1. Research Background and Objective

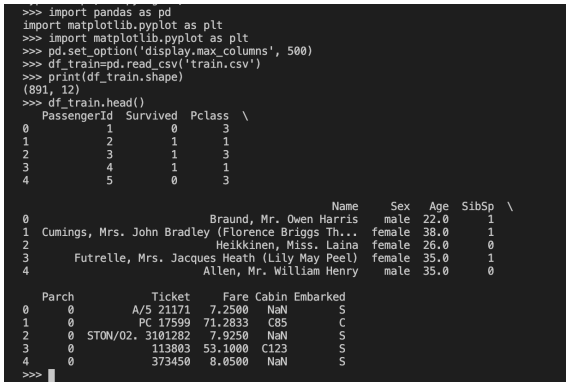
The sinking of the Titanic is one of the most infamous shipwrecks in history. On April 15, 1912, during her maiden voyage, the widely considered “unsinkable” RMS Titanic sank after colliding with an iceberg. Unfortunately, there weren’t enough lifeboats for everyone onboard, resulting in the death of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

This research project is aimed to use machine learning to establish a model that predicts which passengers survived the Titanic shipwreck based on the passengers’ information, ie. name, age, gender, socio-economic class, etc. Several stages are applied in the studying process: define the research goal, data cleaning, data analysis and feature selection, modelling, model estimation and final results.

2. Data Enquiring and Data Cleaning

There are 3 datasets can be downloaded from Kaggle, train.csv, test.csv and gender_submission.csv, which are referred to the training dataset, testing dataset and a sample final submission file.

Training dataset can be loaded for further use using Pandas library. There are 891 sample observations and 12 features and Survived is the target variable and all other variables are the information of passengers.



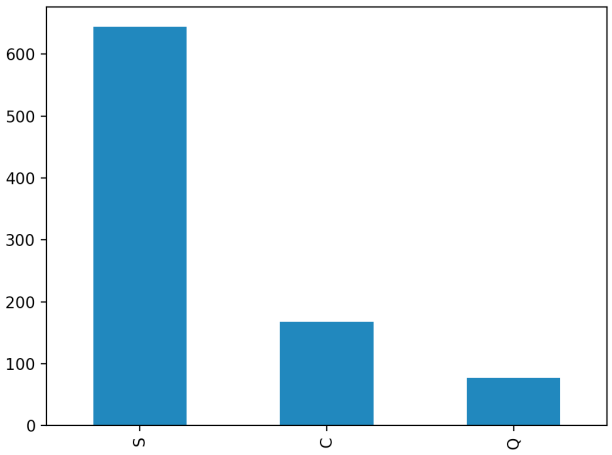
```
>>> import pandas as pd
import matplotlib.pyplot as plt
>>> import matplotlib.pyplot as plt
>>> pd.set_option('display.max_columns', 500)
>>> df_train=pd.read_csv('train.csv')
>>> print(df_train.shape)
(891, 12)
>>> df_train.head()
   PassengerId  Survived  Pclass  \
0            1         0       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
>>>
```

Missing data processing:

The total number of missing values for each column can be calculated, there are 177 missing values for Age, 687 missing values for Cabin and 2 missing values for Embarked. The number of median age can be used to fill in the missing ages.

```
>>> df_train.isnull().sum()
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
>>>
```

Since there are only 2 missing values for Embarked, one way is that this two observation can be simply removed from the dataset, another way is that using the most frequent Embarked value to replace the NA values. Since 'S' has the most frequent value, 'S' is used to replace the original NA values in 'Embarked' column.



