Lung Cancer Prediction:

Abstract:

This paper discusses the prediction of lung cancer survival based on a large dataset containing a variety of patient medical records. The goal of this analysis is to determine the significant contributors to the survival and come up with a strong model to predict whether a given patient survived or not. We tested a few different machine learning models from ones such as XGBoost, K-Nearest Neighbors (KNN), and Random Forest to Decision Trees, Gradient Boosting, and Logistic Regression. Preprocessing consisted of managing categorical variables, scaling numerical data and feature engineering. The models were compared on the basis of accuracy, and the best models were subjected to additional tuning of hyperparameters in order to maximize predictive power. This article provides a complete description of the methodology, results and future improvements on this important medical prediction task.

Introduction:

Lung cancer remains one of the most frequent causes of cancer-death worldwide. In terms of clinical decisions and patient care it is a necessity to predict patient survival accurately and timely. The aim of this study is to utilise ML techniques to develop a prognostic model to estimate the prognosis for lung cancer from a plethora of patient-specific features. Of medical history, lifestyle, and therapy with the above studies results, the model would be able to assist health care workers for retrospective purpose. The paper details the methodology used which includes data collection and pre-processing, model construction, training, and testing, and highlights some of the challenges encountered, and opportunities for further study in this important domain.

Dataset Description:

The dataset, dataset_med. csv, includes a wide range of medical information for lung cancer patients. It is the combination of 890,000 rows and 17 columns which includes id, age, gender, country, diagnosis_date, cancer_stage, family_history, smoking_status, bmi, cholesterol_level, hypertension, asthma, cirrhosis, other_cancer, treatment_type, end_treatment_date and survival (the target variable survived). Survived is a binary outcome variable (0 for no death, 1 for death). Both the outcome and analysis population are represented as consolidated mixed datatypes (numeric, categorical, boolean) providing a unified picture of patient health and treatment information (typically for predictive model development).

Technologies and Libraries Used:

It is heavily based on Python for data wrangling, analysis, and machine learning model construction. Key libraries employed include:

1. Pandas: "Fast, powerful, flexible and easy to use open-source data analysis and data manipulation library built on top of Python. Optimized data's frame in Python".
2. NumPy: n-dimension array computing.
3. Scikit-learn (sklearn): A library for Machine Learning tools for data preprocessing: MinMaxScaler, LabelEncoder and StandardScaler, model selection: train_test_split, GridSearchCV and RandomizedSearchCV and implementations of algorithms like: KNeighborsClassifier, RandomForestClassifier, DecisionTreeClassifier, GradientBoostingClassifier and LogisticRegression.
4. XGBoost: Super-fast implementation of the gradient boosting algorithm, famous for being very fast and very accurate in practice.

Data Preprocessing:

Preprocessing of data was a necessary step to bring the raw dataset into the form ready for training the models. The following steps were performed:

- Feature Dropping: The id column was dropped since it does not help in prediction.
- Data Type Casting: Booleans (hypertension, asthma, cirrhosis, other_cancer) were specifically cast into booleans.
- Normalization of Numeric Values: The values of bmi and cholesterol_level were normalized by MinMaxScaler to bring them down to the range between 0 and 1 to scale down the values.
- Outlier Removal Outliers of bmi and cholesterol_level were dealt using Inter-Quartile Range (IQR) approach for the better robustness for the model.
- Date Feature Engineering: diagnosis_date, end_treatment_date were converted to datetime and a new feature - duration of treatment (in days) - was engineered. The columns of original date were dropped then.
- Encoding Categorical: gender and family_history were assigned numerical values (0, 1). The other categorical features (country,

cancer_stage, smoking_status, treatment_type) were one-hot encoded by pd. dummies to be able to use it in machine-learning models.

Model Architecture:

The project investigated a range of supervised machine learning classification techniques to predict lung cancer survival:

1. XGBoost Classifier: A class of ensemble models using gradient boosting which is popular for its high speed and performance.
2. K-Nearest Neighbors (KNN): A type of instance-based learning, or lazy learning where the function is only approximated locally and all computation is deferred until function evaluation, that determines the classification of k nearest neighbors.
3. Random Forest Classifier: Yet another ensemble technique which constructs multiple decision trees and combines their predictions to obtain better coverage and control over-fitting.
4. Tree Classifier: A model of decisions made by a tree and the resulting consequences, including random event outcomes, cost of resources and utility.
5. Gradient Boosting Classifier: Gradient Boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, which builds the model in a stage-wise fashion and it generalizes them by allowing optimization of an arbitrary differentiable loss function.
6. Logistic Regression: A simple linear model for binary classification (modelled the probability of an example belonging to a class).

Training Configuration:

- The data was split into training and testing sets for each model. The train split was 70/30 for KNN, Random Forest, Decision Trees and Logistic Regression and 80/20 split, stratified, for XGBoost and Gradient Boosting models in order to preserve the balance of classes. Numerical features were scaled with StandardScaler for XGBoost and Gradient Boosting to get better performance.
- Thereafter, for the top three best models (XGBoost, Gradient Boosting, Logistic Regression), hyperparameter optimization was tried using GridSearchCV and RandomizedSearchCV. These techniques search iteratively for the best set of hyperparameters to improve model accuracy

thus, improving the predictive capacity and generalization of the selected models.

