**One Year Life Expectancy Post Thoracic Surgery Using Machine Learning**

# [DOCUMENT TITLE]

**PROFESSIONAL TRAINING REPORT – 2**

NAME : BANDI.MOHAN SAI

REG.NO : 39110128

Sathyabama Institute of Science and Technology

Computer Science and Engineering

# INTRODUCTION

## 1.1.overview

Cancer is a disease in which cells in the body grow out of control and is one of the most serious health problems in the world. Among various different types of cancer, lung cancer is one of the leading causes of death in both men and women. According to World Health Organization, it was observed that in the year 2018 2.09 million cases of lung cancer was registered and a total of 1.76 million died due to lung cancer. One of the reasons for the high death rate due to lung cancer is the late detection of the lung cancer. Also, the treatment and the prognosis depend on the type of the lung cancer, the stage and the patient's performance. Once the lung cancer is detected, possible treatments include thoracic surgery, chemotherapy and radiotherapy

## 1.2 purpose

the United States, lung cancer claims more lives every year than colon cancer Lung cancer is the leading cause of cancer-related deaths in the world. In, prostate cancer, and breast cancer combined.

Despite the very serious prognosis (outlook) of lung cancer, some people with earlier-stage cancers are cured. More than 430,000 people alive today have been diagnosed with lung cancer at some point. The data is dedicated to classification problems related to the post-operative life expectancy in lung cancer patients: class 1 - death within one year after surgery, class 2 - survival.

We will be using classification algorithms such as Decision tree, Random forest, KNN, and xgboost. We will train and test the data with these algorithms. From this best model is selected and saved in pkl format. We will be doing flask integration and IBM deployment.

# 2.LITERATURE SURVEY

## 2.1 Existing problem

Disha Sharma et al. (2011), proposed an approach for the early detection of lung cancer by analyzing lungs CT images using Image Processing techniques[1]. The authors used bit-plane slicing, erosion and Weiner filter image processing techniques to extract the lung regions from the CT image. Further the extracted lung regions were segmented using Region growing segmentation algorithm and later Rule based model was used to detect the cancerous nodules. With the help of diagnostics indicator, it was observed that the proposed method achieved an overall accuracy of 80%. Hamid Bagherieh et al. (2013) gave a methodology to detect and classify the lung nodules using Image processing and Decision-Making techniques[2]. Initially, image preprocessing was carried out on a CT images by using contrast enhancement and linear filtering. Next, the filtered image was segmented using Region growing Segmentation process. Further the features like area and color was given as input to the Fuzzy system which employed fuzzy membership function to find the abnormalities. Sindhu V et al.(2014), proposed an approach where the authors aimed to classify the survival of lung cancer patients post thoracic surgery[3]. The authors used Naïve Bayes, PART, J48, OneR, Random Forest and Decision stump algorithm techniques to classify the target function. The performance measurement shows that Random Forest gave high accuracy of 95.65% compared to other ML techniques used. Prashant Naresh et al.(2014), proposed a methodology to detect the lung cancer using Image processing and Neural Techniques[4]. Initially the CT image of lung was filtered to remove Gaussian white noise and Otsu's threshold technique was used to do the segmentation of the image. The structural and features were extracted and these features were given as input to the classifier. The SVM and ANN techniques were used for the classification and it was found that SVM techniques gave a higher accuracy of 95.12%. Kwetishe Joro Danjuma (2015), proposed a methodology to predict the one-year survival of the patient post thoracic surgery[5]. Naïve Bayes, J48 and Multilayer Perceptron algorithms were used to classify the target class. The Naïve Bayes gave an accuracy of 74.4%, J48 gave an accuracy of 81.8% and MLP gave an accuracy of 82.4%.

## 2.1 proposed solution

A system is developed which detects the lung cancer from the given input CT scanned lung images which are in DICOM (.dcm) format. In addition to this, the system also helps to predict the Life Expectancy Post Thoracic Surgery of the Lung Cancer infected patients.

## 3.THEORITICAL ANALYSIS

### 3.1 Block diagram

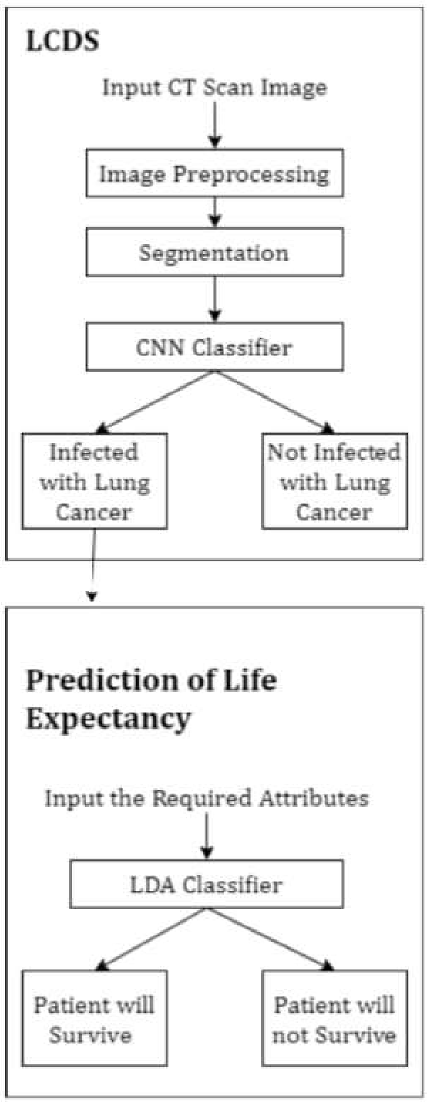



**Fig-1:** Block Diagram of the System

## 3.2 Hardware/software designing

Processor : intel core i3

Speed : 1.19GHz

RAM : 8.0GB

Memory usage : 7.2kb

Keyboard : Standard keyboard

Monitor : 15 VGA color

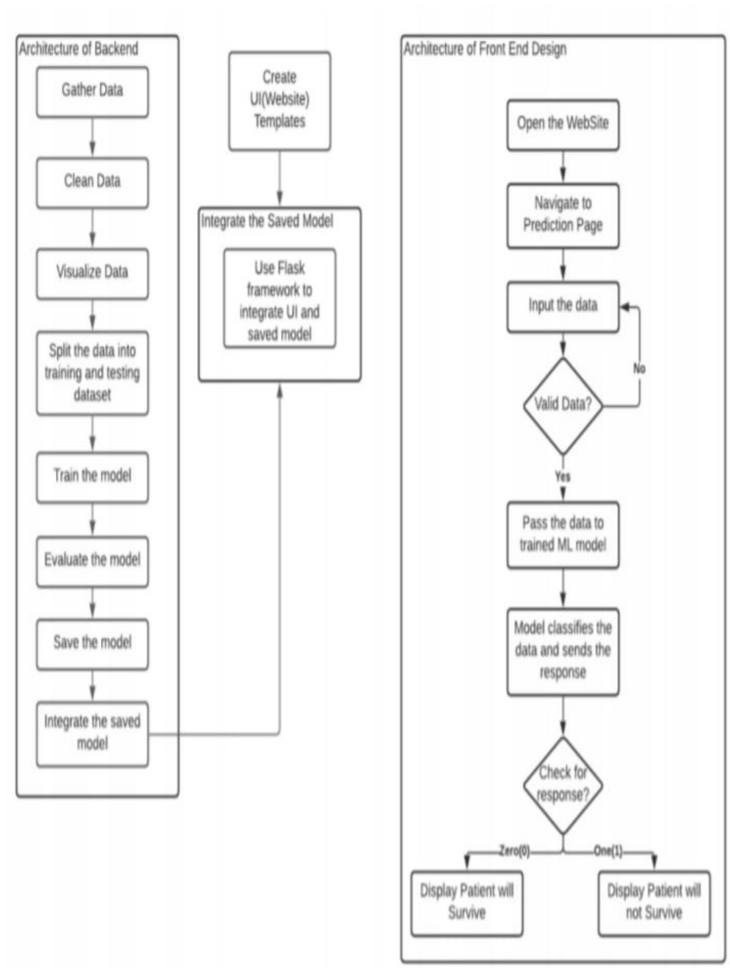
## * Software Requirements

* Operating System :windows
* Coding Langugae:Python3.7

## 4.EXPERIMENTAL INVESTIGATIONS

This is the second part of the system which aims at predicting the survival of lung cancer infected patient post thoracic surgery. The dataset includes 17 attributes which are specified in Table-1. Among all those attributes, Risk1Y is the target class specifying zero if the patient survives for at least one-year post thoracic surgery and one for those who died before completing one-year post surgery. The visualization of the dataset is done using the Matplotlib and Seaborn libraries of Python. Further the essential attributes are found based on the Information Gain (IG) attribute evaluation which is used to find the importance of an attribute by using the Information Gain with respect to the target class. IG (Class, Attribute) = E (Class) – E (Class | Attribute) where, E stands for Entropy After the IG Attribute Evaluation on all the 16 independent attributes in the dataset, it is found that the attributes PRE19 and PRE32 gives the IG value as zero and hence are the least useful attributes for training the model. Therefore, these two attributes are eliminated and the remaining 14 attributes are used to train the model

