**AD3002-Health Care Analytics**

**Assignment-2**

**GitHub Link:** https://github.com/mukeshsundar23/Health-care-analytics

**Aim:**

To Take one medical data set and analyse for below anyone,

• Clinical Prediction Models- Apply any one DL algorithm.

• Visual Analytics for Healthcare.

• NLP based clinical decision making- Apply anyone DL algorithm.

**Code:**

**Importing Libraries:**

import pandas as pd

import numpy as np

from catboost import CatBoostClassifier

from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, GradientBoostingClassifier

from sklearn.tree import DecisionTreeClassifier

from lightgbm import LGBMClassifier

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, f1\_score, recall\_score, precision\_score

import seaborn as sns

import matplotlib.pyplot as plt

import os

pd.set\_option('display.max\_columns', None)

pd.set\_option('display.max\_rows', None)

from google.colab import drive

drive.mount('/content/drive')



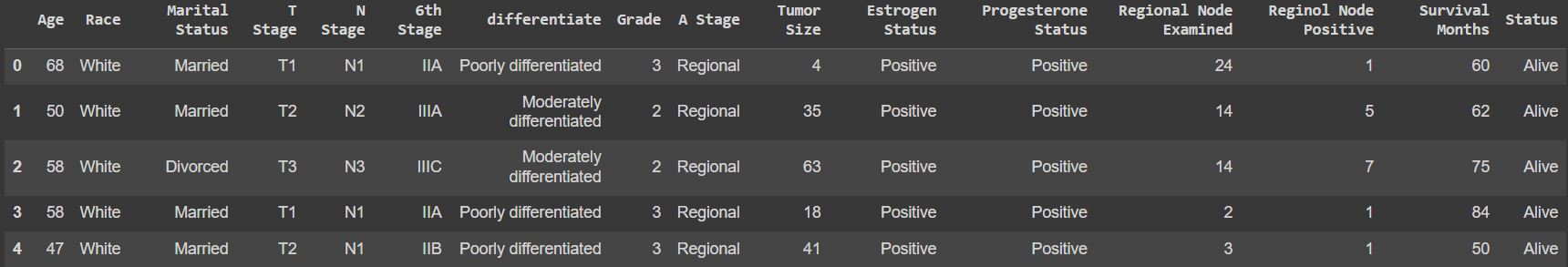
**Preparing Dataframe:**

filepath = "/content/drive/MyDrive/Semester 5/Healthcare Analytics/Breast\_Cancer.csv"

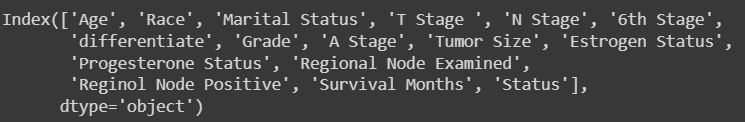
df = pd.read\_csv(filepath)

**General Inspection:**

df.head()



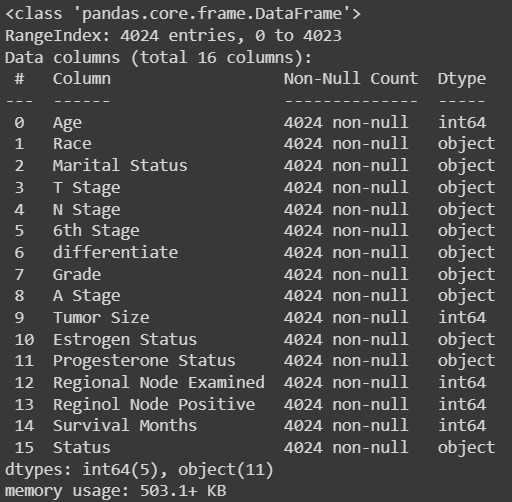
df.columns



df.shape



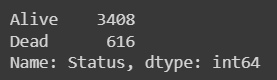
df.info()



df.isna().any().any()



df['Status'].value\_counts()



**Visualizing Data:**

plt.figure(figsize=(10, 6))

sns.histplot(data=df, x='Age', hue='Status', kde=True, palette='Set2', element='step', common\_norm=False)

