The Titanic Dataset

What is the Titanic dataset?

Known as the "Unsinkable ship" the Titanic was the worlds most advanced ship of it's da

- download the data here

For this tutorial, we want to get our hands dirty ASAP. Let's see what happen

Table of Contents:
1. Import libraries
2. load data for e/ csv file

Part 1: Import libraries

[1]
```
'''Imagine you're fixing your TV. You need tools right?
   Python has a large of body of opensource tools built and maintained
   All of these words below are basically the tools you need to fix you
   All you need to do is to import them into your notebook so you can u
   I know... You can't import money... yet.
   You will use these tools often while on your journey to creating hig
'''
from sklearn.metrics import make_scorer, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn import preprocessing
import matplotlib.pylab as pylab
import matplotlib.pyplot as plt
from pandas import get_dummies
import matplotlib as mpl
import xgboost as xgb
```

```
import seaborn as sns
import pandas as pd
import numpy as np
import matplotlib
import warnings
import sklearn
import scipy
import numpy
import json
import sys
import csv
import os
```

Scatter Plot settings

scatter plots are a way of expressing datapoints you have in your dataset.

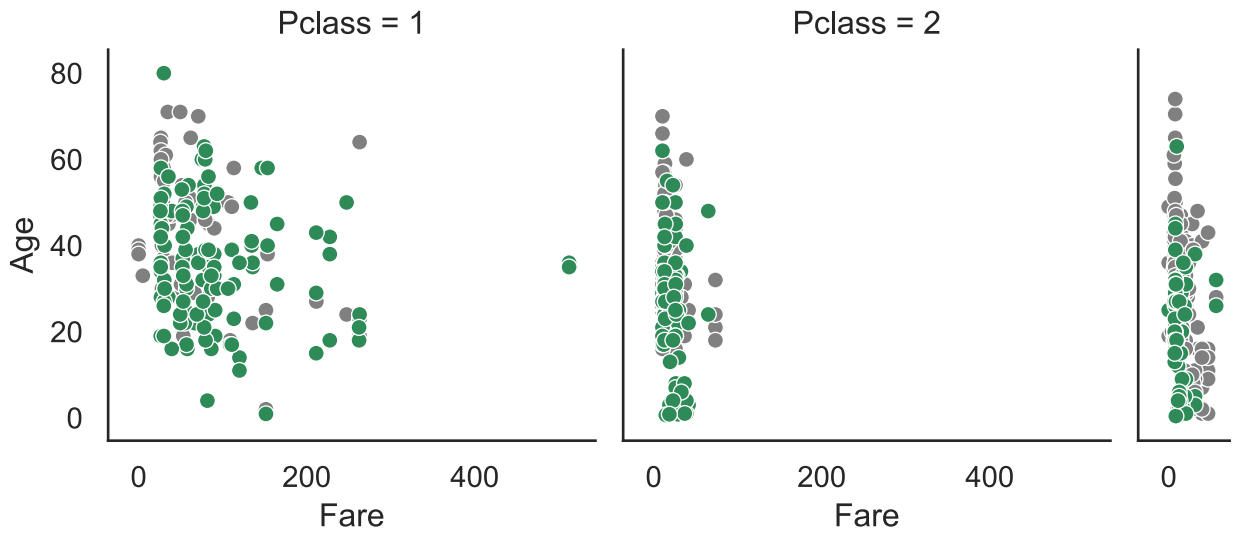
Scatter plots allow you to Identify the type of relationship (if any) between two quantitative va

[6]
```
sns.set(style='white', context='notebook', palette='deep')
pylab.rcParams['figure.figsize'] = 12,8
warnings.filterwarnings('ignore')
mpl.style.use('ggplot')
sns.set_style('white')
%matplotlib inline
```

Connect your notebook to your data (create a new folder if you haven't alread

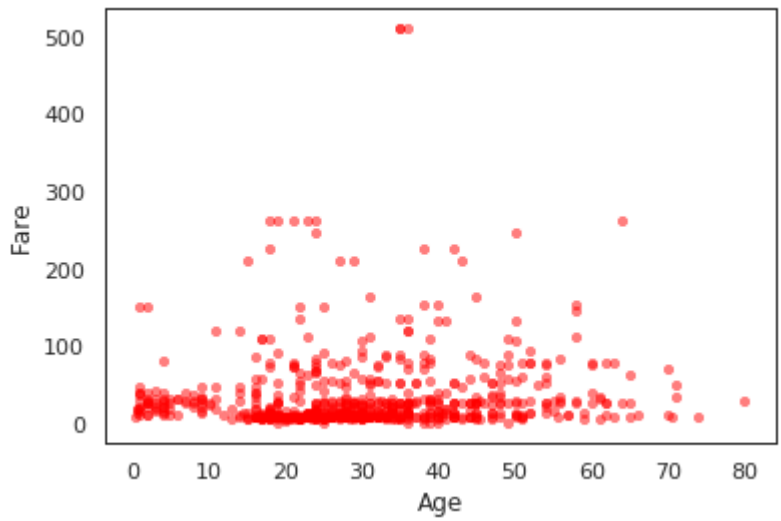
[4]
```
# we match our dataset like this:
# train = what you want to predict with
# test = the actual outcome of who died
df_train = pd.read_csv('titanic/train.csv')
df_test = pd.read_csv('titanic/test.csv')
```

[5]
```
g = sns.FacetGrid(df_train, hue='Survived', col='Pclass', margin_titles
                  palette={1:"seagreen", 0:'gray'})
g=g.map(plt.scatter, "Fare", 'Age', edgecolor='w').add_legend();
```

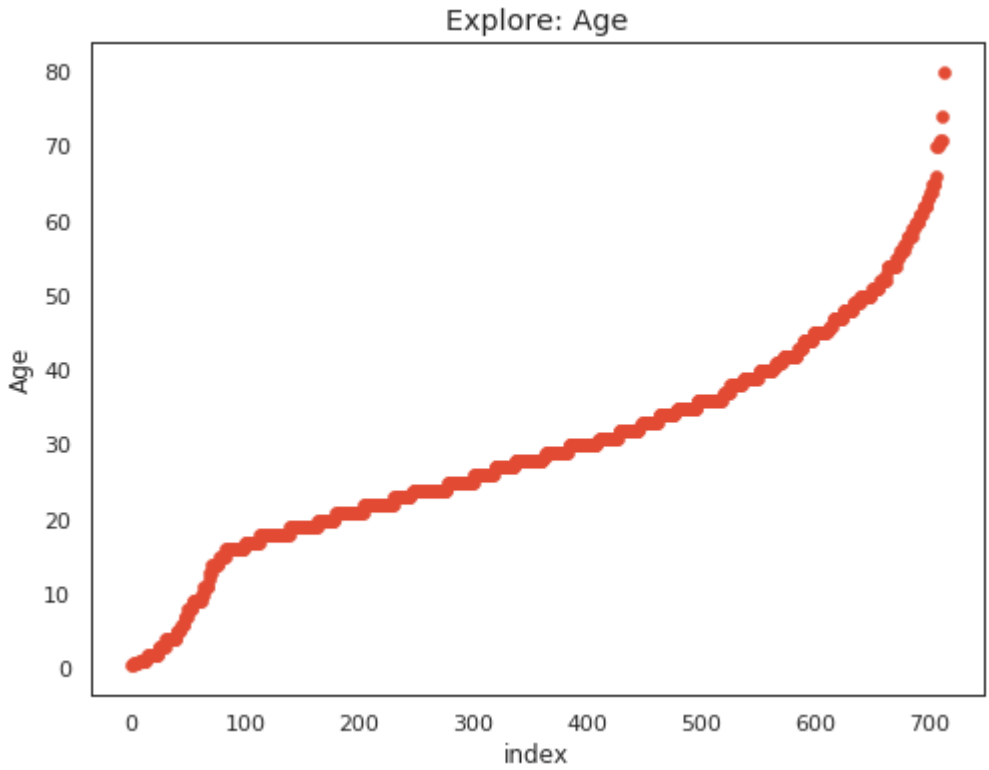


[5]
```
df_train.plot(kind = 'scatter', x='Age', y='Fare', alpha=.5, color='red
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f18ca6be080>
```



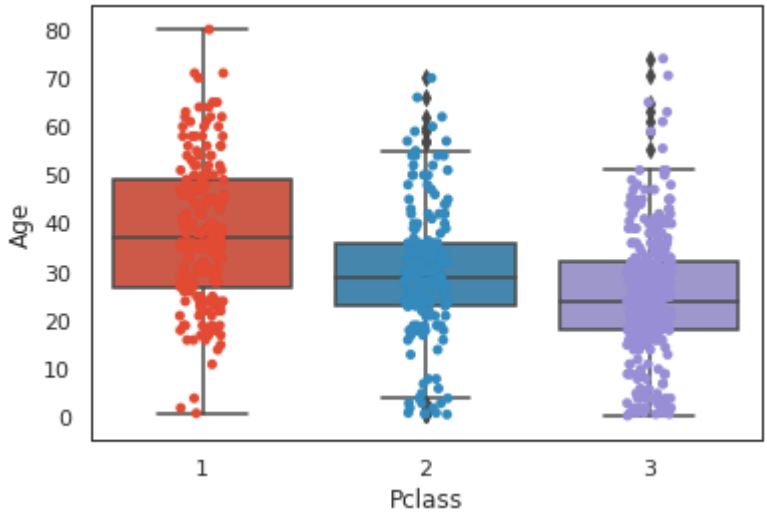
[6]
```
#show scatter with mpl
plt.figure(figsize=(8,6))
plt.scatter(range(df_train.shape[0]), np.sort(df_train['Age'].values))
plt.xlabel("index")
plt.ylabel("Age")
plt.title('Explore: Age')
plt.show()
```



Box Plot

depict groups of numberical data through quartiles.

