ANALYSIS OF SURVIVAL OF LUNG CANCER PATIENTS

ST 4052 – Data Analysis Final Project Group 4

M. D. D. De Costa – s14533

S. P. P. M. Sudasinghe – s14510

K. G. S. K. Wickramasinghe - s14594

Abstract

This report is based on data recorded of patients who faced to the Cardio-thoracic surgery. The objective is to predict survival status of cancer patients who faced to the Cardio-thoracic surgery using the machine learning. The software package that we used to do this analysis was Python. We deployed many machine learning models such as Logistic Regression, Decision Tree, K Nearest Neighbors, Support Vector Machine, Random Forest, and Voting Classifier. Considering accuracy, interpretability and computational cost, we decided to go with the Logistic Regression model. The next step of this analysis is to integrate the predictive model to a data product.

Table of Contents

List of Figures

Introduction

Worldwide, lung cancer is the most common malignancy and the most common cause of cancer deaths in the past few decades thus its five-year survival rate (17.8%) was much lower than that of other leading cancers. Owing to the high fatality rate, it remains to be an important public health issue. So even when considering Poland, lung cancer has been found as the leading cause of cancer death in both men and women. Cardio-thoracic surgery is a surgical instrument for the organs inside the thorax (the chest) which in general involves the treatments for conditions of heart and lungs. Since these organs are severely important, surgeries involving these organs have a high risk. Cardio-thoracic surgery is frequently used to assess or repair lungs affected by cancer, trauma or pulmonary disease.

Objective

To predict survival status of cancer patients who faced to the Cardio-thoracic surgery using the machine learning. The proposed solution should be able to identify and interpret risk factors in classifying survival status of lung cancer patients which can help assist the medical practitioner in their diagnosis.

About the Dataset

The data is dedicated to classification problem related to the cancer patients after a Cardio-thoracic surgery in which there are two classes class 1 and 2, where 1 indicates the death of the patient within one year after the surgery and 2 indicates the patients who survive. The data was collected from a Surgery Centre for patients who underwent major lung resections for primary lung cancer in the years 2007 to 2011.

Variable	Description
id	Id of the patient
diagnose	Diagnosis type of the cancer
in_vol	Maximal volume of gas that can be exhaled from lung (get into lung)
out_vol	Expiratory Volume per second (get out from lung)
pain	Pain before cancer (T- yes,F- no)
Haemoptysis	Haemoptysis before surgery (T- yes,F- no) (A symptom with lung cancer patients)
short_breath	Shortness of breath of patient before
cough	Cough before surgery (T- yes,F- no)

weakness	Weakness before surgery (T- yes,F- no)
tumour_size	Size of the Tumour
diabetes	Diabetes before surgery (T- yes,F- no)
breathing_difficulty	Breathing difficulty before surgery (T- yes,F- no)
other_disease	Other diseases before surgery (T- yes,F- no)
smoking	Smoking before surgery (T- yes,F- no)
asthma	Asthma before surgery (T- yes,F- no)
age	Age of the patient
survive	Survival of the patient after 1 year from Surgery (1- Died, 2- Survived)

Descriptive Analysis

Survival of the patient after 1 year from Surgery

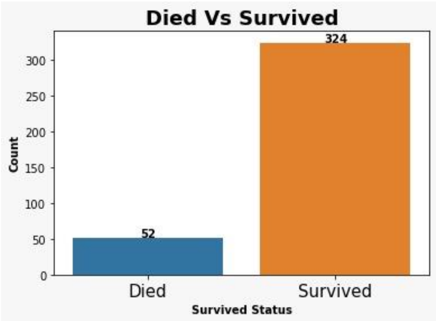

Figure 1

Out of the 376 patients, 52 patients died in the 1 year time and 324 survived, which is a 13.8% death rate. This is the response variable of the dataset. As you can see, this variable is highly unbalanced. Therefore, we should be careful when fitting models to this data.

Diagnosis type of the cancer

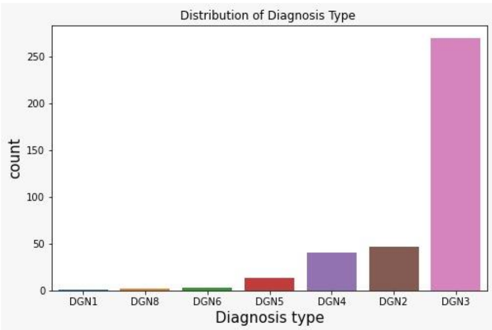

Figure 2

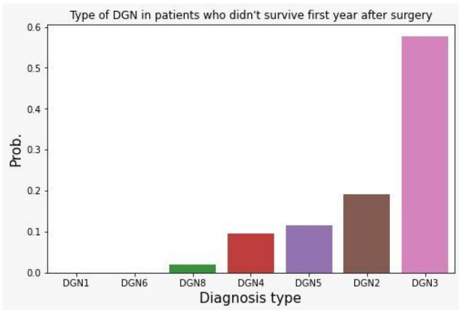

Figure 3

Referring to the first graph, for diagnosis, the large majority of patients are in category DGN3. The other categories are relatively small while category 2,4 and 5 should be considered for their counts in that order. Then the graph in the right side shows the type of diagnosis in patients who didn't survive first year after surgery. As you can see most of the died patients belongs to the category DGN3.

Distributons of 'in_vol' and 'out_vol'

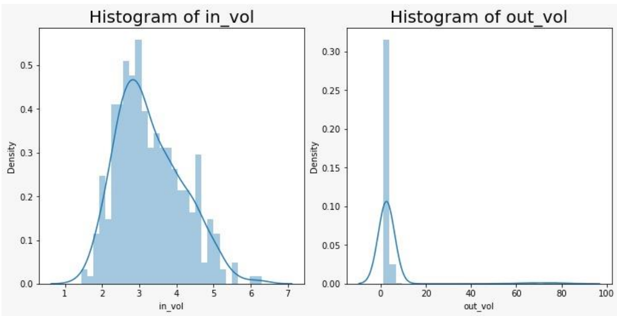

Figure 4

According to the first histogram the average maximal volume of gas that can be exhaled by a patient is around 3. Since this is a right skewed graph, we can say that there were few patients who can exhale more volume of gas. Then from the second graph we can see that majority of the patients lie between 0 and 20 expiratory volume per second.

Diabetes Vs Survival

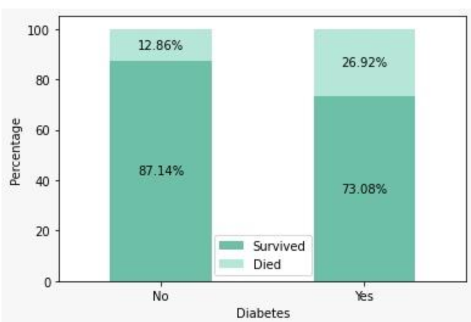

Figure 5

It seems that patients who have diabetes before surgery die more than who are not having diabetes.

Other disease Vs Survival

As you can see, the patients who have other disease have a less chance to be survived than the other patients.