Categorical variables are converted into dummy/indicator variables to make it more applicable during the final modelling process. After converting, the original Embarked variable is transferring into Embarked_Q, Embarked_C and Embarked_S as showing.

```
>>> embarked_dummies = pd.get_dummies(df_train['Embarked'], prefix='Embarked')
>>> df_train = pd.concat([df_train, embarked_dummies], axis=1)
>>> df_train.drop('Embarked', axis=1, inplace=True)
>>> print(df_train.columns)
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked_C', 'Embarked_Q',
       'Embarked_S'],
      dtype='object')
>>>
```

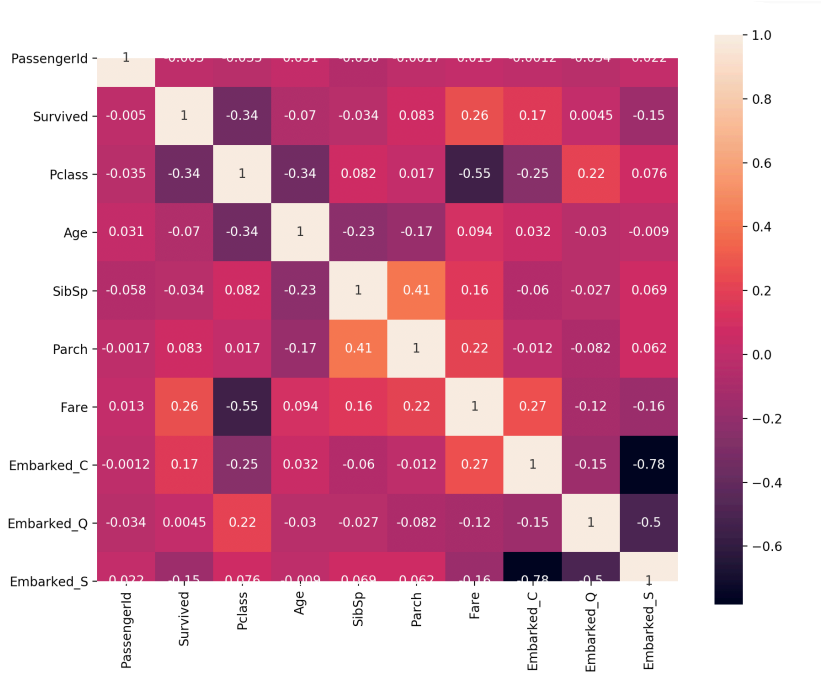
There are lots of missing values for Cabin, 'Missing' label is used to mark it as missing values during the process:

```
>>> df_train['Cabin'].fillna('Missing', inplace=True)
>>> df_train['Cabin'].unique()
array(['Missing', 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
       'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'C83', 'F33', 'F G73',
       'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101', 'F E69',
       'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4', 'A32',
       'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35', 'C87',
       'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19', 'B49',
       'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54', 'B57 B59 B63 B66',
       'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40', 'T', 'C128',
       'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44', 'A34', 'C104',
       'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14', 'B37', 'C30',
       'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38', 'B39', 'B22',
       'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68', 'B41', 'A20',
       'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48', 'E58', 'C126',
       'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63', 'C62 C64',
       'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30', 'E121',
       'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36', 'B102',
       'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42', 'C148'],
      dtype=object)
>>>
```

Similarly, the values of Cabin will be kept with leading categorical letter and removing the following numbers since categorical values are more applicable for logistic regression modelling. Here ‘M’ stands for missing value, if there is an M category existing before, then other letter should be used to avoid mixing.

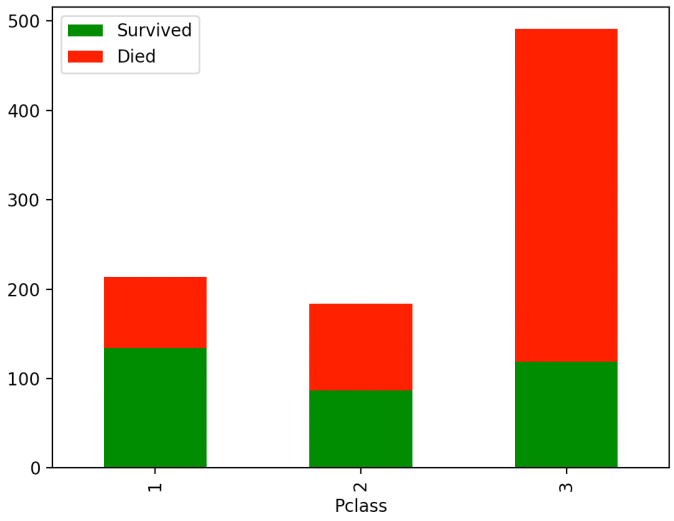
```
>>> df_train['Cabin'].unique()
array(['M', 'C', 'E', 'G', 'D', 'A', 'B', 'F', 'T'], dtype=object)
>>>
```

