Importing Necessery Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score, recall_score, f1_score, roc_auc_score
from imblearn.metrics import geometric_mean_score
from sklearn.metrics import classification_report, confusion_matrix
from imblearn.under_sampling import EditedNearestNeighbours
from imblearn.over_sampling import SMOTE
from imblearn.combine import SMOTEENN
from sklearn.linear_model import LogisticRegression
```

Loading the DataSet

```
df = pd.read_csv("Lung_Cancer_Dataset.csv")
```

```
df.head()
```

	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SHORTNESS OF BREATH	SWAL DIFF
0	M	69	1	2	2	1	1	2	1	2	2	2	2	
1	M	74	2	1	1	1	2	2	2	1	1	1	2	
2	F	59	1	1	1	2	1	2	1	2	1	2	2	
3	M	63	2	2	2	1	1	1	1	1	2	1	1	
4	F	63	1	2	1	1	1	1	1	2	1	2	2	

Next steps: Generate code with df | View recommended plots | New interactive sheet

```
df.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
```

```
le = LabelEncoder()
df["LUNG_CANCER"] = le.fit_transform(df["LUNG_CANCER"])
```

```
le = LabelEncoder()
df["GENDER"] = le.fit_transform(df["GENDER"])
```

```
df.head()
```

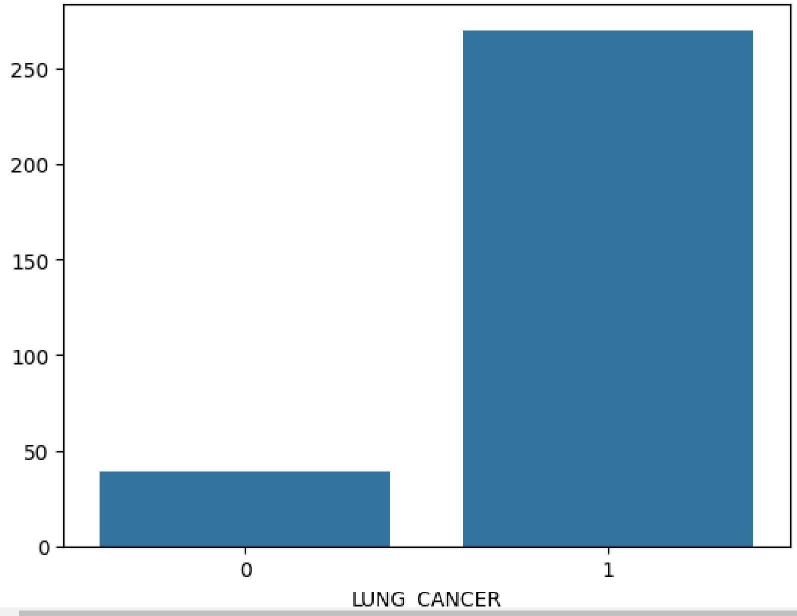
	GENDER	AGE	SMOKING	YELLOW_FINGERS	ANXIETY	PEER_PRESSURE	CHRONIC DISEASE	FATIGUE	ALLERGY	WHEEZING	ALCOHOL CONSUMING	COUGHING	SHORTNESS OF BREATH	SWAL DIFF
0	1	69	1	2	2	1	1	2	1	2	2	2	2	
1	1	74	2	1	1	1	2	2	2	1	1	1	2	
2	0	59	1	1	1	2	1	2	1	2	1	2	2	
3	1	63	2	2	2	1	1	1	1	1	2	1	1	
4	0	63	1	2	1	1	1	1	1	2	1	2	2	

Next steps: Generate code with df View recommended plots New interactive sheet

Checking Data is imbalanced

```
sns.barplot (x=df['LUNG_CANCER'].value_counts().index, y=df['LUNG_CANCER'].value_counts().values)
class_counts = df['LUNG_CANCER'].value_counts()
class_percentages = (class_counts / len(df)) * 100
print(class_percentages)
```

