Titanic Survival Prediction

1.Titanic dataset is used to buid a model that predicts whether a passenger on the Titanic survived or not. This is a classic beginner project with readily available data. 2.The dataset typically used for this project contains information about individual passengers, such as their age, gender, ticket class,fare, cabin and whether or not they survived.

Work flow

Data loading--Data pre-processing--Exploratory Data analysis--train Test split--Logestic Regression--Evaluation

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler  # Import StandardScaler
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import LogisticRegression
from sklearn import tree
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
df = pd.read_csv("/dataset.csv")
```

Data pre-processing

```
df.info()
```

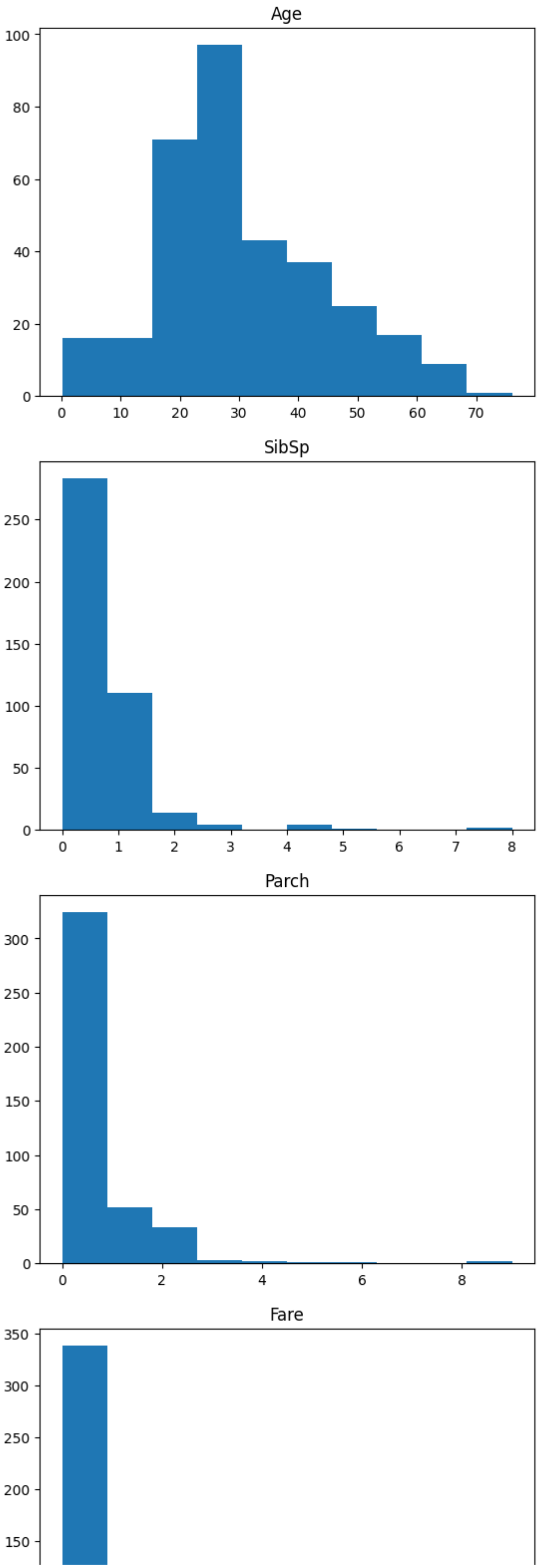
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Survived     418 non-null    int64
 2   Pclass       418 non-null    int64
 3   Name         418 non-null    object
 4   Sex          418 non-null    object
 5   Age          332 non-null    float64
 6   SibSp        418 non-null    int64
 7   Parch        418 non-null    int64
 8   Ticket       418 non-null    object
 9   Fare         417 non-null    float64
 10  Cabin        91 non-null     object
 11  Embarked     418 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 39.3+ KB
```

```
df.describe()
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	418.000000	418.000000	418.000000	332.000000	418.000000	418.000000	417.000000
mean	1100.500000	0.363636	2.265550	30.272590	0.447368	0.392344	35.627188
std	120.810458	0.481622	0.841838	14.181209	0.896760	0.981429	55.907576
min	892.000000	0.000000	1.000000	0.170000	0.000000	0.000000	0.000000
25%	996.250000	0.000000	1.000000	21.000000	0.000000	0.000000	7.895800
50%	1100.500000	0.000000	3.000000	27.000000	0.000000	0.000000	14.454200
75%	1204.750000	1.000000	3.000000	39.000000	1.000000	0.000000	31.500000
max	1309.000000	1.000000	3.000000	76.000000	8.000000	9.000000	512.329200

