In [ ]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import seaborn as sns
```

Reading the diabetes dataset

(Downloaded from [here (https://www.kaggle.com/uciml/pima-indians-diabetes-database)](https://www.kaggle.com/uciml/pima-indians-diabetes-database).)

In [ ]:
```
#reading the diabetes dataset from the folder and storing i
t in a dataframe named diabetes
diabetes = pd.read_csv("./diabetes.csv")
```

Dimensions of the dataframe

In [ ]:
```
diabetes.shape
```

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

Sample data in dataframe

In [ ]:
```
diabetes.head()
```

Out[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPe
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	

Problem description:

- *Binary classification problem*
- *Label : 'Outcome' = 0 or 1*
- *Features :*
    - 'Pregnancies'
    - 'Glucose'
    - 'BloodPressure'
    - 'SkinThickness'
    - 'Insulin'
    - 'BMI'
    - 'DiabetesPedigreeFunction'
    - 'Age'

Summary of the data

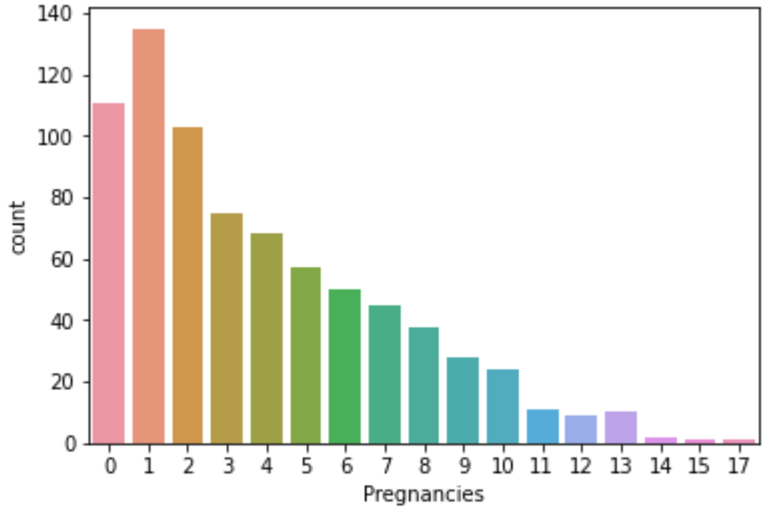
In [ ]:
```
diabetes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype  
---  ------                    --------------  -----  
 0   Pregnancies               768 non-null    int64  
 1   Glucose                   768 non-null    int64  
 2   BloodPressure             768 non-null    int64  
 3   SkinThickness             768 non-null    int64  
 4   Insulin                   768 non-null    int64  
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64  
 8   Outcome                   768 non-null    int64  
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

Step 1: Visualisation (part 1)

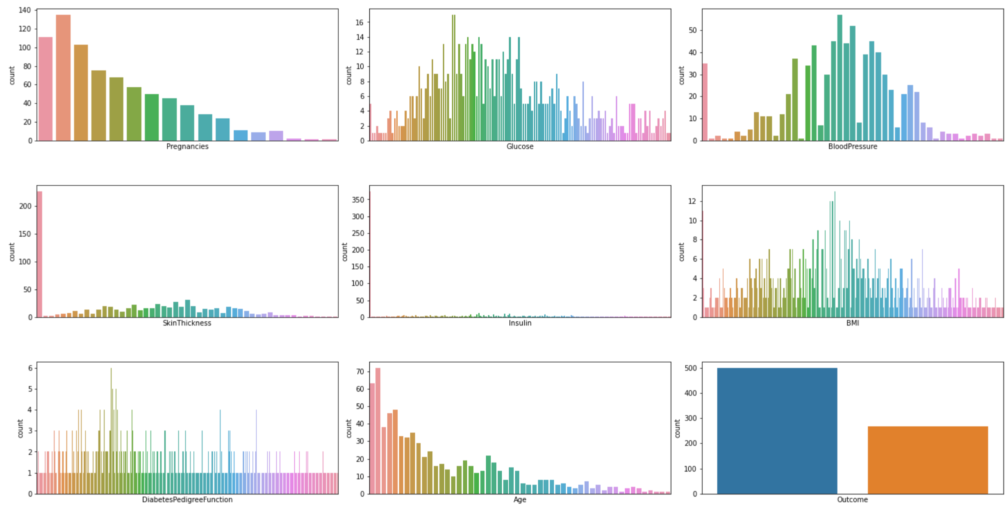
Since we have discrete categorical data, we plot the estimate plots using countplot

