```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
```

```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
```

1.0 Cleaning Data

```
df_train = pd.read_csv('/content/drive/MyDrive/000 DATA ANALYTICS  SCIENCE DOCS/KAGGLE COMP/Titanic/train.csv')
df_test = pd.read_csv('/content/drive/MyDrive/000 DATA ANALYTICS  SCIENCE DOCS/KAGGLE COMP/Titanic/test.csv')
```

```
df_train.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
```

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   PassengerId  418 non-null    int64  
 1   Pclass       418 non-null    int64  
 2   Name         418 non-null    object 
 3   Sex          418 non-null    object 
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64  
 6   Parch        418 non-null    int64  
 7   Ticket       418 non-null    object 
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object 
 10  Embarked     418 non-null    object 
dtypes: float64(2), int64(4), object(5)
memory usage: 36.1+ KB
```

We are going to drop 'Cabin' from both sets. Over 70% of data in that column is missing.

```
df_train.drop('Cabin', axis=1, inplace=True)
df_test.drop('Cabin', axis=1, inplace=True)

df_train.info()
```

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 11 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  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(4)
memory usage: 76.7+ KB
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 10 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(4)
memory usage: 32.8+ KB
```

```
df_test['Fare'].fillna(df_test['Fare'].median(), inplace=True)
df_train['Embarked'].fillna(df_train['Embarked'].mode()[0], inplace=True)
```

```
df_test['Survived'] = np.nan

# Combine them (doesn't sort but keeps rows together)
full_data = pd.concat([df_train, df_test], axis=0, sort=False).reset_index(drop=True)
```

```
full_data['Title'] = full_data['Name'].apply(lambda name: name.split(',')[1].split('.')[0].strip())

# Replace rare titles with 'Rare'
rare_titles = full_data['Title'].value_counts() < 10
full_data['Title'] = full_data['Title'].replace(rare_titles[rare_titles].index, 'Rare')

print(full_data[['Name', 'Title']].head())
print(full_data['Title'].value_counts())
```

```
                                                Name Title
0                            Braund, Mr. Owen Harris    Mr
1  Cumings, Mrs. John Bradley (Florence Briggs Th...   Mrs
2                             Heikkinen, Miss. Laina  Miss
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)   Mrs
4                           Allen, Mr. William Henry    Mr
Title
Mr        757
Miss      260
Mrs       197
Master     61
Rare       34
Name: count, dtype: int64
```

```
full_data['FamilySize'] = full_data['SibSp'] + full_data['Parch']
```

```
full_data['FamilySize'].isnull().sum()
```

```
np.int64(0)
```

```
def extract_ticket_prefix(ticket):
    ticket = ticket.replace('.', '').replace('/', '').strip().split()
    return ticket[0] if not ticket[0].isdigit() else 'NUMERIC'

full_data['TicketPrefix'] = full_data['Ticket'].apply(extract_ticket_prefix)

print(full_data[['Ticket', 'TicketPrefix']].head())
print(full_data['TicketPrefix'].value_counts())
```

```
             Ticket TicketPrefix
0         A/5 21171           A5
1          PC 17599           PC
2  STON/O2. 3101282       STONO2
3            113803      NUMERIC
4            373450      NUMERIC
TicketPrefix
NUMERIC    957
PC          92
CA          68
A5          28
SOTONOQ     24
WC          15
STONO       14
SCPARIS     14
A4          10
FCC          9
SOC          8
C            8
STONO2       7
SOPP         7
SCAH         5
SCParis      5
PP           4
WEP          4
LINE         4
SOTONO2      3
FC           3
PPP          2
SC           2
SWPP         2
SCA4         2
SOP          1
Fa           1
SP           1
SCOW         1
AS           1
CASOTON      1
SCA3         1
STONOQ       1
AQ4          1
A            1
LP           1
AQ3          1
Name: count, dtype: int64
```

```
prefix_counts = full_data['TicketPrefix'].value_counts()
rare_prefixes = prefix_counts[prefix_counts < 10].index

full_data['TicketPrefix'] = full_data['TicketPrefix'].apply(
    lambda x: 'RARE' if x in rare_prefixes else x
)
```

```
pd.crosstab(full_data[full_data['Survived'].notnull()]['TicketPrefix'],
            full_data[full_data['Survived'].notnull()]['Survived'],
            normalize='index')
```