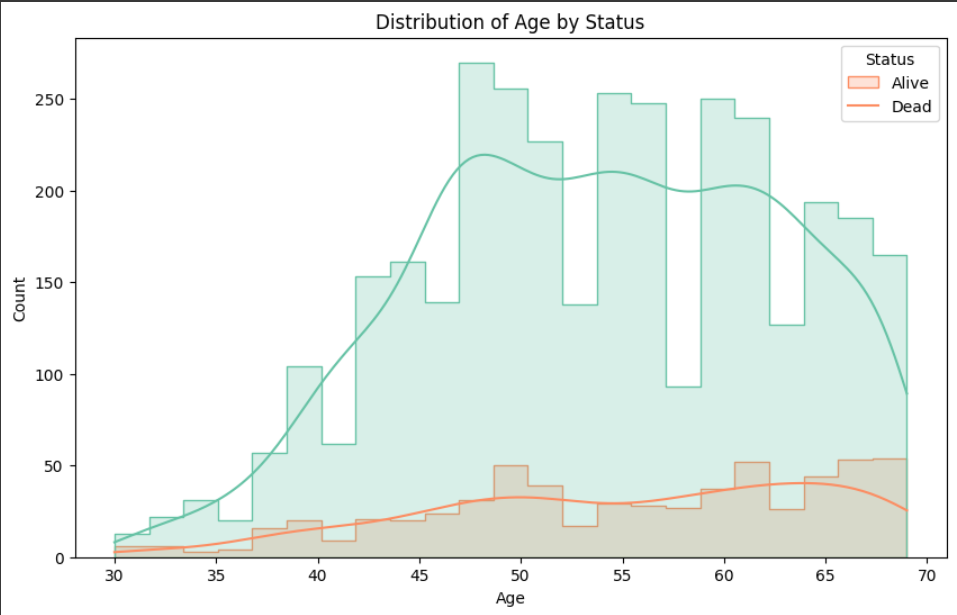
plt.title('Distribution of Age by Status')

plt.xlabel('Age')

plt.ylabel('Count')

plt.legend(title='Status', labels=['Alive', 'Dead'])

plt.show()



plt.figure(figsize=(10, 6))

sns.countplot(data=df, x='Race', hue='Status', palette='coolwarm')

plt.title('Distribution of Race by Status')

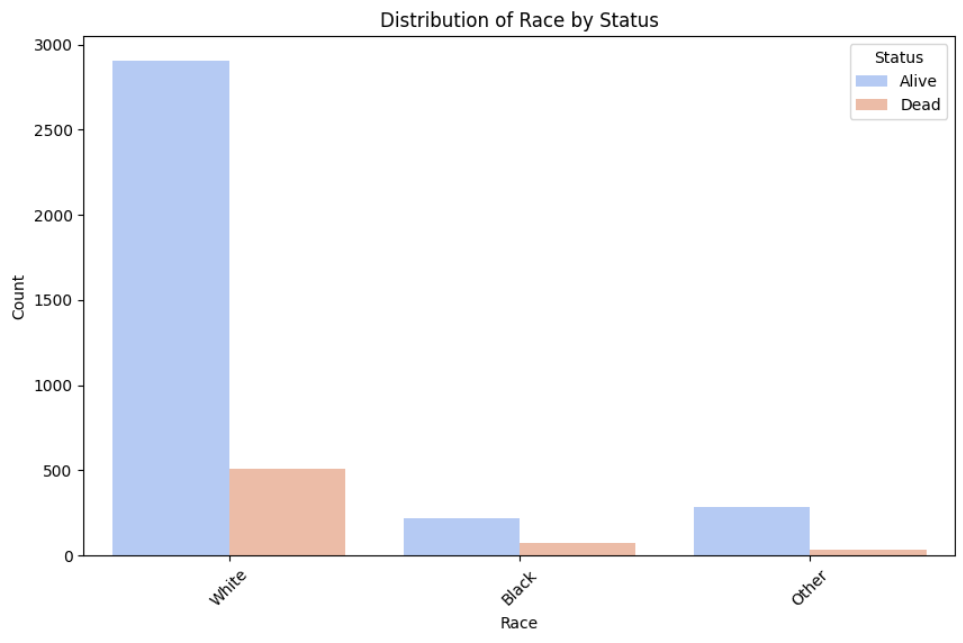
plt.xlabel('Race')

plt.ylabel('Count')

plt.legend(title='Status', labels=['Alive', 'Dead'])

plt.xticks(rotation=45)

plt.show()



alive\_df = df[df['Status'] == 'Alive']

dead\_df = df[df['Status'] == 'Dead'