## 5. FLOWCHART




## 6.RESULT

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 20 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

In [1]:
```python
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix
import itertools
```

In [2]:
```python
df=pd.read_csv("ThoracicSurgery.csv")
```

In [3]:
```python
df.head()
```

Out[3]:

| | Diagnosis | FVC | FEV1 | Performance | Pain | Haemoptysis | Dyspnoea | Cough | Weakness | Tumor_Size | Diabetes_Mellitus | MI_6mo | PAD | Smoking | Asthma | Ag |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 2.88 | 2.16 | 1 | 0 | 0 | 0 | 1 | 1 | 4 | 0 | 0 | 0 | 1 | 0 | 6 |
| 1 | 3 | 3.40 | 1.88 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 5 |
| 2 | 3 | 2.76 | 2.08 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 5 |

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 22 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

```python
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix
import itertools
```

In [2]:
```python
df=pd.read_csv("ThoracicSurgery.csv")
```

In [3]:
```python
df.head()
```


Out[3]:

| | Diagnosis | FVC | FEV1 | Performance | Pain | Haemoptysis | Dyspnoea | Cough | Weakness | Tumor_Size | Diabetes_Mellitus | MI_6mo | PAD | Smoking | Asthma | Ag |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2 | 2.88 | 2.16 | 1 | 0 | 0 | 0 | 1 | 1 | 4 | 0 | 0 | 0 | 1 | 0 | 6 |
| 1 | 3 | 3.40 | 1.88 | 0 | 0 | 0 | 0 | 0 | 0 | 2 | 0 | 0 | 0 | 1 | 0 | 5 |
| 2 | 3 | 2.76 | 2.08 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 5 |
| 3 | 3 | 3.68 | 3.04 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 5 |
| 4 | 3 | 2.44 | 0.96 | 2 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 7 |

In [4]:
```python
df.columns
```

Out[4]:
```
Index(['Diagnosis', 'FVC', 'FEV1', 'Performance', 'Pain', 'Haemoptysis',
       'Dyspnoea', 'Cough', 'Weakness', 'Tumor_Size', 'Diabetes_Mellitus',
       'MI_6mo', 'PAD', 'Smoking', 'Asthma', 'Age', 'Death_1yr'],
      dtype='object')
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 22 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help | Trusted | Python 3

In [5]:
```python
df.describe()
```

Out[5]:

| | Diagnosis | FVC | FEV1 | Performance | Pain | Haemoptysis | Dyspnoea | Cough | Weakness | Tumor_Size | Diabetes_Mellitus | MI_6m |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 454.000000 | 454.000000 | 454.00000 | 454.000000 | 454.000000 | 454.000000 | 454.000000 | 454.000000 | 454.000000 | 454.000000 | 454.000000 | 454.0000 |
| mean | 3.092511 | 3.287952 | 2.51685 | 0.795154 | 0.059471 | 0.136564 | 0.055066 | 0.696035 | 0.171806 | 1.733480 | 0.074890 | 0.0044 |
| std | 0.715817 | 0.872347 | 0.77189 | 0.531459 | 0.236766 | 0.343765 | 0.228361 | 0.460475 | 0.377628 | 0.707499 | 0.263504 | 0.0662 |
| min | 1.000000 | 1.440000 | 0.96000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.0000 |
| 25% | 3.000000 | 2.600000 | 1.96000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 0.0000 |
| 50% | 3.000000 | 3.160000 | 2.36000 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 2.000000 | 0.000000 | 0.0000 |
| 75% | 3.000000 | 3.840000 | 2.97750 | 1.000000 | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 | 2.000000 | 0.000000 | 0.0000 |
| max | 8.000000 | 6.300000 | 5.48000 | 2.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 4.000000 | 1.000000 | 1.0000 |

In [6]:
```python
df.shape
```

Out[6]:
```
(454, 17)
```

In [7]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 454 entries, 0 to 453
Data columns (total 17 columns):
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 22 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [7]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 454 entries, 0 to 453
Data columns (total 17 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Diagnosis          454 non-null    int64
 1   FVC                454 non-null    float64
 2   FEV1               454 non-null    float64
 3   Performance        454 non-null    int64
 4   Pain               454 non-null    int64
 5   Haemoptysis        454 non-null    int64
 6   Dyspnoea           454 non-null    int64
 7   Cough              454 non-null    int64
 8   Weakness           454 non-null    int64
 9   Tumor_Size         454 non-null    int64
 10  Diabetes_Mellitus  454 non-null    int64
 11  MI_6mo             454 non-null    int64
 12  PAD                454 non-null    int64
 13  Smoking            454 non-null    int64
 14  Asthma             454 non-null    int64
 15  Age                454 non-null    int64
 16  Death_1yr          454 non-null    int64
dtypes: float64(2), int64(15)
memory usage: 60.4 KB
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 22 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [8]:
```
df.isnull().sum()
```

Out[8]:
```
Diagnosis            0
FVC                  0
FEV1                 0
Performance          0
Pain                 0
Haemoptysis          0
Dyspnoea             0
Cough                0
Weakness             0
Tumor_Size           0
Diabetes_Mellitus    0
MI_6mo               0
PAD                  0
Smoking              0
Asthma               0
Age                  0
Death_1yr            0
dtype: int64
```

In [9]:
```
live  = df[df['Death_1yr'] == 0]
death = df[df['Death_1yr'] == 1]

cond = ['FVC', 'FEV1', 'Performance', 'Pain', 'Haemoptysis', 'Dyspnoea', 'Cough', 'Weakness',\
        'Tumor_Size', 'Diabetes_Mellitus', 'MI_6mo', 'PAD', 'Smoking', 'Asthma', 'Age']

l = [np.mean(live[c]) for c in cond]
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 22 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

In [9]:
```
live  = df[df['Death_1yr'] == 0]
death = df[df['Death_1yr'] == 1]

cond = ['FVC', 'FEV1', 'Performance', 'Pain', 'Haemoptysis', 'Dyspnoea', 'Cough', 'Weakness',\
        'Tumor_Size', 'Diabetes_Mellitus', 'MI_6mo', 'PAD', 'Smoking', 'Asthma', 'Age']

l = [np.mean(live[c]) for c in cond]
d = [np.mean(death[c]) for c in cond]

ld = pd.DataFrame(data={'Attribute': cond, 'Live 1yr Mean': l, 'Death 1yr Mean': d})
ld = ld.set_index('Attribute')


print('Death: {:d}, Live: {:d}'.format(len(death), len(live)))
print("1 year death: {:.2f}% out of 454 patients".format(np.mean(df.Death_1yr)*100))
ld
```

```
Death: 69, Live: 385
1 year death: 15.20% out of 454 patients
```

Out[9]:

| | Live 1yr Mean | Death 1yr Mean |
|---|---|---|
| **Attribute** | | |
| **FVC** | 1 | 3.195072 |
| **FEV1** | 1 | 2.383188 |
| **Performance** | 1 | 0.913043 |

