1.数据introduction

1.1 Introduction

The sinking of the RMS Titanic is one of the most infamous shipwrecks in history.  On April 15, 1912, during her maiden voyage, the Titanic sank after colliding with an iceberg, killing 1502 out of 2224 passengers and crew. This sensational tragedy shocked the international community and led to better safety regulations for ships.

One of the reasons that the shipwreck led to such loss of life was that there were not enough lifeboats for the passengers and crew. Although there was some element of luck involved in surviving the sinking, some groups of people were more likely to survive than others, such as women, children, and the upper-class.

1.2 Business understanding

In this report, we ask you to complete the analysis of what sorts of people were likely to survive. In particular, we apply the tools of machine learning to predict which passengers survived the tragedy. We find an appropriate and credible dataset and analyze how its features and dimensions help with our predictions. This will be done by choosing appropriate data mining models used in everyday data analytics.

The data science solutions workflow solves for seven major goals.

**1.2.1 Classifying.** We may want to classify or categorize our samples. We may also want to understand the implications or correlation of different classes with our solution goal.

**1.2.2 Correlating.** One can approach the problem based on available features within the training dataset. Which features within the dataset contribute significantly to our solution goal? Statistically speaking is there a [correlation](https://en.wikiversity.org/wiki/Correlation) among a feature and solution goal? As the feature values change does the solution state change as well, and visa-versa? This can be tested both for numerical and categorical features in the given dataset. We may also want to determine correlation among features other than survival for subsequent goals and workflow stages. Correlating certain features may help in creating, completing, or correcting features.

**1.2.3 Converting.** For modeling stage, one needs to prepare the data. Depending on the choice of model algorithm one may require all features to be converted to numerical equivalent values. So for instance converting text categorical values to numeric values.

**1.2.4 Completing.** Data preparation may also require us to estimate any missing values within a feature. Model algorithms may work best when there are no missing values.

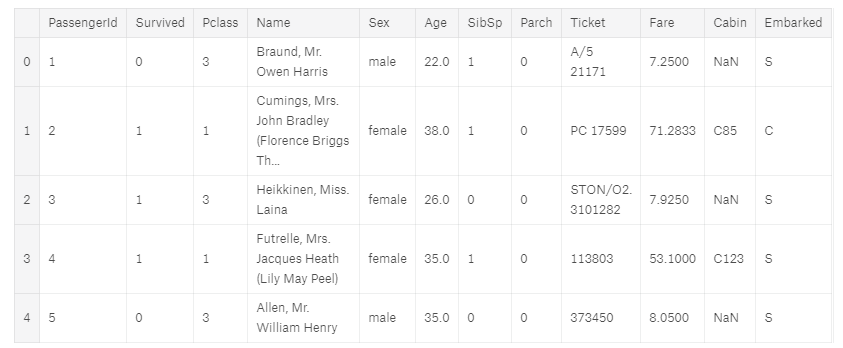
**1.2.5 Correcting.** We may also analyze the given training dataset for errors or possibly innacurate values within features and try to corrent these values or exclude the samples containing the errors. One way to do this is to detect any outliers among our samples or features. We may also completely discard a feature if it is not contribting to the analysis or may significantly skew the results.

**1.2.6 Creating.** Can we create new features based on an existing feature or a set of features, such that the new feature follows the correlation, conversion, completeness goals.

2.数据understanding

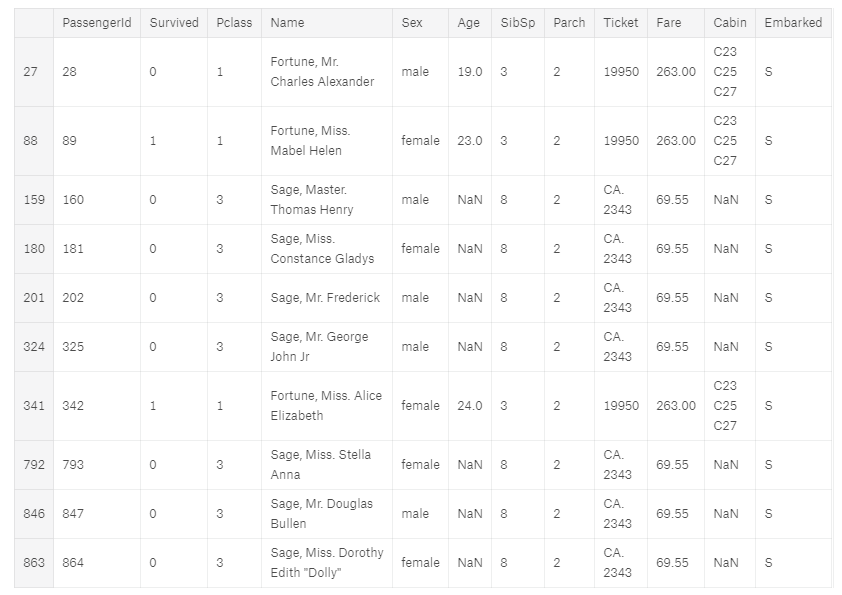
2.1数据来源

数据是kaggle竞赛题中的titanic乘客数据，共有881\*13的训练数据和418\*12的测试数据，共有13个特征值，包括PassengerId乘客ID、Pclass乘客等级(1/2/3等舱位)、Name乘客姓名、Sex性别、Age年龄、SibSp堂兄弟/妹个数、Parch父母与小孩个数、Ticket船票信息、Fare票价、Cabin客舱、Embarked登船港口以及survived是否生还



数据类型分别是：Survived: int、Pclass: int、Name: string、Sex: string、Age: string。**The Survived variable** is the outcome or dependent variable. It is a binary nominal datatype of 1 for "survived" and 0 for "did not survive". **All other variables** are potential predictor or independent variables. The goal is to predict this dependent variable only using the available independent variables. A test dataset has been created to test our algorithm.

通过质量控制"Age","SibSp","Parch","Fare"四列数据，共发现10行数据有异常值，乘客ID是28.89.和342是有更高的票价，其他七个是有异常的SibSP值，我们通过分析将这些值去除。