In [ ]:
```
sns.countplot(diabetes['Pregnancies']);
```



Plotting the rest of the features using countplots

In [ ]:
```
fig, axs = plt.subplots(ncols=3, nrows=3, figsize=(20, 10))
index = 0
axs = axs.flatten()
for i in diabetes.columns:
    sns.countplot(diabetes[i], ax=axs[index])
    axs[index].set_xticks([])
    index+=1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



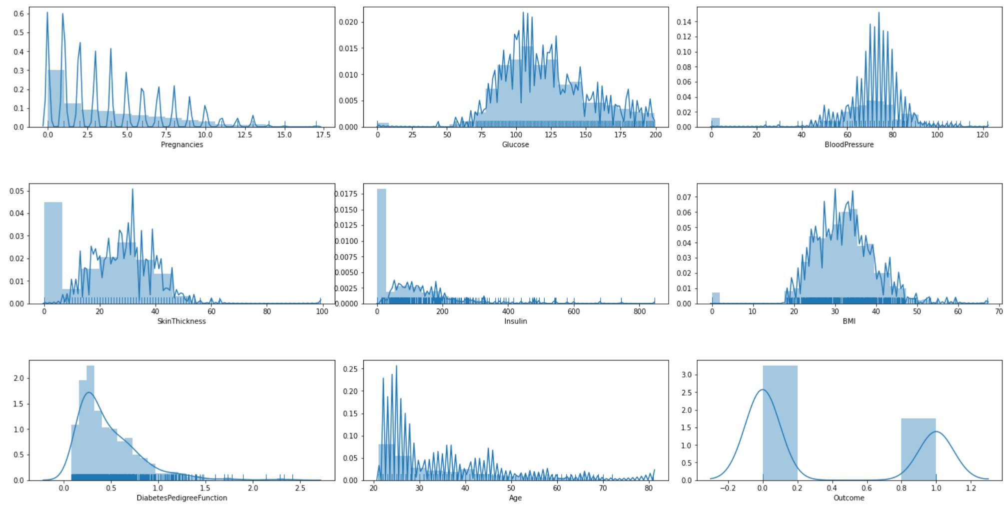
Inference:

- *More data of non-diabetics present than diabetics.*
- *Prediction accuracy whether a person is non-diabetic will be higher than him/her being diabetic.*
- *For features Skin Thickness, and Insulin, zero values must be imputed.*

Visualising using distplot : evenness of data spread

Output: *A curve that roughly fits the distribution.*
(We also add a rugplot which marks each individual point on the x-axis)

In [ ]:
```
fig, axs = plt.subplots(ncols=3, nrows=3, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in diabetes.items():
    sns.distplot(v, ax=axs[index],kde_kws={'bw': 0.1}, rug=True)
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



Inference:

- *Glucose, BP, BMI have approximately normally distributed data.*
- *DiabetesPedigreeFunction peak has a slight shift to the left.*
- *SkinThickness, Insulin have a sharp spike, due to imputation of 0 values with a single fixed values.*
  To handle this, we use normalization/ standardisation further.

Step 2: Checking for missing value and string datatype and abnormal values

In [ ]:
```
diabetes.isnull().sum()
```

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

Inference: *No null data present*

In [ ]:
```
diabetes.dtypes
```

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

Inference: *No string or object datatype present*

In [ ]:
```
pd.DataFrame(diabetes[:]==0).sum()
```

Out[ ]:
```
Pregnancies                 111
Glucose                       5
BloodPressure                35
SkinThickness               227
Insulin                     374
BMI                          11
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64
```

Inference: *Glucose, BP, SkinThickness, Insulin, BMI can't be 0 - data has to be processed*

Step 3: Imputing the abnormal values

In [ ]:
```
diabetes1 = diabetes.copy(deep=True) # copy of dataframe made in order to keep original dataframe unchanged
```

Using mean to impute the zero values for columns: Glucose, BP, SkinThickness, Insulin, BMI

In [ ]:
```
columns = ['Glucose', 'BloodPressure', 'SkinThickness', 'BMI']

for i in columns:
    avg = diabetes1[i][diabetes1[i]>0].mean()
    diabetes1[i] = diabetes1[i].replace(to_replace=0, value=avg)
```

In [ ]:
```
md = diabetes1['Insulin'][diabetes1['Insulin']>0].mode()[0]
diabetes1['Insulin'] = diabetes1['Insulin'].replace(to_replace=0, value=md)
```

In [ ]:
```
pd.DataFrame(diabetes1[:]==0).sum()
```

Out[ ]:
```
Pregnancies                 111
Glucose                       0
BloodPressure                 0
SkinThickness                 0
Insulin                       0
BMI                           0
DiabetesPedigreeFunction      0
Age                           0
Outcome                     500
dtype: int64
```

Inference: *All zero values replaced with mean of the rest of the values.*
Possible difficulty: *The distribution of data maybe spiked since there were lot of zero values in particularly* 'SkinThickness' *and* 'Insulin' *columns.*

Step 4: Data analysis and visualisation(part 2)

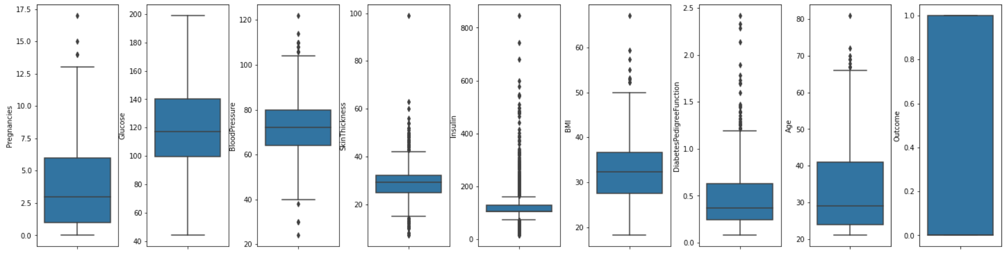
In [ ]:
```
diabetes1.describe()
```

Out[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.0
mean	3.845052	121.686763	72.405184	29.153420	130.932292	32.4
std	3.369578	30.435949	12.096346	8.790942	88.700443	6.8
min	0.000000	44.000000	24.000000	7.000000	14.000000	18.2
25%	1.000000	99.750000	64.000000	25.000000	105.000000	27.5
50%	3.000000	117.000000	72.202592	29.153420	105.000000	32.4
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.6
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.1