Survived	0.0	1.0
TicketPrefix		
A4	1.000000	0.000000
A5	0.904762	0.095238
CA	0.658537	0.341463
NUMERIC	0.615734	0.384266
PC	0.350000	0.650000
RARE	0.614035	0.385965
SCPARIS	0.571429	0.428571
SOTONOQ	0.866667	0.133333
STONO	0.583333	0.416667
WC	0.900000	0.100000

```
from sklearn.preprocessing import LabelEncoder

non_numeric_cols = ['Sex', 'Embarked', 'Title', 'TicketPrefix']
label_encoders = {}

for col in non_numeric_cols:
    le = LabelEncoder()
    full_data[col] = le.fit_transform(full_data[col])
    label_encoders[col] = le  # store the encoder
```

```
dict(zip(label_encoders['Title'].classes_, label_encoders['Title'].transform(label_encoders['Title'].classes_)))
```

```
{'Master': np.int64(0),
 'Miss': np.int64(1),
 'Mr': np.int64(2),
 'Mrs': np.int64(3),
 'Rare': np.int64(4)}
```

```
full_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1309 entries, 0 to 1308
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype  
---  ------        --------------  -----  
 0   PassengerId   1309 non-null   int64  
 1   Survived      891 non-null    float64
 2   Pclass        1309 non-null   int64  
 3   Name          1309 non-null   object 
 4   Sex           1309 non-null   int64  
 5   Age           1046 non-null   float64
 6   SibSp         1309 non-null   int64  
 7   Parch         1309 non-null   int64  
 8   Ticket        1309 non-null   object 
 9   Fare          1309 non-null   float64
 10  Embarked      1309 non-null   int64  
 11  Title         1309 non-null   int64  
 12  FamilySize    1309 non-null   int64  
 13  TicketPrefix  1309 non-null   int64  
dtypes: float64(3), int64(9), object(2)
memory usage: 143.3+ KB
```

Start coding or generate with AI.

1.1 Handling Age

There is a decent number of missing age cells. Median is the simplest solution though, with so many to fill, can really imbalance the set. We do have a few options:

1. Use a few features to more accurately get a better estimate.
2. Train a model to predict age. Which is a little more complex but may help with final results later.

```
train_111 = full_data.copy()
train_112 = full_data.copy()
```

```
train_111['Age'] = train_111.groupby(['Pclass', 'Sex'])['Age'].transform(lambda x: x.fillna(x.median()))
```

```
age_features= [
    'Pclass', 'Sex', 'SibSp', 'Parch', 'Fare',
    'Embarked', 'Title', 'TicketPrefix'
]
```

```
known_age = full_data[full_data['Age'].notnull()]
unknown_age = full_data[full_data['Age'].isnull()]

X_age_train = known_age[age_features]
y_age_train = known_age['Age']
X_age_predict = unknown_age[age_features]
```

```
from sklearn.ensemble import RandomForestRegressor
age_model = RandomForestRegressor(n_estimators=100, random_state=42)
age_model.fit(X_age_train, y_age_train)
```

```
RandomForestRegressor
RandomForestRegressor(random_state=42)
```

```
predicted_ages = age_model.predict(X_age_predict)
```

```
full_data.loc[full_data['Age'].isnull(), 'Age'] = predicted_ages
```

```
full_data.isnull().sum()
```

	0
PassengerId	0
Survived	418
Pclass	0
Name	0
Sex	0
Age	0
SibSp	0
Parch	0
Ticket	0
Fare	0
Embarked	0
Title	0
FamilySize	0
TicketPrefix	0

dtype: int64

```
full_data['Age']
```

	Age
0	22.000000
1	38.000000
2	26.000000
3	35.000000
4	35.000000
...	...
1304	31.974421
1305	39.000000
1306	38.500000
1307	30.924158
1308	3.859560