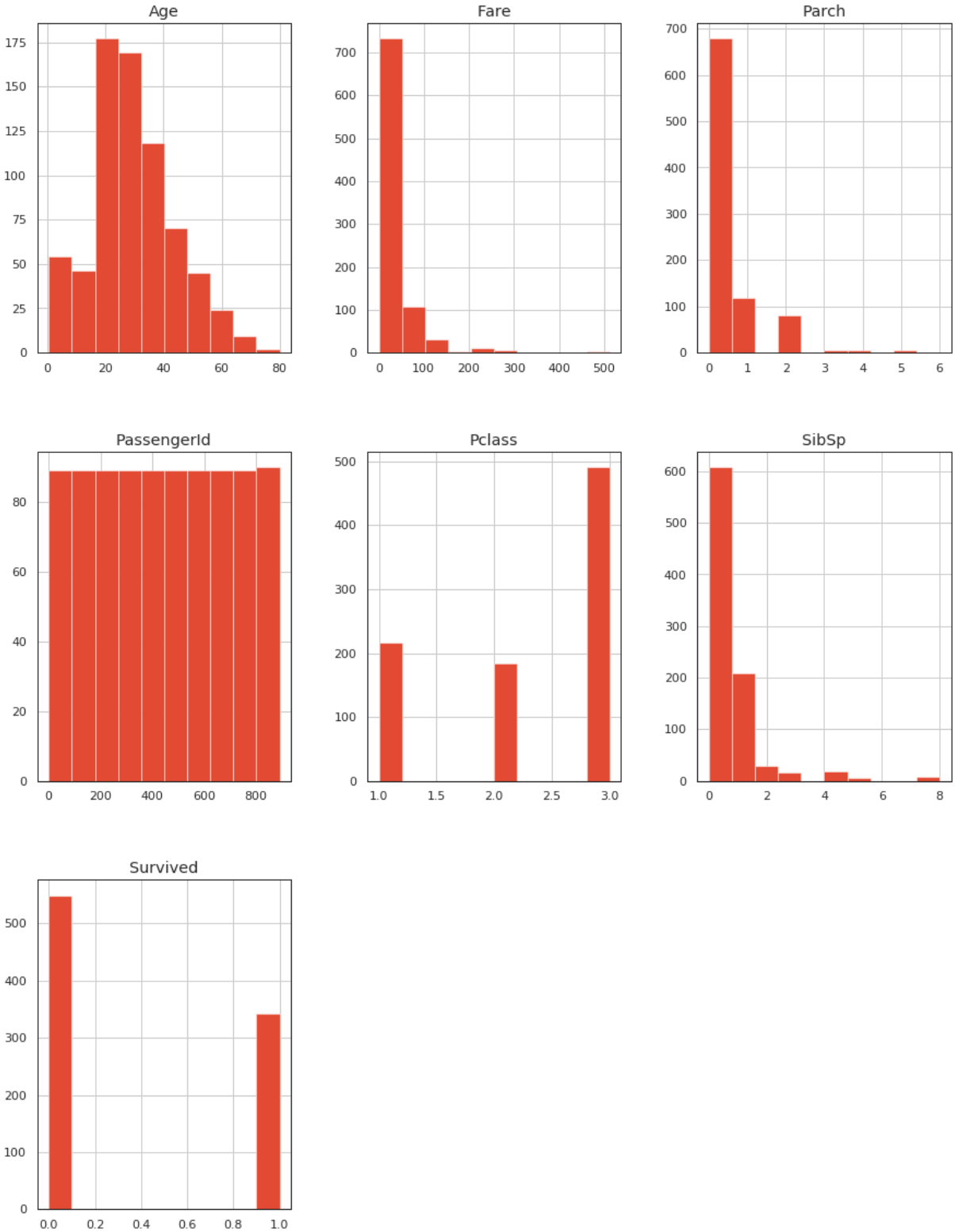
```
[7]  ax= sns.boxplot(x="Pclass", y="Age", data=df_train)
     ax= sns.stripplot(x='Pclass', y='Age', data=df_train, jitter=True, edge
     plt.show()
```



Histogram

Gives an idea of what the distribution is

[8]
```
df_train.hist(figsize=(15, 20))
plt.figure();
```

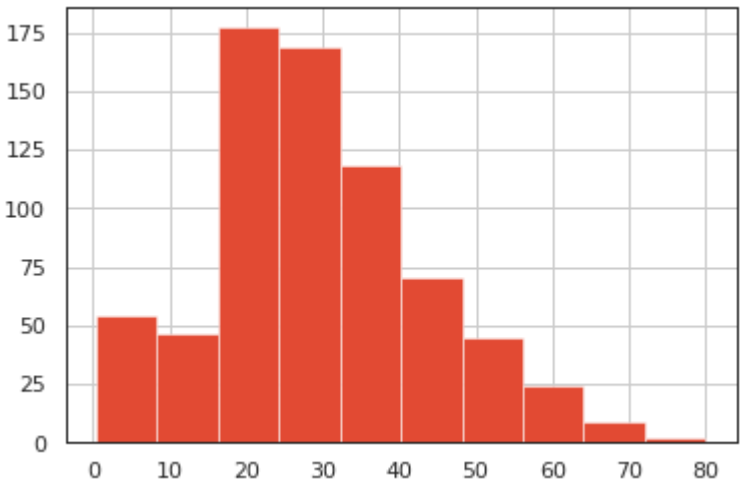



```
<Figure size 432x288 with 0 Axes>
```

Two of the variables have a Gaussian distribution, let's check.

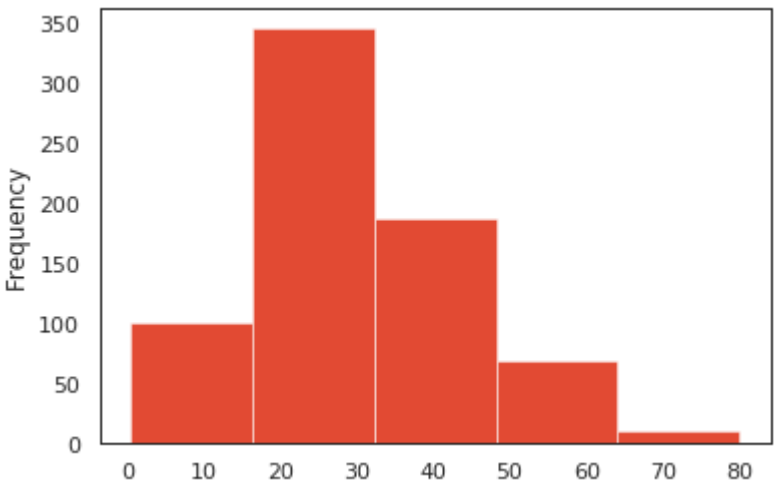
[9]
```
df_train['Age'].hist()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f18c9d13e10>
```



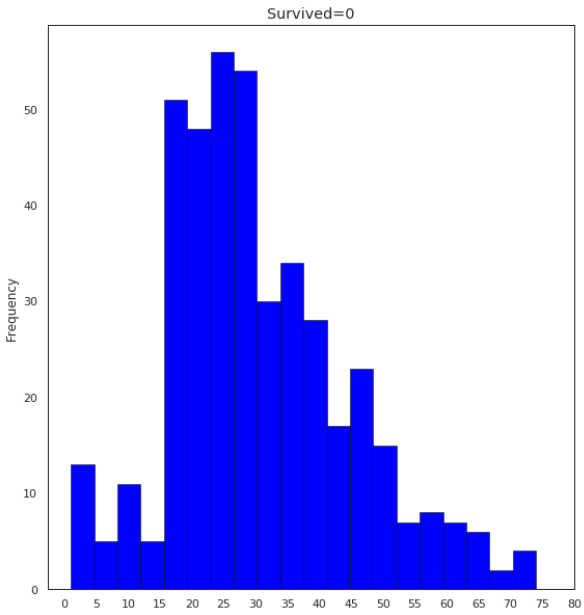
[10]
```
df_train.Age.plot(kind='hist', bins=5)
```

