```
import numpy as np
import pandas as pd
import matplotlib as plt
%matplotlib inline
import seaborn as sns
```

```
df = pd.read_csv('diabetes.csv')
df.head()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFu
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	

```
df.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
```

```
df.describe()
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	

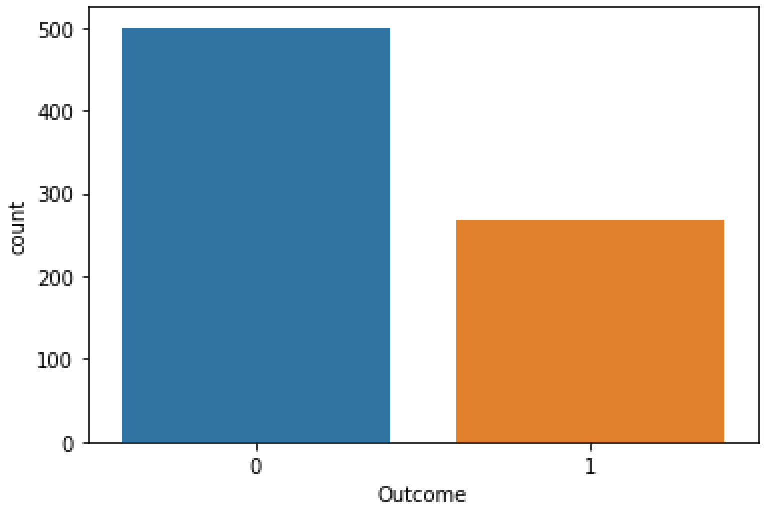
```
df.shape
```

```
(768, 9)
```

```
df.Outcome.value_counts()
```

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

```
sns.countplot(x='Outcome' , data =df);
```



```
df.dtypes
```

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

```
df.describe()
```

```
#There are incorrect values i.e 0's in Glucose, BloodPressure, SkinThickness, Insulin, BMI
```

```
#There are incorrect values i.e.0's in Glucose, BloodPressure, SkinThickness, Insulin, BMI.
# replacing 0 with median of corresponding column.
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
count	768.000000	768.000000	768.000000	768.000000	768.000000	768.000000	
mean	3.845052	120.894531	69.105469	20.536458	79.799479	31.992578	
std	3.369578	31.972618	19.355807	15.952218	115.244002	7.884160	
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	
25%	1.000000	99.000000	62.000000	0.000000	0.000000	27.300000	
50%	3.000000	117.000000	72.000000	23.000000	30.500000	32.000000	
75%	6.000000	140.250000	80.000000	32.000000	127.250000	36.600000	
max	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	

```
dataframe_temp = df.drop(["Pregnancies","Outcome"],axis = 1)
dataframe_temp
medians = dataframe_temp.median()
print("medians",medians)
dataframe_nonzero = dataframe_temp.replace(0,medians)
dataframe_nonzero["Pregnancies"] = df["Pregnancies"]
dataframe_nonzero["Outcome"] = df["Outcome"]
dataframe_nonzero
```

```
medians Glucose                  117.0000
BloodPressure             72.0000
SkinThickness             23.0000
Insulin                   30.5000
BMI                       32.0000
DiabetesPedigreeFunction   0.3725
```

