Titanic Disaster
Final Project

"The sinking of the Titanic is one of the most infamous shipwrecks in history.
On April 15, 1912, during her maiden voyage, the widely considered "unsinkable" RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren't enough lifeboats for everyone onboard, resulting in the death of 1502 out of 2224 passengers and crew.
While there was some element of luck involved in surviving, it seems some groups of people were more likely to survive than others."

From "Titanic - Machine Learning from Disaster. https://www.kaggle.com/competitions/titanic"

INTRODUCTION

The objective of this project is to predict the survival of passengers aboard the Titanic using passenger data.
This project is from the Kaggle competition **Titanic - Machine Learning from Disaster.*

Dataset

The dataset contains information on the passengers, such as age, gender, ticket class and family size.
Train and test data sizes are respectively 891 and 418.

Evaluation

Accuracy is the main evaluation metric.

Method

I will compare 3 models, Logistic Regression, Random Forest and XGBoost, and will use Bayesian Optimization for hyperparameter tuning to improve model performance.

**https://www.kaggle.com/competitions/titanic*

Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFECV
import statsmodels.api as sm
from sklearn.linear_model import LogisticRegression
from xgboost import XGBClassifier
from bayes_opt import BayesianOptimization
import xgboost as xgb
import warnings
```

Bayesian Optimization

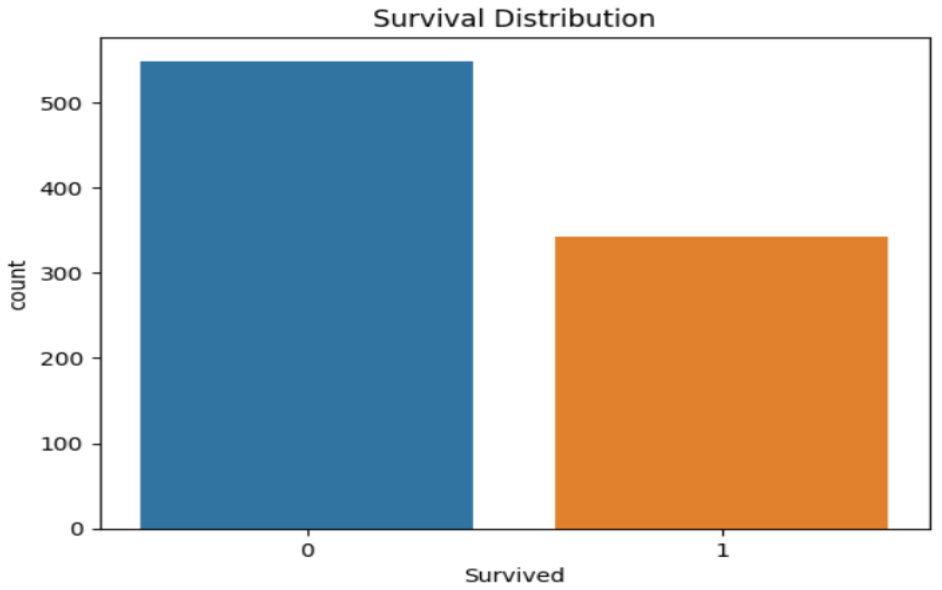
I chose Bayesian Optimization to tune hyperparameters.
It’s efficient and easy to implement.

Exploratory Data Analysis Visualizations Part 1

```
warnings.simplefilter(action="ignore", category=FutureWarning)

data_train = pd.read_csv(r"/kaggle/input/titanic/train.csv")
data_test = pd.read_csv("/kaggle/input/titanic/test.csv")
print(data_train)
print("-"* 80)
print(data_test)
print(data_train.isnull().sum())
print("-"*80)
print(data_test.isnull().sum())
data_test[data_test["Fare"].isna()]
data_test["Fare"].fillna(0, inplace = True)
data_test["Fare"].iloc[152]

