Import libraries and Dataset

In [0]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import sklearn
```

In [0]:
```
train_df = pd.read_csv(os.path.join(os.getcwd(), "train.csv"))
test_df = pd.read_csv(os.path.join(os.getcwd(), "test.csv"))
```

First look at Train and Test dataset

In [0]:
```
train_df.head()
```

Out[0]:

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

In [0]:
```
#Look at statistics
train_df.describe()
```

Out[0]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch	Fare
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.000000	891.000000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594	32.204208
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057	49.693429
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000	7.910400
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000	14.454200
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000	31.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000	512.329200

In [0]:
```
test_df.head()
```

Out[0]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Emba
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	

Check null cells

Null cells at Age, Embarked, Cabin

In [0]:
```
print(train_df.isnull().any()) #check null cells

print("\nCounts of columns") #check counts of columns
train_df.count()
```

```
PassengerId    False
Survived       False
Pclass         False
Name           False
Sex            False
Age             True
SibSp          False
Parch          False
Ticket         False
Fare           False
Cabin           True
Embarked        True
dtype: bool

Counts of columns
```

Out[0]:
```
PassengerId    891
Survived       891
Pclass         891
Name           891
Sex            891
Age            714
SibSp          891
Parch          891
Ticket         891
Fare           891
Cabin          204
Embarked       889
dtype: int64
```

Check ratio of missing values

Cabin has most missing values -> Drop Cabin colum

Missing cells of Age and Embarked could be filled with most frequent value in Age and Embarked columns respectively

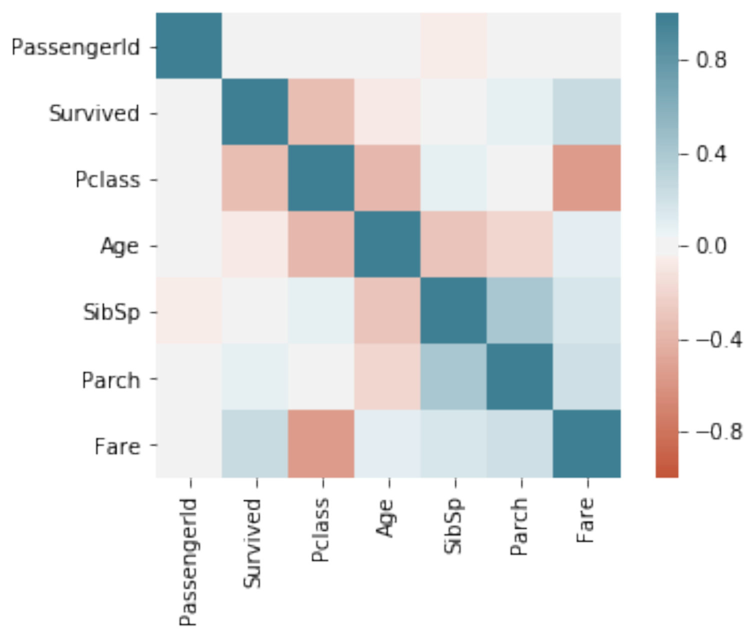
Visualize data and inspect statistics

Correlation heatmap

- Survived-Fare correlation is high -> may mean passengers who paid higher price or high er class might have higher survival rate
- Parch and Sisp is highly correlated that -> means that number of family members could include number of children. May need split children from family members
- Pclass-Fare correlation is low -> means that higher class paid more than lower class
- Pclass-Survived negative correlation -> higher class may have higher rate of survival.

In [0]:
```python
#heatmap to check correlation by pearson
data = train_df.corr(method='pearson')
sns.heatmap(data, vmin = -1, vmax = 1, center = 0, square = True, cmap
=sns.diverging_palette(20, 220, n=200),)
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f879a5fe128>



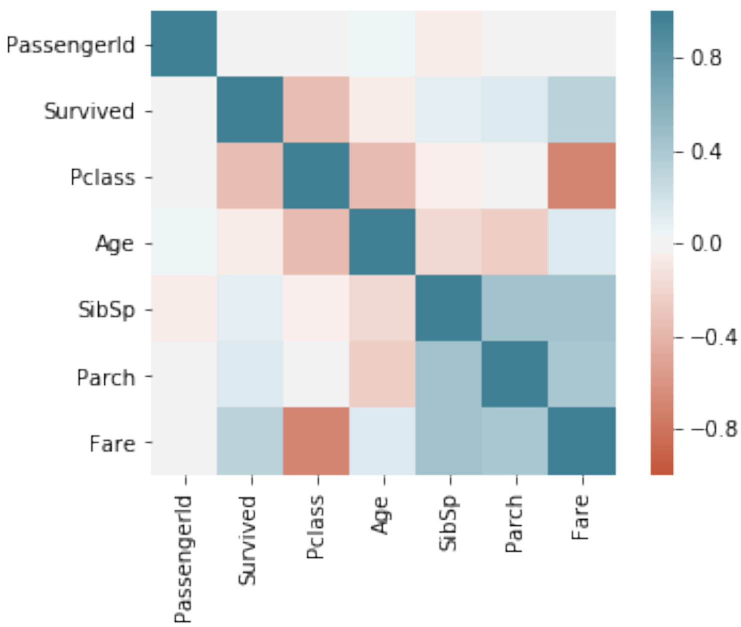
Spearman heatmap

Correlation by pearson may suffer from real values that the correlation cannot show increasing or decreasing order between features

- Agreed with Pearson that high correlation at Fare-Survived, Parch and Sibsp, Pclass-Fare
- More children, higher fare paid
- Passenger with more family members -> higher survival rate

