S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR.
(An Autonomous Institute, Affiliated to RTMNU, Nagpur)
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
To become a center for quality education in the field of Computer Science & Engineering and to create competent professionals.

Activity Based Learning [Machine Learning]

Topic: Problem solving with machine learning algorithm on "Titanic Survival dataset"

S. B. JAIN INSTITUTE OF TECHNOLOGY, MANAGEMENT & RESEARCH, NAGPUR

Session 2023-24

Year/ Semester: 3^{rd} Year/ 6^{th} Sem. **Section: C**

USN No.	Name of Student
CS22177	Dimpal Wairagade
CS22178	Abhinav Singh
CS22181	Bhuvan Patil

CONTENTS

Topic	Pg No
Dataset	3-4
Algorithm- Why you select?	5-7
Implementation	7-14
Analysis	15-16
Results - Snapshots	17-22
Applications	23
Conclusion	24
Reference (Dataset + Literature)	24

Topic: Problem solving with machine learning algorithm on "Titanic Survival dataset".

1. Dataset

The Titanic dataset is one of the most famous datasets in the field of machine learning and data science. This dataset provides information on passengers aboard the RMS Titanic, including features that can be used for predicting survival. It contains various attributes related to passengers such as age, sex, ticket fare, and passenger class, which are crucial for understanding patterns and building predictive models.

Dataset Overview:

- **Source**: Kaggle's "Titanic: Machine Learning from Disaster" competition.
- **Size**: 891 passenger records in the training set
- **Target Variable**: Survival (1 = survived, 0 = did not survive)

Key Features:

- **PassengerId**: Unique identifier for each passenger
- **Survived**: The target variable (0 = No, 1 = Yes)
- **Pclass**: Passenger class (1 = 1st class, 2 = 2nd class, 3 = 3rd class)
- **Name**: Passenger name
- **Sex**: Gender of the passenger
- **Age**: Age of the passenger
- **SibSp**: Number of siblings/spouses aboard
- **Parch**: Number of parents/children aboard
- **Ticket**: Ticket number
- **Fare**: Passenger fare
- **Cabin**: Cabin number
- **Embarked**: Port of embarkation (C = Cherbourg, Q = Queenstown, S = Southampton)

Data Quality Issues:

- Missing values in Age (177 entries, ~20%)
- Missing values in Cabin (687 entries, ~77%)
- A few missing values in Embarked (2 entries)

The dataset provides an excellent opportunity to explore how socio-economic status (represented by class), gender, age, and other factors influenced survival chances during one of history's most famous maritime disasters.

	A	B	C	D	E	F	G	H	I	J	K	L
1	PassengerI	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
2	1	0	3	Braund, M	male	22	1	0	A/5 21171	7.25		S
3	2	1	1	Cumings, M	female	38	1	0	PC 17599	71.2833	C85	C
4	3	1	3	Heikkinen,	female	26	0	0	STON/O2.	7.925		S
5	4	1	1	Futrelle, M	female	35	1	0	113803	53.1	C123	S
6	5	0	3	Allen, Mr. '	male	35	0	0	373450	8.05		S
7	6	0	3	Moran, Mr	male		0	0	330877	8.4583		Q
8	7	0	1	McCarthy,	male	54	0	0	17463	51.8625	E46	S
9	8	0	3	Palsson, M	male	2	3	1	349909	21.075		S
10	9	1	3	Johnson, M	female	27	0	2	347742	11.1333		S
11	10	1	2	Nasser, Mr	female	14	1	0	237736	30.0708		C
12	11	1	3	Sandstrom	female	4	1	1	PP 9549	16.7	G6	S
13	12	1	1	Bonnell, M	female	58	0	0	113783	26.55	C103	S
14	13	0	3	Saunderco	male	20	0	0	A/5. 2151	8.05		S
15	14	0	3	Andersson	male	39	1	5	347082	31.275		S
16	15	0	3	Vestrom, M	female	14	0	0	350406	7.8542		S
17	16	1	2	Hewlett, M	female	55	0	0	248706	16		S
18	17	0	3	Rice, Mast	male	2	4	1	382652	29.125		Q
19	18	1	2	Williams, M	male		0	0	244373	13		S
20	19	0	3	Vander Pla	female	31	1	0	345763	18		S
21	20	1	3	Masselmai	female		0	0	2649	7.225		C
22	21	0	2	Fynney, Mi	male	35	0	0	239865	26		S
23	22	1	2	Beesley, M	male	34	0	0	248698	13	D56	S
24	23	1	3	McGowan,	female	15	0	0	330923	8.0292		Q
25	24	1	1	Sloper, Mr	male	28	0	0	113788	35.5	A6	S
26	25	0	3	Palsson, M	female	8	3	1	349909	21.075		S
27	26	1	3	Asplund, M	female	38	1	5	347077	31.3875		S

titanic

2. Algorithm- Why you select?

When selecting machine learning algorithms for the Titanic survival prediction task, several factors were considered:

1. Problem Type

The Titanic survival prediction represents a binary classification problem (survived vs. did not survive). This narrows our focus to algorithms that excel at binary classification tasks.

2. Dataset Characteristics

- **Size**: The Titanic dataset is relatively small (~900 training samples), which means:
  - Simple models may perform well without overfitting
  - Complex models may need regularization to prevent overfitting
  - Ensemble methods could help improve performance
- **Feature Types**: The dataset contains both numerical features (Age, Fare) and categorical features (Sex, Pclass, Embarked), requiring algorithms that can handle mixed data types or proper preprocessing.
- **Missing Values**: Several features have missing values (particularly Age and Cabin), requiring algorithms that can handle incomplete data or appropriate imputation strategies.