2.2 数据展示

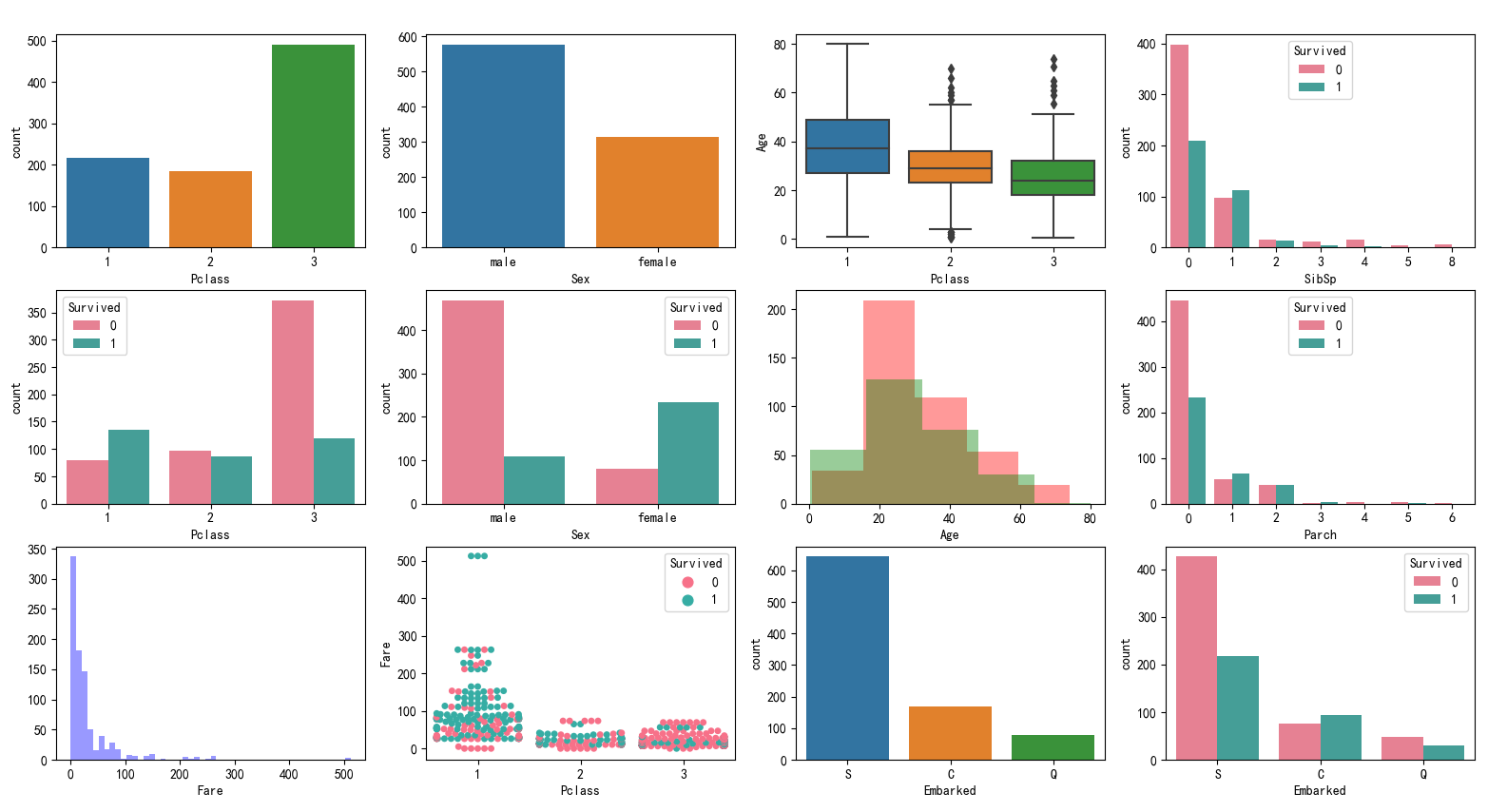
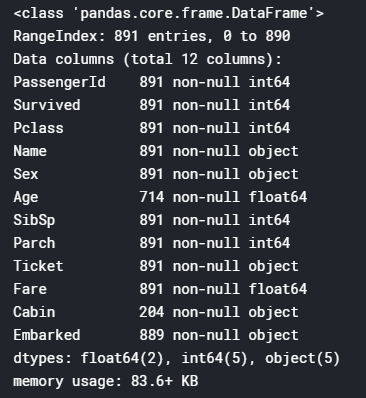


图1数据初步可视化分析图

可以看出乘客等级的数量比例，性比中男女数量的差异，乘客登记中的年龄分布，堂兄姐妹人数的幸存者的数量分布，乘客等级与幸存的数量分布，性别与幸存的数量分布，年龄与幸存分布，父母小孩个数与幸存的数量分布，票价的数量分布，幸存率与票价的散点分布，不同港口的数量分布，以及不同港口与幸存的数量分布。



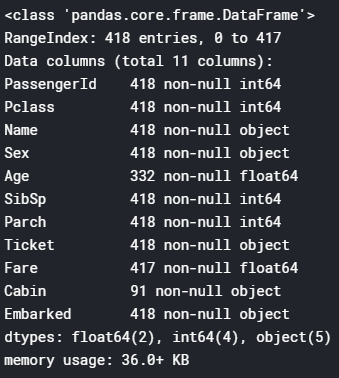


图2 数据完整性

通过info函数显示了数据集的完整性或不完整性。年龄、舱室和登船区域存在空值或缺失数据。丢失的值可能很糟糕，因为某些算法不知道如何处理空值，并且会失败。而其它如决策树，可以处理空值。这里我们完成缺失数据的方法是使用均值，中位数或均值+随机标准差来估算。同时通过对训练数据中的幸存一列进行描述性分析，平均生还率为0.38，标准差为0.49.

**2.2.1 Embarked**

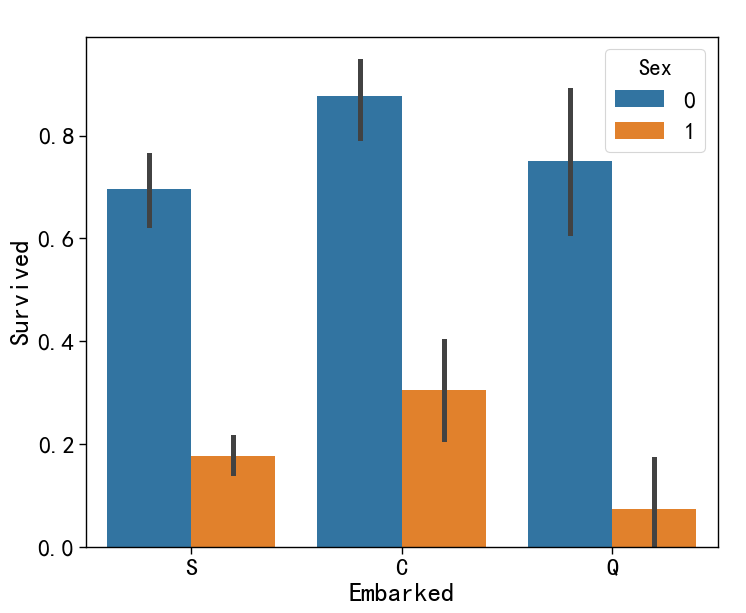


图3 港口与幸存率分布图

不同港口登船的乘客对幸存率没有直接的影响，我们将下面章节进行更深层次的分析，并将舍去那些我们不能直接证明与幸存率有关系的数据。当将乘客的舱位(pclass）与登船港口进行结合分析的话，会发现在S和Q登船的人员明显具有较低的幸存率，我们将把这个特征用于模型

**2.2.2 Named**

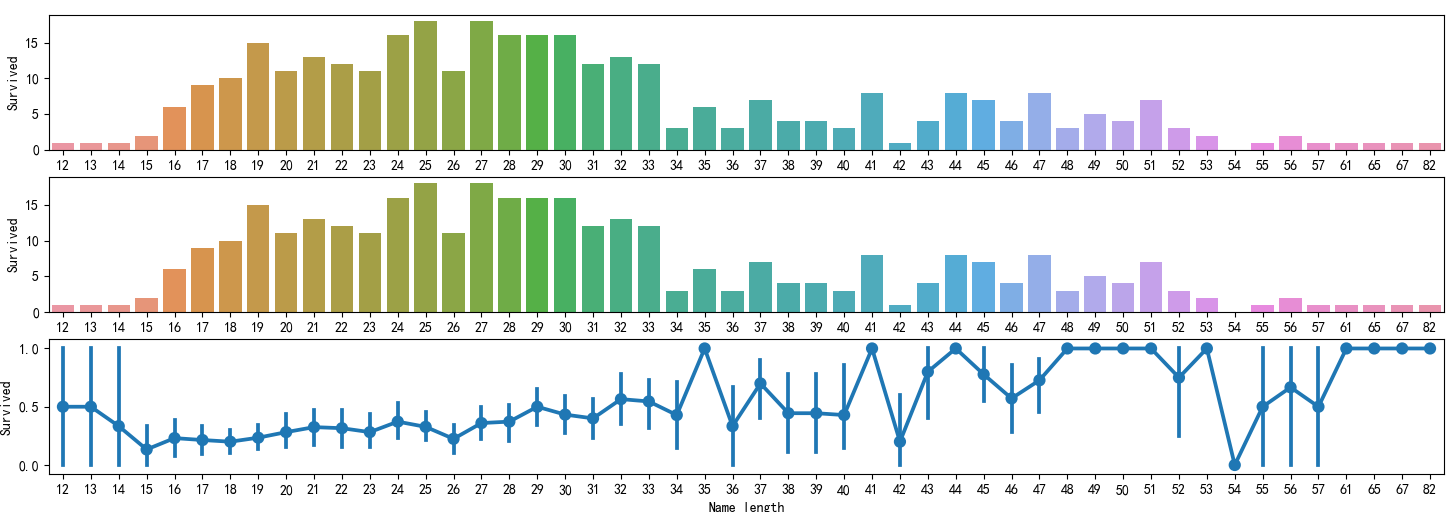


图4 名字长度与幸存率分布图

图4分别展示了姓名长度的总幸存率、平均幸存率的分布图。同时通过统计计算，得出姓名长度低于23的人（大都是男性）有360人，姓名长度在24至28之间的有240人，姓名长度在29至40之间的有201人，姓名长度大于41的人（大都是女性）有90人，这些不同姓名长度分组的组别没有明显的幸存率差异。