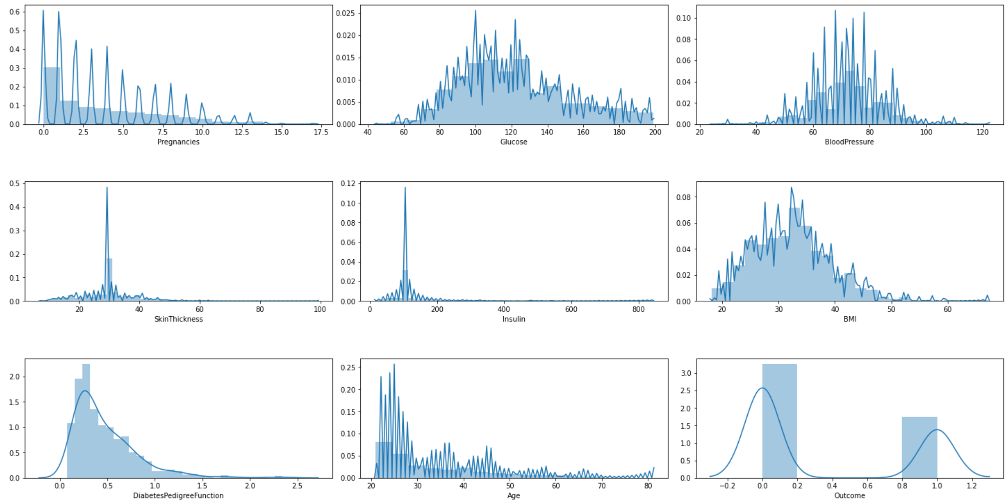
Inference: *High variation in all columns*
Possible solution: *Scaling of data: either normalization or standardization*

In [ ]:
```
fig, axs = plt.subplots(ncols=9, nrows=1, figsize=(20, 5))
index = 0
axs = axs.flatten()

# for k,v in diabetes.items():
#     sns.boxplot(y=v, data=diabetes, ax=axs[index])
#     index += 1

for k,v in diabetes1.items():
    sns.boxplot(y=v, data=diabetes1, ax=axs[index])
    index += 1

plt.tight_layout(pad=0.4, w_pad=0.1, h_pad=5.0)
```



Inference: *Imputation of abnormal values and plotting the boxplot, shows outliers towards the lower range have been successfully removed.*

In [ ]:
```
fig, axs = plt.subplots(ncols=3, nrows=3, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in diabetes1.items():
    sns.distplot(v, ax=axs[index],kde_kws={'bw': 0.1}) # fo
r some prob write kde
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



Inference: *Imputation of abnormal values and plotting the distplot, shows outliers towards the lower range have been successfully removed.*

Step 5: Treating outliers

In [ ]:
```
diabetes1_0 = diabetes1.copy(deep=True)
```

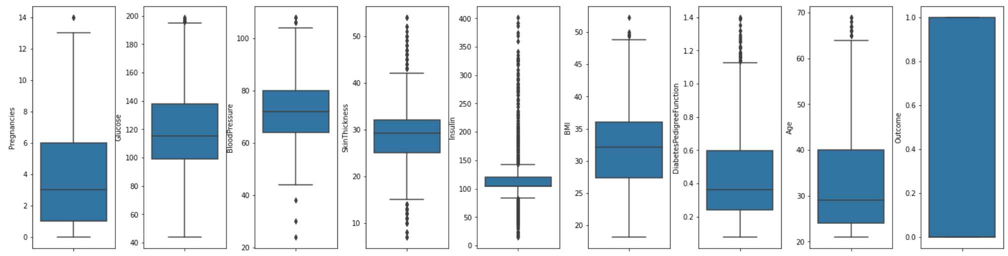
In [ ]:
```
for i in diabetes1_0.columns:
    upper = diabetes1_0[i].mean() + diabetes1_0[i].std()*
3.1
    print(upper)
    diabetes1_0 = diabetes1_0[~(diabetes1_0[i] >= upper)]
```

```
14.290744077699808
215.9613909365196
109.95551519809001
56.38625954346237
407.2634508820364
52.81267394865042
1.4346189055800058
69.44588978318626
1.8024381326507237
```

In [ ]:
```
diabetes1_0.shape
```

Out[ ]:
```
(722, 9)
```

In [ ]:
```
fig, axs = plt.subplots(ncols=9, nrows=1, figsize=(20, 5))
index = 0
axs = axs.flatten()

for k,v in diabetes1_0.items():
    sns.boxplot(y=v, data=diabetes1_0, ax=axs[index])
    index += 1