```
df_num = df[['Age', 'SibSp', 'Parch', 'Fare']]
df_cat = df[['Survived', 'Pclass', 'Sex', 'Ticket', 'Cabin', 'Embarked']]
```

```
for i in df_num.columns:
    plt.hist(df_num[i])
    plt.title(i)
    plt.show()
```




```
print(df_num.corr())
```

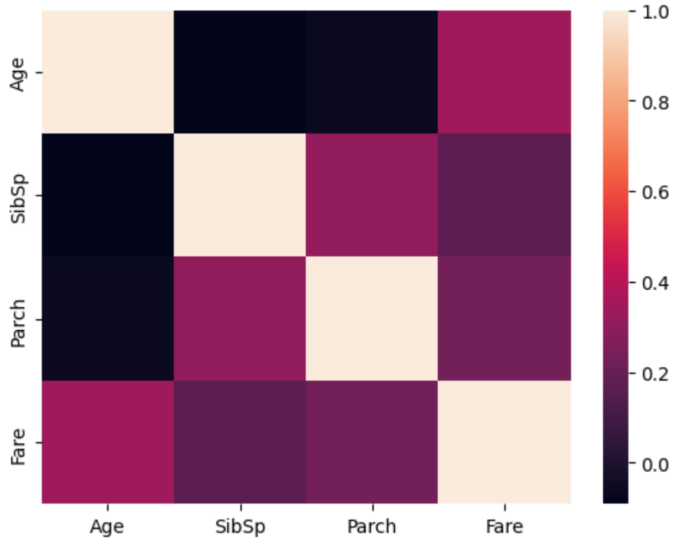
```
             Age     SibSp     Parch      Fare
Age     1.000000 -0.091587 -0.061249  0.337932
SibSp  -0.091587  1.000000  0.306895  0.171539
```

```
Parch -0.061249  0.306895  1.000000  0.230046
Fare   0.337932  0.171539  0.230046  1.000000
```

0 100 200 300 400 500