**2.2.3 Age**

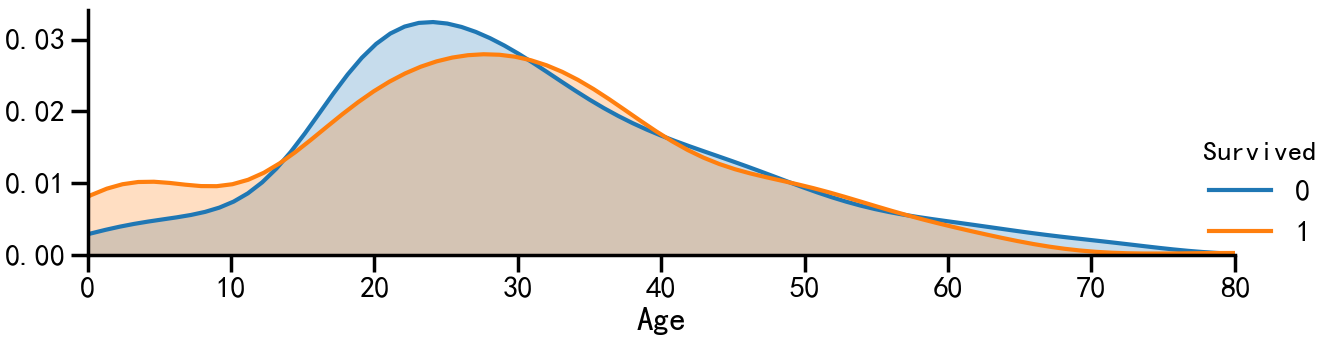


图5 年龄与幸存率分布图

图5展示了年龄与幸存率分布图，共分为6组，其中年龄在小于14岁为一组，幸存率最高为0.58，将14至30岁的人分为一组，幸存率为0.35，将30至40岁的人分为一组，幸存率为0.41，将40至50岁的人分为一组，幸存率为0.36，将50至60岁的人分为一组，幸存率为0.41，将大于60岁的人分为一组，幸存率最低为0.23.

**2.2.4 Family**

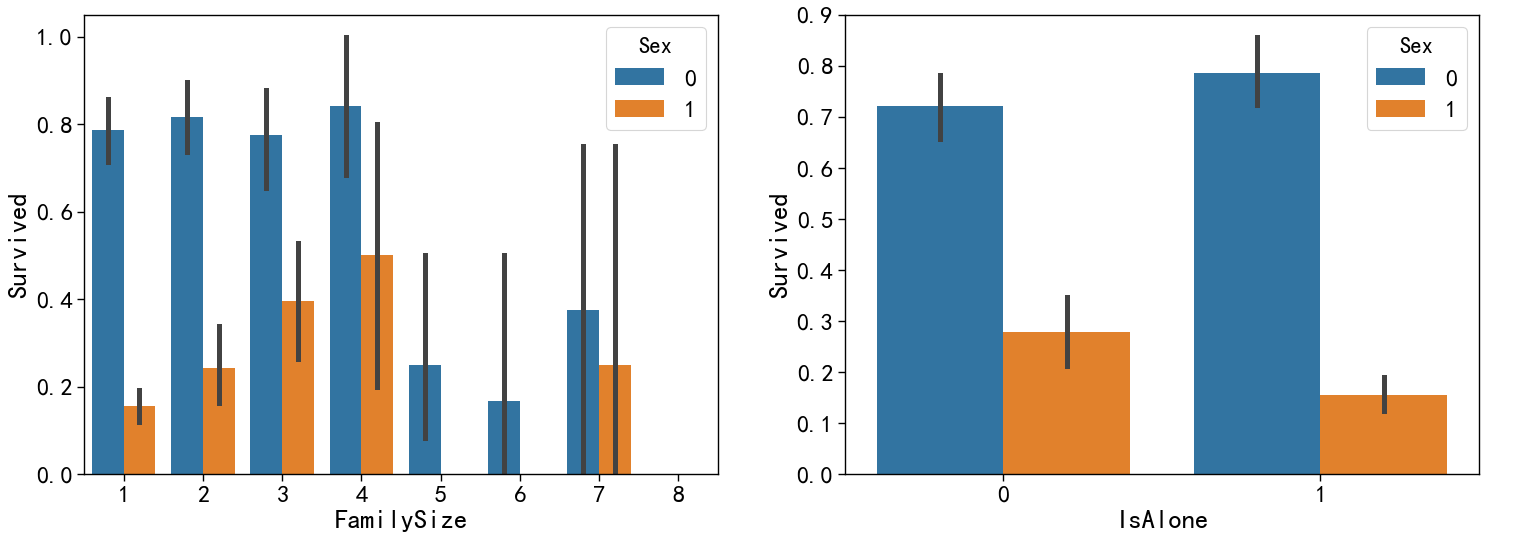
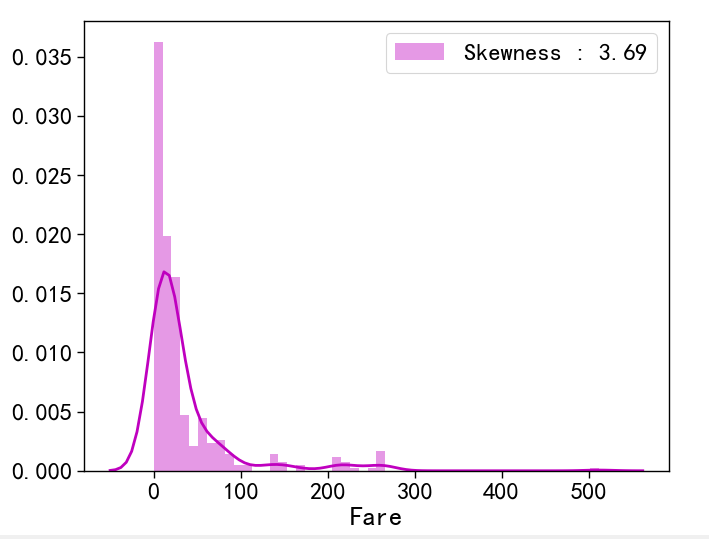


图6 家庭成员和孤儿的幸存率分布图

将具有不同家庭成员数量的人分为两组，通过家庭成员来看待与幸存率的关系，同时将年龄低14岁的孤儿进行区分。通过上图可以看出是否是孤儿对于幸存率没有显著性差异，也就是在性别上，男性和女性的幸存率有轻微的差异，所以是否是孤儿对于幸存率的预测来说不是一个很好的的指标。

**2.2.5 Fare**

图7 小费与幸存率分布图

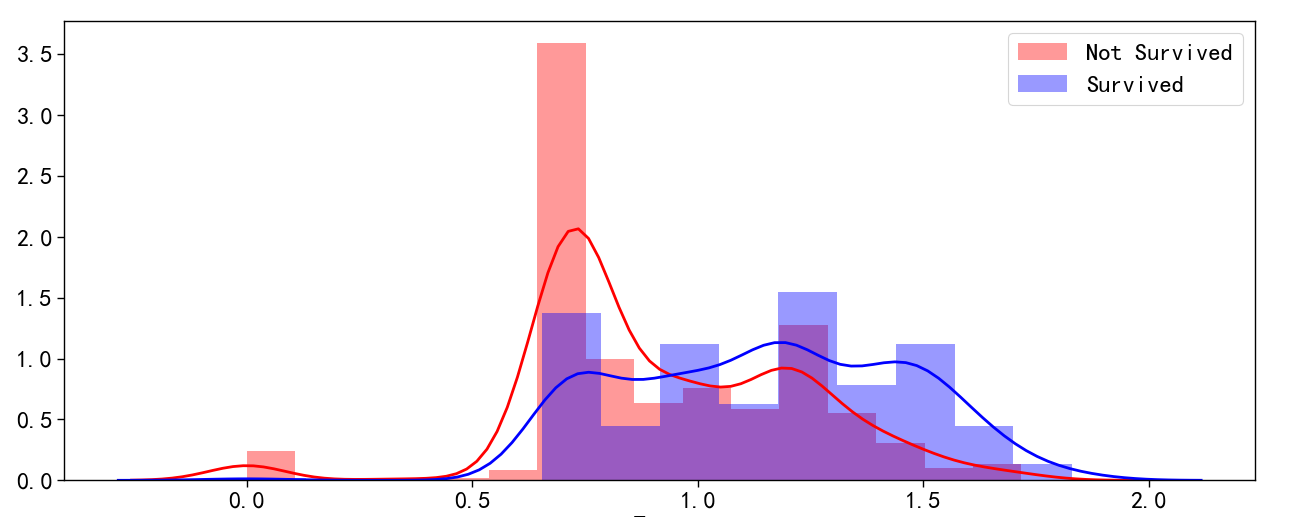


图8 处理后的小费与幸存率分布图

通过小费的分布可以看出数据的偏度向左，这可能导致模型具有很高的值而使权重偏高。因此在这个例子中，最好是通过log函数，减小数据的偏度，然后重新分布数据。从图8中便可看出在小费0-2.7中有很低的幸存率，当小费高于2.7时，幸存率便大大提升。

