Workflow goals

- Classifying
- Correlating
- Converting
- Completing
- Correcting
- Creating
- Charting

Acquire data

```
import warnings
warnings.filterwarnings('ignore')

# data analysis
import pandas as pd
import os
import numpy as np


# visualization
import seaborn as sns
import matplotlib.pyplot as plt


# machine learning
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


os.getcwd()

train_df = pd.read_csv("train.csv")
test_df = pd.read_csv("test.csv")

combine = [train_df, test_df]
```

```
# Preview data
train_df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	7
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	5
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8

```
test_df.head()
```

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

Analyse by describing data

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

```
['PassengerId' 'Survived' 'Pclass' 'Name' 'Sex' 'Age' 'SibSp' 'Parch'
 'Ticket' 'Fare' 'Cabin' 'Embarked']
```

Data Dictionary(https://www.kaggle.com/c/titanic/data (https://www.kaggle.com/c/titanic/data))

Data types

- Castegorical: Survived, Sex, and Embarked. Ordinal: Pclass.
- Continous: Age, Fare. Discrete: SibSp, Parch.
- Mixed: Ticket is a mix of numeric and alphanumeric data types. Cabin is alphanumeric
- Name feature may contain errors or typos.

```
# Check missing values
train_df.tail()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
886	887	0	2	Montvila, Rev. Juozas	male	27.0	0	0	211536	13
887	888	1	1	Graham, Miss. Margaret Edith	female	19.0	0	0	112053	30
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	NaN	1	2	W./C. 6607	23
889	890	1	1	Behr, Mr. Karl Howell	male	26.0	0	0	111369	30
890	891	0	3	Dooley, Mr. Patrick	male	32.0	0	0	370376	7.7

Require correcting in data pre-processing later on

- Cabin > Age > Embarked features contain a number of null values
- Cabin > Age are incomplete in case of test dataset

Helping us during converting goal

- Seven features are integer or floats. Six in case of test dataset.
- Five features are string

```
train_df.shape
```

```
(891, 12)
```

```
test_df.shape # 1 column less than training set
```

```
(418, 11)
```

