```python
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved outside of the current session
```

1. Define data problems

On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. Translated 32% survival rate. One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class. Given the training data set, we want to predict if some passengers survived or not.

2. Import data

```python
train_data = pd.read_csv('../input/titanic/train.csv')
test_data = pd.read_csv('../input/titanic/test.csv')
combine = [train_data, test_data]
```

3. Analyze by describing data

data preview

```python
train_data.head()
```

```python
train_data.tail()
```

```python
print(train_data.columns.values)
```

Categorical features: Survied, Pclass, Sex, Embarked

Numerical features: Age, SibSp(number of siblings), Fare

Mix: Ticket

```
train_data.info()
print('==============================================')
test_data.info()
```

Missing value from training set: Age, Cabin, Embarked

Missing value from testing set: Age, Fare, Cabin

```
train_data.describe()
```

Early numerical features insights

1. According to problem description, there are 1502 out of 2224 passengers were killed. The observed surviving rate is (2224-1502)/2224 = 32%. Whereas in training set, surviving rate is 38%.
2. Most of people (>50%) were Pclass 3.
3. The passengers were relatively young.
4. Most of people (around 50% - 75%) were not travling with siblings or spouse.
5. Near 25% of people were travelling with parents and children.
6. Fares varied significantly with few passengers (<1%) paying as high as $512.

```
train_data.describe(include=['O'])
```

Early categorical features insights

1. Names are unique.
2. 577/891 = 65% are male.
3. Some passengers shared one cabin.
4. Most of people embarked S.

*****Analyze by pivoting ******

1. Pclass - Survived: Significant correlation among Pclass=1 -> feature kept
2. Sex - Survived: Significant correlation among Sex=Female -> feature kept
3. Sibsp - Survived: Some index have zero correlation -> derive and create new features

4. Parch - Survived: Some index have zero correlation -> derive and create new features

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

train_data[['Sex', 'Survived']].groupby(['Sex'],
as_index=True).mean().sort_values(by='Survived', ascending=False)

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

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

Analyze by data visualization

1. Numeric feature: Age - Survived: Mostly age group 18 - 30 did not survive -> Feature kept, fill missing values, create age groups

2. Numeric Ordinal feature: Age - Pclass - Survived: Most people in Class1 survived, whereas most people in Class3 did not; Most infants in Class3 survived. -> Feature kept

3. Categorical feature:Embarked - Pclass - Sex - Survived: Feature kept, filling missing values

4. Categorial & numerical features: Fare - Survived Higher fare paying passengers had better survival. Feature fare kept and creat groups.

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

age = sns.FacetGrid(train_data, col='Survived')
age.map(plt.hist, 'Age', bins = 20)

pclass = sns.FacetGrid(train_data, col='Survived',
row='Pclass',height=2, aspect=1)
pclass.map(plt.hist, 'Age', bins=20)
pclass.add_legend();

embarked = sns.FacetGrid(train_data, row='Embarked', height=2,
aspect=5)
embarked.map(sns.pointplot, 'Pclass','Survived', 'Sex',
paletee='deep', order=[1, 2, 3], hue_order=["male", "female"])
embarked.add_legend()


fare = sns.FacetGrid(train_data, row='Embarked', col='Survived',
height=2, aspect=2)
fare.map(sns.barplot, 'Sex', 'Fare', alpha=0.3, ci=None,
```

```
order=['female', 'male'])
fare.add_legend()
```

Wrangle Data

1. dropping features