**2.2.6 房间号Cabin**

通过数据统计，没有房间号幸存人数为206人，有房间号的幸存人数为136人，但是有房间号的幸存率为0.67，没有房间号的人幸存率为0.30，因此是否有房间号是一个很重要的指标，将保存这一特征。并且从有房间号的乘客中提取房间信息，例如房间号为“C123”，这个与甲板相关，我们将这些乘客的甲板信息提取出来，作为一个单独的特征值，将空缺值转换为0.

**2.2.7 称呼**

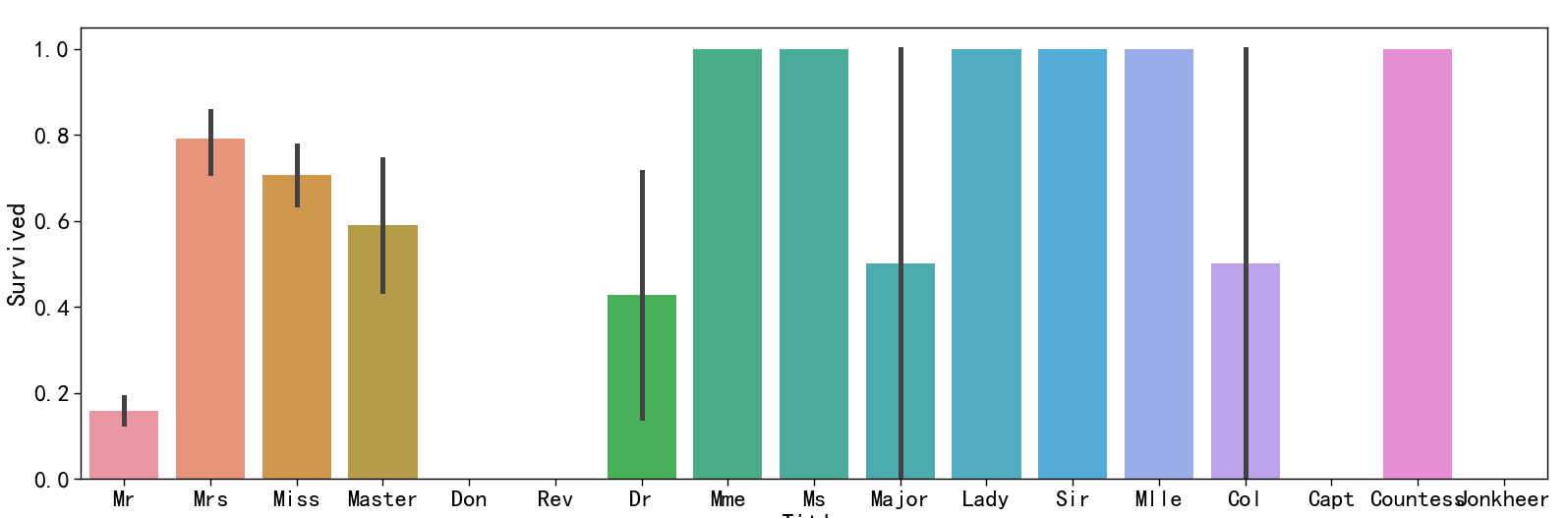


图9 不同称呼的幸存率分布图

通过幸存率可以将不同称呼分为6类，不同类别的人员幸存率具有显著差异，我们将此特征用于接下来的模型预测。

1. Mme, Ms, Lady, Sir, Mlle, Countess: 100%.
2. Mrs, Miss: around 70% survival
3. Master: around 60%
4. Don, Rev, Capt, Jonkheer: no data
5. Dr, Major, Col: around 40%
6. Mr: below 20%

**Initial observations from the descriptive statistics:**

* Only 38% survived, a real tragedy :-(
* Passengers in more expensive classes 1 and 2 had much higher chance of surviving than classes 3 or 4.
* Also, the higher the fare, the higher the chance. Similarly, having a cabin increases the chance of survival.
* Women (0) higher chance than men (1)
* Younger people slightly more chance than older
* Being alone decreased your chance to survive.

3.数据preparation

Correlation analysis - Multi-variate analysis

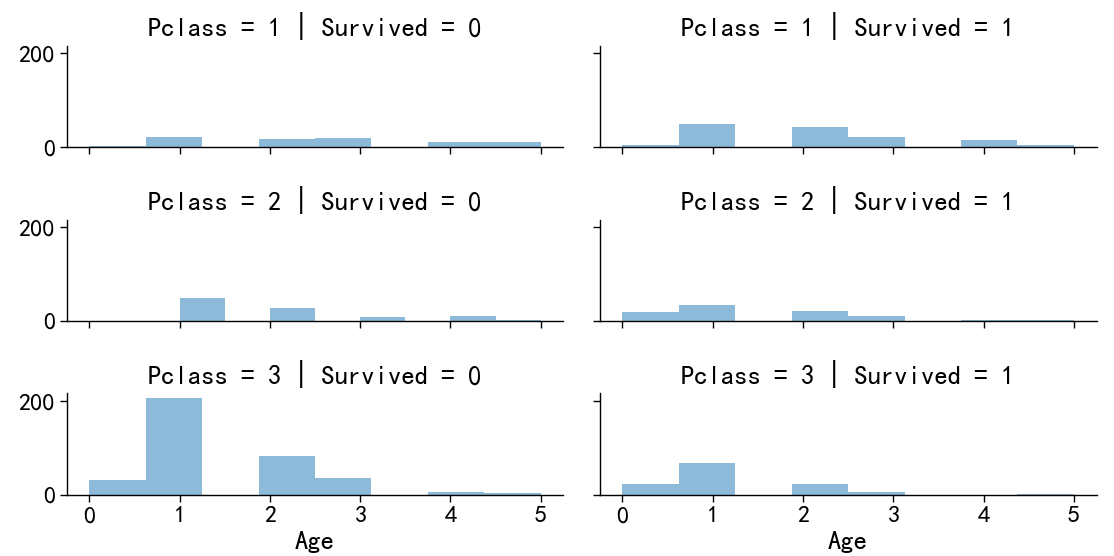
This section summarizes bivariate analysis asthe simplest forms of quantitative (statistical) analysis. It involves the analysis of one or two features, and their relative impact of "Survived". This is a useful frist step of our anblaysis in order to determine the empirical relationship between all features.