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
PassengerId    891 non-null int64
Survived       891 non-null int64
Pclass         891 non-null int64
Name           891 non-null object
Sex            891 non-null object
Age            714 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
dtypes: float64(2), int64(5), object(5)
memory usage: 83.6+ KB
________________________________________
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
PassengerId    418 non-null int64
Pclass         418 non-null int64
Name           418 non-null object
Sex            418 non-null object
Age            332 non-null float64
SibSp          418 non-null int64
Parch          418 non-null int64
Ticket         418 non-null object
Fare           417 non-null float64
Cabin          91 non-null object
Embarked       418 non-null object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

```
train_df.describe(percentiles=[.1, .2, .3, .4, .5, .6, .61, .62, .68, .69, .7, .75, .8, .85, .9, .99])
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch	
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.0
10%	90.000000	0.000000	1.000000	14.000000	0.000000	0.000000	7.5
20%	179.000000	0.000000	1.000000	19.000000	0.000000	0.000000	7.8
30%	268.000000	0.000000	2.000000	22.000000	0.000000	0.000000	8.0
40%	357.000000	0.000000	2.000000	25.000000	0.000000	0.000000	10.
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.
60%	535.000000	0.000000	3.000000	31.800000	0.000000	0.000000	21.
61%	543.900000	0.000000	3.000000	32.000000	0.000000	0.000000	23.
62%	552.800000	1.000000	3.000000	32.000000	0.000000	0.000000	24.
68%	606.200000	1.000000	3.000000	35.000000	0.000000	0.000000	26.
69%	615.100000	1.000000	3.000000	35.000000	1.000000	0.000000	26.
70%	624.000000	1.000000	3.000000	36.000000	1.000000	0.000000	27.
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.
80%	713.000000	1.000000	3.000000	41.000000	1.000000	1.000000	39.
85%	757.500000	1.000000	3.000000	45.000000	1.000000	1.000000	56.
90%	802.000000	1.000000	3.000000	50.000000	1.000000	2.000000	77.
99%	882.100000	1.000000	3.000000	65.870000	5.000000	4.000000	249
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512

- Total samples are 891 or 40% of the actual number of passengers on board the Titanic (2,224).
- Around 38% samples survived representative of the actual survival rate at 32%. percentiles=[.61, .62]
- Most passengers (> 75%) did not travel with parents or children.percentiles=[.75, .8]
- Nearly 30% of the passengers had siblings and/or spouse aboard. percentile=[.68, .69]
- Fares varied significantly with few passengers (<1%) paying as high as $512.
- Few elderly passengers (<1%) within age range 65-80

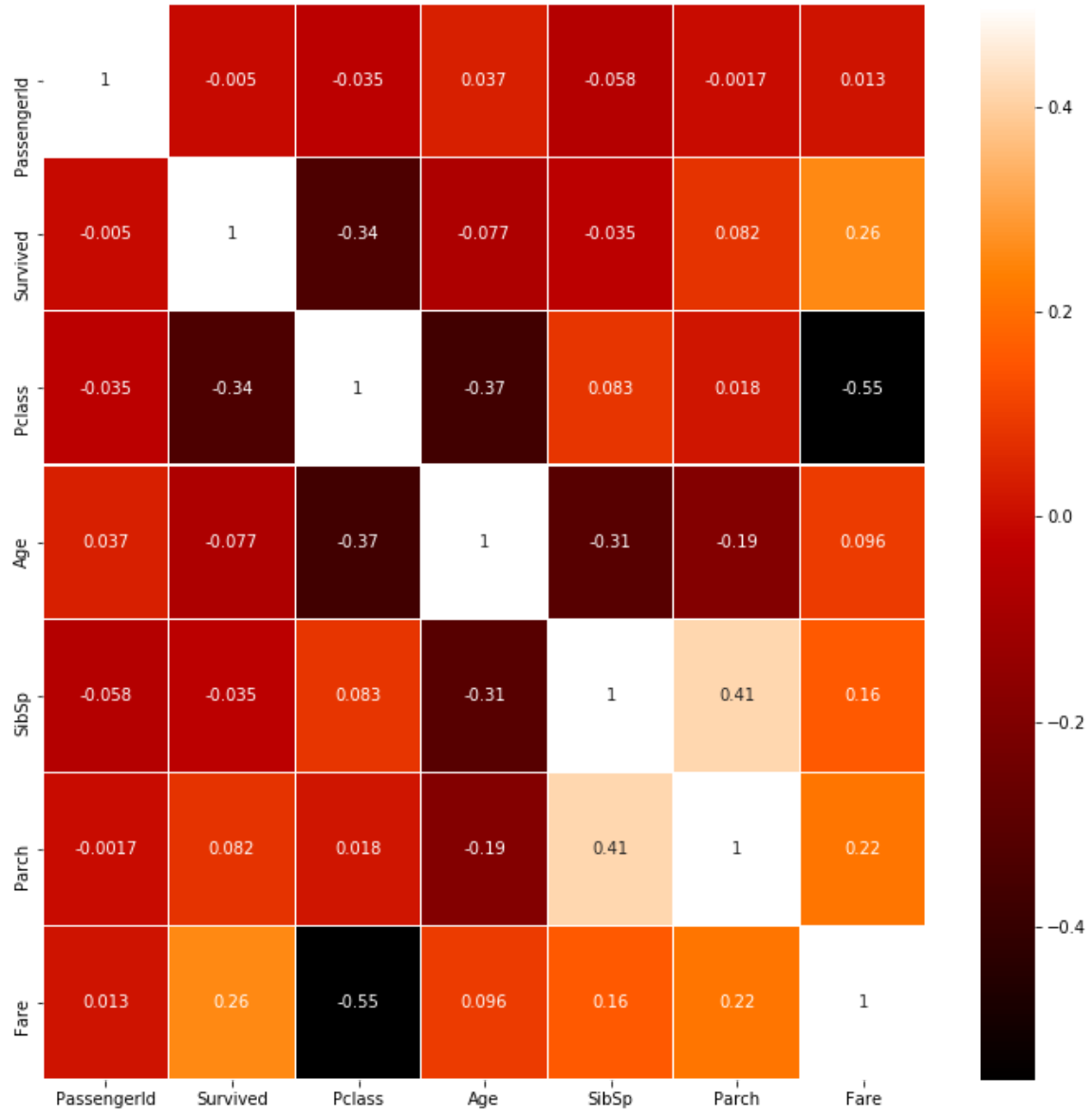
```
train_df.describe(include=['O']) # include object types in the result
```

	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Stanley, Mr. Edward Roland	male	1601	B96 B98	S
freq	1	577	7	4	644

- Names are unique across the dataset (count=unique=891)
- Sex variable as two possible values with 65% male (top=male, freq=577/count=891).
- Cabin values have several dupicates across samples. Alternatively several passengers shared a cabin.
- Embarked takes three possible values. S port used by most passengers (top=S)
- Ticket feature has high ratio (22%) of duplicate values (unique=681).

Correlation Heatmap

