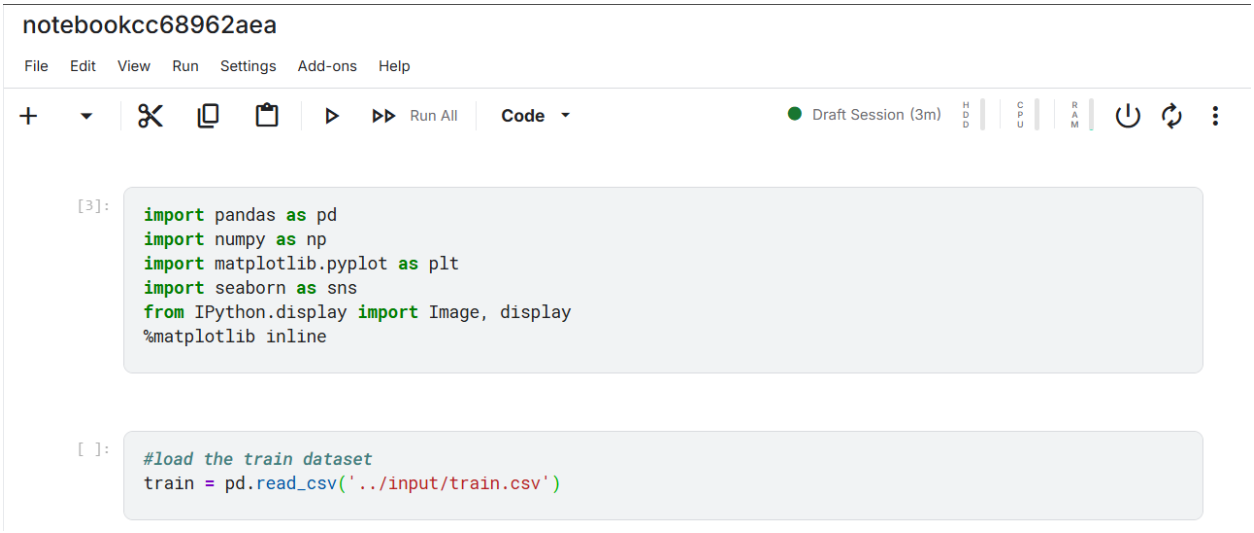
SKILLCRAFT TECHNOLOGY INTERNSHIP TASK-2

Perform data cleaning and exploratory data analysis (EDA) on a dataset of your choice, such as the Titanic dataset from kaggle. Explore the relationships between variables and identify patterns and trends in the dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from IPython.display import Image, display

# Jupyter magic command (do NOT include parentheses)
```




```
#inspect the first few rows of the train dataset
display(train.head())
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

[ ]:
```
#inspect the first few rows of the train dataset
display(train.head())
```

[ ]:
```
# set the index to passengerId
train = train.set_index('PassengerId')
```

[ ]:
```
#load the test dataset
test = pd.read_csv('../input/test.csv')
```

```
#inspect the first few rows of the test dataset
display(test.head())
```

+ Code + Markdown

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	NaN	Q
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	NaN	S
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	NaN	Q
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	NaN	S
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	NaN	S

```python
import pandas as pd

# Load the dataset (adjust path if needed)
train = pd.read_csv('train.csv')

# Display the shape of the dataset
print("Shape of the train dataset:", train.shape)
```

```
Shape of the train dataset: (891, 12)
```

```python
# Check out the data summary
# Age, Cabin and Embarked has missing data
train.head()
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
PassengerId											
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S

```python
# identify datatypes of the 11 columns, add the stats to the datadict
datadict = pd.DataFrame(train.dtypes)
datadict
```

	0
Survived	int64
Pclass	int64
Name	object
Sex	object
Age	float64
SibSp	int64
Parch	int64
Ticket	object
Fare	float64
Cabin	object
Embarked	object

```
# identify missing values of the 11 columns,add the stats to the datadict
datadict['MissingVal'] = train.isnull().sum()
datadict
```

	0	MissingVal
Survived	int64	0
Pclass	int64	0
Name	object	0
Sex	object	0
Age	float64	177
SibSp	int64	0
Parch	int64	0
Ticket	object	0
Fare	float64	0
Cabin	object	687
Embarked	object	2

