```
#import libraries
import pandas as pd
import numpy as np
from scipy import stats
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
import warnings

#suppress warnings
warnings.filterwarnings("ignore")
```

```
#import data
cancer_data = pd.read_csv('/archive (5).zip')
```

```
#look at formatting of entries
cancer_data.head()
```

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLER
0	M	69	1	2	2	1	1	2	
1	M	74	2	1	1	1	2	2	
2	F	59	1	1	1	2	1	2	
3	M	63	2	2	2	1	1	1	
4	F	63	1	2	1	1	1	1	

Next steps: Generate code with `cancer_data` View recommended plots

```
#display null values and data types
cancer_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 309 entries, 0 to 308
Data columns (total 16 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   GENDER                 309 non-null    object
 1   AGE                    309 non-null    int64
 2   SMOKING                309 non-null    int64
 3   YELLOW_FINGERS         309 non-null    int64
 4   ANXIETY                309 non-null    int64
 5   PEER_PRESSURE          309 non-null    int64
 6   CHRONIC DISEASE        309 non-null    int64
 7   FATIGUE                309 non-null    int64
 8   ALLERGY                309 non-null    int64
 9   WHEEZING               309 non-null    int64
 10  ALCOHOL CONSUMING      309 non-null    int64
 11  COUGHING               309 non-null    int64
 12  SHORTNESS OF BREATH    309 non-null    int64
 13  SWALLOWING DIFFICULTY  309 non-null    int64
 14  CHEST PAIN             309 non-null    int64
 15  LUNG_CANCER            309 non-null    object
dtypes: int64(14), object(2)
memory usage: 38.8+ KB
```

```
cancer_data.describe()
```

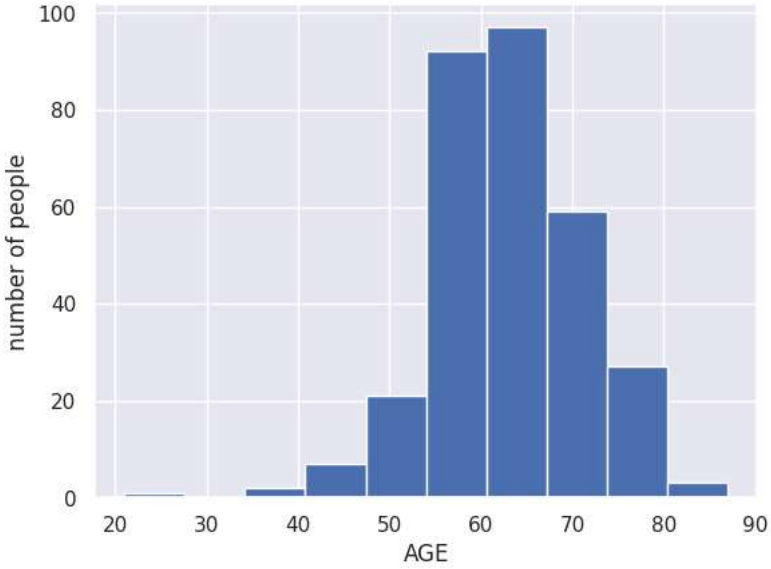
	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	F
count	309.000000	309.000000	309.000000	309.000000	309.000000	309.000000	309.
mean	62.673139	1.563107	1.569579	1.498382	1.501618	1.504854	1.
std	8.210301	0.496806	0.495938	0.500808	0.500808	0.500787	0.
min	21.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.
25%	57.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.
50%	62.000000	2.000000	2.000000	1.000000	2.000000	2.000000	2.
75%	69.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.
max	87.000000	2.000000	2.000000	2.000000	2.000000	2.000000	2.