plt.tight_layout(pad=0.4, w_pad=0.1, h_pad=5.0)
```



Inference :

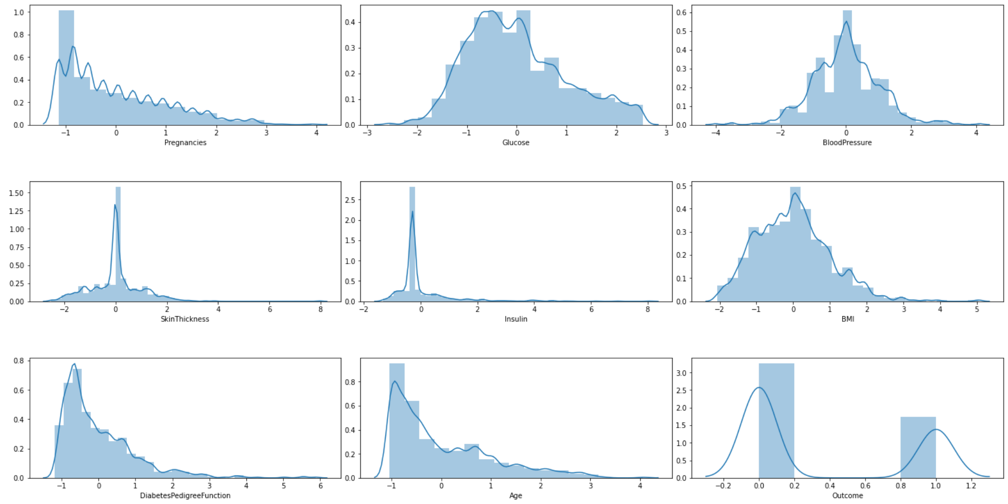
Manual outlier not preferable as automatic outlier detection (see train test part) gives vetter accuracy score for ML models.

Step 6: Normalization/ Standardisation of data (visualisation part-2)

In [ ]:
```
diabetes2 = diabetes1.copy(deep=True)
sc = StandardScaler()
```

In [ ]:
```
# scaling all columns except 1st and last
for i in diabetes2.columns[0:-1]:
    diabetes2[i] = sc.fit_transform(pd.DataFrame(diabetes2.
loc[:, i]).values)
```

In [ ]:
```python
fig, axs = plt.subplots(ncols=3, nrows=3, figsize=(20, 10))
index = 0
axs = axs.flatten()
for k,v in diabetes2.items():
    sns.distplot(v, ax=axs[index],kde_kws={'bw': 0.1}) # fo
r some prob write kde
    index += 1
plt.tight_layout(pad=0.4, w_pad=0.5, h_pad=5.0)
```



Inference:

- Variation is less now; with mean as 0 and standard deviation as 1
- Though the peaks in SkinThickness and Insulin could not be removed, the desired normal distribution has been achieved.

In [ ]:
```python
diabetes2.describe()
```

Out[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin
count	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+02	7.680000e+0
mean	2.544261e-17	-3.301757e-16	6.966722e-16	6.866252e-16	3.122502e-17
std	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+00	1.000652e+0
min	-1.141852e+00	-2.554131e+00	-4.004245e+00	-2.521670e+00	-1.319142e+0
25%	-8.448851e-01	-7.212214e-01	-6.953060e-01	-4.727737e-01	-2.925486e-01
50%	-2.509521e-01	-1.540881e-01	-1.675912e-02	8.087936e-16	-2.925486e-01
75%	6.399473e-01	6.103090e-01	6.282695e-01	3.240194e-01	-4.154084e-0
max	3.906578e+00	2.541850e+00	4.102655e+00	7.950467e+00	8.066856e+0

Inference: *Variation is less now.*

Step 7: Exploring linearity of data (visualisation - part 3)

In [ ]:
```
diabetes2.corr()
```

Out[ ]:

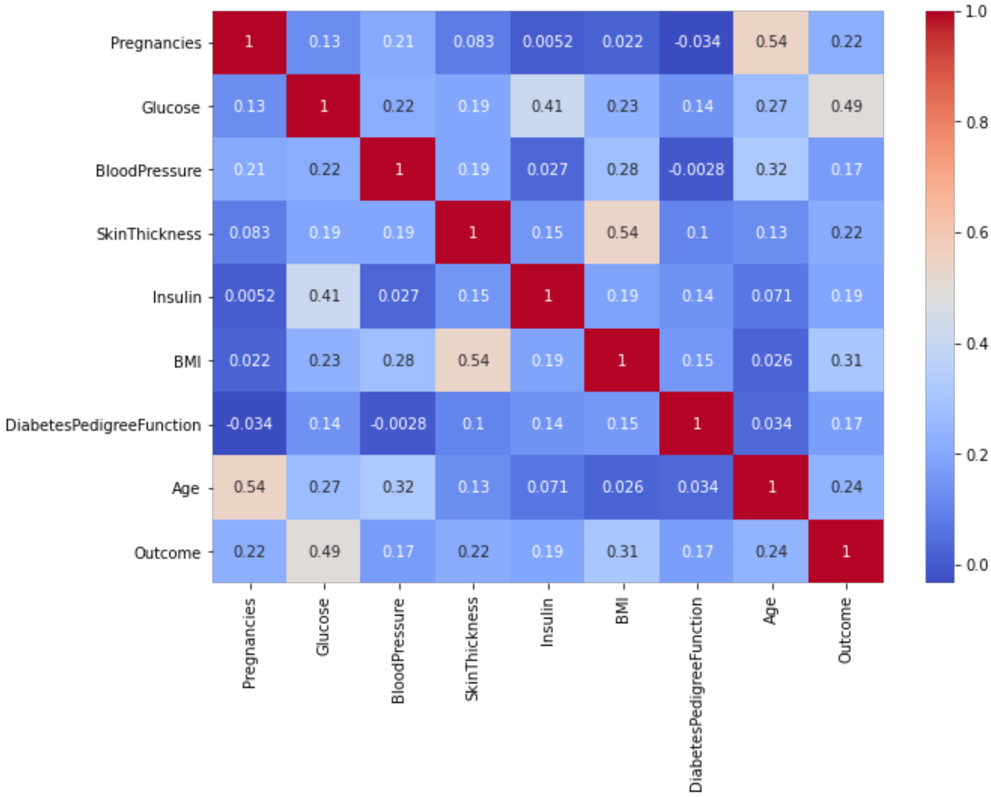
	Pregnancies	Glucose	BloodPressure	SkinThickness
Pregnancies	1.000000	0.127911	0.208522	0.082989
Glucose	0.127911	1.000000	0.218367	0.192991
BloodPressure	0.208522	0.218367	1.000000	0.192816
SkinThickness	0.082989	0.192991	0.192816	1.000000
Insulin	0.005204	0.411642	0.027149	0.150020
BMI	0.021565	0.230941	0.281268	0.542398
DiabetesPedigreeFunction	-0.033523	0.137060	-0.002763	0.100966
Age	0.544341	0.266534	0.324595	0.127872
Outcome	0.221898	0.492928	0.166074	0.215299

Inference: *Comparatively higher correlation between*

- *age and pregnancies, which is normal.*
- *skin thick and BMI, which also can be related*

Here maximum correlation is 0.54.
Had there been any correlation value been > 0.8 we would have selected one feature of the two correlated feature.
Here we are unable to eliminate any features.

In [ ]:
```
fig, ax = plt.subplots(figsize=(10,7))
sns.heatmap(diabetes2.corr(), annot = True, cmap="coolwar
m");
```



Inference: *Comparatively higher correlation between*

- Age and pregnancies, which is normal.
- SkinThickness and BMI, which also can be related

had there been any correlation value been > 0.8 we would have selected one feature of the two

Step 8: Machine Learning Models

This is a **binary classification** problem.
The models we will apply are:

1. Logistic Regression
2. Naive Bayes Classifier (Gaussian)
3. SVM (linear, poly, radial kernel)
4. Decision Tree
5. Random Forest

Then we conclude which is the best model to be applied.

Importing the Libraries

In [ ]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn import svm
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split, KFol
d, cross_val_score
from sklearn.metrics import classification_report, confusio
n_matrix, accuracy_score
```