```
# Identify number of unique values, For object nunique will the number of levels
# Add the stats the data dict
datadict['NUnique']=train.nunique()
datadict
```

	0	MissingVal	NUnique
Survived	int64	0	2
Pclass	int64	0	3
Name	object	0	891
Sex	object	0	2
Age	float64	177	88
SibSp	int64	0	7
Parch	int64	0	7
Ticket	object	0	681
Fare	float64	0	248
Cabin	object	687	147
Embarked	object	2	3

```
# Identify the count for each variable, add the stats to datadict
datadict['Count']=train.count()
datadict
```

	0	MissingVal	NUnique	Count
Survived	int64	0	2	891
Pclass	int64	0	3	891
Name	object	0	891	891
Sex	object	0	2	891
Age	float64	177	88	714
SibSp	int64	0	7	891
Parch	int64	0	7	891
Ticket	object	0	681	891
Fare	float64	0	248	891
Cabin	object	687	147	204
Embarked	object	2	3	889

```
# rename the 0 column
datadict = datadict.rename(columns={0:'DataType'})
datadict
```

	DataType	MissingVal	NUnique	Count
Survived	int64	0	2	891
Pclass	int64	0	3	891
Name	object	0	891	891
Sex	object	0	2	891
Age	float64	177	88	714
SibSp	int64	0	7	891
Parch	int64	0	7	891
Ticket	object	0	681	891
Fare	float64	0	248	891
Cabin	object	687	147	204
Embarked	object	2	3	889

```
# get discripte statistcs on "object" datatypes
train.describe(include=['object'])
```

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Slayter, Miss. Hilda Mary	male	CA. 2343	B96 B98	S
freq	1	577	7	4	644

```
# get discriptive statistcs on "number" datatypes
train.describe(include=['number'])
```

	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

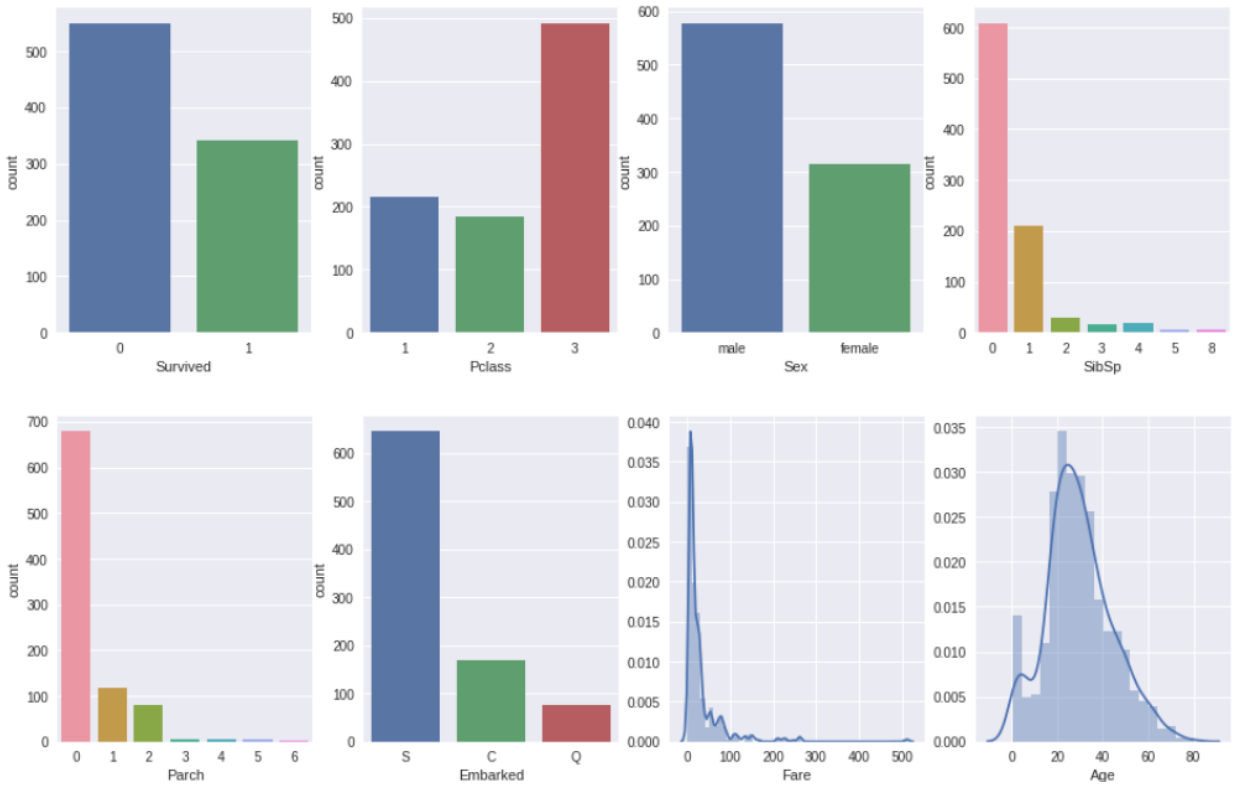
```
train.Survived.value_counts(normalize=True)
```