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 22 minutes ago (autosaved)

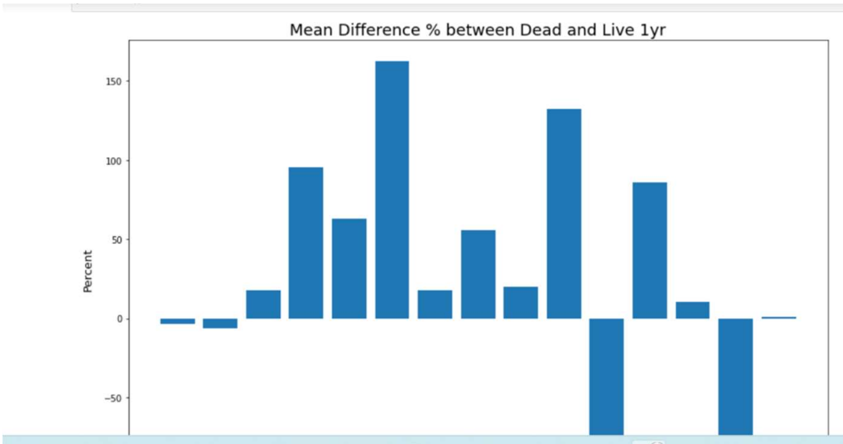
File Edit View Insert Cell Kernel Widgets Help Trusted | Pyth

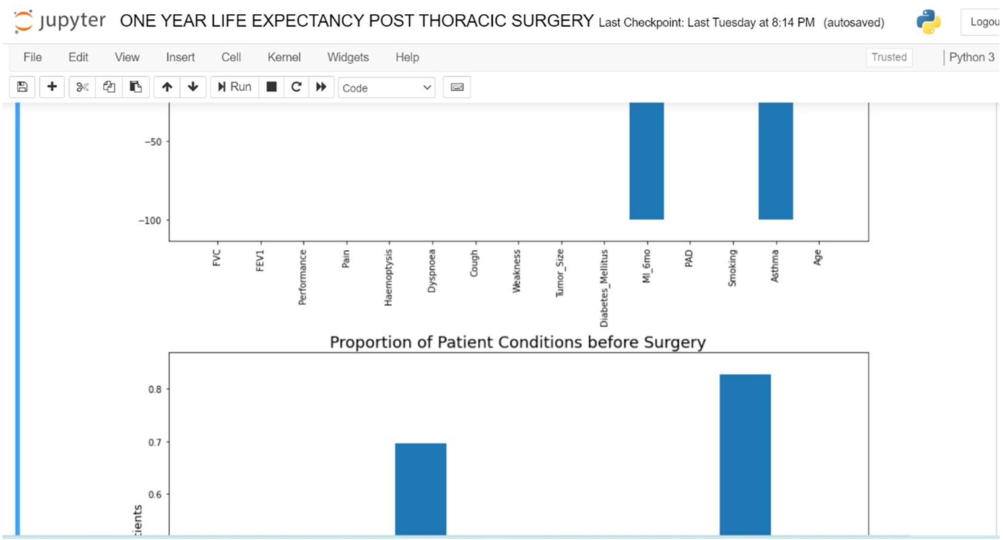
Out[9]:

| Attribute | Live 1yr Mean | Death 1yr Mean |
|---|---|---|
| FVC | 1 | 3.195072 |
| FEV1 | 1 | 2.383188 |
| Performance | 1 | 0.913043 |
| Pain | 1 | 0.101449 |
| Haemoptysis | 1 | 0.202899 |
| Dyspnoea | 1 | 0.115942 |
| Cough | 1 | 0.797101 |
| Weakness | 1 | 0.246377 |
| Tumor_Size | 1 | 2.014493 |
| Diabetes_Mellitus | 1 | 0.144928 |
| MI_6mo | 1 | 0.000000 |
| PAD | 1 | 0.028986 |
| Smoking | 1 | 0.898551 |
| Asthma | 1 | 0.000000 |
| Age | 1 | 63.333333 |

In [10]:

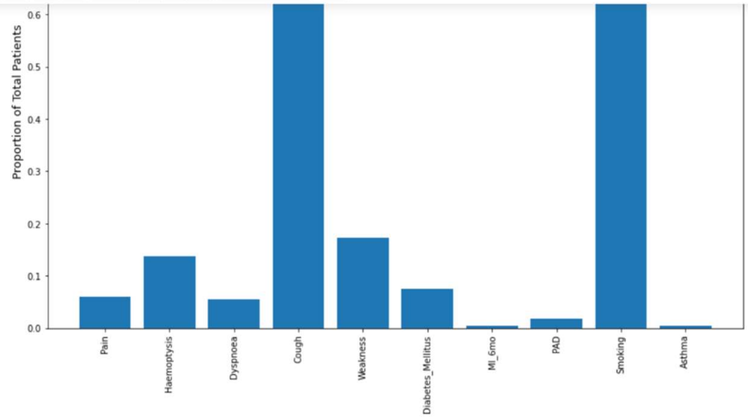
```
d = np.array(d)
l = np.array(l)

p_diff = (d-l)/l*100

fig, axes = plt.subplots(2,1,figsize=(12,18))

axes[0].bar(cond, p_diff)
axes[0].set_title('Mean Difference % between Dead and Live 1yr', fontsize=18)
axes[0].set_xticks(cond)
axes[0].set_xticklabels(cond, rotation=90)
axes[0].set_ylabel('Percent', fontsize=13)

tf_col = ['Pain', 'Haemoptysis', 'Dyspnoea', 'Cough', 'Weakness', 'Diabetes_Mellitus', 'MI_6mo', 'PAD', 'Smoking', 'Asthma']
tf_sum = [df[col].sum()/454 for col in tf_col]

axes[1].bar(tf_col, tf_sum)
axes[1].set_xticks(tf_col)
axes[1].set_xticklabels(tf_col, rotation=90)
axes[1].set_ylabel('Proportion of Total Patients', fontsize=13)
axes[1].set_title('Proportion of Patient Conditions before Surgery', fontsize=18)

plt.tight_layout()

plt.show()
```


Mean Difference % between Dead and Live 1yr
Percent
150
100
50
0
−50


jupyter
ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY
Last Checkpoint: Last Tuesday at 8:14 PM (autosaved)
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
−50
−100
FVC
FEV1
Performance
Pain
Haemoptysis
Dyspnoea
Cough
Weakness
Tumor_Size
Diabetes_Mellitus
MI_6mo
PAD
Smoking
Asthma
Age
Proportion of Patient Conditions before Surgery
0.8
0.7
0.6

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: Last Tuesday at 8:14 PM (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Pytho




jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 23 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [12]:
```
def permutation_sample(data1, data2):
    """Generate a permutation sample from two data sets."""
    data = np.concatenate((data1, data2))
    permuted_data = np.random.permutation(data)

    perm_sample_1 = permuted_data[:len(data1)]
    perm_sample_2 = permuted_data[len(data1):]

    return perm_sample_1, perm_sample_2

def draw_perm_reps(data_1, data_2, func, size=1):
    """Generate multiple permutation replicates."""
    perm_replicates = np.empty(size)

    for i in range(size):
        perm_sample_1, perm_sample_2 = permutation_sample(data_1, data_2)
        perm_replicates[i] = func(perm_sample_1, perm_sample_2)

    return perm_replicates

def diff_of_means(data_1, data_2):
    """Difference in means of two arrays."""
    diff = np.mean(data_1) - np.mean(data_2)
    return diff
```

