Background of Project

This project is a classic Kaggle project, the project requires us to use some information of passengers to predict whether it will survive on the Titanic. So this is a binary classification problem, which is very suitable for using unsupervised clustering algorithms

First let me explain what each feature of the training set represents:

- **survival : Did the passenger survive?**
- **pclass: The class of the passenger's seat**
- **sex : The Geneder of the passenger**
- **Age: Age of passenger**
- **sibsp: # of siblings / spouses aboard the Titanic**
- **parch: # of parents / children aboard the Titanic**
- **ticket: Ticket number**
- **fare: How much did the passenger pay for the seat**
- **cabin: Cabin number**
- **embarked: Port of Embarkation**

In [ ]:

```

```

1. Define the Problem

The training set contains some samples of passengers and tests the score of whether the passengers survived. It is necessary to train a model of whether the passengers in the test set survived. So this is a binary classification problem

In [73]:

```
import pandas as pd
import numpy as np
import random as rnd

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import cross_val_score
```

2. Data Load

In [74]:

```
train_df = pd.read_csv('../input/titanic/train.csv')
test_df = pd.read_csv('../input/titanic/test.csv')
combine = [train_df, test_df]
```

3. EDA

3.1 Analysis of Data

What features are included in the dataset?

In [75]:

```
print(train_df.columns.values)
```

Which features are categorical (qualitative)

- Nominal feature ：PassengerId, Survived, Name, Sex, Ticket, Carbin, Embarked
- Ordinal features：Pclass

Which features are numerical (quantitative)?

- continuous features：Age, Fare
- discrete features：SibSp, Parch

Which features are mixed types

- Ticket：Both pure numbers and characters and numbers
- Cabin is character and numeric

In [76]:

```
train_df.head()
```

Which features have non-standard values

- Name：eg.Feutrelle, Mrs. Jacques Heath (Lily May Peel)

In [77]:

```
train_df.tail()
```

which features have missing values

- Train：Cabin、Age、Embarked
- Test：Cabin、Age、Fare

In [78]:

```
train_df.isnull().sum()
```

In [79]:

```
test_df.isnull().sum()
```

What is the data type of each feature

- Seven characteristics are integer or float
- The five features are strings

In [80]:

```
train_df.info()
print('_'*40)
test_df.info()
```

What is the empirical distribution of numerical features?

The empirical distribution of the data set helps us make preliminary observations on the data set to determine whether the empirical distribution represents the true distribution

In [81]:

```
train_df.describe(percentiles=[.61, .62, .68, .69, .75, .8, .99])
```

From the perspective of percentiles, the distribution of `Fare` is very unbalanced. Let's take a look at its boxplot and column diagram.

In [82]:

```
plt.figure(figsize=[10, 8])
plt.subplot(221)
sns.boxplot(data=train_df['Fare'])
plt.ylabel('Fare')
plt.subplot(222)
sns.histplot(train_df['Fare'])
plt.xlabel('Fare')
```

Also plot the distribution of `Age`

In [83]:

```
plt.figure(figsize=[10, 8])
plt.subplot(221)
sns.boxplot(data=train_df['Age'])
plt.ylabel('Age')
plt.subplot(222)
sns.histplot(train_df['Age'])
plt.xlabel('Age')
```

As we can be seen, there is nothing special about the number of babies

Some conclusions about numerical features

- **The data set contains 891 samples. In the background of the question, it is said that there are 2224 individuals in Titanic. The data set only contains about 40% of the real data.**
- **The survival rate of the samples in the data set is 38%, and the real survival rate is 32% in the question background**
- **Over 75% of the samples were not accompanied by parents or children**
- **Nearly 30% of the sample is accompanied by siblings or spouses**
- **Uneven distribution of fares, as high as $512 for <1% sample**
- **The proportion of the elderly (65-80 years old) is less than 1%**