## 3.1. Correlation analysis with histograms and pivot-tables

**图10 港口、年龄与生存率分布图**

**Observations for Age graph:**

* 0 or blue represent women; 1 or orange represent men. Gender and age seem to have a stronger influece of the survival rate.
* We start to find where most survivors are: older women (48 to 64 year old), and younger passengers.
* What is statistically interesting is that only young boys (Age Category = 0) have high survival rates, unlike other age groups for men.
* We will create a new feature called young boys



**图11 舱位、年龄与生存率分布图**

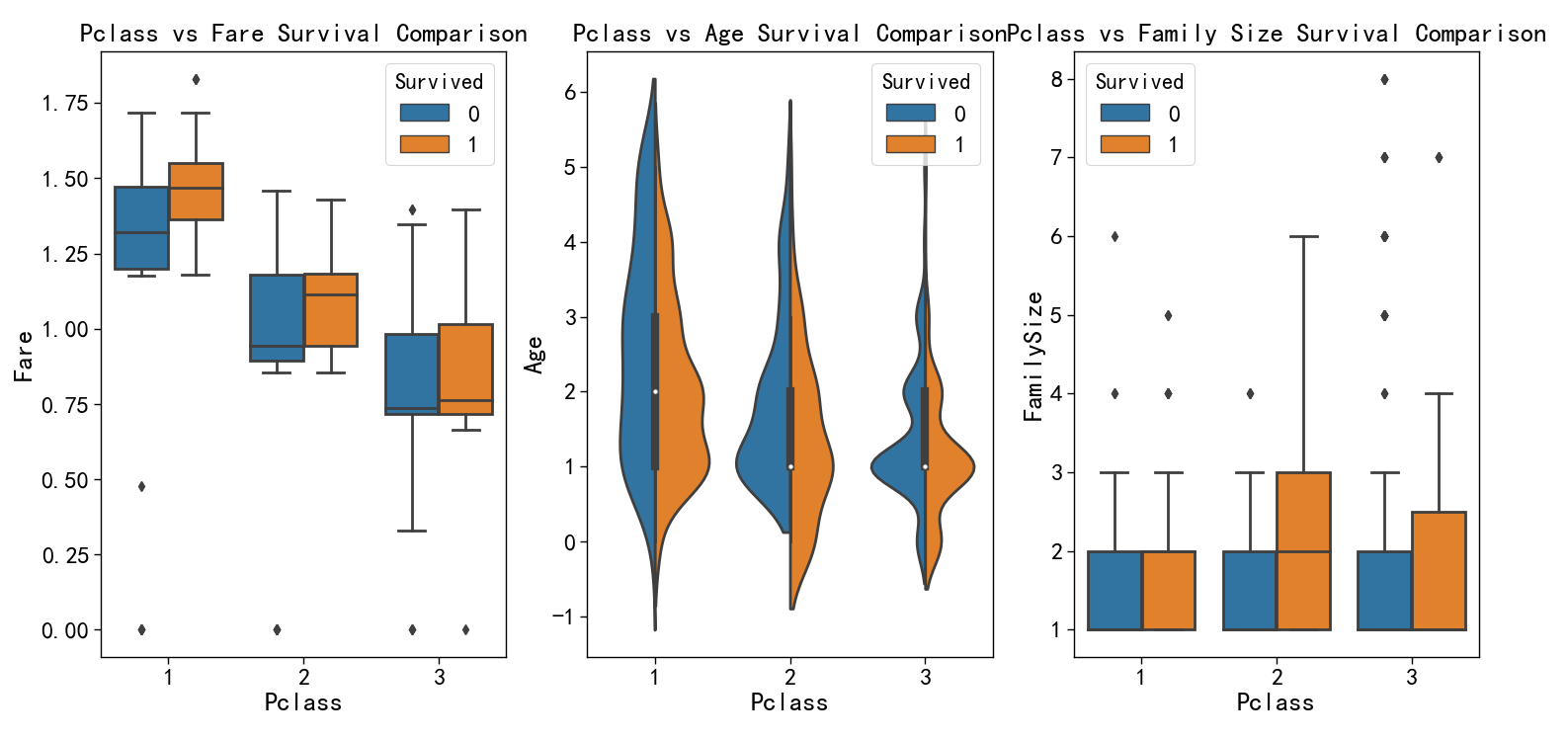
Observations: here are the survivors!

Family-size of 3 or 4 from first pivot

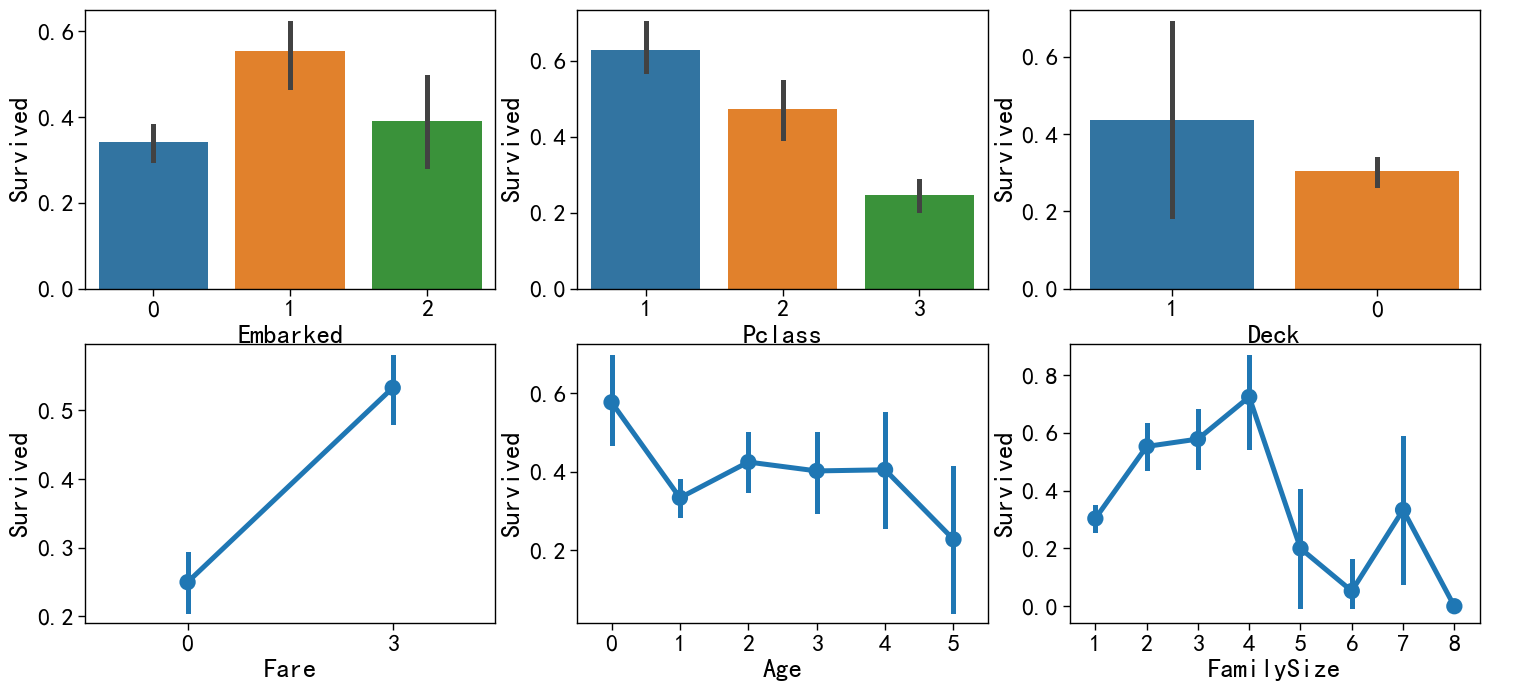
Women and men alone on first class (second pivot, red showing survival rate below 0.4)

Top-right in the graph above: first class and age categories 1 and 2

The not-so lucky are mostly in men, Pclass 3 and age category 1 (younger folks



**图12 舱位与小费、年龄、家庭成员的生存率分布图**



**图13 港口、舱位、甲板、小费、年龄、家庭成员与生存率分布图**

**Observations:**

* The colors represent: blue=0 is for women, green=1 for men
* Clearly, women had more chance of surviving, with or without cabin
* Interesting is that accompanied women without a cabin had less survival chance than women alone without cabin. But this is not true for men. Men alone have less chance than accompanied.