```
plt.figure(figsize=(12, 12))
sns.heatmap(train_df.corr(), linewidths=0.1, vmax=0.5, cmap=plt.cm.gist_heat, linecolor="white",
 annot=True)
plt.show()
```


PassengerId
Survived
Pclass
Age
SibSp
Parch
Fare
1 -0.005 -0.035 0.037 -0.058 -0.0017 0.013
-0.005 1 -0.34 -0.077 -0.035 0.082 0.26
-0.035 -0.34 1 -0.37 0.083 0.018 -0.55
0.037 -0.077 -0.37 1 -0.31 -0.19 0.096
-0.058 -0.035 0.083 -0.31 1 0.41 0.16
-0.0017 0.082 0.018 -0.19 0.41 1 0.22
0.013 0.26 -0.55 0.096 0.16 0.22 1
0.4
0.2
0.0
−0.2
−0.4

Analyse by pivoting features

The reason for pivoting is becase we couldn't find enough information from the correlation heatmap.

We can only do so at this stage for features which do not have any empty values. It also makes sense doing so only for features which are categorical (Sex), ordinal (Pclass) or discrete (SibSp, Parch) type.

- **Pclass:** We observe significant correlation (>0.5) among Pclass=1 and Survived (classifying #3). We decide to include this feature in our model.
- **Sex:** We confirm the observation during problem definition that Sex=female had very high survival rate at 74% (classifying #1).
- **SibSp and Parch:** These features have zero correlation for certain values. It may be best to derive a feature or a set of features from these individual features (creating #1).

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

	Pclass	Survived
0	1	0.629630
1	2	0.472826
2	3	0.242363

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

	Sex	Survived
0	female	0.742038
1	male	0.188908

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

	SibSp	Survived
1	1	0.535885
2	2	0.464286
0	0	0.345395
3	3	0.250000
4	4	0.166667
5	5	0.000000
6	8	0.000000

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

	Parch	Survived
3	3	0.600000
1	1	0.550847
2	2	0.500000
0	0	0.343658
5	5	0.200000
4	4	0.000000
6	6	0.000000

Analyse by visualising

Correlating numerical features

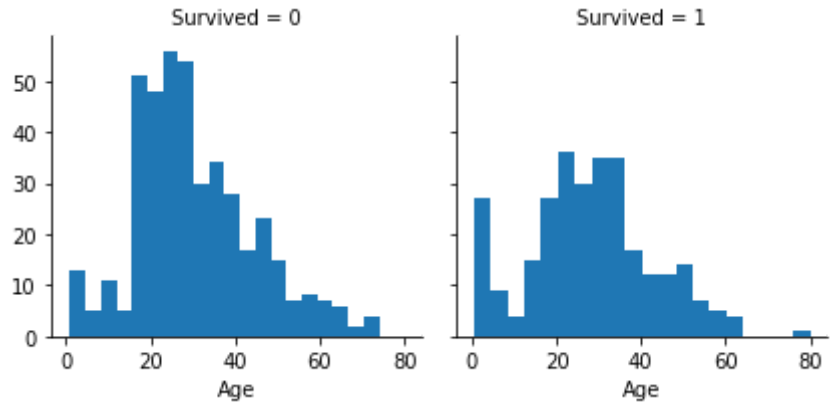
Observations

- Infants (Age <=4) had high survival rate.