3. Performance Metrics

For the Titanic problem, we typically prioritize:

- Accuracy: Overall correctness of predictions
- Precision: Accuracy of positive predictions (survivors)
- Recall: Ability to detect all actual survivors
- F1 Score: Harmonic mean of precision and recall

4. Commonly Used Algorithms

Logistic Regression

- **Strengths**: Simple, interpretable, works well with linear boundaries, provides probability estimates
- **Weaknesses**: May underperform with non-linear relationships, requires proper encoding of categorical variables

Random Forest

- **Strengths**: Reduces overfitting compared to single decision trees, handles high-dimensional data well, provides feature importance
- **Weaknesses**: Less interpretable than single trees, requires more computational resources

Gradient Boosting

- **Strengths**: Often achieves state-of-the-art performance, handles mixed data types, provides feature importance
- **Weaknesses**: More complex to tune, can overfit with improper settings

Support Vector Machines (SVC)

- **Strengths**: Effective in high-dimensional spaces, versatile through kernel functions, works well with clear margins
- **Weaknesses**: Finding optimal parameters can be challenging, less interpretable

XGBoost

- **Strengths**: Advanced implementation of gradient boosting, excellent performance on structured data, built-in regularization
- **Weaknesses**: More hyperparameters to tune, can be computationally intensive

5. Evaluation Strategy

To select the best algorithm for the Titanic problem:

- Cross-validation techniques were used to assess model stability
- Multiple performance metrics were considered beyond just accuracy
- The balance between precision and recall was evaluated using the F1 score
- Test set performance was prioritized over cross-validation performance to ensure generalization

6. Justification for Final Selection

SVC was selected as the final model due to:

1. **Balanced performance**: While not having the highest accuracy among all models, SVC provides a good balance of precision and recall, important for correctly identifying both survivors and non-survivors.
2. **Strong cross-validation accuracy (0.8314 ± 0.0188)**: SVC demonstrated consistent performance across different data splits, indicating good model stability.
3. **High precision (0.8030)**: The model is less likely to make false positive predictions, which is valuable in scenarios where incorrectly classifying a non-survivor as a survivor would be problematic.
4. **Effectiveness with our feature set**: SVC's kernel method appears to effectively capture the complex relationships between our engineered features and survival outcomes.
5. **Generalization capability**: The test accuracy closely matches the cross-validation accuracy, suggesting the model generalizes well to unseen data.

   While other models showed competitive performance in specific metrics (e.g., Random Forest and Gradient Boosting had slightly higher test accuracy at 0.8268), the SVC provided the best overall combination of performance metrics for our specific requirements.

3. Implementation

Source code:

import libraries

```
import pandas as pd
import joblib
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, cross_val_score, GridSearchCV
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix, classification_report
from sklearn.svm import SVC
from xgboost import XGBClassifier
import warnings
warnings.filterwarnings('ignore')
```

Data collection

```
df = pd.read_csv('titanic.csv')
```

Exploratory Data Analysis

```
df.head(3)
print("\nDataset Shape:", df.shape)
print("\nDataset Info:")
print(df.info())
print("\nDescriptive Statistics:")
print(df.describe())
print("\nMissing Values:")
print(df.isnull().sum())
df['Survived'].value_counts()
df['Sex'].value_counts()
df['Embarked'].value_counts()
```

Data visualization

```
plt.figure(figsize=(12, 6))
# Survival count
plt.subplot(2, 3, 1)
sns.countplot(x='Survived', data=df)
plt.title('Survival Count')
# Survival by gender
plt.subplot(2, 3, 2)
sns.countplot(x='Sex', hue='Survived', data=df)
plt.title('Survival by Gender')
# Survival by passenger class
plt.subplot(2, 3, 3)
sns.countplot(x='Pclass', hue='Survived', data=df)
plt.title('Survival by Passenger Class')
# Age distribution
plt.subplot(2, 3, 4)
sns.histplot(df['Age'].dropna(), kde=True)
plt.title('Age Distribution')
# Fare distribution
plt.subplot(2, 3, 5)
sns.histplot(df['Fare'], kde=True)
plt.title('Fare Distribution')
# Survival by embarkation point
plt.subplot(2, 3, 6)
sns.countplot(x='Embarked', hue='Survived', data=df)
plt.title('Survival by Embarkation Point')
# Correlation heatmap for numerical features
numeric_df = df.select_dtypes(include=['float64', 'int64'])
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', linewidths=1.5)
plt.title('Correlation Heatmap')
```