```
cancer_data.columns
```

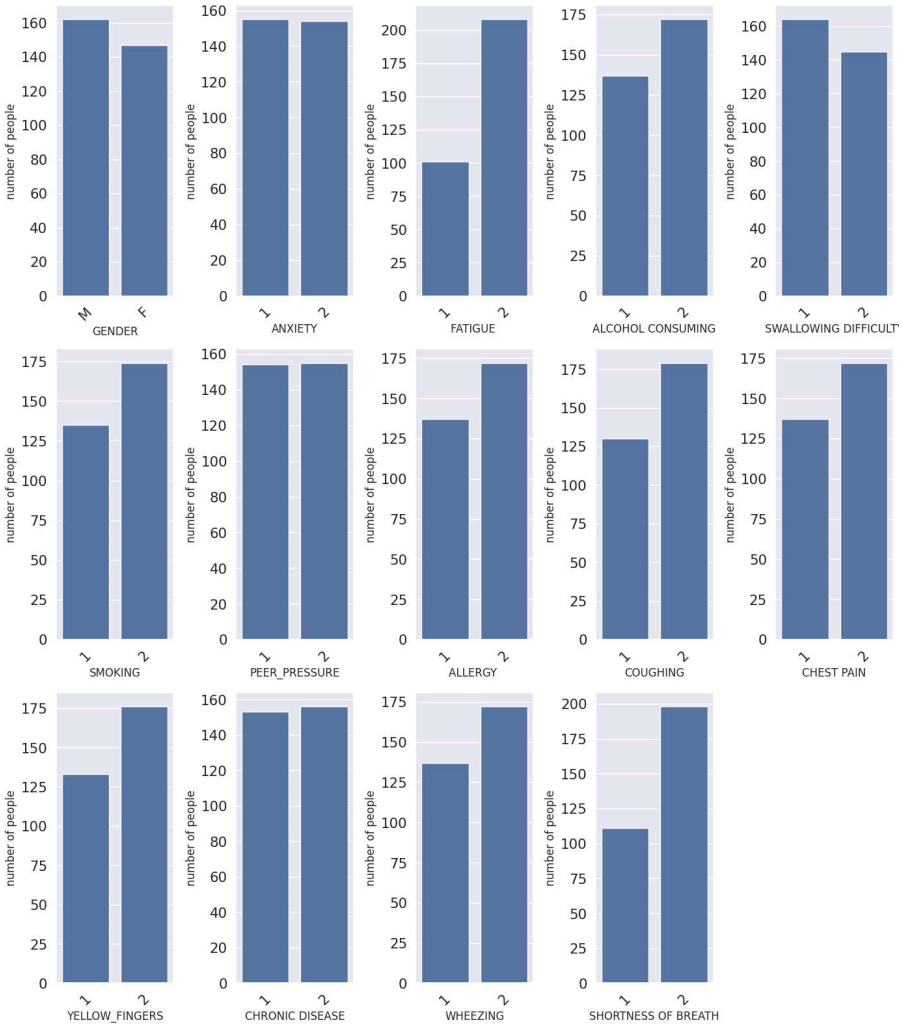
```
Index(['GENDER', 'AGE', 'SMOKING', 'YELLOW_FINGERS', 'ANXIETY',
       'PEER_PRESSURE', 'CHRONIC DISEASE', 'FATIGUE ', 'ALLERGY ', 'WHEEZING',
       'ALCOHOL CONSUMING', 'COUGHING', 'SHORTNESS OF BREATH',
       'SWALLOWING DIFFICULTY', 'CHEST PAIN', 'LUNG_CANCER'],
      dtype='object')
```

```
#numerical features
numerical = [
    'AGE'
]

#categorical features
categorical = [
    'GENDER',
    'SMOKING',
    'YELLOW_FINGERS',
    'ANXIETY',
    'PEER_PRESSURE',
    'CHRONIC DISEASE',
    'FATIGUE ',
    'ALLERGY ',
    'WHEEZING',
    'ALCOHOL CONSUMING',
    'COUGHING',
    'SHORTNESS OF BREATH',
    'SWALLOWING DIFFICULTY',
    'CHEST PAIN',
    'LUNG_CANCER'
]
```

```
#look at numerical data distribution
for i in cancer_data[numerical].columns:
    plt.hist(cancer_data[numerical][i])
    plt.xticks()
    plt.xlabel(i)
    plt.ylabel('number of people')
    plt.show()
```




```
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming 'cancer_data' is your DataFrame and 'categorical' is a list of categorical columns
num_plots = len(cancer_data[categorical].columns)
num_rows = 3  # Number of rows for subplots
num_cols = num_plots // num_rows  # Number of columns for subplots

if num_plots % num_rows != 0:
    num_cols += 1  # Adjust number of columns if there's a remainder

fig, axes = plt.subplots(num_rows, num_cols, figsize=(14, 16))  # Adjust figure size as needed

for i, column in enumerate(cancer_data[categorical].columns):
    row = i % num_rows
    col = i // num_rows
    sns.barplot(x=cancer_data[categorical][column].value_counts().index,
                y=cancer_data[categorical][column].value_counts(), ax=axes[row, col])
    axes[row, col].set_xlabel(column, fontsize=12)  # Adjust font size of x-axis label
    axes[row, col].set_ylabel('number of people', fontsize=12)  # Adjust font size of y-axis label
    axes[row, col].tick_params(axis='x', labelrotation=45)  # Rotate x-axis labels for better readability

# Hide any empty subplots
for i in range(num_plots, num_rows * num_cols):
    axes[i // num_cols, i % num_cols].axis('off')

plt.subplots_adjust(top=0.9)  # Adjust the top margin to leave space for the titles
plt.tight_layout()
plt.show()
```


number of people
GENDER
M
F
ANXIETY
FATIGUE
ALCOHOL CONSUMING
SWALLOWING DIFFICULT
SMOKING
PEER_PRESSURE
ALLERGY
COUGHING
CHEST PAIN
YELLOW_FINGERS
CHRONIC DISEASE
WHEEZING
SHORTNESS OF BREATH

```
X.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 309 entries, 0 to 308
Data columns (total 15 columns):
 #   Column                 Non-Null Count  Dtype 
---  ------                 --------------  ----- 