In [0]:
```
#heatmap to check correlation by spearman
data = train_df.corr(method='spearman')
sns.heatmap(data, vmin = -1, vmax = 1, center = 0, square = True, cmap
=sns.diverging_palette(20, 220, n=200),)
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f87995b2898>



Inspect Survived and Age with Sex

- Heatmap of correlation does not tell correlation of two Survived and Age.
- Theoratically, children and elders are supposed to have higher survival rate.
- Thesis: Pclass and other factors may affect the survival rate that Age does not count greatly in the survival rate.

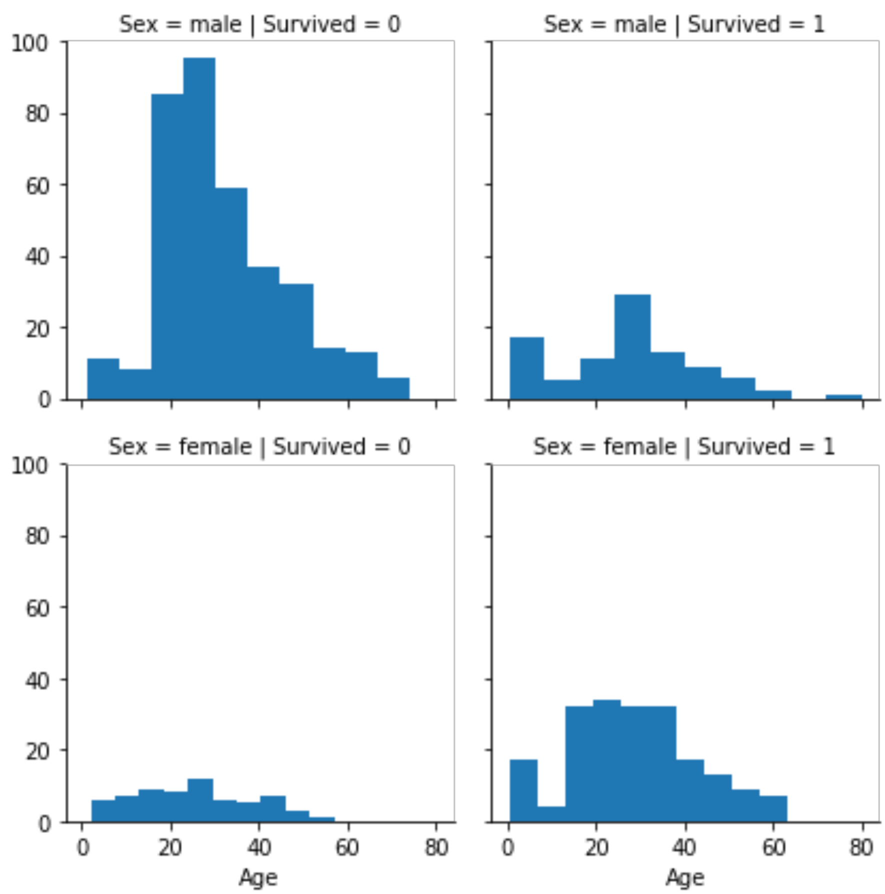
Conclude:

- Female passengers had higher rate of survival than male.
- Splitting sex, it is clear than passenegers who are under 18 has higher rate of survival, especially babies

In [0]:
```
survival_rate = train_df['Survived'][train_df['Survived'] == 1].count(
) / train_df['Survived'].count()
print("General survival rate " + str(survival_rate * 100))
grid = sns.FacetGrid(train_df, col = 'Survived', row = 'Sex')
grid.map(plt.hist, 'Age')

print(train_df[['Sex', 'Survived']].groupby(['Sex']).mean())
```

```
General survival rate 38.38383838383838
        Survived
Sex
female  0.742038
male    0.188908
```

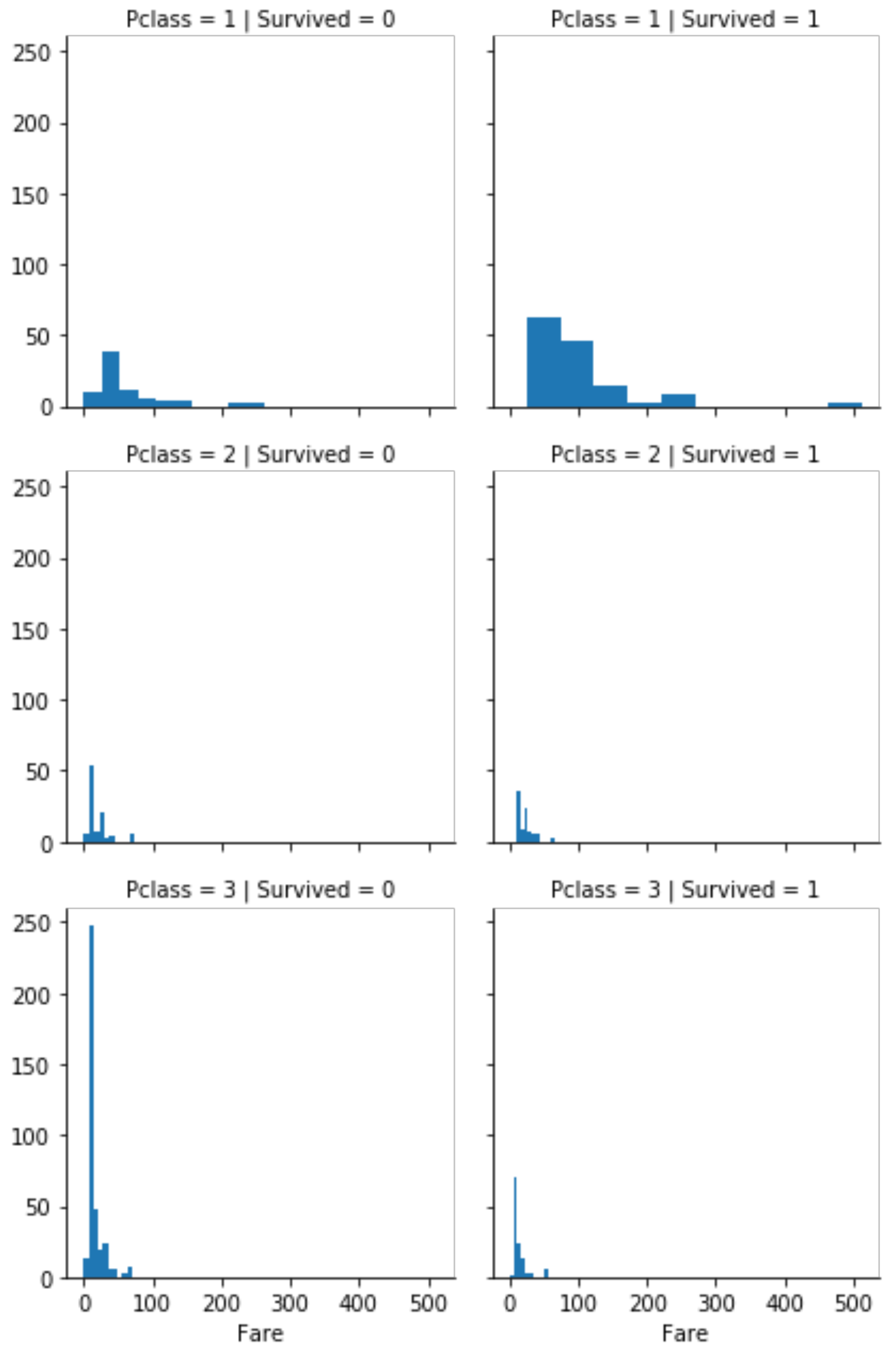


Inspect Fare and Survived

- Pclass = 1 has higher Fare
- Pclass = 1 has higher rate of survival
- Pclass 2 and 3 has lower rate of survival

In [0]:
```
grid = sns.FacetGrid(train_df, col="Survived", row = "Pclass")
grid.map(plt.hist, 'Fare')
```

Out[0]: <seaborn.axisgrid.FacetGrid at 0x7f87c467cd68>



Survival Rate of each group of Fare Perform Feature Engineering based on FareBand groups

In [0]:
```
train_df['FareBand'] = pd.qcut(train_df['Fare'], 5)
```

In [0]:
```
train_df[['FareBand', 'Survived']].groupby(['FareBand']).mean().sort_values(by='FareBand')
```

Out[0]:

	Survived
FareBand	
(-0.001, 7.854]	0.217877
(7.854, 10.5]	0.201087
(10.5, 21.679]	0.424419
(21.679, 39.688]	0.444444
(39.688, 512.329]	0.642045

Inspect Embarked and Survival

- C has highest survival rate -> C's Fare is highst
- Q and S lower survival rate -> Q and S's Fare are lower

In [0]:
```
print(train_df[['Survived', 'Embarked', 'Fare']].groupby(['Embarked']).mean())
```

