UPLOADING DATASET:

Dataset: https://www.kaggle.com/imkrkannan/lung-cancer-dataset-by-staceyinrobert/kernels

```
from google.colab import files
uploaded = files.upload()
```

Choose Files survey lung cancer.csv

- **survey lung cancer.csv**(application/vnd.ms-excel) - 11277 bytes, last modified: 7/9/2020 - 100% don

```
Saving survey lung cancer.csv to survey lung cancer (1).csv
```

IMPORTS:

```
import pandas as pd
import numpy as np
import tensorflow as tf
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.feature_selection import SelectKBest, chi2
from sklearn.preprocessing import MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
```

LOADING DATASET:

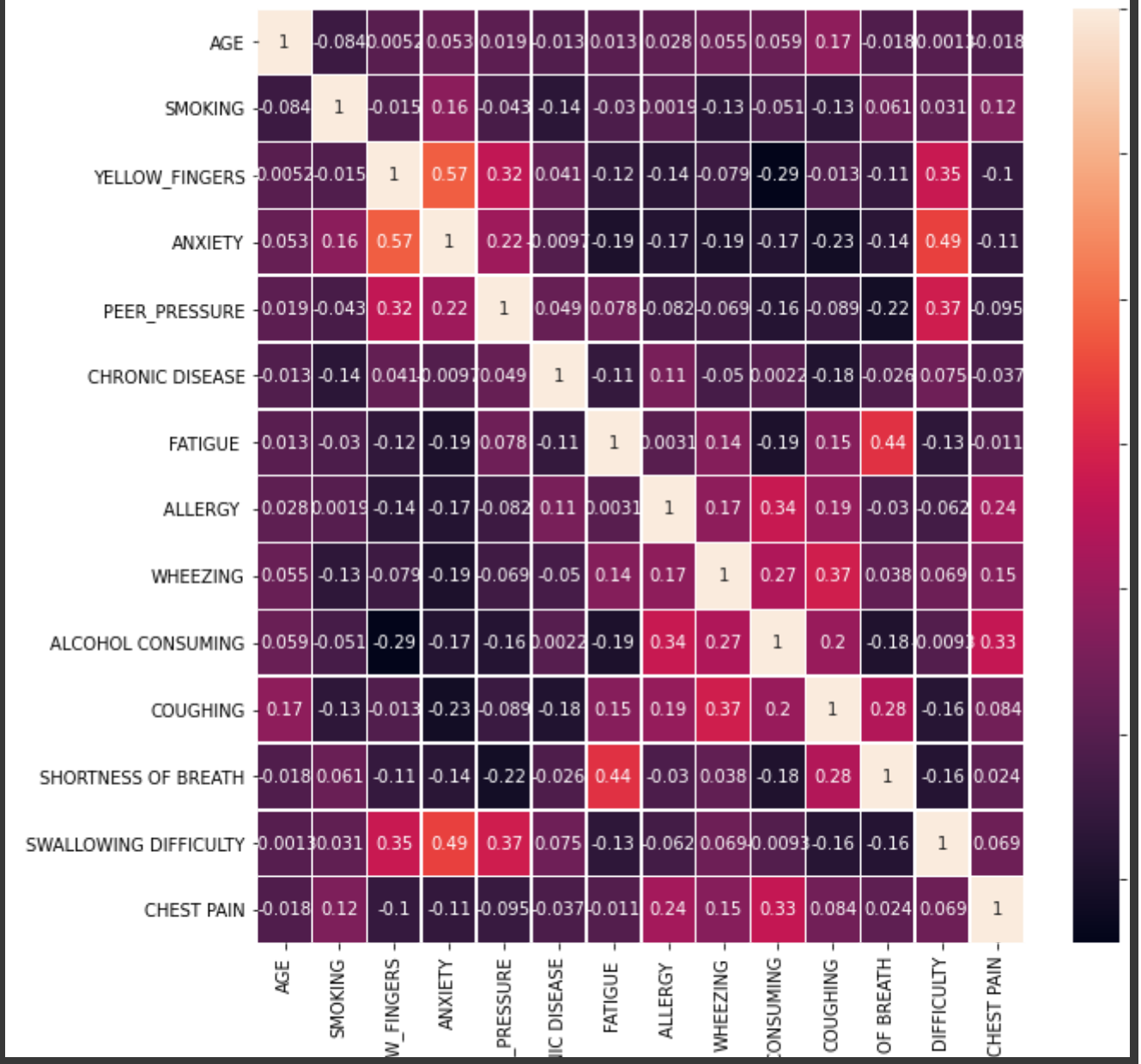
```
data = pd.read_csv('survey lung cancer.csv')
features = data.columns
```