- Oldest passengers (Age = 80) survived.
- Large number of 15-25 year olds did not survive.
- Most passengers are in 15-35 age range.

Decisions

- We should consider Age (our assumption classifying #2) in our model training.
- Complete the Age feature for null values (completing #1).
- We should band age groups (creating #3).

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



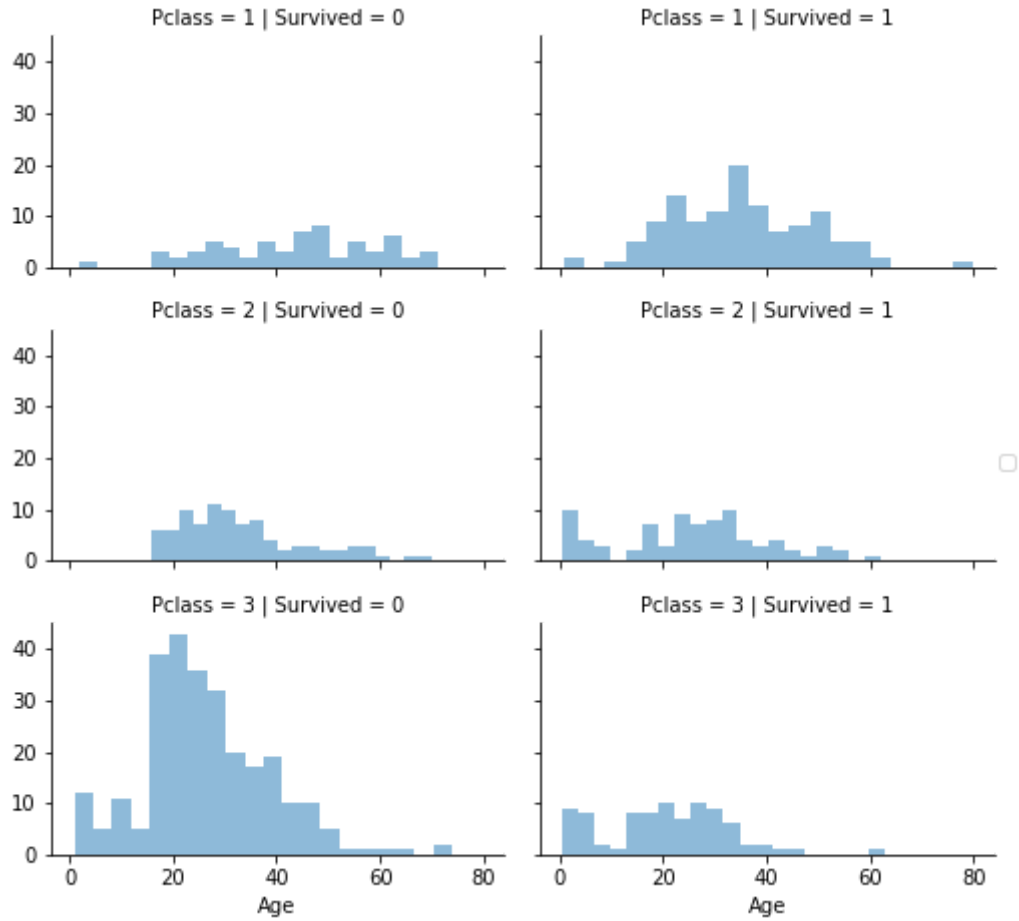
Correlating nemerical and ordinal features

Observations

- Pclass=3 had most passengers, however most did not survive. Confirms our classifying assumption #3.
- Infant passengers in Pclass=2 and Pclass=3 mostly survived. Further qualifies our classifying assumption #2.
- Most passengers in Pclass=1 survived. Confirms our classifying assumption #3.
- Pclass varies in terms of Age distribution of passengers.

Decisions

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



Correlating categorical features

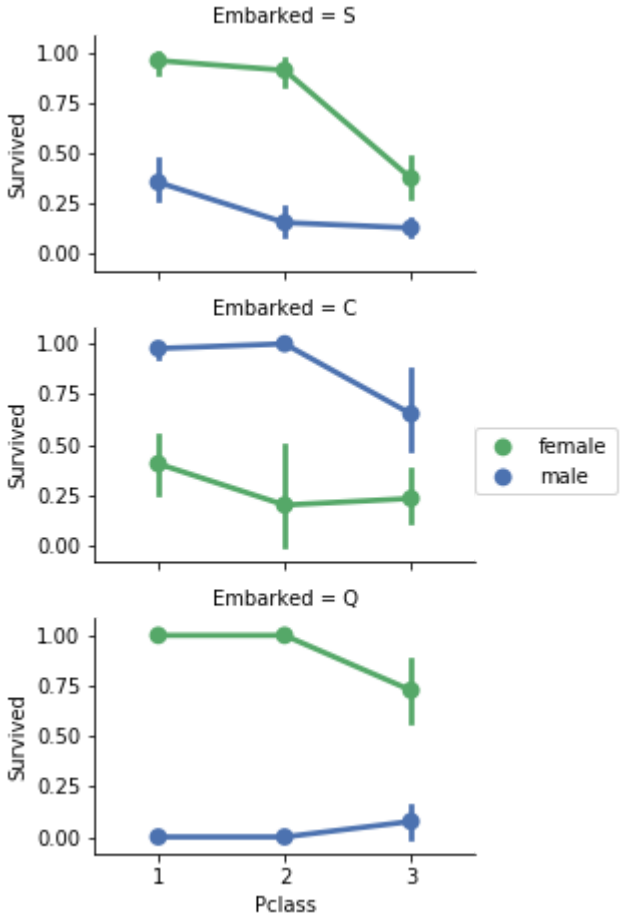
Observations

- Female passengers had much better survival rate than males. Confirms classifying (#1).
- Exception in Embarked=C where males had higher survival rate. This could be a correlation between Pclass and Embarked and in turn Pclass and Survived, not necessarily direct correlation between Embarked and Survived.
- Males had better survival rate in Pclass=3 when compared with Pclass=2 for C and Q ports. Completing (#2).
- Ports of embarkation have varying survival rates for Pclass=3 and among male passengers. Correlating (#1).

Decisions

- Add Sex feature to model training.
- Complete and add Embarked feature to model training.

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



Correlating categorical and numerical features¶

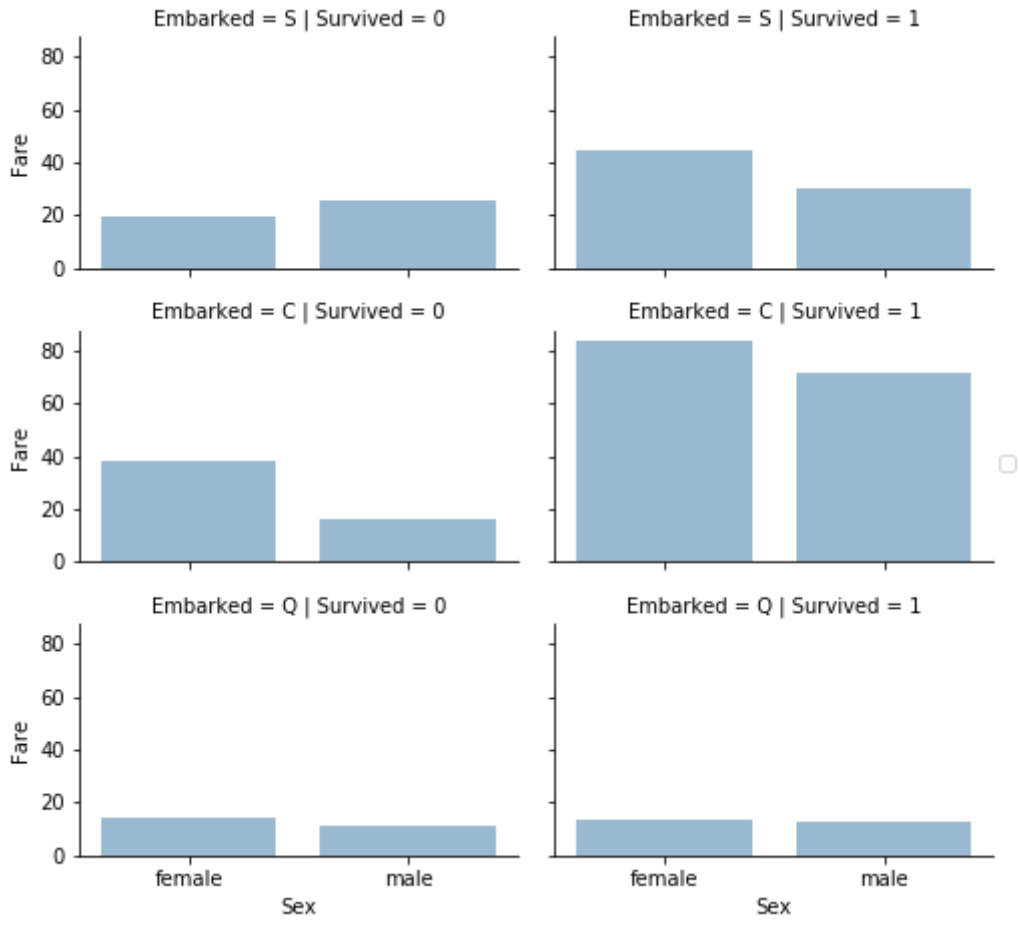
Observations

- Higher fare paying passengers had better survival. Confirms our assumption for creating (#4) fare ranges.
- Port of embarkation correlates with survival rates. Confirms correlating (#1) and completing (#2).

Decisions

- Consider banding Fare feature.