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

sns.countplot(data=alive\_df, x='Marital Status', palette='Set1')

plt.title('Marital Status for Alive Patients')

plt.xlabel('Marital Status')

plt.ylabel('Count')

plt.xticks(rotation=45)

plt.subplot(1, 2, 2)

sns.countplot(data=dead\_df, x='Marital Status', palette='Set2')

plt.title('Marital Status for Dead Patients')

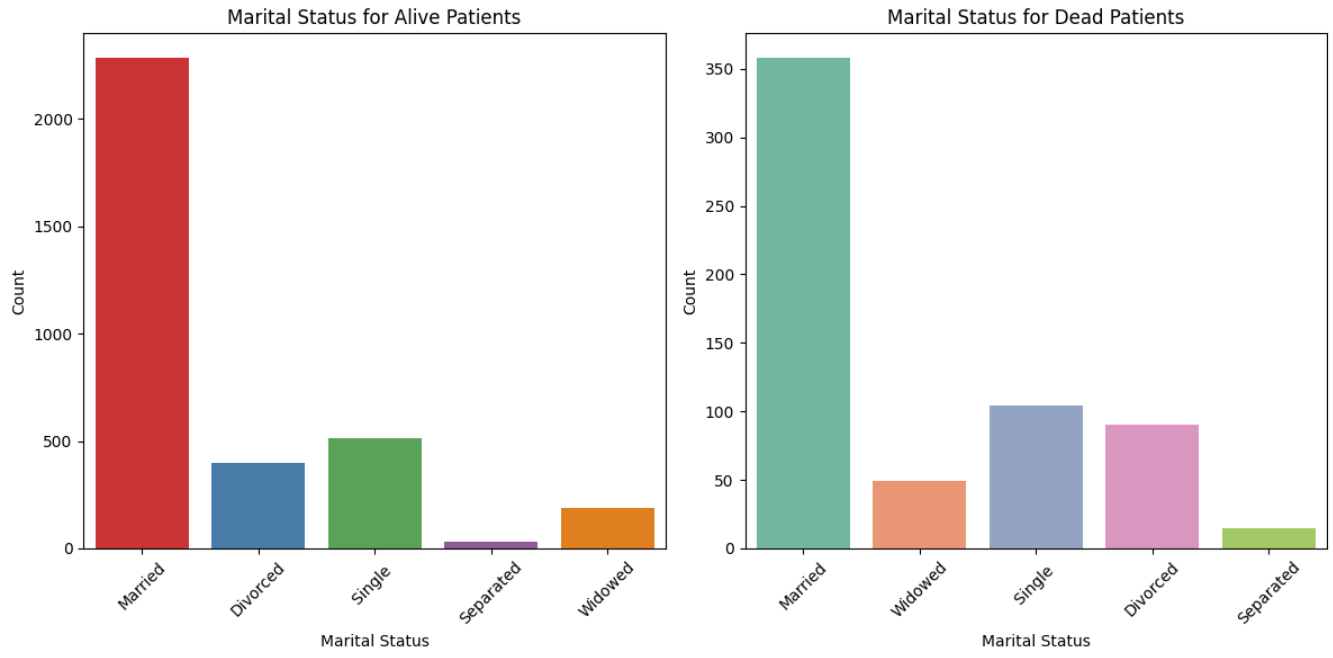
plt.xlabel('Marital Status')

plt.ylabel('Count')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()



plt.figure(figsize=(10, 6))

sns.boxplot(data=df, x='Status', y='Tumor Size', palette='hls')