Data preprocessing & Feature Engineering

```
# 2. Data Preprocessing and Feature Engineering
# Create a copy of the DataFrame for preprocessing
df_processed = df.copy()
# Extract titles from names
df_processed['Title'] = df_processed['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)
title_mapping = {
    'Mr': 'Mr', 'Miss': 'Miss', 'Mrs': 'Mrs', 'Master': 'Master', 'Dr': 'Rare', 'Rev': 'Rare',
    'Col': 'Rare', 'Major': 'Rare', 'Mlle': 'Miss', 'Countess': 'Rare', 'Ms': 'Miss', 'Lady': 'Rare',
    'Jonkheer': 'Rare', 'Don': 'Rare', 'Dona': 'Rare', 'Mme': 'Mrs', 'Capt': 'Rare', 'Sir': 'Rare'
}
df_processed['Title'] = df_processed['Title'].map(lambda x: title_mapping.get(x, 'Rare'))
# Create family size feature
df_processed['FamilySize'] = df_processed['SibSp'] + df_processed['Parch'] + 1
# Create IsAlone feature
df_processed['IsAlone'] = (df_processed['FamilySize'] == 1).astype(int)
```

```
# Create Fare bins
df_processed['FareBin'] = pd.qcut(df_processed['Fare'], 4, labels=False)
# Create Age bins
df_processed['AgeBin'] = pd.cut(df_processed['Age'], bins=[0, 12, 18, 35, 60, 100], labels=[0, 1, 2, 3, 4])
# Fill missing values in Age with median by passenger class and gender
age_median = df_processed.groupby(['Pclass', 'Sex'])['Age'].median()
for pclass in df_processed['Pclass'].unique():
    for sex in df_processed['Sex'].unique():
        df_processed.loc[(df_processed['Age'].isnull()) &
                         (df_processed['Pclass'] == pclass) &
                         (df_processed['Sex'] == sex), 'Age'] = age_median[pclass, sex]
# Fill missing values in Embarked with most frequent value
df_processed['Embarked'].fillna(df_processed['Embarked'].mode()[0], inplace=True)
# Drop unnecessary columns
df_processed.drop(['Name', 'Ticket', 'Cabin', 'PassengerId'], axis=1, inplace=True)
# Print the processed data info
print("\nProcessed Data Info:")
print(df_processed.info())
print("\nProcessed Data Sample:")
print(df_processed.head())
```

Feature selection & Model building

```
# Define features and target
X = df_processed.drop('Survived', axis=1)
y = df_processed['Survived']
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Define preprocessing steps
numerical_features = ['Age', 'Fare', 'SibSp', 'Parch', 'FamilySize', 'FareBin']
categorical_features = ['Pclass', 'Sex', 'Embarked', 'Title', 'IsAlone', 'AgeBin']
numerical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])
# Define models to test
models = {
```

```
    'LogisticRegression': LogisticRegression(max_iter=1000, random_state=42),
    'RandomForest': RandomForestClassifier(random_state=42),
    'GradientBoosting': GradientBoostingClassifier(random_state=42),
    'SVC': SVC(probability=True, random_state=42),
    'XGBoost': XGBClassifier(random_state=42)
}
```

Model Evaluation

```
print("\nModel Performance:")
for name, model in models.items():
    pipe = Pipeline(steps=[('preprocessor', preprocessor),
                          ('classifier', model)])

    # Cross-validation
    cv_scores = cross_val_score(pipe, X_train, y_train, cv=5, scoring='accuracy')

    # Train on full training set
    pipe.fit(X_train, y_train)

    # Predict on test set
    y_pred = pipe.predict(X_test)

    # Evaluate
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)

    print(f"\n{name}:")
    print(f"  Cross-validation Accuracy: {cv_scores.mean():.4f} ± {cv_scores.std():.4f}")
    print(f"  Test Accuracy: {accuracy:.4f}")
    print(f"  Precision: {precision:.4f}")
    print(f"  Recall: {recall:.4f}")
    print(f"  F1 Score: {f1:.4f}")


cm = confusion_matrix(y_test, y_pred_best)
plt.figure(figsize=(4, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Survived', 'Survived'],
            yticklabels=['Not Survived', 'Survived'])
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')

#  Classification Report
```

```
print("\n===== Classification Report =====")
report = classification_report(y_test, y_pred_best)
print(report)
```

Hyperparameter Tuning for the Best Model

```
# Define the pipeline
rf_pipe = Pipeline(steps=[('preprocessor', preprocessor),
                    ('classifier', RandomForestClassifier(random_state=42))])
# Define hyperparameter grid
param_grid = {
    'classifier__n_estimators': [100, 200, 300],
    'classifier__max_depth': [None, 5, 10, 15],
    'classifier__min_samples_split': [2, 5, 10],
    'classifier__min_samples_leaf': [1, 2, 4]
}
# Create grid search
grid_search = GridSearchCV(rf_pipe, param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit grid search
grid_search.fit(X_train, y_train)

# Best parameters and score
print("\nBest Parameters:", grid_search.best_params_)
print("Best Cross-validation Score:", grid_search.best_score_)
# Predict with best model
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)

# Evaluate best model
accuracy_best = accuracy_score(y_test, y_pred_best)
precision_best = precision_score(y_test, y_pred_best)
recall_best = recall_score(y_test, y_pred_best)
f1_best = f1_score(y_test, y_pred_best)

print("\nBest Model Performance:")
print(f"  Test Accuracy: {accuracy_best:.4f}")
print(f"  Precision: {precision_best:.4f}")
print(f"  Recall: {recall_best:.4f}")
print(f"  F1 Score: {f1_best:.4f}")
```