```
grid = sns.FacetGrid(train_df, row='Embarked', col='Survived', size=2.2, aspect=1.6)
grid.map(sns.barplot, 'Sex', 'Fare', alpha=.5, ci=None)
grid.add_legend()
plt.show()
```



Wrangling data

Correcting by dropping features¶

- Based on our assumptions and decisions we want to drop the Cabin (correcting #2) and Ticket (correcting #1) features.

```
print("Before", train_df.shape, test_df.shape, combine[0].shape, combine[1].shape)

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

"After", train_df.shape, test_df.shape, combine[0].shape, combine[1].shape
```

```
Before (891, 12) (418, 11) (891, 12) (418, 11)
```

```
('After', (891, 10), (418, 9), (891, 10), (418, 9))
```

Creating new feature extracting from existing¶

- We want to analyze if Name feature can be engineered to extract titles and test correlation between titles and survival, before dropping Name and PassengerId features.

Observations

- Most titles band Age groups accurately. For example: Master title has Age mean of 5 years.
- Survival among Title Age bands varies slightly.
- Certain titles mostly survived (Mme, Lady, Sir) or did not (Don, Rev, Jonkheer).

Decisions

- We decide to retain the new Title feature for model training.

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

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

Sex	female	male
Title		
Capt	0	1
Col	0	2
Countess	1	0
Don	0	1
Dr	1	6
Jonkheer	0	1
Lady	1	0
Major	0	2
Master	0	40
Miss	182	0
Mlle	2	0
Mme	1	0
Mr	0	517
Mrs	125	0
Ms	1	0
Rev	0	6
Sir	0	1

We can replace many titles with a more common name or classify them as Rare.

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

	Title	Survived
0	Master	0.575000
1	Miss	0.702703
2	Mr	0.156673
3	Mrs	0.793651
4	Rare	0.347826

We can convert the categorical titles to ordinal.

```
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_df.head()
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Fare	Em
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	7.2500	S
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	71.2833	C
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	7.9250	S
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	53.1000	S
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	8.0500	S

Now we can safely drop the Name feature from training and testing datasets. We also do not need the PassengerId feature in the training dataset.