```
          Survived       Fare
Embarked
C         0.553571  59.954144
Q         0.389610  13.276030
S         0.336957  27.079812
```

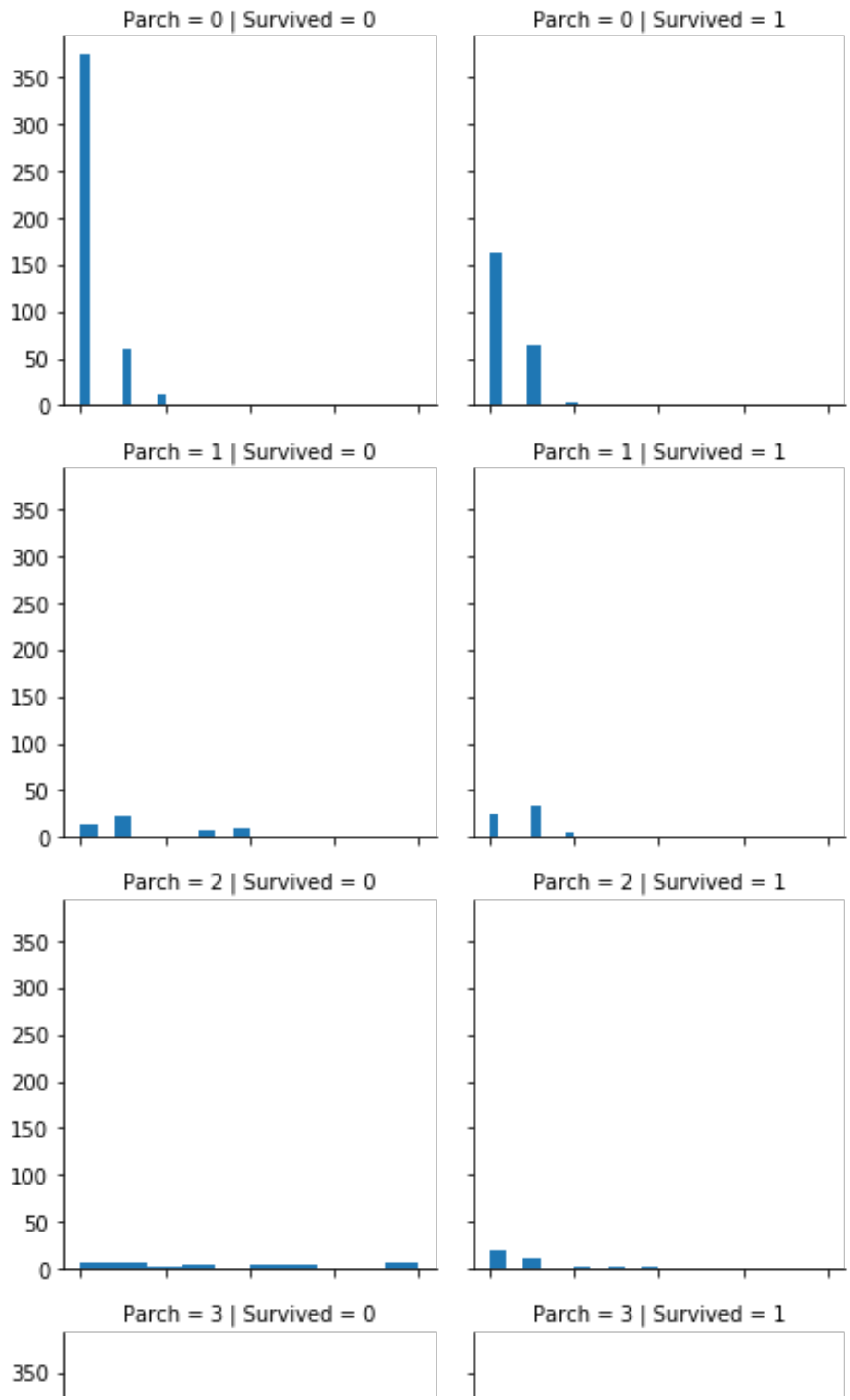
Inspect Parch and SibSp

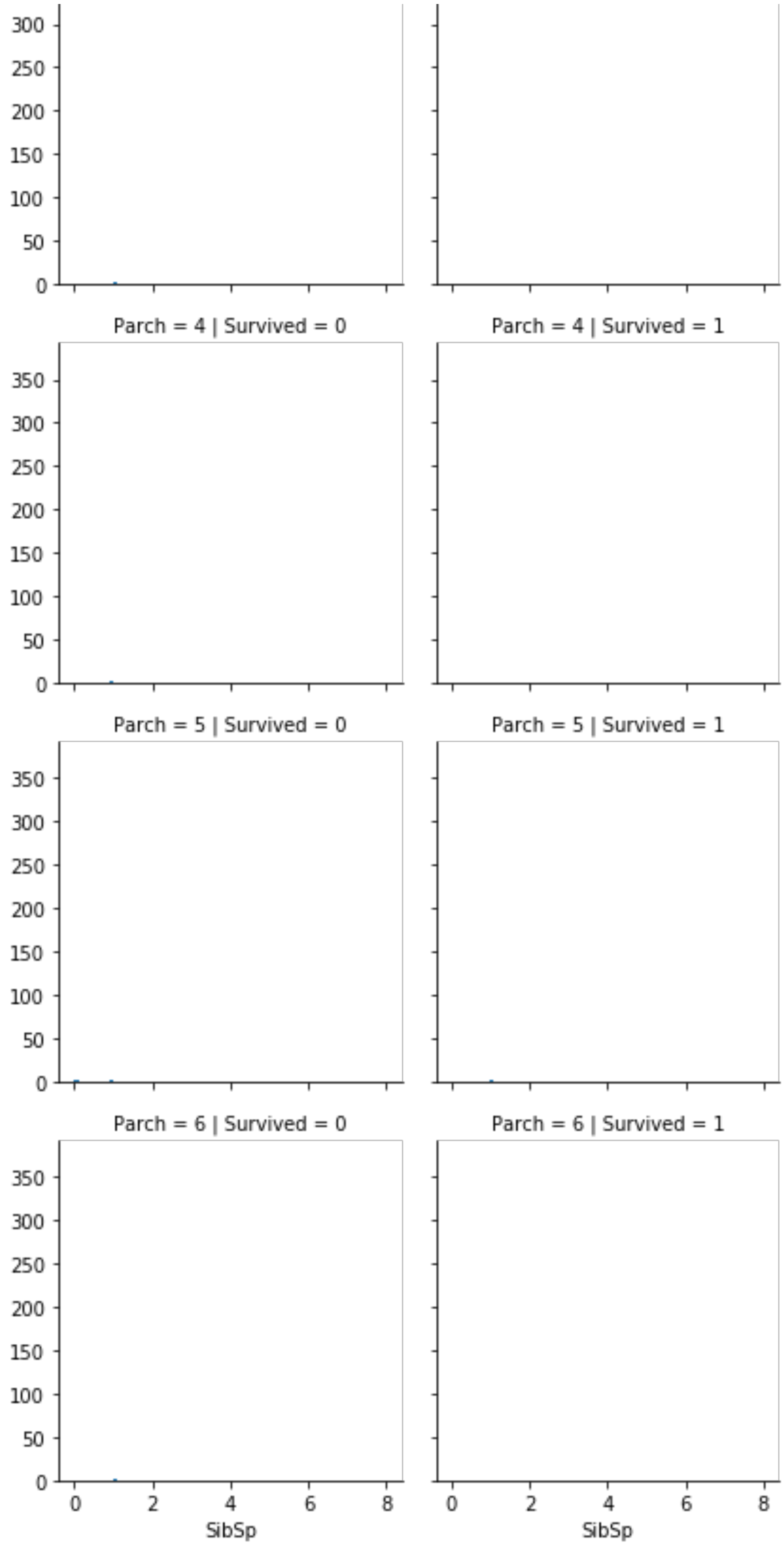
- Passengers have parents/children more than 3, it is hgiher survival rate
- Passengers have parents/children less than 3, it is lower survival rate
- Passengers have parents/children less than 3, does not have much data

In [0]:
```
grid = sns.FacetGrid(train_df, col='Survived', row='Parch')
grid.map(plt.hist, 'SibSp')