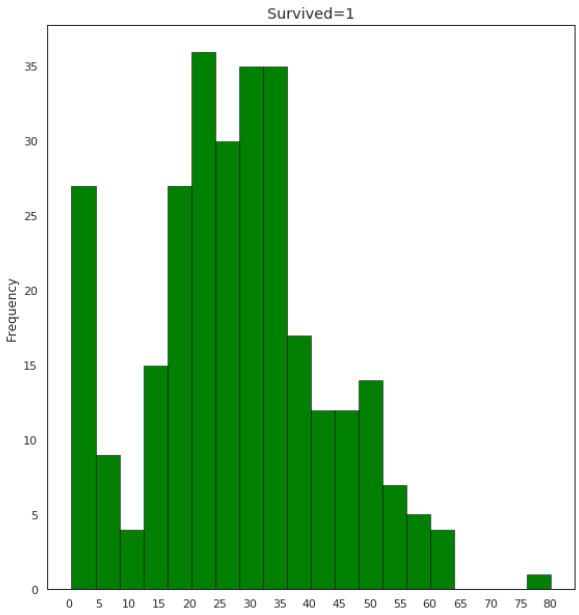
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f18ca319f60>
```



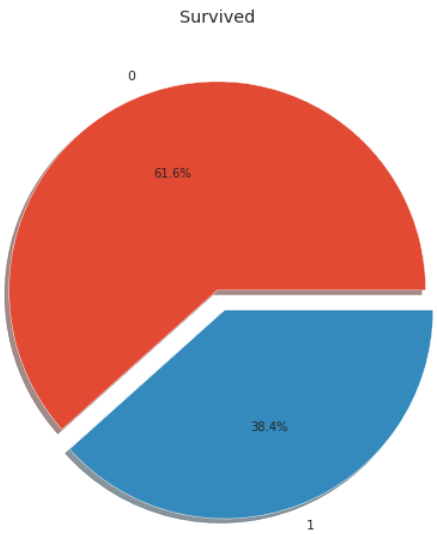
[11]
```
f, ax=plt.subplots(1,2, figsize=(20,10))
df_train[df_train['Survived']==0].Age.plot.hist(ax=ax[0],bins=20, edgec
ax[0].set_title('Survived=0')
x1=list(range(0,85,5))
ax[0].set_xticks(x1)
df_train[df_train['Survived'] ==1].Age.plot.hist(ax=ax[1], color='green
```

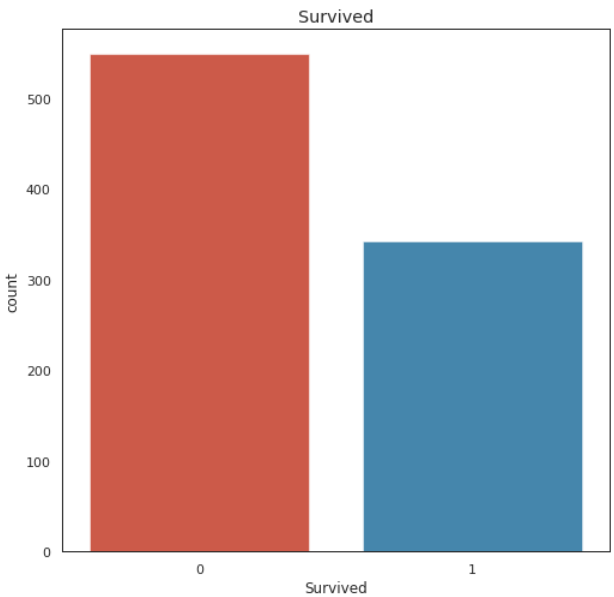
```
    ax[1].set_title('Survived=1')
    x2=list(range(0,85,5))
    ax[1].set_xticks(x2)
    plt.show()
```





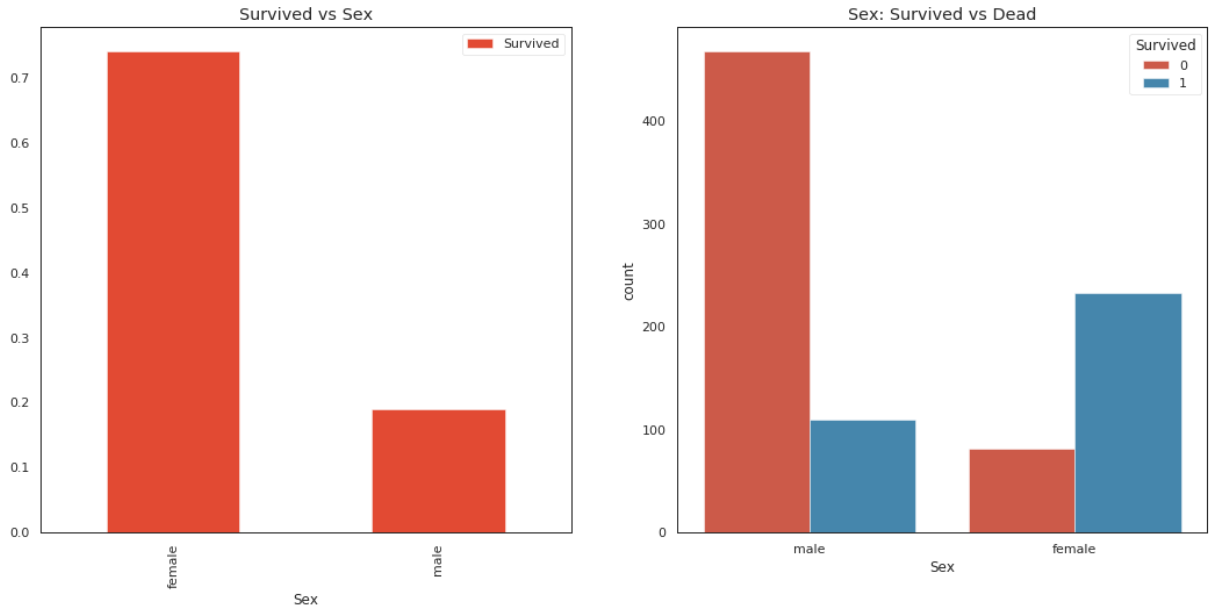
[12]
```
    f, ax=plt.subplots(1,2,figsize=(18,8))
    df_train['Survived'].value_counts().plot.pie(explode=[0,0.1],autopct='%
    ax[0].set_title('Survived')
    ax[0].set_ylabel('')
    sns.countplot('Survived', data=df_train, ax=ax[1])
    ax[1].set_title('Survived')
    plt.show()
```



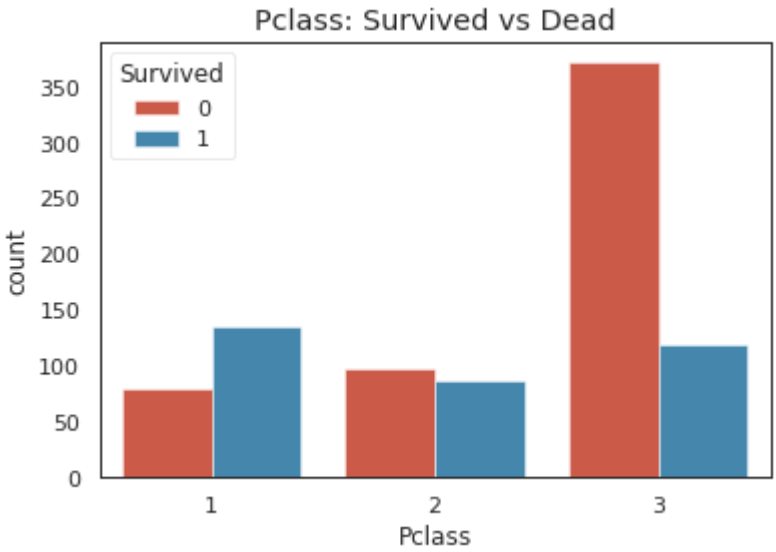


[13]
```
    f, ax=plt.subplots(1,2, figsize=(18, 8))
    df_train[['Sex', 'Survived']].groupby(['Sex']).mean().plot.bar(ax=ax[0]
    ax[0].set_title('Survived vs Sex')
```