In [13]:
```
condition = ['FVC', 'FEV1', 'Performance', 'Pain', 'Haemoptysis', 'Dyspnoea', 'Cough', 'Weakness',\
             'Tumor_Size', 'Diabetes_Mellitus', 'MI_6mo', 'PAD', 'Smoking', 'Asthma', 'Age']
p_val = []
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 23 minutes ago (autosaved) Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [13]:
```
condition = ['FVC', 'FEV1', 'Performance', 'Pain', 'Haemoptysis', 'Dyspnoea', 'Cough', 'Weakness',\
             'Tumor_Size', 'Diabetes_Mellitus', 'MI_6mo', 'PAD', 'Smoking', 'Asthma', 'Age']
p_val = []

for c in condition:
    empirical_diff_means = diff_of_means(death[c], live[c])
    perm_replicates = draw_perm_reps(death[c], live[c], diff_of_means, size=10000)
    if empirical_diff_means > 0:
        p = np.sum(perm_replicates >= empirical_diff_means) / len(perm_replicates)
        p_val.append(p)
    else:
        p = np.sum(perm_replicates <= empirical_diff_means) / len(perm_replicates)
        p_val.append(p)

print(list(zip(condition, p_val)))
```

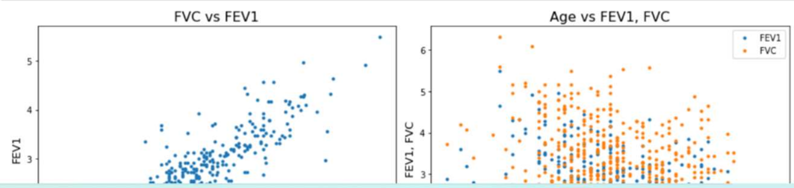
```
[('FVC', 0.1737), ('FEV1', 0.0546), ('Performance', 0.03), ('Pain', 0.0973), ('Haemoptysis', 0.0644), ('Dyspnoea', 0.0265), ('C
ough', 0.0315), ('Weakness', 0.0559), ('Tumor_Size', 0.0008), ('Diabetes_Mellitus', 0.0201), ('MI_6mo', 0.7166), ('PAD', 0.357
2), ('Smoking', 0.0603), ('Asthma', 0.7149), ('Age', 0.2742)]
```

In [14]:
```
fig, axes = plt.subplots(1,2,figsize=(13,5))
axes[0].plot(df.FVC, df.FEV1, linestyle='none', marker='.')

axes[0].set_xlabel('FVC', fontsize=13)
axes[0].set_ylabel('FEV1', fontsize=13)
axes[0].set_title('FVC vs FEV1', fontsize=16)
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 24 minutes ago (autosaved)

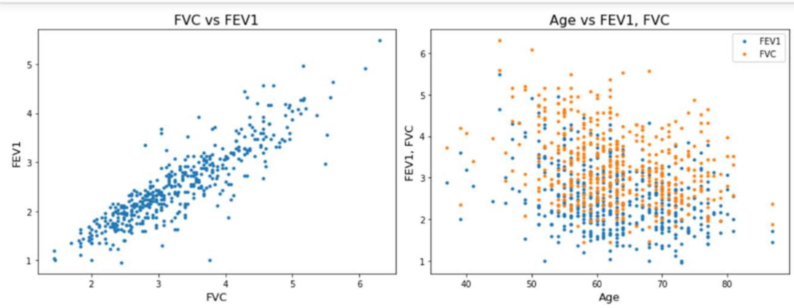
File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

In [14]:
```
fig, axes = plt.subplots(1,2,figsize=(13,5))
axes[0].plot(df.FVC, df.FEV1, linestyle='none', marker='.')

axes[0].set_xlabel('FVC', fontsize=13)
axes[0].set_ylabel('FEV1', fontsize=13)
axes[0].set_title('FVC vs FEV1', fontsize=16)

axes[1].plot(df.Age, df.FEV1, linestyle='none', marker='.', label='FEV1')
axes[1].plot(df.Age, df.FVC, linestyle='none', marker='.', label='FVC')
axes[1].set_xlabel('Age', fontsize=13)
axes[1].set_ylabel('FEV1, FVC', fontsize=13)
axes[1].legend()
axes[1].set_title('Age vs FEV1, FVC', fontsize=16)

plt.tight_layout()
```




jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 24 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3




In [15]:
```
np.corrcoef(df.FVC, df.FEV1)[0,1]
```

Out[15]: 0.8875452733829001

In [16]:
```
np.corrcoef(df.Age, df.FVC)[0,1]
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 24 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [16]:
```
np.corrcoef(df.Age, df.FVC)[0,1]
```
Out[16]:
```
-0.2994299196604911
```
In [17]:
```
np.corrcoef(df.Age, df.FEV1)[0,1]
```
Out[17]:
```
-0.30961662730798917
```
In [18]:
```
def ecdf(data):
    """Compute ECDF for a one-dimensional array of measurements."""
    n = len(data)
    x = np.sort(data)
    y = np.arange(1, n+1) / n
    return x, y
```
In [19]:
```
x_fvc, y_fvc = ecdf(df.FVC)
x_fev1, y_fev1 = ecdf(df.FEV1)
x_age, y_age = ecdf(df.Age)

fig, axes = plt.subplots(1,2,figsize=(13,5))
axes[0].plot(x_fvc, y_fvc, marker='.', linestyle='none', label='FVC')
axes[0].plot(x_fev1, y_fev1, marker='.', linestyle='none', label='FEV1')

axes[0].set_xlabel('Numerical Value', fontsize=13)
axes[0].set_ylabel('ECDF', fontsize=13)
axes[0].legend(loc='upper left')
axes[0].set_title('ECDF of FVC & FEV1', fontsize=16)
```

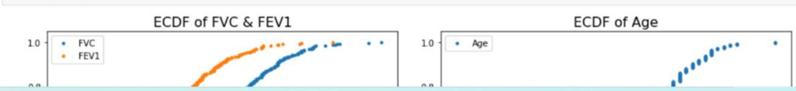
jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 24 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
    return x, y
```
In [19]:
```
x_fvc, y_fvc = ecdf(df.FVC)
x_fev1, y_fev1 = ecdf(df.FEV1)
x_age, y_age = ecdf(df.Age)

fig, axes = plt.subplots(1,2,figsize=(13,5))
axes[0].plot(x_fvc, y_fvc, marker='.', linestyle='none', label='FVC')
axes[0].plot(x_fev1, y_fev1, marker='.', linestyle='none', label='FEV1')

axes[0].set_xlabel('Numerical Value', fontsize=13)
axes[0].set_ylabel('ECDF', fontsize=13)
axes[0].legend(loc='upper left')
axes[0].set_title('ECDF of FVC & FEV1', fontsize=16)

axes[1].plot(x_age, y_age, marker='.', linestyle='none', label='Age')
axes[1].set_xlabel('Years Old', fontsize=13)
axes[1].set_ylabel('ECDF', fontsize=13)
axes[1].legend(loc='upper left')
axes[1].set_title('ECDF of Age', fontsize=16)

plt.tight_layout()
```



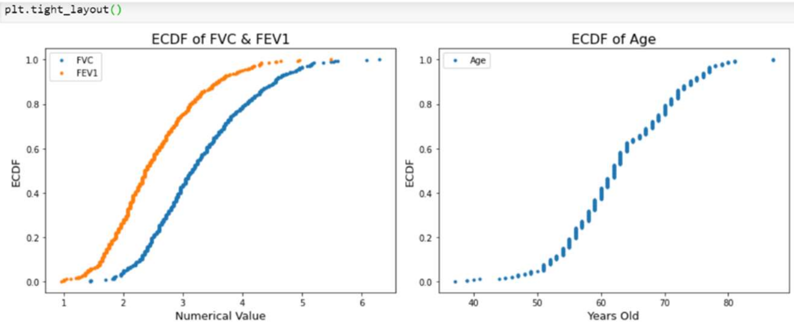

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 24 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

```
plt.tight_layout()
```




In [20]:
```
df.drop(['FVC'],axis=1, inplace=True)
```
In [21]:
```
x=df.iloc[:,0:15].values
y=df.iloc[:,15:16].values
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 24 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

In [20]:
```
df.drop(['FVC'],axis=1, inplace=True)
```

In [21]:
```
x=df.iloc[:,0:15].values
y=df.iloc[:,15:16].values
```

In [22]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=0)
```

