Predicting Breast Cancer Outcome of a new Patient

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn_pandas import DataFrameMapper
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix


orig_dataset=pd.read_excel('BreastCancer_Prognostic_v1.xlsx',na_values = "?",
sep=",")
dataset = orig_dataset.copy()

dataset.isna().sum()
'''
Checking the presence of NA values and dropping them we use
dataset.isna().sum()
'''


dataset = dataset.dropna()
y = dataset['Outcome']
dropped_params = ['ID', 'Time', 'Outcome']
dataset = dataset.drop(dropped_params, 1)

dataset.describe()
'''
to view statistics related to data
'''


plot = sns.countplot(y,label="Count")
N, R = y.value_counts()
print('Total Non-Recurring Cases are',N)
```

```
print('Total Recurring Cases are', R)


dataset_without_fe = dataset

fig = plt.subplots(figsize = (32, 32))
sns.set(font_scale=1.6)
sns.heatmap(dataset.corr(),square =
True,cbar=True,annot=True,annot_kws={'size': 10})
plt.savefig("Heat_Map_for_predicting_breast_cancer_outcome.png")




dropped_params =
['texture_mean','perimeter_mean','area_mean','smoothness_mean','compactnes
s_mean','concavity_mean','symmetry_mean',

'radius_std_dev','texture_std_dev','perimeter_std_dev','area_std_dev','smoothne
ss_std_dev','compactness_std_dev',
        'concavity_std_dev','concave_points_std_dev','symmetry_std_dev',

'Worst_texture','Worst_perimeter','Worst_area','Worst_smoothness','Worst_co
mpactness',

'Worst_concavity','Worst_concave_points','Worst_symmetry','Tumor_Size','Lymp
h_Node_Status']

'''
These Dropped parameters are highly correlated variables because
it could introduce a problem of multicollinearity which further has a negative
impact on the accuracy of the model.
'''

featureEngineered_dataset = dataset.drop(dropped_params,axis = 1 )
featureEngineered_dataset.head()
```

```
mapper = DataFrameMapper([(featureEngineered_dataset.columns,
StandardScaler())])
scaled_features = mapper.fit_transform(featureEngineered_dataset.copy(), 4)
scaled_features_df = pd.DataFrame(scaled_features,
index=featureEngineered_dataset.index,
columns=featureEngineered_dataset.columns)

'''
scaled_features_df is the dataset on which feaured engineering
has been performed
'''


scaled_features_df.describe()

'''
Uncomment to save the predictions in an excel file
i=0
'''
def running_and_evaluating_model(x, y):
   '''
   Uncomment to save the predictions in an excel file
   global i
   i=i+1
   '''
   x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,
random_state=10)
   regr = linear_model.LogisticRegression(solver = "lbfgs", max_iter = 3000)
   regr.fit(x_train, y_train)
   y_pred = regr.predict(x_test)
   '''
   Uncomment to save the predictions in an excel file
   df=pd.DataFrame({'y_test':y_test,'y_pred':y_pred})
   s="Heat_Map_for_predicting_breast_cancer_outcome"+str(i)+".xlsx"
   df.to_excel(s)
   '''
   accuracy = regr.score(x_test, y_test)
   print("Accuracy: " , accuracy * 100)
```

```
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm,annot=True,fmt="d")


running_and_evaluating_model(dataset_without_fe, y)

running_and_evaluating_model(scaled_features_df, y)
```

Predicting Recurrence Time for patients whose outcome is R.

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn_pandas import DataFrameMapper
from sklearn import linear_model
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error



orig_dataset=pd.read_excel('BreastCancer_Prognostic_v1.xlsx',na_values = "?",
sep=",")
dataset = orig_dataset.copy()

dataset.isna().sum()

dataset = dataset.dropna()

dataset = dataset[dataset['Outcome'] == 'R']
'''
Using dataset which have 'R' as outcome
'''
dataset = dataset.drop('Outcome', 1)
dataset = dataset.drop('ID', 1)

dataset.describe()

dataset_without_fe = dataset
dataset_without_fe = dataset_without_fe.drop('Time', axis = 1)
```

```
map = plt.subplots(figsize = (30, 30))
sns.set(font_scale=1.6)
sns.heatmap(dataset.corr(),square =
True,cbar=True,annot=True,annot_kws={'size': 10})
plt.savefig("Heat_Map_for_predicting_recurrence_time.png")
'''
saving the heat map to analyse it in more detail
'''


single_attribute_dataset = dataset[['area_mean']]
'''
apart from area_mean we can any other attribute to find the single attribute
result
'''



drop_Items =
['texture_mean','perimeter_mean','area_mean','smoothness_mean','compactnes
s_mean','concavity_mean','symmetry_mean',

'radius_std_dev','texture_std_dev','perimeter_std_dev','area_std_dev','smoothne
ss_std_dev','compactness_std_dev',
        'concavity_std_dev','concave_points_std_dev','symmetry_std_dev',

'fractal_dimension_std_dev','Worst_texture','Worst_perimeter','Worst_area','Wo
rst_compactness',

'Worst_concavity','Worst_concave_points','Worst_symmetry','Tumor_Size','Lymp
h_Node_Status']
featureEngineered_dataset = dataset.drop(drop_Items,axis = 1 )
featureEngineered_dataset.head()
```

```
Original_Time = featureEngineered_dataset.pop('Time')
mapper = DataFrameMapper([(featureEngineered_dataset.columns,
StandardScaler())])
scaled_features = mapper.fit_transform(featureEngineered_dataset.copy(), 4)
scaled_features_df = pd.DataFrame(scaled_features,
index=featureEngineered_dataset.index,
columns=featureEngineered_dataset.columns)

'''
scaled_features_df is the dataset on which feaured engineering
has been performed
'''



scaled_features_df.describe()
'''
Uncomment to save the output in an excel file
i=0
'''
def running_and_evaluating_model(x, y):
    '''
    Uncomment to save the output in an excel file

    global i
    i=i+1
    '''
    x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.20,
random_state=30)
    regr = linear_model.LinearRegression()
    regr.fit(x_train, y_train)
    y_pred = regr.predict(x_test)
    '''
    Uncomment to save the output in an excel file
    df=pd.DataFrame({'y_test':y_test,'y_pred':y_pred})
```

```
    s="Heat_Map_for_predicting_recurrence_time"+str(i)+".xlsx"
    df.to_excel(s)
    storing predicitons into a excel file
    '''
    print('Attributes Coefficients: \n', regr.coef_)
    print("Mean squared error: %.2f" % mean_squared_error(y_test, y_pred))
    print("Original_Time: \n", np.array(y_test).astype(int))
    print("Predicted values: \n", y_pred.astype(int))



running_and_evaluating_model(dataset_without_fe, Original_Time)
running_and_evaluating_model(single_attribute_dataset, Original_Time)
running_and_evaluating_model(featureEngineered_dataset, Original_Time)
```