EXPLORING THE DATA:

Correlation

```
%matplotlib inline
plt.figure(figsize = (10,10))
sns.heatmap(data.corr(),annot=True , linewidths=.5)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f186f22bc18>
```



RE-ENCODING GENDER and STATUS

```
data = data.values
```

```
data.T[0] = np.array([0 if i=='M' else 1 for i in data.T[0]])
data.T[-1] = np.array([0 if i=='NO' else 1 for i in data.T[-1]])
```

FEATURE-WISE EXPLORATION

```
print(features)
print(len(features))
```

```
Index(['GENDER', 'AGE', 'SMOKING', 'YELLOW_FINGERS', 'ANXIETY',
       'PEER_PRESSURE', 'CHRONIC DISEASE', 'FATIGUE ', 'ALLERGY ', 'WHEEZING',
```

```
Y = data.T[-1]
X = data.T[:-1]

print('Data',X.shape,'\n', X)
print('\n\nY',Y.shape,'\n', Y)
```

```
Data (15, 309)
 [[0 0 1 ... 0 0 0]
 [69 74 59 ... 58 67 62]
 [1 2 1 ... 2 2 1]
 ...
 [2 2 2 ... 1 2 1]
 [2 2 1 ... 1 1 2]
 [2 2 2 ... 2 2 1]]


Y (309,)
 [1 1 0 0 0 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1 0 1 1 1 1 0 1 0 0 1 0 1 0 1 1
 0 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 0 0 1 0 0 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 0
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 1 1 1 1 1 1
 1 1 1 0 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1 0 0 0 1 1 1 0 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

CHI-Square Method:

```
print(X.shape, Y.shape)
X = X.T
print(X.shape, Y.shape)
```

```
(15, 309) (309,)
(309, 15) (309,)
```

```
scaler = MinMaxScaler()
scaler.fit(X)
x_norm = scaler.transform(X)
feature_scores = []
fs = SelectKBest(score_func=chi2, k='all')
fs.fit(x_norm, np.array([y for y in Y]))
for i in range(len(fs.scores_)):
  print('Feature %d: %f' % (i, fs.scores_[i]))
  feature_scores.append(fs.scores_[i])
```

```
Feature 0: 0.732746
Feature 1: 0.060419
Feature 2: 0.456946
Feature 3: 4.373548
Feature 4: 3.256499
Feature 5: 5.350014
Feature 6: 1.881416
Feature 7: 2.292936
Feature 8: 14.717954
```

DISCARDING USELESS FEATURES in X_red

```
Feature 12: 0.409497
```

```
X_red = np.array([X.T[3], X.T[5], X.T[8], X.T[9], X.T[10], X.T[11], X.T[13], X.T[14],])
X_red = X_red.T
print(X_red.shape)
```

```
(309, 8)
```

USING PCA in X_pca

```
pca = PCA(n_components=2)
pca.fit(X)
print(pca.explained_variance_ratio_)
X_pca = pca.fit_transform(X)
print(X_pca.shape)
```

```
[0.95172303 0.0099867 ]
(309, 2)
```

CLASSIFYING USING REDUCED X:

```
X_red = np.asarray(X_red).astype(np.float32)
Y = np.asarray(Y).astype(np.float32)
```

Simple Perceptron

On X_red

```
model = tf.keras.models.Sequential([
                                    tf.keras.layers.Dense(1, activation='sigmoid', input_s
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_red, Y, epochs=1000, verbose=0, validation_split=0.2)
model.evaluate(X_red, Y)
model.summary()
```

```
10/10 [==============================] - 0s 2ms/step - loss: 0.2809 - accuracy: 0.873
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_7 (Dense)              (None, 1)                 9
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```