```
sns.countplot('Sex', hue='Survived', data=df_train,ax=ax[1])
ax[1].set_title('Sex: Survived vs Dead')
plt.show()
```



[14]
```
sns.countplot('Pclass', hue='Survived', data=df_train)
plt.title('Pclass: Survived vs Dead')
plt.show()
```



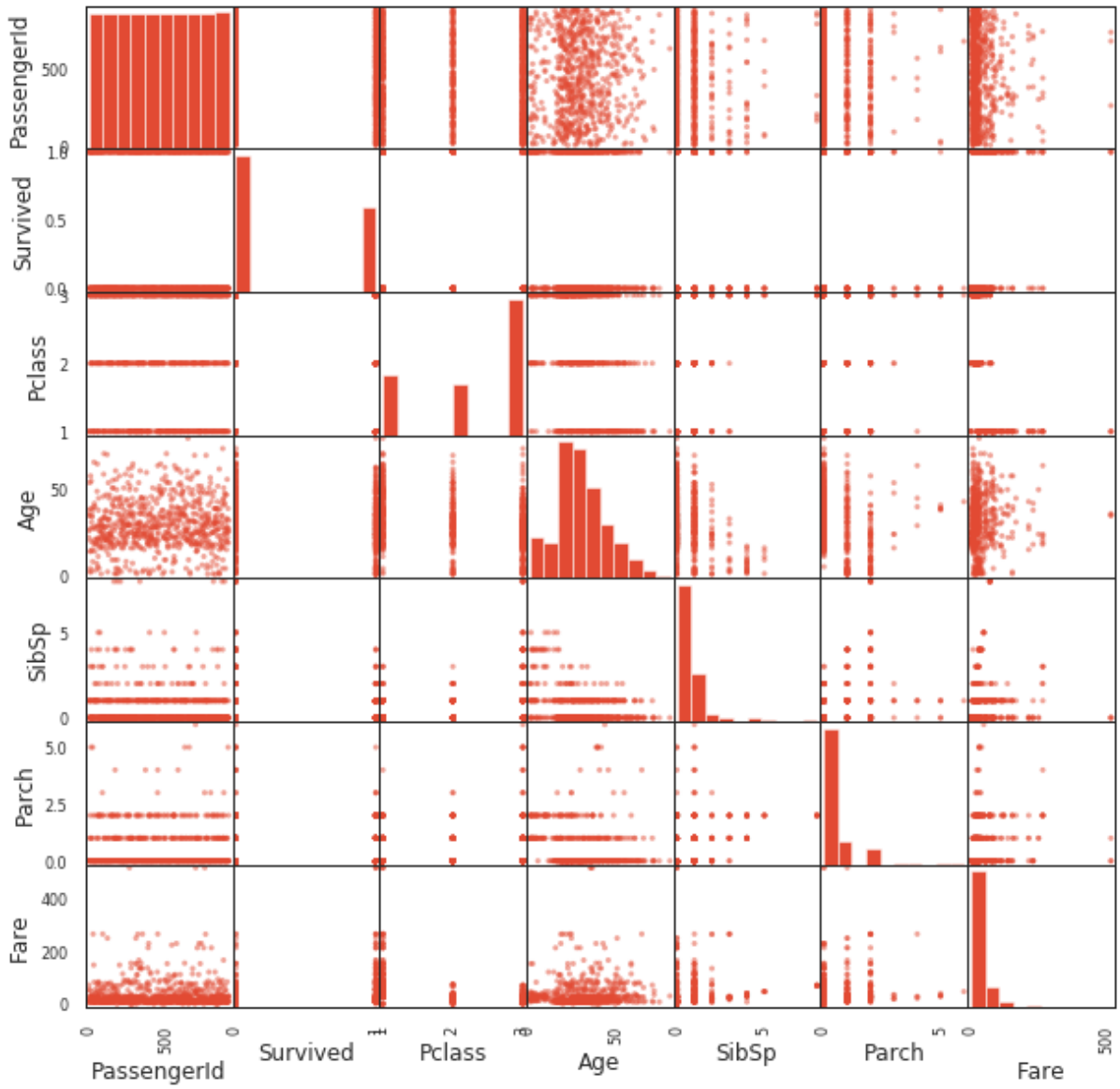
Multivariate Plots

Look at interactions between the variables

Good for spotting structured relationshios between inpiut variables

[15]
```
#scatter plot matrix
pd.plotting.scatter_matrix(df_train, figsize=(10,10))
plt.figure()
```

```
<Figure size 432x288 with 0 Axes>
```



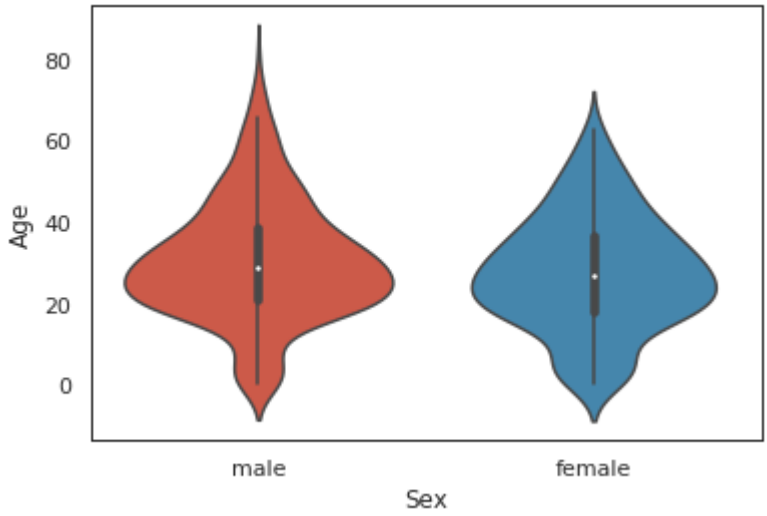

```
<Figure size 432x288 with 0 Axes>
```

Violinplots

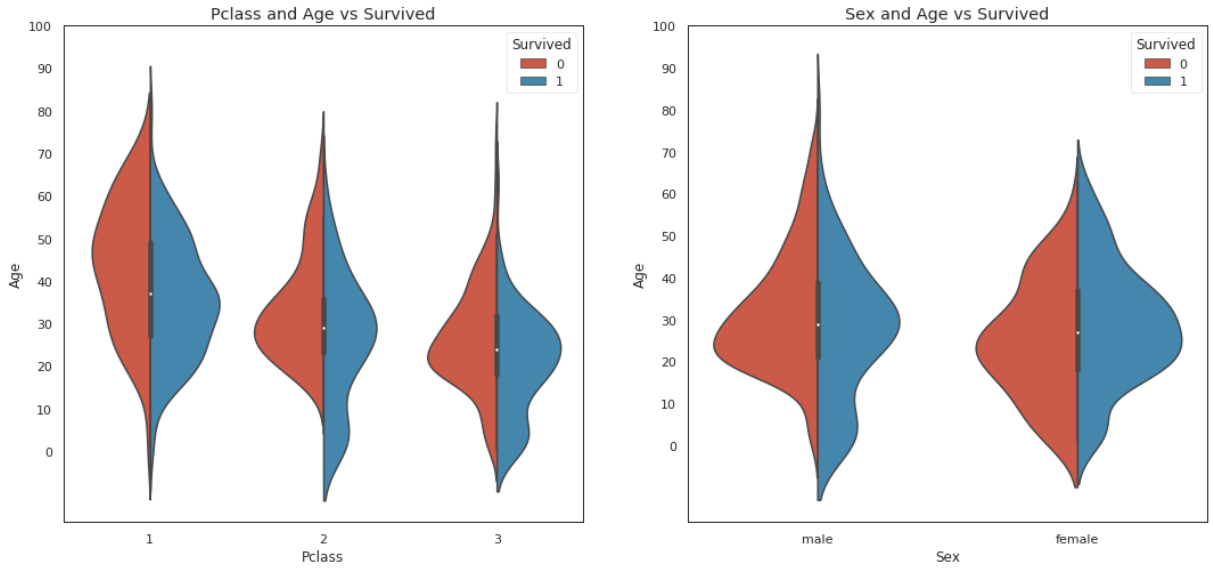
Shows distribution of quantitative data across several levels of one (or more) categorical varia

[16]
```
sns.violinplot(data=df_train, x='Sex', y='Age')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f18c983edd8>
```



[17]
```
f, ax = plt.subplots(1, 2, figsize=(18, 8))
sns.violinplot("Pclass", "Age", hue='Survived', data=df_train, split=Tr
ax[0].set_title('Pclass and Age vs Survived')
ax[0].set_yticks(range(0, 110, 10))
sns.violinplot("Sex", "Age", hue='Survived', data=df_train, split=True,
ax[1].set_title('Sex and Age vs Survived')
ax[1].set_yticks(range(0,110, 10))
plt.show()
```



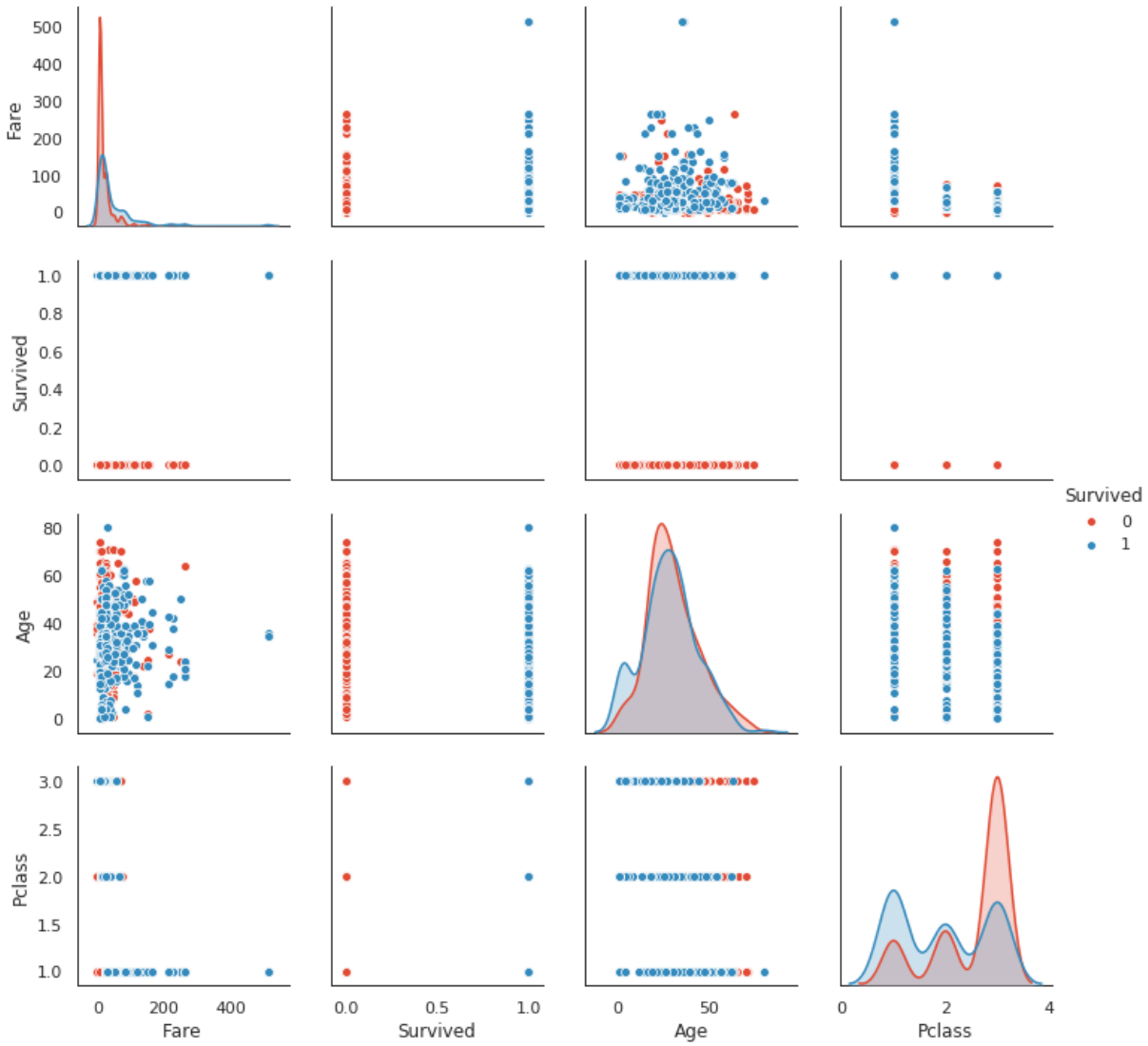
Pairplot

[18]
```
# use seaborn pairplot to see the bivariate relationship between each p