In [23]:
```
print('Shape of x_train {}'.format(x_train.shape))
print('Shape of y_train {}'.format(y_train.shape))
print('Shape of x_test {}'.format(x_test.shape))
print('Shape of y_test {}'.format(y_test.shape))
```

```
Shape of x_train (363, 15)
Shape of y_train (363, 1)
Shape of x_test (91, 15)
Shape of y_test (91, 1)
```

In [24]:
```
def decisionTree(x_train, x_test, y_train, y_test):
    dt=DecisionTreeClassifier()
    dt.fit(x_train,y_train.ravel())
    yPred = dt.predict(x_test)
    print('***DecisionTreeClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 24 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

In [24]:
```
def decisionTree(x_train, x_test, y_train, y_test):
    dt=DecisionTreeClassifier()
    dt.fit(x_train,y_train.ravel())
    yPred = dt.predict(x_test)
    print('***DecisionTreeClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```


In [25]:
```
def randomForest(x_train, x_test, y_train, y_test):
    rf = RandomForestClassifier()
    rf.fit(x_train,y_train.ravel())
    yPred = rf.predict(x_test)
    print('***RandomForestClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

In [26]:
```
def KNN(x_train, x_test, y_train, y_test):
    knn = KNeighborsClassifier()
    knn.fit(x_train,y_train.ravel())
    yPred = knn.predict(x_test)
    print('***KNeighborsClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 24 minutes ago (autosaved)

ile Edit View Insert Cell Kernel Widgets Help Trusted | Python 3

In [26]:
```python
def KNN(x_train, x_test, y_train, y_test):
    knn = KNeighborsClassifier()
    knn.fit(x_train,y_train.ravel())
    yPred = knn.predict(x_test)
    print('***KNeighborsClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

In [27]:
```python
def xgboost(x_train, x_test, y_train, y_test):
    xg = GradientBoostingClassifier()
    xg.fit(x_train,y_train.ravel())
    yPred = xg.predict(x_test)
    print('***GrandientBoostingClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
```

In [28]:
```python
def compareModel(x_train, x_test, y_train, y_test):
    decisionTree(x_train, x_test, y_train, y_test)
    print('-'*100)
    randomForest(x_train, x_test, y_train, y_test)
    print('-'*100)
    KNN(x_train, x_test, y_train, y_test)
    print('-'*100)
    xgboost(x_train, x_test, y_train, y_test)
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 24 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted

In [29]:
```python
compareModel(x_train, x_test, y_train, y_test)
```

```
***DecisionTreeClassifier***
Confusion matrix
[[62 12]
 [13  4]]
Classification report
              precision    recall  f1-score   support

           0       0.83      0.84      0.83        74
           1       0.25      0.24      0.24        17

    accuracy                           0.73        91
   macro avg       0.54      0.54      0.54        91
weighted avg       0.72      0.73      0.72        91

----------------------------------------------------------------------------------------------------
***RandomForestClassifier***
Confusion matrix
[[73  1]
 [17  0]]
Classification report
              precision    recall  f1-score   support

           0       0.81      0.99      0.89        74
           1       0.00      0.00      0.00        17

    accuracy                           0.80        91
   macro avg       0.41      0.49      0.45        91
```

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3

```
 [17  0]]
Classification report
              precision    recall  f1-score   support

           0       0.81      0.99      0.89        74
           1       0.00      0.00      0.00        17

    accuracy                           0.80        91
   macro avg       0.41      0.49      0.45        91
weighted avg       0.66      0.80      0.72        91

----------------------------------------------------------------------------------------------------
***KNeighborsClassifier***
Confusion matrix
[[74  0]
 [17  0]]
Classification report
              precision    recall  f1-score   support

           0       0.81      1.00      0.90        74
           1       0.00      0.00      0.00        17

    accuracy                           0.81        91
   macro avg       0.41      0.50      0.45        91
weighted avg       0.66      0.81      0.73        91

----------------------------------------------------------------------------------------------------
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 25 minutes ago (autosaved) Logo

File Edit View Insert Cell Kernel Widgets Help Trusted | Python 3

```
***GrandientBoostingClassifier***
Confusion matrix
[[70  4]
 [17  0]]
Classification report
              precision    recall  f1-score   support

           0       0.80      0.95      0.87        74
           1       0.00      0.00      0.00        17

    accuracy                           0.77        91
   macro avg       0.40      0.47      0.43        91
weighted avg       0.65      0.77      0.71        91
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 2 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

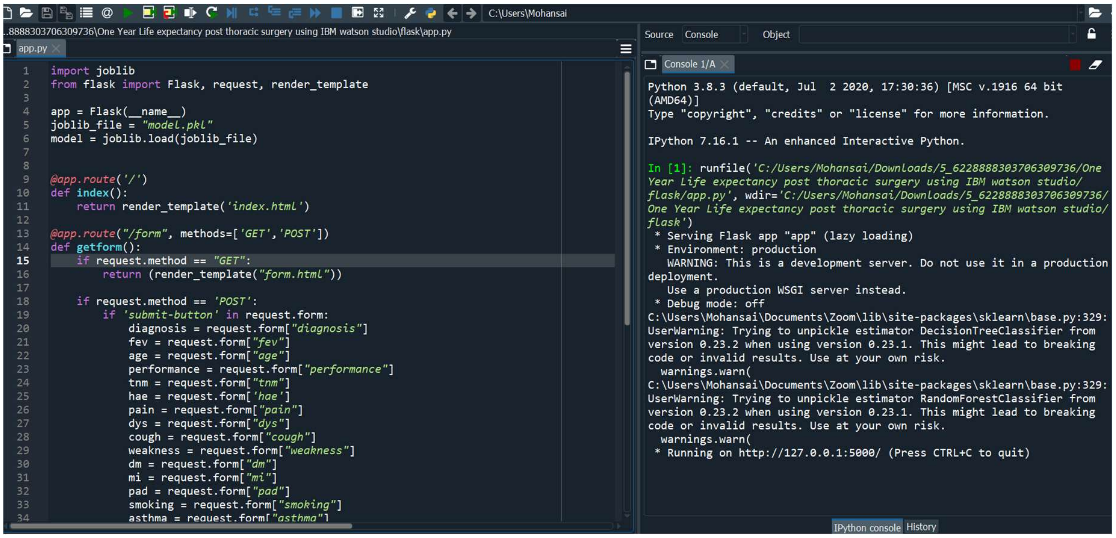
```
In [30]: from sklearn.model_selection import cross_val_score
         rf= RandomForestClassifier()
         rf.fit(x_train,y_train.ravel())
         yPred = rf.predict(x_test)

In [31]: f1_score(yPred,y_test,average='weighted')
Out[31]: 0.8476461809795143

In [32]: cv = cross_val_score(rf,x,y.ravel(),cv=5)

In [33]: np.mean(cv)
Out[33]: 0.8414163614163614

In [34]: dt = DecisionTreeClassifier()
         dt.fit(x_train,y_train)
         yPred_dt = dt.predict(x_test)

In [35]: f1_score(yPred_dt,y_test,average='weighted')
Out[35]: 0.7358388247087242

In [36]: cv = cross_val_score(rf,x,y.ravel(),cv=5)
```