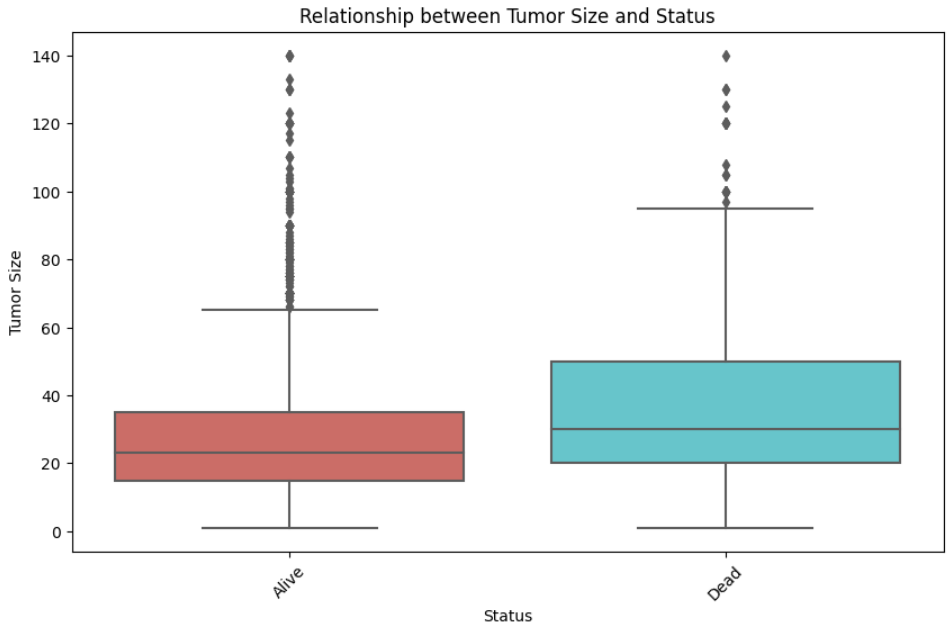
plt.title('Relationship between Tumor Size and Status')

plt.xlabel('Status')

plt.ylabel('Tumor Size')

plt.xticks(rotation=45)

plt.show()



plt.figure(figsize=(10, 6))

sns.boxplot(data=df, x='Status', y='Survival Months', palette='hls')

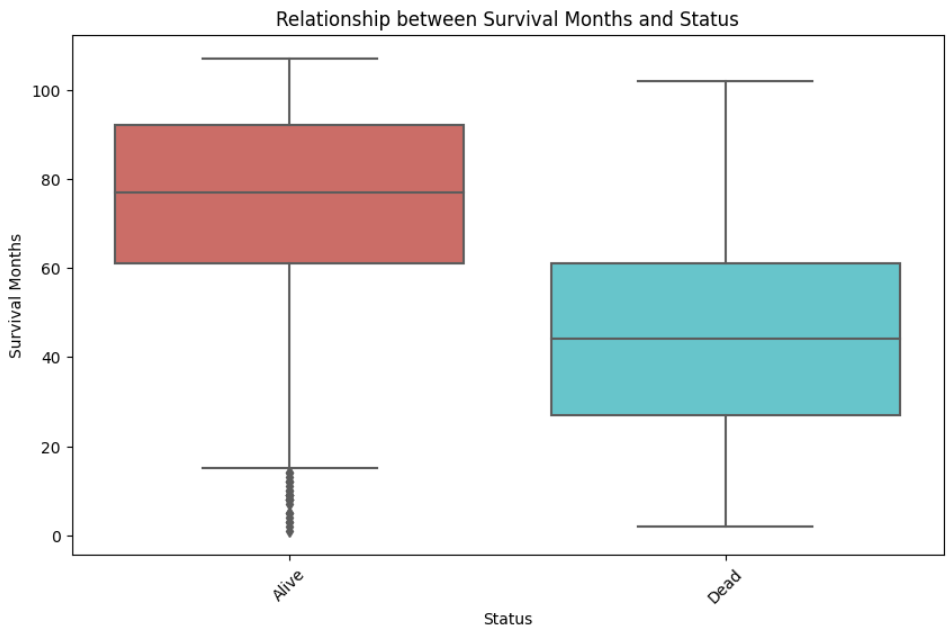
plt.title('Relationship between Survival Months and Status')

plt.xlabel('Status')

plt.ylabel('Survival Months')

plt.xticks(rotation=45)

plt.show()