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

```
((891, 9), (418, 9))
```

```
train_df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	male	22.0	1	0	7.2500	S	1
1	1	1	female	38.0	1	0	71.2833	C	3
2	1	3	female	26.0	0	0	7.9250	S	2
3	1	1	female	35.0	1	0	53.1000	S	3
4	0	3	male	35.0	0	0	8.0500	S	1

Converting a categorical feature

Let us start by converting Sex feature to a new feature called Gender where female=1 and male=0.

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

train_df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22.0	1	0	7.2500	S	1
1	1	1	1	38.0	1	0	71.2833	C	3
2	1	3	1	26.0	0	0	7.9250	S	2
3	1	1	1	35.0	1	0	53.1000	S	3
4	0	3	0	35.0	0	0	8.0500	S	1

Completing a numerical continuous feature

Now we should start estimating and completing features with missing or null values. We will first do this for the Age feature.

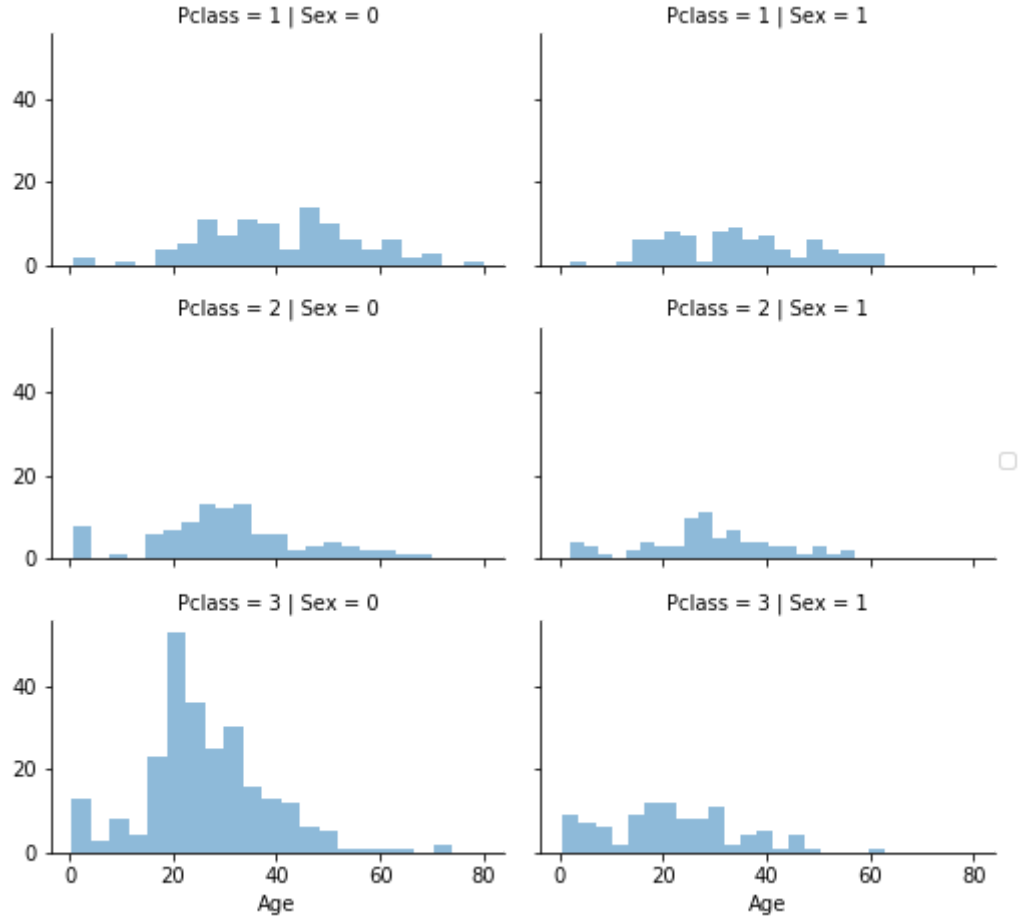
1. A simple way is to generate random numbers between mean and standard deviation.

1. More accurate way of guessing missing values is to use other correlated features. In our case we note correlation among Age, Gender, and Pclass. Guess Age values using median values for Age across sets of Pclass and Gender feature combinations. So, median Age for Pclass=1 and Gender=0, Pclass=1 and Gender=1, and so on...

1. Combine methods 1 and 2. So instead of guessing age values based on median, use random numbers between mean and standard deviation, based on sets of Pclass and Gender combinations.

Method 1 and 3 will introduce random noise into our models. The results from multiple executions might vary. We will prefer method 2.

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



Let us start by preparing an empty array to contain guessed Age values based on Pclass x Gender combinations.