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

Parch	Survived	Age
3	0.600000	33.200000
1	0.550847	24.422000
2	0.500000	17.216912
0	0.343658	32.178503
5	0.200000	39.200000
4	0.000000	44.500000
6	0.000000	43.000000




Parch = 4 | Survived = 0
Parch = 4 | Survived = 1
Parch = 5 | Survived = 0
Parch = 5 | Survived = 1
Parch = 6 | Survived = 0
Parch = 6 | Survived = 1
SibSp
SibSp

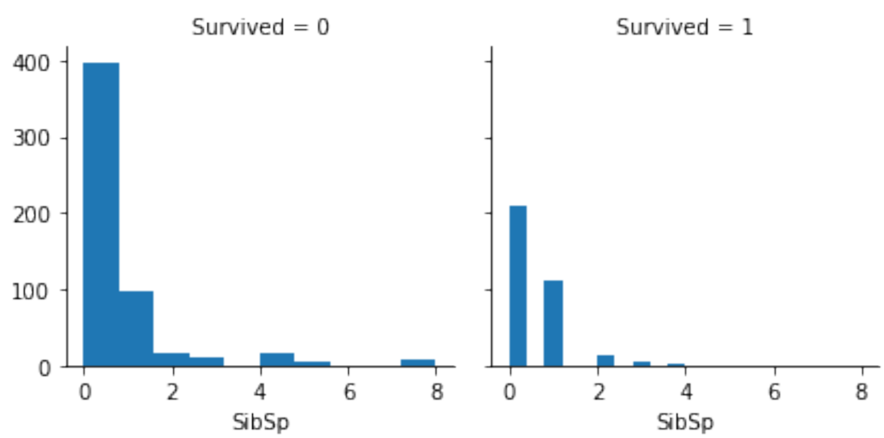
Inspect Survived and SibSp

- Passengers with <=2 siblisings -> higher survival rate
- Passengers with more than 2 siblings -> lower survival rate

In [0]:
```
print(train_df[['Survived', 'SibSp', 'Age']].groupby(['SibSp']).mean()
.sort_values(by='Survived', ascending=False))
grid = sns.FacetGrid(train_df, col='Survived')
grid.map(plt.hist, 'SibSp')
```

```
       Survived        Age
SibSp
1      0.535885  30.089727
2      0.464286  22.620000
0      0.345395  31.397558
3      0.250000  13.916667
4      0.166667   7.055556
5      0.000000  10.200000
8      0.000000        NaN
```