```
import pandas as pd

# Load the Titanic dataset (adjust path as needed)
train = pd.read_csv('train.csv')

# Calculate normalized value counts of the 'Survived' column
survival_distribution = train.Survived.value_counts(normalize=True)

# Display the result
print("Survival Distribution (Normalized):")
print(survival_distribution)
```

```
Survival Distribution (Normalized):
0    0.616162
1    0.383838
Name: Survived, dtype: float64
```

```
fig, axes = plt.subplots(2, 4, figsize=(16, 10))
sns.countplot('Survived',data=train,ax=axes[0,0])
sns.countplot('Pclass',data=train,ax=axes[0,1])
sns.countplot('Sex',data=train,ax=axes[0,2])
sns.countplot('SibSp',data=train,ax=axes[0,3])
sns.countplot('Parch',data=train,ax=axes[1,0])
sns.countplot('Embarked',data=train,ax=axes[1,1])
sns.distplot(train['Fare'], kde=True,ax=axes[1,2])
sns.distplot(train['Age'].dropna(),kde=True,ax=axes[1,3])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa6366bdd8>
```

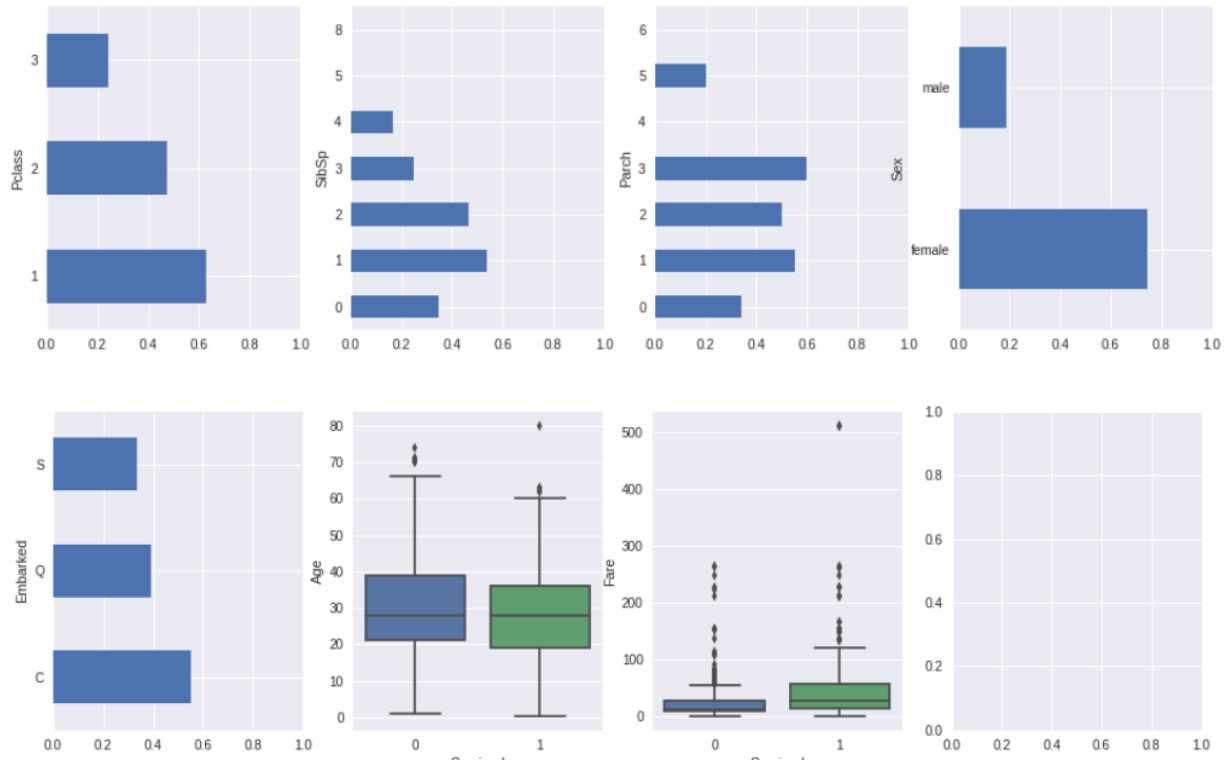


[ ]:

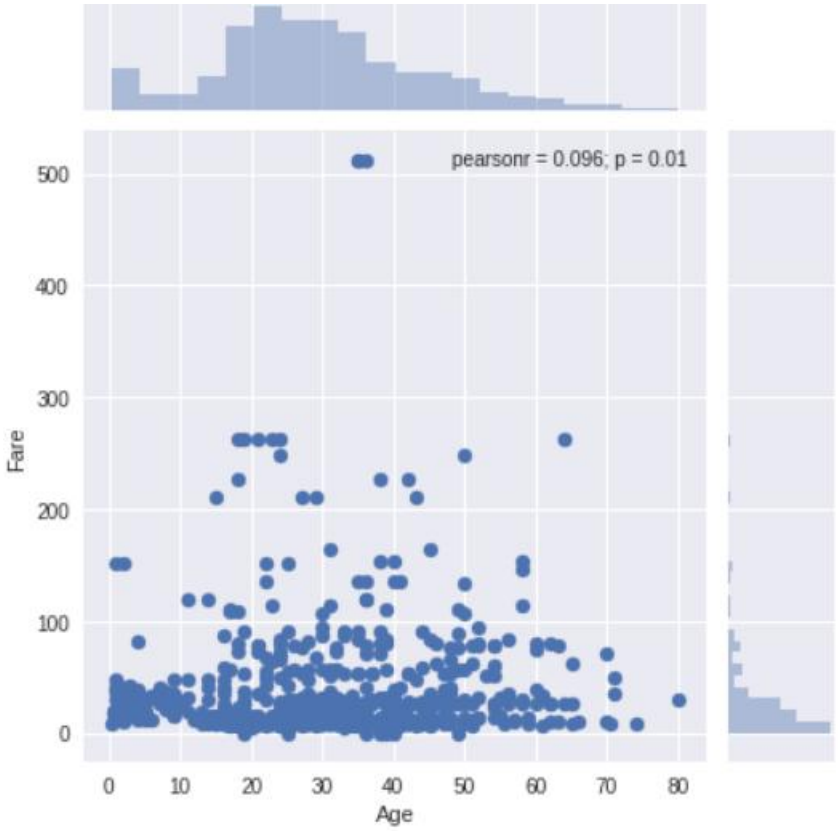
```
fig, axes = plt.subplots(2, 4, figsize=(16, 10))
sns.countplot('Survived',data=train,ax=axes[0,0])
sns.countplot('Pclass',data=train,ax=axes[0,1])
sns.countplot('Sex',data=train,ax=axes[0,2])
sns.countplot('SibSp',data=train,ax=axes[0,3])
sns.countplot('Parch',data=train,ax=axes[1,0])
sns.countplot('Embarked',data=train,ax=axes[1,1])
sns.distplot(train['Fare'], kde=True,ax=axes[1,2])
sns.distplot(train['Age'].dropna(),kde=True,ax=axes[1,3])
```