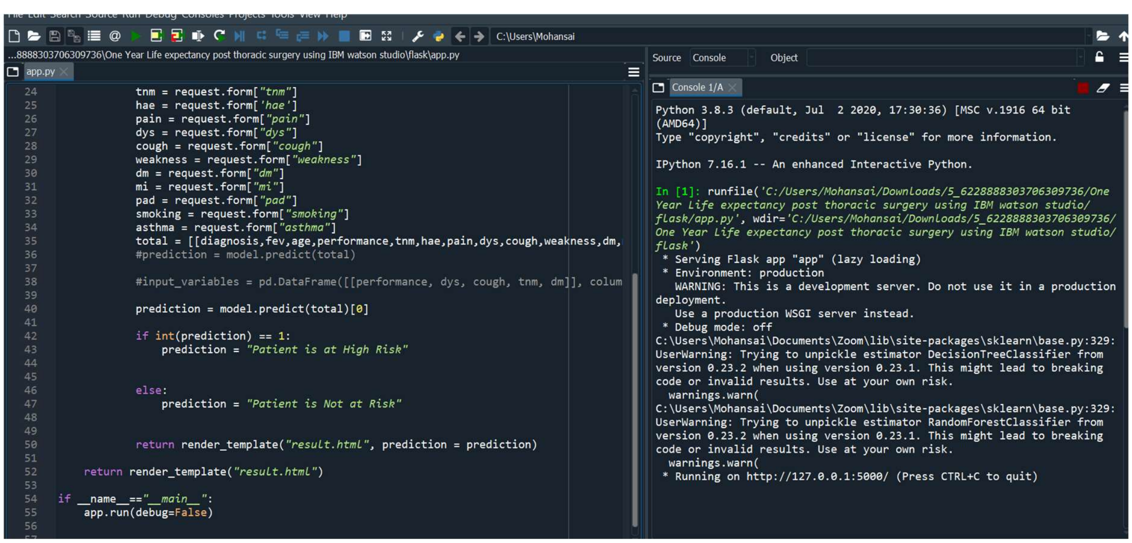
```
In [37]: np.mean(cv)
Out[37]: 0.8414163614163614

In [38]: kn = KNeighborsClassifier()
         kn.fit(x_train,y_train.ravel())
         yPred_kn = kn.predict(x_test)

In [39]: f1_score(yPred_kn,y_test,average='weighted')
Out[39]: 0.896969696969697

In [40]: cv = cross_val_score(kn,x,y.ravel(),cv=5)

In [41]: np.mean(cv)
Out[41]: 0.8326495726495727

In [42]: gb = GradientBoostingClassifier()
         gb.fit(x_train,y_train.ravel())
         yPred_gb = gb.predict(x_test)

In [43]: f1_score(yPred_gb,y_test,average='weighted')
Out[43]: 0.8313425704730053
```

jupyter ONE YEAR LIFE EXPECTANCY POST THORACIC SURGERY Last Checkpoint: 3 minutes ago (autosaved)

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

```
In [44]: cv = cross_val_score(gb,x,y.ravel(),cv=5)

In [45]: np.mean(cv)
Out[45]: 0.8260073260073261

In [46]: x_test[4]
Out[46]: array([ 3. ,  3.2,  0. ,  0. ,  0. ,  0. ,  1. ,  0. ,  2. ,  0. ,  0. ,
                 0. ,  1. ,  0. , 55. ])

In [47]: gb.predict([[3,4.08,0,0,0,0 ,1 ,0,2,0,0,0,1,0,55]])
Out[47]: array([0], dtype=int64)

In [48]: rf.predict([[3,4.08,0,0,0,0 ,1 ,0,2,0,0,0,1,0,55]])
Out[48]: array([0], dtype=int64)

In [49]: dt.predict([[3,4.08,0,0,0,0 ,1 ,0,2,0,0,0,1,0,55]])
Out[49]: array([0], dtype=int64)

In [50]: kn.predict([[3,4.08,0,0,0,0 ,1 ,0,2,0,0,0,1,0,55]])
Out[50]: array([0], dtype=int64)
```


C:\Users\Mohansai
..8888303706309736\One Year Life expectancy post thoracic surgery using IBM watson studio\flask\app.py
app.py
import joblib
from flask import Flask, request, render_template
app = Flask(__name__)
joblib_file = "model.pkl"
model = joblib.load(joblib_file)
@app.route('/')
def index():
return render_template('index.html')
@app.route("/form", methods=['GET','POST'])
def getform():
if request.method == "GET":
return (render_template("form.html"))
if request.method == 'POST':
if 'submit-button' in request.form:
diagnosis = request.form["diagnosis"]
fev = request.form["fev"]
age = request.form["age"]
performance = request.form["performance"]
tnm = request.form["tnm"]
hae = request.form['hae']
pain = request.form["pain"]
dys = request.form["dys"]
cough = request.form["cough"]
weakness = request.form["weakness"]
dm = request.form["dm"]
mi = request.form["mi"]
pad = request.form["pad"]
smoking = request.form["smoking"]
asthma = request.form["asthma"]
Source
Console
Object
Console 1/A
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 7.16.1 -- An enhanced Interactive Python.
In [1]: runfile('C:/Users/Mohansai/Downloads/5_6228888303706309736/One Year Life expectancy post thoracic surgery using IBM watson studio/flask/app.py', wdir='C:/Users/Mohansai/Downloads/5_6228888303706309736/One Year Life expectancy post thoracic surgery using IBM watson studio/flask')
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
C:\Users\Mohansai\Documents\Zoom\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator DecisionTreeClassifier from version 0.23.2 when using version 0.23.1. This might lead to breaking code or invalid results. Use at your own risk.
warnings.warn(
C:\Users\Mohansai\Documents\Zoom\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator RandomForestClassifier from version 0.23.2 when using version 0.23.1. This might lead to breaking code or invalid results. Use at your own risk.
warnings.warn(
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
IPython console
History

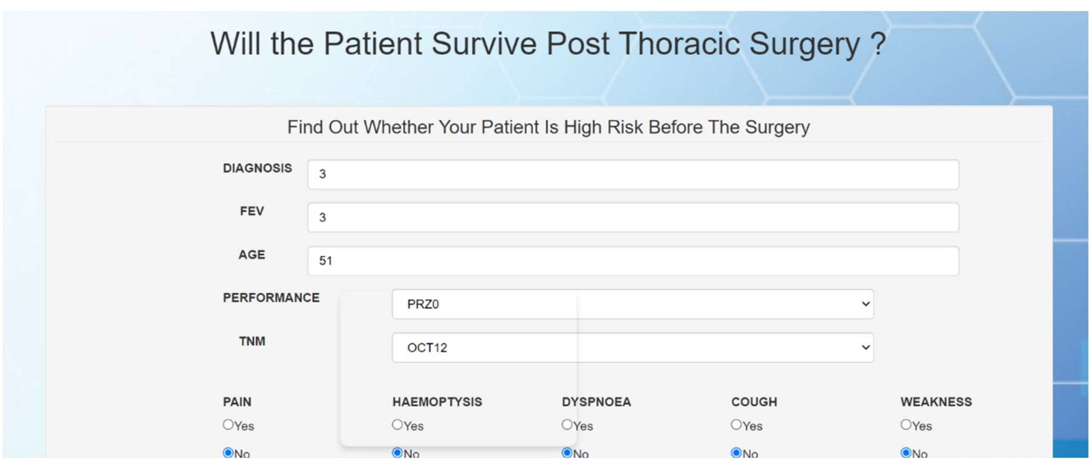

C:\Users\Mohansai
...8888303706309736\One Year Life expectancy post thoracic surgery using IBM watson studio\flask\app.py
app.py
tnm = request.form["tnm"]
hae = request.form['hae']
pain = request.form["pain"]
dys = request.form["dys"]
cough = request.form["cough"]
weakness = request.form["weakness"]
dm = request.form["dm"]
mi = request.form["mi"]
pad = request.form["pad"]
smoking = request.form["smoking"]
asthma = request.form["asthma"]
total = [[diagnosis,fev,age,performance,tnm,hae,pain,dys,cough,weakness,dm,
#prediction = model.predict(total)
#input_variables = pd.DataFrame([[performance, dys, cough, tnm, dm]], colum
prediction = model.predict(total)[0]
if int(prediction) == 1:
prediction = "Patient is at High Risk"
else:
prediction = "Patient is Not at Risk"
return render_template("result.html", prediction = prediction)
return render_template("result.html")
if __name__=="__main__":
app.run(debug=False)
Source
Console
Object
Console 1/A
Python 3.8.3 (default, Jul 2 2020, 17:30:36) [MSC v.1916 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.
IPython 7.16.1 -- An enhanced Interactive Python.
In [1]: runfile('C:/Users/Mohansai/Downloads/5_6228888303706309736/One Year Life expectancy post thoracic surgery using IBM watson studio/flask/app.py', wdir='C:/Users/Mohansai/Downloads/5_6228888303706309736/One Year Life expectancy post thoracic surgery using IBM watson studio/flask')
* Serving Flask app "app" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
C:\Users\Mohansai\Documents\Zoom\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator DecisionTreeClassifier from version 0.23.2 when using version 0.23.1. This might lead to breaking code or invalid results. Use at your own risk.
warnings.warn(
C:\Users\Mohansai\Documents\Zoom\lib\site-packages\sklearn\base.py:329: UserWarning: Trying to unpickle estimator RandomForestClassifier from version 0.23.2 when using version 0.23.1. This might lead to breaking code or invalid results. Use at your own risk.
warnings.warn(
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)