Out[0]: <seaborn.axisgrid.FacetGrid at 0x7f8796be2a58>

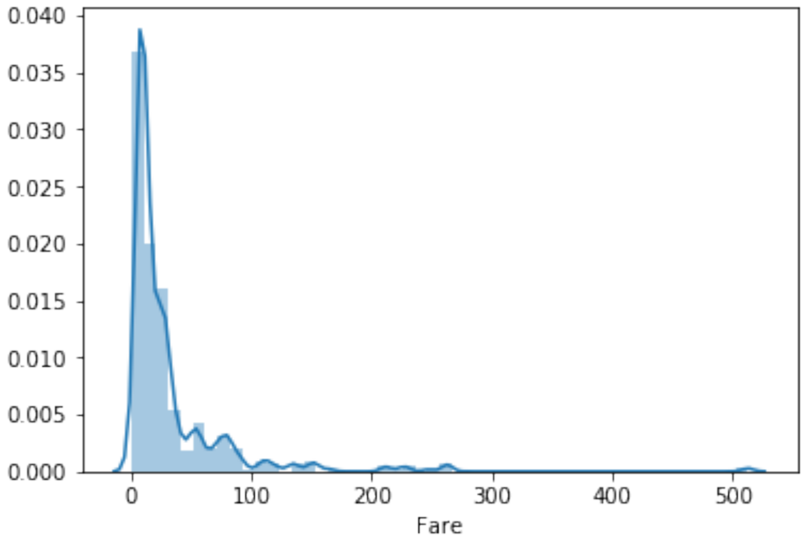


Inspecst Outliers at Fare

- Outlier detected at 500 -> Drop the row

In [0]:
```
sns.distplot(train_df['Fare'])
```

Out[0]: <matplotlib.axes._subplots.AxesSubplot at 0x7f879689a2b0>



Clean and Pre-process dataset

Make copies of train and test dataset

In [0]:
```
train = train_df
test = test_df
```

Drop PassengerId and Cabin

Assuming PassengerId does not affect to the survival rate

Cabin column has most missing cells

In [0]:
```
train = train.drop(['Cabin', 'Ticket', 'Name'], 'columns')
test = test.drop(['Cabin', 'Ticket', 'Name'], 'columns')
```

Remove outlier Fare

In [0]:
```
train = train[train['Fare'] <= 500]
```

Fill Age and Embarked columns with their most frequent values

In [0]:
```
#Find the most frequent value of age
most_age_train = train['Age'].value_counts()
most_age_test = test['Age'].value_counts()
most_age_train = most_age_train.index[0]
most_age_test = most_age_test.index[0]
```

In [0]:
```
#Fill age column
train['Age'] = train['Age'].fillna(most_age_train)
print("Age in training data")
print(train['Age'].isnull().any()) #check missing values in age
print(train.count())

print("\nAge in testing data")
test['Age'] = test['Age'].fillna(most_age_test)
print(test['Age'].isnull().any()) #check missing values in age
print(test.count())
```

```
Age in training data
False
PassengerId    888
Survived       888
Pclass         888
Sex            888
Age            888
SibSp          888
Parch          888
Fare           888
Embarked       886
FareBand       888
dtype: int64

Age in testing data
False
PassengerId    418
Pclass         418
Sex            418
Age            418
SibSp          418
Parch          418
Fare           417
Embarked       418
dtype: int64
```

In [0]:
```
#Find the most frequent value of embarked
most_embarked_train = train['Embarked'].value_counts()
print("Training Embarked\n ")
print(most_embarked_train)
most_embarked_train = most_embarked_train.index[0]

most_embarked_test = test['Embarked'].value_counts()
print("\nTesting Embarked ")
print(most_embarked_test)
most_embarked_tset = most_embarked_test.index[0]
```

```
Training Embarked

S    644
C    165
Q     77
Name: Embarked, dtype: int64

Testing Embarked
S    270
C    102
Q     46
Name: Embarked, dtype: int64
```

In [0]:
```
#Fill nan values of Embarked with most_embarked_train
print("Training Embarked")
train['Embarked'] = train['Embarked'].fillna(most_embarked_train)
print(train.isnull().any()) #check null values in Embarked
train.count()
```

```
Training Embarked
PassengerId    False
Survived       False
Pclass         False
Sex            False
Age            False
SibSp          False
Parch          False
Fare           False
Embarked       False
FareBand       False
dtype: bool
```

Out[0]:

```
PassengerId    888
Survived       888
Pclass         888
Sex            888
Age            888
SibSp          888
Parch          888
Fare           888
Embarked       888
FareBand       888
dtype: int64
```

In [0]:
```
#Fill nan values of Embarked with most_embarked_test
print("\nTesting Embarked")
test['Embarked'] = test['Embarked'].fillna(most_embarked_test)
print(test.isnull().any()) #check null values in Embarked
test.count()
```

```
Testing Embarked
PassengerId    False
Pclass         False
Sex            False
Age            False
SibSp          False
Parch          False
Fare            True
Embarked       False
dtype: bool
```

Out[0]:
```
PassengerId    418
Pclass         418
Sex            418
Age            418
SibSp          418
Parch          418
Fare           417
Embarked       418
dtype: int64
```

Fill Fare of Test

Observations on train shows that fare and embarked is closely related to each other -> Fill null Fare cells with median Fare of corresponding Embarked

In [0]:
```
#meadian fare of corresponding embarked
test_fare = test[['Fare', 'Embarked']].groupby(['Embarked']).median()
test['Fare'] = test.apply((lambda x: test_fare.loc[x['Embarked']]['Far
e'] if pd.isna(x['Fare']) else x['Fare']), axis=1)
```

Finall check on train and test dataset

In [0]:
```
print("Train\n" + str(train.count()))
print("Test\n" + str(test.count()))
train.head()
```

```
Train
PassengerId    888
Survived       888
Pclass         888
Sex            888
Age            888
SibSp          888
Parch          888
Fare           888
Embarked       888
FareBand       888
dtype: int64
Test
PassengerId    418
Pclass         418
Sex            418
Age            418
SibSp          418
Parch          418
Fare           418
Embarked       418
dtype: int64
```

Out[0]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	FareBand
0	1	0	3	male	22.0	1	0	7.2500	S	(-0.001, 7.854]
1	2	1	1	female	38.0	1	0	71.2833	C	(39.688, 512.329]
2	3	1	3	female	26.0	0	0	7.9250	S	(7.854, 10.5]
3	4	1	1	female	35.0	1	0	53.1000	S	(39.688, 512.329]
4	5	0	3	male	35.0	0	0	8.0500	S	(7.854, 10.5]

Convert categorical features to numerical features

- Sex -> Binary
- Emabrked -> Classes

In [0]:
```
#Sex to binary
train['Sex'] = train['Sex'].map({'male': 0, 'female':1})
test['Sex'] = test['Sex'].map({'male':0, 'female':1}).astype(int)
train.head()
```

Out[0]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	FareBand
0	1	0	3	0	22.0	1	0	7.2500	S	(-0.001, 7.854]
1	2	1	1	1	38.0	1	0	71.2833	C	(39.688, 512.329]
2	3	1	3	1	26.0	0	0	7.9250	S	(7.854, 10.5]
3	4	1	1	1	35.0	1	0	53.1000	S	(39.688, 512.329]
4	5	0	3	0	35.0	0	0	8.0500	S	(7.854, 10.5]

In [0]:
```
#convert embarked
train['Embarked'] = train['Embarked'].map({'S':1, 'Q':2, 'C':3}).astyp
e(int)
test['Embarked'] = test['Embarked'].map({'S':1, 'Q':2, 'C':3}).astype(
int)
test.head()
```