Classification feature distribution

In [84]:

```
train_df.describe(include='O')
```

In [85]:

```
plt.figure(figsize=[15, 8])
plt.subplot(231)
sns.countplot(x='Embarked', data=train_df)
plt.subplot(232)
sns.countplot(x='Pclass', data=train_df)
plt.subplot(233)
sns.countplot(x='Sex', data=train_df)
```

Some conclusions about classification characteristics

- **Names have no duplicate values, no one has the same name**
- **577/891 (65%) were male**
- **Cabin has many duplicate values, and many passengers share cabins**
- **Embarked has only 3 values, and port S has the most people on board**
- **Tickets have many duplicate values, and many passengers share tickets?**

3.3 Analyze the relationship between features and labels

3.3.1 Analysis of classification features by groupby

The previous conjectures about the classification characteristics are as follows:

- `Pclass` **: The ticket type represents the income level of the passengers, we believe that the higher income group is more likely to survive**
- `Sex` **: We think female groups are more likely to survive**
- `Embarked` **: We think the port of embarkation may be related to survival**
- `family` **: separate groupby datasets according to** `SibSp` **and** `Parch`

Pclass

In [86]:

```
train_df[['Pclass', 'Survived']].groupby(['Pclass'], as_index=False).mean().sort_values(
by='Survived', ascending=False)
```

It is found that the group with Pclass=1 does have a higher survival rate, which confirms our previous hypothesis

Sex

In [87]:

```
train_df[['Sex', 'Survived']].groupby(['Sex'], as_index=False).mean().sort_values(by='Su
rvived', ascending=False)
```

It was found that the survival rate of the female group was as high as 74%, and the survival rate of the male group was less than 19%, which verified the previous hypothesis

Embarked

In [88]:

```
train_df[['Embarked', 'Survived']].groupby(['Embarked'], as_index=False).mean().sort_val
ues(by='Survived', ascending=False)
```

Passengers boarding at port C had a higher survival rate, validating previous assumptions

SibSp

In [89]:

```
train_df[['SibSp', 'Survived']].groupby(['SibSp'], as_index=False).mean().sort_values(by
='Survived', ascending=False)
```

Parch

In [90]:

```
train_df[['Parch', 'Survived']].groupby(['Parch'], as_index=False).mean().sort_values(by
='Survived', ascending=False)
```

Neither of these two characteristics alone are directly related to the survival rate.

3.3.2 Exploring Numerical Features with Bar Charts

Previous conjectures for numerical features are:

- `Age`: Age is closely related to birth

First explore the relationship between age `Age` and `Survived`

Age

In [91]:

```
g = sns.FacetGrid(train_df, col='Survived')
g.map(plt.hist, 'Age', bins=20)
```

in conclusion

1. The infant group (Age <= 4) has a higher survival rate
2. The oldest passenger (Age = 80) survived
3. A large number of young and middle-aged passengers (15 <= Age <= 25) died
4. Passengers aged 15-35 account for the most

Conclusion about age

Based on the above four observations, we confirm the conjecture that age is closely related to birth, and several previous ideas about age are reasonable

- The model needs to take into account the `Age` feature
- Fill in missing values of `Age` feature
- binning by age

3.3.3 Combine multiple feature analysis using bar chart

We combine `Pclass` and `Age` to consider the relationship with survival, and also explore the age distribution of people at different income levels

Pclass and Age

In [92]:

```
grid = sns.FacetGrid(train_df, col='Survived', row='Pclass', height=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=0.5, bins=20)
grid.add_legend()
```

Observed

- The passenger fatality rate of `Pclass=3` is the highest, and the passenger fatality rate of `Pclass=1` is the lowest, and the difference is quite obvious, once again verifying the previous conjecture that `Pclass` is related to the survival rate
- Most babies with `Pclass=2 or 3` survived (the number of babies with `Pclass=1` is too small to be considered), age binning does help analysis
- The age distribution of different passenger groups in `Pclass` is different, among which passengers with `Pclass=3` are particularly young and middle-aged

Conclusions on passenger income levels

The survival rate of passenger groups with different income levels varies greatly, especially the survival rate of passengers with low income levels is the lowest. Our model needs to consider the `Pclass` feature