3. Feature Selection and Engineering

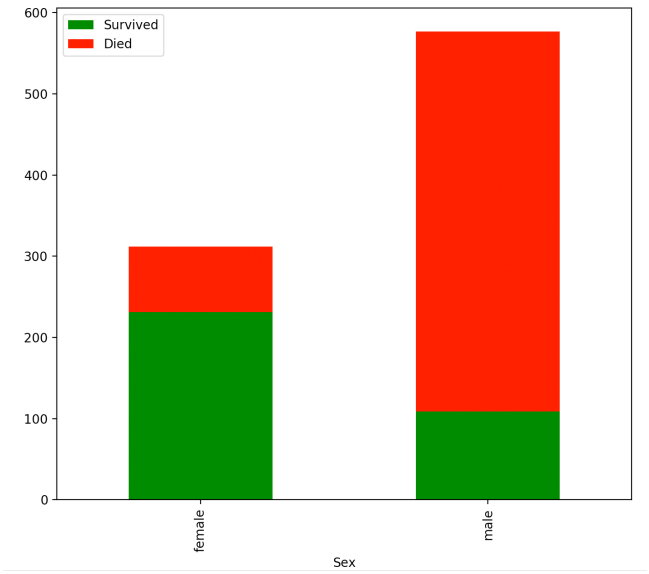


A quick review of correlations between each two variables are as snapshot showing:

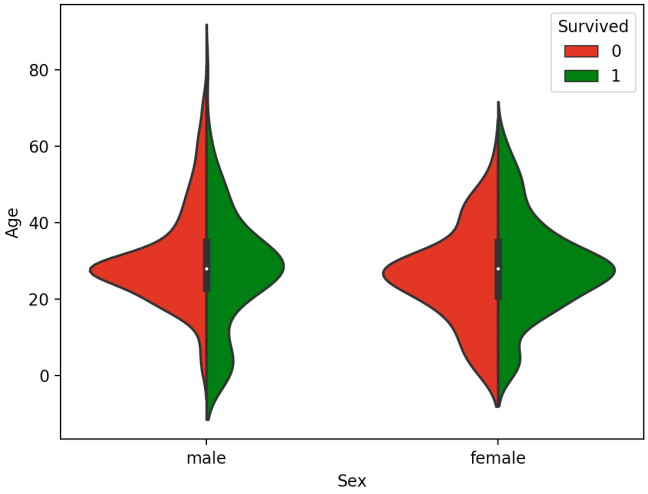
The proportion of survived and died is differ in different Pclass category. The survival probability is much higher than other two categories when Pclass is 1.



Similarly, female passengers have a higher probability to be survived during the sinking of Titanic event.

