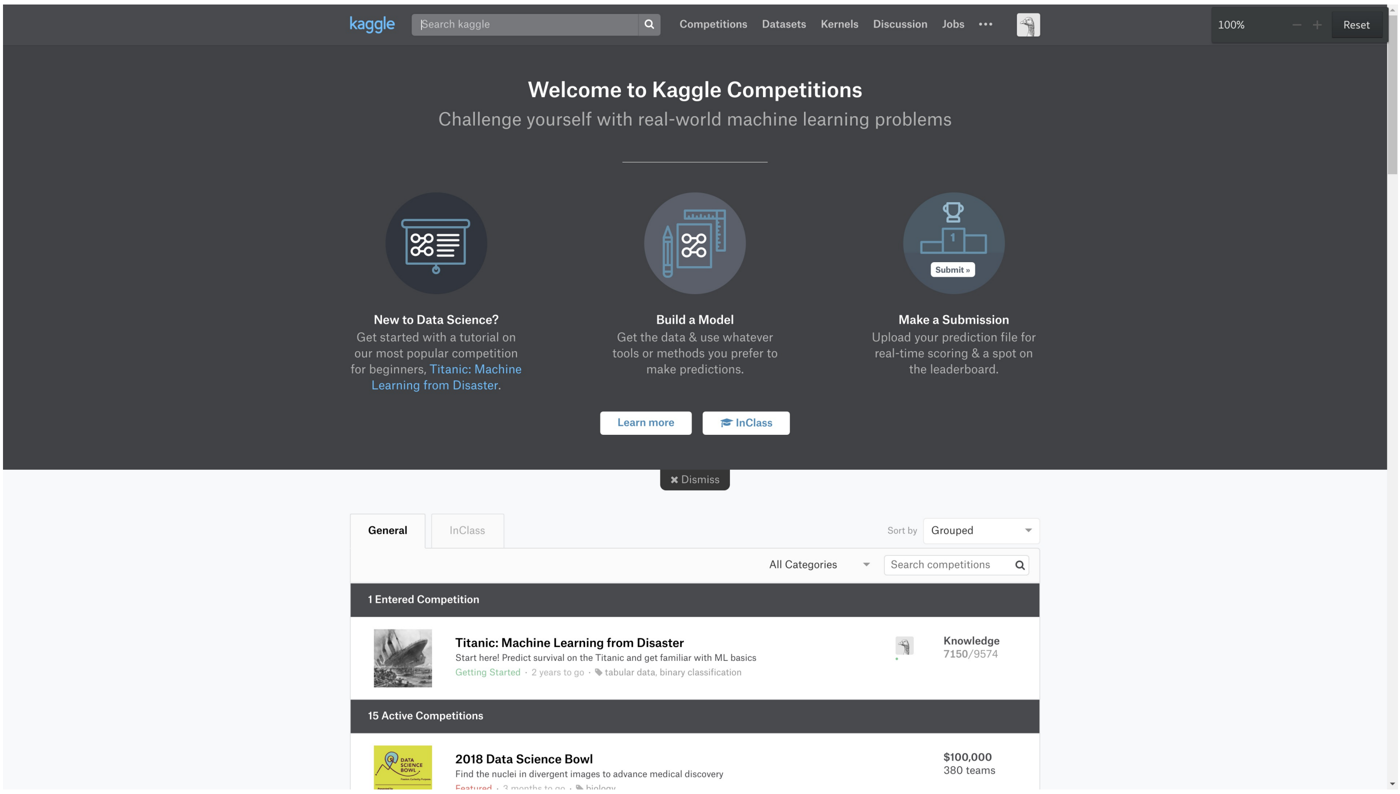