```
sns.heatmap(df_num.corr())
```

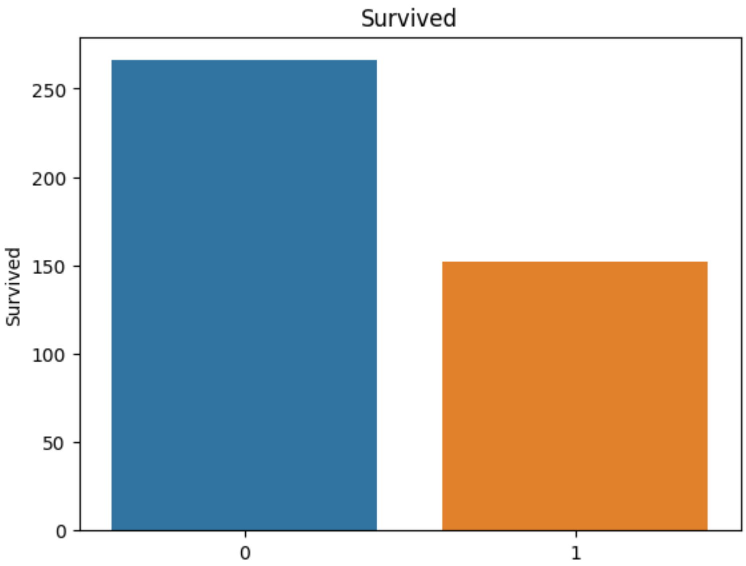
```
<Axes: >
```

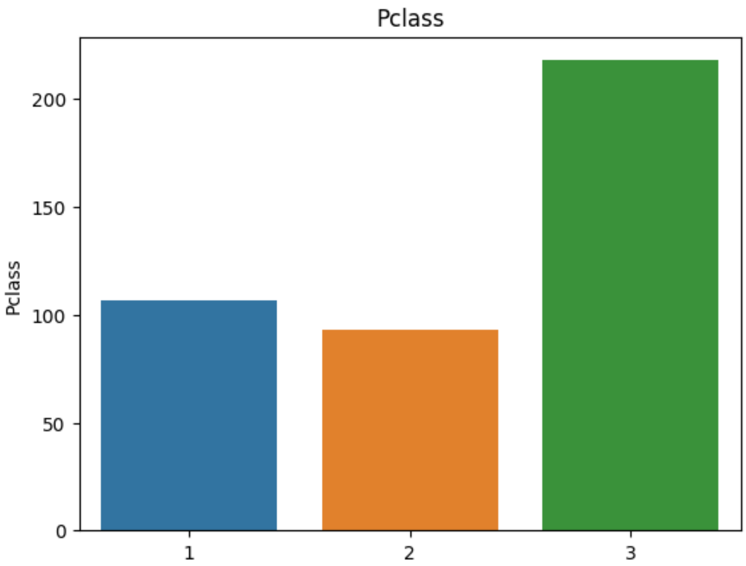


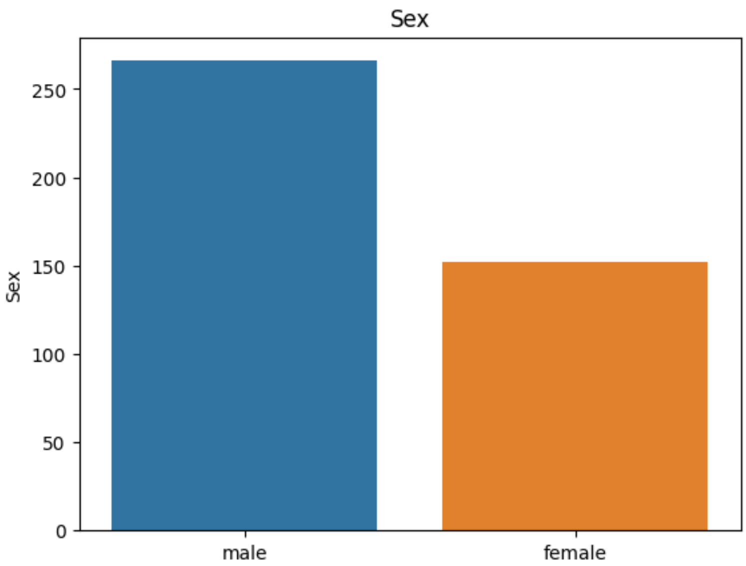