```
figbi, axesbi = plt.subplots(2, 4, figsize=(16, 10))
train.groupby('Pclass')['Survived'].mean().plot(kind='barh',ax=axesbi[0,0],xlim=[0,1])
train.groupby('SibSp')['Survived'].mean().plot(kind='barh',ax=axesbi[0,1],xlim=[0,1])
train.groupby('Parch')['Survived'].mean().plot(kind='barh',ax=axesbi[0,2],xlim=[0,1])
train.groupby('Sex')['Survived'].mean().plot(kind='barh',ax=axesbi[0,3],xlim=[0,1])
train.groupby('Embarked')['Survived'].mean().plot(kind='barh',ax=axesbi[1,0],xlim=[0,1])
sns.boxplot(x="Survived", y="Age", data=train,ax=axesbi[1,1])
sns.boxplot(x="Survived", y="Fare", data=train,ax=axesbi[1,2])
```

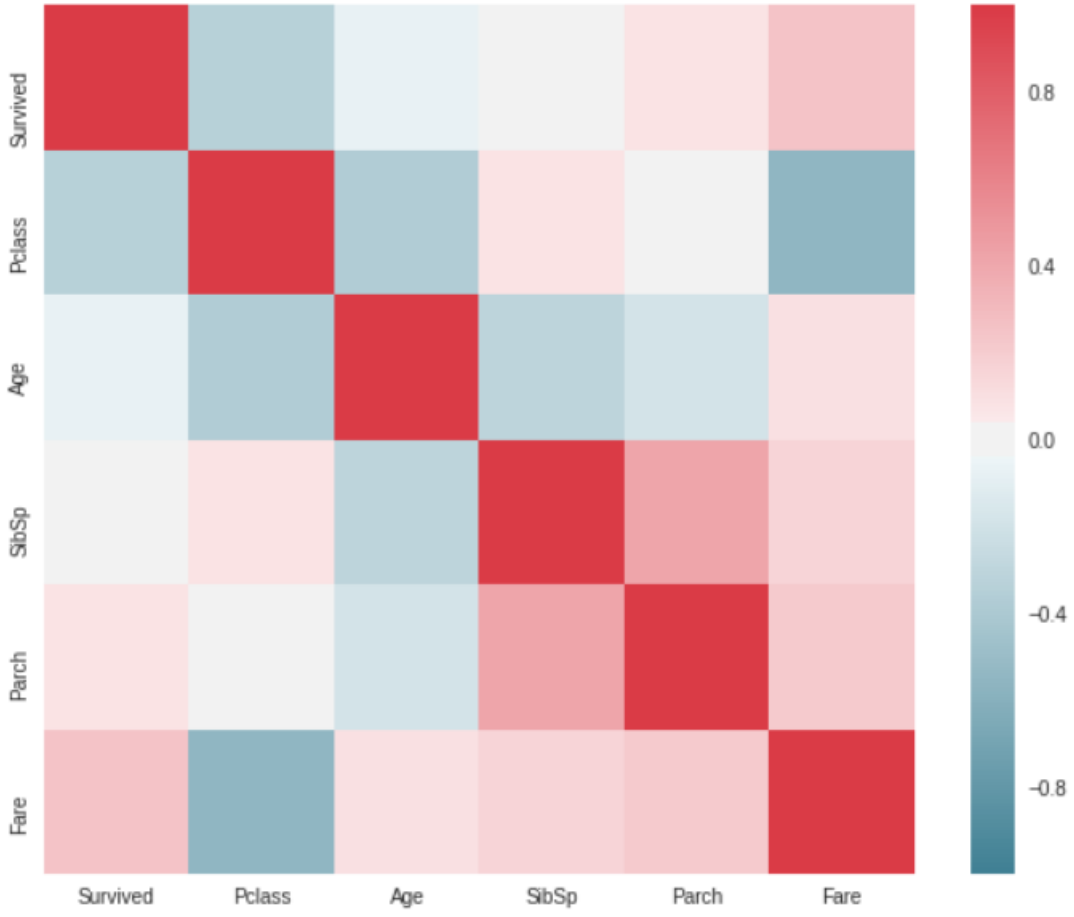
```
<matplotlib.axes._subplots.AxesSubplot at 0x7ffa631ff9e8>
```