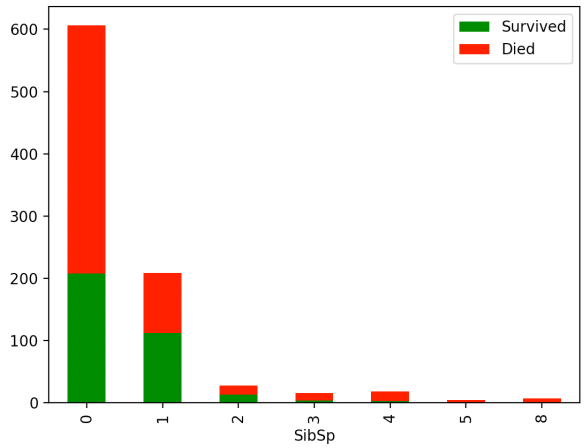


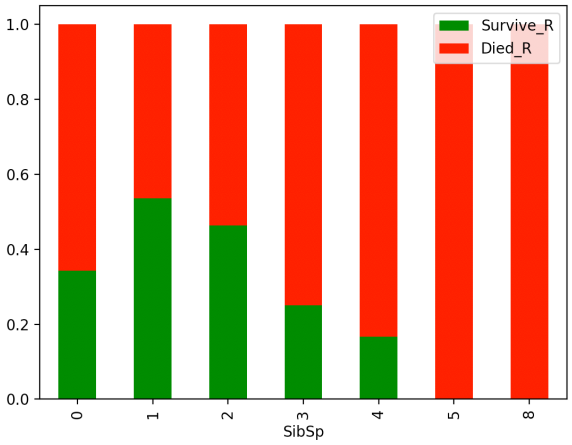
By plotting Age and Sex in a violinplot, the trend is that younger passages are more likely to survive than elder people and female passengers are having a higher probability to be saved than male.



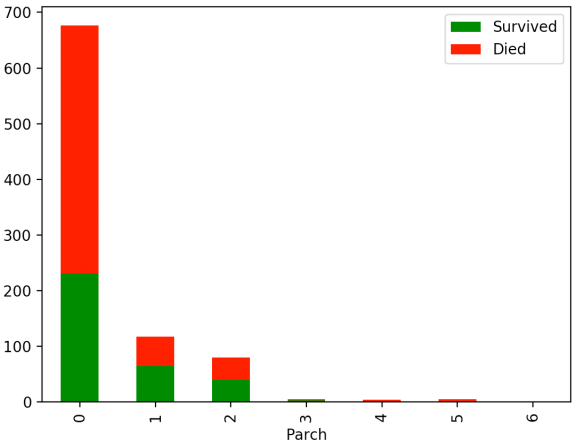
Regarding the feature SibSp, different number of SibSp abroad Titanic have different survival probability from the left plot and the probability can not be observed when the SibSp number greater than 3.

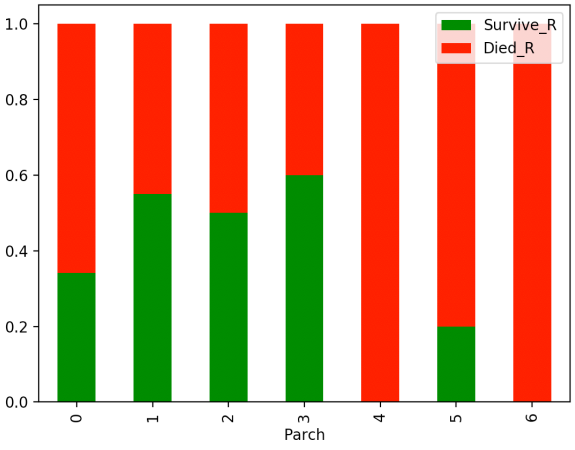
Therefore, the survival rate and died rate can be transferred from the origin number of SibSp as shown in the right plot. The survival rate is increasing to the peak from 0 SibSp to 1, then it's in deceasing trend, and the survival rate is almost zero when the SibSp number is 5 or more.



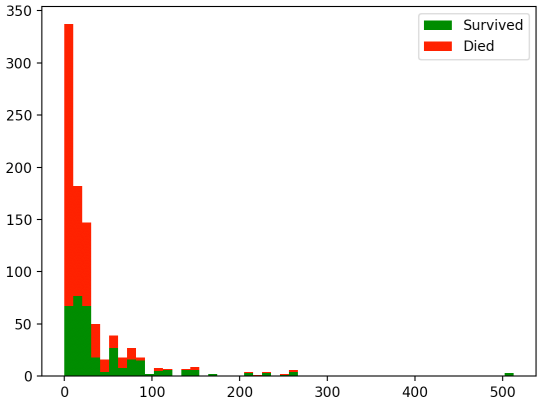


The similar process can be applied when studying the feature Parch, which refers to parent and children aboard Titanic. When transferring the origin Parch number into proportion, the survival rate is biggest when the Parch number is 3, then is 1, 2 and 1 in sequence, and the survival rate is almost 0 when Parch number is 4 or 6.