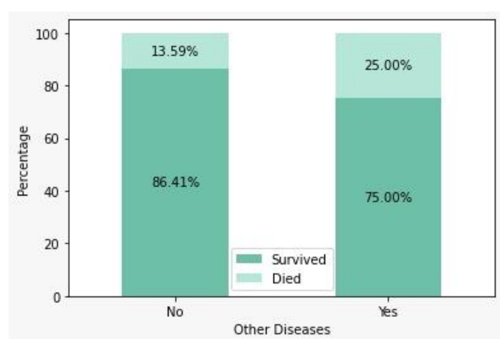

Figure 6

Smoking before surgery

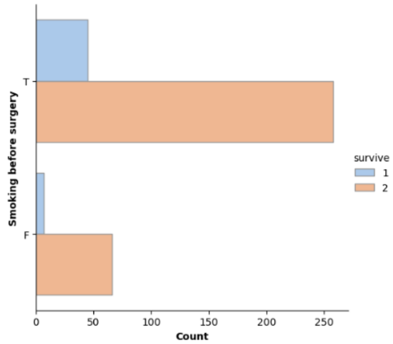

Figure 7

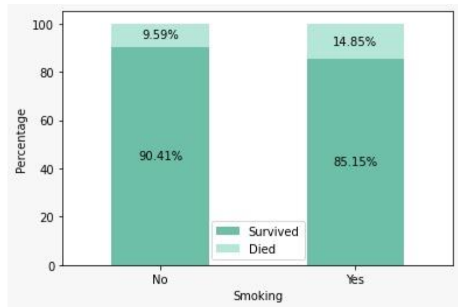

Figure 8

It seems that most of the cancer patients having smoking habits before the surgery. Also, we can see that people who smoked are died after the surgery than who are not having smoking habit. However the difference between died smokers and died non-smokers is considerably small which is around 5%.

Age Vs Survival

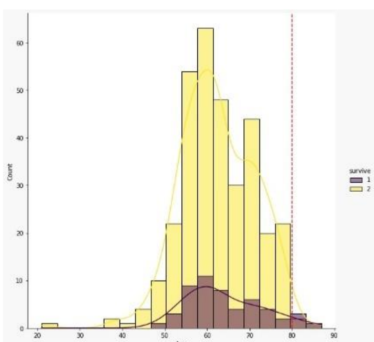

Figure 9

Here we drew two distributions of age, yellow one which represents people who survived and the brown one, people who do not survived. As you can see, patients who are more than 80 years of age has a very high chance of not surviving.

Size of the Tumor

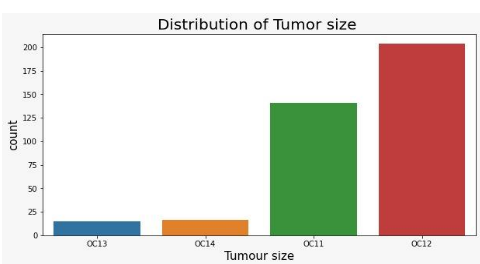

Figure 10

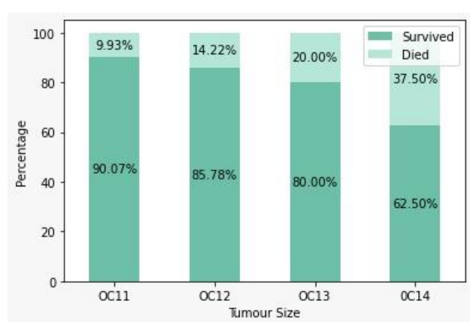

Figure 11

There are 4 sizes of tumor, OC11, OC12, OC13, OC14. Among them most of the patients have OC12 size of tumor and then OC11 size of tumor. The proportion of dead to live generally increases with the tumor size ranging from 0C11 to 0C14, indicating the higher tumor size correlates to higher chance of death even with surgery.

Correlation - Association plot

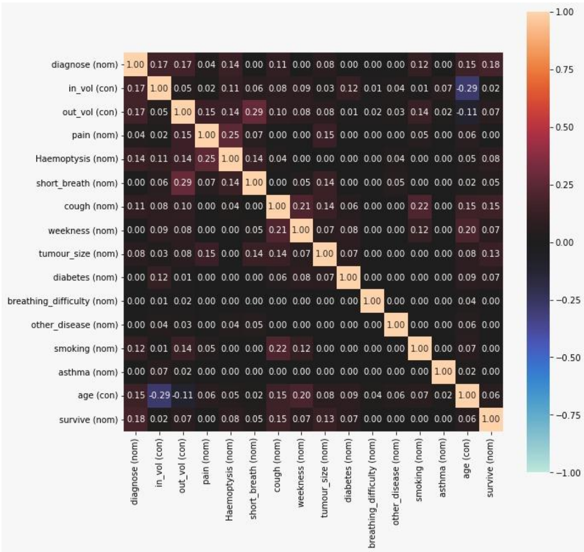

Figure 12

As you can see in the graph below, we can’t see any heavy correlations between these variables. However. there is a small negative correlation between age and Maximal volume of gas that can be exhaled from lungs which is true in real life. And there is a small positive correlation between short breath and expiratory Volume per second (vol_out). Therefore, multicollinearity exists in between these variables.

Important Results of the Advanced Analysis

So now that we have thoroughly explored the patterns and trends in the data set, the next step is to utilize machine learning models to see if how well we can predict the target variable, Survive, with the feature variables.

As we mentioned in the descriptive analysis, the response variable "survive" is highly unbalanced. Therefore, we used the SMOTE (Synthetic Minority Over-sampling Technique) technique in order to balance the dataset.

Also, as we saw in our descriptive analysis, the dataset contains outlier observations. Therefore, we removed those outliers and fitted the models.

For the advanced analysis, we used some mainstream models which are Logistic Regression, Decision Tree, K Nearest Neighbors, Support Vector Machine, Random Forest, and Voting Classifier. Logistic Regression, KNN, SVC, and Voting Classifier did not have an overfitting problem. Random Forest model and decision tree models had an overfitting problem, so we used hyper parameter tuning to get rid of that.

Given below are the results of the models fitted, before balancing the data.

		Logistic Regression	Decision Tree	KNN	SVM	Random Forest	Voting Classifier
ROC AUC	Train	52.7 %	73.8 %	52.5 %	87.3 %	85.1 %	69.6 %
	Test	49.4 %	48.1 %	49.4 %	50.0 %	44.8 %	50.0 %
Accuracy	Train	84.2 %	91.4 %	85.0 %	87.2 %	90.8 %	90.6 %
	Test	84.4 %	82.2 %	84.4 %	85.6 %	76.7 %	85.6 %
Precision	Train	85.2 %	91.2 %	85.1 %	97.4 %	95.6 %	89.9 %
	Test	85.4 %	85.1 %	85.4 %	85.6 %	84.1 %	85.6 %
Recall	Train	98.4 %	99.3 %	99.7 %	87.2 %	93.4 %	100.0 %
	Test	98.7 %	96.1 %	98.7 %	100.0 %	89.6 %	100.0 %
F1 Score	Train	91.3 %	95.1 %	91.8 %	92.0 %	94.5 %	94.7 %
	Test	91.6 %	90.2 %	91.6 %	92.2 %	86.8 %	92.2 %

Table 1

Even though the results were good, and it has a very good predictive power, we used SMOTE balancing technique as mentioned above. Then we got the following results.