**Pre-processing steps:**

df.rename(columns={"T Stage ": "T Stage"}, inplace=True)

df["Grade"].replace({" anaplastic; Grade IV": "4"}, inplace=True)

df["Grade"] = df["Grade"].astype(int)

df["T Stage"].replace({"T1":1, "T2": 2, "T3":3, "T4": 4}, inplace=True)

df["N Stage"].replace({"N1":1, "N2": 2, "N3":3}, inplace=True)

df["6th Stage"].replace({"IIA":1, "IIB": 2, "IIIA":3, "IIIB": 4,"IIIC":5}, inplace=True)

df["differentiate"].replace({"Moderately differentiated": 2,

"Poorly differentiated": 1,

"Well differentiated": 3,

"Undifferentiated": 0}, inplace=True)

df["A Stage"].replace({"Regional":1, "Distant": 0}, inplace=True)

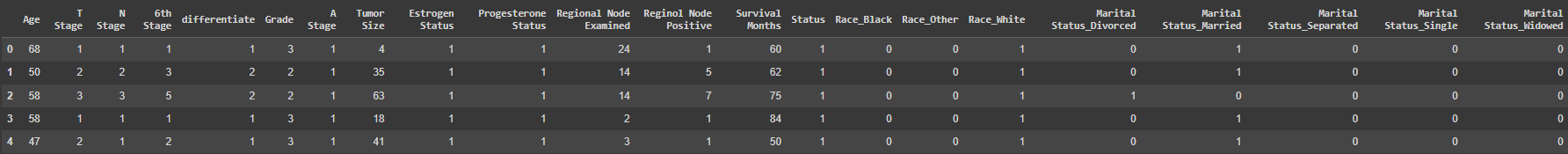
df["Estrogen Status"].replace({"Positive":1, "Negative": 0}, inplace=True)

df["Progesterone Status"].replace({"Positive":1, "Negative": 0}, inplace=True)

df["Status"].replace({"Alive":1, "Dead": 0}, inplace=True)

columns\_to\_encode = ["Race", "Marital Status"]

df = pd.get\_dummies(df, columns=columns\_to\_encode, dtype=int)



**Scaling Features:**

def scale\_features(X):

scaler = StandardScaler()

cols = X.columns

X\_scaled = scaler.fit\_transform(X)

return pd.DataFrame(X\_scaled, columns=cols)

**Split Train and Test Data:**

X = df.drop(["Status"], axis=1)

X = scale\_features(X)

