```
#!pip install plotly

# Required Packages

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

from sklearn import datasets, linear_model

from mpl_toolkits.mplot3d import axes3d

import seaborn as sns

import plotly.plotly as py

from sklearn.preprocessing import scale

import sklearn.linear_model as skl_lm

from sklearn.metrics import mean_squared_error, r2_score

#For LR

import statsmodels.api as sm

#For LR That looks like R

import statsmodels.formula.api as smf

from statsmodels.graphics.mosaicplot import mosaic

from sklearn.decomposition import PCA

from sklearn.preprocessing import scale

print("Packages LOADED")




import os

print(os.getcwd())
```

```
os.chdir('E:\\Locker\\Sai\\SaiHCourseNait\\DecBtch\\R_Datasets\\')
print(os.getcwd())


data = pd.read_csv('diabetes2.csv')


data.info()


get_ipython().magic('matplotlib inline')
sns.boxplot(data.Outcome,data.Glucose)


sns.boxplot(data.Outcome,data.BloodPressure)


sns.boxplot(data.Outcome,data.SkinThickness)


sns.boxplot(data.Outcome,data.Insulin)


sns.boxplot(data.Outcome,data.BMI)


sns.boxplot(data.Outcome,data.DiabetesPedigreeFunction)


sns.boxplot(data.Outcome,data.Age)


data_n=data[['Glucose','Age','DiabetesPedigreeFunction','BMI','Insulin','SkinThickness','BloodPressure']]
sns.pairplot(data_n)
```

```
corr = data.corr()

print(corr)

print('-'*30)

mask = np.zeros_like(corr, dtype=np.bool)

print(mask)

print('-'*30)

mask[np.triu_indices_from(mask)] = True

f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap

cmap = sns.diverging_palette(220, 10, as_cmap=True)

sns.heatmap(corr, mask=mask, cmap=cmap, vmax=.3,

            square=True,

            linewidths=.5, cbar_kws={"shrink": .5}, ax=ax)
```

```
colormap = plt.cm.viridis

plt.figure(figsize=(12,12))

plt.title('Pearson Correlation of Features', y=1.05, size=15)

sns.heatmap(data.corr(),linewidths=0.1,vmax=1.0, square=True, cmap=colormap, linecolor='white',
annot=True)
```

```
data.describe()


truediabetes= data.loc[data['Outcome']==1]


truediabetes.head()


len(truediabetes)


truediabetes.mean()


falsediabetes= data.loc[data['Outcome']==0]


len(falsediabetes)


falsediabetes.mean()



%matplotlib inline

import matplotlib.pyplot as plt

import seaborn as sb

sb.pairplot(data.dropna(), hue='Outcome', palette="husl")
```

```
import numpy as np

import seaborn as sns

cols =
['Pregnancies','Glucose','BloodPressure','SkinThickness','Insulin','BMI','DiabetesPedigreeFunction','Age',
'Outcome']

cm = np.corrcoef(data[cols].values.T)

sns.set(font_scale=1.5)

hm = sns.heatmap(cm,cbar=True,annot=True,square=True,fmt='.2f',annot_kws={'size':
15},yticklabels=cols,xticklabels=cols)

plt.show()
```

```
data.loc[data['Outcome'] == 0, 'Glucose'].hist()
```

```
data.loc[data['Outcome']==1, 'Glucose'].hist()
```

```
data.loc[data['Outcome']==0, 'Insulin'].hist()
```

```
data.loc[data['Outcome']==1, 'Insulin'].hist()
```

```
data.loc[data['Outcome']==1, 'BMI'].hist()
data.loc[data['Outcome']==0, 'BMI'].hist()
```

```
data.loc[data['Outcome']==1, 'Age'].hist()
```

```
data.loc[data['Outcome']==0, 'Age'].hist()
```

```
plt.figure(figsize=(20, 20))
for column_index, column in enumerate(falsediabetes.columns):
    if column == 'Outcome':
        continue
    plt.subplot(4, 4, column_index + 1)
    sb.violinplot(x='Outcome', y=column, data=falsediabetes)
```

```
plt.figure(figsize=(20, 20))
for column_index, column in enumerate(truediabetes.columns):
    if column == 'Outcome':
        continue
    plt.subplot(4, 4, column_index + 1)

    sb.violinplot(x='Outcome', y=column, data=truediabetes)





# class distribution

print(" == class distribution  ==")

print(data.groupby('Outcome').size())





print(" == Univariate Plots: box and whisker plots. determine outliers = ")

data.plot(kind='box', subplots=True, layout=(3,3), sharex=False, sharey=False)

plt.show()



print(" == Univariate Plots: histograms. determine if the distribution is normal-like == ")

data.hist()

plt.show()
```

```
import pandas

#from pandas.plotting import scatter_matrix

print("== Multivariate Plots: scatter plot matrix. spot structured relationships between input variables
==")

#scatter_matrix(data)

plt.show()

#***


#!pip install pydot

#import pandas as pd

#import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import re

#import xgboost as xgb

import pydot

from IPython.display import Image

from sklearn.cross_validation import train_test_split, cross_val_score

from sklearn.externals.six import StringIO

from sklearn.tree import DecisionTreeRegressor, DecisionTreeClassifier, export_graphviz

from sklearn.ensemble import BaggingClassifier, RandomForestClassifier, BaggingRegressor,
RandomForestRegressor, GradientBoostingRegressor

from sklearn.metrics import confusion_matrix, classification_report, mean_squared_error

import plotly.offline as py

py.init_notebook_mode(connected=True)

import plotly.graph_objs as go
import plotly.tools as tls

from sklearn import tree

from sklearn.metrics import accuracy_score

from sklearn.cross_validation import KFold

from sklearn.cross_validation import cross_val_score

from IPython.display import Image as PImage

from subprocess import check_call

from PIL import Image, ImageDraw, ImageFont

pd.set_option('display.notebook_repr_html', False)

get_ipython().magic('matplotlib inline')

plt.style.use('seaborn-white')

print("Package Loaded")



#*** Logistic Reg


import sklearn

array = data.values

array

type(array)


X = array[:,0:8] # ivs for train

X


y = array[:,8] # dv
```

```
y

test_size = 0.33

from sklearn.model_selection import train_test_split
#pip install -U scikit-learn
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=test_size)
print('Partitioning Done!')



regr = skl_lm.LogisticRegression()

regr.fit(X_train, y_train)

pred = regr.predict(X_test)

regr.score(X_test,y_test)

cm_df = pd.DataFrame(confusion_matrix(y_test, pred).T, index=regr.classes_,
                     columns=regr.classes_)
cm_df.index.name = 'Predicted'
cm_df.columns.name = 'True'
print(cm_df)
print(classification_report(y_test, pred))
```

```
regr.score(X_test,y_test)
```