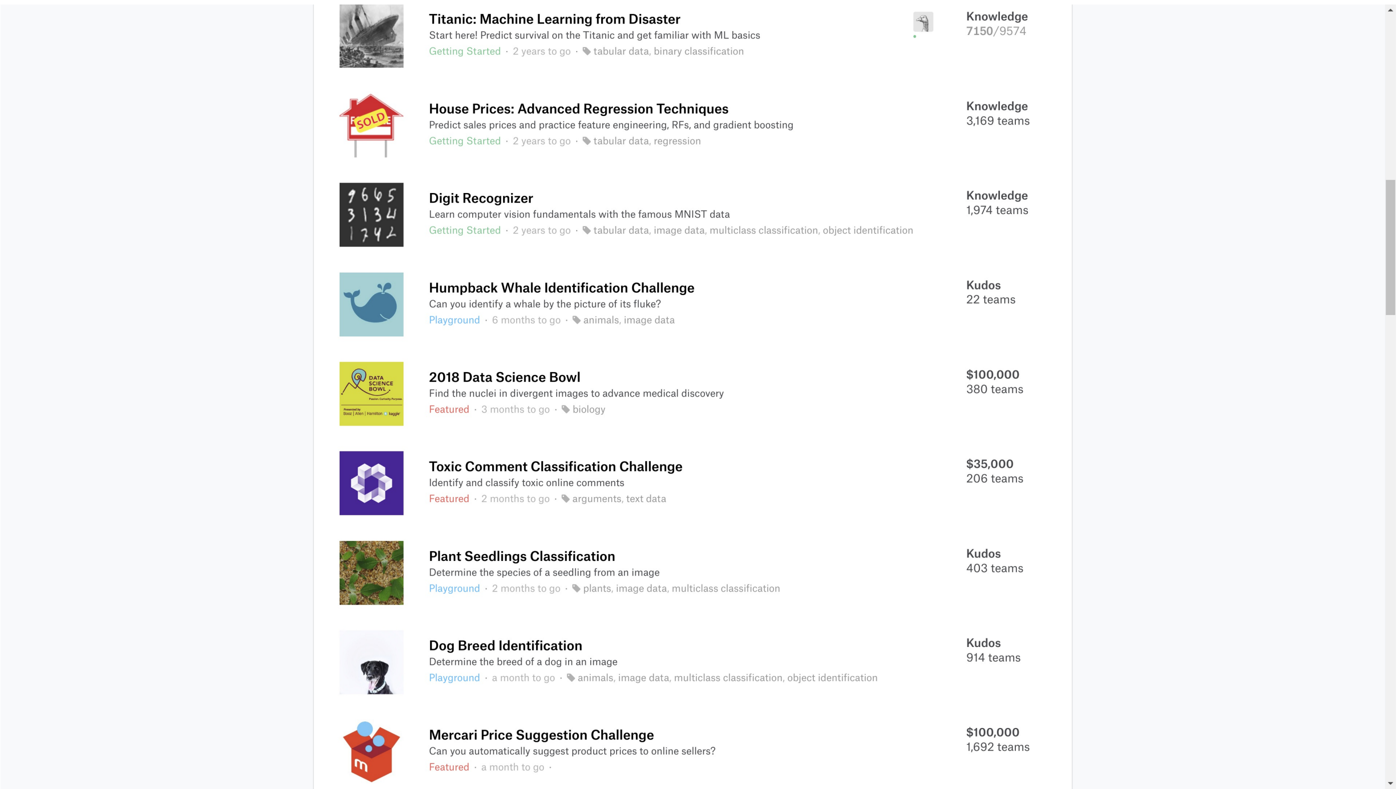
專題計劃書