```
guess_ages = np.zeros((2,3))
guess_ages
```

```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

Now we iterate over Sex (0 or 1) and Pclass (1, 2, 3) to calculate guessed values of Age for the six combinations.

```
for dataset in combine:
    for i in range(0, 2):
        for j in range(0, 3):
            guess_df = dataset[(dataset['Sex'] == i) & (dataset['Pclass'] == j+1)]['Age'].dropna
()

            # age_mean = guess_df.mean()
            # age_std = guess_df.std()
            # age_guess = rnd.uniform(age_mean - age_std, age_mean + age_std)

            age_guess = guess_df.median()

            # Convert random age float to nearest .5 age
            guess_ages[i,j] = int( age_guess/0.5 + 0.5 ) * 0.5

    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_ages[i,j]

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

train_df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	22	1	0	7.2500	S	1
1	1	1	1	38	1	0	71.2833	C	3
2	1	3	1	26	0	0	7.9250	S	2
3	1	1	1	35	1	0	53.1000	S	3
4	0	3	0	35	0	0	8.0500	S	1

Let us create Age bands and determine correlations with Survived.

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

	AgeBand	Survived
0	(-0.08, 16.0]	0.550000
1	(16.0, 32.0]	0.337374
2	(32.0, 48.0]	0.412037
3	(48.0, 64.0]	0.434783
4	(64.0, 80.0]	0.090909

Let us replace Age with ordinals based on these bands.

```
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']
train_df.head()
```

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title	AgeBand
0	0	3	0	1	1	0	7.2500	S	1	(16.0, 32.0]
1	1	1	1	2	1	0	71.2833	C	3	(32.0, 48.0]
2	1	3	1	1	0	0	7.9250	S	2	(16.0, 32.0]
3	1	1	1	2	1	0	53.1000	S	3	(32.0, 48.0]
4	0	3	0	2	0	0	8.0500	S	1	(32.0, 48.0]

We can remove the AgeBand feature.

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

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	Title
0	0	3	0	1	1	0	7.2500	S	1
1	1	1	1	2	1	0	71.2833	C	3
2	1	3	1	1	0	0	7.9250	S	2
3	1	1	1	2	1	0	53.1000	S	3
4	0	3	0	2	0	0	8.0500	S	1

Create new feature combining existing features¶

We can create a new feature for FamilySize which combines Parch and SibSp. This will enable us to drop Parch and SibSp from our datasets.

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

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

	FamilySize	Survived
3	4	0.724138
2	3	0.578431
1	2	0.552795
6	7	0.333333
0	1	0.303538
4	5	0.200000
5	6	0.136364
7	8	0.000000
8	11	0.000000

We can create another feature called IsAlone.

```
for dataset in combine:
    dataset['IsAlone'] = 0
    dataset.loc[dataset['FamilySize'] == 1, 'IsAlone'] = 1

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

	IsAlone	Survived
0	0	0.505650
1	1	0.303538

Let us drop Parch, SibSp, and FamilySize features in favor of IsAlone.

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

train_df.head()
```

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone
0	0	3	0	1	7.2500	S	1	0
1	1	1	1	2	71.2833	C	3	0
2	1	3	1	1	7.9250	S	2	1
3	1	1	1	2	53.1000	S	3	0
4	0	3	0	2	8.0500	S	1	1

We can also create an artificial feature combining Pclass and Age.

```
for dataset in combine:
    dataset['Age*Class'] = dataset.Age * dataset.Pclass

train_df.loc[:, ['Age*Class', 'Age', 'Pclass']].head(10)
```

	Age*Class	Age	Pclass
0	3	1	3
1	2	2	1
2	3	1	3
3	2	2	1
4	6	2	3
5	3	1	3
6	3	3	1
7	0	0	3
8	3	1	3
9	0	0	2

Completing a categorical feature¶

Embarked feature takes S, Q, C values based on port of embarkation. Our training dataset has two missing values. We simply fill these with the most common occurance.

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

```
'S'
```

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

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

	Embarked	Survived
0	C	0.553571
1	Q	0.389610
2	S	0.339009

Converting categorical feature to numeric¶

We can now convert the EmbarkedFill feature by creating a new numeric Port feature.

```
for dataset in combine:
    dataset['Embarked'] = dataset['Embarked'].map( {'S': 0, 'C': 1, 'Q': 2} ).astype(int)

train_df.head()
```

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	0	3	0	1	7.2500	0	1	0	3
1	1	1	1	2	71.2833	1	3	0	2
2	1	3	1	1	7.9250	0	2	1	3
3	1	1	1	2	53.1000	0	3	0	2
4	0	3	0	2	8.0500	0	1	1	6

Quick completing and converting a numeric feature¶

We can now complete the Fare feature for single missing value in test dataset using mode to get the value that occurs most frequently for this feature.

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

	PassengerId	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	892	3	0	2	7.8292	2	1	1	6
1	893	3	1	2	7.0000	0	3	0	6
2	894	2	0	3	9.6875	2	1	1	6
3	895	3	0	1	8.6625	0	1	1	3
4	896	3	1	1	12.2875	0	3	0	3

We can create FareBand.

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

	FareBand	Survived
0	(-0.001, 7.91]	0.197309
1	(7.91, 14.454]	0.303571
2	(14.454, 31.0]	0.454955
3	(31.0, 512.329]	0.581081

Convert the Fare feature to ordinal values based on the FareBand.