On X_pca

```
model = tf.keras.models.Sequential([
                                    tf.keras.layers.Dense(1, activation='sigmoid', input_s
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_pca, Y, epochs=1000, verbose=0, validation_split=0.2)
model.evaluate(X_pca, Y)
model.summary()
```

```
10/10 [==============================] - 0s 2ms/step - loss: 0.3744 - accuracy: 0.873
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_8 (Dense)              (None, 1)                 3
=================================================================
Total params: 3
Trainable params: 3
Non-trainable params: 0
_________________________________________________________________
```

ARTIFICIAL NEURAL NETWORK

X_red

```
model = tf.keras.models.Sequential([
                                    tf.keras.layers.Dense(4, activation='sigmoid', input_s
                                    tf.keras.layers.Dense(2, activation='sigmoid'),
                                    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_red, Y, epochs=1000, verbose=0, validation_split=0.2)
model.evaluate(X_red, Y)
model.summary()
```

```
10/10 [==============================] - 0s 1ms/step - loss: 0.2393 - accuracy: 0.873
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_9 (Dense)              (None, 4)                 36
_________________________________________________________________
dense_10 (Dense)             (None, 2)                 10
_________________________________________________________________
dense_11 (Dense)             (None, 1)                 3
```

```
model = tf.keras.models.Sequential([
                                    tf.keras.layers.Dense(2, activation='sigmoid', input_s
                                    tf.keras.layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_pca, Y, epochs=1000, verbose=0, validation_split=0.2)
model.evaluate(X_pca, Y)
model.summary()
```

```
10/10 [==============================] - 0s 2ms/step - loss: 0.3738 - accuracy: 0.873
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_12 (Dense)             (None, 2)                 6
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 3
=================================================================
Total params: 9
Trainable params: 9
Non-trainable params: 0
_________________________________________________________________
```

NAIVE BAYES:

X_red

```
gnb = GaussianNB()
y_pred = gnb.fit(X_red[:int(0.8*X_red.shape[0])], Y[:int(0.8*X_red.shape[0])]).predict(X_r
accuracy = 1 - np.mean(np.abs(y_pred - Y[int(0.8*X_red.shape[0]):]))
print(accuracy)
```

```
0.9032258093357086
```

X_pca

```
gnb = GaussianNB()
y_pred = gnb.fit(X_pca[:int(0.8*X_red.shape[0])], Y[:int(0.8*X_red.shape[0])]).predict(X_p
accuracy = 1 - np.mean(np.abs(y_pred - Y[int(0.8*X_red.shape[0]):]))
print(accuracy)
```

0.838709682226181

KNN

X_red

```
neigh = KNeighborsClassifier(n_neighbors=int(X_red.shape[0]*0.1))
neigh.fit(X_red[:int(0.8*X_red.shape[0])], Y[:int(0.8*X_red.shape[0])])
y_pred = neigh.predict(X_red[int(0.8*X_red.shape[0]):])
accuracy = 1 - np.mean(np.abs(y_pred - Y[int(0.8*X_red.shape[0]):]))
print(accuracy)
```

```
0.9193548411130905
```

X_pca

```
neigh = KNeighborsClassifier(n_neighbors=int(X_red.shape[0]*0.1))
neigh.fit(X_pca[:int(0.8*X_red.shape[0])], Y[:int(0.8*X_red.shape[0])])
y_pred = neigh.predict(X_pca[int(0.8*X_red.shape[0]):])
accuracy = 1 - np.mean(np.abs(y_pred - Y[int(0.8*X_red.shape[0]):]))
print(accuracy)
```

```
0.838709682226181
```

RANDOM FOREST:

X_red

```
clf = RandomForestClassifier(max_depth=8, random_state=0)
clf.fit(X_red[:int(0.8*X_red.shape[0])], Y[:int(0.8*X_red.shape[0])])
y_pred = clf.predict(X_red[int(0.8*X_red.shape[0]):])
accuracy = 1 - np.mean(np.abs(y_pred - Y[int(0.8*X_red.shape[0]):]))
print(accuracy)
```

```
0.9354838728904724
```

X_pca

```
clf = RandomForestClassifier(max_depth=8, random_state=0)
clf.fit(X_pca[:int(0.8*X_red.shape[0])], Y[:int(0.8*X_red.shape[0])])
y_pred = clf.predict(X_pca[int(0.8*X_red.shape[0]):])
accuracy = 1 - np.mean(np.abs(y_pred - Y[int(0.8*X_red.shape[0]):]))
print(accuracy)
```

```
0.8225806504487991
```