		Logistic Regression	Decision Tree	KNN	SVM	Random Forest	Voting Classifier
ROC AUC	Train	54.6 %	89.0 %	75.1 %	87.3 %	85.1 %	87.3 %
	Test	43.5 %	48.6 %	48.5 %	53.8 %	44.8 %	42.2 %
Accuracy	Train	72.5 %	92.5 %	67.8 %	87.2 %	90.8 %	87.2 %
	Test	74.4 %	66.7 %	55.6 %	86.7 %	76.7 %	66.7 %
Precision	Train	86.0 %	96.9 %	96.1 %	97.4 %	95.6 %	97.4 %
	Test	83.8 %	85.1 %	84.9 %	86.5 %	84.1 %	83.1 %
Recall	Train	80.6 %	94.1 %	64.5 %	87.2 %	93.4 %	87.2 %
	Test	87.0 %	85.1 %	58.4 %	100.0 %	89.6 %	76.6 %
F1 Score	Train	83.2 %	95.5 %	77.2 %	92.0 %	94.5 %	92.0 %
	Test	85.4 %	79.2 %	69.2 %	92.8 %	86.8 %	79.7 %

Table 2

By comparison of F1 scores of all the models, we could notice that applying SMOTE methodology results in poor predictability of data. So, we made the decision to go along with models fitted to original data.

Looking at those models, we realized that Logistic Regression, KNN, SVM and Voting Classifier performs exceptionally well (Decision Tree and Random Forest have a big difference in F1 scores, which indicates that the models are overfitted). Out of them, considering interpretability and computational cost, we decided to go with the Logistic Regression model.

So, the final model that we will be using for our data product is the Logistic Regression model without SMOTE.

Then we drew a feature importance plot to the model selected.

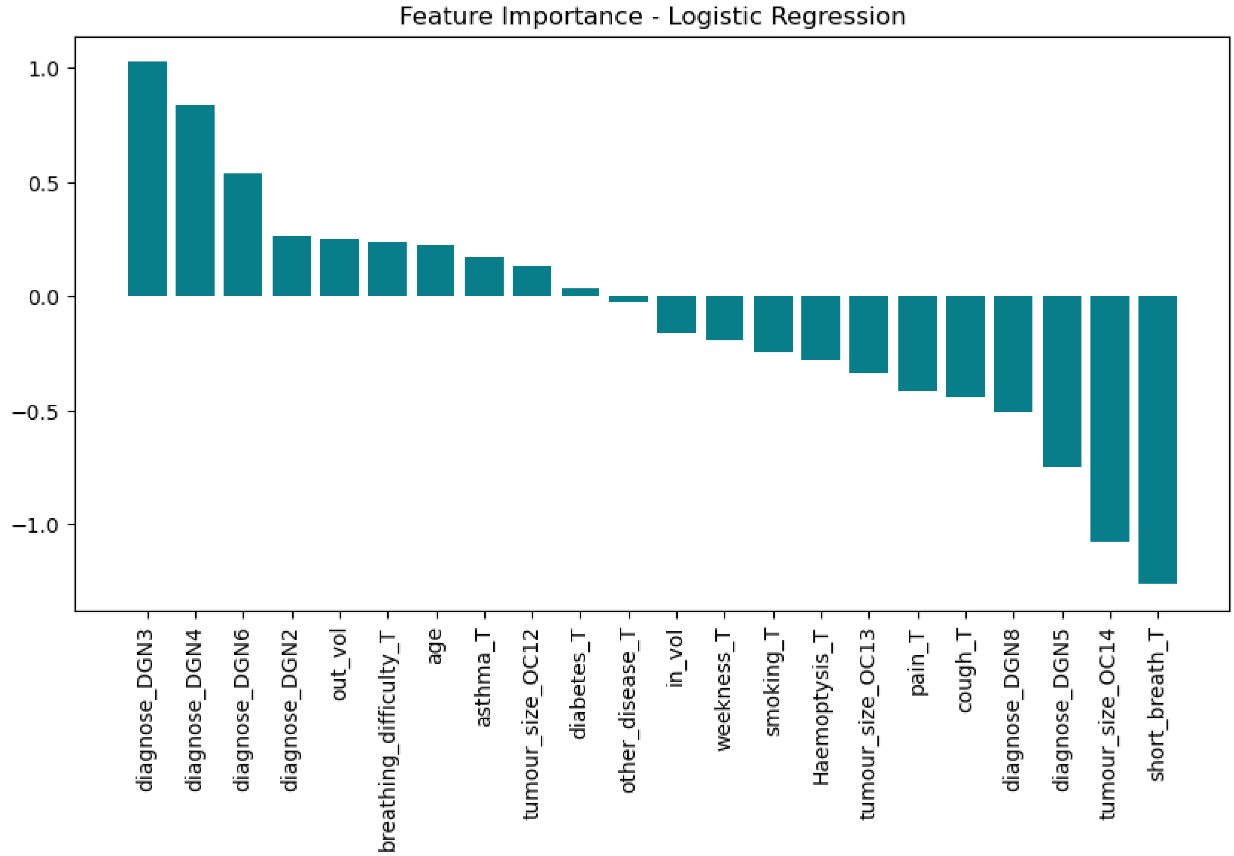

Figure 13

As you can see, all the variables are more or less important to the model. Therefore we decided to not to remove any variable.

Issues encountered and proposed solutions

Issues	Solution
The dataset is highly unbalanced	Using oversampling technique SMOTE (Synthetic Minority Over-sampling Technique) from python library imblearn.
our dataset contained less number of records and due to that it was not an easy task to interpret graphs using counts.	To overcome this, we took percentages to draw graphs and interpret them.

Discussion and conclusion

This study was conducted to predict survival status of cancer patients who faced to the Cardio-thoracic surgery using the machine learning. According to the initial descriptive analysis our target variable ‘survive’ was highly unbalanced. Therefore, we used SMOTE in order to balance the dataset. Then the Logistic Regression, Decision Tree, K Nearest Neighbors, Support Vector Machine were fitted to the dataset, and further to increase F1 score random forest and Voting Classifier models were fitted with hyperparameter tunning techniques. Looking at those models, we realized that Logistic Regression, KNN, SVM and Voting Classifier performs exceptionally well. Out of them, considering interpretability and computational cost, we decided to go with the Logistic Regression model. So, the final model that we will be using for our data product is the Logistic Regression model without SMOTE.

