In [1]:

```
#Importing pandas librarry to get some information about dataset

import pandas as pd

# Loadinf the training  and testing data of Titanic datasets

train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')
```

In [2]:

```
#  exploring the dataset
print(train_data.head())
```

```
   PassengerId  Survived  Pclass  \
0            1         0       3   
1            2         1       1   
2            3         1       3   
3            4         1       1   
4            5         0       3   

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1   
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1   
2                             Heikkinen, Miss. Laina  female  26.0      0   
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1   
4                           Allen, Mr. William Henry    male  35.0      0   

   Parch            Ticket     Fare Cabin Embarked  
0      0         A/5 21171   7.2500   NaN        S  
1      0          PC 17599  71.2833   C85        C  
2      0  STON/O2. 3101282   7.9250   NaN        S  
3      0            113803  53.1000  C123        S  
4      0            373450   8.0500   NaN        S  
```

In [3]:

```
print(train_data.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
```

In [4]:

```
# Droping  unnecessary columns or handling  missing values in the dataset

# here, dropping 'Name', 'Ticket', 'Cabin' columns which might not directly impact

train_data.drop(['Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
test_data.drop(['Name', 'Ticket', 'Cabin'], axis=1, inplace=True)
```

In [5]:
```
#Handling missing values (e.g., filling missing age values with median)

train_data['Age'].fillna(train_data['Age'].median(), inplace=True)
test_data['Age'].fillna(test_data['Age'].median(), inplace=True)
test_data['Fare'].fillna(test_data['Fare'].median(), inplace=True)
```

In [6]:
```
# Converting  categorical variables into numerical ones (e.g., 'Sex', 'Embarked')
train_data = pd.get_dummies(train_data, columns=['Sex', 'Embarked'])
test_data = pd.get_dummies(test_data, columns=['Sex', 'Embarked'])
```

In [7]:
```
# Defining  features and target variable

features = train_data.drop('Survived', axis=1)
target = train_data['Survived']
```

In [8]:
```
from sklearn.model_selection import KFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# Initializing the model with random forest classifier

model = RandomForestClassifier()

# Define the number of folds we want
num_folds = 5

# Initializing  KFolds
kfold = KFold(n_splits=num_folds, shuffle=True, random_state=42)
```

In [15]:
```
# Perform cross-validation
fold_accuracy = []
for fold, (train_idx, val_idx) in enumerate(kfold.split(features, target)):
    X_train, X_val = features.iloc[train_idx], features.iloc[val_idx]
    y_train, y_val = target.iloc[train_idx], target.iloc[val_idx]

    # Train the model
    model.fit(X_train, y_train)

    # Make predictions on validation set
    predictions = model.predict(X_val)

    # Calculate accuracy for this fold
    accuracy = accuracy_score(y_val, predictions)
    fold_accuracy.append(accuracy)
    print(f"Fold {fold+1} Accuracy: {accuracy}")
```

```
Fold 1 Accuracy: 0.8268156424581006
Fold 2 Accuracy: 0.7921348314606742
Fold 3 Accuracy: 0.848314606741573
Fold 4 Accuracy: 0.7808988764044944
Fold 5 Accuracy: 0.8258426966292135
```

In [ ]:

In [16]:
```
# Calculate average accuracy across all folds
average_accuracy = sum(fold_accuracy) / len(fold_accuracy)
print(f"Average Accuracy: {average_accuracy}")
```

```
Average Accuracy: 0.8148013307388112
```

In [17]:
```
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

```python
# Splitting the training data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(features, target, test_size=0.2,

# Initialize a simple model (e.g., RandomForestClassifier)
simple_model = RandomForestClassifier()

# Train the model using the training set
simple_model.fit(X_train, y_train)

# Validate the model using the validation set
val_predictions = simple_model.predict(X_val)

# Evaluate the model's performance on the validation set
val_accuracy = accuracy_score(y_val, val_predictions)
print(f"Validation Accuracy: {val_accuracy}")

# Generate a classification report for detailed performance analysis
print("Classification Report:")
print(classification_report(y_val, val_predictions))
```

```
Validation Accuracy: 0.8212290502793296
Classification Report:
              precision    recall  f1-score   support

           0       0.82      0.89      0.85       105
           1       0.82      0.73      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
```

In [18]:

```python
# Using RandomForestClassifier to get feature importances
feature_importances = model.feature_importances_

# Identify and select important features
important_features = pd.Series(feature_importances, index=features.columns).sort_va
selected_features = important_features[:8].index  # Select top 8 important features

# Retrain the model using selected features
selected_features_model = RandomForestClassifier()
selected_features_model.fit(X_train[selected_features], y_train)

# Validate the model using the selected features
val_predictions_selected = selected_features_model.predict(X_val[selected_features]

# Evaluate the model's performance on the validation set
val_accuracy_selected = accuracy_score(y_val, val_predictions_selected)
print(f"Validation Accuracy with Selected Features: {val_accuracy_selected}")
```