```
LUNG_CANCER
1    87.378641
0    12.621359
Name: count, dtype: float64
```



```
X = df.drop("LUNG_CANCER", axis=1)
y = df["LUNG_CANCER"]
```

Define Necessary Functions

```
def evaluate_model(y_true, y_pred, y_proba):
        print(f"Precision = {precision_score(y_true, y_pred)} ")
        print(f"Recall = {recall_score(y_true, y_pred)} ")
        print(f"F1-score = {f1_score(y_true, y_pred)} ")
        print(f"ROC AUC = {roc_auc_score(y_true, y_proba)} ")
        print(f"G-Mean = {geometric_mean_score(y_true, y_pred, average='binary')} ")
        return
```

```
def plot_confusion_matrix(cf_matrix):
    group_names = ['True Neg', 'False Pos', 'False Neg', 'True Pos']
    group_counts = ["{0:0.0f}".format(value) for value in cf_matrix.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in cf_matrix.flatten() / np.sum(cf_matrix)]

    labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names, group_counts, group_percentages)]
    labels = np.asarray(labels).reshape(2, 2)

    plt.figure(figsize=(5, 4))
    sns.heatmap(cf_matrix, annot=labels, fmt='', cmap='Blues', cbar=False)
    plt.title("Confusion Matrix")
    plt.xlabel("Predicted")
    plt.ylabel("Actual")
    plt.show()
```

Oversampling

```
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X, y)
```

```
print(y.value_counts())
print(y_resampled.value_counts())
```

```
LUNG_CANCER
1    270
0     39
Name: count, dtype: int64
LUNG_CANCER
1    270
0    270
Name: count, dtype: int64
```

```
X_train, X_test, y_train, y_test = train_test_split(X_resampled, y_resampled, test_size=0.3, random_state=42)
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
clf = LogisticRegression(random_state=101)
```

```
clf.fit(X_train_scaled, y_train)
```

```
▾ LogisticRegression  ⓘ ?
LogisticRegression(random_state=101)
```

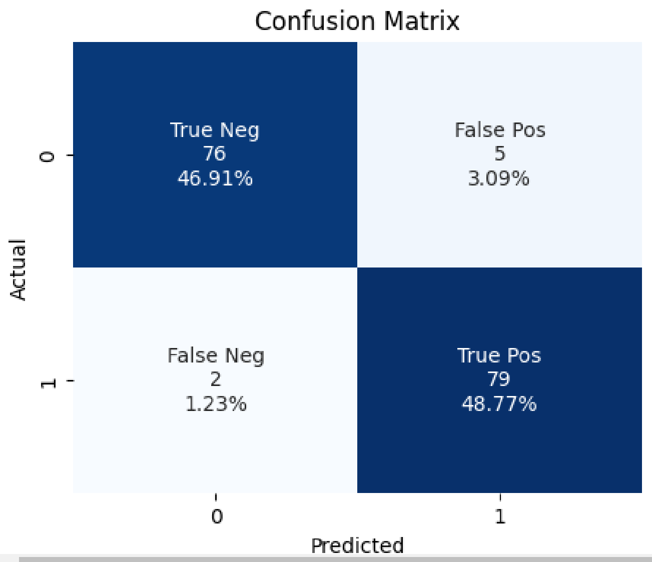
```
y_pred = clf.predict(X_test_scaled)
y_probs = clf.predict_proba(X_test_scaled)[:, 1]
```

```
evaluate_model(y_test, y_pred, y_probs)
```

```
Precision = 0.9404761904761905
Recall = 0.9753086419753086
F1-score = 0.9575757575757575
ROC AUC = 0.9900929736320683
G-Mean = 0.9566108952005193
```

```
cf_matrix = confusion_matrix(y_test, y_pred)
```

```
plot_confusion_matrix(cf_matrix)
```



Undersampling

```
enn = EditedNearestNeighbours()
X_enn, y_enn = enn.fit_resample(X, y)
```

```
print(y.value_counts())
print(y_enn.value_counts())
```

```
LUNG_CANCER
1    270
0     39
Name: count, dtype: int64
LUNG_CANCER
1    224
0     39
Name: count, dtype: int64
```