Kaggle 機器學習競賽

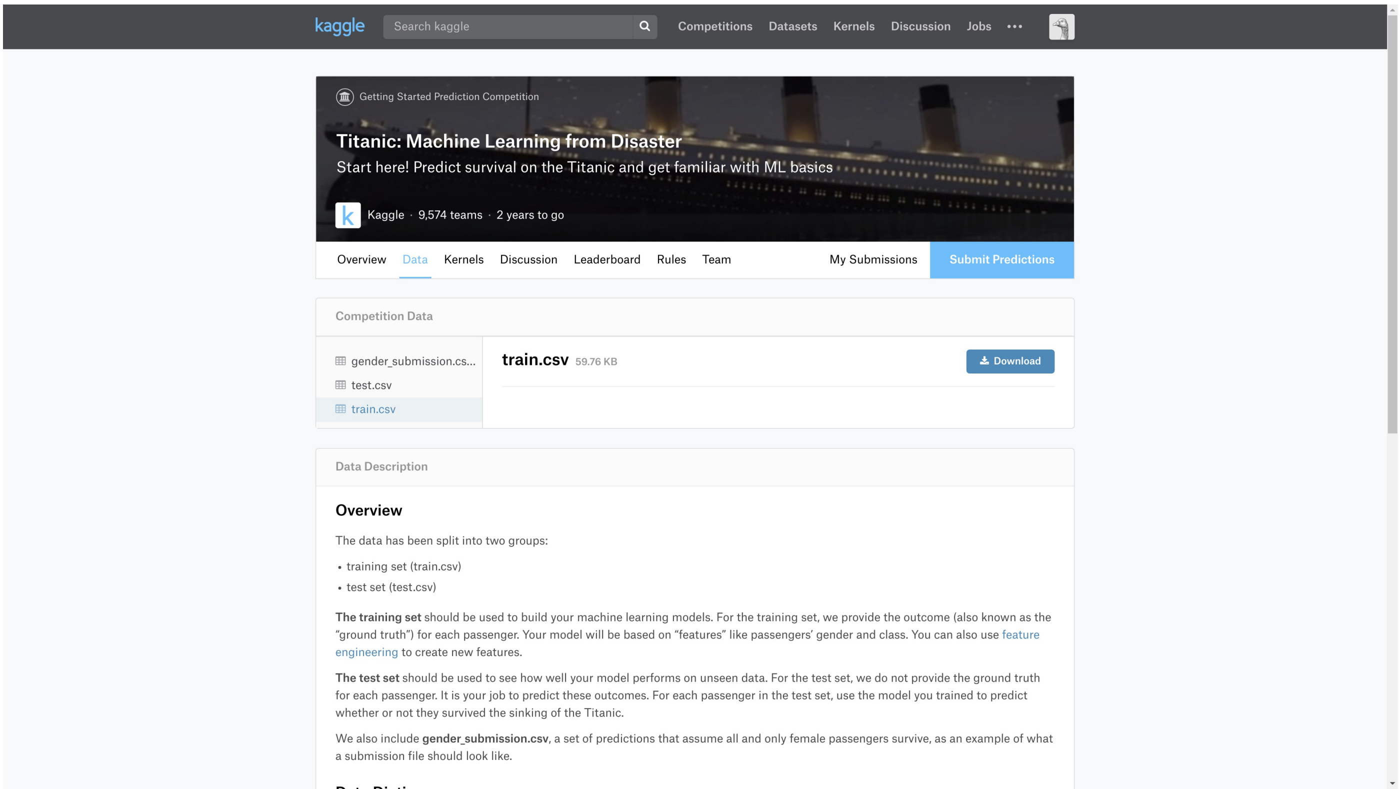
陳鐸元 呂振麒 李宓

選定題目





下載 data set



預測

預備

```
# data analysis and wrangling
import pandas as pd
```

```
import numpy as np
import random as rnd

# Resampling
from imblearn.under_sampling import NearMiss
from imblearn.under_sampling import RandomUnderSampler

# visualization
import seaborn as sns
import matplotlib as mpl
import matplotlib.pyplot as plt
import matplotlib.pylab as pylab
%matplotlib inline

# machine learning
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC, LinearSVC
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_validate
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.feature_selection import RFECV


# session
import pickle
```

```
# helper functions
def plot_histograms( df , variables , n_rows , n_cols ):
    fig = plt.figure( figsize = ( 16 , 12 ) )
    for i, var_name in enumerate( variables ):
        ax=fig.add_subplot( n_rows , n_cols , i+1 )
        df[ var_name ].hist( bins=10 , ax=ax )
        ax.set_title( 'Skew: ' + str( round( float( df[ var_name ].skew() ) , ) ) ) #
+ ' ' + var_name ) #var_name+" Distribution")
        ax.set_xticklabels( [] , visible=False )
        ax.set_yticklabels( [] , visible=False )
    fig.tight_layout()  # Improves appearance a bit.
    plt.show()

def plot_distribution( df , var , target , **kwargs ):
    row = kwargs.get( 'row' , None )
    col = kwargs.get( 'col' , None )
    facet = sns.FacetGrid( df , hue=target , aspect=4 , row = row , col = col )
    facet.map( sns.kdeplot , var , shade= True )
    facet.set( xlim=( 0 , df[ var ].max() ) )
    facet.add_legend()
```

```python
def plot_categories( df , cat , target , **kwargs ):
    row = kwargs.get( 'row' , None )
    col = kwargs.get( 'col' , None )
    facet = sns.FacetGrid( df , row = row , col = col )
    facet.map( sns.barplot , cat , target )
    facet.add_legend()