**Bottom-line: it would have been better for women without cabin to pretend that they were alone. And lone men should join a family to improve their survival rates.**

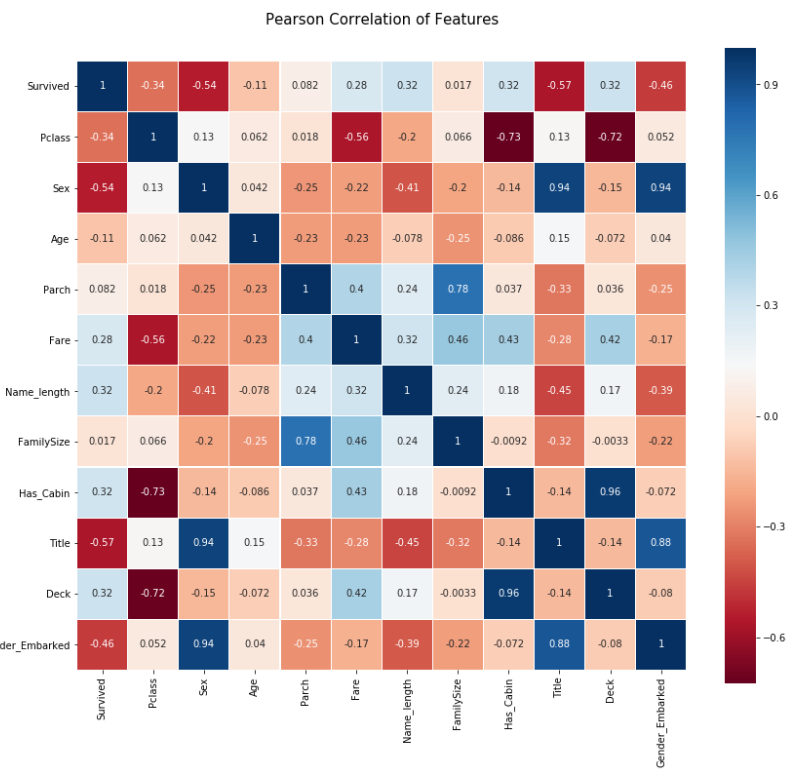
## 3.2. Dropping features

Bottom-line of the bi-variate and tri-variate analysis as well as the feature importance analysis (from running the classifiers multiple times), **I decided to drop less-relevant features**. This happened as an iterative process by reviwing the outcome of the feature importance graph in the next section. The problem with less important features is that they create more noice and actually take over the importance of real features like Sex and Pclass.

**The next step after dropping less-relevant features is to scale them, a very good recommendation from Konstantin's kernel** It helps to boost the score. Scaling features is helpful for many ML algorithms like KNN for example, it really boosts their score. Feature scaling is a method used to standardize the range of independent variables or features of data. In data processing, it is also known as data normalization. Feature standardization makes the values of each feature in the data have zero-mean (when subtracting the mean in the numerator) and unit-variance. The general method of calculation is to determine the distribution mean and standard deviation for each feature. Next we subtract the mean from each feature. Then we divide the values (mean is already subtracted) of each feature by its standard deviation.

## 3.3. Pearson Correlation Heatmap

The Seaborn plotting package allows us to plot heatmaps showing the Pearson product-moment correlation coefficient (PPMCC) correlation between features. Pearson is bivariate correlation, measuring the linear correlation between two features.



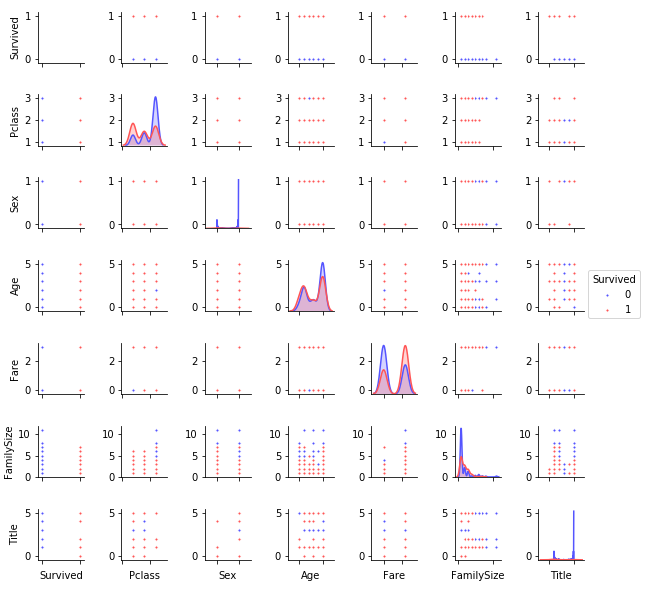
**图14 各因子相关关系热力图**

**Observations from the Pearson analysis:**

* Correlation coefficients with magnitude between 0.5 and 0.7 indicate variables which can be considered **moderately correlated**.
* We can see from the red cells that many features are "moderately" correlated: specifically, IsAlone, Pclass, Name\_length, Fare, Sex.
* This is influenced by the following two factors: 1) Women versus men (and the compounding effect of Name\_length) and 2) Passengers paying a high price (Fare) have a higher chance of survival: there are also in first class, have a title.

## 3.4. Pairplots

Finally let us generate some pairplots to observe the distribution of data from one feature to the other. The Seaborn pairplot class will help us visualize the distribution of a feature in relationship to each others.



**图15 各因子相关关系分布图**

**Observations**

* The pairplot graph all trivariate analysis into one figure.
* The clustering of red dots indicates the combination of two features results in higher survival rates, or the opposite (clustering of blue dots = lower survival) For example:
* Smaller family sizes in first and second class
* Middle age with Pclass in third category = only blue dot This can be used to validate that we extracted the right features or help us define new ones.

4. Predictive modelling, cross-validation, hyperparameters and ensembling

* 4.1. Logistic Regression
* 4.2. Support Vector Machines (supervised)
* 4.3. k-Nearest Neighbors algorithm (k-NN)
* 4.4. Naive Bayes classifier
* 4.5. Perceptron
* 4.6 Linear SVC
* 4.7 Stochastic Gradient Descent
* 4.8. Decision tree
* 4.9 Random Forrest
* 4.10 Model summary
* 4.11. Model cross-validation with K-Fold
* 4.12 Hyperparameter tuning & learning curves for selected classifiers
* 4.13 Selecting and combining the best classifiers
* 4.14 Ensembling
* 4.15. Summary of most important features

4.1. Logistic Regression

Logistic regression measures the relationship between the categorical dependent feature (in our case Survived) and the other independent features. It estimates probabilities using a cumulative logistic distribution:

* The first value shows the accuracy of this model
* The table after this shows the importance of each feature according this classifier.

logreg = LogisticRegression()

logreg.fit(X\_train, Y\_train)

Y\_pred1 = logreg.predict(x\_test)

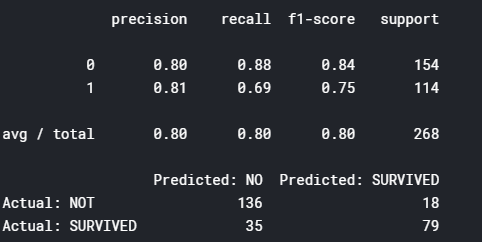
acc\_log = round(logreg.score(x\_test, y\_test) \* 100, 2)

from sklearn.metrics import confusion\_matrix, classification\_report

print(classification\_report(y\_test, Y\_pred1))

cm = pd.DataFrame(confusion\_matrix(y\_test, Y\_pred1), ['Actual: NOT', 'Actual: SURVIVED'], ['Predicted: NO', 'Predicted: SURVIVED'])

print(cm)



**Observation:**

* This classfier confirms the importance of Name\_length
* FamilySize did not show a strong Pearson correlation with Survived but comes here to the top. This can be due to its strong relationship with other features such as Is\_Alone or Parch (Parent-Children).

## 4.2. Support Vector Machines (supervised)

Given a set of training samples, each sample is marked as belonging to one or the other of two categories.