```
sns.jointplot(x="Age", y="Fare", data=train);
```




```
import seaborn as sns

f, ax = plt.subplots(figsize=(10, 8))
corr = train.corr()
sns.heatmap(corr,
            mask=np.zeros_like(corr, dtype=np.bool),
            cmap=sns.diverging_palette(220, 10, as_cmap=True),
            square=True, ax=ax)
```


Survived
Pclass
Age
SibSp
Parch
Fare
Survived
Pclass
Age
SibSp
Parch
Fare
0.8
0.4
0.0
−0.4
−0.8

[ ]:
```
train['Name_len']=train.Name.str.len()
```

[ ]:
```
train['Ticket_First']=train.Ticket.str[0]
```

[ ]:
```
train['FamilyCount']=train.SibSp+train.Parch
```

[ ]:
```
train['Cabin_First']=train.Cabin.str[0]
```

[ ]:
```
# Regular expression to get the title of the Name
train['title'] = train.Name.str.extract('\, ([A-Z][^ ]*\.)',expand=False)
```

[ ]:
```
train['Cabin_First']=train.Cabin.str[0]
```

[ ]:
```
# Regular expression to get the title of the Name
train['title'] = train.Name.str.extract('\, ([A-Z][^ ]*\.)',expand=False)
```

▷
```
train.title.value_counts().reset_index()
```

	index	title
0	Mr.	517
1	Miss.	182
2	Mrs.	125
3	Master.	40
4	Dr.	7
5	Rev.	6
6	Mlle.	2
7	Major.	2
8	Col.	2
9	Ms.	1
10	Jonkheer.	1
11	Don.	1
12	Mme.	1
13	Sir.	1
14	Capt.	1
15	Lady.	1

[ ]:

```
# we see that there are 15 Zero values and its reasonbale
# to flag them as missing values since every ticket
# should have a value greater than 0
print((train.Fare == 0).sum())
```

```
import pandas as pd

# Load dataset
train = pd.read_csv('train.csv')

# Count the number of zero Fare entries
zero_fares = (train.Fare == 0).sum()

# Display the result
print("Number of passengers with Fare = 0:", zero_fares)
```

```
Number of passengers with Fare = 0: 15
```

```
import pandas as pd

# Load the Titanic dataset
train = pd.read_csv('train.csv')

# Validate that there are no more zero Fare values
zero_fares_remaining = (train.Fare == 0).sum()

# Output the result
print("Number of passengers with Fare = 0:", zero_fares_remaining)
```

```
Number of passengers with Fare = 0: 15
```

```
Number of passengers with Fare = 0: 0
```

```
# keep the index
train[train.Fare.isnull()].index
```

```
Int64Index([180, 264, 272, 278, 303, 414, 467, 482, 598, 634, 675, 733, 807,
            816, 823],
           dtype='int64', name='PassengerId')
```

```
train.Fare.mean()
```

```
32.75564988584471
```

```
# impute the missing Fare values with the mean Fare value
train.Fare.fillna(train.Fare.mean(),inplace=True)
```

+ Code + Markdown

```
# validate if any null values are present after the imputation
train[train.Fare.isnull()]
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Name_len	Ticket_First	Fa
PassengerId														

```
# we see that there are 0 Zero values
print((train.Age == 0).sum())
```

```
# impute the missing Age values with the mean Fare value
train.Age.fillna(train.Age.mean(),inplace=True)
```

+ Code + Markdown

```
# validate if any null values are present after the imputation
train[train.Age.isnull()]
```