def plot_correlation_map( df ):
    corr = df.corr()
    _ , ax = plt.subplots( figsize =( 12 , 10 ) )
    cmap = sns.diverging_palette( 220 , 10 , as_cmap = True )
    _ = sns.heatmap(
        corr,
        cmap = cmap,
        square=True,
        cbar_kws={ 'shrink' : .9 },
        ax=ax,
        annot = True,
        annot_kws = { 'fontsize' : 12 }
    )

def describe_more( df ):
    var = [] ; l = [] ; t = []
    for x in df:
        var.append( x )
        l.append( len( pd.value_counts( df[ x ] ) ) )
        t.append( df[ x ].dtypes )
    levels = pd.DataFrame( { 'Variable' : var , 'Levels' : l , 'Datatype' : t } )
    levels.sort_values( by = 'Levels' , inplace = True )
    return levels

def plot_variable_importance( X , y ):
    tree = RandomForestClassifier( n_estimators = 10, n_jobs = -1 )
    tree.fit( X , y )
    plot_model_var_imp( tree , X , y )

def plot_model_var_imp( model , X , y ):
    imp = pd.DataFrame(
        model.feature_importances_  ,
        columns = [ 'Importance' ] ,
        index = X.columns
    )
    imp = imp.sort_values( [ 'Importance' ] , ascending = True )
    imp[ : 10 ].plot( kind = 'barh' )
    print (model.score( X , y ))

def plot_RFECV( X , y ):
    # Create the RFE object and compute a cross-validated score.
    svc = SVC(kernel="linear", n_jobs = -1)
    # The "accuracy" scoring is proportional to the number of correct
    # classifications
    rfecv = RFECV(estimator=svc, step=1, cv=StratifiedKFold(3),
                  scoring='accuracy')
    rfecv.fit(X, y)

    print("Optimal number of features : %d" % rfecv.n_features_)

    # Plot number of features VS. cross-validation scores
    plt.figure()
    plt.xlabel("Number of features selected")
    plt.ylabel("Cross validation score (nb of correct classifications)")
    plt.plot(range(1, len(rfecv.grid_scores_) + 1), rfecv.grid_scores_)
    plt.show()
```