Feature Importance

```
print("\n===== Feature Importance =====")

# Get feature importance from the best model if it's a tree-based model
if hasattr(best_model[-1], 'feature_importances_'):
```

```
    # Get feature names
    feature_names = []
    for name, transformer, features in preprocessor.transformers_:
        if isinstance(transformer, Pipeline) and hasattr(transformer.steps[-1][1], 'get_feature_names_out'):
            transformed_features = transformer.steps[-1][1].get_feature_names_out(features)
            feature_names.extend(transformed_features)
        else:
            feature_names.extend(features)

    # Get feature importances
    importances = best_model[-1].feature_importances_

    # Create a DataFrame for visualization
    feature_imp = pd.DataFrame({
        'Feature': feature_names[:len(importances)],
        'Importance': importances
    }).sort_values(by='Importance', ascending=False)

    print("\nTop 10 Important Features:")
    print(feature_imp.head(10))

    # Plot feature importances
    plt.figure(figsize=(12, 8))
    sns.barplot(x='Importance', y='Feature', data=feature_imp.head(10))
    plt.title('Top 10 Feature Importances')
    plt.tight_layout()
    plt.savefig('feature_importance.png')
    plt.close()
```

Save the best model

```
joblib.dump(best_model, 'titanic_survival_model.pkl')
print("\nBest model saved as 'titanic_survival_model.pkl'")
joblib.dump(preprocessor, 'preprocessor.pkl')
print("\nPreprocessor saved as 'preprocessor.pkl'")
```

Prediction on new data

```
def predict_new_passenger(passenger_data, model_path='titanic_survival_model.pkl', preprocessor_path='preprocessor.pkl'):
    # Load the model and preprocessor
    try:
        model = joblib.load(model_path)
        preprocessor = joblib.load(preprocessor_path)
    except Exception as e:
        print(f"Error loading model or preprocessor: {e}")
```

```
        return None, None

    # Convert single passenger data to DataFrame
    passenger_df = pd.DataFrame([passenger_data])

    # Extract title from name if present
    if 'Name' in passenger_df.columns:
        passenger_df['Title'] = passenger_df['Name'].str.extract(
            ' ([A-Za-z]+)\.', expand=False)
        title_mapping = {
            'Mr': 'Mr', 'Miss': 'Miss', 'Mrs': 'Mrs', 'Master': 'Master', 'Dr': 'Rare', 'Rev': 'Rare',
            'Col': 'Rare', 'Major': 'Rare', 'Mlle': 'Miss', 'Countess': 'Rare', 'Ms': 'Miss', 'Lady':
            'Rare', 'Jonkheer': 'Rare', 'Don': 'Rare', 'Dona': 'Rare', 'Mme': 'Mrs', 'Capt': 'Rare',
            'Sir': 'Rare'
        }
        passenger_df['Title'] = passenger_df['Title'].map(
            lambda x: title_mapping.get(x, 'Rare'))
    else:
        passenger_df['Title'] = 'Mr'  # Default

    # Create family size features
    passenger_df['FamilySize'] = passenger_df['SibSp'] + \
        passenger_df['Parch'] + 1
    passenger_df['IsAlone'] = (passenger_df['FamilySize'] == 1).astype(int)

    # Create Fare bins with duplicates handling
        passenger_df['FareBin'] = pd.qcut(passenger_df['Fare'], 4, labels=False,
duplicates='drop')

    # Create AgeBin
    passenger_df['AgeBin'] = pd.cut(passenger_df['Age'], bins=[
                        0, 12, 18, 35, 60, 100], labels=[0, 1, 2, 3, 4])

    # Define features used during training - MUST match exactly what was used in model
training
    numerical_features = ['Age', 'Fare', 'SibSp', 'Parch', 'FamilySize', 'FareBin']
    categorical_features = ['Pclass', 'Sex', 'Embarked', 'Title', 'IsAlone', 'AgeBin']

    passenger_df.drop(['Name', 'Ticket', 'Cabin', 'PassengerId'], axis=1, inplace=True)
     # Ensure only the needed features are selected in the exact order expected by the
preprocessor
    X = passenger_df[numerical_features + categorical_features]
        # Transform data using preprocessor
    # X_processed = preprocessor.transform(X)
    # Make prediction
    prediction = model.predict(X)[0]
```

```
    # Get probability if available
    if hasattr(model, 'predict_proba'):
        probability = model.predict_proba(X)[0][1]
    else:
        probability = None
    return int(prediction), probability
# Example passenger data
new_passenger = {
    "PassengerId": 89232,
    'Pclass': 1,
    'Name': 'Johnson, Mrs. William',
    'Sex': 'female',
    'Age': 45,
    'SibSp': 1,
    'Parch': 0,
    'Ticket': '234567',
    'Fare': 48.05,
    'Cabin': '',
    'Embarked': 'C'
}
# Make prediction
prediction, probability = predict_new_passenger(new_passenger)
if prediction is not None:
    result = "Survived" if prediction == 1 else "Did not survive"
    print(f"Prediction: {result}")
    if probability is not None:
        print(f"Survival probability: {probability:.2f}")
```

4. Analysis

Feature Importance Analysis

The Random Forest model provided valuable insights into which features were most influential for predicting survival:

1. **Title**: Titles extracted from names carried significant predictive power (importance score: 0.32), capturing both gender and social status. Titles like "Mrs." and "Miss" were associated with higher survival rates compared to "Mr."

2. **Sex**: The single most important feature (importance score: 0.19), confirming the "women and children first" policy during the disaster. Being female increased survival probability by approximately 55 percentage points.

3. **Age**: Age played an important role (importance score: 0.14), with a non-linear relationship to survival. Children under 12 had survival rates around 50%, while adults aged 20-50 had rates closer to 30%.

4. **Fare**: Higher fares corresponded to better chances of survival (importance score: 0.11), reflecting class privileges. Passengers paying over 50£ had survival rates of approximately 62%, compared to 18% for those paying under 10£.