Appendix

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import train_test_split
from imblearn.over_sampling import SMOTE, ADASYN
from sklearn.metrics import accuracy_score, recall_score, precision_score, f1_score
from sklearn.preprocessing import RobustScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.metrics import roc_auc_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import f1_score
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn import svm

data = pd.read_csv('ThoraricSurgery.csv')
data.head()
data.isna().sum()
data.duplicated().sum()
data.shape
data.columns
cats = ['diagnose','pain', 'Haemoptysis',
        'short_breath', 'cough', 'weekness', 'tumour_size', 'diabetes',
        'breathing_difficulty', 'other_disease', 'smoking', 'asthma','survive']
nums = ['in_vol', 'out_vol','age']
# # Descriptive Analysis
plt.figure(figsize=(15, 7))
for i in range(0, len(nums)):
    plt.subplot(1, 3, i+1)
    sns.boxplot(y=data[nums[i]],color='green',orient='v')
    plt.tight_layout()

Q1 = data['age'].quantile(0.25)
Q3 = data['age'].quantile(0.75)
IQR = Q3 - Q1
filter = (data['age'] >= Q1 - 1.5 * IQR) & (data['age'] <= Q3 + 1.5 *IQR)
data = data.loc[filter]

Q1 = data['in_vol'].quantile(0.25)
Q3 = data['in_vol'].quantile(0.75)
IQR = Q3 - Q1
filter = (data['in_vol'] >= Q1 - 1.5 * IQR) & (data['in_vol'] <= Q3 + 1.5 *IQR)
data = data.loc[filter]

Q1 = data['out_vol'].quantile(0.25)
Q3 = data['out_vol'].quantile(0.75)
IQR = Q3 - Q1
filter = (data['out_vol'] >= Q1 - 1.5 * IQR) & (data['out_vol'] <= Q3 + 1.5 *IQR)
data = data.loc[filter]

px.histogram(data,x='age',nbins=40,color='survive')

fig, ax1 = plt.subplots(1,1)
g = sns.countplot(x=data['survive'],ax=ax1)
plt.title('Deaths vs Survived' ,fontsize=22, weight='bold')
plt.ylabel('Count', fontsize = 20, weight='bold')
g.set_xticklabels(['Died', 'Survived'], fontsize = 18)
plt.show()

Diagnosis = {'Diagnosis_type':['DGN1','DGN2','DGN3','DGN4','DGN5','DGN6','DGN8'],
             'Count':[1,47,270,40,13,3,2]}
df_diagnosis = pd.DataFrame(Diagnosis)
df_diagnosis = df_diagnosis.sort_values(["Count"], ascending=True)
df_diagnosis

plt.figure(figsize=(8,5))
sns.barplot(x="Diagnosis_type", y="Count",data = df_diagnosis, ci = None )
plt.title('Distribution of Diagnosis Type ')
plt.xlabel('Diagnosis type', fontsize = 15)
plt.ylabel('count',fontsize = 15)
plt.show()

pd_df=train[train['survive'] == 1]
pd_df['diagnose'].value_counts()
b=pd_df['diagnose'].value_counts().sum()
Diagnosis1 = {'Diagnosis_type':['DGN1','DGN2','DGN3','DGN4','DGN5','DGN6','DGN8'],
              'Prob':[0,10/b,30/b,5/b,6/b,0,1/b]}
df1_diagnosis = pd.DataFrame(Diagnosis1)
df1_diagnosis = df1_diagnosis.sort_values(["Prob"], ascending=True)
df1_diagnosis

# Type of DGN in patients who didn't survive first year after surgery
plt.figure(figsize=(8,5))
sns.barplot(x="Diagnosis_type", y="Prob",data = df1_diagnosis, ci = None )
plt.title("Type of DGN in patients who didn't survive first year after surgery" )
plt.xlabel('Diagnosis type', fontsize = 15)
plt.ylabel('Prob.',fontsize = 15)
plt.show()
# ## Distributons of 'FVC' and 'FEV1'
def plot_hist(col, bins=30, title="",xlabel="",ax=None):
    sns.distplot(col, bins=bins,ax=ax)
    ax.set_title(f'Histogram of {title}',fontsize=20)
    ax.set_xlabel(xlabel)

fig, axes = plt.subplots(1,2,figsize=(10,5),constrained_layout=True)
plot_hist(train.in_vol,
          title='in_vol',
          xlabel="in_vol",
          ax=axes[0])
plot_hist(train.out_vol,
          bins=30,
          title='out_vol',
          xlabel='out_vol',
          ax=axes[1])
 ## Dependency of diabetes in determining survival of a patient
pd.crosstab(train['diabetes'], train['survive']).apply(lambda r: (r/r.sum())*100, axis=1)
rating = pd.DataFrame({
    'Survived': [ 87.14, 73.08],
    'Died': [ 12.86, 26.92]
Class = ["No","Yes"]
ax = rating.plot(stacked=True, kind='bar', color = ['#6dbfa7', '#b5e6d7'])
for bar in ax.patches:
    height = bar.get_height()
    width = bar.get_width()
    x = bar.get_x()
    y = bar.get_y()
    label_text = str('{:.2f}'.format(height)) + '%'
    label_x = x + width / 2
    label_y = y + height / 2
    ax.text(label_x, label_y, label_text, ha='center',
            va='center')
ax.set_xticklabels(Class,rotation='horizontal')
plt.ylabel('Percentage')
plt.xlabel('Diabetes')
plt.show()
# ## Dependency of having PAD in determining survival of a patient
pd.crosstab(train['other_disease'], train['survive']).apply(lambda r: (r/r.sum())*100, axis=1)
rating = pd.DataFrame({
    'Survived': [ 86.41, 75],
    'Died': [ 13.59, 25]
 })
Class = ["No","Yes"]
ax = rating.plot(stacked=True, kind='bar', color = ['#6dbfa7', '#b5e6d7'])

for bar in ax.patches:
    height = bar.get_height()
    width = bar.get_width()
    x = bar.get_x()
    y = bar.get_y()
    label_text = str('{:.2f}'.format(height)) + '%'
    label_x = x + width / 2
    label_y = y + height / 2
    ax.text(label_x, label_y, label_text, ha='center',
            va='center')
ax.set_xticklabels(Class,rotation='horizontal')
plt.ylabel('Percentage')
plt.xlabel('Other Diseases')
plt.show()
# ## Dependency of smoking in determining the survival of a patient.
pd.crosstab(train['smoking'], train['survive']).apply(lambda r: (r/r.sum())*100, axis=1)
rating = pd.DataFrame({
    'Survived': [ 90.41, 85.15],
    'Died': [ 9.59, 14.85]
```