```
print("Before", train_data.shape, test_data.shape, combine[0].shape, 
combine[1].shape)

train_data = train_data.drop(['Ticket', 'Cabin'], axis=1)
test_data = test_data.drop(['Ticket', 'Cabin'], axis=1)


print("After", train_data.shape, test_data.shape, combine[0].shape, 
combine[1].shape)

combine=[train_data, test_data]
```

Create an ordinal feature Title in training set and testing set

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

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

title_mapping = {"Mr":1, "Miss":2, "Mrs":3, "Master":4, "Rare":5}
for dataset in combine:
    dataset['Title'] = dataset['Title'].map(title_mapping)
    dataset['Title'] = dataset['Title'].fillna(0)
train_data = train_data.drop(['Name','PassengerId'], axis=1)
test_data = test_data.drop(['Name'], axis=1)
combine = [train_data, test_data]
train_data.shape, test_data.shape
```

Converting feature:Sex to 0 and 1

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

Complete and Convert a numerical continuous feature: Age

```
age = sns.FacetGrid(train_data, row='Pclass', col='Sex', height=2,
aspect=2)
age.map(plt.hist, 'Age', bins=20)
age.add_legend()
```

Generate an array to contain Pclass-Sex age values

```
import numpy as np
guess_age = np.zeros((2,3))
guess_age
```

Calculate 6 entires for 6 combinations: Pclass=1 Female, Pclass=2 Female, Pclass=3 Female; Pclass=1 Male, Pclass=2 Male, Pclass=3 Male;

```
for dataset in combine:
    for i in range(0, 2):
        for j in range(0, 3):
            guess_data = dataset[(dataset['Sex']==i) &
(dataset['Pclass']==j+1)]['Age'].dropna()
            age_guess = guess_data.mean()
            guess_age[i,j] = int(age_guess)
    for i in range(0, 2):
        for j in range(0, 3):
            dataset.loc[(dataset.Age.isnull()) & (dataset.Sex == i) &
(dataset.Pclass == j+1),'Age'] = guess_age[i,j]
    dataset['Age'] = dataset['Age'].astype(int)
train_data.head()
train_data.info()
guess_age
```

create age cut

```
train_data['Age_cut'] = pd.cut(train_data['Age'], 5)
#train_data[['Age_cut', 'Survived']].groupby(['Age_cut'],
as_index=False).mean().sort_values(by='Age_cut', ascending=True)
train_data[['Age_cut', 'Survived']].groupby(['Age_cut'],
as_index=False).mean().sort_values(by='Age_cut', ascending=True)

#Creat ordinals age
for dataset in combine:
    dataset.loc[(dataset.Age <= 16), 'Age'] = 0
    dataset.loc[(dataset.Age > 16) & (dataset.Age <= 32),'Age'] = 1
    dataset.loc[(dataset.Age > 32) & (dataset.Age <= 48),'Age'] = 2
    dataset.loc[(dataset.Age > 48) & (dataset.Age <= 64),'Age'] = 3
    dataset.loc[(dataset.Age > 64),'Age'] = 4
train_data.head()
#remove age_cut
#train_data = train_data.drop(['Age_cut'], axis=1)
#train_data.head()
```

Create a feature Family size combining Parch and SibSp

```
for dataset in combine:
    dataset['Family_size'] = dataset['Parch'] + dataset['SibSp'] + 1
train_data[['Family_size', 'Survived']].groupby(['Family_size'],
as_index=False).mean().sort_values(by='Family_size', ascending=False)
```

Create feature Is_alone for 1 if Family_size=0; 0 if Family_size > 1

```
for dataset in combine:
    dataset['Is_alone'] = 0
    dataset.loc[dataset['Family_size']==1, 'Is_alone'] = 1
train_data.head()
train_data[['Is_alone', 'Survived']].groupby(['Is_alone'],
as_index=False).mean()
```

drop parch, sibsp, family_size

```
train_data = train_data.drop(['Parch', 'SibSp', 'Family_size'],
axis=1)
test_data = test_data.drop(['Parch', 'SibSp', 'Family_size'], axis=1)
combine = [train_data, test_data]

train_data.head()
```

Creat a feature combining age and pclass by age * pclass

```
for dataset in combine:
    dataset['Age*Pclass'] = dataset['Age'] * dataset['Pclass']
train_data.loc[:, ['Age', 'Pclass', 'Age*Pclass']].head()
```