3.3.4 Combine multiple feature analysis with line chart

We combined the boarding port `Embarked`, the ticket type `Pclass`, and the gender `Age` to examine the survival rate

Embarked、Pclass ， Age

In [93]:

```
grid = sns.FacetGrid(train_df, row='Embarked', height=2.2, aspect=1.6)
grid.map(sns.pointplot, 'Pclass', 'Survived', 'Sex', palette='deep')
grid.add_legend()
```

Observed

- **Women who board ships at ports S and Q generally have a higher survival rate than men, but men who board ships at port C have a higher survival rate.**
- **Low-income men boarding ships at port Q have a higher survival rate than high-income men**

in conclusion

The previous belief that women generally have a higher survival rate than men is not entirely true, nor is it entirely true that higher-income groups generally have higher survival rates than lower-income groups

The survival rate is also affected by the port of boarding, and different ports of boarding have a great impact on these two trends

- **Missing values for the `Embarked` feature need to be filled and the model needs to take this feature into account**

We combined `Embarked` embarkation port, `Sex` gender, `Fare` fare to analyze their association with survival rate

In [94]:

```
grid = sns.FacetGrid(train_df, row='Embarked', col='Survived', height=2.2, aspect=1.6)
grid.map(sns.barplot, 'Sex', 'Fare', alpha=0.5, ci=None)
grid.add_legend()
```

Observed

- **Passengers with higher fares have higher survival rates**
- **Passenger fares for boarding at port Q are generally lower, and the survival rate does not seem to be related to the fare**

Conclusion on fares

- **The model needs to take into account the `Fare` feature**
- **Need to bin fares**

4. Data preprocessing

In the EDA stage, we have conducted a detailed inspection of the entire data set, explored the relationship between each feature and the relationship with the predicted target (survival), put forward many preliminary

between each feature and the relationship with the predicted target (survival), put forward many preliminary ideas and conjectures, and made simple verification

In the data preprocessing phase, we will process the dataset following previously validated ideas and conjectures. At this stage, we will continue to propose other conjectures, verify and process

4.1 Deleting features

According to the previous idea, we need to remove the features `Cabin`, `Ticket`, `Name` and `PassengerId`

Before deleting, we can first verify that these features are not related to whether the passenger survives or not, and then delete

In order to omit the space here, only verify that it is not related to survival before deleting `Name`, and delete `Cabin` and `Ticket` directly

Passenger ID number `PassengerId` needs to be removed in the training set and kept in the test set (for submission)

Remove `Cabin` and `Ticket`

In [95]:

```
train_df = train_df.drop(columns=['Ticket', 'Cabin'])
test_df = test_df.drop(['Ticket', 'Cabin'], axis=1)
```

Remove `PassengerId` in training set

In [96]:

```
train_df = train_df.drop(columns=['PassengerId'])
combine = [train_df, test_df]
```

4.2 Constructing new features

Note that the `Name` feature contains the title of the sample, which to some extent reflects the status of the sample

Although the name itself is not directly related to the survival rate, the status may be related to the survival. Therefore, before deleting the `Name` feature, we can first extract the identity status of the sample, construct a new feature `Title`, and then delete `Name` feature

Extract title information from `Name` `Title`

The title usually has a period. It is separated from the name, such as Mr. Mrs., etc., you can build a regular expression based on this feature to extract the title of the sample

It doesn't matter if you don't know regular expressions, you can use the `apply()` method and the `split()` method to separate the fields by periods. Then extract the part before the period.