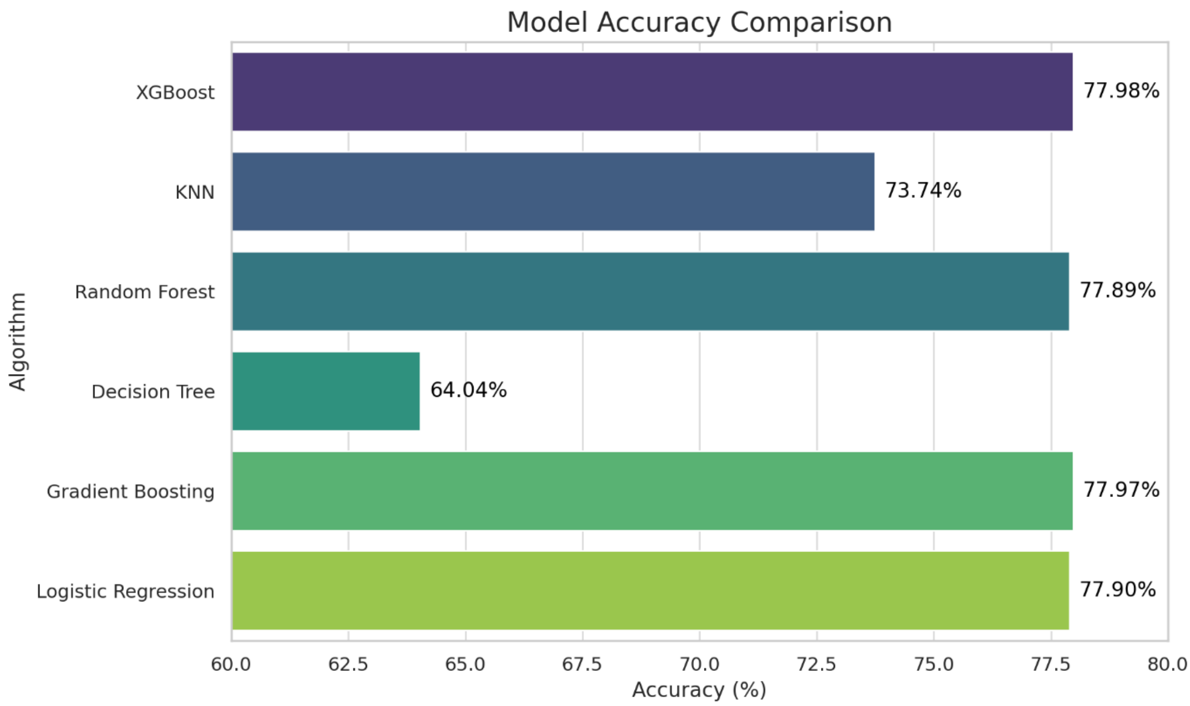
Results and Evaluation:

Initial model evaluations provided baseline accuracies for each algorithm. The top three performers were XGBoost, Gradient Boosting, and Logistic Regression:

Algorithm	Accuracy
XGBoost	77.98%
KNN	73.74%
Random Forest	77.89%
Decision Tree	64.04%
Gradient Boosting	77.97%
Logistic Regression	77.9%

The hyperparameter optimization for Logistic Regression, Gradient Boosting and XGBoost was also not fully performed as intended in the given notebook. As a result, the ultimate optimized accuracy scores for these models are not known. Yet with these baseline results XGBoost and Gradient boosting also presented an interesting performance which suggests accuracy can be further enhanced with correct parameterisation.

Results Graph:



Challenges Faced:

The main shortcoming we encountered is the computational time required for hyperparameters optimization, especially for complex models such as XGBoost and Gradient Boosting. This is in the face of a wealth of data and known good processing, but: it was still a relatively high-dimensional grid of the best hyperparameters that would need to be scanned through GridSearchCV and RandomizedSearchCV. This restriction was obstructing the complete execution of the iterative step of hyperparameter tuning, and hence had a shielding effect on the model's accuracy and level of model optimization. This underscores the call for more powerful processing systems for large ML workloads.

Future Scope:

This project has many potential future applications. 1) Initiate the hyperparameter tuning for XGBoost, Gradient Boosting and Logistic Regression either on-cloud or on even faster machines on your side to maximize their predictive power. Secondly one can try more sophisticated deep learning architectures such as Convolutional neuron networks or Recurrent neural networks, if the data is sequential, leading to higher accuracies. Moreover, enrichment of the model in different patient datasets (for example, where genetic information, detailed treatment responses or long-term health records are available) would improve its robustness and make it more generalizable. Lastly, if the model is implemented as a user-friendly web-based application it can be used for real-time predictions for healthcare workers.