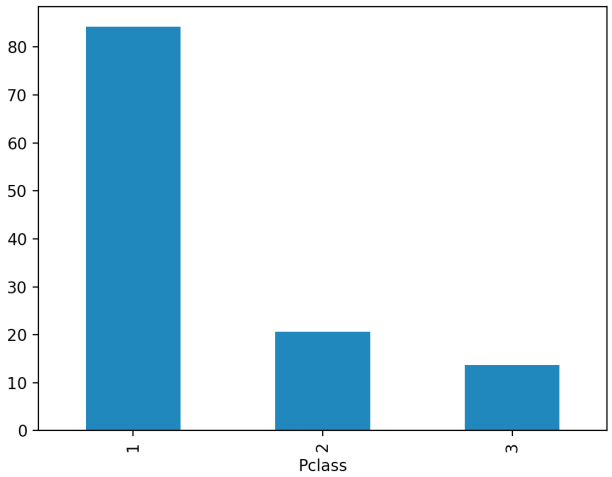




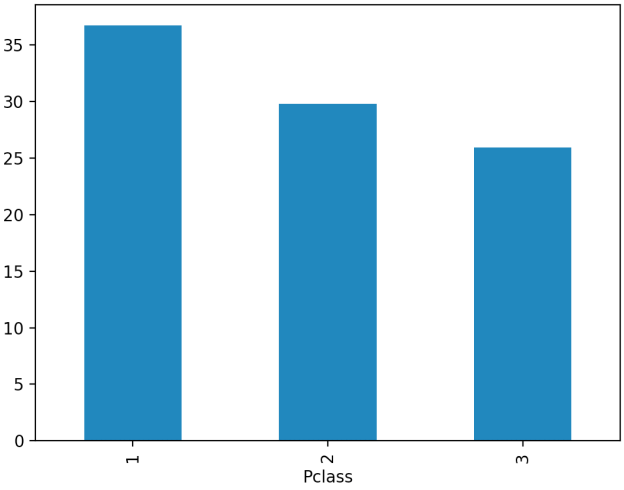
Regarding the ticket fare, it seems that the more expensive the ticket is, the higher survival rate for the passengers.



The mean value of ticket fare regarding different kinds of Pclass is plotted as showing snapshot, and Pclass 1 has the highest ticket fare.



The mean value of age regarding different types of Pclass is plotted as right snapshot, and Pclass 1 have the highest mean age value.



4. Modelling process

Load the test dataset and remove the survived column since it’s the final target.

```
>>> df_test = pd.read_csv('test.csv')
>>>
>>> targets = df_train['Survived']
>>> df_train.drop(['Survived'], 1, inplace=True)
>>>
>>> print(df_train.columns)
Index(['PassengerId', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch',
       'Ticket', 'Fare', 'Cabin', 'Embarked_C', 'Embarked_Q', 'Embarked_S',
       'Died'],
      dtype='object')
>>>
```