In [97]:

```
for dataset in combine:
    dataset['Title'] = dataset['Name'].str.extract('([A-Za-z]+)\.', expand=False)
```

Many titles and genders are directly related, which we can observe through the cross table of `Title` and `Sex`

In [98]:

```
pd.crosstab(train_df['Title'], train_df['Sex'])
```

Next, we replace the title with the same meaning with the same title, and the title with only a small number of samples is collectively called `Rare`

In [99]:

```
for dataset in combine:
    dataset['Title'] = dataset['Title'].replace(['Lady', 'Countess', 'Capt', 'Col', 'Don
',
                                                  'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer
',
                                                 'Dona'], 'Rare')
    dataset['Title'] = dataset['Title'].replace('Mlle', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Ms', 'Miss')
    dataset['Title'] = dataset['Title'].replace('Mme', 'Mrs')
```

Analysis of the relationship between appellation and survival by groupby

In [100]:

```
train_df[['Title', 'Survived']].groupby(['Title'], as_index=False).mean()
```

It is found that there is a connection between the title and the survival

- **Feminine titles have a higher survival rate and male titles have a lower survival rate**
- **Other appellations had a lower survival rate of 34%**
- **Master's survival rate is about 57%**

Remove `Name` feature

In [101]:

```
train_df = train_df.drop(['Name'], axis=1)
test_df = test_df.drop(columns=['Name'])
combine = [train_df, test_df]
train_df.head()
```

Construct `FamilySize` and `IsAlone` features

Combining `Parch` and `SibSp` to construct the `FamilySize` feature

Distinguish singleness by `FamilySize`

In [102]:

```
for dataset in combine:
    dataset['FamilySize'] = dataset['SibSp'] + dataset['Pa rch'] + 1
```

In [103]:

```
train_df[['FamilySize', 'Survived']].groupby(['FamilySize'], as_index=False).mean().sort
_values(by='Survived', ascending=False)
```

Observed

- **People with large family sizes (FamilySize >= 5) have lower survival rates**
- **People with moderate family sizes (FamilySize = 2, 3, 4) have a higher survival rate**
- **Single people (FamilySize = 1) also have a low survival rate**

In [104]:

```
for dataset in combine:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1
train_df[['IsAlone', 'Survived']].groupby(['IsAlone'], as_index=False).mean()
```

The survival rate of single passengers is relatively low, and the survival rate of non-single passengers reaches 50%

Based on the above observations, we choose to keep the `IsAlone` and `FamilySize` features and delete the two original features

In [105]:

```
train_df = train_df.drop(['Parch', 'SibSp'], axis=1)
test_df = test_df.drop(['Parch', 'SibSp'], axis=1)
combine = [train_df, test_df]
train_df.head()
```

4.3 Converting data types

For subsequent input to the model for prediction, you need to convert the string type field to a numeric type

Use get_dummies() to change `Title` to one-hot encoding

In [106]:

```
train_df = train_df.join(pd.get_dummies(train_df['Title']))
train_df = train_df.drop(columns=['Title'])
test_df = test_df.join(pd.get_dummies(test_df['Title']))
test_df = test_df.drop(columns=['Title'])
combine = [train_df, test_df]
```

In [107]:

```
train_df
```

Convert the `Sex` feature to a numeric type, using 0 for males and 1 for females

In [108]:

```
for dataset in combine:
    dataset['Sex'] = dataset['Sex'].map({
        'female': 1,
        'male': 0,
    }).astype(int)
```

Convert the `Embarked` feature to a numeric type. Since this field has missing values, you need to fill in the missing values before converting the data type

4.4 Missing value handling

Populate Typed Features- Embarked field

Because there are only two missing values in this field, it is possible to take a simple filling method or simply delete the two missing data.

Here we take the method of mode filling

In [109]:

```
freq_port = train_df['Embarked'].dropna().mode()[0]
print("Port:{}".format(freq_port))
for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].fillna(freq_port)
```

Convert `Embarked` to one-hot encoding after filling

In [110]:

```
train_df = train_df.join(pd.get_dummies(train_df['Embarked']))
train_df = train_df.drop(columns=['Embarked'])
test_df = test_df.join(pd.get_dummies(test_df['Embarked']))
test_df = test_df.drop(columns=['Embarked'])
combine = [train_df, test_df]
```

In [111]:

```
train_df.head()
```

Filling Numeric Features - `Age` Field

Fields with missing values that need to be processed include `Age` and `Embarked`, first fill in missing values for the `Age` field

Simple filling methods can fill in median, mean, random numbers, etc. More precise methods are based on other related features

Here we fill in missing values for `Age` based on `Pclass` and `Gender` (more advanced ones have missing value filling based on other machine learning algorithms)

There are three kinds of `Pclass` and two kinds of `Gender`. There are six cases in their combination. We fill in `Age` according to the median of these six groups.