```
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]

train_df.head(10)
```

	Survived	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	0	3	0	1	0	0	1	0	3
1	1	1	1	2	3	1	3	0	2
2	1	3	1	1	1	0	2	1	3
3	1	1	1	2	3	0	3	0	2
4	0	3	0	2	1	0	1	1	6
5	0	3	0	1	1	2	1	1	3
6	0	1	0	3	3	0	1	1	3
7	0	3	0	0	2	0	4	0	0
8	1	3	1	1	1	0	3	0	3
9	1	2	1	0	2	1	3	0	0

And the test dataset.

```
test_df.head(10)
```

	PassengerId	Pclass	Sex	Age	Fare	Embarked	Title	IsAlone	Age*Class
0	892	3	0	2	0	2	1	1	6
1	893	3	1	2	0	0	3	0	6
2	894	2	0	3	1	2	1	1	6
3	895	3	0	1	1	0	1	1	3
4	896	3	1	1	1	0	3	0	3
5	897	3	0	0	1	0	1	1	0
6	898	3	1	1	0	2	2	1	3
7	899	2	0	1	2	0	1	0	2
8	900	3	1	1	0	1	3	1	3
9	901	3	0	1	2	0	1	0	3

Model, predict and solve

- here are 60+ predictive modelling algorithms to choose from.
- Our problem is a classification and regression problem
- Supervised Learning plus Classification and Regression
  - Logistic Regression
  - KNN or k-Nearest Neighbors
  - Support Vector Machines
  - Naive Bayes classifier
  - Decision Tree
  - Random Forrest
  - Perceptron
  - Artificial neural network
  - RVM or Relevance Vector Machine

```
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
```

```
((891, 8), (891,), (418, 8))
```

```
# Logistic Regression

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
```

80.36

We can use Logistic Regression to validate our assumptions and decisions for feature creating and completing goals.

- Sex is highest positivie coefficient, implying as the Sex value increases (male: 0 to female: 1), the probability of Survived=1 increases the most.
- Inversely as Pclass increases, probability of Survived=1 decreases the most.
- This way Age*Class is a good artificial feature to model as it has second highest negative correlation with Survived.
- So is Title as second highest positive correlation.

```
coeff_df = pd.DataFrame(train_df.columns.delete(0))
coeff_df.columns = ['Feature']
coeff_df["Correlation"] = pd.Series(logreg.coef_[0])

coeff_df.sort_values(by='Correlation', ascending=False)
```

	Feature	Correlation
1	Sex	2.201527
5	Title	0.398234
2	Age	0.287164
4	Embarked	0.261762
6	IsAlone	0.129140
3	Fare	-0.085150
7	Age*Class	-0.311199
0	Pclass	-0.749006

```
# Support Vector Machines

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
```

83.84

```
# KNN
knn = KNeighborsClassifier(n_neighbors = 3)
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
```

84.74

```
# Gaussian Naive Bayes

gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)
Y_pred = gaussian.predict(X_test)
acc_gaussian = round(gaussian.score(X_train, Y_train) * 100, 2)
acc_gaussian
```

72.28

```
# Perceptron

perceptron = Perceptron()
perceptron.fit(X_train, Y_train)
Y_pred = perceptron.predict(X_test)
acc_perceptron = round(perceptron.score(X_train, Y_train) * 100, 2)
acc_perceptron
```

78.0

```
# Linear SVC

linear_svc = LinearSVC()
linear_svc.fit(X_train, Y_train)
Y_pred = linear_svc.predict(X_test)
acc_linear_svc = round(linear_svc.score(X_train, Y_train) * 100, 2)
acc_linear_svc
```

79.12

```
# Stochastic Gradient Descent

sgd = SGDClassifier()
sgd.fit(X_train, Y_train)
Y_pred = sgd.predict(X_test)
acc_sgd = round(sgd.score(X_train, Y_train) * 100, 2)
acc_sgd
```

80.81

```
# Decision Tree

decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)
Y_pred = decision_tree.predict(X_test)
acc_decision_tree = round(decision_tree.score(X_train, Y_train) * 100, 2)
acc_decision_tree
```

86.76

```
# Random Forest

random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)
Y_pred = 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)
acc_random_forest
```

86.76

Model Evaluation

```
models = pd.DataFrame({
    'Model': ['Support Vector Machines', 'KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes', 'Perceptron',
              'Stochastic Gradient Decent', 'Linear SVC',
              'Decision Tree'],
    'Score': [acc_svc, acc_knn, acc_log,
              acc_random_forest, acc_gaussian, acc_perceptron,
              acc_sgd, acc_linear_svc, acc_decision_tree]})
models.sort_values(by='Score', ascending=False)
```

	Model	Score
3	Random Forest	86.76
8	Decision Tree	86.76
1	KNN	84.74
0	Support Vector Machines	83.84
6	Stochastic Gradient Decent	80.81
2	Logistic Regression	80.36
7	Linear SVC	79.12
5	Perceptron	78.00
4	Naive Bayes	72.28

```
submission = pd.DataFrame({
        "PassengerId": test_df["PassengerId"],
        "Survived": Y_pred
    })
submission.to_csv('./titanic_submission.csv', index=False)
```