```
X_train, X_test, y_train, y_test = train_test_split(X_enn, y_enn, test_size=0.3, random_state=42)
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
clf1 = LogisticRegression(random_state=101)
clf1.fit(X_train_scaled, y_train)
```

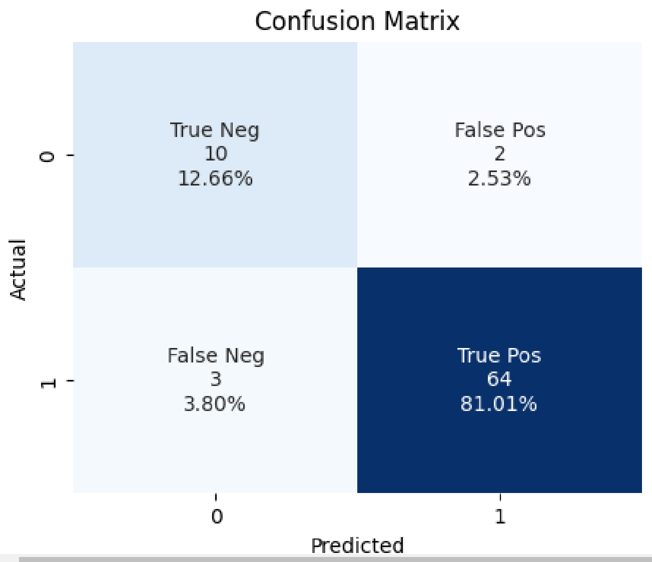
```
▾      LogisticRegression      ⓘ ?
LogisticRegression(random_state=101)
```

```
y_pred = clf1.predict(X_test_scaled)
y_probs = clf1.predict_proba(X_test_scaled)[:, 1]
```

```
evaluate_model(y_test, y_pred, y_probs)
```

```
Precision = 0.9696969696969697
Recall = 0.9552238805970149
F1-score = 0.9624060150375939
ROC AUC = 0.9713930348258707
G-Mean = 0.8921994734909411
```

```
cf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cf_matrix)
```



Ensamble Method

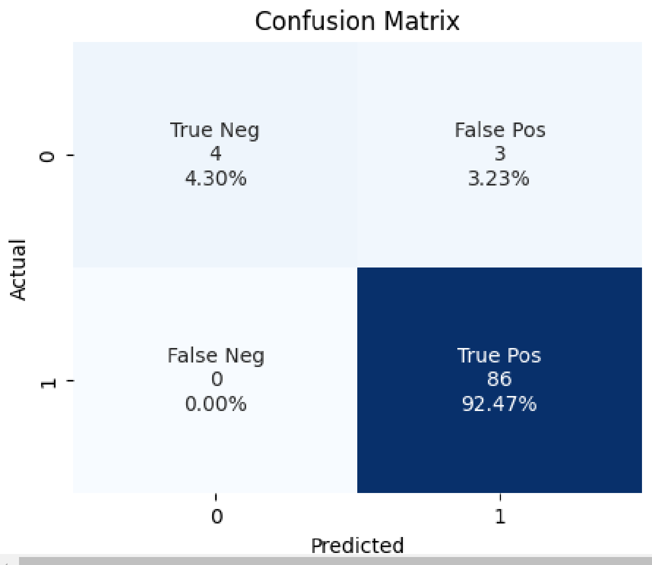
```
from sklearn.ensemble import AdaBoostClassifier


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


ada = AdaBoostClassifier(random_state=42)
ada.fit(X_train_scaled, y_train)
```

```
▾     AdaBoostClassifier     i ?
AdaBoostClassifier(random_state=42)
```

```
ada_pred = ada.predict(X_test_scaled)
ada_probs = ada.predict_proba(X_test_scaled)[:, 1]


evaluate_model(y_test, ada_pred, ada_probs)
```

```
Precision = 0.9662921348314607
Recall = 1.0
F1-score = 0.9828571428571429
ROC AUC = 0.9659468438538206
G-Mean = 0.7559289460184544
```