sns.pairplot(data=df_train[["Fare", "Survived", "Age", "Pclass"]],
```

```
                hue= "Survived", dropna=True)
```

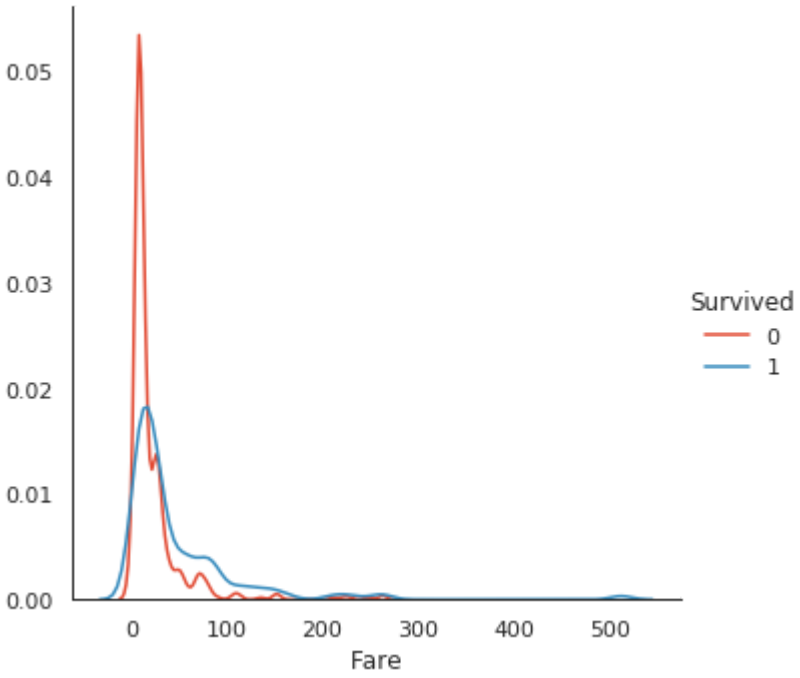
```
<seaborn.axisgrid.PairGrid at 0x7f18c98f9320>
```



KDEplot

replace histograms shows in the diagonal of the pairplot by kde

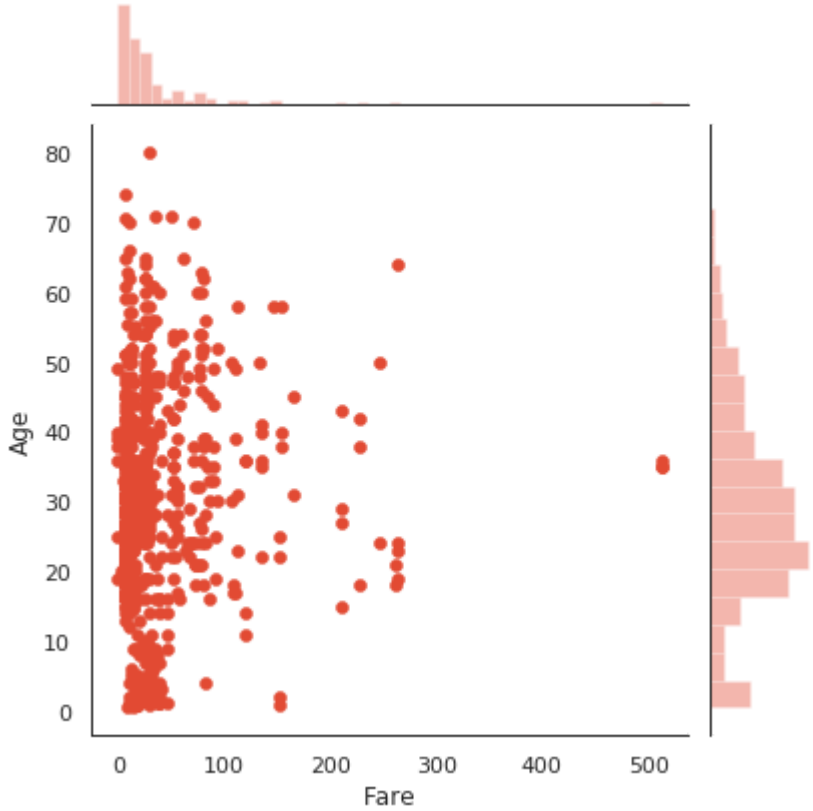
```
[19]  sns.FacetGrid(df_train, hue='Survived', size=5).map(sns.kdeplot, "Fare"
      plt.show()
```



Jointplot

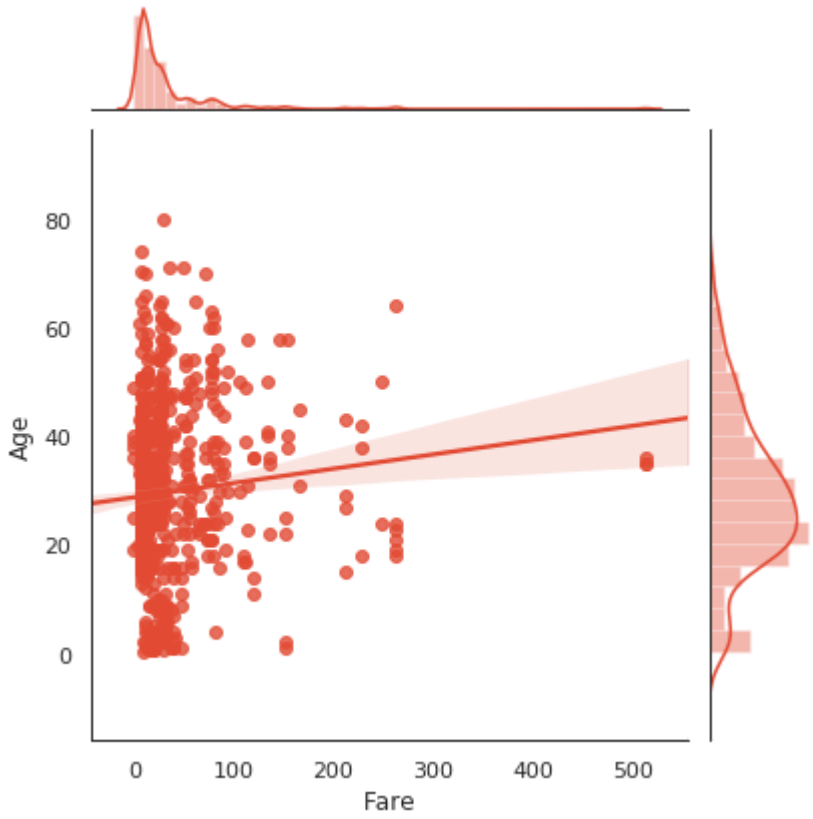
[20]
```
sns.jointplot(x='Fare', y='Age', data=df_train)
```

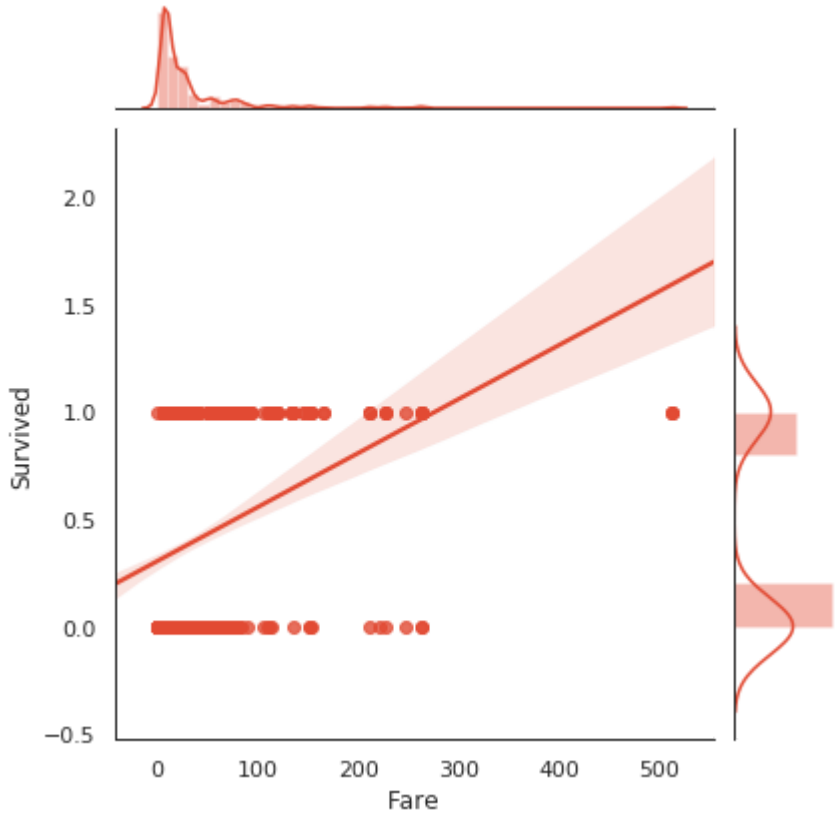
```
<seaborn.axisgrid.JointGrid at 0x7f18ca583ba8>
```



[21]
```
sns.jointplot(x='Fare', y='Age', data=df_train, kind='reg')
sns.jointplot(x='Fare', y='Survived', data=df_train, kind='reg')
```

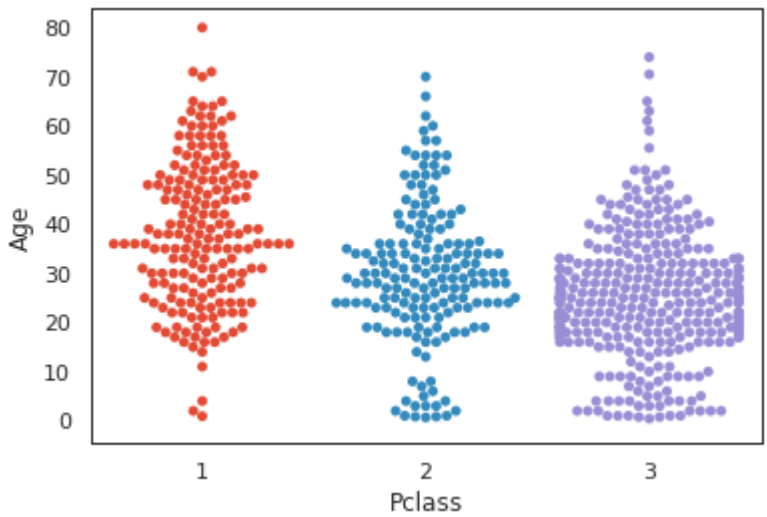
```
<seaborn.axisgrid.JointGrid at 0x7f18c993e5f8>
```





[22]
```
sns.swarmplot(x='Pclass', y='Age', data=df_train)
```

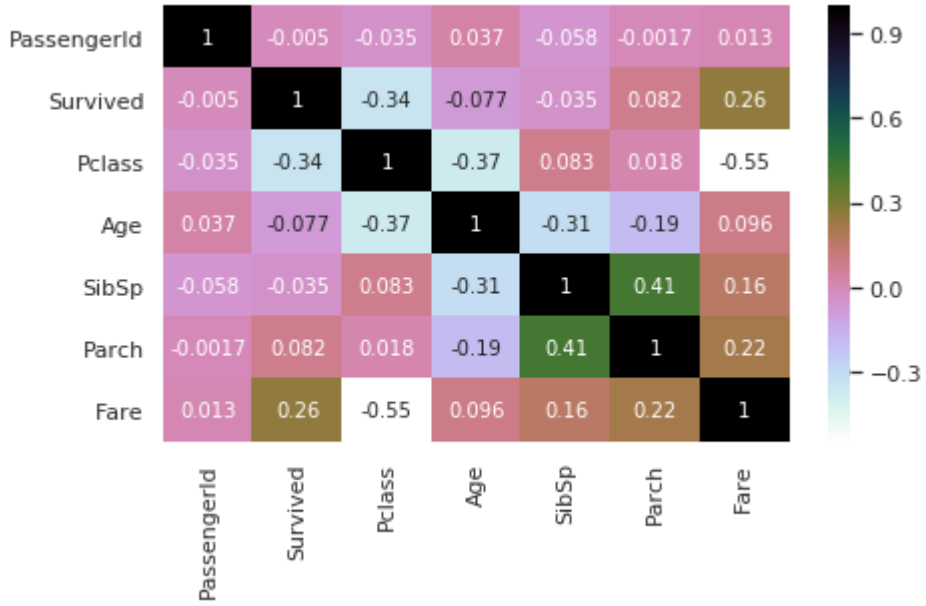
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f18ca3fbdd8>
```