Out[0]:

	PassengerId	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	892	3	0	34.5	0	0	7.8292	2
1	893	3	1	47.0	1	0	7.0000	1
2	894	2	0	62.0	0	0	9.6875	2
3	895	3	0	27.0	0	0	8.6625	1
4	896	3	1	22.0	1	1	12.2875	1

In [0]:
```
train.head()
```

Out[0]:

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	FareBand
0	1	0	3	0	22.0	1	0	7.2500	1	(-0.001, 7.854]
1	2	1	1	1	38.0	1	0	71.2833	3	(39.688, 512.329]
2	3	1	3	1	26.0	0	0	7.9250	1	(7.854, 10.5]
3	4	1	1	1	35.0	1	0	53.1000	1	(39.688, 512.329]
4	5	0	3	0	35.0	0	0	8.0500	1	(7.854, 10.5]

Conclusion

Traning data includes Survived, Sex, Age, Pclass, Embarked, Fare, Parck, SibSp

In [0]:
```
train_X = train.drop(['Survived', 'FareBand', 'PassengerId'], axis='co
lumns')
train_Y = train['Survived']
test_X = test.drop(['PassengerId'], axis='columns')
```

In [0]:
```
from sklearn.model_selection import train_test_split
#train_X, val_X, train_Y, val_Y = train_test_split(train_X, train_Y, t
est_size = 1/10, random_state = 0, shuffle = True)
```

Predictive Models

- Logistic Regression
- K-Nearest Neighbors
- SVM
- LinearSVC
- Naive Bayes
- Random Forest

Logistic Regression

In [0]:
```
from sklearn.linear_model import LogisticRegression
logistic_model = LogisticRegression(random_state = 0, solver='liblinea
r', verbose = 1).fit(train_X, train_Y)
pred = logistic_model.predict(test_X)
train_log = round(logistic_model.score(train_X, train_Y) * 100, 2)
#val_log = round(logistic_model.score(val_X, val_Y) * 100, 2)
print("Train Acc: " + str(train_log))
#print("Val Acc: " + str(val_log))
```

```
[LibLinear]Train Acc: 79.95
```

KNeighborsClassifier

In [0]:
```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors = 3).fit(train_X, train_Y
)
pred = knn_model.predict(test_X)
train_log = round(knn_model.score(train_X, train_Y) * 100, 2)
#val_log = round(knn_model.score(val_X, val_Y) * 100, 2)
print("Train Acc: " + str(train_log))
#print("Val Acc: " + str(val_log))
```

```
Train Acc: 84.23
```

SVM

In [0]:
```
#SVC
from sklearn import svm
sigmoid_svc = svm.SVC(gamma = 'auto', random_state = 0).fit(train_X, t
rain_Y)
pred = sigmoid_svc.predict(test_X)
train_log = round(sigmoid_svc.score(train_X, train_Y) * 100, 2)
#val_log = round(sigmoid_svc.score(val_X, val_Y) * 100, 2)
print("Train Acc: " + str(train_log))
#print("Val Acc: " + str(val_log))

linear_svc = svm.LinearSVC(random_state = 0, tol=1e-5, max_iter = 1000
).fit(train_X, train_Y)
pred = linear_svc.predict(test_X)
train_log = round(linear_svc.score(train_X, train_Y) * 100, 2)
#val_log = round(linear_svc.score(val_X, val_Y) * 100, 2)
print("Train Acc: " + str(train_log))
#print("Val Acc: " + str(val_log))
```

```
Train Acc: 90.2
Train Acc: 75.0
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/svm/base.py:929: Conv
ergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
```

Navie Bayes

In [0]:
```
from sklearn.naive_bayes import GaussianNB
gaussiannb = GaussianNB().fit(train_X, train_Y)
pred = gaussiannb.predict(test_X)
train_log = round(gaussiannb.score(train_X, train_Y) * 100, 2)
#val_log = round(gaussiannb.score(val_X, val_Y) * 100, 2)
print("Train Acc: " + str(train_log))
#print("Val Acc: " + str(val_log))
```

```
Train Acc: 78.94
```

Decision Tree

In [0]:
```
from sklearn.tree import DecisionTreeClassifier
decisiontree_model = DecisionTreeClassifier(criterion='entropy', rando
m_state = 0).fit(train_X, train_Y)
pred = decisiontree_model.predict(test_X)
train_log = decisiontree_model.score(train_X, train_Y) * 100
#val_log = decisiontree_model.score(val_X, val_Y) * 100
print("Train Acc: " + str(train_log))
#print("Val Acc: " + str(val_log))
```

```
Train Acc: 98.08558558558559
```

In [0]:
```
len(pred)
```

Out[0]:
```
418
```

In [0]:
```
#submission = pd.DataFrame(columns = ['PassengerId', 'Survived'])
#id = pd.DataFrame([range(1, 418)], columns = 'PassengerId')
submission = pd.DataFrame()
submission['PassengerId'] = test['PassengerId']
submission['Survived'] = pred
#submission = pd.DataFrame(pred, columns = ['Survived'])
#submission['PassengerId'] = range(1, len(pred) + 1)
```

In [0]:
```
submission.head()
submission.to_csv("titanic_prediction.csv", index=False)
```

In [0]:
```
res = pd.read_csv('titanic_prediction.csv')
```

In [0]:
```
res.head()
```

Out[0]:

	PassengerId	Survived
0	892	0
1	893	0
2	894	1
3	895	1
4	896	1

Random Forest

In [0]:
```
from sklearn.ensemble import RandomForestClassifier
randomforest_model = RandomForestClassifier(criterion = 'entropy', ran
dom_state = 0, ).fit(train_X, train_Y)
pred = randomforest_model.predict(test_X)
train_log = randomforest_model.score(train_X, train_Y) * 100
#val_log = randomforest_model.score(val_X, val_Y) * 100
print("Train Acc: " + str(train_log))
#print("Val Acc: " + str(val_log))
```

```
Train Acc: 95.83333333333334

/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:24
5: FutureWarning: The default value of n_estimators will change from
10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

Conclusion => Overfitting

Feature Engineering

In [0]:
```
train.head()
```

Out[0]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	FareBand
0	0	3	0	22.0	1	0	7.2500	1	(-0.001, 7.854]
1	1	1	1	38.0	1	0	71.2833	3	(39.688, 512.329]
2	1	3	1	26.0	0	0	7.9250	1	(7.854, 10.5]
3	1	1	1	35.0	1	0	53.1000	1	(39.688, 512.329]
4	0	3	0	35.0	0	0	8.0500	1	(7.854, 10.5]

In [0]:
```
test.describe()
```

Out[0]:

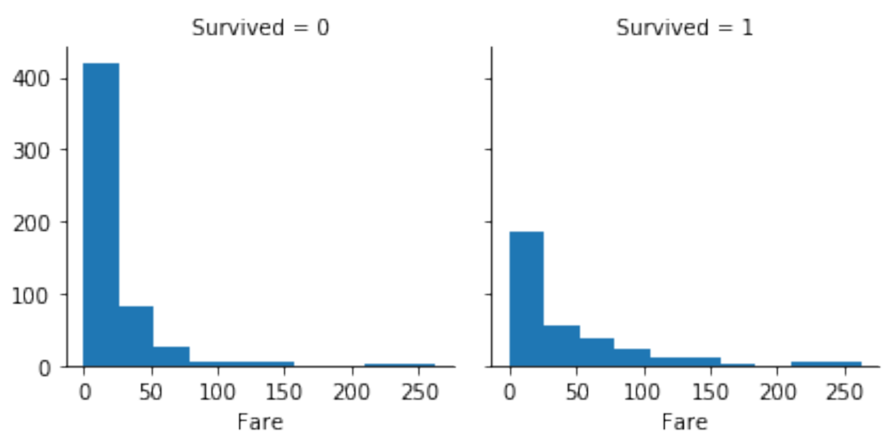
	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
count	418.000000	418.000000	418.000000	418.000000	418.000000	418.000000	418.000000
mean	2.265550	0.363636	28.982057	0.447368	0.392344	35.574911	1.598086
std	0.841838	0.481622	12.887063	0.896760	0.981429	55.850729	0.854496
min	1.000000	0.000000	0.170000	0.000000	0.000000	0.000000	1.000000
25%	1.000000	0.000000	23.000000	0.000000	0.000000	7.895800	1.000000
50%	3.000000	0.000000	24.000000	0.000000	0.000000	14.454200	1.000000
75%	3.000000	1.000000	35.750000	1.000000	0.000000	31.471875	2.000000
max	3.000000	1.000000	76.000000	8.000000	9.000000	512.329200	3.000000

Convert Fare into Categorical Features

- Fare is evenly distributed for Survived = 0 or 1
- Fare varies from 0 to 250 => Large variance => Categorize: based on FareBand

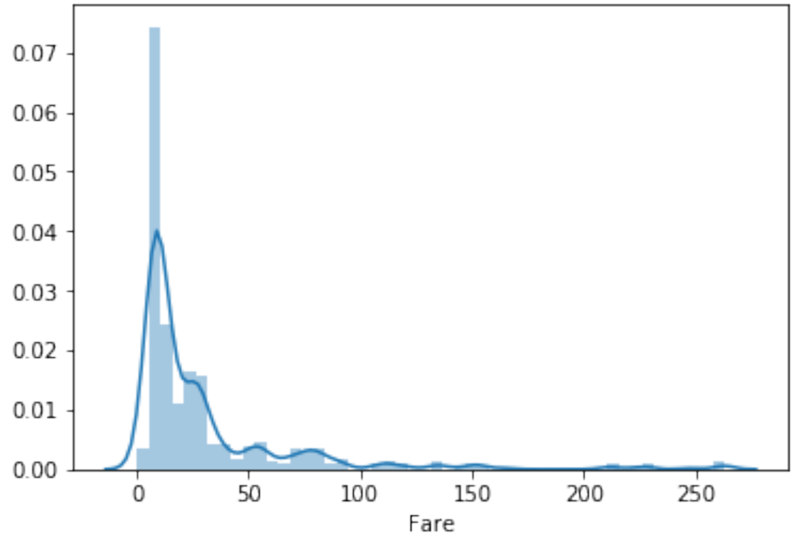
In [0]:
```
#understand distribution of Fare
grid = sns.FacetGrid(train, col='Survived')
grid.map(plt.hist, 'Fare')
```

Out[0]: <seaborn.axisgrid.FacetGrid at 0x7f06dffaa048>



In [0]:
```
sns.distplot(train['Fare'])
```

Out[0]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f06dffaa128>
```



In [0]:
```
train['FareBand'] = pd.qcut(train['Fare'], 5)
```

In [0]:
```
train[['FareBand', 'Survived']].groupby(['FareBand']).mean().sort_valu
es(by='FareBand')
```

Out[0]:

	Survived
FareBand	
(-0.001, 7.854]	0.217877
(7.854, 10.5]	0.201087
(10.5, 21.075]	0.426901
(21.075, 39.688]	0.441989
(39.688, 263.0]	0.635838

In [0]:
```
def categorize_fare(fare):
  if fare <= 7.854:
    return 0
  elif fare <= 10.5:
    return 1
  elif fare <= 21.08:
    return 3
  elif fare <= 39.69:
    return 4
  else:
    return 5
train['Fare'] = train['Fare'].apply(lambda x: categorize_fare(x))
test['Fare'] = test['Fare'].apply(lambda x: categorize_fare(x))
```

In [0]:
```
train['Fare'].value_counts()
```

Out[0]:
```
1    197
4    181
5    173
3    171
0    166
Name: Fare, dtype: int64
```

Replace Parch and SibSp with Family_member

In [0]:
```
train['Family'] = train.apply(lambda x: x['Parch'] + x['SibSp'], axis=
'columns')
test['Family'] = test.apply(lambda x: x['Parch'] + x['SibSp'], axis='c
olumns')
```

In [0]:
```
train[['Family', 'Survived']].groupby(['Family']).mean().sort_values('
Family')
```

Out[0]:

Family	Survived
0	0.300935
1	0.550000
2	0.578431
3	0.724138
4	0.200000
5	0.136364
6	0.333333
7	0.000000
10	0.000000

Passenger with Family = 0, 6 -> Survival Rate = 30-33% Passenger with Family = 1, 2 -> Survival Rate = 50% Passenger with Family = 3 -> Survival Rate = 70% Passenger with Family = 4,5 -> Survival Rate = 20% Passenger with Family = 7, 10 -> Survival Rate = 0%