sns.countplot(x="Survived", data=data_train)
plt.title('Survival Distribution')
```

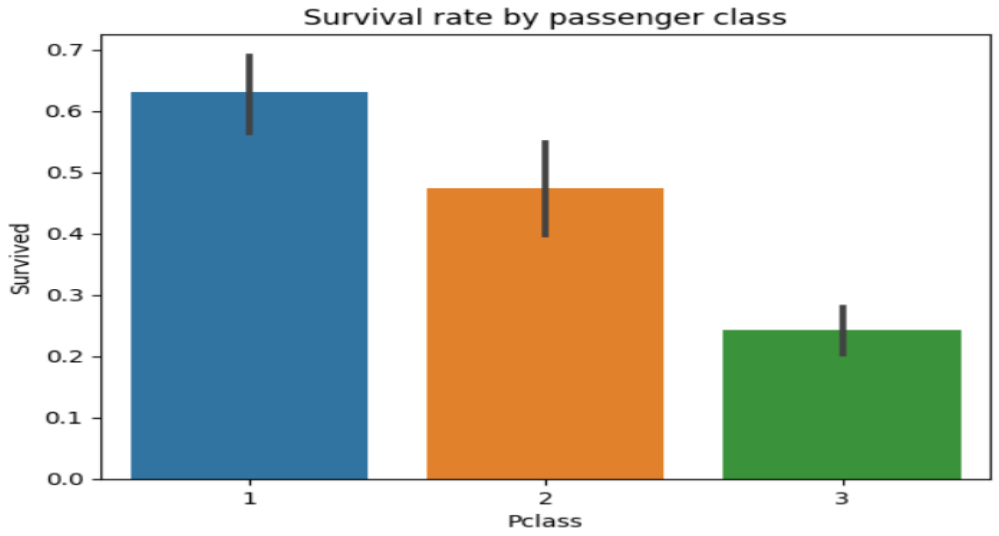
Classes Survived = 1 / Did Not Survive = 0

Classes are significantly imbalanced.

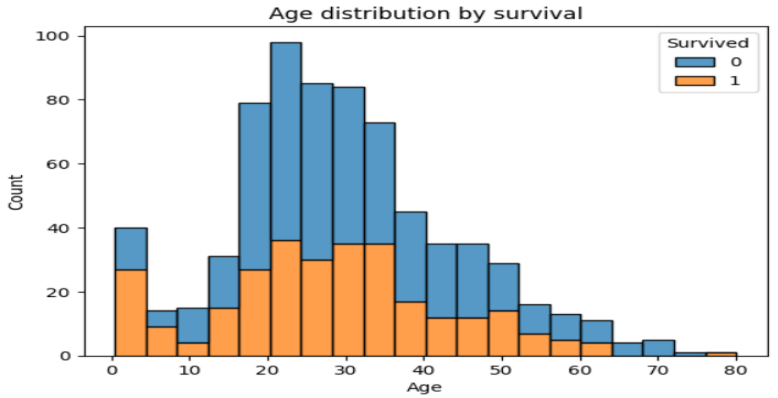


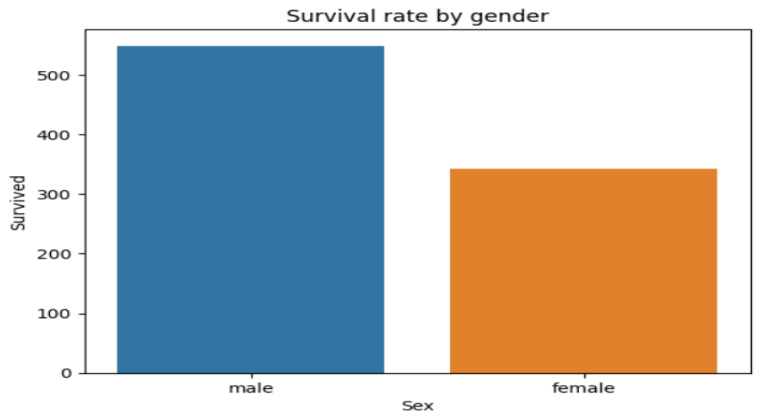
Exploratory Data Analysis Visualizations Part 2

```
sns.barplot(x="Sex", y="Survived", data=data_train)
plt.title("Survival rate by gender")
plt.show()

sns.barplot(x="Pclass", y="Survived", data=data_train)
plt.title("Survival rate by passenger class")
plt.show()

sns.histplot(data=data_train, x="Age", hue="Survived", multiple="stack")
plt.title("Age distribution by survival")
plt.show()
```



Survival rate by element





Exploratory Data Analysis
Data Preparation

```
data_train.dtypes
data_train["Sex"] = LabelEncoder().fit_transform(data_train["Sex"])
data_test["Sex"] = LabelEncoder().fit_transform(data_test["Sex"])

data_train_eval_NA = data_train.dropna()
X_eval_NA = pd.DataFrame(data_train_eval_NA, columns = ["Pclass", "Sex", "Age", "SibSp","Parch", "Fare"])
y_eval_NA = data_train_eval_NA["Survived"]

logit_model = sm.Logit(y_eval_NA, X_eval_NA)
result = logit_model.fit()
print(result.summary())

data_train[["Fare", "Cabin", "Age"]] = data_train[["Fare", "Cabin", "Age"]].fillna(0)
data_test[["Fare", "Cabin", "Age"]] = data_test[["Fare", "Cabin", "Age"]].fillna(0)

data_train["Sex"] = LabelEncoder().fit_transform(data_train["Sex"])
data_test["Sex"] = LabelEncoder().fit_transform(data_test["Sex"])