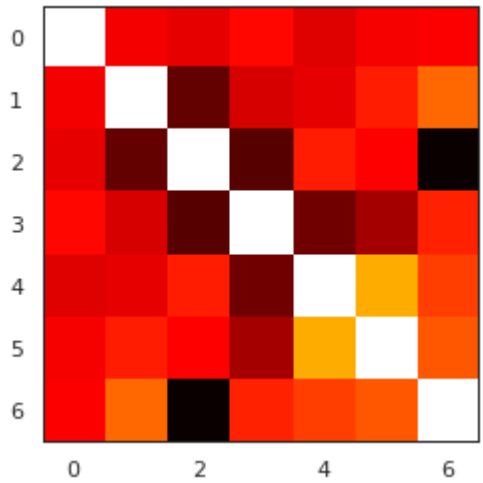
Heatmap

A heatmap is a two-dimensional graphgical representation of data where the indiviudla value

[23]
```
plt.figure(figsize=(7, 4))
sns.heatmap(df_train.corr(), annot=True, cmap='cubehelix_r')#draws heat
plt.show()
```



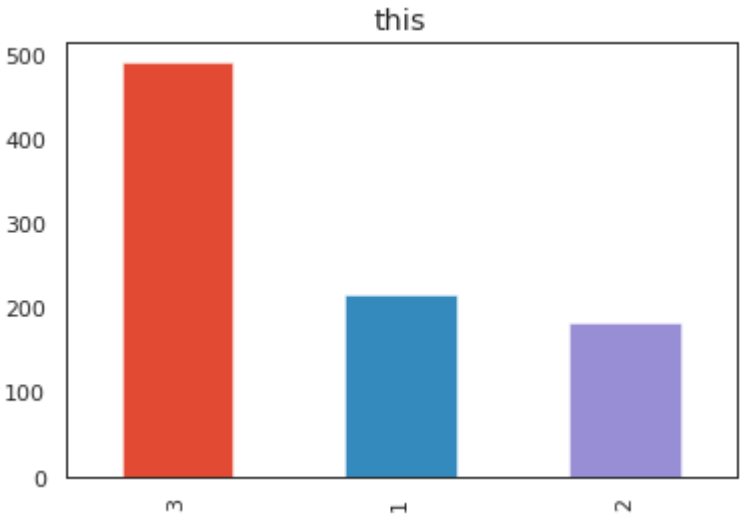
[24]
```
plt.imshow(df_train.corr(), cmap='hot', interpolation='nearest')
plt.show()
```



Bar Plot

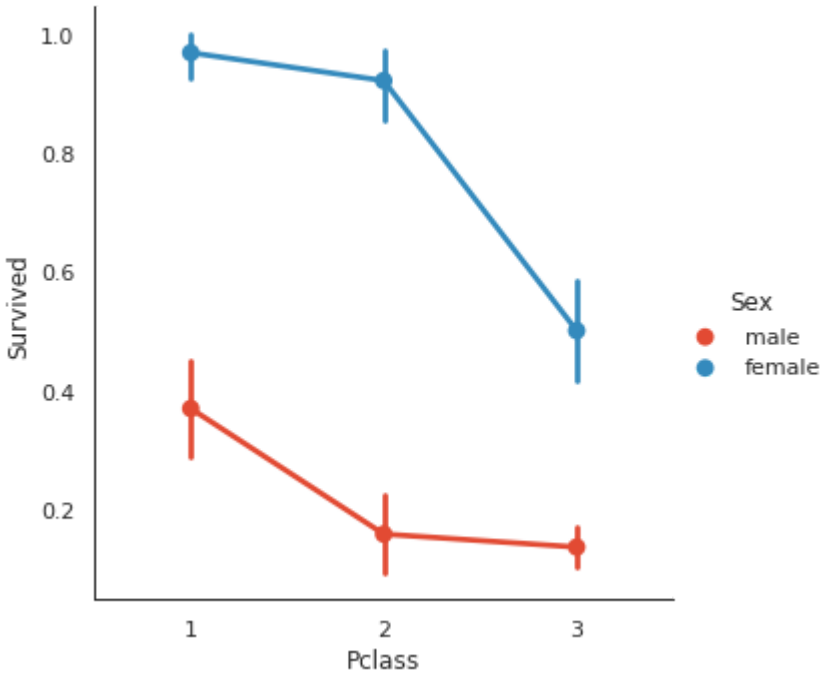
[25]
```
df_train['Pclass'].value_counts().plot(kind="bar")
plt.title("this")
```

```
Text(0.5, 1.0, 'this')
```