```
cf_ada = confusion_matrix(y_test, ada_pred)
plot_confusion_matrix(cf_ada)
```



Threshold Moving

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


clf3 = LogisticRegression(random_state=101)
clf3.fit(X_train_scaled, y_train)
```

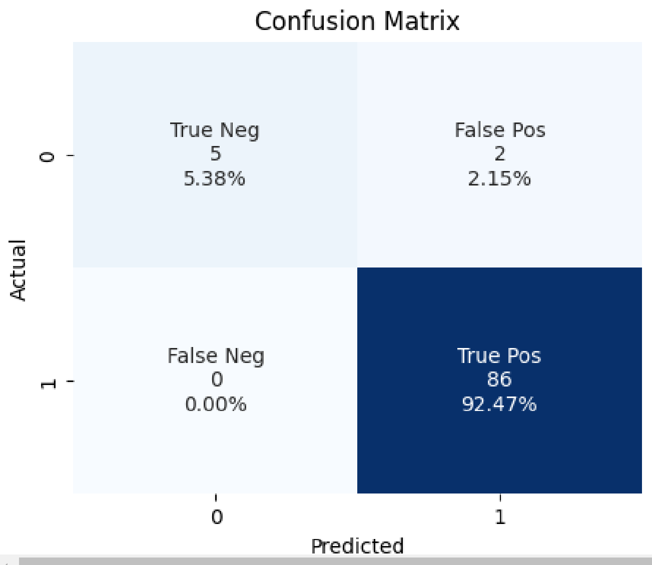
```
▾ LogisticRegression
LogisticRegression(random_state=101)
```

```python
def find_best_threshold(y_test, y_probs):
    thresholds = np.linspace(0, 1, 1000)
    best_f1 = -1
    best_threshold = 0.0

    for thresh in thresholds:
        y_pred = (y_probs >= thresh).astype(int)
        precision = precision_score(y_test, y_pred, zero_division=0)
        recall = recall_score(y_test, y_pred, zero_division=0)

        if (precision + recall) == 0:
            f1 = 0
        else:
            f1 = 2 * (precision * recall) / (precision + recall)

        if (f1 > best_f1):
            best_f1 = f1
            best_threshold = thresh

    return best_threshold, best_f1


y_probs = clf3.predict_proba(X_test_scaled)[:,1]


best_thrsh = find_best_threshold(y_test, y_probs)


y_pred_final = (y_probs >= best_thrsh[0]).astype(int)
```

```
evaluate_model(y_test, y_pred_final, y_probs)
```

```
Precision = 0.9772727272727273
Recall = 1.0
F1-score = 0.9885057471264368
ROC AUC = 0.9717607973421927
G-Mean = 0.8451542547285166
```

```
cf_matrix = confusion_matrix(y_test, y_pred_final)
plot_confusion_matrix(cf_matrix)
```



```
find_best_threshold(y_test, y_probs)
```

```
(0.3153153153153153, 0.9885057471264368)
```

Combining Sampling and Threshold Moving

```
smote_enn = SMOTEENN(random_state=42)
X_senn, y_senn = smote_enn.fit_resample(X, y)
```

```
X_train, X_test, y_train, y_test = train_test_split(X_senn, y_senn, test_size=0.3, random_state=42)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
clf4 = LogisticRegression(random_state=101)
clf4.fit(X_train_scaled, y_train)
```

```
▾ LogisticRegression
LogisticRegression(random_state=101)
```

```
y_probs1 = clf4.predict_proba(X_test_scaled)[:,1]
```

```
best_thresh = find_best_threshold(y_test, y_probs1)
```