Pre-step 1: Diving into train and test set

In [ ]:
```
X = diabetes2.iloc[:, :8].values
```

In [ ]:
```
Y = diabetes2.iloc[:,8].values
```

In [ ]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, t
est_size=0.1, random_state=1)
```

In [ ]:
```
# kfold = KFold(n_splits=10, random_state=10)
```

Pre-step 2: automatic outlier detection

In [ ]:
```
from sklearn.svm import OneClassSVM
ee = OneClassSVM(nu=0.01)
yhat = ee.fit_predict(X_train)
# select all rows that are not outliers
mask = yhat != -1
X_train, Y_train = X_train[mask, :], Y_train[mask]
```

Setting DataFrame to store accuracies

In [ ]:
```
acc_stats = pd.DataFrame(columns=['Algorithm used', 'Train
Score', 'Test Score'])
```

Model 1: Logistic Regression

In [ ]:
```
log = LogisticRegression()
```

In [ ]:
```
log.fit(X_train, Y_train)
```

Out[ ]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fi
t_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_
iter=100,
                   multi_class='auto', n_jobs=None, penalty
='l2',
                   random_state=None, solver='lbfgs', tol=
0.0001, verbose=0,
                   warm_start=False)
```

In [ ]:
```
Y_pred = log.predict(X_test)
```

In [ ]:
```
Y_pred
```

Out[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0,
1, 0, 0, 0, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 1,
0, 1, 0, 0, 0,
       1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 1])
```

In [ ]:
```
Y_test
```

Out[ ]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1,
0, 0, 0, 1, 1,
       1, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0,
0, 0, 0, 0, 1,
       0, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
0, 1, 0, 1, 0,
       1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1])
```

In [ ]:
```
tstSc = accuracy_score(Y_pred, Y_test) # store test score
tstSc
```

Out[ ]:
```
0.7662337662337663
```

In [ ]:
```
trSc = log.score(X_train, Y_train) # store train scrore
trSc
```

Out[ ]:
```
0.7791411042944786
```

In [ ]:
```
confusion_matrix(Y_pred, Y_test)
```

Out[ ]:
```
array([[41, 11],
       [ 7, 18]])
```

In [ ]:
```
print(classification_report(Y_pred, Y_test))
```

```
              precision    recall  f1-score   support

           0       0.85      0.79      0.82        52
           1       0.62      0.72      0.67        25

    accuracy                           0.77        77
   macro avg       0.74      0.75      0.74        77
weighted avg       0.78      0.77      0.77        77
```

In [ ]:
```
# cross_val_score(log, X, Y, cv=kfold, scoring='accuracy').
mean()
```

In [ ]:
```
temp = pd.DataFrame([["Logistic Regression", trSc, tstSc]],
columns=['Algorithm used', 'Train Score', 'Test Score'])
acc_stats = pd.concat([acc_stats, temp], sort=False, ignore
_index=True)
```

Inference:

Train Score : 75

Test Score : 77

Model 2: Naive Bayes (Gaussian)

In [ ]:
```
nvclassifier = GaussianNB()
nvclassifier.fit(X_train, Y_train)
y_pred = nvclassifier.predict(X_test)
tstSc = accuracy_score(y_pred, Y_test)
trSc = nvclassifier.score(X_train, Y_train)
print("Test Score: ", tstSc, "\nTrain Score: ", trSc)
```

```
Test Score:  0.7662337662337663
Train Score:  0.7484662576687117
```

In [ ]:
```
confusion_matrix(y_pred, Y_test)
```

Out[ ]:
```
array([[41, 11],
       [ 7, 18]])
```

In [ ]:
```
print(classification_report(y_pred, Y_test))
```

```
              precision    recall  f1-score   support

           0       0.85      0.79      0.82        52
           1       0.62      0.72      0.67        25

    accuracy                           0.77        77
   macro avg       0.74      0.75      0.74        77
weighted avg       0.78      0.77      0.77        77
```