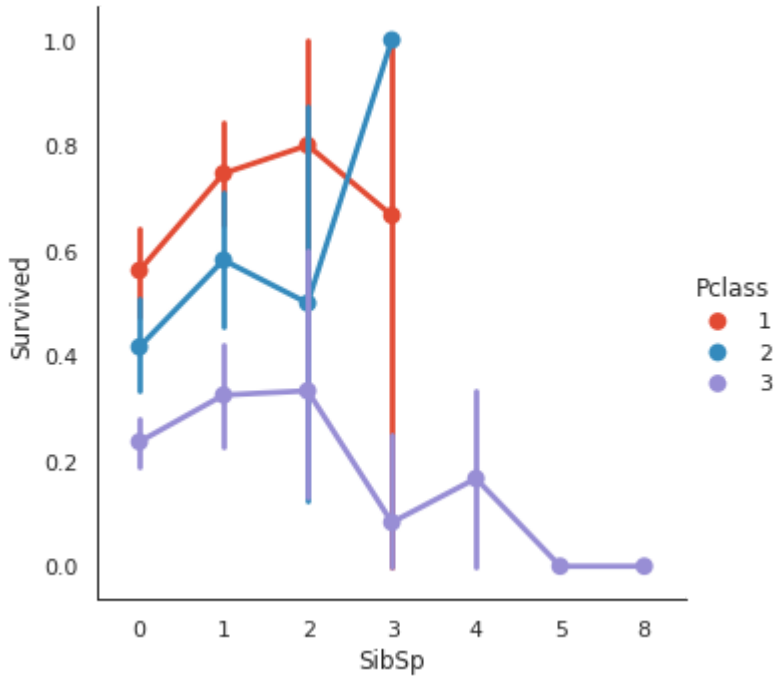


Factorplot

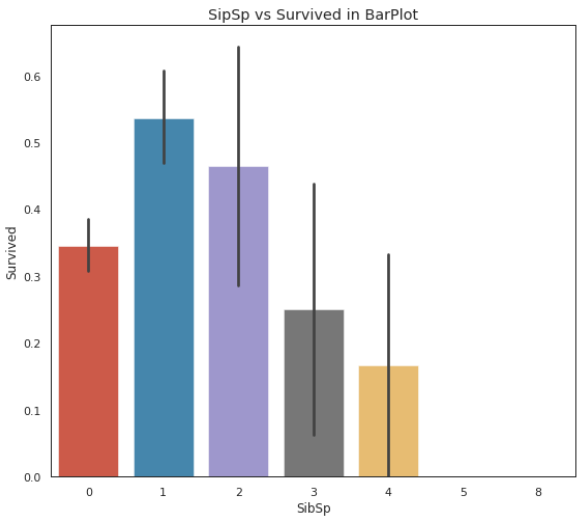
[26]
```
sns.factorplot('Pclass','Survived',hue='Sex',data=df_train)
plt.show();
```

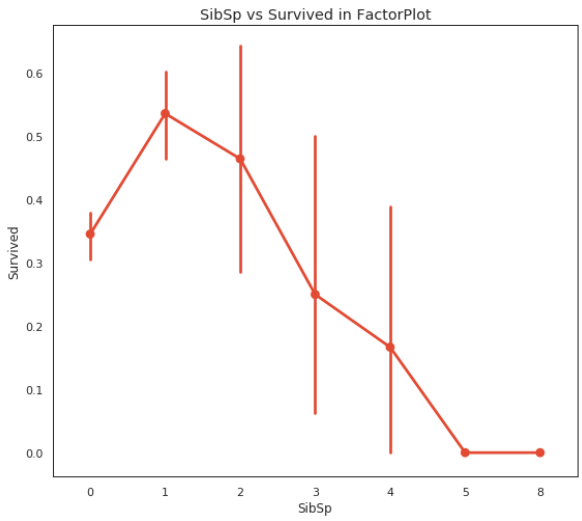


[27]
```
sns.factorplot('SibSp','Survived',hue='Pclass',data=df_train)
plt.show()
```



[28]
```
#let's see some others factorplot
f,ax=plt.subplots(1,2,figsize=(20,8))
sns.barplot('SibSp','Survived', data=df_train,ax=ax[0])
ax[0].set_title('SipSp vs Survived in BarPlot')
sns.factorplot('SibSp','Survived', data=df_train,ax=ax[1])
ax[1].set_title('SibSp vs Survived in FactorPlot')
plt.close(2)
plt.show();
```

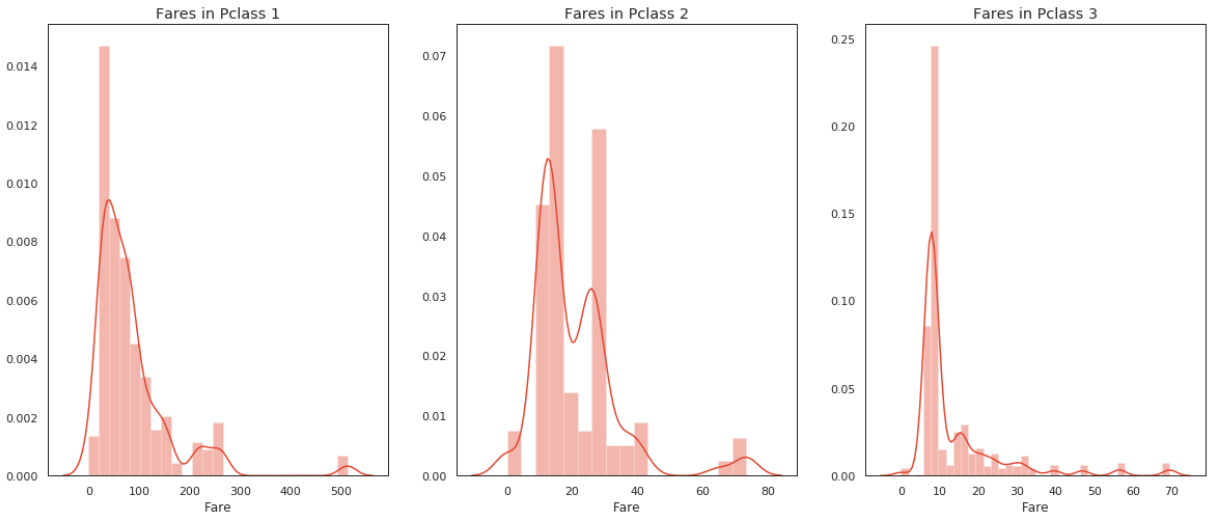




Distplot

[29]
```
f,ax=plt.subplots(1,3,figsize=(20,8))
```

```
sns.distplot(df_train[df_train['Pclass']==1].Fare,ax=ax[0])
ax[0].set_title('Fares in Pclass 1')
sns.distplot(df_train[df_train['Pclass']==2].Fare,ax=ax[1])
ax[1].set_title('Fares in Pclass 2')
sns.distplot(df_train[df_train['Pclass']==3].Fare,ax=ax[2])
ax[2].set_title('Fares in Pclass 3')
plt.show()
```



Preprocessing

[30]
```
def check_missing_data(df):
    flag=df.isna().sum().any()
    if flag==True:
        total = df.isnull().sum()
        percent = (df.isnull().sum())/(df.isnull().count()*100)
        output = pd.concat([total, percent], axis=1, keys=['Total', 'Pe
        data_type = []
        for col in df.columns:
            dtype = str(df[col].dtype)
            data_type.append(dtype)
        output['Types'] = data_type
        return(np.transpose(output))
    else:
        return(False)
```

[31]
```
check_missing_data(df_train)
```

	PassengerId	Survived	Pclass	Name	Sex	Age

	PassengerId	Survived	Pclass	Name	Sex	Age
Total	0	0	0	0	0	177
Percent	0	0	0	0	0	0.00198653
Types	int64	int64	int64	object	object	float64

[32]
```
check_missing_data(df_test)
```

	PassengerId	Pclass	Name	Sex	Age	SibSp
Total	0	0	0	0	86	0
Percent	0	0	0	0	0.00205742	0
Types	int64	int64	object	object	float64	int64

[33]
```
print(df_train.shape)
```

```
(891, 12)
```

[34]
```
df_train.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
```

[35]
```
df_train['Embarked'].unique()
```

```
array(['S', 'C', 'Q', nan], dtype=object)
```

[36]
```
df_train.groupby('Survived').count()
```

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch
Survived							
0	549	549	549	549	424	549	549
1	342	342	342	342	290	342	342

[37]
```
df_train.columns
```

```
Index(['PassengerId', 'Survived', 'Pclass', 'Name', 'Sex', 'Age', 'SibSp',
       'Parch', 'Ticket', 'Fare', 'Cabin', 'Embarked'],
      dtype='object')
```

[38]
```
df_train[df_train['Age']==30]
```

	PassengerId	Survived	Pclass	
79	80	1	3	Dowdell, Miss. Elizabeth
157	158	0	3	Corn, Mr. Harry
178	179	0	2	Hale, Mr. Reginald
213	214	0	2	Givard, Mr. Hans Kristensen
219	220	0	2	Harris, Mr. Walter
244	245	0	3	Attalah, Mr. Sleiman
253	254	0	3	Lobb, Mr. William Arthur
257	258	1	1	Cherry, Miss. Gladys

	PassengerId	Survived	Pclass	
286	287	1	3	de Mulder, Mr. Theodore
308	309	0	2	Abelson, Mr. Samuel
309	310	1	1	Francatelli, Miss. Laura Mabel
322	323	1	2	Slayter, Miss. Hilda Mary
365	366	0	3	Adahl, Mr. Mauritz Nils Martin
418	419	0	2	Matthews, Mr. William John
452	453	0	1	Foreman, Mr. Benjamin Laventall
488	489	0	3	Somerton, Mr. Francis William
520	521	1	1	Perreault, Miss. Anne
534	535	0	3	Cacic, Miss. Marija
537	538	1	1	LeRoy, Miss. Bertha
606	607	0	3	Karaic, Mr. Milan
726	727	1	2	Renouf, Mrs. Peter Henry (Lillian Jeffer
747	748	1	2	Sinkkonen, Miss. Anna
798	799	0	3	Ibrahim Shawah, Mr. Youssef
799	800	0	3	Van Impe, Mrs. Jean Baptiste (Rosalie
842	843	1	1	Serepeca, Miss. Augusta

Separate the data into dependent and independent variables

[39]
```
X = df_train.iloc[:, :-1].values
y = df_train.iloc[:, -1].values
```

[40]
```
train = df_train.dropna()
```

Cleaning :

1. Tranforming

[41]
```
def simplify_ages(df):
    df.Age = df.Age.fillna(-0.5)
    bins = (-1, 0, 5, 12, 18, 25, 35, 60, 120)
    group_names = ['Unknown', 'Baby', 'Child', 'Teenager', 'Student', '
    categories = pd.cut(df.Age, bins, labels=group_names)
    df.Age = categories
    return df

def simplify_cabins(df):
    df.Cabin = df.Cabin.fillna('N')
    df.Cabin = df.Cabin.apply(lambda x: x[0])
    return df

def simplify_fares(df):
    df.Fare = df.Fare.fillna(-0.5)
    bins = (-1, 0, 8, 15, 31, 1000)
    group_names = ['Unknown', '1_quartile', '2_quartile', '3_quartile',
    categories = pd.cut(df.Fare, bins, labels=group_names)
    df.Fare = categories
    return df

def format_name(df):
    df['Lname'] = df.Name.apply(lambda x: x.split(' ')[0])
    df['NamePrefix'] = df.Name.apply(lambda x: x.split(' ')[1])
    return df

def drop_features(df):
    return df.drop(['Ticket', 'Name', 'Embarked'], axis=1)

def transform_features(df):
    df = simplify_ages(df)
    df = simplify_cabins(df)
    df = simplify_fares(df)
    df = format_name(df)
    df = drop_features(df)
    return df

df_train = transform_features(df_train)
df_test = transform_features(df_test)
df_train.head()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	
0	1	0	3	male	Student	1	

	PassengerId	Survived	Pclass	Sex	Age	SibSp	
1	2	1	1	female	Adult	1	
2	3	1	3	female	Young Adult	0	
3	4	1	1	female	Young Adult	1	
4	5	0	3	male	Young Adult	0	

Feature Encoding

Must ALWAYS change categorical variables to numerical values.

[42]
```
def encode_features(df_train, df_test):
    features = ['Fare', 'Cabin', 'Age', 'Sex', 'Lname', 'NamePrefix']
    df_combined = pd.concat([df_train[features], df_test[features]])

    for feature in features:
        le = preprocessing.LabelEncoder()
        le = le.fit(df_combined[feature])
        df_train[feature] = le.transform(df_train[feature])
        df_test[feature] = le.transform(df_test[feature])
    return df_train, df_test
```

Model Deployment

Encode Dataset

[43]
```
df_train, df_test = encode_features(df_train, df_test)
df_train.head()
```

	PassengerId	Survived	Pclass	Sex	Age	SibSp	Parch	
0	1	0	3	1	4	1	0	
1	2	1	1	0	0	1	0	
2	3	1	3	0	7	0	0	
3	4	1	1	0	7	1	0	
4	5	0	3	1	7	0	0	

[44]
```
x_all = df_train.drop(['Survived', 'PassengerId'], axis=1)
y_all = df_train['Survived']
```

[45]
```
num_test = .3
X_train, X_test, y_train, y_test = train_test_split(x_all, y_all, test_
```

Accuracy and precision

```
We know that the titanic problem is a binary classification and to ev
```

accuracy

```
Your score is the percentage of passengers you correctly predict. Thi
```

precision :

```
In pattern recognition, information retrieval and binary classificati
```

recall :

```
recall is the fraction of relevant instances that have been retrieved
```

F-score :

```
 the F1 score is a measure of a test's accuracy. It considers both th
```

What is the difference between accuracy and precision?

```
"Accuracy" and "precision" are general terms throughout science. A go
```

RandomForestClassifier

[52]
```
rfc = RandomForestClassifier()

# Choose some parameter combinations to try
parameters = {'n_estimators': [4, 6, 9],
              'max_features': ['log2', 'sqrt','auto'],
              'criterion': ['entropy', 'gini'],
              'max_depth': [2, 3, 5, 10],
              'min_samples_split': [2, 3, 5],
              'min_samples_leaf': [1,5,8]
             }

# Type of scoring used to compare parameter combinations
acc_scorer = make_scorer(accuracy_score)

# Run the grid search
grid_obj = GridSearchCV(rfc, parameters, scoring=acc_scorer)
grid_obj = grid_obj.fit(X_train, y_train)

# Set the clf to the best combination of parameters
rfc = grid_obj.best_estimator_

# Fit the best algorithm to the data.
rfc.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini'
                       max_depth=10, max_features='sqrt', max_leaf_nodes=N
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=8, min_samples_split=5,
                       min_weight_fraction_leaf=0.0, n_estimators=4,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

[53]
```
rfc_prediction = rfc.predict(X_test)
rfc_score=accuracy_score(y_test, rfc_prediction)
print(rfc_score)
```

```
0.7947761194029851
```

Gradient Boosting

[54]
```
xgboost = xgb.XGBClassifier(max_depth=3, n_estimators=300, learning_rat
```

Prediction

[55]
```
xgb_prediction = xgboost.predict(X_test)
xgb_score=accuracy_score(y_test, xgb_prediction)
print(xgb_score)
```

```
0.8134328358208955
```

Logreg

[56]
```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0
                   warm_start=False)
```

[57]
```
logreg_prediction = logreg.predict(X_test)
logreg_score=accuracy_score(y_test, logreg_prediction)
print(logreg_score)
```

```
0.7947761194029851
```

Decision Tree Regressor

Mean Squared Error is to meature quality of the split.

[59]
```
from sklearn.tree import DecisionTreeRegressor

# define model. specify a number for random_state to ensure same result
dt = DecisionTreeRegressor(random_state=1)
```

[60]
```
#fit model

dt.fit(X_train, y_train)
```

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=1, splitter='best')
```

[61]
```
dt_prediction = dt.predict(X_test)
dt_score = accuracy_score(y_test, dt_prediction)
print(dt_score)
```

```
0.7835820895522388
```

Extra Tree Regressor

differ: WHen looking for the best split to sepearate the sames opf a node into two groups, ranc

[64]
```
from sklearn.tree import ExtraTreeRegressor
#define model. SPecify a number for random_state to ensure same results

etr = ExtraTreeRegressor()
```

[65]
```
# fit model
etr.fit(X_train, y_train)
```

```
ExtraTreeRegressor(criterion='mse', max_depth=None, max_features='auto',
                   max_leaf_nodes=None, min_impurity_decrease=0.0,
```

```
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      random_state=None, splitter='random')
```

[66]
```
etr_prediction = etr.predict(X_test)
etr_score = accuracy_score(y_test, etr_prediction)
print(etr_score)
```

```
0.7723880597014925
```

[67]
```
X_train = df_train.drop("Survived", axis=1)
y_train = df_train['Survived']
```

[68]
```
X_train = X_train.drop("PassengerId", axis=1)
X_test = df_test.drop("PassengerId", axis=1)
```

[70]
```
xgboost = xgb.XGBClassifier(max_depth=3, n_estimators=300, learning_rat
```

[71]
```
y_pred = xgboost.predict(X_test)
```

[73]
```
submission = pd.DataFrame({
        "PassengerId": df_test["PassengerId"],
        "Survived": y_pred
    })
submission.to_csv('submission.csv', index=False)
```

[ ]