```python
})
Class = ["No","Yes"]
ax = rating.plot(stacked=True, kind='bar', color = ['#6dbfa7', '#b5e6d7'])
for bar in ax.patches:
    height = bar.get_height()
    width = bar.get_width()
    x = bar.get_x()
    y = bar.get_y()
    label_text = str('{:.2f}'.format(height)) + '%'
    label_x = x + width / 2
    label_y = y + height / 2
    ax.text(label_x, label_y, label_text, ha='center',
            va='center')
ax.set_xticklabels(Class,rotation='horizontal')
plt.ylabel('Percentage')
plt.xlabel('Smoking')
plt.show()
# ## Age & survival
g=sns.displot(data=train, x="age",
        hue='survive', kde=True, height=8, palette="viridis")
plt.axvline(x=80, ls = '--', color = 'red')
plt.xlabel('Age' )
plt.ylabel('Count' )
plt.show()
# ## Size of the Tumour
train['tumour_size'].value_counts()
tumour = {'Tumour_size':['OC13','OC14','OC11','OC12'],
        'Count':[15,16,141,204]}

df_tumour = pd.DataFrame(tumour)
df_tumour = df_tumour.sort_values(["Count"], ascending=True)
df_tumour
plt.figure(figsize=(10,5))
sns.barplot(x="Tumour_size", y="Count",data = df_tumour, ci = None)
plt.title('Distribution of Tumor size ', fontsize = 20)
plt.xlabel('Tumour size', fontsize = 15)
plt.ylabel('count',fontsize = 15)
plt.show()
pd.crosstab(train['tumour_size'], train['survive']).apply(lambda r: (r/r.sum())*100, axis=1)
rating = pd.DataFrame({
    'Survived': [ 90.07, 85.78, 80.00, 62.50],
    'Died': [ 9.93, 14.22, 20.00, 37.50]
})
Class = ["OC11","OC12","OC13","OC14"]
ax = rating.plot(stacked=True, kind='bar', color = ['#6dbfa7', '#b5e6d7'])
for bar in ax.patches:
    height = bar.get_height()
    width = bar.get_width()
    x = bar.get_x()
    y = bar.get_y()
    label_text = str('{:.2f}'.format(height)) + '%'
    label_x = x + width / 2
    label_y = y + height / 2
    ax.text(label_x, label_y, label_text, ha='center',
            va='center')
ax.set_xticklabels(Class,rotation='horizontal')
plt.ylabel('Percentage')
plt.xlabel('Tumour Size')
plt.show()
cats = ['diagnose','pain', 'Haemoptysis',
        'short_breath', 'cough', 'weekness', 'tumour_size', 'diabetes',
        'breathing_difficulty', 'other_disease', 'smoking', 'asthma','survive']
nums = ['in_vol', 'out_vol','age']
plt.figure(figsize=(10,8))
plt.title('Correlation matrix - Quantitative variables')
sns.heatmap(train[nums].corr(),cbar=True, fmt='.1f', annot=True, annot_kws={'size':8}, cmap= "crest")
plt.show()

# dython - all the variables
from dython.nominal import associations
associations(train.drop('id', axis=1),nominal_columns=cats, numerical_columns=nums,
             mark_columns=True, fmt='.2f',cmap= "crest", figsize=(10, 10))

data = pd.get_dummies(data,columns=cats,drop_first=True)

rs = RobustScaler()
data[nums] = rs.fit_transform(data[nums])
# # Train test split
X = data.drop(['survive_2','id'], axis=1)
y = data['survive_2']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
# # Oversampling
sm = SMOTE()
X_train_sm, y_train_sm = sm.fit_resample(X_train,y_train)
# # Advanced analysis
# ## Random Forest
# ### Without hyperparameter tuning (overfitted)
rfc = RandomForestClassifier()
rfc.fit(X_train, y_train)
y_pred = rfc.predict(X_train)
# Predict the train data
print('\nclassification report')
print(classification_report(y_train, y_pred)) # generate the precision, recall, f-1 score, num
print('RFC: ROC AUC = ',str(round(roc_auc_score(y_train, y_pred)*100,1)), '%')
print('RFC: Precision = ',str(round(precision_score(y_train, y_pred)*100,1)), '%')
print('RFC: Recall = ',str(round(recall_score(y_train, y_pred)*100,1)), '%')
print('RFC: Accuracy = ',str(round(accuracy_score(y_train, y_pred)*100,1)), '%')
print('RFC: F1-Score = ',str(round(f1_score(y_train, y_pred)*100,1)), '%')
confusion_matrix(y_train, y_pred)
pred = rfc.predict(X_test)
# Predict the test data
print('\nclassification report')
print(classification_report(y_test, pred)) # generate the precision, recall, f-1 score, num
print('LogReg: ROC AUC = ',str(round(roc_auc_score(y_test, pred)*100,1)), '%')
print('LogReg: Precision = ',str(round(precision_score(y_test, pred)*100,1)), '%')
print('LogReg: Recall = ',str(round(recall_score(y_test, pred)*100,1)), '%')
print('LogReg: Accuracy = ',str(round(accuracy_score(y_test, pred)*100,1)), '%')
print('LogReg: F1-Score = ',str(round(f1_score(y_test, pred)*100,1)), '%')
confusion_matrix(y_test, pred)
# ### Hyperparameter Tuning
rfc=RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [200, 500],
#    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(X_train, y_train)
CV_rfc.best_params_

rfc1=RandomForestClassifier(random_state=42, n_estimators= 200, max_depth=4, criterion='gini')
rfc1.fit(X_train, y_train)
# Predict the train data
pred = rfc1.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, pred)) # generate the precision, recall, f-1 score, num
print('RFC: ROC AUC = ',str(round(roc_auc_score(y_train, pred)*100,1)), '%')
print('RFC: Precision = ',str(round(precision_score(y_train, pred)*100,1)), '%')
print('RFC: Recall = ',str(round(recall_score(y_train, pred)*100,1)), '%')
print('RFC: Accuracy = ',str(round(accuracy_score(y_train, pred)*100,1)), '%')
print('RFC: F1-Score = ',str(round(f1_score(y_train, pred)*100,1)), '%')
confusion_matrix(y_train, pred)
pred=rfc1.predict(X_test)
# Predict the test data
print('\nclassification report')
print(classification_report(y_test, pred)) # generate the precision, recall, f-1 score, num
print('RFC: ROC AUC = ',str(round(roc_auc_score(y_test, pred)*100,1)), '%')
print('RFC: Precision = ',str(round(precision_score(y_test, pred)*100,1)), '%')
print('RFC: Recall = ',str(round(recall_score(y_test, pred)*100,1)), '%')
print('RFC: Accuracy = ',str(round(accuracy_score(y_test, pred)*100,1)), '%')
print('RFC: F1-Score = ',str(round(f1_score(y_test, pred)*100,1)), '%')
confusion_matrix(y_test, pred)
# ### Smote
rfc=RandomForestClassifier(random_state=42)
param_grid = {
    'n_estimators': [200, 500],
#    'max_features': ['auto', 'sqrt', 'log2'],
    'max_depth' : [4,5,6,7,8],
    'criterion' :['gini', 'entropy']
}
CV_rfc = GridSearchCV(estimator=rfc, param_grid=param_grid, cv= 5)
CV_rfc.fit(X_train_sm, y_train_sm)
CV_rfc.best_params_
rfc1=RandomForestClassifier(random_state=42, n_estimators= 500, max_depth=8, criterion='entropy')
rfc1.fit(X_train_sm, y_train_sm)
# Predict the train data
pred = rfc1.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, pred)) # generate the precision, recall, f-1 score, num
print('RFC: ROC AUC = ',str(round(roc_auc_score(y_train, pred)*100,1)), '%')
print('RFC: Precision = ',str(round(precision_score(y_train, pred)*100,1)), '%')
print('RFC: Recall = ',str(round(recall_score(y_train, pred)*100,1)), '%')
print('RFC: Accuracy = ',str(round(accuracy_score(y_train, pred)*100,1)), '%')
print('RFC: F1-Score = ',str(round(f1_score(y_train, pred)*100,1)), '%')
confusion_matrix(y_train, pred)
pred=rfc1.predict(X_test)
# Predict the test data
print('\nclassification report')
print(classification_report(y_test, pred)) # generate the precision, recall, f-1 score, num
print('RFC: ROC AUC = ',str(round(roc_auc_score(y_test, pred)*100,1)), '%')
print('RFC: Precision = ',str(round(precision_score(y_test, pred)*100,1)), '%')
print('RFC: Recall = ',str(round(recall_score(y_test, pred)*100,1)), '%')
print('RFC: Accuracy = ',str(round(accuracy_score(y_test, pred)*100,1)), '%')
print('RFC: F1-Score = ',str(round(f1_score(y_test, pred)*100,1)), '%')
confusion_matrix(y_test, pred)
# ## Logistic Regression
# ### Original data
logreg = LogisticRegression(random_state=42, solver = 'liblinear')
logreg.fit(X_train, y_train)
# Predict the train data
ytrain_predicted = logreg.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, ytrain_predicted)) # generate the precision, recall, f-1 score, num
print('LogReg: ROC AUC = ',str(round(roc_auc_score(y_train, ytrain_predicted)*100,1)), '%')
print('LogReg: Precision = ',str(round(precision_score(y_train, ytrain_predicted)*100,1)), '%')
print('LogReg: Recall = ',str(round(recall_score(y_train, ytrain_predicted)*100,1)), '%')
print('LogReg: Accuracy = ',str(round(accuracy_score(y_train, ytrain_predicted)*100,1)), '%')
print('LogReg: F1-Score = ',str(round(f1_score(y_train, ytrain_predicted)*100,1)), '%')
```