 0   GENDER                 309 non-null    object
 1   AGE                    309 non-null    int64 
 2   SMOKING                309 non-null    object
 3   YELLOW_FINGERS         309 non-null    object
 4   ANXIETY                309 non-null    object
 5   PEER_PRESSURE          309 non-null    object
 6   CHRONIC DISEASE        309 non-null    object
 7   FATIGUE                309 non-null    object
 8   ALLERGY                309 non-null    object
 9   WHEEZING               309 non-null    object
 10  ALCOHOL CONSUMING      309 non-null    object
 11  COUGHING               309 non-null    object
 12  SHORTNESS OF BREATH    309 non-null    object
 13  SWALLOWING DIFFICULTY  309 non-null    object
 14  CHEST PAIN             309 non-null    object
dtypes: int64(1), object(14)
memory usage: 36.3+ KB
```

```
X_mi = X.copy()
```

```
#label encoding for categorical variables
for colname in X_mi.select_dtypes("object"):
    X_mi[colname], _ = X_mi[colname].factorize()

X_mi['AGE']=X_mi['AGE'].astype('float64')

#all discrete features have int dtypes
discrete_features = X_mi.dtypes == int
```

```
discrete_features
```

```
GENDER                   True
AGE                     False
SMOKING                  True
YELLOW_FINGERS           True
ANXIETY                  True
PEER_PRESSURE            True
CHRONIC DISEASE          True
FATIGUE                  True
ALLERGY                  True
WHEEZING                 True
ALCOHOL CONSUMING        True
COUGHING                 True
SHORTNESS OF BREATH      True
SWALLOWING DIFFICULTY    True
CHEST PAIN               True
dtype: bool
```

```
#use classification since the target variable is discrete
from sklearn.feature_selection import mutual_info_classif

#define a function to produce mutual information scores
def make_mi_scores(X_mi, y, discrete_features):
    mi_scores = mutual_info_classif(X_mi, y, discrete_features=discrete_features)
    mi_scores = pd.Series(mi_scores, name="MI Scores", index=X_mi.columns)
    mi_scores = mi_scores.sort_values(ascending=False)
    return mi_scores

#compute mutual information scores
mi_scores = make_mi_scores(X_mi, y, discrete_features)
mi_scores
```

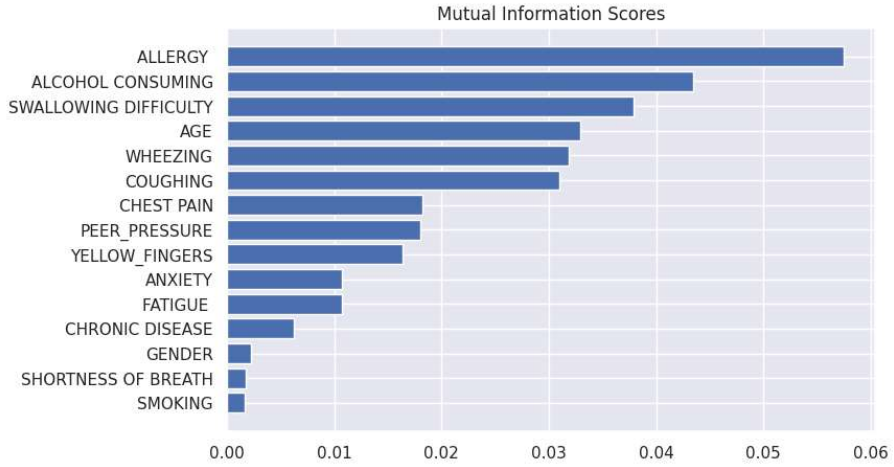
```
ALLERGY                  0.057504
ALCOHOL CONSUMING        0.043414
SWALLOWING DIFFICULTY    0.037858
WHEEZING                 0.031806
COUGHING                 0.031021
AGE                      0.019226
```

```
CHEST PAIN              0.018221
PEER_PRESSURE           0.018029
YELLOW_FINGERS          0.016365
ANXIETY                 0.010750
FATIGUE                 0.010725