In [0]:
```
#Categorize Family for train data
train.loc[train['Family'] == 0, 'FamilyBand'] = 0
train.loc[train['Family'] == 6, 'FamilyBand'] = 0
train.loc[train['Family'] == 1, 'FamilyBand'] = 1
train.loc[train['Family'] == 2, 'FamilyBand'] = 1
train.loc[train['Family'] == 3, 'FamilyBand'] = 3
train.loc[train['Family'] == 4, 'FamilyBand'] = 4
train.loc[train['Family'] == 5, 'FamilyBand'] = 4
train.loc[train['Family'] == 7, 'FamilyBand'] = 5
train.loc[train['Family'] == 10, 'FamilyBand'] = 5

#Categorize Family for test data
test.loc[test['Family'] == 0, 'FamilyBand'] = 0
test.loc[test['Family'] == 6, 'FamilyBand'] = 0
test.loc[test['Family'] == 1, 'FamilyBand'] = 1
test.loc[test['Family'] == 2, 'FamilyBand'] = 1
test.loc[test['Family'] == 3, 'FamilyBand'] = 3
test.loc[test['Family'] == 4, 'FamilyBand'] = 4
test.loc[test['Family'] == 5, 'FamilyBand'] = 4
test.loc[test['Family'] == 7, 'FamilyBand'] = 5
test.loc[test['Family'] == 10, 'FamilyBand'] = 5
```

In [0]:
```
train['FamilyBand'] = train['FamilyBand'].astype(int)
test['FamilyBand'] = test['FamilyBand'].astype(int)
```

Check data

In [0]:
```
test.head()
```

Out[0]:

	Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked	FareBand	Family	FamilyBand
0	0	3	0	22.0	1	0	0	1	(-0.001, 7.854]	1	1
1	1	1	1	38.0	1	0	5	3	(39.688, 263.0]	1	1
2	1	3	1	26.0	0	0	1	1	(7.854, 10.5]	0	0
3	1	1	1	35.0	1	0	5	1	(39.688, 263.0]	1	1
4	0	3	0	35.0	0	0	1	1	(7.854, 10.5]	0	0

Conclusion

Traning data includes Survived, Sex, Age, Pclass, Embarked, Fare, Parck, SibSp

In [0]:
```
train_X = train.drop(['Survived', 'Parch', 'SibSp', 'FareBand', 'Famil
y'], axis='columns')
train_Y = train['Survived']
test_X = test.drop(['Parch', 'SibSp', 'Family'], axis='columns')
```

In [0]:
```
#Check train and test
train_X.head()
```

Out[0]:

	Pclass	Sex	Age	Fare	Embarked	FamilyBand
0	3	0	22.0	0	1	1
1	1	1	38.0	5	3	1
2	3	1	26.0	1	1	0
3	1	1	35.0	5	1	1
4	3	0	35.0	1	1	0

In [0]:
```
from sklearn.model_selection import train_test_split
train_X, val_X, train_Y, val_Y = train_test_split(train_X, train_Y, te
st_size = 1/10, random_state = 0, shuffle = True)
```

Predictive Models

Logistic Regression

In [0]:
```
logistic_model = LogisticRegression(random_state = 0, solver='liblinea
r', verbose = 1).fit(train_X, train_Y)
pred = logistic_model.predict(test_X)
train_log = round(logistic_model.score(train_X, train_Y) * 100, 2)
val_log = round(logistic_model.score(val_X, val_Y) * 100, 2)
print("Train Acc: " + str(train_log))
print("Val Acc: " + str(val_log))
```

```
[LibLinear]Train Acc: 81.1
Val Acc: 71.91
```

KNN

In [0]:

```
from sklearn.neighbors import KNeighborsClassifier
knn_model = KNeighborsClassifier(n_neighbors = 3).fit(train_X, train_Y
)
pred = knn_model.predict(test_X)
train_log = round(knn_model.score(train_X, train_Y) * 100, 2)
val_log = round(knn_model.score(val_X, val_Y) * 100, 2)
print("Train Acc: " + str(train_log))
print("Val Acc: " + str(val_log))
```

```
Train Acc: 86.86
Val Acc: 71.91
```

SVM

In [0]:

```
#SVC
from sklearn import svm
sigmoid_svc = svm.SVC(gamma = 'auto', random_state = 0).fit(train_X, t
rain_Y)
pred = sigmoid_svc.predict(test_X)
train_log = round(sigmoid_svc.score(train_X, train_Y) * 100, 2)
val_log = round(sigmoid_svc.score(val_X, val_Y) * 100, 2)
print("Train Acc: " + str(train_log))
print("Val Acc: " + str(val_log))

linear_svc = svm.LinearSVC(random_state = 0, tol=1e-5, max_iter = 1000
).fit(train_X, train_Y)
pred = linear_svc.predict(test_X)
train_log = round(linear_svc.score(train_X, train_Y) * 100, 2)
val_log = round(linear_svc.score(val_X, val_Y) * 100, 2)
print("Train Acc: " + str(train_log))
print("Val Acc: " + str(val_log))
```

```
Train Acc: 87.98
Val Acc: 70.79
Train Acc: 79.22
Val Acc: 70.79

/usr/local/lib/python3.6/dist-packages/sklearn/svm/base.py:929: Conv
ergenceWarning: Liblinear failed to converge, increase the number of
iterations.
  "the number of iterations.", ConvergenceWarning)
```

Naiv Bayes

In [0]:
```
from sklearn.naive_bayes import GaussianNB
gaussiannb = GaussianNB().fit(train_X, train_Y)
pred = gaussiannb.predict(test_X)
train_log = round(gaussiannb.score(train_X, train_Y) * 100, 2)
val_log = round(gaussiannb.score(val_X, val_Y) * 100, 2)
print("Train Acc: " + str(train_log))
print("Val Acc: " + str(val_log))
```

```
Train Acc: 78.72
Val Acc: 69.66
```

Decision Tree

In [0]:
```
from sklearn.tree import DecisionTreeClassifier
decisiontree_model = DecisionTreeClassifier(criterion='entropy', rando
m_state = 0).fit(train_X, train_Y)
pred = decisiontree_model.predict(test_X)
train_log = decisiontree_model.score(train_X, train_Y) * 100
val_log = decisiontree_model.score(val_X, val_Y) * 100
print("Train Acc: " + str(train_log))
print("Val Acc: " + str(val_log))
```

```
Train Acc: 94.86858573216522
Val Acc: 75.28089887640449
```

Random Forest

In [0]:
```
from sklearn.ensemble import RandomForestClassifier
randomforest_model = RandomForestClassifier(criterion = 'entropy', ran
dom_state = 0, ).fit(train_X, train_Y)
pred = randomforest_model.predict(test_X)
train_log = randomforest_model.score(train_X, train_Y) * 100
val_log = randomforest_model.score(val_X, val_Y) * 100
print("Train Acc: " + str(train_log))
print("Val Acc: " + str(val_log))
```

```
Train Acc: 93.99249061326658
Val Acc: 79.7752808988764
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:24
5: FutureWarning: The default value of n_estimators will change from
10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
```

In [0]: