})
```

In [6]:

```
# Check data-types
aimd.dtypes
```

Out[6]:

```
Pregnancies        int64
Glucose            int64
BloodPressure      int64
SkinThickness      int64
Insulin            int64
BMI              float64
Pedigree         float64
Age                int64
Outcome            int64
dtype: object
```

In [7]:

```
# Check some samples of the data
aimd.head(3)
```

Out[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Pedigree	Age	Outcc
0	6	148	72	35	0	33.6	0.627	50	
1	1	85	66	29	0	26.6	0.351	31	
2	8	183	64	0	0	23.3	0.672	32	

In [8]:

```
# Check some information about data
print("""DataFrame Dimensions = {0}
Dataframe Shape = {1}""".format(aimd.ndim, aimd.shape))
```

```
DataFrame Dimensions = 2
Dataframe Shape = (768, 9)
```

In [9]:

```
# Check some relevant stats about the data
aimd.describe(percentiles=[.05,.25,.5,.75,.99]).T
# The data looks neatly distributed
```

Out[9]:

	count	mean	std	min	5%	25%	50%	75
Pregnancies	768.0	3.845052	3.369578	0.000	0.00000	1.00000	3.0000	6.000
Glucose	768.0	120.894531	31.972618	0.000	79.00000	99.00000	117.0000	140.250
BloodPressure	768.0	69.105469	19.355807	0.000	38.70000	62.00000	72.0000	80.000
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	0.00000	23.0000	32.000
Insulin	768.0	79.799479	115.244002	0.000	0.00000	0.00000	30.5000	127.250
BMI	768.0	31.992578	7.884160	0.000	21.80000	27.30000	32.0000	36.600
Pedigree	768.0	0.471876	0.331329	0.078	0.14035	0.24375	0.3725	0.626
Age	768.0	33.240885	11.760232	21.000	21.00000	24.00000	29.0000	41.000
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.00000	0.0000	1.000

In [10]:

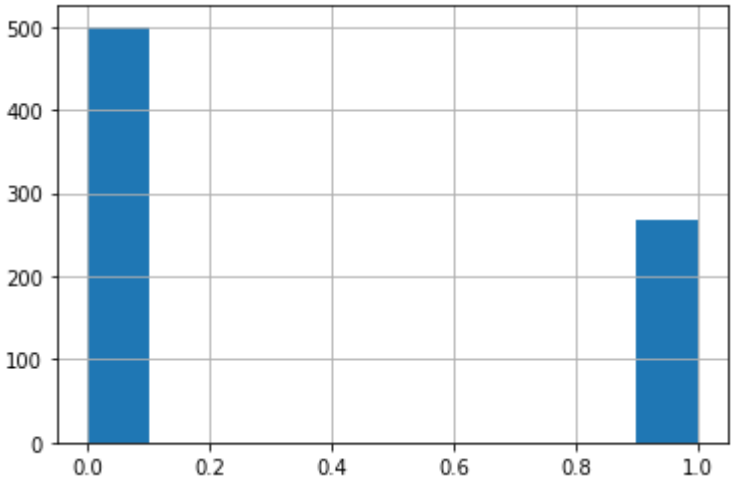
```
# Check that Outcome is either 0/1
print(aimd.Outcome.unique())
```

```
[1 0]
```

In [11]:

```
# Check class balance
print(aimd.Outcome.value_counts())
aimd.Outcome.hist()
plt.show()
```

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



In [12]:

```
# Check duplicates, no duplicates if shape = (0, M)
aimd[aimd.duplicated()].shape
```

Out[12]:

```
(0, 9)
```

In [13]:

```
# Check null values
aimd.isna().sum()
```

Out[13]:

```
Pregnancies      0
Glucose          0
BloodPressure    0
SkinThickness    0
Insulin          0
BMI              0
Pedigree         0
Age              0
Outcome          0
dtype: int64
```

In [14]:

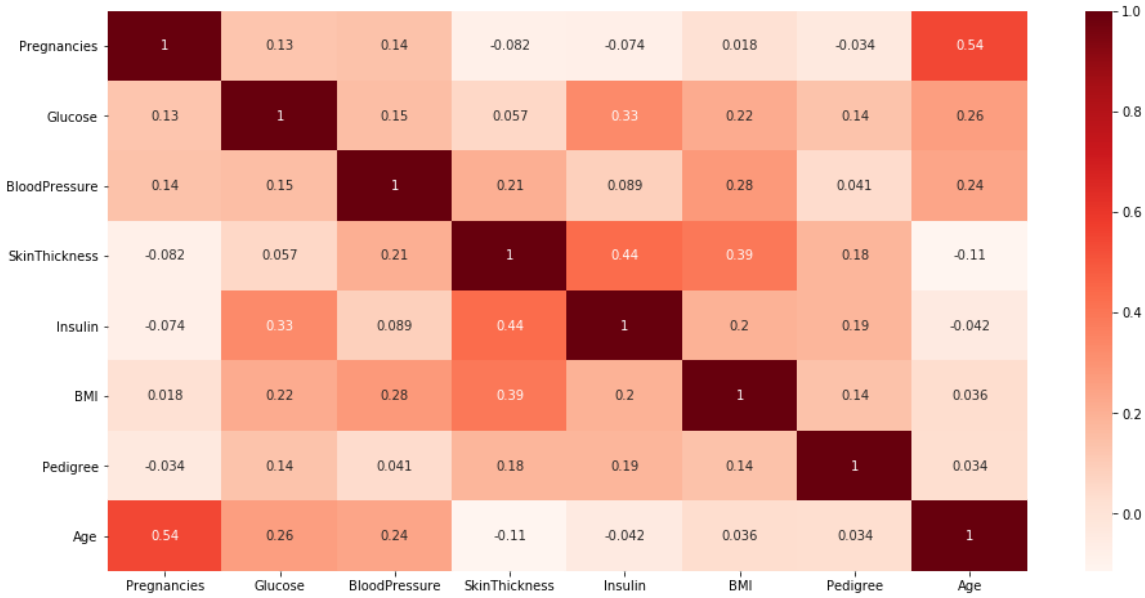
```
# Check the correlation matrix (exclude Outcome)
corr = aimd.iloc[:,:-1].corr()
corr
```

Out[14]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	
Pregnancies	1.000000	0.129459	0.141282	-0.081672	-0.073535	0.017683	-
Glucose	0.129459	1.000000	0.152590	0.057328	0.331357	0.221071	
BloodPressure	0.141282	0.152590	1.000000	0.207371	0.088933	0.281805	
SkinThickness	-0.081672	0.057328	0.207371	1.000000	0.436783	0.392573	
Insulin	-0.073535	0.331357	0.088933	0.436783	1.000000	0.197859	
BMI	0.017683	0.221071	0.281805	0.392573	0.197859	1.000000	
Pedigree	-0.033523	0.137337	0.041265	0.183928	0.185071	0.140647	
Age	0.544341	0.263514	0.239528	-0.113970	-0.042163	0.036242	

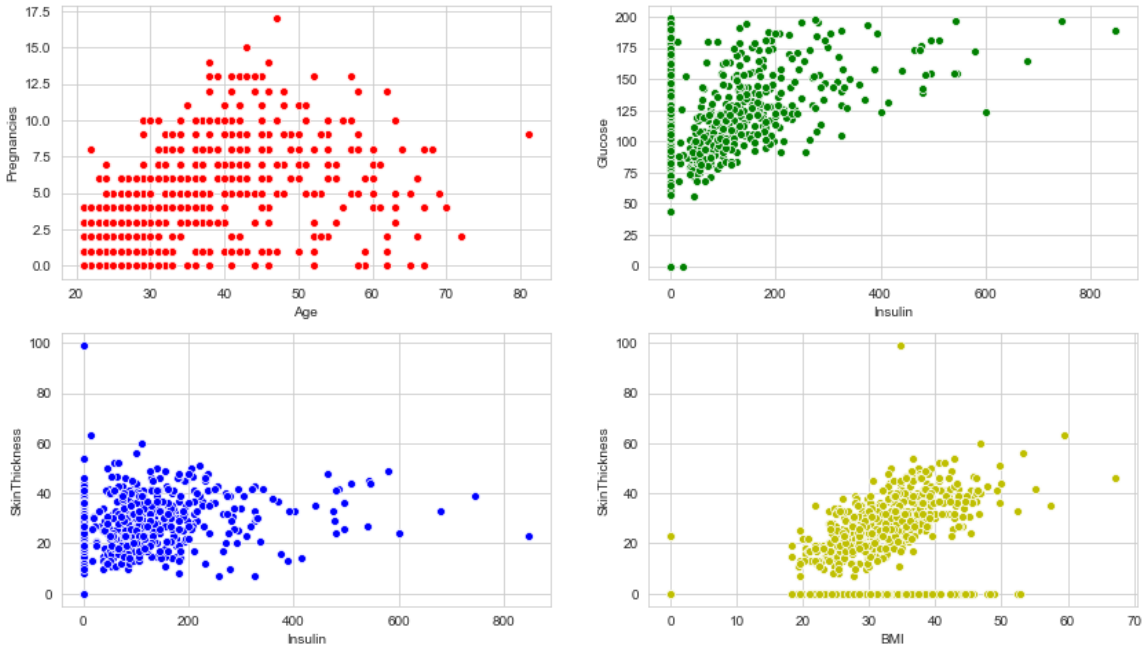
In [15]:

```
plt.figure(figsize=(16,8))
sb.heatmap(corr, cmap='Reds', annot=True)
plt.show()
```



In [16]:

```
# Check correlations for >30% above using multi-variate plots
sb.set_style('whitegrid')
plt.figure(figsize=[14,8])
plt.subplot(2,2,1)
sb.scatterplot(x=aimd.Age, y=aimd.Pregnancies, color='r')
plt.subplot(2,2,2)
sb.scatterplot(x=aimd.Insulin, y=aimd.Glucose, color='g')
plt.subplot(2,2,3)
sb.scatterplot(x=aimd.Insulin, y=aimd.SkinThickness, color='b')
plt.subplot(2,2,4)
sb.scatterplot(x=aimd.BMI, y=aimd.SkinThickness, color='y')
plt.show()
```

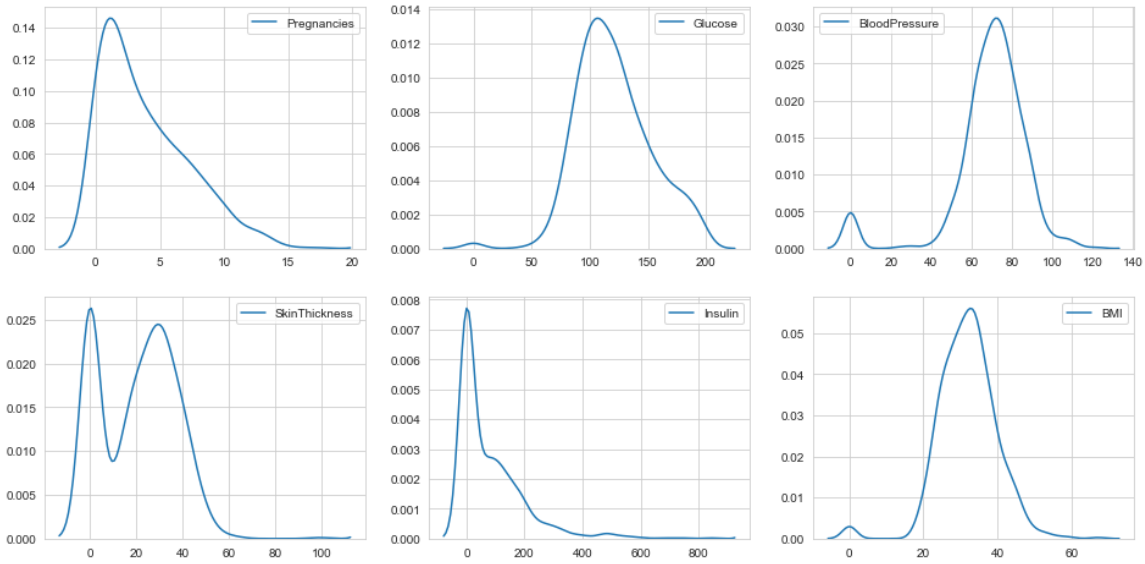


In [17]:

```
# Scatter and pair plots of data
sb.pairplot(aimd, diag_kind='kde', hue='Outcome')
plt.show()
```

In [18]:

```
# Minimum values appears as 0 for few features, check the distributions
plt.figure(figsize=[16,8])
plt.subplot(2,3,1)
sb.kdeplot(aimd.Pregnancies)
plt.subplot(2,3,2)
sb.kdeplot(aimd.Glucose)
plt.subplot(2,3,3)
sb.kdeplot(aimd.BloodPressure)
plt.subplot(2,3,4)
sb.kdeplot(aimd.SkinThickness)
plt.subplot(2,3,5)
sb.kdeplot(aimd.Insulin)
plt.subplot(2,3,6)
sb.kdeplot(aimd.BMI)
plt.show()
```



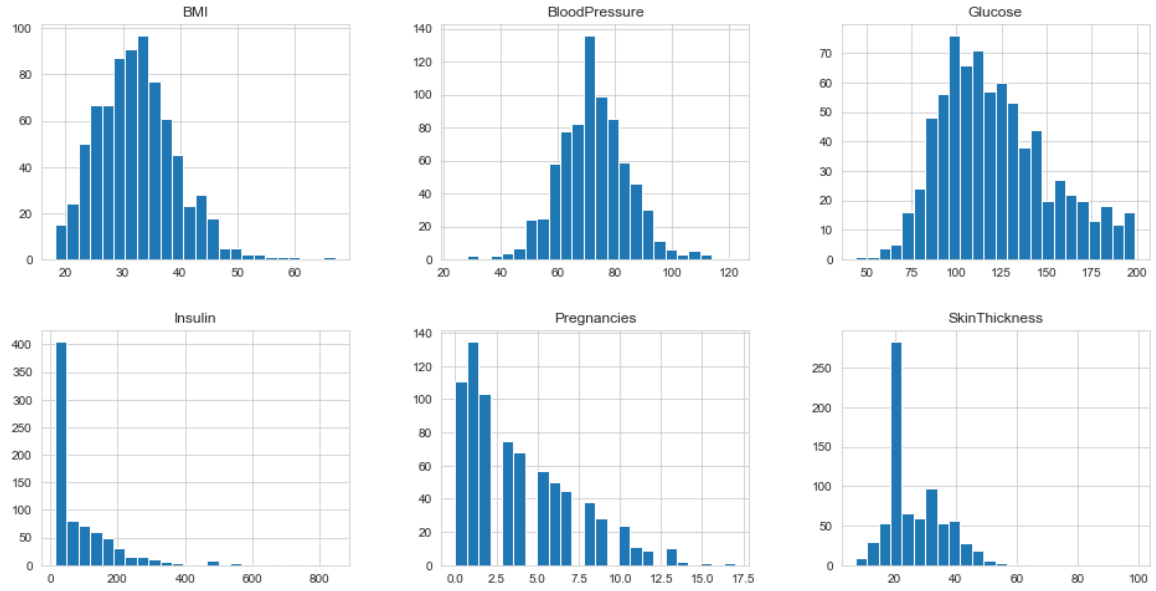
In [19]:

```
# Glucose, BP, SkinThickness and BMI are distributed normally, replace with mean
aimd.loc[aimd[aimd['Glucose'] == 0].index, 'Glucose'] = aimd['Glucose'].mean()
aimd.loc[aimd[aimd['BloodPressure'] == 0].index, 'BloodPressure'] = aimd['BloodPressur
e'].mean()
aimd.loc[aimd[aimd['SkinThickness'] == 0].index, 'SkinThickness'] = aimd['SkinThicknes
s'].mean()
aimd.loc[aimd[aimd['BMI'] == 0].index, 'BMI'] = aimd['BMI'].mean()

# Insulin is right-skewed, replace this with median
aimd.loc[aimd[aimd['Insulin'] == 0].index, 'Insulin'] = aimd['Insulin'].median()
```

In [20]:

```
aimd[['Pregnancies','Glucose','BloodPressure','SkinThickness','BMI','Insulin']].hist(fi
gsize=[16,8], bins=24, layout=(2,3))
plt.show()
```



In [ ]:

```

```

Modelling using SKLearn

In [21]:

```
# Import relevant sklearn libs
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC, LinearSVC
from sklearn.feature_selection import RFE
from sklearn.metrics import confusion_matrix, classification_report, accuracy_score, f1
_score
```

In [22]:

```
# Feature Importance
y=['Outcome']
X=[i for i in aimd.columns if i!='Outcome']
print(X)
rfe = RFE(SVC(kernel='linear')).fit(aimd[X], aimd[y].values.ravel())
print(rfe.support_)
print(rfe.ranking_)
```

```
['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'B
MI', 'Pedigree', 'Age']
[ True  True False False False  True  True False]
[1 1 3 2 5 1 1 4]
```

In [23]:

```
# Gather data and take the features for modelling
# Not considering the above unimportant columns, they are also highly correlated from t
he earlier EDA

X = aimd.loc[:, ['Pregnancies', 'Glucose', 'BMI', 'Pedigree']]
y = aimd['Outcome']

# Normalize/scale the features to have SVM find correct margins
X = StandardScaler(copy=False).fit(X).transform(X)

# Split to train and test (stratified)
X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=2/3, test_size=1/3
, stratify=y, random_state=0)
```

In [24]:

```
# Check best params
cv_shuffle = StratifiedKFold(n_splits=10, shuffle=True, random_state=0)
best_svm = GridSearchCV(SVC(), param_grid = {'kernel':['linear','rbf']}, cv = cv_shuffl
e)
best_svm.fit(X,y)
best_svm.best_score_, best_svm.best_params_
```

Out[24]:

```
(0.7734107997265892, {'kernel': 'rbf'})
```

In [25]:

```
# Create the SVM model
model = SVC(kernel='rbf')
model.fit(X_train, y_train)
y_pred = model.predict(X_test)
print('Prediction Accuracy:', round(accuracy_score(y_test, y_pred), 4) * 100, '%')
```

```
Prediction Accuracy: 75.39 %
```

In [ ]:

Evaluate Model Performance

In [26]:

```
# Print the confusion matrix
pd.DataFrame(pd.crosstab(y_test, y_pred, rownames=['Actual'], colnames=['Predicted'], m
argins_name='Total', margins=True))
```

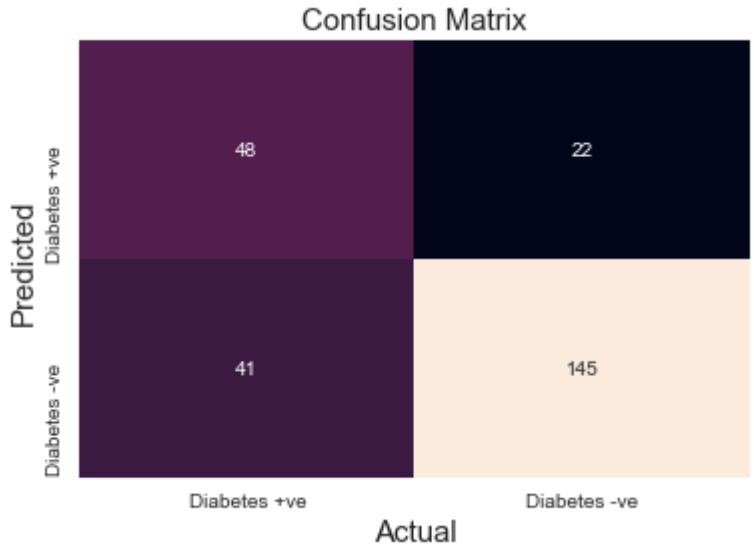
Out[26]:

Predicted	0	1	Total
Actual			
0	145	22	167
1	41	48	89
Total	186	70	256

In [27]:

```
classes=['Diabetes +ve', 'Diabetes -ve']
sb.heatmap(np.rot90(confusion_matrix(y_test,y_pred),2).T,annot=True,fmt='d',xticklabels
=classes,yticklabels=classes,cbar=False)
plt.xlabel('Actual', size=15)
plt.ylabel('Predicted', size=15)
plt.title('Confusion Matrix', size=15)
plt.show()

# Print full classification report showing precision, recall, etc.
print('\033[1m\033[4m' + 'Report for Support Vector Machine (Non-Linear)\n' + '\033[0m'
)
print(classification_report(y_test, y_pred, digits=3, target_names=['Diabetes -ve', 'Di
abetes +ve']))
```



Report for Support Vector Machine (Non-Linear)

	precision	recall	f1-score	support
Diabetes -ve	0.780	0.868	0.822	167
Diabetes +ve	0.686	0.539	0.604	89
accuracy			0.754	256
macro avg	0.733	0.704	0.713	256
weighted avg	0.747	0.754	0.746	256