```
Validation Accuracy with Selected Features: 0.8156424581005587
```

In [19]:

```python
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

# Apply regularization with a Logistic Regression model
scaler = StandardScaler()
regularized_model = make_pipeline(scaler, LogisticRegression(penalty='l1', solver='
regularized_model.fit(X_train, y_train)

# Validate the model using regularization
val_predictions_regularized = regularized_model.predict(X_val)
```

```
# Evaluate the model's performance on the validation set
val_accuracy_regularized = accuracy_score(y_val, val_predictions_regularized)
print(f"Validation Accuracy with Regularization: {val_accuracy_regularized}")
```

Validation Accuracy with Regularization: 0.8044692737430168

In [20]:
```
from sklearn.ensemble import GradientBoostingClassifier

# Experimenting with GradientBoostingClassifier
gradient_boost_model = GradientBoostingClassifier(n_estimators=100, learning_rate=0
gradient_boost_model.fit(X_train, y_train)

# Validating the model using Gradient Boosting Classifier
val_predictions_gradient_boost = gradient_boost_model.predict(X_val)

# Evaluating the model's performance on the validation set
val_accuracy_gradient_boost = accuracy_score(y_val, val_predictions_gradient_boost)
print(f"Validation Accuracy with GradientBoostingClassifier: {val_accuracy_gradient
```

Validation Accuracy with GradientBoostingClassifier: 0.8156424581005587

In [21]:
```
from sklearn.svm import SVC

# Experimenting with Support Vector Machines
svm_model = SVC(kernel='linear', C=1.0, random_state=42)
svm_model.fit(X_train, y_train)

# Validate the model using Support Vector Machine
val_predictions_svm = svm_model.predict(X_val)

# Evaluate the model's performance on the validation data set
val_accuracy_svm = accuracy_score(y_val, val_predictions_svm)
print(f"Validation Accuracy with SVM: {val_accuracy_svm}")
```

Validation Accuracy with SVM: 0.776536312849162

In [9]:
```
###Experiments
```

In [10]:
```
####1  Experiment 1: Feature Engineering and Model Comparison
```

In [22]:
```
# Creating a 'FamilySize' feature by combining  SibSp and Parch
train_data['FamilySize'] = train_data['SibSp'] + train_data['Parch']
test_data['FamilySize'] = test_data['SibSp'] + test_data['Parch']

# Drop SibSp and Parch as they are now redundant
train_data.drop(['SibSp', 'Parch'], axis=1, inplace=True)
test_data.drop(['SibSp', 'Parch'], axis=1, inplace=True)

# Define features and target with new feature
features_with_family = train_data.drop('Survived', axis=1)
target = train_data['Survived']

# Performing train-test split to split data
X_train_with_family, X_val_with_family, y_train, y_val = train_test_split(features_
```

In [12]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression

# Train and evaluate different models using new features
models = {
    'RandomForest': RandomForestClassifier(random_state=42),
    'LogisticRegression': LogisticRegression(random_state=42)
}
```

```python
for model_name, model in models.items():
    model.fit(X_train_with_family, y_train)
    predictions = model.predict(X_val_with_family)
    accuracy = accuracy_score(y_val, predictions)
    print(f"{model_name} Accuracy: {accuracy}")
```

```
RandomForest Accuracy: 0.8324022346368715
LogisticRegression Accuracy: 0.7988826815642458
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: C
onvergenceWarning: 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(
```

In [23]:

```python
###Experiment 2 - Hyperparameter Tuning and Ensemble Methods
```

In [24]:

```python
from sklearn.model_selection import GridSearchCV

# Hyperparameter tuning for RandomForestClassifier
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 5, 10, 20],
    'min_samples_split': [2, 5, 10]
}

rf_model = RandomForestClassifier(random_state=42)
grid_search = GridSearchCV(rf_model, param_grid, cv=5)
grid_search.fit(X_train_with_family, y_train)

best_rf_model = grid_search.best_estimator_
best_rf_model.fit(X_train_with_family, y_train)
rf_predictions = best_rf_model.predict(X_val_with_family)
rf_accuracy = accuracy_score(y_val, rf_predictions)
print(f"RandomForest Tuned Accuracy: {rf_accuracy}")
```

```
RandomForest Tuned Accuracy: 0.8044692737430168
```

In [15]:

```python
from sklearn.ensemble import VotingClassifier

# Ensemble using VotingClassifier with the best-performing models
voting_model = VotingClassifier(
    estimators=[('RandomForest', best_rf_model), ('LogisticRegression', models['Log
    voting='hard'
)

voting_model.fit(X_train_with_family, y_train)
voting_predictions = voting_model.predict(X_val_with_family)
voting_accuracy = accuracy_score(y_val, voting_predictions)
print(f"Voting Classifier Accuracy: {voting_accuracy}")
```

```
Voting Classifier Accuracy: 0.8044692737430168
```

```
C:\Users\User\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:458: C
onvergenceWarning: 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(
```

```
In [ ]:
```