Conclusions:

The current work has effectively built up a preliminary machine learning model pipeline for lung cancer survival prediction. After extensive data preprocessing (e.g. normalization, outlier treatment, feature engineering...), the dataset was trained with model-robust. At first glance evaluations revealed that XGBoost, Gradient Boosting and Logistic Regression looks appropriate with accuracy scores close to 78%. Limited resources precluded a complete hyperparameter search, but the estimated baseline results indicate that machine learning approaches might be feasible in this important medical area. The project is a baseline and intended to open the door to many future improvements to the predictions and data, given more computational investment and input data available.

```
import pandas as pd
import numpy as np

df = pd.read_csv(r"E:\Projects\lung_cancer\Lung Cancer\
dataset_med.csv")

df.head()

   id   age  gender      country diagnosis_date cancer_stage  
family_history  \
0   1  64.0    Male       Sweden     2016-04-05      Stage I            
Yes   
1   2  50.0  Female  Netherlands     2023-04-20    Stage III            
Yes   
2   3  65.0  Female      Hungary     2023-04-05    Stage III            
Yes   
3   4  51.0  Female      Belgium     2016-02-05      Stage I            
No   
4   5  37.0    Male   Luxembourg     2023-11-29      Stage I            
No   

   smoking_status   bmi  cholesterol_level  hypertension  asthma  
cirrhosis  \
0  Passive Smoker  29.4                199             0       0          
1   
1  Passive Smoker  41.2                280             1       1          
0   
2   Former Smoker  44.0                268             1       1          
0   
3  Passive Smoker  43.0                241             1       1          
0   
4  Passive Smoker  19.7                178             0       0          
0   

   other_cancer treatment_type end_treatment_date  survived  
0             0   Chemotherapy         2017-09-10         0  
1             0        Surgery         2024-06-17         1  
2             0       Combined         2024-04-09         0  
3             0   Chemotherapy         2017-04-23         0  
4             0       Combined         2025-01-08         0  

df['hypertension'] = df['hypertension'].astype(bool)

df['asthma'] = df['asthma'].astype(bool)

df['cirrhosis'] = df['cirrhosis'].astype(bool)

df['other_cancer'] = df['other_cancer'].astype(bool)

df = df.drop('id', axis=1)
```

```
df.head()

    age  gender      country diagnosis_date cancer_stage
family_history  \
0  64.0    Male       Sweden     2016-04-05      Stage I
Yes
1  50.0  Female  Netherlands     2023-04-20    Stage III
Yes
2  65.0  Female      Hungary     2023-04-05    Stage III
Yes
3  51.0  Female      Belgium     2016-02-05      Stage I
No
4  37.0    Male   Luxembourg     2023-11-29      Stage I
No

   smoking_status   bmi  cholesterol_level  hypertension  asthma
cirrhosis  \
0  Passive Smoker  29.4                199         False   False
True
1  Passive Smoker  41.2                280          True    True
False
2   Former Smoker  44.0                268          True    True
False
3  Passive Smoker  43.0                241          True    True
False
4  Passive Smoker  19.7                178         False   False
False

   other_cancer treatment_type end_treatment_date  survived
0         False   Chemotherapy         2017-09-10         0
1         False        Surgery         2024-06-17         1
2         False       Combined         2024-04-09         0
3         False   Chemotherapy         2017-04-23         0
4         False       Combined         2025-01-08         0

df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 890000 entries, 0 to 889999
Data columns (total 16 columns):
 #   Column              Non-Null Count   Dtype
---  ------              --------------   -----
 0   age                 890000 non-null  float64
 1   gender              890000 non-null  object
 2   country             890000 non-null  object
 3   diagnosis_date      890000 non-null  object
 4   cancer_stage        890000 non-null  object
 5   family_history      890000 non-null  object
 6   smoking_status      890000 non-null  object
 7   bmi                 890000 non-null  float64
```

```
 8   cholesterol_level   890000 non-null  int64
 9   hypertension        890000 non-null  bool
 10  asthma              890000 non-null  bool
 11  cirrhosis           890000 non-null  bool
 12  other_cancer        890000 non-null  bool
 13  treatment_type      890000 non-null  object
 14  end_treatment_date  890000 non-null  object
 15  survived            890000 non-null  int64
dtypes: bool(4), float64(2), int64(2), object(8)
memory usage: 84.9+ MB
```