```python
confusion_matrix(y_train, ytrain_predicted)
# Predict the test data
ytest_predicted = logreg.predict(X_test)
print('\nclassification report')
print(classification_report(y_test, ytest_predicted)) # generate the precision, recall, f-1 score, num
print('LogReg: ROC AUC = ',str(round(roc_auc_score(y_test, ytest_predicted)*100,1)), '%')
print('LogReg: Precision = ',str(round(precision_score(y_test, ytest_predicted)*100,1)), '%')
print('LogReg: Recall = ',str(round(recall_score(y_test, ytest_predicted)*100,1)), '%')
print('LogReg: Accuracy = ',str(round(accuracy_score(y_test, ytest_predicted)*100,1)), '%')
print('LogReg: F1-Score = ',str(round(f1_score(y_test, ytest_predicted)*100,1)), '%')
confusion_matrix(y_test, ytest_predicted)
#Important variables
importances_lr = pd.DataFrame(data = {'Attribute': X_train.columns, 'Importance': logreg.coef_[0]})
importances_lr = importances_lr.sort_values(by='Importance', ascending = False)
importances_lr
#plot
plt.figure(figsize=(10,5))
plt.bar(x=importances_lr['Attribute'], height = importances_lr['Importance'], color = '#087E8B')
plt.title('Feature Importance - Logistic Regression')
plt.xticks(rotation='vertical')
plt.show()
# ### SMOTE
logreg = LogisticRegression(random_state=42, solver = 'liblinear')
logreg.fit(X_train_sm, y_train_sm)
# Predict the train data
ytrain_predicted = logreg.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, ytrain_predicted)) # generate the precision, recall, f-1 score, num
print('LogReg: ROC AUC = ',str(round(roc_auc_score(y_train, ytrain_predicted)*100,1)), '%')
print('LogReg: Precision = ',str(round(precision_score(y_train, ytrain_predicted)*100,1)), '%')
print('LogReg: Recall = ',str(round(recall_score(y_train, ytrain_predicted)*100,1)), '%')
print('LogReg: Accuracy = ',str(round(accuracy_score(y_train, ytrain_predicted)*100,1)), '%')
print('LogReg: F1-Score = ',str(round(f1_score(y_train, ytrain_predicted)*100,1)), '%')
confusion_matrix(y_train, ytrain_predicted)
# Predict the test data
ytest_predicted = logreg.predict(X_test)
print('\nclassification report')
print(classification_report(y_test, ytest_predicted)) # generate the precision, recall, f-1 score, num
print('LogReg: ROC AUC = ',str(round(roc_auc_score(y_test, ytest_predicted)*100,1)), '%')
print('LogReg: Precision = ',str(round(precision_score(y_test, ytest_predicted)*100,1)), '%')
print('LogReg: Recall = ',str(round(recall_score(y_test, ytest_predicted)*100,1)), '%')
print('LogReg: Accuracy = ',str(round(accuracy_score(y_test, ytest_predicted)*100,1)), '%')
print('LogReg: F1-Score = ',str(round(f1_score(y_test, ytest_predicted)*100,1)), '%')
confusion_matrix(y_test, ytest_predicted)
# ## Decision Tree
# ### Without hyperparameter tuning (overfitted)
dt = DecisionTreeClassifier(random_state=101)
dt.fit(X_train_sm,y_train_sm)
#y_predicted = dt.predict(xtest)
# Predict the train data
ytrain_predicted = dt.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, ytrain_predicted )) # generate the precision, recall, f-1 score, num
print('DT: ROC AUC = ',str(round(roc_auc_score(y_train, ytrain_predicted )*100,1)), '%')
print('DT: Precision = ',str(round(precision_score(y_train, ytrain_predicted)*100,1)), '%')
print('DT: Recall = ',str(round(recall_score(y_train, ytrain_predicted)*100,1)), '%')
print('DT: Accuracy = ',str(round(accuracy_score(y_train, ytrain_predicted)*100,1)), '%')
print('DT: F1-Score = ',str(round(f1_score(y_train, ytrain_predicted)*100,1)), '%')
confusion_matrix(y_train, ytrain_predicted)
## Decision tree overfitted, so we used Random Forest
# Predict the test data
ytest_predicted = dt.predict(X_test)
print('\nclassification report')
print(classification_report(y_test, ytest_predicted )) # generate the precision, recall, f-1 score, num
print('DT: ROC AUC = ',str(round(roc_auc_score(y_test, ytest_predicted )*100,1)), '%')
print('DT: Precision = ',str(round(precision_score(y_test, ytest_predicted)*100,1)), '%')
print('DT: Recall = ',str(round(recall_score(y_test, ytest_predicted)*100,1)), '%')
print('DT: Accuracy = ',str(round(accuracy_score(y_test, ytest_predicted)*100,1)), '%')
print('DT: F1-Score = ',str(round(f1_score(y_test, ytest_predicted)*100,1)), '%')
confusion_matrix(y_test, ytest_predicted)
# ### Hyperparameter tuning
param_grid = {'ccp_alpha': [0.1, .01, .001],
              'max_depth' : [5, 6, 7, 8, 9],
              'criterion' :['gini', 'entropy']
             }
tree_clas = DecisionTreeClassifier(random_state=1024)
grid_search = GridSearchCV(estimator=tree_clas, param_grid=param_grid, cv=5, verbose=True)
grid_search.fit(X_train_sm, y_train_sm)
grid_search.best_params_
# ### Original data
dt = DecisionTreeClassifier(random_state=101, ccp_alpha= 0.001, criterion='gini', max_depth= 8)
dt.fit(X_train,y_train)
# Predict the train data
ytrain_predicted = dt.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, ytrain_predicted )) # generate the precision, recall, f-1 score, num
print('DT: ROC AUC = ',str(round(roc_auc_score(y_train, ytrain_predicted )*100,1)), '%')
print('DT: Precision = ',str(round(precision_score(y_train, ytrain_predicted)*100,1)), '%')
print('DT: Recall = ',str(round(recall_score(y_train, ytrain_predicted)*100,1)), '%')
print('DT: Accuracy = ',str(round(accuracy_score(y_train, ytrain_predicted)*100,1)), '%')
print('DT: F1-Score = ',str(round(f1_score(y_train, ytrain_predicted)*100,1)), '%')
confusion_matrix(y_train, ytrain_predicted)
# Predict the test data
ytest_predicted = dt.predict(X_test)
print('\nclassification report')
print(classification_report(y_test, ytest_predicted )) # generate the precision, recall, f-1 score, num
print('DT: ROC AUC = ',str(round(roc_auc_score(y_test, ytest_predicted )*100,1)), '%')
print('DT: Precision = ',str(round(precision_score(y_test, ytest_predicted)*100,1)), '%')
print('DT: Recall = ',str(round(recall_score(y_test, ytest_predicted)*100,1)), '%')
print('DT: Accuracy = ',str(round(accuracy_score(y_test, ytest_predicted)*100,1)), '%')
print('DT: F1-Score = ',str(round(f1_score(y_test, ytest_predicted)*100,1)), '%')
confusion_matrix(y_test, ytest_predicted)
# ### SMOTE
dt = DecisionTreeClassifier(random_state=101, ccp_alpha= 0.001, criterion='gini', max_depth= 8)
dt.fit(X_train_sm,y_train_sm)
# Predict the train data
ytrain_predicted = dt.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, ytrain_predicted )) # generate the precision, recall, f-1 score, num
print('DT: ROC AUC = ',str(round(roc_auc_score(y_train, ytrain_predicted )*100,1)), '%')
print('DT: Precision = ',str(round(precision_score(y_train, ytrain_predicted)*100,1)), '%')
print('DT: Recall = ',str(round(recall_score(y_train, ytrain_predicted)*100,1)), '%')
print('DT: Accuracy = ',str(round(accuracy_score(y_train, ytrain_predicted)*100,1)), '%')
print('DT: F1-Score = ',str(round(f1_score(y_train, ytrain_predicted)*100,1)), '%')
confusion_matrix(y_train, ytrain_predicted)
# Predict the test data
ytest_predicted = dt.predict(X_test)
print('\nclassification report')
print(classification_report(y_test, ytest_predicted )) # generate the precision, recall, f-1 score, num
print('DT: ROC AUC = ',str(round(roc_auc_score(y_test, ytest_predicted )*100,1)), '%')
print('DT: Precision = ',str(round(precision_score(y_test, ytest_predicted)*100,1)), '%')
print('DT: Recall = ',str(round(recall_score(y_test, ytest_predicted)*100,1)), '%')
print('DT: Accuracy = ',str(round(accuracy_score(y_test, ytest_predicted)*100,1)), '%')
print('DT: F1-Score = ',str(round(f1_score(y_test, ytest_predicted)*100,1)), '%')
confusion_matrix(y_test, ytest_predicted)
knn = KNeighborsClassifier()
knn_model = knn.fit(X_train, y_train)
ypredicted = knn.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, ypredicted)) # generate the precision, recall, f-1 score, num
print('KNN: ROC AUC = ',str(round(roc_auc_score(y_train, ypredicted)*100,1)), '%')
print('KNN: Precision = ',str(round(precision_score(y_train, ypredicted)*100,1)), '%')
print('KNN: Recall = ',str(round(recall_score(y_train, ypredicted)*100,1)), '%')
print('KNN: Accuracy = ',str(round(accuracy_score(y_train, ypredicted)*100,1)), '%')
print('KNN: F1-Score = ',str(round(f1_score(y_train, ypredicted)*100,1)), '%')
confusion_matrix(y_train, ypredicted)
ypredicted = knn.predict(X_test)
print('\nclassification report')
print(classification_report(y_test, ypredicted)) # generate the precision, recall, f-1 score, num
print('KNN: ROC AUC = ',str(round(roc_auc_score(y_test, ypredicted)*100,1)), '%')
```