X = pd.DataFrame(data_train, columns = ["Pclass", "Sex", "Fare", "Age"])
y = data_train["Survived"]

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.2, random_state = 42)
print(X_train.shape, X_val.shape, y_train.shape, y_val.shape)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
```

Data cleaning and preparation

- Categorical data labeled
- Data scaled
- Statistical significance of features evaluated with logit function before dropping /keeping

Logit Regression Results

Dep. Variable:	Survived	No. Observations:	183
Model:	Logit	Df Residuals:	177
Method:	MLE	Df Model:	5
Date:	Sat, 12 Oct 2024	Pseudo R-squ.:	0.1970
Time:	21:29:58	Log-Likelihood:	-92.973
converged:	True	LL-Null:	-115.78
Covariance Type:	nonrobust	LLR p-value:	1.092e-08

	coef	std err	z	P>|z|	[0.025	0.975]
Pclass	0.9035	0.270	3.343	0.001	0.374	1.433
Sex	-2.1268	0.392	-5.419	0.000	-2.896	-1.358
Age	0.0056	0.009	0.604	0.546	-0.013	0.024
SibSp	0.4204	0.333	1.263	0.207	-0.232	1.073
Parch	-0.2875	0.280	-1.026	0.305	-0.837	0.262
Fare	0.0079	0.004	2.186	0.029	0.001	0.015

Model 1
Logistic Regression

```
logreg = LogisticRegression(max_iter = 2000)
logreg.fit(X_train_scaled, y_train)

y_pred_val = logreg.predict(X_val)
baseline_accuracy = accuracy_score(y_val, y_pred_val)
print(f"Baseline Logistic Regression Accuracy: {baseline_accuracy:.4f}")

X_test = pd.DataFrame(data_test, columns = ["Pclass", "Sex", "Fare", "Age"])
X_test_scaled = scaler.transform(X_test)

y_pred_test = logreg.predict(X_test_scaled)
logreg_predictions = (y_pred_test > 0.5).astype("int64")
passenger_ids = data_test["PassengerId"].values
results = pd.DataFrame({"PassengerId" : passenger_ids, "Survived" : logreg_predictions})
results.to_csv("logreg_predictions.csv", index=False)
```

Best performance

Accuracy on test data : **0.76794**

Model 2
Random Forest

```python
def rf_cv(n_estimators, max_depth, min_samples_split):
    rf = RandomForestClassifier(
        n_estimators=int(n_estimators),
        max_depth=int(max_depth),
        min_samples_split=int(min_samples_split),
        random_state=42
    )
    return cross_val_score(rf, X_train, y_train, cv=5).mean()
pbounds_rf = {
    "n_estimators": (50, 500),
    "max_depth": (3, 20),
    "min_samples_split": (2, 10)
}
optimizer_rf = BayesianOptimization(f=rf_cv, pbounds=pbounds_rf, random_state=42)
optimizer_rf.maximize(init_points=5, n_iter=20)
best_params_rf = optimizer_rf.max["params"]
best_rf = RandomForestClassifier(
    n_estimators=int(best_params_rf["n_estimators"]),
    max_depth=int(best_params_rf["max_depth"]),
    min_samples_split=int(best_params_rf["min_samples_split"]),
    random_state=42
)
best_rf.fit(X_train, y_train)
predicted_values = best_rf.predict(X_test)
predictions_rf = (predicted_values > 0.5).astype("int64")
rf_predictions = pd.DataFrame({"PassengerId": passenger_ids, "Survived": predictions_rf})
rf_predictions.to_csv("rf_predictions.csv", index = False)
```

Best performance

Accuracy on test data : **0.75827**

Model 3
XGBoost

```python
def xgb_cv(n_estimators, max_depth, learning_rate):
    xgboost_model = xgb.XGBClassifier(
        n_estimators=int(n_estimators),
        max_depth=int(max_depth),
        learning_rate=learning_rate,
        objective="binary:logistic",
        use_label_encoder=False,
        eval_metric='logloss'
    )
    return cross_val_score(xgboost_model, X_train, y_train, cv=5).mean()
pbounds_xgb = {
    'n_estimators': (50, 500),
    'max_depth': (3, 20),
    'learning_rate': (0.01, 0.3)
}
optimizer_xgb = BayesianOptimization(f=xgb_cv, pbounds=pbounds_xgb, random_state=42)
optimizer_xgb.maximize(init_points=5, n_iter=20)
best_params_xgb = optimizer_xgb.max["params"]
best_xgb = xgb.XGBClassifier(
    n_estimators=int(best_params_xgb["n_estimators"]),
    max_depth=int(best_params_xgb["max_depth"]),
    learning_rate=best_params_xgb["learning_rate"],
    use_label_encoder=False,
    eval_metric="logloss"
)
best_xgb.fit(X_train, y_train)
predicted_values = best_xgb.predict(X_test)
predictions_xgb= (predicted_values > 0.5).astype("int64")
predicted_xgb = pd.DataFrame({"PassengerId": passenger_ids, "Survived": predictions_xgb})
predicted_xgb.to_csv("xgb_predictions.csv", index = False)
```

Best performance

Accuracy on test data : **0.78229**

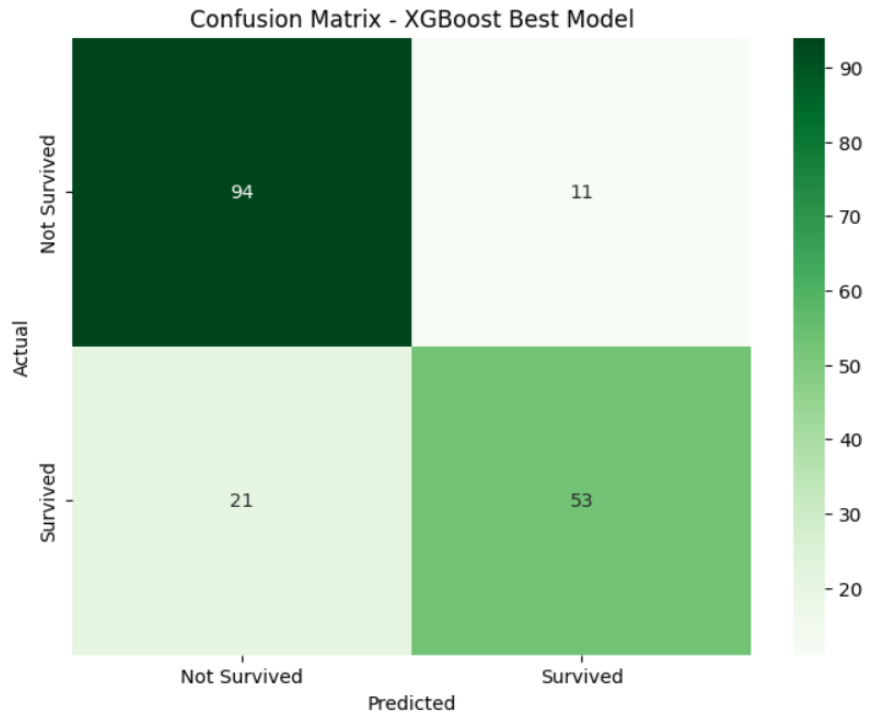
Best Model Performance XGBoost

```
y_val_pred = best_xgb.predict(X_val)

conf_matrix = confusion_matrix(y_val, y_val_pred)
print("Confusion Matrix:")
print(conf_matrix)

class_report = classification_report(y_val, y_val_pred)
print("\nClassification Report:")
print(class_report)

plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Greens", xticklabels=["Not Survived", "Survived"], yticklabels=["Not Survived", "Survived"])
plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.title("Confusion Matrix - XGBoost Best Model")
plt.show()
```

Classification Report:

	precision	recall	f1-score	support
0	0.82	0.90	0.85	105
1	0.83	0.72	0.77	74
accuracy			0.82	179
macro avg	0.82	0.81	0.81	179
weighted avg	0.82	0.82	0.82	179

Best model confusion matrix and classification report



CONCLUSION

In this project, I explored three models to predict outcomes in the Titanic dataset:
Logistic Regression, **Random Forest** and **XGBoost**.

Performances on test data :

- **XGBoost**: Accuracy = **0.78229**
- **Logistic Regression**: Accuracy = **0.76794**
- **Random Forest**: Accuracy = **0.75827**

XGBoost showed the best performance, in particular recall for class 0 was 0.90, meaning it correctly identified 90% of the passengers who did not survive.

It might be interesting for next time to further tune the XGBoost parameters along with deeper data preparation, especially regarding the replacement of missing values (NAs) in the data.