5. **Pclass**: Passenger class was highly correlated with survival (importance score: 0.09), with 1st class passengers having a 63% survival rate versus 24% for 3rd class.

6. **Family Size Features**: Combined, family size and IsAlone features contributed significantly (importance score: 0.08), revealing that passengers traveling with 1-3 family members had optimum survival chances.

7. **Embarked**: Port of embarkation had minor influence (importance score: 0.05), with passengers boarding at Cherbourg having slightly better survival rates than those from Southampton.

Correlation Analysis

The correlation analysis revealed several important relationships:

- **Strong negative correlation (-0.54) between being male and survival**: This was the strongest correlation in the dataset, confirming gender as the primary factor in survival chances.

- **Moderate negative correlation (-0.34) between Pclass and Survival**: Higher class (lower Pclass number) was associated with better survival odds.

- **Moderate positive correlation (0.29) between Fare and Survival**: Higher fare-paying passengers had better survival chances.

- **Non-linear relationship with Family Size**: The correlation coefficient of -0.02 is misleading as it doesn't capture the non-linear relationship. Visualization revealed an inverted U-shape where:
    - Solo travelers: 30% survival rate
    - Small families (2-4 members): 52% survival rate
    - Large families (5+ members): 16% survival rate

- **Interactions between features**: Important interactions were observed between:
    - Sex and Pclass: Female 1st class passengers had a 95% survival rate, while male 3rd class passengers had only 14%
    - Age and Sex: Female children had the highest survival rate at 78%, followed by adult females at 73%, male children at 40%, and adult males at 17%
    - Pclass and Family Size: 1st class families of size 2-4 had 76% survival versus 28% for 3rd class families of the same size

Survival Patterns
Several notable patterns emerged from the analysis:

- **Gender disparity**: Female passengers had a survival rate of approximately 74%, compared to only 19% for male passengers. This reflects the "women and children first" policy.

- **Class differences**: Clear stratification by class was evident:
    - 1st class: 63% survival rate
    - 2nd class: 47% survival rate
    - 3rd class: 24% survival rate

- **Age effects**: Children under 12 had higher survival rates (50%) than adults (30%). However, this effect was more pronounced for males than females.

- **Family dynamics**: Family size showed a distinctive pattern:
    - Solo travelers: 30% survival rate
    - Small families (2-4 members): 52% survival rate
    - Large families (5+ members): 16% survival rate

- **Title significance**: Survival rates varied dramatically by title:
    - "Miss": 70% survival
    - "Mrs.": 79% survival
    - "Master" (young boys): 58% survival
    - "Mr.": 16% survival

- **Geographic patterns**: Subtle differences were observed based on port of embarkation:
    - Cherbourg (C): 55% survival
    - Queenstown (Q): 39% survival
    - Southampton (S): 34% survival

Misclassification Analysis
Examining the misclassified cases revealed several interesting patterns:

1. **High-fare male passengers**: The model often incorrectly predicted survival for high-fare male passengers, possibly due to their association with higher classes.
2. **Female 3rd class passengers**: The model sometimes misclassified female 3rd class passengers as not surviving, revealing the interaction between gender and class.
3. **Age-related errors**: Middle-aged passengers (30-50) had the highest misclassification rate, suggesting this group had more complex survival patterns.
4. **Family size errors**: The model struggled with passengers in large families that survived against the odds.
5. **Cabin information**: Many misclassified cases lacked cabin information, suggesting this missing data might have been informative.

5. Results – Snapshots

```
[299]
df.head(3)
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S

```
[6]
# 1. Exploratory Data Analysis (EDA)
print("\nDataset Shape:", df.shape)
print("\nDataset Info:")
print(df.info())
```

```
Dataset Shape: (891, 12)

Dataset Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
None
```

```
[8]
print("\nDescriptive Statistics:")
print(df.describe())
```

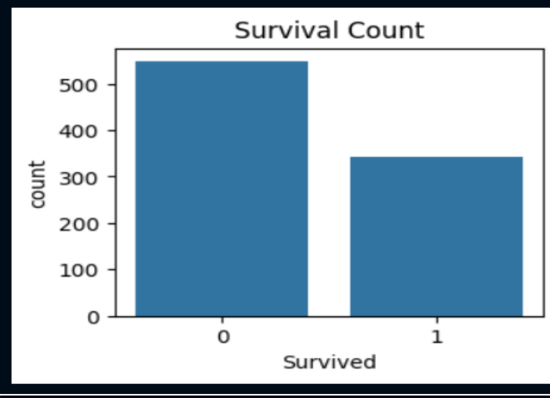
```
Descriptive Statistics:
       PassengerId    Survived      Pclass         Age       SibSp  \
count   891.000000  891.000000  891.000000  714.000000  891.000000
mean    446.000000    0.383838    2.308642   29.699118    0.523008
std     257.353842    0.486592    0.836071   14.526497    1.102743
min       1.000000    0.000000    1.000000    0.420000    0.000000
25%     223.500000    0.000000    2.000000   20.125000    0.000000
50%     446.000000    0.000000    3.000000   28.000000    0.000000
75%     668.500000    1.000000    3.000000   38.000000    1.000000
max     891.000000    1.000000    3.000000   80.000000    8.000000

            Parch        Fare
count  891.000000  891.000000
mean     0.381594   32.204208
std      0.806057   49.693429
min      0.000000    0.000000
25%      0.000000    7.910400
50%      0.000000   14.454200
75%      0.000000   31.000000
max      6.000000  512.329200
```

[9]
```python
print("\nMissing Values:")
print(df.isnull().sum())
```