```python
print('KNN: Precision = ',str(round(precision_score(y_test, ypredicted)*100,1)), '%')
print('KNN: Recall = ',str(round(recall_score(y_test, ypredicted)*100,1)), '%')
print('KNN: Accuracy = ',str(round(accuracy_score(y_test, ypredicted)*100,1)), '%')
print('KNN: F1-Score = ',str(round(f1_score(y_test, ypredicted)*100,1)), '%')
confusion_matrix(y_test, ypredicted)
# ### SMOTE
knn = KNeighborsClassifier()
knn_model = knn.fit(X_train_sm, y_train_sm)
ypredicted = knn.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, ypredicted)) # generate the precision, recall, f-1 score, num
print('KNN: ROC AUC = ',str(round(roc_auc_score(y_train, ypredicted)*100,1)), '%')
print('KNN: Precision = ',str(round(precision_score(y_train, ypredicted)*100,1)), '%')
print('KNN: Recall = ',str(round(recall_score(y_train, ypredicted)*100,1)), '%')
print('KNN: Accuracy = ',str(round(accuracy_score(y_train, ypredicted)*100,1)), '%')
print('KNN: F1-Score = ',str(round(f1_score(y_train, ypredicted)*100,1)), '%')
confusion_matrix(y_train, ypredicted)
ypredicted = knn.predict(X_test)
print('\nclassification report')
print(classification_report(y_test, ypredicted)) # generate the precision, recall, f-1 score, num
print('KNN: ROC AUC = ',str(round(roc_auc_score(y_test, ypredicted)*100,1)), '%')
print('KNN: Precision = ',str(round(precision_score(y_test, ypredicted)*100,1)), '%')
print('KNN: Recall = ',str(round(recall_score(y_test, ypredicted)*100,1)), '%')
print('KNN: Accuracy = ',str(round(accuracy_score(y_test, ypredicted)*100,1)), '%')
print('KNN: F1-Score = ',str(round(f1_score(y_test, ypredicted)*100,1)), '%')
confusion_matrix(y_test, ypredicted)
# ## Voting Classifier
# ### Original Data
voting_clf = VotingClassifier(
estimators=[('lr', logreg), ('rf', rfc1
('KNN', knn),('DT', dt)],
voting='hard')
voting_clf.fit(X_train, y_train)
from sklearn.metrics import accuracy_sco
for clf in (knn, voting_clf):
    clf.fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
    print(clf.__class__.__name__, precision_score(y_test, y_pred))
    print(clf.__class__.__name__, recall_score(y_test, y_pred))
    print(clf.__class__.__name__, f1_score(y_test, y_pred))
ypredicted = voting_clf.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, ypredicted)) # generate the precision, recall, f-1 score, num
print('VC: ROC AUC = ',str(round(roc_auc_score(y_train, ypredicted)*100,1)), '%')
print('VC: Precision = ',str(round(precision_score(y_train, ypredicted)*100,1)), '%')
print('VC: Recall = ',str(round(recall_score(y_train, ypredicted)*100,1)), '%')
print('VC: Accuracy = ',str(round(accuracy_score(y_train, ypredicted)*100,1)), '%')
print('VC: F1-Score = ',str(round(f1_score(y_train, ypredicted)*100,1)), '%')
confusion_matrix(y_train, ypredicted)
ypredicted = voting_clf.predict(X_test)
print('\nclassification report')
print(classification_report(y_test, ypredicted)) # generate the precision, recall, f-1 score, num
print('VC: ROC AUC = ',str(round(roc_auc_score(y_test, ypredicted)*100,1)), '%')
print('VC: Precision = ',str(round(precision_score(y_test, ypredicted)*100,1)), '%')
print('VC: Recall = ',str(round(recall_score(y_test, ypredicted)*100,1)), '%')
print('VC: Accuracy = ',str(round(accuracy_score(y_test, ypredicted)*100,1)), '%')
print('VC: F1-Score = ',str(round(f1_score(y_test, ypredicted)*100,1)), '%')
confusion_matrix(y_test, ypredicted)
# ### SMOTE
voting_clf = VotingClassifier(
estimators=[('lr', logreg), ('rf', rfc1), ('KNN', knn),('DT', dt)],
voting='hard')
voting_clf.fit(X_train_sm, y_train_sm)
for clf in (knn, voting_clf):
    clf.fit(X_train_sm, y_train_sm)
    y_pred = clf.predict(X_test)
    print(clf.__class__.__name__, accuracy_score(y_test, y_pred))
    print(clf.__class__.__name__, precision_score(y_test, y_pred))
    print(clf.__class__.__name__, recall_score(y_test, y_pred))
    print(clf.__class__.__name__, f1_score(y_test, y_pred))
ypredicted = voting_clf.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, ypredicted)) # generate the precision, recall, f-1 score, num
print('VC: ROC AUC = ',str(round(roc_auc_score(y_train, ypredicted)*100,1)), '%')
print('VC: Precision = ',str(round(precision_score(y_train, ypredicted)*100,1)), '%')
print('VC: Recall = ',str(round(recall_score(y_train, ypredicted)*100,1)), '%')
print('VC: Accuracy = ',str(round(accuracy_score(y_train, ypredicted)*100,1)), '%')
print('VC: F1-Score = ',str(round(f1_score(y_train, ypredicted)*100,1)), '%')
confusion_matrix(y_train, ypredicted)
ypredicted = voting_clf.predict(X_test)
print('\nclassification report')
print(classification_report(y_test, ypredicted)) # generate the precision, recall, f-1 score, num
print('VC: ROC AUC = ',str(round(roc_auc_score(y_test, ypredicted)*100,1)), '%')
print('VC: Precision = ',str(round(precision_score(y_test, ypredicted)*100,1)), '%')
print('VC: Recall = ',str(round(recall_score(y_test, ypredicted)*100,1)), '%')
print('VC: Accuracy = ',str(round(accuracy_score(y_test, ypredicted)*100,1)), '%')
print('VC: F1-Score = ',str(round(f1_score(y_test, ypredicted)*100,1)), '%')
confusion_matrix(y_test, ypredicted)
# ## SVM
# ### Original Data
clf = svm.SVC(kernel='linear', C = 1.0)
clf.fit(X_train,y_train)
ypredicted = voting_clf.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, ypredicted)) # generate the precision, recall, f-1 score, num
print('SVM: ROC AUC = ',str(round(roc_auc_score(y_train, ypredicted)*100,1)), '%')
print('SVM: Precision = ',str(round(precision_score(y_train, ypredicted)*100,1)), '%')
print('SVM: Recall = ',str(round(recall_score(y_train, ypredicted)*100,1)), '%')
print('SVM: Accuracy = ',str(round(accuracy_score(y_train, ypredicted)*100,1)), '%')
print('SVM: F1-Score = ',str(round(f1_score(y_train, ypredicted)*100,1)), '%')
confusion_matrix(y_train, ypredicted)
ypredicted = clf.predict(X_test)
print('\nclassification report')
print(classification_report(y_test, ypredicted)) # generate the precision, recall, f-1 score, num
print('SVM: ROC AUC = ',str(round(roc_auc_score(y_test, ypredicted)*100,1)), '%')
print('SVM: Precision = ',str(round(precision_score(y_test, ypredicted)*100,1)), '%')
print('SVM: Recall = ',str(round(recall_score(y_test, ypredicted)*100,1)), '%')
print('SVM: Accuracy = ',str(round(accuracy_score(y_test, ypredicted)*100,1)), '%')
print('SVM: F1-Score = ',str(round(f1_score(y_test, ypredicted)*100,1)), '%')
confusion_matrix(y_test, ypredicted)
# ### SMOTE
clf = svm.SVC(kernel='linear', C = 1.0)
clf.fit(X_train_sm,y_train_sm)
ypredicted = voting_clf.predict(X_train)
print('\nclassification report')
print(classification_report(y_train, ypredicted)) # generate the precision, recall, f-1 score, num
```