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

fpr, tpr, _ = roc_curve(y_test, pred)

# Calculate the AUC

roc_auc = auc(fpr, tpr)

print('ROC AUC: %0.2f' % roc_auc)

# Plot of a ROC curve for a specific class

plt.figure()

plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)

plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05])

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.legend(loc="lower right")

plt.show()

regr.score(X_test,y_test)
```

```
'''

train, test = sklearn.cross_validation.train_test_split(data, train_size = 0.7)

print("For Main Data Set :",data["Outcome"].count())

print("For Train Set :",train["Outcome"].count())

print("For Test Set :",test["Outcome"].count())

x_train=train[['Glucose','Age','DiabetesPedigreeFunction','BMI','Insulin','SkinThickness','BloodPressure','
Pregnancies']]

x_test=test[['Glucose','Age','DiabetesPedigreeFunction','BMI','Insulin','SkinThickness','BloodPressure','Pr
egnancies']]

y_train=train["Outcome"]

y_test=test["Outcome"]




est = smf.Logit(y_train,x_train).fit()

est.summary()


regr = skl_lm.LogisticRegression()

regr.fit(x_train, y_train)

pred = regr.predict(x_test)
```

```
cm_df = pd.DataFrame(confusion_matrix(y_test, pred).T, index=regr.classes_,
                     columns=regr.classes_)
cm_df.index.name = 'Predicted'
cm_df.columns.name = 'True'
print(cm_df)
print(classification_report(y_test, pred))
```

```
from sklearn.metrics import roc_curve, auc, roc_auc_score, cohen_kappa_score
fpr, tpr, _ = roc_curve(y_test, pred)
# Calculate the AUC
roc_auc = auc(fpr, tpr)
print('ROC AUC: %0.2f' % roc_auc)
# Plot of a ROC curve for a specific class
plt.figure()
plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```

```
regr.score(x_test,y_test)

'''
```