Titanic Disaster

- Business Understanding
- Data Exploratory Analysis
- Data Preprocessing
- Model Building

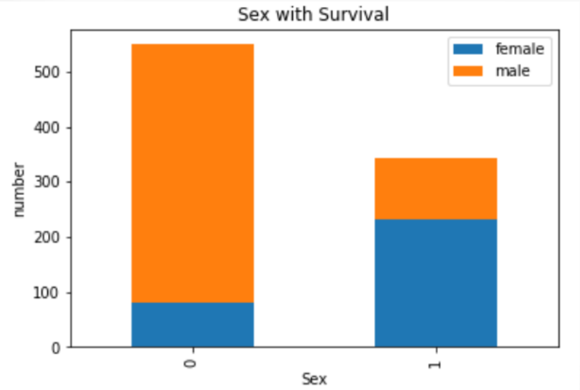
Business Understanding

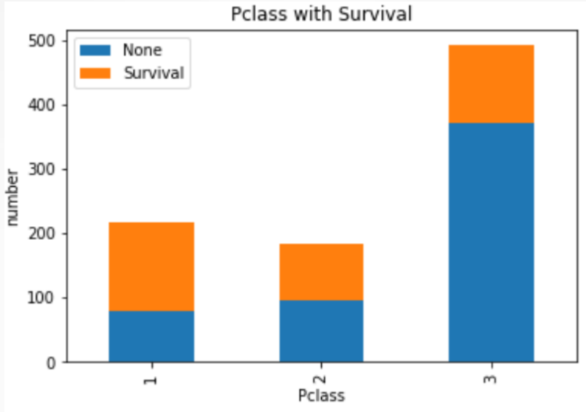
Business Understanding

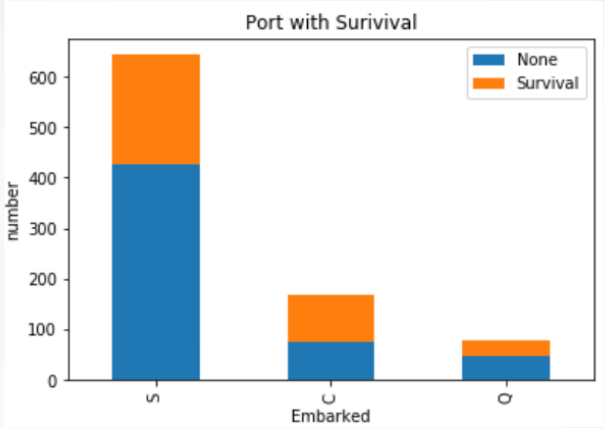
Variable	Definition	Key
survival	Survival	0 = No, 1 = Yes
pclass	Ticket class	1 = 1st, 2 = 2nd, 3 = 3rd
sex	Sex	
Age	Age in years	
sibsp	# of siblings / spouses aboard the Titanic	
parch	# of parents / children aboard the Titanic	
ticket	Ticket number	
fare	Passenger fare	
cabin	Cabin number	
embarked	Port of Embarkation	C = Cherbourg, Q = Queenstown, S = Southampton

Data Exploratory Analysis

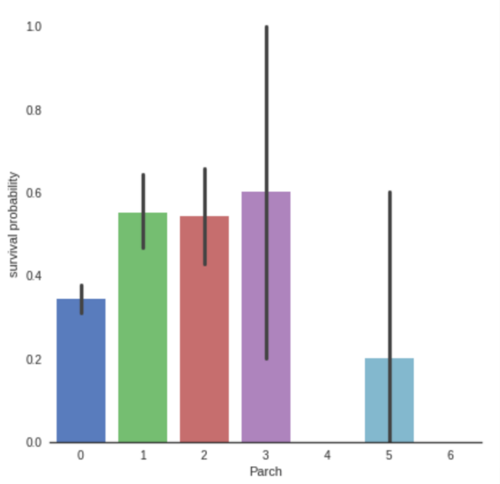
Data Exploratory

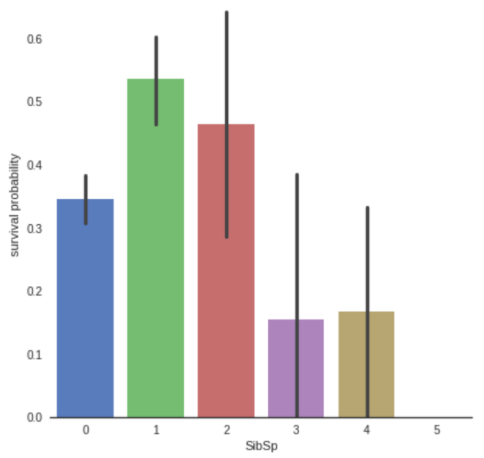


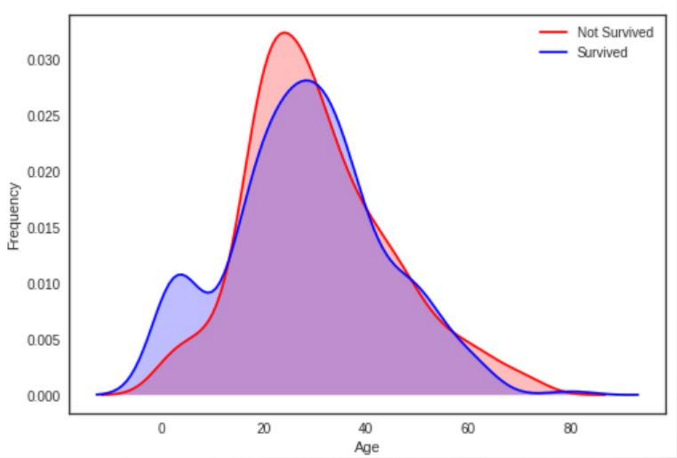




Data Exploratory







Data Exploratory

Data don’t tell the lie

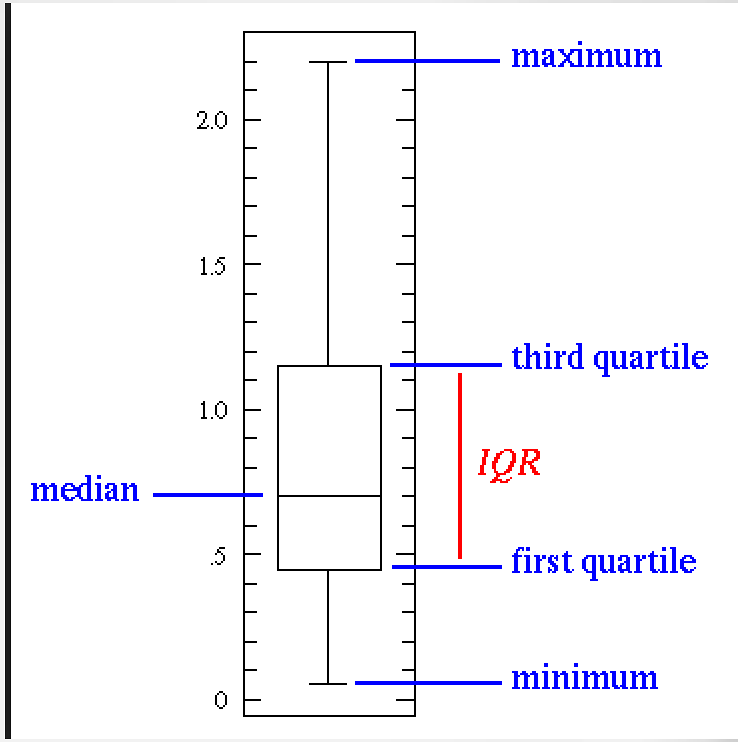
Survived	Pclass	Nan	Sex	/	Sib	Parch
0	3	Goo	female		1	6
0	3	And	male		1	5
1	3	Asp	female		1	5
0	3	And	female		1	5
0	3	Pan	female		0	5
0	3	Rice	female		0	5
0	3	Sko	female		1	4
0	3	Sko	male		1	4
0	3	Pals	female		0	4
0	3	Forc	male		1	3
0	3	Forc	female		1	3

Data Preprocessing

Outlier detection

Tukey–Kramer method

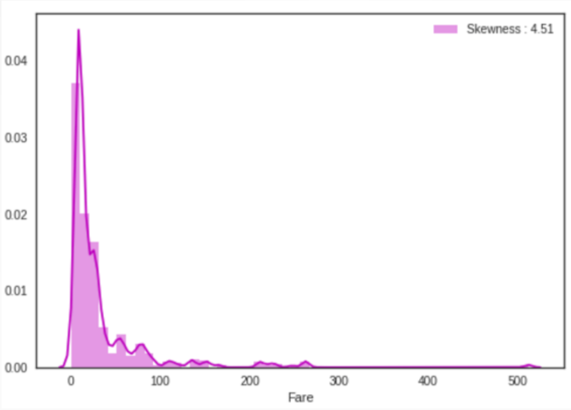
```python
from collections import Counter
def detect_outliers(df,n,features):

    outlier_indices = []

    # iterate over features(columns)
    for col in features:
        # 1st quartile (25%)
        Q1 = np.percentile(df[col], 25)
        # 3rd quartile (75%)
        Q3 = np.percentile(df[col],75)
        # Interquartile range (IQR)
        IQR = Q3 - Q1

        # outlier step
        outlier_step = 1.5 * IQR

        # Determine a list of indices of outliers for feature col
        outlier_list_col = df[(df[col] < Q1 - outlier_step) | (df[col] > Q3 + outlier_step )].index

        # append the found outlier indices for col to the list of outlier indices
        outlier_indices.extend(outlier_list_col)

    # select observations containing more than 2 outliers
    outlier_indices = Counter(outlier_indices)
    multiple_outliers = list( k for k, v in outlier_indices.items() if v > n )

    return multiple_outliers
# detect outliers from Age, SibSp , Parch and Fare
Outliers_to_drop = detect_outliers(train,2,["Age","SibSp","Parch","Fare"])
```



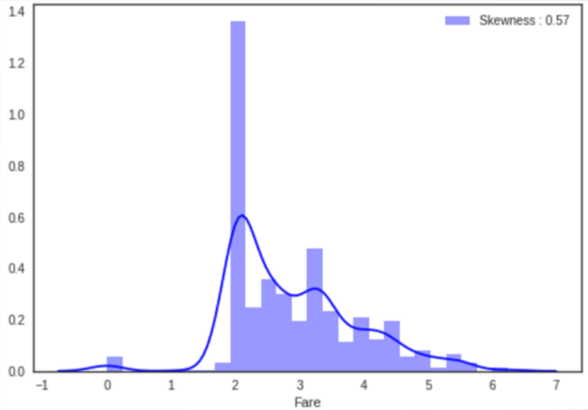
Data Preprocessing

Transformation

The “Fare” distribution is very skewed, which will lead to overweight values in the model. In this case, it’s better to transform it with the log function to reduce the skew.



Before



After

Data Preprocessing

Missing Value

Embarked & Cabin

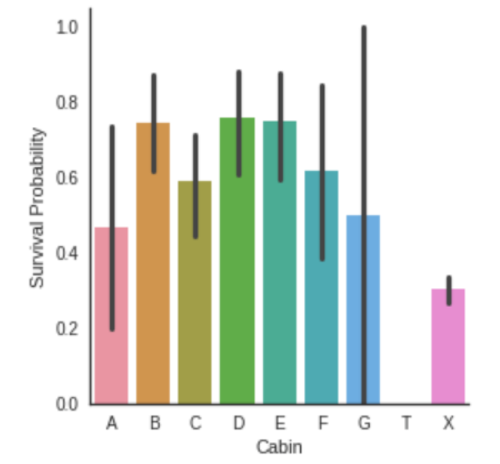
“Embarked” attribute:

```
def fill_missing_embarked(data):
    freq_port = data['Embarked'].mode()[0]
    data['Embarked'] = data['Embarked'].fillna(freq_port)
    data['Embarked'] = data['Embarked'].map({'S': 0, 'Q': 1, 'C': 2}).astype(int)
    return data

dataset = fill_missing_embarked(dataset)
```

```
train_data.isnull().sum()
```

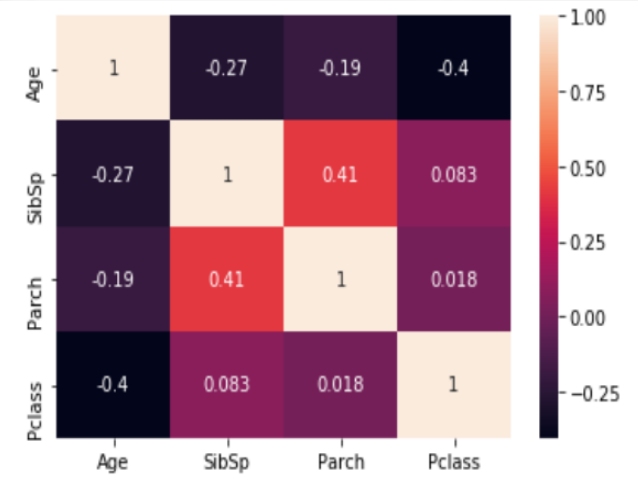
```
Survived      0
Pclass        0
Name          0
Sex           0
Age         177
SibSp         0
Parch         0
Ticket        0
Fare          0
Cabin       687
Embarked      2
dtype: int64
```

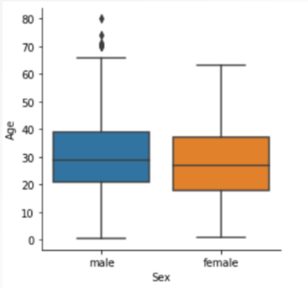
```
def cabin_extract(data):  #  1-L  2-M  3-H
    # classify Cabin by fare
    data['Cabin'] = data['Cabin'].fillna('X')
    data['Cabin'] = data['Cabin'].apply(lambda x: str(x)[0])
    data['Cabin'] = data['Cabin'].replace(['A', 'D', 'E', 'T'], int(1))
    data['Cabin'] = data['Cabin'].replace(['B', 'C'], int(2))
    data['Cabin'] = data['Cabin'].replace(['F', 'G'], int(3))
    data['Cabin'] = data['Cabin'].replace(['X'], int(0))
   # data['Cabin'] = data['Cabin'].map({'X': 0, 'L': 1, 'M': 2, 'H': 3}).astype(int)
    return data
```

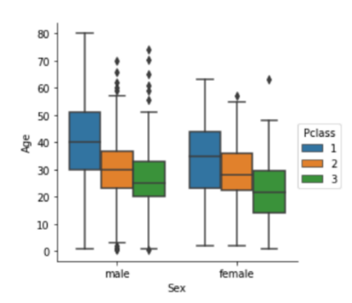


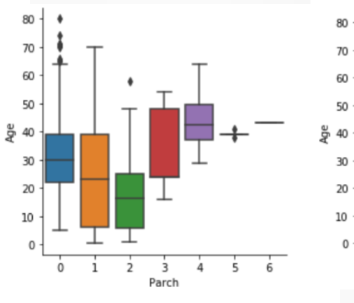
Missing Value

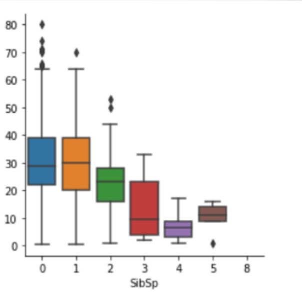
Age











1st class passengers are older than 2nd class passengers who are also older than 3rd class.
the more a passenger has parents/children the older he is and the more a passenger has siblings/spouses the younger he is.

Data Preprocessing

Missing Value

Embarked & Cabin

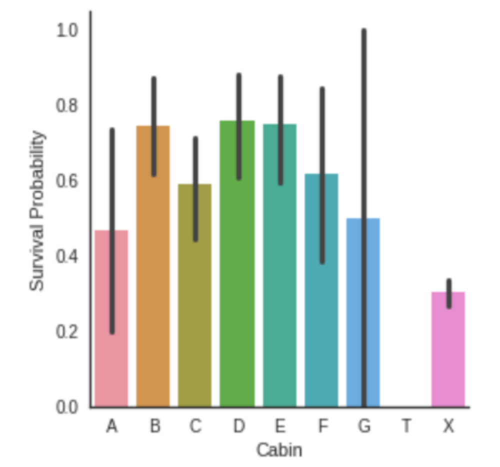
"Embarked" attribute:

```
def fill_missing_embarked(data):
    freq_port = data['Embarked'].mode()[0]
    data['Embarked'] = data['Embarked'].fillna(freq_port)
    data['Embarked'] = data['Embarked'].map({'S': 0, 'Q': 1, 'C': 2}).astype(int)
    return data

dataset = fill_missing_embarked(dataset)
```

```
train_data.isnull().sum()
```

```
Survived       0
Pclass         0
Name           0
Sex            0
Age          177
SibSp          0
Parch          0
Ticket         0
Fare           0
Cabin        687
Embarked       2
dtype: int64
```

```
def cabin_extract(data):  #  1-L  2-M  3-H
    # classify Cabin by fare
    data['Cabin'] = data['Cabin'].fillna('X')
    data['Cabin'] = data['Cabin'].apply(lambda x: str(x)[0])
    data['Cabin'] = data['Cabin'].replace(['A', 'D', 'E', 'T'], int(1))
    data['Cabin'] = data['Cabin'].replace(['B', 'C'], int(2))
    data['Cabin'] = data['Cabin'].replace(['F', 'G'], int(3))
    data['Cabin'] = data['Cabin'].replace(['X'], int(0))
   # data['Cabin'] = data['Cabin'].map({'X': 0, 'L': 1, 'M': 2, 'H': 3}).astype(int)
    return data
```

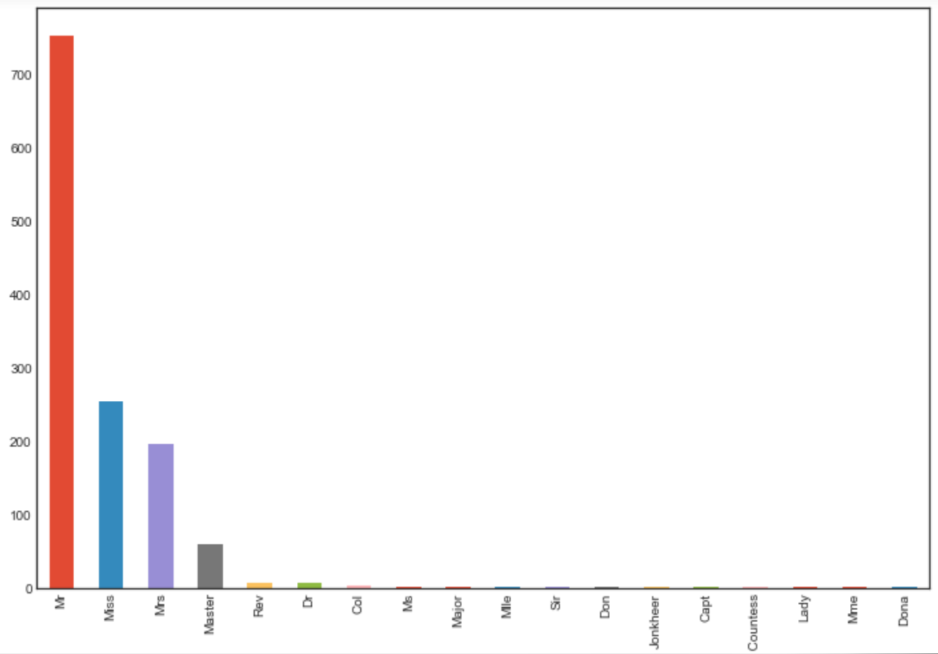


Data Preprocessing

Feature Engineering

Title

```python
def name_extract(data):
    # extract Title from name
    data['Title'] = data.Name.str.extract('([A-Za-z]+)\.', expand=False)# A-Za-z and end with a mark
    # delete rare title
    data['Title'] = data['Title'].replace(['Lady', 'Countess', 'Capt', 'Col',
        'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'Rare')
    data['Title'] = data['Title'].replace(['Mlle', 'Ms'], 'Miss')
    data['Title'] = data['Title'].replace('Mme', 'Mrs')

    title_map = {"Mr": 1, "Miss": 2, "Mrs": 3, "Master": 4, "Rare": 5}
    data['Title'] = data['Title'].map(title_map).astype(int)

    return data.drop('Name', axis=1)
```



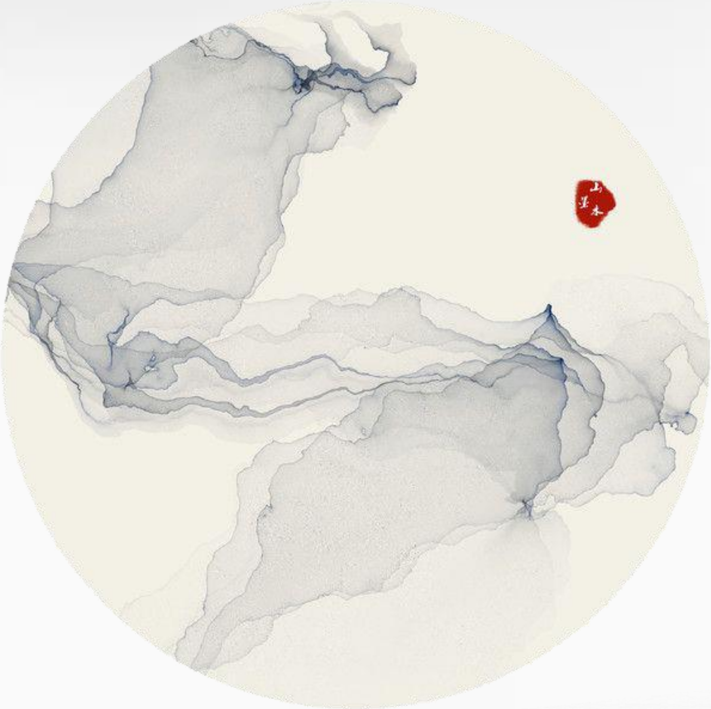
Data Preprocessing

Feature Engineering

Fare_Stage & Family & Alone

```
# Here, deal with the Fare, combine the train and test dataset of mean to discretized values
def fare_stage(data, mean_fare):
    data.loc[data['Fare'] > mean_fare[1], 'FareStage'] = 1
    data.loc[(data['Fare'] > mean_fare[2]) & (data['Fare'] <= mean_fare[1]), 'FareStage'] = 2
    data.loc[(data['Fare'] > mean_fare[3]) & (data['Fare'] <= mean_fare[2]), 'FareStage'] = 3
    data.loc[data['Fare'] <=mean_fare[3],'FareStage'] = 4
    return data
combine = pd.concat([train_data.drop('Survived',axis=1), test_data])
mean_fare = combine.groupby('Pclass')['Fare'].mean().astype(int)
mean_fare.plot()
print(mean_fare)
train_data = fare_stage(train_data, mean_fare)
test_data = fare_stage(test_data, mean_fare)
```

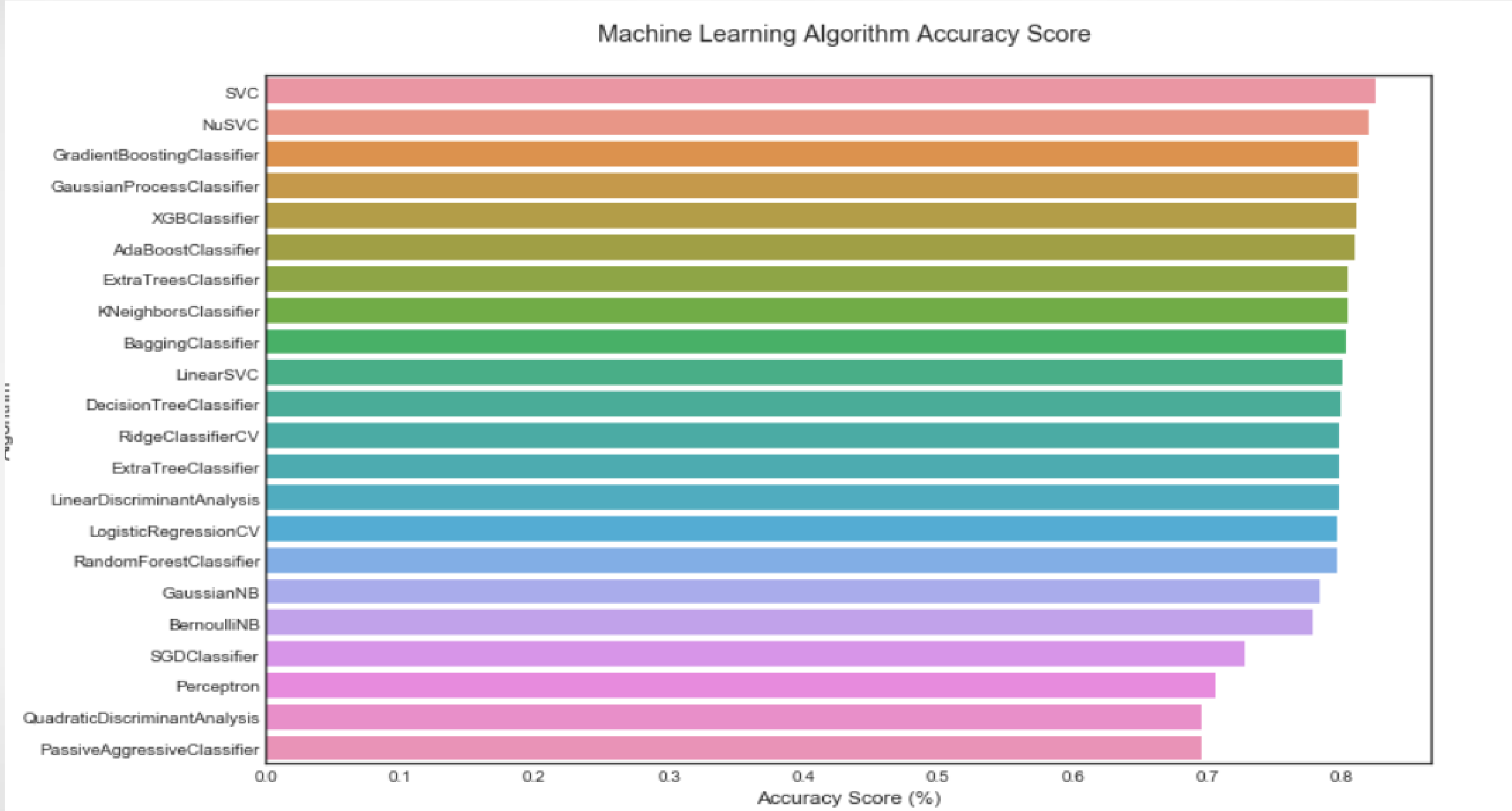
```
def family_info(data):
    data['FamliySize'] = data['SibSp'] + data['Parch'] + 1 #plus himself
    data['Alone'] = data['Alone'] = (data['SibSp'] == 0) & (data['Parch'] == 0)
    data['Alone'] = data['Alone'].astype(int)
    return data
```



Model Building

Model Building

Model Selection

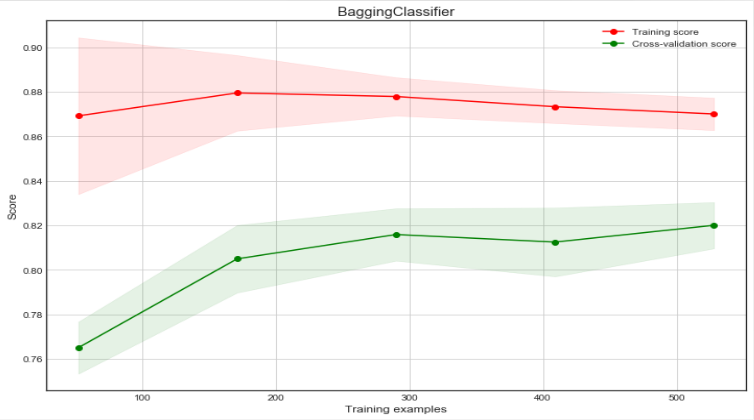


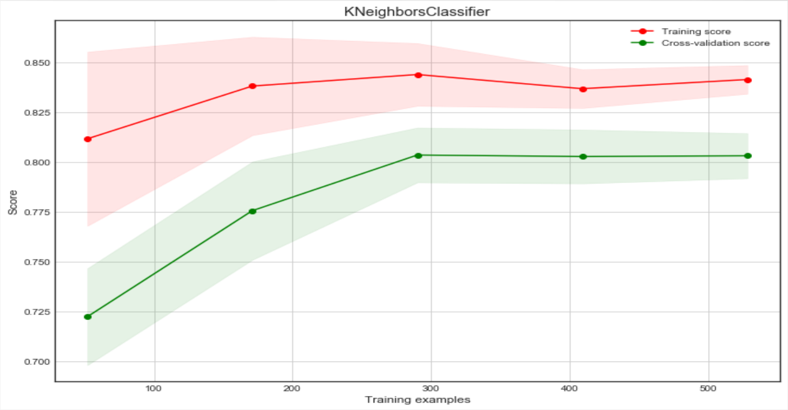
Model Selection

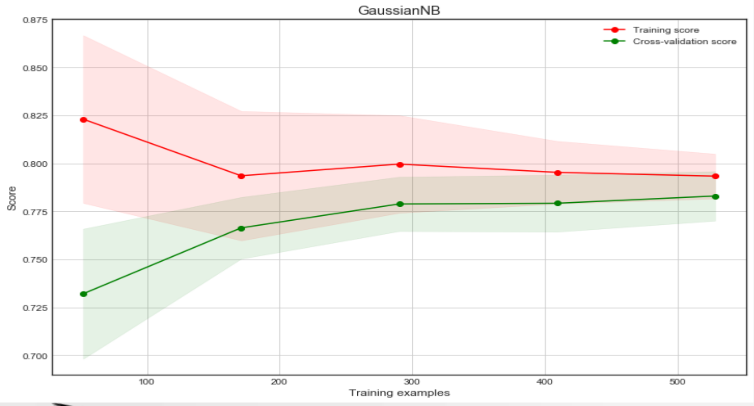
```
vote_set = [
    #Ensemble Methods: http://scikit-learn.org/stable/modules/ensemble.html
    ('ada', ensemble.AdaBoostClassifier()),
    ('bc', ensemble.BaggingClassifier()),
    ('etc',ensemble.ExtraTreesClassifier()),
    ('gbc', ensemble.GradientBoostingClassifier()),
    ('rfc', ensemble.RandomForestClassifier()),

    #GLM: http://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
    ('lr', linear_model.LogisticRegressionCV()),

    #Navies Bayes: http://scikit-learn.org/stable/modules/naive_bayes.html
    ('bnb', naive_bayes.BernoulliNB()),
    ('gnb', naive_bayes.GaussianNB()),

    #Nearest Neighbor: http://scikit-learn.org/stable/modules/neighbors.html
    ('knn', neighbors.KNeighborsClassifier()),

    #SVM: http://scikit-learn.org/stable/modules/svm.html
    ('svc', svm.SVC(probability=True)),

    #xgboost: http://xgboost.readthedocs.io/en/latest/model.html
    ('xgb', XGBClassifier())

]
```

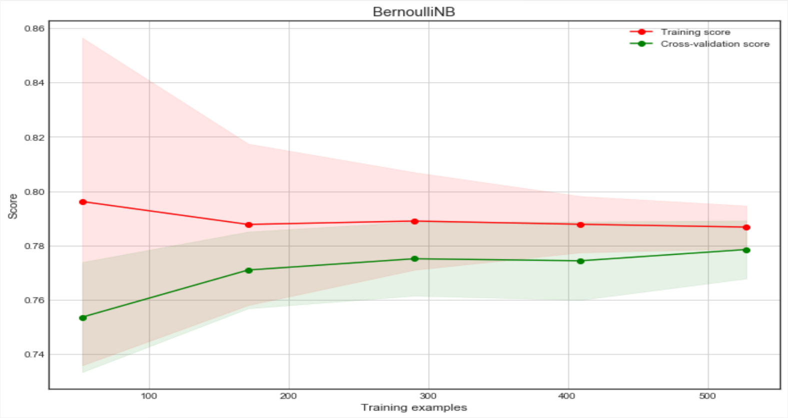
Model Building

Learning Curve









Model Building

Parameter Tuning

```python
for clf, param in zip(vote_set, grid_param):
    best_search = model_selection.GridSearchCV(estimator=clf[1], param_grid= param , cv = cv_split, scoring = 'roc_auc')
    best_search.fit(X_train, Y_train)
#     gsRFC = GridSearchCV(RFC,param_grid = rf_param_grid, cv=kfold, scoring="accuracy", n_jobs= 4, verbose = 1)
#     gsRFC.fit(X_train,Y_train)
#     RFC_best = gsRFC.best_estimator_
#     this is a way to apply the best parameter
    best_param = best_search.best_params_
    print(" the best param for {} is {} ".format(clf[1].__class__.__name__, best_param))
    print("************************************************")
    # this is another way to apply
    clf[1].set_params(**best_param)
```

```python
grid_n_estimator = [10, 50, 100, 300]
grid_ratio = [.1, .25, .5, .75, 1.0]
grid_learn = [.01, .03, .05, .1, .25]
grid_max_depth = [2, 4, 6, 8, 10, None]
grid_min_samples = [5, 10, .03, .05, .10]
grid_criterion = ['gini', 'entropy']
grid_bool = [True, False]
grid_seed = [0]
```

GridSearchCV exhaustively considers all parameter combinations to search the hyper-parameter for the best cross validation score.

Model Building

VotingClassifier

Hard voting VS Soft voting

If 'hard', uses predicted class labels for majority rule voting.

Else if 'soft', predicts the class label based on the argmax of the sums of the predicted probabilities.

```
# compare the hard and soft voting method.

grid_hard = ensemble.VotingClassifier(estimators = vote_set , voting = 'hard')
grid_hard_cv = model_selection.cross_validate(grid_hard, X_train,Y_train, cv  = cv_split)
grid_hard.fit(X_train,Y_train)

print("Hard Voting w/Tuned Hyperparameters Training w/bin score mean: {:.2f}". format(grid_hard_cv['train_score'].mean()*100)
print("Hard Voting w/Tuned Hyperparameters Test w/bin score mean: {:.2f}". format(grid_hard_cv['test_score'].mean()*100))
print("Hard Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- {:.2f}". format(grid_hard_cv['test_score'].std()*100*3
print('-'*10)

#Soft Vote or weighted probabilities w/Tuned Hyperparameters
grid_soft = ensemble.VotingClassifier(estimators = vote_set , voting = 'soft')
grid_soft_cv = model_selection.cross_validate(grid_soft, X_train,Y_train, cv  = cv_split)
grid_soft.fit(X_train,Y_train)

print("Soft Voting w/Tuned Hyperparameters Training w/bin score mean: {:.2f}". format(grid_soft_cv['train_score'].mean()*100)
print("Soft Voting w/Tuned Hyperparameters Test w/bin score mean: {:.2f}". format(grid_soft_cv['test_score'].mean()*100))
print("Soft Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- {:.2f}". format(grid_soft_cv['test_score'].std()*100*3
print('-'*10)
```

```
Hard Voting w/Tuned Hyperparameters Training w/bin score mean: 84.56
Hard Voting w/Tuned Hyperparameters Test w/bin score mean: 82.23
Hard Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- 3.10
----------
Soft Voting w/Tuned Hyperparameters Training w/bin score mean: 84.19
Soft Voting w/Tuned Hyperparameters Test w/bin score mean: 81.77
Soft Voting w/Tuned Hyperparameters Test w/bin score 3*std: +/- 2.63
----------
```

Thank You