CHRONIC DISEASE         0.006217
GENDER                  0.002261
SHORTNESS OF BREATH     0.001804
SMOKING                 0.001680
Name: MI Scores, dtype: float64
```

```
#use classification since the target variable is discrete
from sklearn.feature_selection import mutual_info_classif

#define a function to produce mutual information scores
def make_mi_scores(X_mi, y, discrete_features):
    mi_scores = mutual_info_classif(X_mi, y, discrete_features=discrete_features)
    mi_scores = pd.Series(mi_scores, name="MI Scores", index=X_mi.columns)
    mi_scores = mi_scores.sort_values(ascending=False)
    return mi_scores

#compute mutual information scores
mi_scores = make_mi_scores(X_mi, y, discrete_features)
mi_scores
```

```
ALLERGY                 0.057504
ALCOHOL CONSUMING       0.043414
SWALLOWING DIFFICULTY   0.037858
WHEEZING                0.031806
COUGHING                0.031021
AGE                     0.018838
CHEST PAIN              0.018221
PEER_PRESSURE           0.018029
YELLOW_FINGERS          0.016365
ANXIETY                 0.010750
FATIGUE                 0.010725
CHRONIC DISEASE         0.006217
GENDER                  0.002261
SHORTNESS OF BREATH     0.001804
SMOKING                 0.001680
Name: MI Scores, dtype: float64
```

```
#define a function to plot mutual information scores
def plot_mi_scores(scores):
    scores = scores.sort_values(ascending=True)
    width = np.arange(len(scores))
    ticks = list(scores.index)
    plt.barh(width, scores)
    plt.yticks(width, ticks)
    plt.title("Mutual Information Scores")

#plot the scores
plt.figure(dpi=100, figsize=(8, 5))
plot_mi_scores(mi_scores)
```




```
#import libraries
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
```

```
#get feature names
X = pd.concat([X[numerical],pd.get_dummies(X[categorical])],axis=1)
feature_names = X.columns

# train/test split with stratify making sure classes are evenlly represented across splits
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, train_size=0.75, random_state=1)

#define scaler
scaler=MinMaxScaler()

#apply preprocessing to split data with scaler
X_train[numerical] = scaler.fit_transform(X_train[numerical])
X_test[numerical] = scaler.transform(X_test[numerical])
```

```
from sklearn.model_selection import cross_val_score
from numpy import mean, std
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
#logistic regression with five-fold cross validation
lr = LogisticRegression(max_iter = 2000)
cv = cross_val_score(lr,X_train,y_train,cv=5)
print(mean(cv), '+/-', std(cv))
```

```
0.9308048103607771 +/- 0.0158848177577188
```

```