In [112]:

```
grid = sns.FacetGrid(train_df, row='Pclass', col='Sex', height=2.2, aspect=1.6)
grid.map(plt.hist, 'Age', alpha=.5, bins=20)
grid.add_legend()
```

In [113]:

```
guess_ages = np.zeros((2, 3))
for dataset in combine:
    for i in range(2):
        for j in range(3):
            age_guess = dataset[(dataset['Sex'] == i) & (dataset['Pclass'] == j+1)]['Age
'].dropna().median()
            guess_ages[i, j] = round(age_guess)

    # 填充
    for i in range(2):
        for j in range(3):
            dataset.loc[(dataset['Age'].isnull()) & (dataset['Sex'] == i) &
                        (dataset['Pclass'] == j+1), 'Age'] = guess_ages[i, j]

    dataset['Age'] = dataset['Age'].astype(int)
```

In [114]:

```
train_df.head()
```

Populate Fare field in test set

Because there is only one missing value in the Fare field in the test set, a simple median filling can be used.

In [115]:

```
test_df['Fare'].fillna(test_df['Fare'].dropna().median(), inplace=True)
```

4.5 Other

4.5 Other

Populate Fare field in test set

Because there is only one missing value in the Fare field in the test set, a simple median filling can be used.

In [116]:

```
train_df['AgeBand'] = pd.cut(train_df['Age'], 5)
train_df[['AgeBand', 'Survived']].groupby(['AgeBand'], as_index=False).mean().sort_value
s(by='AgeBand', ascending=True)
```

Convert the numeric attribute `Age` into an ordinal attribute according to the result of binning, delete the `AgeBand` field

In [117]:

```
for dataset in combine:
    dataset.loc[dataset['Age'] <= 16, 'Age'] = 0
    dataset.loc[(dataset['Age'] <= 32) & (dataset['Age'] > 16), 'Age'] = 1
    dataset.loc[(dataset['Age'] <= 48) & (dataset['Age'] > 32), 'Age'] = 2
    dataset.loc[(dataset['Age'] <= 64) & (dataset['Age'] > 48), 'Age'] = 3
    dataset.loc[dataset['Age'] > 64, 'Age'] = 4
train_df = train_df.drop(['AgeBand'], axis=1)
combine = [train_df, test_df]
train_df.head()
```

In [118]:

```
train_df['FareBand'] = pd.qcut(train_df['Fare'], 4)
train_df[['FareBand', 'Survived']].groupby(['FareBand'], as_index=False).mean().sort_val
ues(by='FareBand', ascending=True)
```

In [119]:

```
for dataset in combine:
    dataset.loc[ dataset['Fare'] <= 7.91, 'Fare'] = 0
    dataset.loc[(dataset['Fare'] > 7.91) & (dataset['Fare'] <= 14.454), 'Fare'] = 1
    dataset.loc[(dataset['Fare'] > 14.454) & (dataset['Fare'] <= 31), 'Fare']   = 2
    dataset.loc[ dataset['Fare'] > 31, 'Fare'] = 3
    dataset['Fare'] = dataset['Fare'].astype(int)

train_df = train_df.drop(['FareBand'], axis=1)
combine = [train_df, test_df]
```

In [120]:

```
train_df.head()
```

5. Build the model and predict the solution

Re-clarify the data mining task: we want to predict whether the passenger survives or not based on the passenger's information

This is a classification/regression task. Models can be trained to make predictions using supervised machine learning algorithms

Models suitable for this task include, but are not limited to:

-Logistic Regression

- **Support Vector Machines**
- **k-nearest neighbors**
- **Naive Bayes Classifier**
- **Decision tree**
- **Random Forest**