The test dataset can be merged with train dataset first and then do the data engineering process together.

```
>>> combined = df_train.append(df_test)
>>> combined.reset_index(inplace=True)
>>> combined.drop(['index', 'PassengerId'], inplace=True, axis=1)
>>> print(combined.columns)
Index(['Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare',
       'Cabin', 'Embarked_C', 'Embarked_Q', 'Embarked_S', 'Died', 'Embarked'],
      dtype='object')
>>>
```

```
>>> print(combined.shape)
(1307, 14)
>>> combined.head()
   Pclass                                               Name     Sex   Age  \
0       3                            Braund, Mr. Owen Harris    male  22.0
1       1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0
2       3                             Heikkinen, Miss. Laina  female  26.0
3       1       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0
4       3                           Allen, Mr. William Henry    male  35.0

   SibSp  Parch            Ticket     Fare Cabin  Embarked_C  Embarked_Q  \
0      1      0         A/5 21171   7.2500     M         0.0         0.0
1      1      0          PC 17599  71.2833     C         1.0         0.0
2      0      0  STON/O2. 3101282   7.9250     M         0.0         0.0
3      1      0            113803  53.1000     C         0.0         0.0
4      0      0            373450   8.0500     M         0.0         0.0

   Embarked_S  Died Embarked
0         1.0   1.0      NaN
1         0.0   0.0      NaN
2         1.0   0.0      NaN
3         1.0   0.0      NaN
4         1.0   1.0      NaN
>>>
```

Different passengers have different titles which can be observed in the name column from the dataset. Title info can be abstracted by splitting name column.

```
>>> titles = set()
>>> for name in combined['Name']:
...     titles.add(name.split(',')[1].split('.')[0].strip())
...
>>> print(titles)
{'Jonkheer', 'Dona', 'Ms', 'Don', 'Miss', 'Col', 'Mme', 'Sir', 'the Countess', 'Major', 'Capt', 'Dr', 'Mrs', 'Lady', 'Mlle', 'Mr', 'Master', 'Rev'}
>>>
```

Then a title dictionary can be defined and bin passengers with different titles into different categories.

```
>>> Title_Dictionary = {
...     "Capt": "Officer",
...     "Col": "Officer",
...     "Major": "Officer",
...     "Jonkheer": "Royalty",
...     "Don": "Royalty",
...     "Dona": "Royalty",
...     "Sir" : "Royalty",
...     "Dr": "Officer",
...     "Rev": "Officer",
...     "the Countess":"Royalty",
...     "Mme": "Mrs",
...     "Mlle": "Miss",
...     "Ms": "Mrs",
...     "Mr" : "Mr",
...     "Mrs" : "Mrs",
...     "Miss" : "Miss",
...     "Master" : "Master",
...     "Lady" : "Royalty"
... }
```

```
>>> combined['Title'] = combined['Name'].map(lambda name:name.split(',')[1].split('.')[0].strip())
>>>
>>> combined['Title'].unique()
array(['Mr', 'Mrs', 'Miss', 'Master', 'Don', 'Rev', 'Dr', 'Mme', 'Ms',
       'Major', 'Lady', 'Sir', 'Mlle', 'Col', 'Capt', 'the Countess',
       'Jonkheer', 'Dona'], dtype=object)
>>> combined['Title'] = combined['Title'].map(Title_Dictionary)
>>>
>>> combined['Title'].unique()
array(['Mr', 'Mrs', 'Miss', 'Master', 'Royalty', 'Officer'], dtype=object)
>>>
```

Now there is no missing value in Title column by checking isnull() function.

```
>>> combined[combined['Title'].isnull()]
Empty DataFrame
Columns: [Pclass, Name, Sex, Age, SibSp, Parch, Ticket, Fare, Cabin, Embarked_C, Embarked_Q, Embarked_S, Died, Embarked, Title]
Index: []
>>>
```

There are 86 missing age in test dataset by checking missing values in train dataset and test dataset separately.

```
>>> print(combined.iloc[:891]['Age'].isnull().sum())
0
>>> print(combined.iloc[891:]['Age'].isnull().sum())
86
>>>
```