	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	Name_len	Ticket_First	Fa
PassengerId														

```
# We see that a majority 77% of the Cabin variable has missing values.
# Hence will drop the column from training a machine learnign algorithem
train.Cabin.isnull().mean()
```

```
0.77104377104377109
```

```
train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 891 entries, 1 to 891
Data columns (total 16 columns):
Survived        891 non-null int64
Pclass          891 non-null int64
Name            891 non-null object
Sex             891 non-null object
Age             891 non-null float64
SibSp           891 non-null int64
Parch           891 non-null int64
Ticket          891 non-null object
Fare            891 non-null float64
Cabin           204 non-null object
Embarked        889 non-null object
Name_len        891 non-null int64
Ticket_First    891 non-null object
FamilyCount     891 non-null int64
Cabin_First     204 non-null object
title           890 non-null object
dtypes: float64(2), int64(6), object(8)
memory usage: 158.3+ KB
```

```
train.columns
```

```
Index(['Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket',
       'Fare', 'Cabin', 'Embarked', 'Name_len', 'Ticket_First', 'FamilyCount',
       'Cabin_First', 'title'],
      dtype='object')
```

```
trainML = train[['Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp', 'Parch', 'Ticket',
       'Fare', 'Embarked', 'Name_len', 'Ticket_First', 'FamilyCount',
       'title']]
```

```
# drop rows of missing values
trainML = trainML.dropna()
```

```
# check the datafram has any missing values
trainML.isnull().sum()
```

```
Survived        0
Pclass          0
Name            0
Sex             0
Age             0
SibSp           0
Parch           0
Ticket          0
Fare            0
Embarked        0
Name_len        0
Ticket_First    0
FamilyCount     0
title           0
dtype: int64
```

```
# check the datafram has any missing values
trainML.isnull().sum()
```

```
# Import Estimator AND Instantiate estimator class to create an estimator object
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
```

```
X_Age = trainML[['Age']].values
y = trainML['Survived'].values
# Use the fit method to train
lr.fit(X_Age,y)
# Make a prediction
y_predict = lr.predict(X_Age)
y_predict[:10]
(y == y_predict).mean()
```

```
0.6182432432432432
```

```
X_Fare = trainML[['Fare']].values
y = trainML['Survived'].values
# Use the fit method to train
lr.fit(X_Fare,y)
# Make a prediction
y_predict = lr.predict(X_Fare)
y_predict[:10]
(y == y_predict).mean()
```

```
0.66216216216216217
```

```
X_sex = pd.get_dummies(trainML['Sex']).values
y = trainML['Survived'].values
# Use the fit method to train
lr.fit(X_sex, y)
# Make a prediction
y_predict = lr.predict(X_sex)
y_predict[:10]
(y == y_predict).mean()
```

```
0.786036036036036
```

```
X_sex = pd.get_dummies(trainML['Sex']).values
y = trainML['Survived'].values
# Use the fit method to train
lr.fit(X_sex, y)
# Make a prediction
y_predict = lr.predict(X_sex)
y_predict[:10]
(y == y_predict).mean()
```

```
X_pclass = pd.get_dummies(trainML['Pclass']).values
y = trainML['Survived'].values
lr = LogisticRegression()
lr.fit(X_pclass, y)
# Make a prediction
y_predict = lr.predict(X_pclass)
y_predict[:10]
(y == y_predict).mean()
```

```
0.67792792792792789
```

```
X_pclass = pd.get_dummies(trainML['Pclass']).values
y = trainML['Survived'].values
lr = LogisticRegression()
lr.fit(X_pclass, y)
# Make a prediction
y_predict = lr.predict(X_pclass)
y_predict[:10]
(y == y_predict).mean()
```

```
from sklearn.ensemble import RandomForestClassifier
X=trainML[['Age', 'SibSp', 'Parch',
       'Fare', 'Name_len', 'FamilyCount']].values # Taking all the numerical values
y = trainML['Survived'].values
RF = RandomForestClassifier()
RF.fit(X, y)
# Make a prediction
y_predict = RF.predict(X)
y_predict[:10]
(y == y_predict).mean()
```

```
0.96734234234234229
```