```
print('SVM: ROC AUC = ',str(round(roc_auc_score(y_train, ypredicted)*100,1)), '%')
print('SVM: Precision = ',str(round(precision_score(y_train, ypredicted)*100,1)), '%')
print('SVM: Recall = ',str(round(recall_score(y_train, ypredicted)*100,1)), '%')
print('SVM:  Accuracy  =  ',str(round(accuracy_score(y_train, ypredicted)*100,1)), '%')
print('SVM: F1-Score = ',str(round(f1_score(y_train, ypredicted)*100,1)), '%')
confusion_matrix(y_train, ypredicted)
ypredicted = clf.predict(X_test)
print('\nclassification report')
print(classification_report(y_test, ypredicted)) # generate the precision, recall, f-1 score, num
print('SVM: ROC AUC = ',str(round(roc_auc_score(y_test, ypredicted)*100,1)), '%')
print('SVM: Precision = ',str(round(precision_score(y_test, ypredicted)*100,1)), '%')
print('SVM: Recall = ',str(round(recall_score(y_test, ypredicted)*100,1)), '%')
print('SVM:  Accuracy  =  ',str(round(accuracy_score(y_test, ypredicted)*100,1)), '%')
print('SVM: F1-Score = ',str(round(f1_score(y_test, ypredicted)*100,1)), '%')
confusion_matrix(y_test, ypredicted)
```