#random forest classifier with five-fold cross validation
rf = RandomForestClassifier(random_state = 1)
cv = cross_val_score(rf,X_train,y_train,cv=5)
print(mean(cv), '+/-', std(cv))
```

```
0.9350601295097132 +/- 0.013760232121339587
```

```
#support vector classifier with five-fold cross validation
svc = SVC(probability = True)
cv = cross_val_score(svc,X_train,y_train,cv=5)
print(mean(cv), '+/-', std(cv))
```

```
0.9351526364477335 +/- 0.023485469326540147
```

```
#ml algorithm tuner
from sklearn.model_selection import GridSearchCV

#performance reporting function
def clf_performance(classifier, model_name):
    print(model_name)
    print('Best Score: {} +/- {}'.format(str(classifier.best_score_),str(classifier.cv_results_['std_test_score'][classifier.best_index_])))
    print('Best Parameters: ' + str(classifier.best_params_))
```

```
#logistic regression performance tuner
lr = LogisticRegression()
param_grid = {'max_iter' : [15000],
              'C' : np.arange(.1,.6,.1)
             }
clf_lr = GridSearchCV(lr, param_grid = param_grid, cv = 5, n_jobs = -1)
best_clf_lr = clf_lr.fit(X_train,y_train)
clf_performance(best_clf_lr,'Logistic Regression')
```

```
Logistic Regression
Best Score: 0.9264569842738206 +/- 0.01712752904500742
Best Parameters: {'C': 0.30000000000000004, 'max_iter': 15000}
```

```
#random forest performance tuner
rf = RandomForestClassifier(random_state = 1)
param_grid =  {
                'n_estimators': np.arange(8,20,2),
                'bootstrap': [True,False], #bagging (T) vs. pasting (F)
                'max_depth': [10],
                'max_features': ['auto','sqrt'],
#                 'min_samples_leaf': np.arange(2,6,1),
#                 'min_samples_split': np.arange(2,6,1)
              }
clf_rf_rnd = GridSearchCV(rf, param_grid = param_grid, cv = 5, n_jobs = -1)
best_clf_rf_rnd = clf_rf_rnd.fit(X_train,y_train)
clf_performance(best_clf_rf_rnd,'Random Forest')
```

```
Random Forest
Best Score: 0.9394079555966698 +/- 0.021250436398270147
Best Parameters: {'bootstrap': True, 'max_depth': 10, 'max_features': 'auto', 'n_estimators': 10}
```

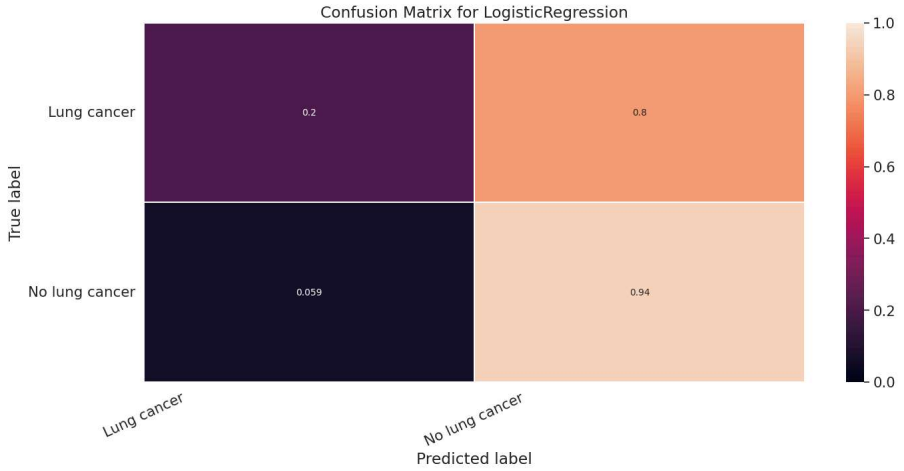
```
#support vector classifier performance tuner
svc = SVC(probability = True, random_state = 1)
param_grid = {
              'kernel': ['linear', 'poly', 'sigmoid','rbf'],
              'gamma': [1, 1e-1, 1e-2, 1e-3, 1e-4],
              'C': np.arange(40,70,5)
             }
clf_svc = GridSearchCV(svc, param_grid = param_grid, cv = 5, n_jobs = -1)
best_clf_svc = clf_svc.fit(X_train,y_train)
clf_performance(best_clf_svc,'Support Vector Classifier')
```

```
Support Vector Classifier