Extract the first 891 rows which are from train dataset and group then by Sex, Pclass and Title, then get the median value for each group.

```
>>> grouped_train = combined.iloc[:891].groupby(['Sex','Pclass','Title'])
>>> grouped_median_train = grouped_train.median()
>>> grouped_median_train = grouped_median_train.reset_index()[['Sex', 'Pclass', 'Title', 'Age']]
>>>
>>> print(grouped_median_train)
      Sex  Pclass    Title   Age
0   female       1     Miss  29.0
1   female       1      Mrs  37.0
2   female       1  Officer  49.0
3   female       1  Royalty  40.5
4   female       2     Miss  24.0
5   female       2      Mrs  31.5
6   female       3     Miss  22.0
7   female       3      Mrs  29.0
8     male       1   Master   4.0
9     male       1       Mr  36.0
10    male       1  Officer  50.0
11    male       1  Royalty  40.0
12    male       2   Master   1.0
13    male       2       Mr  30.0
14    male       2  Officer  46.5
15    male       3   Master   6.5
16    male       3       Mr  28.0
>>>
```

Define a function through which return the median function for each Sex, Pclass and Title group category. Then using lambda function to fill in number returned by function when the values is not a number.

```
>>> def fill_age(row):
...     condition = (
...         (grouped_median_train['Sex'] == row['Sex']) &
...         (grouped_median_train['Title'] == row['Title']) &
...         (grouped_median_train['Pclass'] == row['Pclass'])
...     )
...     return grouped_median_train[condition]['Age'].values[0]
...
>>> combined['Age'] = combined.apply(lambda row: fill_age(row) if np.isnan(row['Age']) else row['Age'], axis=1)
>>>
>>> combined['Age'].unique()
array([22.  , 38.  , 26.  , 35.  , 28.  , 54.  ,  2.  , 27.  , 14.  ,
        4.  , 58.  , 20.  , 39.  , 55.  , 31.  , 34.  , 15.  ,  8.  ,
       19.  , 40.  , 66.  , 42.  , 21.  , 18.  ,  3.  ,  7.  , 49.  ,
       29.  , 65.  , 28.5 ,  5.  , 11.  , 45.  , 17.  , 32.  , 16.  ,
       25.  ,  0.83, 30.  , 33.  , 23.  , 24.  , 46.  , 59.  , 71.  ,
       37.  , 47.  , 14.5 , 70.5 , 32.5 , 12.  ,  9.  , 36.5 , 51.  ,
       55.5 , 40.5 , 44.  ,  1.  , 61.  , 56.  , 50.  , 36.  , 45.5 ,
       20.5 , 62.  , 41.  , 52.  , 63.  , 23.5 ,  0.92, 43.  , 60.  ,
       10.  , 64.  , 13.  , 48.  ,  0.75, 53.  , 57.  , 80.  , 70.  ,
       24.5 ,  6.  ,  0.67, 30.5 ,  0.42, 34.5 , 74.  , 22.5 , 18.5 ,
       31.5 , 67.  , 76.  , 26.5 , 60.5 , 11.5 ,  0.33,  6.5 ,  0.17,
       38.5 ])
>>> print(combined['Age'].isnull().sum())
0
>>>
```

Then the name column can be dropped since the effective information have been extracted.

```
combined.drop('Name', axis=1, inplace=True)
```

Convert the categorical variable title into dummy variables and then drop the original Title column.

```
>>> titles_dummies = pd.get_dummies(combined['Title'], prefix='Title')
>>> combined = pd.concat([combined, titles_dummies], axis=1)
>>> print(combined.columns)
Index(['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Cabin',
       'Embarked_C', 'Embarked_Q', 'Embarked_S', 'Died', 'Embarked', 'Title',
       'Title_Master', 'Title_Miss', 'Title_Mr', 'Title_Mrs', 'Title_Officer',
       'Title_Royalty'],
      dtype='object')
>>> combined.drop('Title', axis=1, inplace=True)
```

Using mean value of ticket fare to fill in the missing fares.

Filling Embarked missing values with the most frequent value ’S’ as observed.

Similarly, categorical column embarked needs to be converted into dummy variables and drop the original Embarked column.