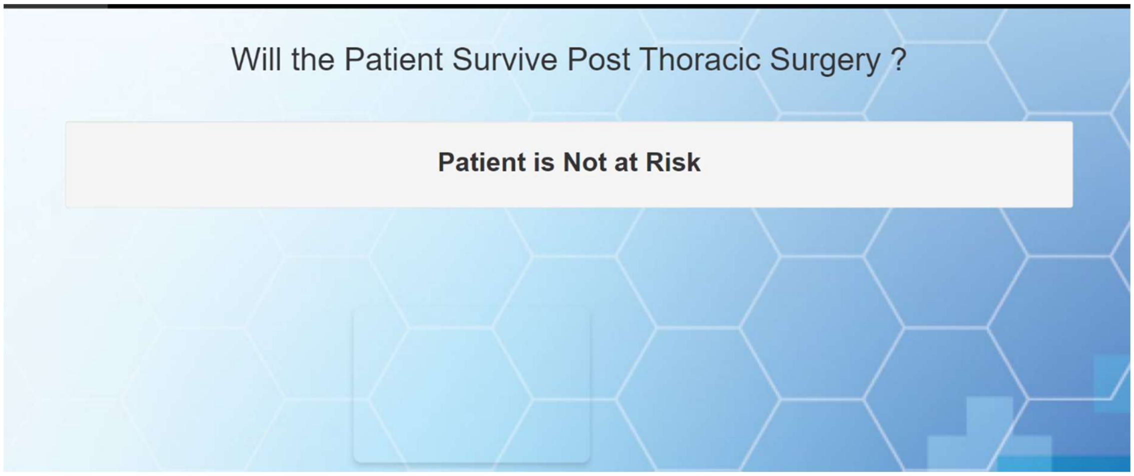

## LUNGEVITY

Lung cancer is the leading cause of cancer-related deaths in the world. In the United States, lung cancer claims more lives every year than colon cancer, prostate cancer, and breast cancer combined. Patients who receive Thoracic Surgery for Lung Cancer do so with the expectation that their lives will be prolonged for a sufficient amount of time after the surgery.

The problem this app tries to solve is to find whether there is a way to determine Post-Operative Life Expectancy from patient attributes in the dataset. If there is pattern to be recognized, this would help physicians and patients make a more educated decision on whether they should proceed forward with the surgery
Not only would this influence physicians and patients but also health insurance companies, national health organizations and clinical researchers.

View the APP | How was this made?






# 7.ADVANTAGES & DISADVANTAGES

| Procedure | Application | Advantages | Disadvantages |
|---|---|---|---|
| Rigid bronchoscopy | Diagnosis of carcinoma<br>Clear airway obstruction | Removal of secretions/blood<br>Large biopsy specimen may reveal submucosal tumor<br>Opportunity to debulk inoperable tumors or excise benign lesions<br>Assessment of carinal fixation | Poor view of secondary bronchial divisions |
| Cervical and scalene lymph node biopsy | Diagnosis of enlarged nodes | Safer technique than fine-needle aspiration (FNA) for deeper nodes<br>Large tissue sample | Operative procedure; occasional technical difficulty |
| Mediastinoscopy | Diagnosis of middle mediastinal pathology (node/mass) | Direct access to mediastinal abnormality<br>Much larger tissue sample than FNA | Operative field restricted to paratracheal and subcarinal areas |
| Mediastinotomy | Diagnosis of anterior mediastinal pathology (node/mass) | Reaches areas of anterior mediastinum inaccessible to mediastinoscopy<br>Useful for subaortic node sampling<br>May be safer to use than mediastinoscopy in patients who have superior vena cava obstruction<br>Much larger tissue sample than in FNA | Operative field restricted to upper anterior mediastinum<br>More complex operative procedure |
| Thoracoscopy | Pleural biopsy<br>Biopsy of visceral pleura deposits | Can be effected through single puncture<br>Visually directed biopsy | Limited to small biopsies<br>Dependent on pleural cavity being free of adhesions |
| Video-assisted thoracoscopic/ thoracic surgery (VATS) | Any ipsilateral biopsy or sampling procedure | Complementary to mediastinoscopy and mediastinotomy<br>Excellent view and full operative intervention possible (e.g., wedge excision of pulmonary nodule, dissection of mediastinal mass) | Dependent on pleural cavity being free of adhesions<br>Significant surgical expertise required |

## 8.CONCLUSION

Detection of lung cancer is one of the challenging problems in medical field due to structure of cancer cells, where most of the cells are overlapped to each other. Detection of lung cancer in the early stage is curable. The system contains two parts. One is Lung Cancer Detection part and the other is the Prediction of Life Expectancy Post Thoracic Surgery. Both the parts can run independently. The system is provided as a web application where anyone can upload a CT scan image of the lung in the front end and check out whether that image is infected with Lung Cancer. At the backend the image uploaded is preprocessed, segmented and predicted using the CNN model which is already trained. Also, the system provides a form requesting for the 14 attributes required to predict the survival span post Thoracic Surgery. Once the form is submitted, the form inputs are run on the LDA model and a response of whether the patient will survive or not is produced. CNN model used to detect lung cancer gave an accuracy of 95% and the LDA classifier gave the highest accuracy of 83.76% compared to other 3 algorithms used. The system considers only CT lung images as input, but further the system can be enhanced to take MRI (Magnetic Resonance Imaging) or PET (Position Emission Tomography) as input. Also, in the prediction of survival part, higher accuracies of the classifier can be achieved by considering large amount of datase.

## 9.FUTURE SCOPE

Thoracic surgery is living an enthusiastic era of thriving innovations. The changes in the fields of diagnostics, in the therapeutic options, the development of surgical techniques and the adoption of novel technologies has radically changed the horizons of our specialty. We must embrace the innovations, learn navigational bronchoscopy, understand the multiple targeted therapies, and learn new ways to operate on advanced cases. We will not be able to stay complacent with our current 3 ports video-assisted thoracoscopic surgery (VATS) technique with open instruments, therefore every surgeon needs to be watching for each latest development as it happens. In this brief manuscript we will summarize some of the most important development over the recent years and forecast how this will impact on the patients affected by thoracic malignancies. As thoracic surgeons we have to embraces these developments in order to redefine our specialty.

## 10.BIBILOGRAPHY

As we need to connect to internet for opening websites and gather information about the skills required for the jobs, we can also use other programming languages like C, C++, Java but its effective to use and also have predefined functions to use in the python language since the syntax for this programming language is simple to use and is more effective  comparatively. So we have used python for this project for accessing sites and recommended skills as per the present trending technologies