y = df["Status"]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

catboost\_hyperparameters = {

'nan\_mode': 'Min',

'eval\_metric': 'Logloss',

'iterations': 1000,

'sampling\_frequency': 'PerTree',

'leaf\_estimation\_method': 'Newton',

'random\_score\_type': 'NormalWithModelSizeDecrease',

'grow\_policy': 'SymmetricTree',

'penalties\_coefficient': 1,

'boosting\_type': 'Plain',

'model\_shrink\_mode': 'Constant',

'feature\_border\_type': 'GreedyLogSum',

'eval\_fraction': 0,

'l2\_leaf\_reg': 3,

'random\_strength': 1,

'rsm': 1,

'boost\_from\_average': False,

'model\_size\_reg': 0.5,

'subsample': 0.800000011920929,

'use\_best\_model': False,

'class\_names': [0, 1],

'random\_seed': 2584,

'depth': 6,

'posterior\_sampling': False,

'border\_count': 254,

'classes\_count': 0,

'auto\_class\_weights': 'None',

'sparse\_features\_conflict\_fraction': 0,

'leaf\_estimation\_backtracking': 'AnyImprovement',

'best\_model\_min\_trees': 1,

'model\_shrink\_rate': 0,

'min\_data\_in\_leaf': 1,

'loss\_function': 'Logloss',

'learning\_rate': 0.015429000370204449,

'score\_function': 'Cosine',

'task\_type': 'CPU',

'leaf\_estimation\_iterations': 10,

'bootstrap\_type': 'MVS',

'max\_leaves': 64,

'verbose': False

}

random\_forest\_hyperparameters = {

'bootstrap': True,

'ccp\_alpha': 0.0,

'class\_weight': None,

'criterion': 'gini',

'max\_depth': None,

'max\_features': 'sqrt',

'max\_leaf\_nodes': None,

'max\_samples': None,

'min\_impurity\_decrease': 0.0,

'min\_samples\_leaf': 1,

'min\_samples\_split': 2,

'min\_weight\_fraction\_leaf': 0.0,

'n\_estimators': 100,

'n\_jobs': -1,

'oob\_score': False,

'random\_state': 2584,

'verbose': 0,

'warm\_start': False

}

adaboost\_hyperparameters = {

'algorithm': 'SAMME.R',

'base\_estimator': DecisionTreeClassifier(),

'learning\_rate': 0.2,

'n\_estimators': 190,

'random\_state': 2584

}

gradient\_boosting\_hyperparameters = {

'ccp\_alpha': 0.0,

'criterion': 'friedman\_mse',

'init': None,

'learning\_rate': 0.1,

'loss': 'log\_loss',

'max\_depth': 3,

'max\_features': None,

'max\_leaf\_nodes': None,

'min\_impurity\_decrease': 0.0,

'min\_samples\_leaf': 1,

'min\_samples\_split': 2,

'min\_weight\_fraction\_leaf': 0.0,

'n\_estimators': 100,

'n\_iter\_no\_change': None,

'random\_state': 2584,

'subsample': 1.0,

'tol': 0.0001,

'validation\_fraction': 0.1,

'verbose': 0,

'warm\_start': False

}

lgbm\_hyperparameters = {

'bagging\_fraction': 0.8,

'bagging\_freq': 0,

'boosting\_type': 'gbdt',

'class\_weight': None,

'colsample\_bytree': 1.0,

'feature\_fraction': 0.6,

'importance\_type': 'split',

'learning\_rate': 0.15,

'max\_depth': -1,

'min\_child\_samples': 26,

'min\_child\_weight': 0.001,

'min\_split\_gain': 0.4,

'n\_estimators': 40,

'n\_jobs': -1,

'num\_leaves': 60,

'objective': None,

'random\_state': 2584,

'reg\_alpha': 5,

'reg\_lambda': 1,

'silent': 'warn',

'subsample': 1.0,

'subsample\_for\_bin': 200000,

'subsample\_freq': 0

}

catboost\_classifier = CatBoostClassifier(\*\*catboost\_hyperparameters)

catboost\_classifier.fit(X\_train, y\_train)

random\_forest\_classifier = RandomForestClassifier(\*\*random\_forest\_hyperparameters)

random\_forest\_classifier.fit(X\_train, y\_train)

adaboost\_classifier = AdaBoostClassifier(\*\*adaboost\_hyperparameters)

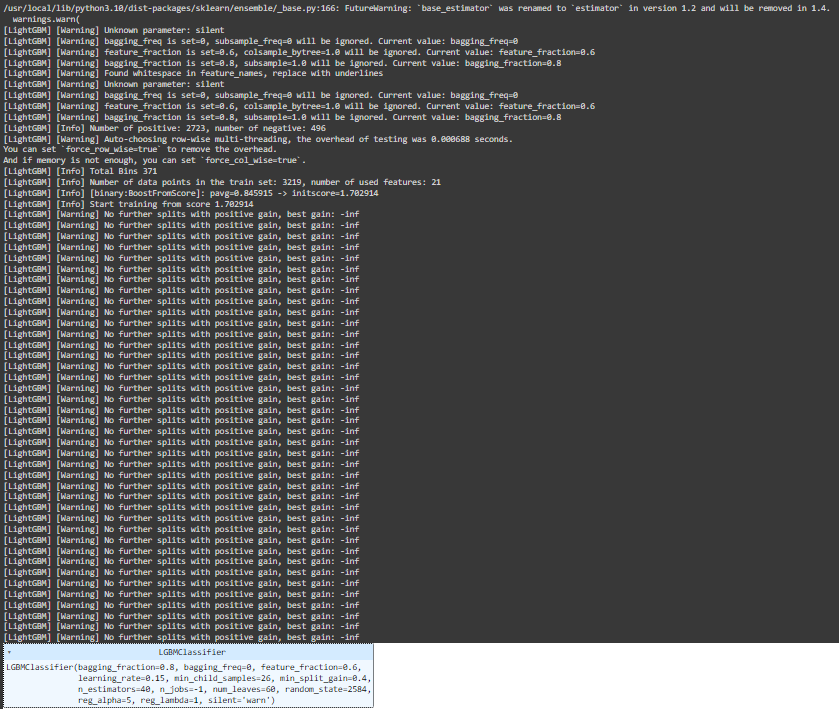
adaboost\_classifier.fit(X\_train, y\_train)