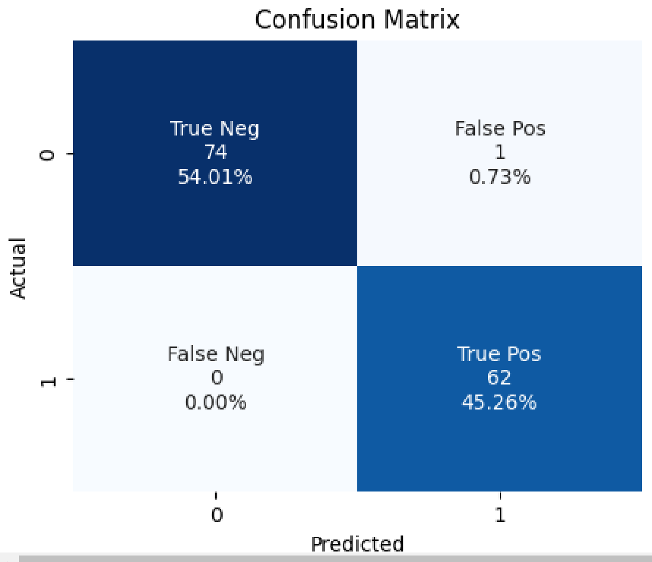
```
y_pred = (y_probs1 >= best_thresh[0]).astype(int)
```

```
evaluate_model(y_test, y_pred, y_probs1)
```

```
Precision = 0.9841269841269841
Recall = 1.0
F1-score = 0.992
ROC AUC = 0.9997849462365592
```

```
G-Mean = 0.993310961716756
```

```
cf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cf_matrix)
```



BaseLine Model

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

```
clf5 = LogisticRegression(random_state=101)
clf5.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
▾      LogisticRegression        ⓘ ?
LogisticRegression(random_state=101)
```

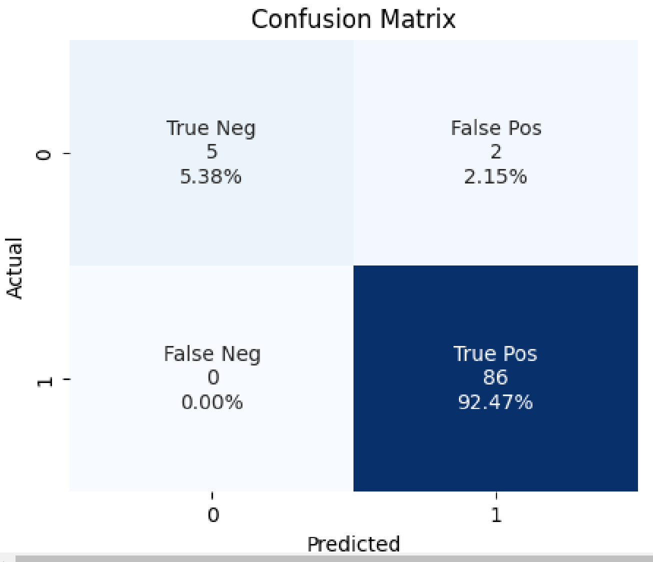
```
y_pred = clf5.predict(X_test)
y_probs = clf5.predict_proba(X_test_scaled)[:, 1]
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/utils/validation.py:2739: UserWarning: X does not have valid feature names, but Logistic
  warnings.warn(
```

```
evaluate_model(y_test, y_pred, y_probs)
```

```
Precision = 0.9772727272727273
Recall = 1.0
F1-score = 0.9885057471264368
ROC AUC = 0.9767441860465116
G-Mean = 0.8451542547285166
```

```
from sklearn.metrics import confusion_matrix
cf_matrix = confusion_matrix(y_test, y_pred)
plot_confusion_matrix(cf_matrix)
```



```
results = {
    "Method": ["Baseline", "Oversampling", "Undersampling", "Threshold Moving", "AdaBoost", "Combination"],
    "G": [ 0.845, 0.956, 0.892, 0.845, 0.965, 0.993]
}

df = pd.DataFrame(results)

plt.figure(figsize=(7, 5))

sns.lineplot(x="Method", y="G", data=df, marker="o", color="b", linewidth=2, markersize=8)

plt.xticks(rotation=30, ha='right', fontsize=10)
plt.title("G-mean Comparison")
plt.xlabel("Method")
plt.ylabel("G-mean")
plt.ylim(0.82, 1.03)

plt.grid(True ,linestyle='--')
plt.tight_layout()
plt.show()
```