## APPENDIX

## A.Source code

```
import pandas as pd
import numpy as np
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingCl
assifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import f1_score
from sklearn.metrics import classification_report, confusion_matrix
import itertools
df=pd.read_csv("ThoracicSurgery.csv")
df.head()
df.columns
df.describe()
df.shape
df.info()
df.isnull().sum()
live  = df[df['Death_1yr'] == 0]
death = df[df['Death_1yr'] == 1]

cond = ['FVC', 'FEV1', 'Performance', 'Pain', 'Haemoptysis', 'Dyspnoea'
, 'Cough', 'Weakness',\
```

```
        'Tumor_Size', 'Diabetes_Mellitus', 'MI_6mo', 'PAD', 'Smoking',
'Asthma', 'Age']

l = [np.mean(live[c]) for c in cond]
d = [np.mean(death[c]) for c in cond]

ld = pd.DataFrame(data={'Attribute': cond, 'Live 1yr Mean': l, 'Death 1
yr Mean': d})
ld = ld.set_index('Attribute')


print('Death: {:d}, Live: {:d}'.format(len(death), len(live)))
print("1 year death: {:.2f}% out of 454 patients".format(np.mean(df.Dea
th_1yr)*100))
ld
d = np.array(d)
l = np.array(l)

p_diff = (d-l)/l*100

fig, axes = plt.subplots(2,1,figsize=(12,18))

axes[0].bar(cond, p_diff)
axes[0].set_title('Mean Difference % between Dead and Live 1yr', fontsi
ze=18)
axes[0].set_xticks(cond)
axes[0].set_xticklabels(cond, rotation=90)
axes[0].set_ylabel('Percent', fontsize=13)

tf_col = ['Pain', 'Haemoptysis', 'Dyspnoea', 'Cough', 'Weakness', 'Diab
etes_Mellitus', 'MI_6mo', 'PAD', 'Smoking', 'Asthma']
tf_sum = [df[col].sum()/454 for col in tf_col]

axes[1].bar(tf_col, tf_sum)
axes[1].set_xticks(tf_col)
axes[1].set_xticklabels(tf_col, rotation=90)
axes[1].set_ylabel('Proportion of Total Patients', fontsize=13)
axes[1].set_title('Proportion of Patient Conditions before Surgery', fo
ntsize=18)

plt.tight_layout()

plt.show()
fig, axes = plt.subplots(3,1,figsize=(10,15))

sns.countplot(x='Diagnosis', hue='Death_1yr', data=df, palette='Blues_d
', ax=axes[0]).set_title('Diagnosis', fontsize=18)
sns.countplot(x='Tumor_Size', hue='Death_1yr', data=df, palette='Blues_
d', ax=axes[1]).set_title('Tumor_Size', fontsize=18)
sns.countplot(x='Performance', hue='Death_1yr', data=df, palette='Blues
_d', ax=axes[2]).set_title('Performance', fontsize=18)

plt.tight_layout()
def permutation_sample(data1, data2):
    """Generate a permutation sample from two data sets."""
    data = np.concatenate((data1, data2))
    permuted_data = np.random.permutation(data)
```

```
    perm_sample_1 = permuted_data[:len(data1)]
    perm_sample_2 = permuted_data[len(data1):]

    return perm_sample_1, perm_sample_2

def draw_perm_reps(data_1, data_2, func, size=1):
    """Generate multiple permutation replicates."""
    perm_replicates = np.empty(size)

    for i in range(size):
        perm_sample_1, perm_sample_2 = permutation_sample(data_1, data_
2)
        perm_replicates[i] = func(perm_sample_1, perm_sample_2)

    return perm_replicates

def diff_of_means(data_1, data_2):
    """Difference in means of two arrays."""
    diff = np.mean(data_1) - np.mean(data_2)
    return diff
condition = ['FVC', 'FEV1', 'Performance', 'Pain', 'Haemoptysis', 'Dysp
noea', 'Cough', 'Weakness',\
             'Tumor_Size', 'Diabetes_Mellitus', 'MI_6mo', 'PAD', 'Smoki
ng', 'Asthma', 'Age']
p_val = []

for c in condition:
    empirical_diff_means = diff_of_means(death[c], live[c])
    perm_replicates = draw_perm_reps(death[c], live[c], diff_of_means,
size=10000)
    if empirical_diff_means > 0:
        p = np.sum(perm_replicates >= empirical_diff_means) / len(perm_
replicates)
        p_val.append(p)
    else:
        p = np.sum(perm_replicates <= empirical_diff_means) / len(perm_
replicates)
        p_val.append(p)

print(list(zip(condition, p_val)))
fig, axes = plt.subplots(1,2,figsize=(13,5))
axes[0].plot(df.FVC, df.FEV1, linestyle='none', marker='.')

axes[0].set_xlabel('FVC', fontsize=13)
axes[0].set_ylabel('FEV1', fontsize=13)
axes[0].set_title('FVC vs FEV1', fontsize=16)

axes[1].plot(df.Age, df.FEV1, linestyle='none', marker='.', label='FEV1
')
axes[1].plot(df.Age, df.FVC, linestyle='none', marker='.', label='FVC')
axes[1].set_xlabel('Age', fontsize=13)
axes[1].set_ylabel('FEV1, FVC', fontsize=13)
axes[1].legend()
axes[1].set_title('Age vs FEV1, FVC', fontsize=16)

plt.tight_layout()
bnp.corrcoef(df.FVC, df.FEV1)[0,1]
np.corrcoef(df.Age, df.FVC)[0,1]
```

```
np.corrcoef(df.Age, df.FEV1)[0,1]
def ecdf(data):
    """Compute ECDF for a one-dimensional array of measurements."""
    n = len(data)
    x = np.sort(data)
    y = np.arange(1, n+1) / n
    return x, y
x_fvc, y_fvc = ecdf(df.FVC)
x_fev1, y_fev1 = ecdf(df.FEV1)
x_age, y_age = ecdf(df.Age)

fig, axes = plt.subplots(1,2,figsize=(13,5))
axes[0].plot(x_fvc, y_fvc, marker='.', linestyle='none', label='FVC')
axes[0].plot(x_fev1, y_fev1, marker='.', linestyle='none', label='FEV1'
)

axes[0].set_xlabel('Numerical Value', fontsize=13)
axes[0].set_ylabel('ECDF', fontsize=13)
axes[0].legend(loc='upper left')
axes[0].set_title('ECDF of FVC & FEV1', fontsize=16)

axes[1].plot(x_age, y_age, marker='.', linestyle='none', label='Age')
axes[1].set_xlabel('Years Old', fontsize=13)
axes[1].set_ylabel('ECDF', fontsize=13)
axes[1].legend(loc='upper left')
axes[1].set_title('ECDF of Age', fontsize=16)

plt.tight_layout()
df.drop(['FVC'],axis=1, inplace=True)
x=df.iloc[:,0:15].values
y=df.iloc[:,15:16].values
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random
_state=0)
print('Shape of x_train {}'.format(x_train.shape))
print('Shape of y_train {}'.format(y_train.shape))
print('Shape of x_test {}'.format(x_test.shape))s
print('Shape of y_test {}'.format(y_test.shape))
def decisionTree(x_train, x_test, y_train, y_test):
    dt=DecisionTreeClassifier()
    dt.fit(x_train,y_train.ravel())
    yPred = dt.predict(x_test)
    print('***DecisionTreeClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
def randomForest(x_train, x_test, y_train, y_test):
    rf = RandomForestClassifier()
    rf.fit(x_train,y_train.ravel())
    yPred = rf.predict(x_test)
    print('***RandomForestClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
def KNN(x_train, x_test, y_train, y_test):
    knn = KNeighborsClassifier()
    knn.fit(x_train,y_train.ravel())
```

```
    yPred = knn.predict(x_test)
    print('***KNeighborsClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
def xgboost(x_train, x_test, y_train, y_test):
    xg = GradientBoostingClassifier()
    xg.fit(x_train,y_train.ravel())
    yPred = xg.predict(x_test)
    print('***GrandientBoostingClassifier***')
    print('Confusion matrix')
    print(confusion_matrix(y_test,yPred))
    print('Classification report')
    print(classification_report(y_test,yPred))
def compareModel(x_train, x_test, y_train, y_test):
    decisionTree(x_train, x_test, y_train, y_test)
    print('-'*100)
    randomForest(x_train, x_test, y_train, y_test)
    print('-'*100)
    KNN(x_train, x_test, y_train, y_test)
    print('-'*100)
    xgboost(x_train, x_test, y_train, y_test)
compareModel(x_train, x_test, y_train, y_test)
from sklearn.model_selection import cross_val_score
rf= RandomForestClassifier()
rf.fit(x_train,y_train.ravel())
yPred = rf.predict(x_test)
f1_score(yPred,y_test,average='weighted')
cv = cross_val_score(rf,x,y.ravel(),cv=5)
np.mean(cv)
import pickle
pickle.dump(rf,open('model.pkl','wb'))
```