The SVM training algorithm builds a model that assigns new test samples to one category or the other, making it a non-probabilistic binary linear classifier.

svc=SVC()

svc.fit(X\_train, Y\_train)

Y\_pred2 = svc.predict(x\_test)

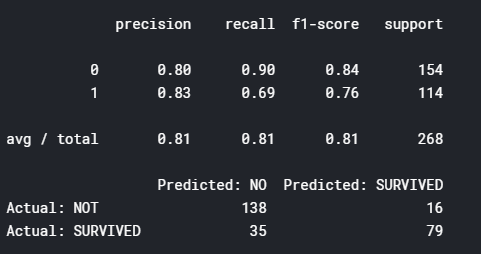
acc\_svc = round(svc.score(x\_test, y\_test) \* 100, 2)

acc\_svc

print(classification\_report(y\_test, Y\_pred2))

cm = pd.DataFrame(confusion\_matrix(y\_test, Y\_pred2), ['Actual: NOT', 'Actual: SURVIVED'], ['Predicted: NO', 'Predicted: SURVIVED'])

print(cm)



## 4.3. k-Nearest Neighbors algorithm (k-NN)

This is a non-parametric method used for classification and regression. A sample is classified by a majority vote of its neighbors, with the sample being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

knn = KNeighborsClassifier(algorithm='auto', leaf\_size=26, metric='minkowski',

metric\_params=None, n\_jobs=1, n\_neighbors=10, p=2,

weights='uniform')

knn.fit(X\_train, Y\_train)

knn\_predictions = knn.predict(x\_test)

acc\_knn = round(knn.score(x\_test, y\_test) \* 100, 2)

*# Preparing data for Submission 1*

test\_Survived = pd.Series(knn\_predictions, name="Survived")

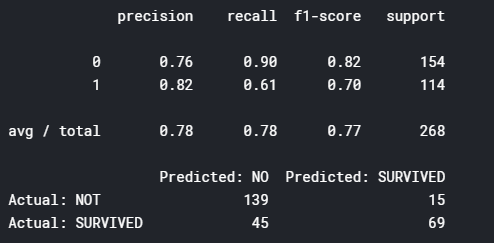
Submission1 = pd.concat([PassengerId,test\_Survived],axis=1)

acc\_knn

print(classification\_report(y\_test, knn\_predictions))

cm = pd.DataFrame(confusion\_matrix(y\_test, knn\_predictions), ['Actual: NOT', 'Actual: SURVIVED'], ['Predicted: NO', 'Predicted: SURVIVED'])

print(cm)



x\_trainknn, x\_testknn, y\_trainknn, y\_testknn = train\_test\_split(X\_train,Y\_train,test\_size = .33, random\_state = 0)

nn\_scores = []

best\_prediction = [-1,-1]

for i **in** range(1,100):

knn = KNeighborsClassifier(n\_neighbors=i, weights='distance', metric='minkowski', p =2)

knn.fit(x\_trainknn, y\_trainknn)

score = accuracy\_score(y\_testknn, knn.predict(x\_testknn))

*#print i, score*

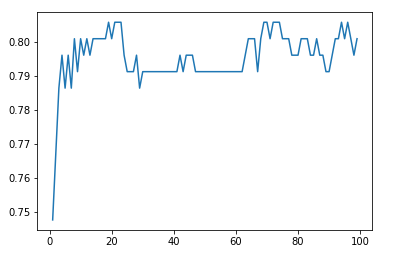
if score > best\_prediction[1]:

best\_prediction = [i, score]

nn\_scores.append(score)

print (best\_prediction)

plt.plot(range(1,100),nn\_scores)



**图16 迭代次数与预测精度关系图**

## 4.4. Naive Bayes classifier

This is a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features. Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of features in a learning problem.

gaussian = GaussianNB()

gaussian.fit(X\_train, Y\_train)

Y\_pred3 = gaussian.predict(x\_test)

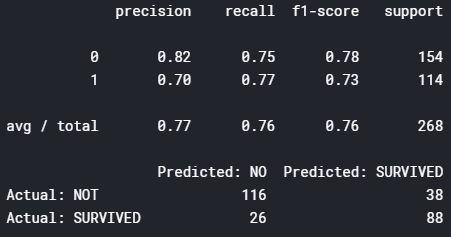
acc\_gaussian = round(gaussian.score(x\_test, y\_test) \* 100, 2)

acc\_gaussian

print(classification\_report(y\_test, Y\_pred3))

cm = pd.DataFrame(confusion\_matrix(y\_test, Y\_pred3), ['Actual: NOT', 'Actual: SURVIVED'], ['Predicted: NO', 'Predicted: SURVIVED'])

print(cm)



## 4.5. Perceptron

This is an algorithm for supervised learning of binary classifiers: like the other classifiers before, it decides whether an input, represented by a vector of numbers, belongs to some specific class or not. It is a type of linear classifier, i.e. a classification algorithm that makes its predictions based on a linear predictor function combining a set of weights with the feature vector. The algorithm allows for online learning, in that it processes elements in the training set one at a time.

perceptron = Perceptron()

perceptron.fit(X\_train, Y\_train)

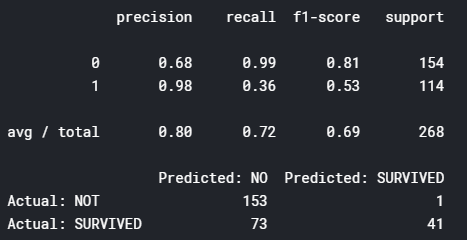
Y\_pred4 = perceptron.predict(x\_test)

acc\_perceptron = round(perceptron.score(x\_test, y\_test) \* 100, 2)

print(classification\_report(y\_test, Y\_pred4))

cm = pd.DataFrame(confusion\_matrix(y\_test, Y\_pred4), ['Actual: NOT', 'Actual: SURVIVED'], ['Predicted: NO', 'Predicted: SURVIVED'])

print(cm)



## 4.6. Linear SVC

This is another implementation of Support Vector Classification (similar to 4.2.) for the case of a linear kernel

linear\_svc = LinearSVC()

linear\_svc.fit(X\_train, Y\_train)

Y\_pred5 = linear\_svc.predict(x\_test)

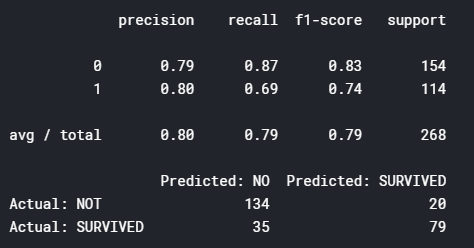
acc\_linear\_svc = round(linear\_svc.score(x\_test, y\_test) \* 100, 2)

acc\_linear\_svc

print(classification\_report(y\_test, Y\_pred5))

cm = pd.DataFrame(confusion\_matrix(y\_test, Y\_pred5), ['Actual: NOT', 'Actual: SURVIVED'], ['Predicted: NO', 'Predicted: SURVIVED'])

print(cm)



## 4.7. Stochastic Gradient Descent (sgd)

This is a stochastic approximation of the gradient descent optimization and iterative method for minimizing an objective function that is written as a sum of differentiable functions. In other words, SGD tries to find minima or maxima by iteration.

sgd = SGDClassifier()

sgd.fit(X\_train, Y\_train)

Y\_pred6 = sgd.predict(x\_test)

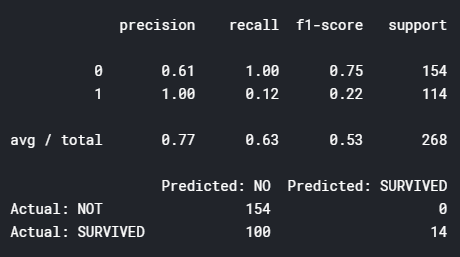
acc\_sgd = round(sgd.score(x\_test, y\_test) \* 100, 2)

acc\_sgd

print(classification\_report(y\_test, Y\_pred6))

cm = pd.DataFrame(confusion\_matrix(y\_test, Y\_pred6), ['Actual: NOT', 'Actual: SURVIVED'], ['Predicted: NO', 'Predicted: SURVIVED'])

print(cm)



## 4.8. Decision tree

This predictive model maps features (tree branches) to conclusions about the target value (tree leaves).