Cabin missing values need to be filled in with ‘M’, which stands for missing value for both train and test dataset.

```
>>> train_cabin, test_cabin = set(), set()
>>> for c in combined.iloc[:891]['Cabin']:
...     try:
...         train_cabin.add(c[0])
...     except:
...         train_cabin.add('M')
...
>>> for c in combined.iloc[891:]['Cabin']:
...     try:
...         test_cabin.add(c[0])
...     except:
...         test_cabin.add('M')
...
>>> print(train_cabin)
{'D', 'T', 'C', 'M', 'E', 'G', 'A', 'F', 'B'}
>>> print(test_cabin)
{'D', 'C', 'M', 'E', 'G', 'A', 'F', 'B'}
>>>
```

Convert the Cabin variable into dummy variables as well and drop the original Cabin column. Similarly, Pclass is applied the same process.

Mapping the male as 1 and female as 0 for Sex column.

Combine SibSp and Parch together into a new column as FamilySize. And categorise them into Single category if FamilySize is 1, small family if FamilySize is between 2 to 4, otherwise large family.

```
>>> print(combined.columns)
Index(['Sex', 'Age', 'SibSp', 'Parch', 'Ticket', 'Fare', 'Embarked_C',
       'Embarked_Q', 'Embarked_S', 'Died', 'Title_Master', 'Title_Miss',
       'Title_Mr', 'Title_Mrs', 'Title_Officer', 'Title_Royalty', 'Embarked_C',
       'Embarked_Q', 'Embarked_S', 'Cabin_A', 'Cabin_B', 'Cabin_C', 'Cabin_D',
       'Cabin_E', 'Cabin_F', 'Cabin_G', 'Cabin_M', 'Cabin_T', 'Pclass_1',
       'Pclass_2', 'Pclass_3', 'FamilySize', 'Single', 'SmallFamily',
       'LargeFamily'],
      dtype='object')
>>>
```

Different formats of values of ticket column can be observed, and some of them have non-digital prefix and some only have digital numbers. Therefore, the non-digital letters will be kept for ticket value and set it as NONE if it's all digital number for ticket value, through this way, the ticket column can be more categorical and converted into dummy variables.

```
>>> combined
      Sex   Age  SibSp  Parch             Ticket      Fare  Embarked_C  \
0       1  22.0      1      0          A/5 21171    7.2500         0.0
1       0  38.0      1      0           PC 17599   71.2833         1.0
2       0  26.0      0      0   STON/O2. 3101282    7.9250         0.0
3       0  35.0      1      0             113803   53.1000         0.0
4       1  35.0      0      0             373450    8.0500         0.0
...   ...   ...    ...    ...                ...       ...         ...
1302    1  28.0      0      0          A.5. 3236    8.0500         NaN
1303    0  39.0      0      0           PC 17758  108.9000         NaN
1304    1  38.5      0      0  SOTON/O.Q. 3101262    7.2500         NaN
1305    1  28.0      0      0              359309    8.0500         NaN
1306    1   6.5      1      1                2668   22.3583         NaN
```

```
>>> def cleanTicket(ticket):
...     ticket = ticket.replace('.','')
...     ticket = ticket.replace('/','')
...     ticket = ticket.split()
...     ticket = map(lambda t : t.strip(), ticket)
...     ticket = [x for x in ticket if not x.isdigit()]
...     if len(ticket) > 0:
...         return ticket[0]
...     else:
...         return 'NONE'
...
>>> combined['Ticket'] = combined['Ticket'].map(cleanTicket)
>>>
>>> combined['Ticket'].unique()
array(['A5', 'PC', 'STONO2', 'NONE', 'PP', 'CA', 'SCParis', 'SCA4', 'A4',
       'SP', 'SOC', 'WC', 'SOTONOQ', 'WEP', 'STONO', 'C', 'SCPARIS',
       'SOP', 'Fa', 'LINE', 'FCC', 'SWPP', 'SCOW', 'PPP', 'SC', 'SCAH',
       'AS', 'SOPP', 'FC', 'SOTONO2', 'CASOTON', 'SCA3', 'STONOQ', 'AQ4',
       'A', 'LP', 'AQ3'], dtype=object)
>>>
```