```
pd.pivot_table(df, index='Survived', values=['Age', 'SibSp', 'Parch', 'Fare'])
```

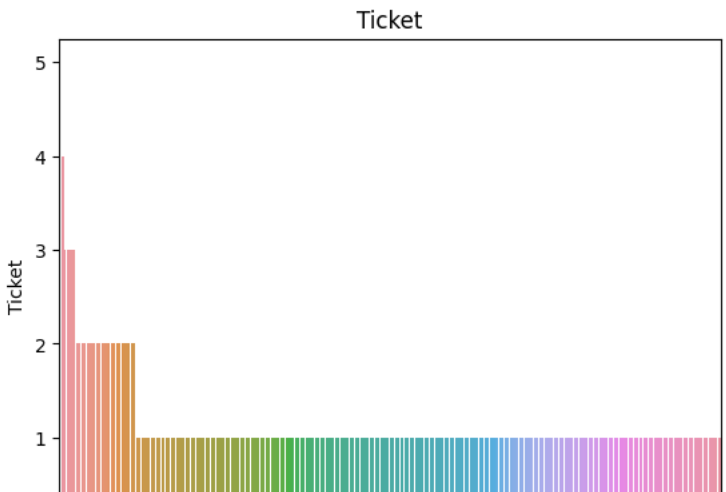
	Age	Fare	Parch	SibSp
Survived				
0	30.272732	27.527877	0.274436	0.379699
1	30.272362	49.747699	0.598684	0.565789

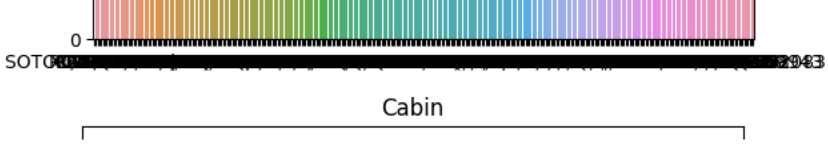
```
for i in df_cat.columns:
    sns.barplot(x=df_cat[i].value_counts().index, y=df_cat[i].value_counts()).set_title(i)
    plt.show()
```


Survived
Survived
0
50
100
150
200
250
0
1


Pclass
Pclass
0
50
100
150
200
1
2
3

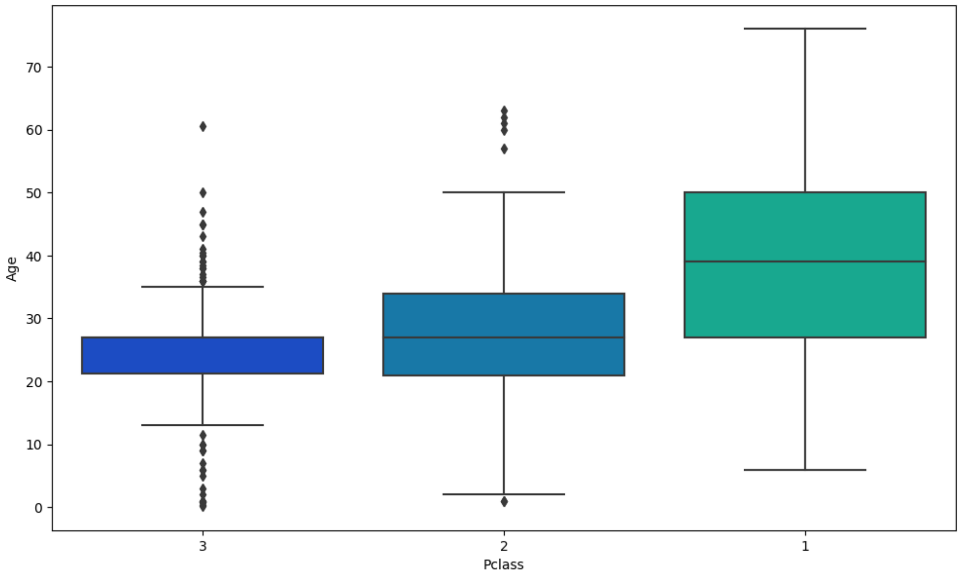

Sex
Sex
0
50
100
150
200
250
male
female


Ticket
Ticket
1
2
3
4
5



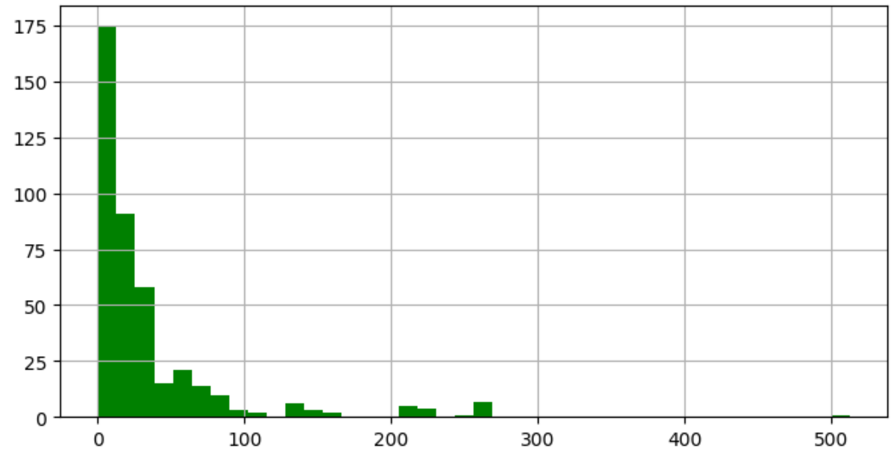