Best Score: 0.9438482886216466 +/- 0.016747588503435138
Best Parameters: {'C': 50, 'gamma': 1, 'kernel': 'linear'}
```

```
lr = LogisticRegression(C= 0.3, max_iter= 15000)
lr.fit(X_train, y_train)

y_pred_lr = lr.predict(X_test)

#assess accuracy
print('LogisticRegression test accuracy: {}'.format(accuracy_score(y_test, y_pred_lr)))
```

```
LogisticRegression test accuracy: 0.8461538461538461
```

```
#create and reshape confusion matrix data
matrix = confusion_matrix(y_test, y_pred_lr)
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

#plot as heatmap
plt.figure(figsize=(16,7))
sns.set(font_scale=1.4)
sns.heatmap(matrix, annot=True, annot_kws={'size':10},
            linewidths=0.2, vmin=0, vmax=1)

#plot settings
class_names = ['Lung cancer', 'No lung cancer']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
plt.yticks(tick_marks2, class_names, rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for LogisticRegression')
plt.show()
```




```
print('LogisticRegression')
print(classification_report(y_test, y_pred_lr))
```

```
     LogisticRegression
                  precision    recall  f1-score   support

              NO       0.33      0.20      0.25        10
             YES       0.89      0.94      0.91        68

        accuracy                           0.85        78
       macro avg       0.61      0.57      0.58        78
    weighted avg       0.82      0.85      0.83        78
```

```
#create support vector classifier model with tuned parameters
svc = SVC(probability = True, random_state = 1,C= 50, gamma = 1, kernel= 'linear')
svc.fit(X_train,y_train)
y_pred_svc = svc.predict(X_test)

‟
#create and reshape confusion matrix data
matrix = confusion_matrix(y_test, y_pred_svc)
matrix = matrix.astype('float') / matrix.sum(axis=1)[:, np.newaxis]

#plot as heatmap
plt.figure(figsize=(16,7))
sns.set(font_scale=1.4)
sns.heatmap(matrix, annot=True, annot_kws={'size':10},
            linewidths=0.2, vmin=0, vmax=1)

#plot settings
class_names = ['Lung cancer', 'No lung cancer']
tick_marks = np.arange(len(class_names))
tick_marks2 = tick_marks + 0.5
plt.xticks(tick_marks, class_names, rotation=25)
plt.yticks(tick_marks2, class_names, rotation=0)
plt.xlabel('Predicted label')
plt.ylabel('True label')
plt.title('Confusion Matrix for SVC')
plt.show()
```


```
print('SVC')
print(classification_report(y_test, y_pred_svc))
```

```
SVC
              precision    recall  f1-score   support

          NO       0.62      0.50      0.56        10
         YES       0.93      0.96      0.94        68

    accuracy                           0.90        78
   macro avg       0.78      0.73      0.75        78
```