Now , lets proceed with data preprocessing

```
#Normalization

from sklearn.preprocessing import MinMaxScaler

numerical_cols = ['bmi', 'cholesterol_level']

scaler = MinMaxScaler()

df[numerical_cols] = scaler.fit_transform(df[numerical_cols])

df.head()

    age  gender      country diagnosis_date cancer_stage
family_history  \
0  64.0    Male       Sweden     2016-04-05      Stage I
Yes
1  50.0  Female  Netherlands     2023-04-20    Stage III
Yes
2  65.0  Female      Hungary     2023-04-05    Stage III
Yes
3  51.0  Female      Belgium     2016-02-05      Stage I
No
4  37.0    Male   Luxembourg     2023-11-29      Stage I
No

   smoking_status       bmi  cholesterol_level  hypertension
asthma  \
0  Passive Smoker  0.462069           0.326667         False   False

1  Passive Smoker  0.868966           0.866667          True    True

2   Former Smoker  0.965517           0.786667          True    True

3  Passive Smoker  0.931034           0.606667          True    True

4  Passive Smoker  0.127586           0.186667         False   False
```

```
   cirrhosis  other_cancer treatment_type end_treatment_date  survived

0       True         False   Chemotherapy         2017-09-10         0

1      False         False        Surgery         2024-06-17         1

2      False         False       Combined         2024-04-09         0

3      False         False   Chemotherapy         2017-04-23         0

4      False         False       Combined         2025-01-08         0
```

```
df.shape
```

```
(890000, 16)
```

```
#Outlier removal

numerical_cols_for_outlier_removal = ['bmi', 'cholesterol_level']

for col in numerical_cols_for_outlier_removal:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df = df[(df[col] >= lower_bound) & (df[col] <= upper_bound)]

df.shape
```

```
(890000, 16)
```

```
gender_mapping = {'Female': 0, 'Male': 1}
df['gender'] = df['gender'].map(gender_mapping)

df.head()
```

```
    age  gender      country diagnosis_date cancer_stage
family_history  \
0  64.0       1       Sweden     2016-04-05      Stage I
Yes
1  50.0       0  Netherlands     2023-04-20    Stage III
Yes
2  65.0       0      Hungary     2023-04-05    Stage III
Yes
3  51.0       0      Belgium     2016-02-05      Stage I
No
4  37.0       1   Luxembourg     2023-11-29      Stage I
No
```

```
   smoking_status       bmi  cholesterol_level  hypertension
asthma  \
0  Passive Smoker  0.462069           0.326667         False   False

1  Passive Smoker  0.868966           0.866667          True    True

2   Former Smoker  0.965517           0.786667          True    True

3  Passive Smoker  0.931034           0.606667          True    True

4  Passive Smoker  0.127586           0.186667         False   False


   cirrhosis  other_cancer treatment_type end_treatment_date  survived

0       True         False   Chemotherapy         2017-09-10         0

1      False         False        Surgery         2024-06-17         1

2      False         False       Combined         2024-04-09         0

3      False         False   Chemotherapy         2017-04-23         0

4      False         False       Combined         2025-01-08         0


df['diagnosis_date'] = pd.to_datetime(df['diagnosis_date'])
df['end_treatment_date'] = pd.to_datetime(df['end_treatment_date'])

df['duration of treatment'] = (df['end_treatment_date'] -
df['diagnosis_date']).dt.days

df.head()

    age  gender      country diagnosis_date cancer_stage
family_history  \
0  64.0       1       Sweden     2016-04-05      Stage I
Yes
1  50.0       0  Netherlands     2023-04-20    Stage III
Yes
2  65.0       0      Hungary     2023-04-05    Stage III
Yes
3  51.0       0      Belgium     2016-02-05      Stage I
No
4  37.0       1   Luxembourg     2023-11-29      Stage I
No

   smoking_status       bmi  cholesterol_level  hypertension
asthma  \
0  Passive Smoker  0.462069           0.326667         False   False
```

```
1  Passive Smoker  0.868966           0.866667         True    True

2   Former Smoker  0.965517           0.786667         True    True

3  Passive Smoker  0.931034           0.606667         True    True

4  Passive Smoker  0.127586           0.186667        False   False


   cirrhosis  other_cancer treatment_type end_treatment_date  survived  \
0       True         False   Chemotherapy         2017-09-10         0

1      False         False        Surgery         2024-06-17         1

2      False         False       Combined         2024-04-09         0

3      False         False   Chemotherapy         2017-04-23         0

4      False         False       Combined         2025-01-08         0


   duration of treatment
0                    523
1                    424
2                    370
3                    443
4                    406
```

```
df = df.drop(columns=['diagnosis_date', 'end_treatment_date'])

df.head()
```

```
    age  gender      country cancer_stage family_history
smoking_status  \
0  64.0       1       Sweden      Stage I            Yes  Passive
Smoker
1  50.0       0  Netherlands    Stage III            Yes  Passive
Smoker
2  65.0       0      Hungary    Stage III            Yes   Former
Smoker
3  51.0       0      Belgium      Stage I             No  Passive
Smoker
4  37.0       1   Luxembourg      Stage I             No  Passive
Smoker

        bmi  cholesterol_level  hypertension  asthma  cirrhosis
other_cancer  \
0  0.462069           0.326667         False   False       True
False
1  0.868966           0.866667          True    True      False
```

```
False
2  0.965517           0.786667          True    True      False
False
3  0.931034           0.606667          True    True      False
False
4  0.127586           0.186667         False   False      False
False

  treatment_type  survived  duration of treatment
0   Chemotherapy         0                    523
1        Surgery         1                    424
2       Combined         0                    370
3   Chemotherapy         0                    443
4       Combined         0                    406

family_history_mapping = {'No': 0, 'Yes': 1}
df['family_history'] =
df['family_history'].map(family_history_mapping)

df.head()

    age  gender      country cancer_stage  family_history
smoking_status  \
0  64.0       1       Sweden      Stage I               1  Passive
Smoker
1  50.0       0  Netherlands    Stage III               1  Passive
Smoker
2  65.0       0      Hungary    Stage III               1   Former
Smoker
3  51.0       0      Belgium      Stage I               0  Passive
Smoker
4  37.0       1   Luxembourg      Stage I               0  Passive
Smoker

        bmi  cholesterol_level  hypertension  asthma  cirrhosis
other_cancer  \
0  0.462069           0.326667         False   False       True
False
1  0.868966           0.866667          True    True      False
False
2  0.965517           0.786667          True    True      False
False
3  0.931034           0.606667          True    True      False
False
4  0.127586           0.186667         False   False      False
False

  treatment_type  survived  duration of treatment
0   Chemotherapy         0                    523
1        Surgery         1                    424
```

```
2       Combined         0                     370
3   Chemotherapy         0                     443
4       Combined         0                     406
```

```
df['family_history'] = df['family_history'].astype(bool)

df.head()
```

```
    age  gender      country cancer_stage  family_history
smoking_status  \
0  64.0       1       Sweden      Stage I            True  Passive
Smoker
1  50.0       0  Netherlands    Stage III            True  Passive
Smoker
2  65.0       0      Hungary    Stage III            True   Former
Smoker
3  51.0       0      Belgium      Stage I           False  Passive
Smoker
4  37.0       1   Luxembourg      Stage I           False  Passive
Smoker

        bmi  cholesterol_level  hypertension  asthma  cirrhosis
other_cancer  \
0  0.462069           0.326667         False   False       True
False
1  0.868966           0.866667          True    True      False
False
2  0.965517           0.786667          True    True      False
False
3  0.931034           0.606667          True    True      False
False
4  0.127586           0.186667         False   False      False
False

  treatment_type  survived  duration of treatment
0   Chemotherapy         0                     523
1        Surgery         1                     424
2       Combined         0                     370
3   Chemotherapy         0                     443
4       Combined         0                     406
```

```
df.describe()
```

```
                 age         gender            bmi  cholesterol_level
\
count  890000.000000  890000.000000  890000.000000      890000.000000

mean       55.007008       0.500151       0.499799           0.557559

std         9.994485       0.500000       0.288570           0.289549
```

```
min        4.000000      0.000000      0.000000          0.000000
25%       48.000000      0.000000      0.251724          0.306667
50%       55.000000      1.000000      0.500000          0.613333
75%       62.000000      1.000000      0.748276          0.806667
max      104.000000      1.000000      1.000000          1.000000

            survived  duration of treatment
count  890000.000000          890000.000000
mean        0.220229             458.087170
std         0.414401             139.326048
min         0.000000             183.000000
25%         0.000000             367.000000
50%         0.000000             458.000000
75%         0.000000             550.000000
max         1.000000             730.000000
```