```
corr = dataframe_nonzero.corr()
corr
```

	Glucose	BloodPressure	SkinThickness	Insulin	BMI	Dia
Glucose	1.000000	0.218937	0.172143	0.357573	0.231400	
BloodPressure	0.218937	1.000000	0.147809	-0.028721	0.281132	
SkinThickness	0.172143	0.147809	1.000000	0.238188	0.546951	
Insulin	0.357573	-0.028721	0.238188	1.000000	0.189022	
BMI	0.231400	0.281132	0.546951	0.189022	1.000000	
DiabetesPedigreeFunction	0.137327	-0.002378	0.142977	0.178029	0.153506	
Age	0.266909	0.324915	0.054514	-0.015413	0.025744	
Pregnancies	0.128213	0.208615	0.032568	-0.055697	0.021546	
Outcome	0.492782	0.165723	0.189065	0.148457	0.312249	

Takeaway : outcome is positively corelated to Glucose feature.

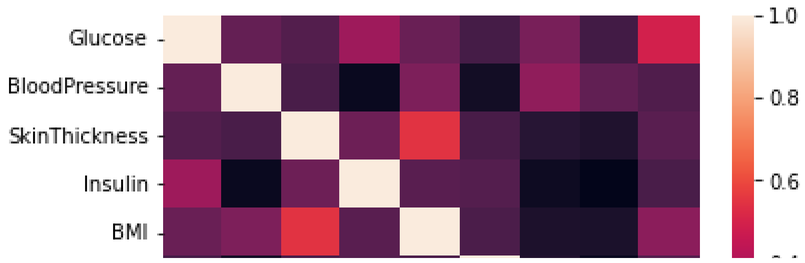
Age & no. of pregencies have positive corelation.

BMI & Skin thickness has positive corelation

No other strong negetive corelation is observed.

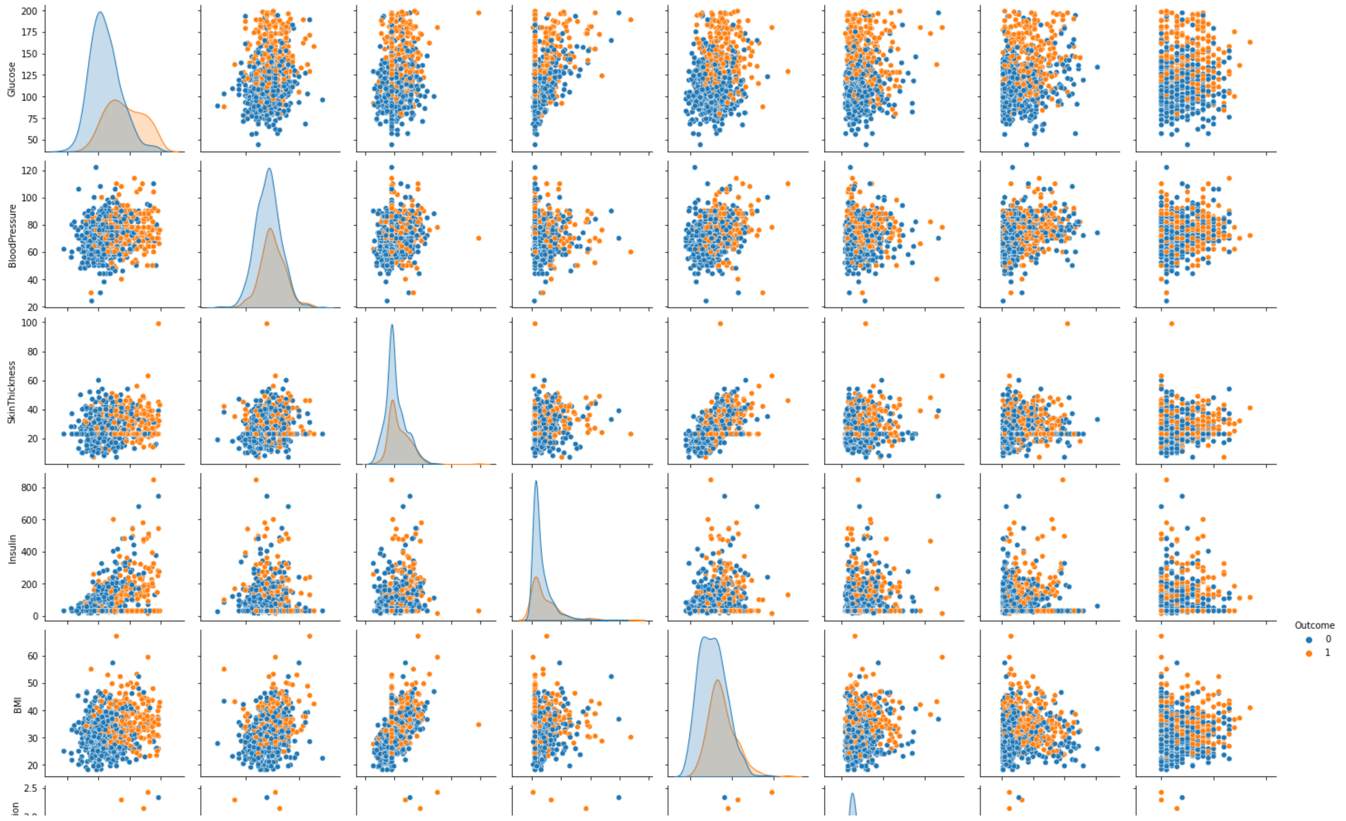
```
sns.heatmap(corr)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f945b12a150>
```



```
sns.pairplot(dataframe_nonzero, diag_kind='kde', hue="Outcome") # plotting pairplot
```