The target features take a finite set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

decision\_tree = DecisionTreeClassifier()

decision\_tree.fit(X\_train, Y\_train)

Y\_pred7 = decision\_tree.predict(x\_test)

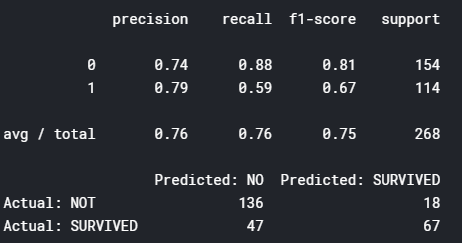
acc\_decision\_tree = round(decision\_tree.score(x\_test, y\_test) \* 100, 2)

acc\_decision\_tree

print(classification\_report(y\_test, Y\_pred7))

cm = pd.DataFrame(confusion\_matrix(y\_test, Y\_pred7), ['Actual: NOT', 'Actual: SURVIVED'], ['Predicted: NO', 'Predicted: SURVIVED'])

print(cm)



## 4.9. Random Forests

This is one of the most popular classfier. Random forests or random decision forests are an ensemble learning method for classification, regression and other tasks, that operate by constructing a multitude of decision trees (n\_estimators=100) at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees

random\_forest = RandomForestClassifier(n\_estimators=100)

random\_forest.fit(X\_train, Y\_train)

random\_forest\_predictions = random\_forest.predict(x\_test)

acc\_random\_forest = round(random\_forest.score(x\_test, y\_test) \* 100, 2)

*# Preparing data for Submission 2*

test\_Survived = pd.Series(random\_forest\_predictions, name="Survived")

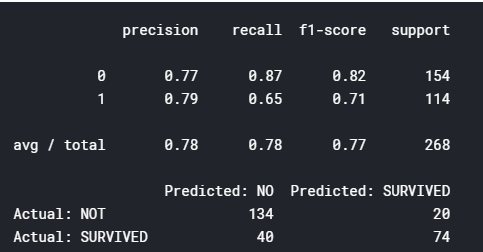
Submission2 = pd.concat([PassengerId,test\_Survived],axis=1)

acc\_random\_forest

print(classification\_report(y\_test, random\_forest\_predictions))

cm = pd.DataFrame(confusion\_matrix(y\_test, random\_forest\_predictions), ['Actual: NOT', 'Actual: SURVIVED'], ['Predicted: NO', 'Predicted: SURVIVED'])

print(cm)

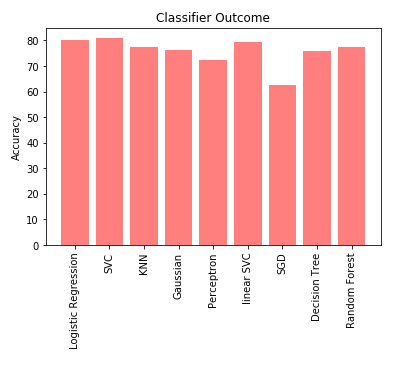


## 4.10. Model summary

I found that the picture illustrates the various model better than words. This should be taken with a grain of salt, as the intuition conveyed by these two-dimensional examples does not necessarily carry over to real datasets. The reality os that the algorithms work with many dimensions (11 in our case).

But it shows how each classifier algorithm partitions the same data in different ways. The three rows represent the three different data set on the right. The plots show training points in solid colors and testing points semi-transparent. The lower right shows the classification accuracy on the test set.

For instance, the visualization helps understand how RandomForest uses multiple Decision Trees, the linear SVC, or Nearest Neighbors grouping sample by their relative distance to each others.



**图17 各种模型精度分布图**

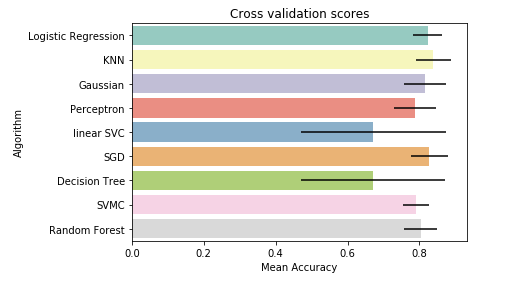
**Observations**

* The above models (classifiers) were applied to a split training and x\_test datasets.
* This results in some classifiers (Decision\_tree and Random\_Forest) over-fitting the model to the training data.
* This happens when the classifiers use many input features (to include noise in each feature) on the complete dataset, and ends up “memorizing the noise” instead of finding the signal.
* This overfit model will then make predictions based on that noise. It performs unusually well on its training data, but will not necessarilyimprove the prediction quality with new data from the test dataset.
* In the next section, we will cross-validate the models using sample data against each others. We will this by using StratifiedKFold to train and test the models on sample data from the overall dataset. Stratified K-Folds is a cross validation iterator. It provides train/test indices to split data in train test sets. This cross-validation object is a variation of KFold, which returns stratified folds. The folds are made by preserving the percentage of samples for each class.

5. Evaluation

## 5.1. Model cross-validation with K-Fold

The fitting process applied above optimizes the model parameters to make the model fit the training data as well as possible. Cross-validation is a way to predict the fit of a model to a hypothetical validation set when an explicit validation set is not available. In simple words, it allows to test how well the model performs on new data. In our case, cross-validation will also be applied to compare the performances of different predictive modeling procedures.



**图18各模型交叉分布图**

## 5.2 Hyperparameter tuning & learning curves for selected classifiers

**5.1.1. Adaboost** is used in conjunction with many other types of learning algorithms to improve performance. The output of the other learning algorithms ('weak learners') is combined into a weighted sum that represents the final output of the boosted classifier. AdaBoost is adaptive in the sense that subsequent weak learners are tweaked in favor of those instances misclassified by previous classifiers. AdaBoost is sensitive to noisy data and outliers.

**5.1.2. ExtraTrees** implements a meta estimator that fits a number of randomized decision trees (a.k.a. extra-trees) on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting.

**5.1.3. RandomForest**operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

**5.1.4. GradientBoost**produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

**5.1.5. SVMC, or Support Vector Machines.**vGiven a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier.

**Observations to fine-tune our models**

First, let's compare their best score after fine-tuning their parameters:

1. Adaboost: 80
2. ExtraTrees: 83
3. RandomForest: 82
4. GradientBoost: 82
5. SVC: 83

It appears that GBC and SVMC are doing the best job on the Train data. This is good because we want to keep the model as close to the training data as possible. But not too close! The two major sources of error are bias and variance; as we reduce these two, then we could build more accurate models:

* **Bias**: The less biased a method, the greater its ability to fit data well.
* **Variance**: with a lower bias comes typically a higher the variance. And therefore the risk that the model will not adapt accurately to new test data. This is the case here with Gradient Boost: high score but cross-validation is very distant.

The reverse also holds: the greater the bias, the lower the variance. A high-bias method builds simplistic models that generally don't fit well training data. We can see the red and green curves from ExtraTrees, RandomForest and SVC are pretty close. **This points to a lower variance, i.e. a stronger ability to apply the model to new data.**

I used the above graphs to optimize the parameters for Adaboost, ExtraTrees, RandomForest, GradientBoost and SVC. This resulted in a significant improvement of the prediction accuracy on the test data (score).

In addition, I found out that AdaBoost does not do a good job with this dataset as the training score and cross-validation score are quite far apart.

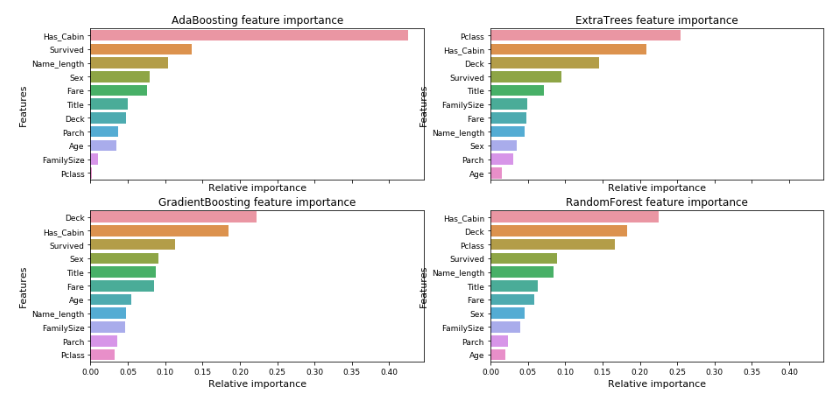
## 5.3 Selecting and combining the best classifiers

We will first compare in the next section the classifiers; results between themselves and applied to the same test data. Then "ensemble" them together with an automatic function callled voting.

**Observations:**

* As indicated before, Adaboost has the lowest correlations when compared to other predictors. This indicates that it predicts differently than the others when it comes to the test data.
* We will therefore 'ensemble' the remaining four predictors.

## 5.4. Summary of most important features



**图19 各种算法的特征值重要性分布图**

Nice graphics, but the obsevation is unclear in my opinion:

* On one side, we hope as analyst that the models come out with similar patterns. An easy direction to follow.
* At the same time, "there have been quite a few articles and Kaggle competition winner stories about the merits of having trained models that are more uncorrelated with one another producing better scores". As we say in business, diversity brings better results, this seems to be true with algorithms as well!