Null value removal not needed hence we can now proceed with the training now

```python
# XGBoost

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score

categorical_cols = df.select_dtypes(include='object').columns
for col in categorical_cols:
    df[col] = LabelEncoder().fit_transform(df[col])

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

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
test_size=0.2, random_state=42)

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

xgb = XGBClassifier(n_estimators=200, learning_rate=0.1, max_depth=6,
use_label_encoder=False, eval_metric='logloss', random_state=42)
xgb.fit(X_train_scaled, y_train)

y_pred = xgb.predict(X_test_scaled)
print("XGBoost Model Accuracy:", accuracy_score(y_test, y_pred))
```

```
C:\Users\tanis\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\xgboost\training.py:183: UserWarning:
[14:48:22] WARNING: C:\actions-runner\_work\xgboost\xgboost\src\
learner.cc:738:
Parameters: { "use_label_encoder" } are not used.

  bst.update(dtrain, iteration=i, fobj=obj)

XGBoost Model Accuracy: 0.7797752808988764
```

```
#KNN

from sklearn.neighbors import KNeighborsClassifier

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

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

knn = KNeighborsClassifier(n_neighbors=5)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"KNN Model Accuracy: {accuracy}")
```

```
KNN Model Accuracy: 0.737438202247191
```

```
#Random Forest

from sklearn.ensemble import RandomForestClassifier

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

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

rf = RandomForestClassifier()

rf.fit(X_train, y_train)

y_pred = rf.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Random Forest Model Accuracy: {accuracy}")
```

```
Random Forest Model Accuracy: 0.7788539325842697
```

```python
#Decision trees

from sklearn.tree import DecisionTreeClassifier

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

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

dt = DecisionTreeClassifier()

dt.fit(X_train, y_train)

y_pred = dt.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Decision Tree Model Accuracy: {accuracy}")
```

```
Decision Tree Model Accuracy: 0.6403970037453184
```

```python
#Gradient Boosting

from sklearn.ensemble import GradientBoostingClassifier

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

X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
test_size=0.2, random_state=42)

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

gb = GradientBoostingClassifier(n_estimators=200, learning_rate=0.1,
max_depth=6, random_state=42)
gb.fit(X_train_scaled, y_train)

y_pred = gb.predict(X_test_scaled)
print("Gradient Boost Model Accuracy:", accuracy_score(y_test,
y_pred))
```

```
Gradient Boost Model Accuracy: 0.7796910112359551
```

```python
#Logistic Regression

from sklearn.linear_model import LogisticRegression
```

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

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

lr = LogisticRegression()

lr.fit(X_train, y_train)

y_pred = lr.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)

print(f"Logistic Regression Model Accuracy: {accuracy}")

Logistic Regression Model Accuracy: 0.7789662921348315

C:\Users\tanis\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\linear_model\
_logistic.py:469: 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(
```

Now , upon looking at the algorithms used we can see the accuracy is coming out as very low , we hence choose the top 3 of the above algorithms and will try them again using hyperparameter tuning to get the best possible results.

The top 3 algorithms yet are: 1.XGBoost 2.Gradient Boosting 3.Logistic Regression

```
#Updated Logistic Regression

from sklearn.model_selection import train_test_split, GridSearchCV,
RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,
roc_auc_score, confusion_matrix

X = df.drop('survived', axis=1)
y = df['survived']
```

```
categorical_cols = X.select_dtypes(include=['object',
'category']).columns

X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

log_reg_baseline = LogisticRegression(random_state=42,
solver='liblinear')
log_reg_baseline.fit(X_train, y_train)
y_pred_baseline = log_reg_baseline.predict(X_test)
y_proba_baseline = log_reg_baseline.predict_proba(X_test)[:, 1]

param_grid = {
    'penalty': ['l1', 'l2'],
    'C': np.logspace(-4, 4, 10),
    'solver': ['liblinear', 'saga'],
    'max_iter': [100, 200, 500],
    'class_weight': [None, 'balanced']
}

log_reg = LogisticRegression(random_state=42)

grid_search = GridSearchCV(estimator=log_reg,
                           param_grid=param_grid,
                           cv=5,
                           scoring='accuracy',
                           verbose=0,
                           n_jobs=-1)

grid_search.fit(X_train, y_train)

best_log_reg_grid = grid_search.best_estimator_
y_pred_grid = best_log_reg_grid.predict(X_test)
y_proba_grid = best_log_reg_grid.predict_proba(X_test)[:, 1]

log_reg_rand = LogisticRegression(random_state=42)

param_distributions_simplified = {
    'penalty': ['l1', 'l2'],
    'C': np.logspace(-4, 4, 100),
    'solver': ['liblinear', 'saga'],
    'max_iter': [100, 200, 500, 1000],
    'class_weight': [None, 'balanced']
}

rand_search = RandomizedSearchCV(estimator=log_reg_rand,

param_distributions=param_distributions_simplified,
```

```
                                  n_iter=100,
                                  cv=5,
                                  scoring='accuracy',
                                  verbose=0,
                                  random_state=42,
                                  n_jobs=-1)

rand_search.fit(X_train, y_train)

best_log_reg_rand = rand_search.best_estimator_
y_pred_rand = best_log_reg_rand.predict(X_test)
y_proba_rand = best_log_reg_rand.predict_proba(X_test)[:, 1]

final_log_reg = LogisticRegression(**grid_search.best_params_,
random_state=42)

final_log_reg.fit(X_train, y_train)

y_pred_final = final_log_reg.predict(X_test)
y_proba_final = final_log_reg.predict_proba(X_test)[:, 1]
```

```
----------------------------------------------------------------------
-----
KeyboardInterrupt                         Traceback (most recent call
last)
Cell In[30], line 38
     29 log_reg = LogisticRegression(random_state=42)
     31 grid_search = GridSearchCV(estimator=log_reg,
     32                            param_grid=param_grid,
     33                            cv=5,
     34                            scoring='accuracy',
     35                            verbose=0,
     36                            n_jobs=-1)
---> 38 grid_search.fit(X_train, y_train)
     40 best_log_reg_grid = grid_search.best_estimator_
     41 y_pred_grid = best_log_reg_grid.predict(X_test)

File ~\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\base.py:1474, in
_fit_context.<locals>.decorator.<locals>.wrapper(estimator, *args,
**kwargs)
   1467     estimator._validate_params()
   1469 with config_context(
   1470     skip_parameter_validation=(
   1471         prefer_skip_nested_validation or
global_skip_validation
   1472     )
   1473 ):
-> 1474     return fit_method(estimator, *args, **kwargs)
```

```
File ~\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\model_selection\
_search.py:970, in BaseSearchCV.fit(self, X, y, **params)
    964     results = self._format_results(
    965         all_candidate_params, n_splits, all_out,
all_more_results
    966     )
    968     return results
--> 970 self._run_search(evaluate_candidates)
    972 # multimetric is determined here because in the case of a
callable
    973 # self.scoring the return type is only known after calling
    974 first_test_score = all_out[0]["test_scores"]

File ~\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\model_selection\
_search.py:1527, in GridSearchCV._run_search(self,
evaluate_candidates)
   1525 def _run_search(self, evaluate_candidates):
   1526     """Search all candidates in param_grid"""
-> 1527     evaluate_candidates(ParameterGrid(self.param_grid))

File ~\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\model_selection\
_search.py:916, in
BaseSearchCV.fit.<locals>.evaluate_candidates(candidate_params, cv,
more_results)
    908 if self.verbose > 0:
    909     print(
    910         "Fitting {0} folds for each of {1} candidates,"
    911         " totalling {2} fits".format(
    912             n_splits, n_candidates, n_candidates * n_splits
    913         )
    914     )
--> 916 out = parallel(
    917     delayed(_fit_and_score)(
    918         clone(base_estimator),
    919         X,
    920         y,
    921         train=train,
    922         test=test,
    923         parameters=parameters,
    924         split_progress=(split_idx, n_splits),
    925         candidate_progress=(cand_idx, n_candidates),
    926         **fit_and_score_kwargs,
    927     )
```

```
    928     for (cand_idx, parameters), (split_idx, (train, test)) in
product(
    929         enumerate(candidate_params),
    930         enumerate(cv.split(X, y,
**routed_params.splitter.split)),
    931     )
    932 )
    934 if len(out) < 1:
    935     raise ValueError(
    936         "No fits were performed. "
    937         "Was the CV iterator empty? "
    938         "Were there no candidates?"
    939     )

File ~\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\sklearn\utils\parallel.py:67, in
Parallel.__call__(self, iterable)
     62 config = get_config()
     63 iterable_with_config = (
     64     (_with_config(delayed_func, config), args, kwargs)
     65     for delayed_func, args, kwargs in iterable
     66 )
---> 67 return super().__call__(iterable_with_config)