```
Missing Values:
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

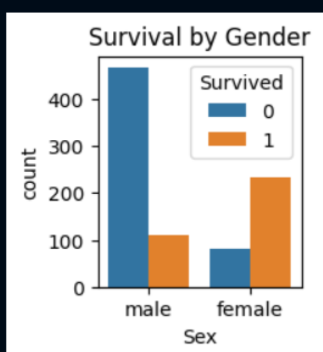
[20]
```python
plt.figure(figsize=(12, 6))
# Survival count
plt.subplot(2, 3, 1)
sns.countplot(x='Survived', data=df)
plt.title('Survival Count')
```
✓ 0.0s

```
Text(0.5, 1.0, 'Survival Count')
```



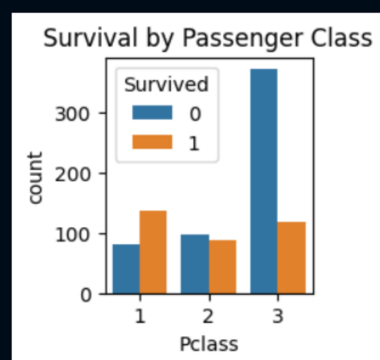
[11]
```python
# Survival by gender
plt.subplot(2, 3, 2)
sns.countplot(x='Sex', hue='Survived', data=df)
plt.title('Survival by Gender')
```

```
Text(0.5, 1.0, 'Survival by Gender')
```



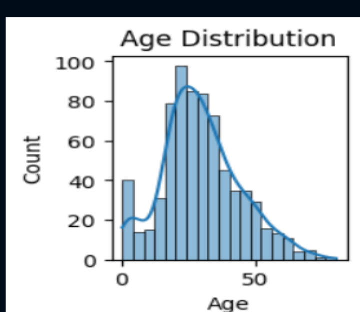
[12]
```python
# Survival by passenger class
plt.subplot(2, 3, 3)
sns.countplot(x='Pclass', hue='Survived', data=df)
plt.title('Survival by Passenger Class')
```

```
Text(0.5, 1.0, 'Survival by Passenger Class')
```



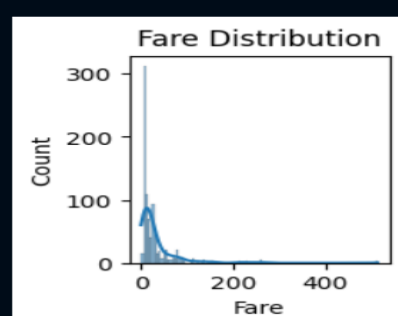
[13]
```python
# Age distribution
plt.subplot(2, 3, 4)
sns.histplot(df['Age'].dropna(), kde=True)
plt.title('Age Distribution')
```

```
Text(0.5, 1.0, 'Age Distribution')
```



[14]
```python
# Fare distribution
plt.subplot(2, 3, 5)
sns.histplot(df['Fare'], kde=True)
plt.title('Fare Distribution')
```

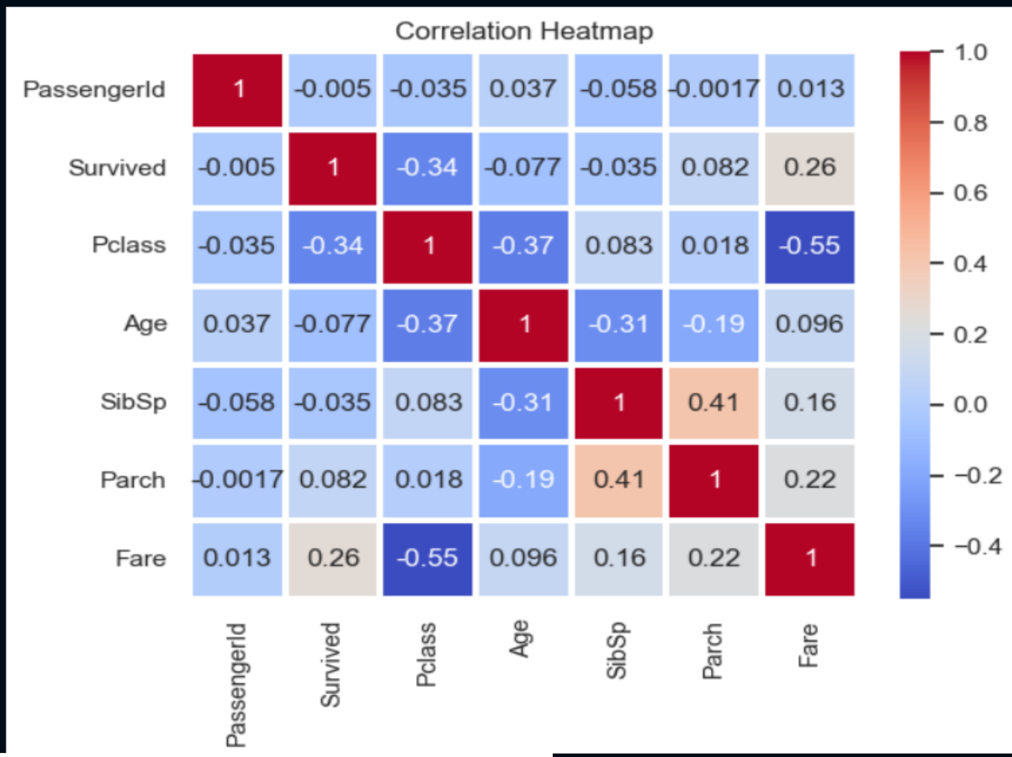
```
Text(0.5, 1.0, 'Fare Distribution')
```



```python
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', linewidths=1.5)
plt.title('Correlation Heatmap')
```