In [ ]:
```
temp = pd.DataFrame([["Naive Bayes Classifier (Gaussian)", trSc, tstSc]], columns=['Algorithm used', 'Train Score', 'Test Score'])
acc_stats = pd.concat([acc_stats, temp], sort=False, ignore_index=True)
```

Model 3: SVM

Using Linear Kernel

In [ ]:
```
clf = svm.SVC(kernel='linear')
clf.fit(X_train, Y_train)
y_pred = clf.predict(X_test)
tstSc = accuracy_score(y_pred, Y_test)
trSc = clf.score(X_train, Y_train)
print("Test Score: ", tstSc, "\nTrain Score: ", trSc)
```

```
Test Score:  0.7922077922077922
Train Score:  0.7714723926380368
```

In [ ]:
```
temp = pd.DataFrame([["SVM (Linear Kernel)", trSc, tstSc]], columns=['Algorithm used', 'Train Score', 'Test Score'])
acc_stats = pd.concat([acc_stats, temp], sort=False, ignore_index=True)
```

Inference: 76 76

Using Polynomial Kernel

In [ ]:
```
clf = svm.SVC(kernel='poly', C=0.1)
clf.fit(X_train, Y_train)
y_pred = clf.predict(X_test)
tstSc = accuracy_score(y_pred, Y_test)
trSc = clf.score(X_train, Y_train)
print("Test Score: ", tstSc, "\nTrain Score: ", trSc)
```

```
Test Score:  0.6883116883116883
Train Score:  0.74079754601227
```

In [ ]:
```
temp = pd.DataFrame([["SVM (Polynomial Kernel)", trSc, tstSc]], columns=['Algorithm used', 'Train Score', 'Test Score'])
acc_stats = pd.concat([acc_stats, temp], sort=False, ignore_index=True)
```

Using Radial Kernel

In [ ]:
```
clf = svm.SVC(kernel='rbf')
clf.fit(X_train, Y_train)
y_pred = clf.predict(X_test)
tstSc = accuracy_score(y_pred, Y_test)
trSc = clf.score(X_train, Y_train)
print("Test Score: ", tstSc, "\nTrain Score: ", trSc)
```

```
Test Score:  0.8051948051948052
Train Score:  0.8205521472392638
```

In [ ]:
```
temp = pd.DataFrame([["SVM (Radial Kernel)", trSc, tstSc]], columns=['Algorithm used', 'Train Score', 'Test Score'])
acc_stats = pd.concat([acc_stats, temp], sort=False, ignore_index=True)
```

In [ ]:
```
print("Confusion matrix:\n", confusion_matrix(y_pred, Y_test))
print(classification_report(y_pred, Y_test))
```

```
Confusion matrix:
 [[45 12]
 [ 3 17]]
              precision    recall  f1-score   support

           0       0.94      0.79      0.86        57
           1       0.59      0.85      0.69        20

    accuracy                           0.81        77
   macro avg       0.76      0.82      0.78        77
weighted avg       0.85      0.81      0.81        77
```

Inference: This model has the best accuracy, as we have compared later. Analysis of Confusion matrix:

- Precision and recall values of 0 (no diabetes) is significantly more than 1.
  Thus, the model can make better prediction that a person does **NOT have Diabetes.**

Model 4: Decision Tree

In [ ]:
```
model=DecisionTreeClassifier(criterion='entropy',splitter='best',random_state=1, min_samples_split=0.1)
model.fit(X_train,Y_train)
```

Out[ ]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=None, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=0.1,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=1, splitter='best')
```

In [ ]:
```
y_pred=model.predict(X_test)
tstSc = accuracy_score(y_pred, Y_test)
trSc = model.score(X_train, Y_train)
print("Test Score: ", tstSc, "\nTrain Score: ", trSc)
print("Conf matirx:\n", confusion_matrix(y_pred, Y_test))
print(classification_report(y_pred, Y_test))
```

```
Test Score:  0.8051948051948052
Train Score:  0.8128834355828221
Conf matirx:
 [[41  8]
 [ 7 21]]
              precision    recall  f1-score   support

           0       0.85      0.84      0.85        49
           1       0.72      0.75      0.74        28

    accuracy                           0.81        77
   macro avg       0.79      0.79      0.79        77
weighted avg       0.81      0.81      0.81        77
```

In [ ]:
```
temp = pd.DataFrame([["Decision Tree", trSc, tstSc]], colum
ns=['Algorithm used', 'Train Score', 'Test Score'])
acc_stats = pd.concat([acc_stats, temp], sort=False, ignore
_index=True)
```

Model 5: Random Forest

In [ ]:
```
model=RandomForestClassifier(n_estimators=150,criterion='en
tropy',random_state=1, min_samples_split=0.1)
model.fit(X_train,Y_train)
y_pred=model.predict(X_test)
tstSc = accuracy_score(y_pred, Y_test)
trSc = model.score(X_train, Y_train)
print("Test Score: ", tstSc, "\nTrain Score: ", trSc)
print("Conf matirx:\n", confusion_matrix(y_pred, Y_test))
print(classification_report(y_pred, Y_test))
```

```
Test Score:  0.7922077922077922
Train Score:  0.8236196319018405
Conf matirx:
 [[44 12]
 [ 4 17]]
              precision    recall  f1-score   support