1309 rows × 1 columns

dtype: float64

2.0 Building Predicting Models

```
# Make copies for each version
data_simple = full_data.copy()
data_complex = full_data.copy()
```

```
def split_for_modeling(data):
    train = data[data['Survived'].notnull()]
    test = data[data['Survived'].isnull()]

    X = train.drop(columns=['Survived', 'Name', 'Ticket', 'PassengerId'])
    y = train['Survived'].astype(int)
    X_test = test.drop(columns=['Survived', 'Name', 'Ticket', 'PassengerId'])
    return X, y, X_test, test['PassengerId']

X_simple, y_simple, X_test_simple, test_ids = split_for_modeling(data_simple)
X_complex, y_complex, X_test_complex, _ = split_for_modeling(data_complex)
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import cross_val_score
import numpy as np
```

```
try:
    from xgboost import XGBClassifier
    xgb_available = True
except ImportError:
    xgb_available = False
```

2.1 Testing Multiple Models

```
def get_models():
    models = {
        'LogisticRegression': LogisticRegression(max_iter=1000),
        'RandomForest': RandomForestClassifier(n_estimators=100, random_state=42),
        'KNN': KNeighborsClassifier()
    }
    if xgb_available:
        models['XGBoost'] = XGBClassifier(use_label_encoder=False, eval_metric='logloss')
    return models


def evaluate_models(X, y, label=''):
    models = get_models()
    results = {}
    for name, model in models.items():
        scores = cross_val_score(model, X, y, cv=5, scoring='accuracy')
        results[f'{label}_{name}'] = {
            'Mean Accuracy': np.mean(scores),
            'Std Dev': np.std(scores)
        }
    return results


# Evaluate models using group-median Age (Simple)
results_simple = evaluate_models(X_simple, y_simple, label='SimpleAge')

# Evaluate models using predicted Age (Complex)
results_complex = evaluate_models(X_complex, y_complex, label='ComplexAge')
```

```
import pandas as pd

# Combine both result dictionaries into a dataframe
df_results = pd.concat([
    pd.DataFrame(results_simple).T,
    pd.DataFrame(results_complex).T
])


# Sort by mean accuracy
df_results = df_results.sort_values(by='Mean Accuracy', ascending=False)

# Show the table
print(df_results)
```

```
                              Mean Accuracy   Std Dev
SimpleAge_RandomForest             0.822660  0.026322
ComplexAge_RandomForest            0.822660  0.026322
ComplexAge_XGBoost                 0.820438  0.027207
SimpleAge_XGBoost                  0.820438  0.027207
ComplexAge_LogisticRegression      0.798004  0.015539
SimpleAge_LogisticRegression       0.798004  0.015539
SimpleAge_KNN                      0.717224  0.030732
ComplexAge_KNN                     0.717224  0.030732
```

2.2 Hyper-Tuning Model

```
from sklearn.model_selection import GridSearchCV

param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [4, 6, 8, None],
    'min_samples_split': [2, 5],
    'min_samples_leaf': [1, 2],
    'max_features': ['sqrt', 'log2']
}

grid = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5, scoring='accuracy')
grid.fit(X_complex, y_complex)
```

```
print("Best Score:", grid.best_score_)
print("Best Params:", grid.best_params_)
```

```
Best Score: 0.8395016006528152
Best Params: {'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 2, 'min_samples_split': 5, 'n_estimator:
```

```
best_model = RandomForestClassifier(
    n_estimators=300,
    max_depth=None,
    max_features='sqrt',
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42
)
```

```
best_model.fit(X_complex, y_complex)
```

```
RandomForestClassifier

RandomForestClassifier(min_samples_leaf=2, min_samples_split=5,
                       n_estimators=300, random_state=42)
```

```
y_test_preds = best_model.predict(X_test_complex)
```

```
submission = pd.DataFrame({
    'PassengerId': test_ids,
    'Survived': y_test_preds.astype(int)
})
```

```
submission.to_csv('/content/drive/MyDrive/000 DATA ANALYTICS  SCIENCE DOCS/KAGGLE COMP/Titanic/submission.csv', index=Fal:
```

Start coding or generate with AI.

```
# **3.0 Summary**
The model on the test group was 76.79%...

Which is not great but real.
```

3.0 Summary

The model on the test group was 76.79%...

Which is not great but real.