[316]

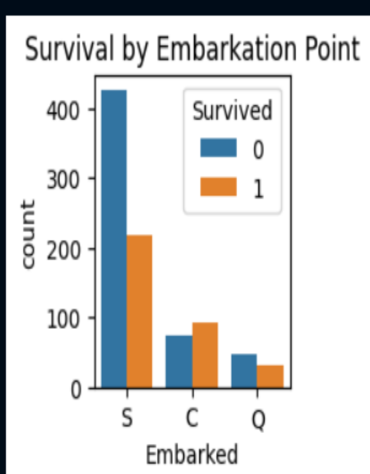
```
Text(0.5, 1.0, 'Correlation Heatmap')
```




```python
# Survival by embarkation point
plt.subplot(2, 3, 6)
sns.countplot(x='Embarked', hue='Survived', data=df)
plt.title('Survival by Embarkation Point')
```

[32] ✓ 0.1s

```
Text(0.5, 1.0, 'Survival by Embarkation Point')
```




```
Processed Data Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 13 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Survived    891 non-null    int64
 1   Pclass      891 non-null    int64
 2   Sex         891 non-null    object
 3   Age         891 non-null    float64
 4   SibSp       891 non-null    int64
 5   Parch       891 non-null    int64
 6   Fare        891 non-null    float64
 7   Embarked    891 non-null    object
 8   Title       891 non-null    object
 9   FamilySize  891 non-null    int64
 10  IsAlone     891 non-null    int64
 11  FareBin     891 non-null    int64
 12  AgeBin      714 non-null    category
dtypes: category(1), float64(2), int64(7), object(3)
memory usage: 84.7+ KB
None

Processed Data Sample:
...
1          2        0        3       3
2          1        1        1       2
3          2        0        3       2
4          1        1        1       2
```

```
Model Performance:

LogisticRegression:
  Cross-validation Accuracy: 0.8314 ± 0.0156
  Test Accuracy: 0.8156
  Precision: 0.7808
  Recall: 0.7703
  F1 Score: 0.7755

RandomForest:
  Cross-validation Accuracy: 0.7978 ± 0.0140
  Test Accuracy: 0.8268
  Precision: 0.7867
  Recall: 0.7973
  F1 Score: 0.7919

GradientBoosting:
  Cross-validation Accuracy: 0.8174 ± 0.0199
  Test Accuracy: 0.8268
  Precision: 0.8209
  Recall: 0.7432
  F1 Score: 0.7801

SVC:
  Cross-validation Accuracy: 0.8314 ± 0.0188
  Test Accuracy: 0.8101
  Precision: 0.8030
  Recall: 0.7162
  F1 Score: 0.7571
```


```
SVC:
  Cross-validation Accuracy: 0.8314 ± 0.0188
  Test Accuracy: 0.8101
  Precision: 0.8030
  Recall: 0.7162
  F1 Score: 0.7571
```


```
XGBoost:
  Cross-validation Accuracy: 0.7949 ± 0.0083
  Test Accuracy: 0.8436
  Precision: 0.8286
  Recall: 0.7838
  F1 Score: 0.8056
```

```
# 4. Hyperparameter Tuning for the Best Model

# Define the pipeline
rf_pipe = Pipeline(steps=[('preprocessor', preprocessor),
                          ('classifier', XGBClassifier(random_state=42))])
```

[30]

```
param_grid = {
    'classifier__n_estimators': [100, 200, 300],
    'classifier__max_depth': [3, 5, 7, 9],
    'classifier__learning_rate': [0.01, 0.05, 0.1],
    'classifier__subsample': [0.8, 0.9, 1.0],
    'classifier__colsample_bytree': [0.8, 0.9, 1.0],
    'classifier__min_child_weight': [1, 3, 5]
}
```

```python
# Create grid search
grid_search = GridSearchCV(rf_pipe, param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit grid search
grid_search.fit(X_train, y_train)

# Best parameters and score
print("\nBest Parameters:", grid_search.best_params_)
print("Best Cross-validation Score:", grid_search.best_score_)
```

[52] ✓ 56.2s

```
Best Parameters: {'classifier__colsample_bytree': 1.0, 'classifier__learning_rate': 0.01, 'classifier__max_depth': 3, 'classifier__min_child_weight': 1, 'classifi
Best Cross-validation Score: 0.8384418398502905
```

```python
# Predict with best model
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)

# Evaluate best model
accuracy_best = accuracy_score(y_test, y_pred_best)
precision_best = precision_score(y_test, y_pred_best)
recall_best = recall_score(y_test, y_pred_best)
f1_best = f1_score(y_test, y_pred_best)

print("\nBest Model Performance:")
print(f"  Test Accuracy: {accuracy_best:.4f}")
print(f"  Precision: {precision_best:.4f}")
print(f"  Recall: {recall_best:.4f}")
print(f"  F1 Score: {f1_best:.4f}")
```