- Perceptron
- Neural Networks
- Kmeans

First, without adjusting the hyperparameters, directly use the default parameters for each algorithm to fit the entire training set, and calculate the accuracy on the training set. Then, the generalization ability of the model is estimated by 10-fold cross-validation,

First intuitively feel the difference between the tree model and other simple models in the ability to fit the data set

In [121]:

```
X_train = train_df.drop('Survived', axis=1)
Y_train = train_df['Survived']
X_test = test_df.drop('PassengerId', axis=1).copy()
X_train.shape, Y_train.shape, X_test.shape
```

5.1 Logistic Regression

In [122]:

```
LR = LogisticRegression()
LR.fit(X_train, Y_train)
Y_pred_LR = LR.predict(X_test)
acc_LR = round(LR.score(X_train, Y_train) * 100, 2)
print("SCORE:{}%".format(acc_LR))

acc_LR_cv = round(cross_val_score(LR, X_train, Y_train, cv=10, scoring='accuracy').mean(
) * 100, 2)
print("ACC:{}%".format(acc_LR_cv))
```

Because LR is highly interpretable, the weight of each feature represents the contribution to the prediction target

It is usually recommended to use Logistic Regression when building the baseline. The importance of the feature can be judged by observing the weight of each feature.

At the same time, it can also verify our previous ideas and conjectures, and can also evaluate whether the new features we build are really helpful for predicting the target

In [123]:

```
coeff_df = pd.DataFrame(train_df.columns.delete(0))
coeff_df.columns = ['Feature']
coeff_df['Correlation'] = pd.Series(LR.coef_[0])
coeff_df.sort_values(by='Correlation', ascending=False)
```

Support Vector Machines

In [124]:

```
svc = SVC()
svc.fit(X_train, Y_train)
Y_pred_svc = svc.predict(X_test)
acc_svc = round(svc.score(X_train, Y_train) * 100, 2)
print("SCORE%".format(acc_svc))

acc_svc_cv = round(cross_val_score(svc, X_train, Y_train, cv=10, scoring='accuracy').mea
n() * 100, 2)
print("ACC:{}%".format(acc_svc_cv))
```

KNN

In [125]:

```
knn = KNeighborsClassifier(n_neighbors = 9)
knn.fit(X_train, Y_train)
Y_pred_knn = knn.predict(X_test)
acc_knn = round(knn.score(X_train, Y_train) * 100, 2)
print("SCORE：{}%".format(acc_knn))
```

Random Forest

In [126]:

```
random_forest = RandomForestClassifier()
random_forest.fit(X_train, Y_train)
Y_pred_random_forest = random_forest.predict(X_test)
random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_train, Y_train) * 100, 2)
print("SCORE：{}%".format(acc_random_forest))

acc_random_forest_cv = round(cross_val_score(random_forest, X_train, Y_train, cv=10, sco
ring='accuracy').mean() * 100, 2)
print("ACC：{}%".format(acc_random_forest_cv))
```

KMEANS

In [127]:

```
from sklearn.cluster import KMeans
alg = KMeans(n_clusters=2, random_state=0)
alg.fit(X_train)
predict = alg.predict(X_train)
correct = 0
for i in range (len(Y_train)):
    if predict[i] == Y_train[i] :
        correct += 1

print(correct / len(Y_train))
```

In [129]:

```
from sklearn.cluster import KMeans
alg = KMeans(n_clusters=5, random_state=0)
alg.fit(X_train)
predict = alg.predict(X_train)
correct = 0
for i in range (len(Y_train)):
    if predict[i] == Y_train[i] :
        correct += 1

print(correct / len(Y_train))
```

In the KNN algorithm, changing the parameter n_neighbors to 5 has a slightly higher accuracy. But in fact, it is not suitable to use the K-Means algorithm in this data set, because this is a clustering algorithm, and clustering is an unsupervised algorithm. In this Titanic data set, lable has been given, so this algorithm may not be suitable, but it can be used.

After that, I also want to try to integrate multiple models, whether it is an unsupervised model or a supervised model, I think this will make the final accuracy higher