           0       0.92      0.79      0.85        56
           1       0.59      0.81      0.68        21

    accuracy                           0.79        77
   macro avg       0.75      0.80      0.76        77
weighted avg       0.83      0.79      0.80        77
```

In [ ]:
```
temp = pd.DataFrame([["Random Forest", trSc, tstSc]], colum
ns=['Algorithm used', 'Train Score', 'Test Score'])
acc_stats = pd.concat([acc_stats, temp], sort=False, ignore
_index=True)
```

Conclusion

Comparing accuracies to select the best model

In [ ]:
```
acc_stats
```

Out[ ]:

	Algorithm used	Train Score	Test Score
0	Logistic Regression	0.779141	0.766234
1	Naive Bayes Classifier (Gaussian)	0.748466	0.766234
2	SVM (Linear Kernel)	0.771472	0.792208
3	SVM (Polynomial Kernel)	0.740798	0.688312
4	SVM (Radial Kernel)	0.820552	0.805195
5	Decision Tree	0.812883	0.805195
6	Random Forest	0.823620	0.792208

Conclusion:

In NaiveBayes and SVM(linear) we are getting underfitting, so we do not consider those two.
From remaining, we can see SVM(radial) and DecTree have best Test Score.
Among those two, SVM(Radial) has slightly better score.
Therefore,

Best model: SVM (Radial kernel)
Achieved accuracy: 82.05%

However, if we look at the classification reports the best balanced model is **Decision Tree**
Achieved accuracy: 81.28%
with **f1 score:**

- 0=> 85
- 1 =>74

Step 9: Further analysis on the data and visualisation

Clustering - Unsurpervised Learning

In [ ]:
```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters = 2, random_state = 0)
```

In [ ]:
```
kmeans.fit(X)
```

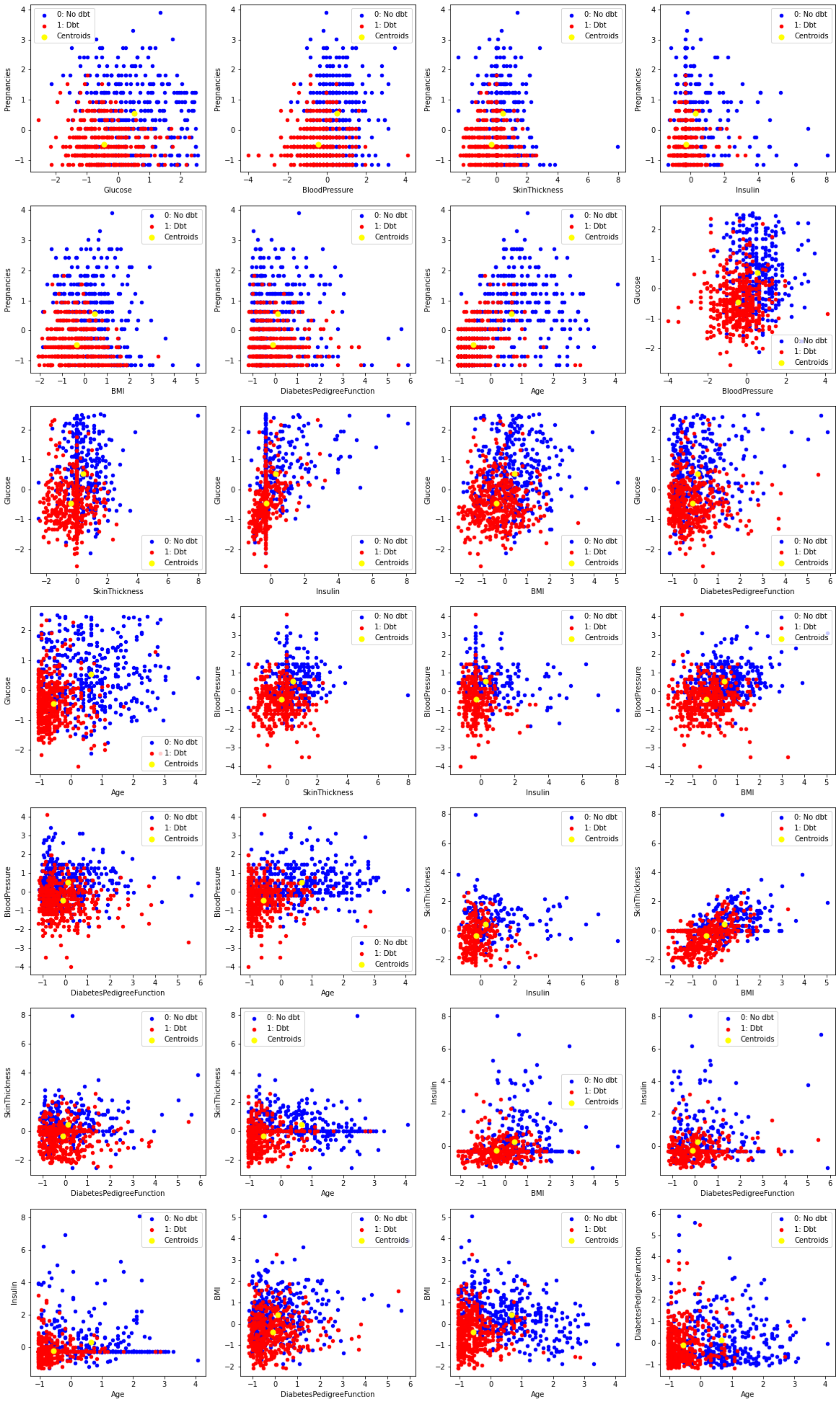
Out[ ]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max
_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_dis
tances='auto',
       random_state=0, tol=0.0001, verbose=0)
```