File ~\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\joblib\parallel.py:2007, in
Parallel.__call__(self, iterable)
   2001 # The first item from the output is blank, but it makes the
interpreter
   2002 # progress until it enters the Try/Except block of the
generator and
   2003 # reach the first `yield` statement. This starts the
aynchronous
   2004 # dispatch of the tasks to the workers.
   2005 next(output)
-> 2007 return output if self.return_generator else list(output)

File ~\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\joblib\parallel.py:1650, in
Parallel._get_outputs(self, iterator, pre_dispatch)
   1647     yield
   1649     with self._backend.retrieval_context():
-> 1650         yield from self._retrieve()
   1652 except GeneratorExit:
   1653     # The generator has been garbage collected before being
fully
   1654     # consumed. This aborts the remaining tasks if possible
```

```
and warn
   1655     # the user if necessary.
   1656     self._exception = True

File ~\AppData\Local\Packages\
PythonSoftwareFoundation.Python.3.11_qbz5n2kfra8p0\LocalCache\local-
packages\Python311\site-packages\joblib\parallel.py:1762, in
Parallel._retrieve(self)
   1757 # If the next job is not ready for retrieval yet, we just wait
for
   1758 # async callbacks to progress.
   1759 if ((len(self._jobs) == 0) or
   1760     (self._jobs[0].get_status(
   1761         timeout=self.timeout) == TASK_PENDING)):
-> 1762     time.sleep(0.01)
   1763     continue
   1765 # We need to be careful: the job list can be filling up as
   1766 # we empty it and Python list are not thread-safe by
   1767 # default hence the use of the lock

KeyboardInterrupt:
```

```
#Upgraded Gradient Boosting

from sklearn.model_selection import train_test_split, GridSearchCV,
RandomizedSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import accuracy_score, classification_report,
roc_auc_score, confusion_matrix

X = df.drop('survived', axis=1)
y = df['survived']

categorical_cols = X.select_dtypes(include=['object',
'category']).columns
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

print(f"Shape of X_train after encoding: {X_train.shape}")
print(f"Shape of X_test after encoding: {X_test.shape}")

gb_baseline = GradientBoostingClassifier(random_state=42)
gb_baseline.fit(X_train, y_train)
y_pred_baseline = gb_baseline.predict(X_test)
y_proba_baseline = gb_baseline.predict_proba(X_test)[:, 1]

param_grid_gb = {
    'n_estimators': [50, 100, 200],
```

```
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2]
}

gb_model = GradientBoostingClassifier(random_state=42)

grid_search_gb = GridSearchCV(estimator=gb_model,
                              param_grid=param_grid_gb,
                              cv=5,
                              scoring='accuracy',
                              verbose=1,
                              n_jobs=-1)

grid_search_gb.fit(X_train, y_train)

best_gb_grid = grid_search_gb.best_estimator_
y_pred_grid_gb = best_gb_grid.predict(X_test)
y_proba_grid_gb = best_gb_grid.predict_proba(X_test)[:, 1]

param_distributions_gb_rand = {
    'n_estimators': [int(x) for x in np.linspace(start = 20, stop =
200, num = 10)],
    'learning_rate': [0.01, 0.05, 0.1, 0.15, 0.2],
    'max_depth': [3, 4, 5, 6, 7],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4],
    'subsample': [0.7, 0.8, 0.9, 1.0],
    'max_features': ['sqrt', 'log2', None]
}

gb_model_rand = GradientBoostingClassifier(random_state=42)

rand_search_gb = RandomizedSearchCV(estimator=gb_model_rand,

param_distributions=param_distributions_gb_rand,
                                    n_iter=50,
                                    cv=5,
                                    scoring='accuracy',
                                    verbose=1,
                                    random_state=42,
                                    n_jobs=-1)

rand_search_gb.fit(X_train, y_train)

best_gb_rand = rand_search_gb.best_estimator_
y_pred_rand_gb = best_gb_rand.predict(X_test)
y_proba_rand_gb = best_gb_rand.predict_proba(X_test)[:, 1]
```

```
final_gb_model =
GradientBoostingClassifier(**grid_search_gb.best_params_,
random_state=42)
final_gb_model.fit(X_train, y_train)

y_pred_final_gb = final_gb_model.predict(X_test)
y_proba_final_gb = final_gb_model.predict_proba(X_test)[:, 1]

#Upgraded XGBoost

from sklearn.model_selection import train_test_split, GridSearchCV,
RandomizedSearchCV
from sklearn.metrics import accuracy_score, classification_report,
roc_auc_score, confusion_matrix
import xgboost as xgb

X = df.drop('survived', axis=1)
y = df['survived']

categorical_cols = X.select_dtypes(include=['object',
'category']).columns
X = pd.get_dummies(X, columns=categorical_cols, drop_first=True)