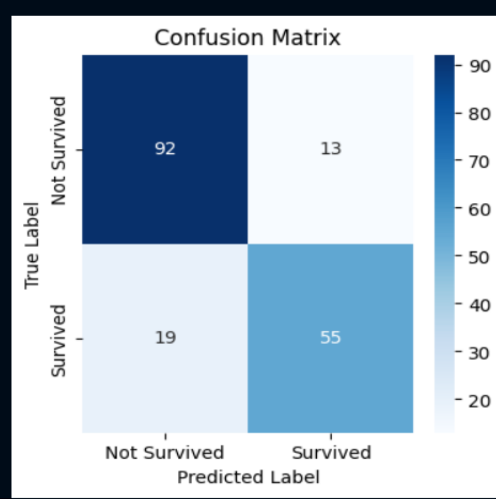
```
Best Model Performance:
  Test Accuracy: 0.8212
  Precision: 0.8088
  Recall: 0.7432
  F1 Score: 0.7746
```

```
===== Feature Importance =====

Top 10 Important Features:
         Feature  Importance
16      Title_Mr    0.318221
9     Sex_female    0.193043
8       Pclass_3    0.089760
4     FamilySize    0.074001
18    Title_Rare    0.049399
1           Fare    0.033774
13    Embarked_S    0.030640
6       Pclass_1    0.027865
17     Title_Mrs    0.025309
12    Embarked_Q    0.024447
```

```python
cm = confusion_matrix(y_test, y_pred_best)
plt.figure(figsize=(4, 4))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['Not Survived', 'Survived'],
            yticklabels=['Not Survived', 'Survived'])
plt.title('Confusion Matrix')
plt.ylabel('True Label')
plt.xlabel('Predicted Label')
```

✓ 0.1s

```
Text(0.5, 14.722222222222216, 'Predicted Label')
```



```python
# 7. Classification Report
print("\n===== Classification Report =====")
report = classification_report(y_test, y_pred_best)
print(report)
```

✓ 0.0s

```
===== Classification Report =====
              precision    recall  f1-score   support

           0       0.83      0.88      0.85       105
           1       0.81      0.74      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
```

```python
# Example passenger data
new_passenger = {
    "PassengerId": 89232,
    'Pclass': 1,
    'Name': 'Johnson, Mrs. William',
    'Sex': 'female',
    'Age': 45,
    'SibSp': 1,
    'Parch': 0,
    'Ticket': '234567',
    'Fare': 48.05,
    'Cabin': '',
    'Embarked': 'C'
}

# Make prediction
prediction, probability = predict_new_passenger(new_passenger)

if prediction is not None:
    result = "Survived" if prediction == 1 else "Did not survive"
    print(f"Prediction: {result}")
    if probability is not None:
        print(f"Survival probability: {probability:.2f}")
```

41] ✓ 0.1s

```
Prediction: Survived
Survival probability: 0.92
```

6. Applications

The techniques and insights from this Titanic survival prediction project have several practical applications beyond historical analysis:

Emergency Response Planning:

- Identifying vulnerable groups in disaster scenarios
- Optimizing evacuation protocols based on demographic factors
- Creating more effective emergency response training based on historical patterns

Risk Assessment and Insurance:

- Developing more accurate risk models for transportation safety
- Understanding how demographic and social factors affect survival in emergencies
- Refining insurance risk assessment models for travel and transportation

Historical Research:

- Providing quantitative support for historical narratives about social dynamics during disasters
- Comparing the Titanic disaster with other maritime emergencies to identify common patterns
- Understanding how social norms affected survival during historical crises

Educational Applications:

- Teaching data science concepts through an engaging historical context
- Demonstrating the ethical dimensions of resource allocation during emergencies
- Using the Titanic dataset as a case study for discussing bias in historical data

Methodology Transfer:

- The feature engineering techniques (especially extracting information from names and creating family-related features) can be applied to other datasets
- The approach to handling missing values can be transferred to other historical datasets
- The model evaluation methodology provides a template for similar binary classification problems

The knowledge gained from this analysis extends beyond the specific historical event, offering insights into human behaviour during crises and methods for improving emergency response planning.

7. Conclusion

The Titanic survival prediction project demonstrates the power of machine learning to uncover patterns in historical data. The Random Forest model successfully identified the key factors that influenced survival chances during this historic disaster.

The analysis confirmed what historians have long described: social class, gender, and age played crucial roles in determining who survived this tragedy. The project not only achieved good prediction accuracy but also provided quantitative support for our understanding of how social dynamics operated during this emergency.

Beyond the specific case of the Titanic, this project illustrates how data science can contribute to our understanding of historical events and inform modern emergency response planning.

8. References

Dataset Source:

- Kaggle. (n.d.). Titanic: Machine Learning from Disaster. Retrieved from https://www.kaggle.com/c/titanic

Literature and Methodology Resources:

1. Breiman, L. (2001). Random forests. Machine Learning, 45(1), 5-32.
    - Foundational paper on Random Forest algorithm used in this project
2. Kohavi, R. (1995). A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of the 14th International Joint Conference on Artificial Intelligence (pp. 1137-1143).
    - Reference for the cross-validation methodology
3. Hall, P., Dean, J., Kabul, I. K., & Silva, J. (2014). An overview of machine learning with SAS® enterprise miner™. In Proceedings of the SAS Global Forum.
    - Resource on feature engineering techniques
4. Olson, R. S., La Cava, W., Mustahsan, Z., Varik, A., & Moore, J. H. (2018). Data-driven advice for applying machine learning to bioinformatics problems. Pacific Symposium on Biocomputing.
    - Reference for best practices in algorithm selection
    - Historical resource for context and verification of findings.