```
<seaborn.axisgrid.PairGrid at 0x7f9458409390>
```



```
from sklearn.model_selection import train_test_split
X = dataframe_nonzero.drop('Outcome', axis=1)
Y = dataframe_nonzero['Outcome']
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30, random_state=1)
```

Training Support vector Machines

```
from sklearn import svm
from sklearn.svm import SVC

clf = svm.SVC(C = 100,gamma= "scale")
clf.fit(X_train,Y_train)
```

```
SVC(C=100)
```

```
score1 = clf.score(X_test,Y_test)
score1
```

```
0.7705627705627706
```

```
from sklearn import metrics
Y_pred = clf.predict(X_test)
print( metrics.confusion_matrix(Y_test,Y_pred))
```

```
[[131  15]
```

```
      [ 38  47]]
```

Scaling the datapoints using MinMax Scalar

```
from sklearn.preprocessing import MinMaxScaler
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.fit_transform(X_test)

#Zscore
from scipy.stats import zscore
X_train_z = X_train.apply(zscore) # converting to Z score
X_test_z = X_test.apply(zscore)
```

```
# Model score on Minmax scaled values
clf = svm.SVC(C = 10,gamma= "scale")
clf.fit(X_train_scaled,Y_train)
score2 = clf.score(X_test_scaled,Y_test)
score2
```

```
     0.7532467532467533
```

```
# Model score using zscore  values
clf = svm.SVC(C = 10,gamma= "scale")
clf.fit(X_train_z,Y_train)
score3 = clf.score(X_test_z,Y_test)
score3
```

```
     0.7272727272727273
```

We can try increasing either C or gamma to fit a more complex model.

```
clf = svm.SVC(C = 1000,gamma= "scale")
clf.fit(X_train,Y_train)
score4 = clf.score(X_test,Y_test)
print("Model score for non-scaled datapoints", score4)

# model accuracy has increased on non-scaled data,however for scaled values with c = 1000, mo
```

```
     Model score for non-scaled datapoints 0.7835497835497836
```

```
import multiprocessing
from sklearn.model_selection import GridSearchCV
```

```
param_grid = [    {
     'kernel': ['linear', 'rbf', 'poly', 'sigmoid'],
     'C': [ 0.1, 0.2, 0.4, 0.5, 1.0, 1.5, 1.8, 2.0, 2.5, 3.0 ]    } ]

gs = GridSearchCV(estimator=SVC(), param_grid=param_grid,scoring='accuracy', cv=10, n_jobs=mu
```

```
gs.fit(X_train_scaled, Y_train)
```

```
     GridSearchCV(cv=10, estimator=SVC(), n_jobs=2,
                  param_grid=[{'C': [0.1, 0.2, 0.4, 0.5, 1.0, 1.5, 1.8, 2.0, 2.5,
                                     3.0],
                               'kernel': ['linear', 'rbf', 'poly', 'sigmoid']}],
                  scoring='accuracy')
```

```
gs.best_estimator_
```

```
     SVC(kernel='linear')
```

```
gs.best_score_
```

```
     0.7653738644304683
```

Calculate AUC score and plot ROC curve

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

```
auc = roc_auc_score(Y_test,Y_pred)
print("AUC %0.3f" %auc)
```

```
     AUC 0.725
```

AUC measures how true positive rate (recall) and false positive rate trade off

```
from sklearn import model_selection
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
```

```
seed = 7
models = []
models.append(('LR', LogisticRegression()))
models.append(('LDA', LinearDiscriminantAnalysis()))
models.append(('KNN', KNeighborsClassifier()))
models.append(('CART', DecisionTreeClassifier()))
models.append(('NB', GaussianNB()))
models.append(('SVM', SVC()))
models.append(('RFC', RandomForestClassifier()))
results = []
names = []
scoring = 'accuracy'
import warnings
warnings.filterwarnings("ignore")
for name, model in models:
    kfold = model_selection.KFold(n_splits=10, random_state=None)
    cv_results = model_selection.cross_val_score(model, X_train, Y_train,
cv=kfold, scoring=scoring)
    results.append(cv_results)
    names.append(name)
    msg = "%s: %f (%f)" % (name, cv_results.mean(), cv_results.std())
    print(msg)
```

```
LR: 0.763347 (0.061653)
LDA: 0.770790 (0.062164)
KNN: 0.696261 (0.085826)
CART: 0.683368 (0.060197)
NB: 0.730119 (0.054511)
SVM: 0.744794 (0.054232)
RFC: 0.757757 (0.055708)
```

Linear Discriminant Analysis is giving a better accuracy of 77% as compared with the other models.

✓ 3s completed at 12:28