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

print(f"Shape of X_train after encoding: {X_train.shape}")
print(f"Shape of X_test after encoding: {X_test.shape}")

xgb_baseline = xgb.XGBClassifier(objective='binary:logistic',
eval_metric='logloss', use_label_encoder=False, random_state=42)
xgb_baseline.fit(X_train, y_train)
y_pred_baseline = xgb_baseline.predict(X_test)
y_proba_baseline = xgb_baseline.predict_proba(X_test)[:, 1]

print(f"Baseline Accuracy: {accuracy_score(y_test,
y_pred_baseline):.4f}")
print(f"Baseline Classification Report:\
n{classification_report(y_test, y_pred_baseline)}")
print(f"Baseline AUC-ROC: {roc_auc_score(y_test,
y_proba_baseline):.4f}")
print(f"Baseline Confusion Matrix:\n{confusion_matrix(y_test,
y_pred_baseline)}")

param_grid_xgb = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.7, 0.9, 1.0],
    'colsample_bytree': [0.7, 0.9, 1.0],
```

```
    'gamma': [0, 0.1, 0.2],
    'reg_alpha': [0, 0.001, 0.1],
    'reg_lambda': [1, 10, 100]
}

xgb_model = xgb.XGBClassifier(objective='binary:logistic',
eval_metric='logloss', use_label_encoder=False, random_state=42)

grid_search_xgb = GridSearchCV(estimator=xgb_model,
                               param_grid=param_grid_xgb,
                               cv=5,
                               scoring='accuracy',
                               verbose=0,
                               n_jobs=-1)

grid_search_xgb.fit(X_train, y_train)

print(f"Best parameters from GridSearchCV:
{grid_search_xgb.best_params_}")
print(f"Best cross-validation accuracy from GridSearchCV:
{grid_search_xgb.best_score_:.4f}")

best_xgb_grid = grid_search_xgb.best_estimator_
y_pred_grid_xgb = best_xgb_grid.predict(X_test)
y_proba_grid_xgb = best_xgb_grid.predict_proba(X_test)[:, 1]

print(f"Test Accuracy (GridSearchCV XGB): {accuracy_score(y_test,
y_pred_grid_xgb):.4f}")
print(f"Classification Report (GridSearchCV XGB):\
n{classification_report(y_test, y_pred_grid_xgb)}")
print(f"AUC-ROC (GridSearchCV XGB): {roc_auc_score(y_test,
y_proba_grid_xgb):.4f}")
print(f"Confusion Matrix (GridSearchCV XGB):\
n{confusion_matrix(y_test, y_pred_grid_xgb)}")

param_distributions_xgb_rand = {
    'n_estimators': [int(x) for x in np.linspace(start = 50, stop =
500, num = 10)],
    'learning_rate': [0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3],
    'max_depth': [3, 4, 5, 6, 7, 8, 9, 10],
    'subsample': [0.6, 0.7, 0.8, 0.9, 1.0],
    'colsample_bytree': [0.6, 0.7, 0.8, 0.9, 1.0],
    'gamma': [0, 0.05, 0.1, 0.2, 0.3, 0.4],
    'reg_alpha': [0, 0.0001, 0.001, 0.01, 0.1, 1, 10],
    'reg_lambda': [1, 5, 10, 50, 100]
}

xgb_model_rand = xgb.XGBClassifier(objective='binary:logistic',
eval_metric='logloss', use_label_encoder=False, random_state=42)
```

```
rand_search_xgb = RandomizedSearchCV(estimator=xgb_model_rand,

param_distributions=param_distributions_xgb_rand,
                                     n_iter=50,
                                     cv=5,
                                     scoring='accuracy',
                                     verbose=0,
                                     random_state=42,
                                     n_jobs=-1)

rand_search_xgb.fit(X_train, y_train)

print(f"Best parameters from RandomizedSearchCV:
{rand_search_xgb.best_params_}")
print(f"Best cross-validation accuracy from RandomizedSearchCV:
{rand_search_xgb.best_score_:.4f}")

best_xgb_rand = rand_search_xgb.best_estimator_
y_pred_rand_xgb = best_xgb_rand.predict(X_test)
y_proba_rand_xgb = best_xgb_rand.predict_proba(X_test)[:, 1]

print(f"Test Accuracy (RandomizedSearchCV XGB):
{accuracy_score(y_test, y_pred_rand_xgb):.4f}")
print(f"Classification Report (RandomizedSearchCV XGB):\
n{classification_report(y_test, y_pred_rand_xgb)}")
print(f"AUC-ROC (RandomizedSearchCV XGB): {roc_auc_score(y_test,
y_proba_rand_xgb):.4f}")
print(f"Confusion Matrix (RandomizedSearchCV XGB):\
n{confusion_matrix(y_test, y_pred_rand_xgb)}")

final_xgb_model = xgb.XGBClassifier(**grid_search_xgb.best_params_,
objective='binary:logistic', eval_metric='logloss',
use_label_encoder=False, random_state=42)
final_xgb_model.fit(X_train, y_train)

y_pred_final_xgb = final_xgb_model.predict(X_test)
y_proba_final_xgb = final_xgb_model.predict_proba(X_test)[:, 1]

print(f"Final XGBoost Model Evaluation (Using Best Params on Test
Set)")
print(f"Final Test Accuracy: {accuracy_score(y_test,
y_pred_final_xgb):.4f}")
print(f"Final Classification Report:\n{classification_report(y_test,
y_pred_final_xgb)}")
print(f"Final AUC-ROC: {roc_auc_score(y_test,
y_proba_final_xgb):.4f}")
print(f"Final Confusion Matrix:\n{confusion_matrix(y_test,
y_pred_final_xgb)}")
```

Unfortunately , due to the lack of resources , i wasnt able to run the hypertuning part on my laptop.