```python
plt.figure(figsize=(12, 7))
sns.boxplot(x='Pclass',y='Age',data=df,palette='winter')
```

```
<Axes: xlabel='Pclass', ylabel='Age'>
```



```python
df['Fare'].hist(color='green',bins=40,figsize=(8,4))
```

```
<Axes: >
```




```python
print(pd.pivot_table(df, index='Survived', columns='Pclass', values='Ticket', aggfunc='count'))
print(pd.pivot_table(df, index='Survived', columns='Sex', values='Ticket', aggfunc='count'))
print(pd.pivot_table(df, index='Survived', columns='Embarked', values='Ticket', aggfunc='count'))
```

```
Pclass     1   2    3
Survived
0         57  63  146
1         50  30   72
Sex       female   male
Survived
0            NaN  266.0
1          152.0    NaN
Embarked   C   Q    S
Survived
0         62  22  182
1         40  24   88
```

```python
# Create a new feature 'numeric_ticket' to indicate if the ticket is numeric
df['numeric_ticket'] = df.Ticket.apply(lambda x: 1 if x.isnumeric() else 0)
df['ticket_letters'] = df.Ticket.apply(lambda x: ''.join(x.split(' ')[:-1]).replace('.', '').replace('/', '').lower() if len(x.split(' ')[:-1]) > 0 el
df['name_title'] = df.Name.apply(lambda x: x.split(',')[1].split('.')[0].strip())
print(df['name_title'].value_counts())
```

```
Mr        240
Miss       78
Mrs        72
Master     21
Col         2
Rev         2
Ms          1
Dr          1
Dona        1
Name: name_title, dtype: int64
```

Spliting the training and test data

```
df.Age = df.Age.fillna(df.Age.median())
df.Fare = df.Fare.fillna(df.Fare.median())
df.dropna(subset=['Embarked'], inplace=True)
df['norm_sibsp'] = np.log(df.SibSp + 1)
df['norm_fare'] = np.log(df.Fare + 1)
df.Pclass = df.Pclass.astype(str)
all_dummies = pd.get_dummies(df[['Pclass', 'Sex', 'Age', 'SibSp', 'Parch', 'norm_fare', 'Embarked','numeric_ticket', 'name_title']])
scale = StandardScaler()
all_dummies_scaled = all_dummies.copy()
all_dummies_scaled[['Age', 'SibSp', 'Parch', 'norm_fare']] = scale.fit_transform(all_dummies_scaled[['Age', 'SibSp', 'Parch', 'norm_fare']])
y = df.Survived
X_train, X_test, y_train, y_test = train_test_split(all_dummies_scaled, y, test_size=0.33, random_state=42)
gnb = GaussianNB()
cv = cross_val_score(gnb, X_train, y_train, cv=5)
print(cv)
print(cv.mean())
```

```
[1. 1. 1. 1. 1.]
1.0
```

```
lr = LogisticRegression(max_iter=2000)
cv = cross_val_score(lr, X_train, y_train, cv=5)
print(cv)
print(cv.mean())
```

```
[1. 1. 1. 1. 1.]
1.0
```

```
dt = tree.DecisionTreeClassifier(random_state=1)
cv = cross_val_score(dt, X_train, y_train, cv=5)
print(cv)
print(cv.mean())
```

```
[1. 1. 1. 1. 1.]
1.0
```

Conclusion

The logistic algorithm is likely a very strong model for the given problem.It has learned the underlying patterns in the training data accurately and is capable of making accurate predictions on given 'Titanic' dataset.