Filling missing values for categorical features: Embarked by most occurance

```
freq_port = train_data.Embarked.dropna().mode()[0]
freq_port

for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].fillna(freq_port)
train_data.info()

train_data[['Embarked','Survived']].groupby('Embarked',
as_index=False).mean().sort_values(by='Survived', ascending=False)

#convert port to ordinary numeric
for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].map({'C':2, 'Q':1,
'S':0}).astype(int)
train_data.head()
```

filling Fare in test_data by mode and convert

```
test_data.info()
```

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

```
train_data['Fare_cut'] = pd.qcut(train_data['Fare'], 4)
train_data[['Fare_cut', 'Survived']].groupby('Fare_cut',
as_index=False).mean().sort_values(by='Survived', ascending=True)
```

Convert cuts to ordinal numerical values

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

```
train_data.head()
```

drop age_cut, fare_cut

```
train_data.head()
train = train_data.drop('Age_cut', axis=1)
train = train.drop('Fare_cut', axis=1)
train.head()
test = test_data
```

```
test.head()
```

Modeling, predict, and solve

1. Logistic regression
2. KNN or K-nearest Neighbours
3. SVM
4. Naive Bayes
5. Decision tree
6. Random Forest

prepair train set and test set:

TrainX, TrainY

TestX, TestY

```python
X_train = train.drop("Survived", axis=1)
Y_train = train["Survived"]
X_test  = test.drop("PassengerId", axis=1).copy()
X_train.shape, Y_train.shape, X_test.shape

# logistic regression
from sklearn.linear_model import LogisticRegression
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
Y_pred = logreg.predict(X_test)
acc_log = round(logreg.score(X_train, Y_train) * 100 ,2)
acc_log

# logistic regression correlation
coef = pd.DataFrame(train.columns.delete(0))
coef.columns = ['Feature']
coef['correlation'] = pd.Series(logreg.coef_[0])
coef.sort_values(by='correlation', ascending=False)

# Support vector Machines
from sklearn.svm import SVC, LinearSVC
svc = SVC()
svc.fit(X_train, Y_train)
Y_pred = svc.predict(X_test)
acc_svc = round(svc.score(X_train, Y_train)*100 ,2)
acc_svc

# KNN
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier()
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_train, Y_train)*100, 2)
acc_knn

# Gussian NB
from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
nb.fit(X_train, Y_train)
Y_pred = nb.predict(X_test)
acc_nb = round(100 * nb.score(X_train, Y_train),2)
acc_nb

#Decision tree
from sklearn.tree import DecisionTreeClassifier
tree = DecisionTreeClassifier()
tree.fit(X_train, Y_train)
Y_pred = tree.predict(X_test)
acc_tree = round(100* tree.score(X_train, Y_train),2)
acc_tree
```

```
# Random forest
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, Y_train)
Y_pred = rf.predict(X_test)
acc_rf = round(100 * rf.score(X_train, Y_train),2)
acc_rf

# SGD
from sklearn.linear_model import SGDClassifier
sgd = SGDClassifier()
sgd.fit(X_train, Y_train)
Y_pred = sgd.predict(X_test)
acc_sgd = round(100*sgd.score(X_train, Y_train), 2)
acc_sgd
```

MODEL EVALUATION

```
models = pd.DataFrame({
    'Model' : ['Logistic Regression', 'Support Vector Machine', 'KNN',
'Gaussian NB', 'Decision Tree', 'Random Forest', 'SGD'],
    'Score' : [acc_log, acc_svc, acc_knn, acc_nb, acc_tree, acc_rf,
acc_sgd ]
})
models.sort_values(by='Score', ascending=False)

submission = pd.DataFrame({
        "PassengerId": test["PassengerId"],
        "Survived": Y_pred
    })
submission.to_csv('Desktop\Kaggle\Titanic\submission.csv',
index=False)
```