In [ ]:
```
y_pred = kmeans.predict(X)
```

In [ ]:
```
# y_pred
```

In [ ]:
```
# Y
```

In [ ]:
```
fig, axs = plt.subplots(ncols=4, nrows=7, figsize=(20, 35))
index = 0
axs = axs.flatten()
for i in range(len(X[0])):
    for j in range(i+1,len(X[0])):
        axs[index].scatter(X[y_pred == 0, j], X[y_pred == 0, i], s = 20, c = 'blue', label = '0: No dbt')
        axs[index].scatter(X[y_pred == 1, j], X[y_pred == 1, i], s = 20, c = 'red', label = '1: Dbt')
        axs[index].scatter(kmeans.cluster_centers_[:,j], kmeans.cluster_centers_[:,i], s = 50, c = 'yellow', label = 'Centroids')
        axs[index].legend()
        axs[index].set_xlabel(diabetes2.columns[j])
        axs[index].set_ylabel(diabetes2.columns[i])
        index+=1
```


0: No dbt
1: Dbt
Centroids
Pregnancies
Glucose
BloodPressure
SkinThickness
Insulin
BMI
DiabetesPedigreeFunction
Age

Inference:
From the scatter plot of clustering we can conclude that if we are given values for following pair of features we can cluster the data into 2 groups (Diabetes and No diabetes):

- *Glucose-Age*
- *Blood Pressure-Age*
- *Pregnancies-Age*
- *Pregnancies-Blood Pressure*

Age and Diabetes

In [ ]:
```
pd.DataFrame(diabetes1.groupby(['Age', 'Outcome'])['Outcome
'].count())
```

Out[ ]:

		Outcome
Age	**Outcome**	
21	**0**	58
	1	5
22	**0**	61
	1	11
23	**0**	31
...	**...**	...
68	**0**	1
69	**0**	2
70	**1**	1
72	**0**	1
81	**0**	1

96 rows × 1 columns

Inference: *The age has several discrete values for that we can divide it into ranges. Automatic dividing age into range is done using cut*

In [ ]:
```
diabetes3 = diabetes1.copy(deep=True)
diabetes3['AgeBand'] = pd.cut(diabetes1['Age'], 8)
```

In [ ]:
```
diabetes3[['AgeBand', 'Outcome']].groupby('AgeBand', as_index=False).agg({'Outcome': ['sum','count']})
# how many have diabetes in that range
```

Out[ ]:

	AgeBand	Outcome	
		sum	count
0	(20.94, 28.5]	71	367
1	(28.5, 36.0]	70	147
2	(36.0, 43.5]	56	113
3	(43.5, 51.0]	38	68
4	(51.0, 58.5]	22	38
5	(58.5, 66.0]	9	26
6	(66.0, 73.5]	2	8
7	(73.5, 81.0]	0	1

In [ ]:
```
diabetes3['AgeBand'] = 0
```

Age is divided manually into age bands taking 10 years as range.
For easier realisation and interpretation of data

In [ ]:
```
diabetes3.loc[diabetes3['Age'] <= 30, 'AgeBand'] = 0
diabetes3.loc[(diabetes3['Age'] > 30) & (diabetes3['Age'] <= 40), 'AgeBand'] = 1
diabetes3.loc[(diabetes3['Age'] > 40) & (diabetes3['Age'] <= 50), 'AgeBand'] = 2
diabetes3.loc[(diabetes3['Age'] > 50) & (diabetes3['Age'] <= 60), 'AgeBand'] = 3
diabetes3.loc[diabetes3['Age'] > 60, 'AgeBand'] = 4
```

In [ ]:
```
diabetes3
```

Out[ ]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Diabetes
0	6	148.0	72.0	35.00000	105	33.6	
1	1	85.0	66.0	29.00000	105	26.6	
2	8	183.0	64.0	29.15342	105	23.3	
3	1	89.0	66.0	23.00000	94	28.1	
4	0	137.0	40.0	35.00000	168	43.1	
...	...	...	...	...	...	...	
763	10	101.0	76.0	48.00000	180	32.9	
764	2	122.0	70.0	27.00000	105	36.8	
765	5	121.0	72.0	23.00000	112	26.2	
766	1	126.0	60.0	29.15342	105	30.1	
767	1	93.0	70.0	31.00000	105	30.4	

768 rows × 10 columns

In [ ]:
```
ageb_outcome_count = diabetes3.groupby(['AgeBand', 'Outcome
'])['Outcome'].count()
```

In [ ]:
```
ageb_count = diabetes3.groupby(['AgeBand'])['Outcome'].coun
t()
```

In [ ]:
```
ageb_analysis = ageb_outcome_count.div(ageb_count, level='A
geBand') * 100
ageb_analysis
```

Out[ ]:
```
AgeBand  Outcome
0        0          78.417266
         1          21.582734
1        0          51.592357
         1          48.407643
2        0          43.362832
         1          56.637168
3        0          42.592593
         1          57.407407
4        0          74.074074
         1          25.925926
Name: Outcome, dtype: float64
```

In [ ]:
```
ageb_analysis[:,1]
```

Out[ ]:
```
AgeBand
0    21.582734
1    48.407643
2    56.637168
3    57.407407
4    25.925926
Name: Outcome, dtype: float64
```

Inference:
On comparing the relative percentages of people having diabetes in the different age groupps, we can conclude that:
Most prone age group is group 3 that corresponds to *(40, 50]*