gradient\_boosting\_classifier = GradientBoostingClassifier(\*\*gradient\_boosting\_hyperparameters)

gradient\_boosting\_classifier.fit(X\_train, y\_train)

lgbm\_classifier = LGBMClassifier(\*\*lgbm\_hyperparameters)

lgbm\_classifier.fit(X\_train, y\_train)



y\_pred\_random\_forest = random\_forest\_classifier.predict(X\_test)

y\_pred\_adaboost = adaboost\_classifier.predict(X\_test)

y\_pred\_gradient\_boosting = gradient\_boosting\_classifier.predict(X\_test)

y\_pred\_lgbm = lgbm\_classifier.predict(X\_test)

y\_pred\_catboost = catboost\_classifier.predict(X\_test)

accuracy\_rf = accuracy\_score(y\_test, y\_pred\_random\_forest)

accuracy\_adaboost = accuracy\_score(y\_test, y\_pred\_adaboost)

accuracy\_gradient\_boosting = accuracy\_score(y\_test, y\_pred\_gradient\_boosting)

accuracy\_lgbm = accuracy\_score(y\_test, y\_pred\_lgbm)

accuracy\_catboost = accuracy\_score(y\_test, y\_pred\_catboost)

f1\_rf = f1\_score(y\_test, y\_pred\_random\_forest)

f1\_adaboost = f1\_score(y\_test, y\_pred\_adaboost)

f1\_gradient\_boosting = f1\_score(y\_test, y\_pred\_gradient\_boosting)

f1\_lgbm = f1\_score(y\_test, y\_pred\_lgbm)

f1\_catboost = f1\_score(y\_test, y\_pred\_catboost)

recall\_rf = recall\_score(y\_test, y\_pred\_random\_forest)

recall\_adaboost = recall\_score(y\_test, y\_pred\_adaboost)

recall\_gradient\_boosting = recall\_score(y\_test, y\_pred\_gradient\_boosting)

recall\_lgbm = recall\_score(y\_test, y\_pred\_lgbm)

recall\_catboost = recall\_score(y\_test, y\_pred\_catboost)

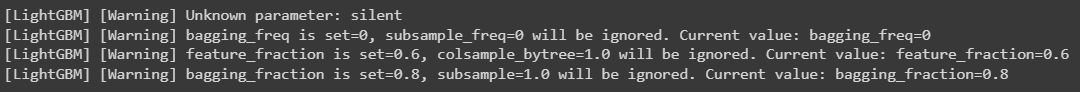
precision\_rf = precision\_score(y\_test, y\_pred\_random\_forest)

precision\_adaboost = precision\_score(y\_test, y\_pred\_adaboost)

precision\_gradient\_boosting = precision\_score(y\_test, y\_pred\_gradient\_boosting)

precision\_lgbm = precision\_score(y\_test, y\_pred\_lgbm)

precision\_catboost = precision\_score(y\_test, y\_pred\_catboost)



models = ['Random Forest', 'AdaBoost', 'Gradient Boosting', 'LGBM', 'CatBoost']

metrics = {

'Model': models,

'Accuracy': [accuracy\_score(y\_test, y\_pred) for y\_pred in [y\_pred\_random\_forest, y\_pred\_adaboost, y\_pred\_gradient\_boosting, y\_pred\_lgbm, y\_pred\_catboost]],

'F1 Score': [f1\_score(y\_test, y\_pred) for y\_pred in [y\_pred\_random\_forest, y\_pred\_adaboost, y\_pred\_gradient\_boosting, y\_pred\_lgbm, y\_pred\_catboost]],

'Recall': [recall\_score(y\_test, y\_pred) for y\_pred in [y\_pred\_random\_forest, y\_pred\_adaboost, y\_pred\_gradient\_boosting, y\_pred\_lgbm, y\_pred\_catboost]],

'Precision': [precision\_score(y\_test, y\_pred) for y\_pred in [y\_pred\_random\_forest, y\_pred\_adaboost, y\_pred\_gradient\_boosting, y\_pred\_lgbm, y\_pred\_catboost]]

}

df\_metrics = pd.DataFrame(metrics)

df\_metrics