載入 data set

```
# load data
train = pd.read_csv('./train.csv', low_memory=False)
test = pd.read_csv('./test.csv', low_memory=False)
```

分析

```
# analysis
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
```

```
train.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs	female	38.0	1	0

				Th...				
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0

```
train.describe()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	PassengerId	Survived	Pclass	Age	SibSp	Parch
count	891.000000	891.000000	891.000000	714.000000	891.000000	891.00000
mean	446.000000	0.383838	2.308642	29.699118	0.523008	0.381594
std	257.353842	0.486592	0.836071	14.526497	1.102743	0.806057
min	1.000000	0.000000	1.000000	0.420000	0.000000	0.000000
25%	223.500000	0.000000	2.000000	20.125000	0.000000	0.000000
50%	446.000000	0.000000	3.000000	28.000000	0.000000	0.000000
75%	668.500000	1.000000	3.000000	38.000000	1.000000	0.000000
max	891.000000	1.000000	3.000000	80.000000	8.000000	6.000000

```
train.describe(include=['O'])
```

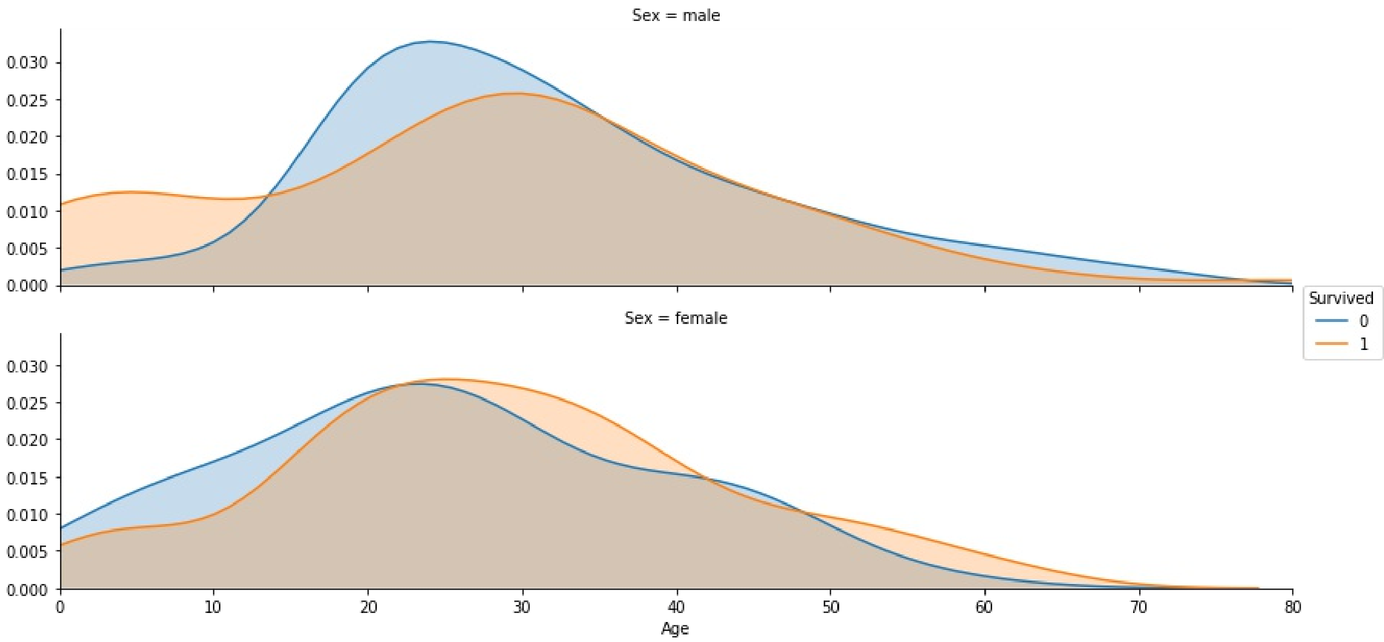
```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

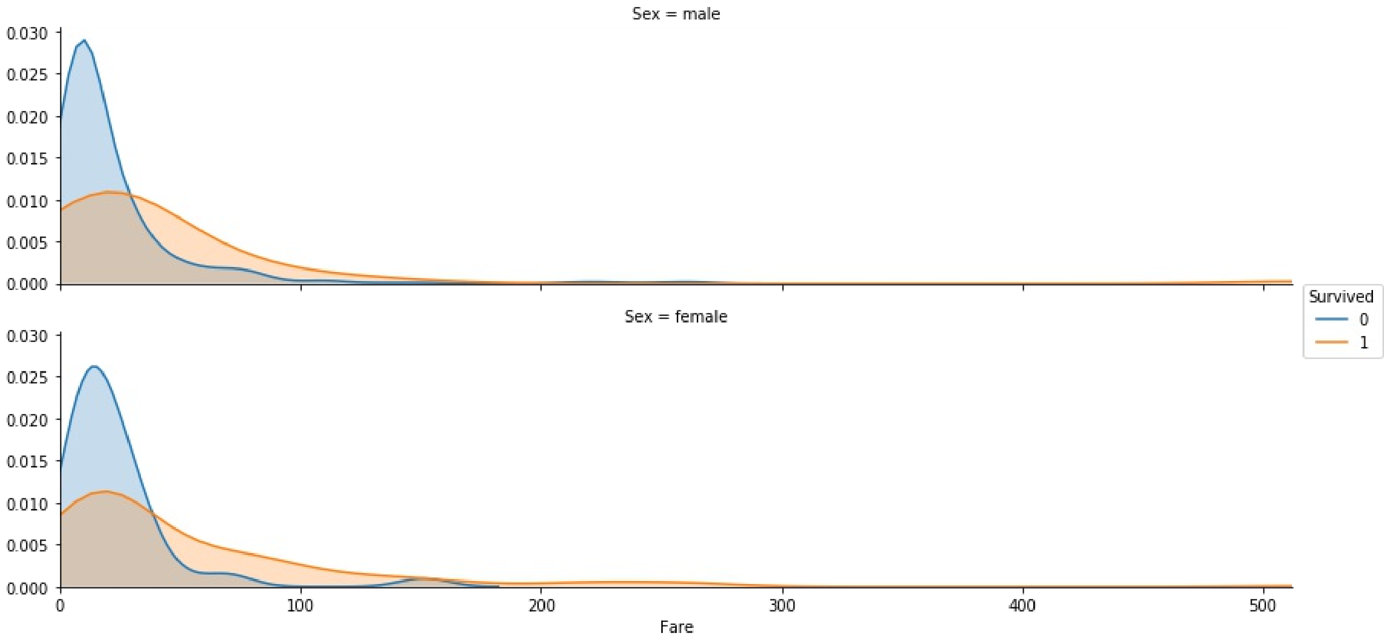
	Name	Sex	Ticket	Cabin	Embarked
count	891	891	891	204	889
unique	891	2	681	147	3
top	Olsson, Mr. Nils Johan Goransson	male	CA. 2343	B96 B98	S
freq	1	577	7	4	644

```
plot_correlation_map( train )
```


```
plot_distribution( train , var = 'Age' , target = 'Survived' , row = 'Sex' )
```

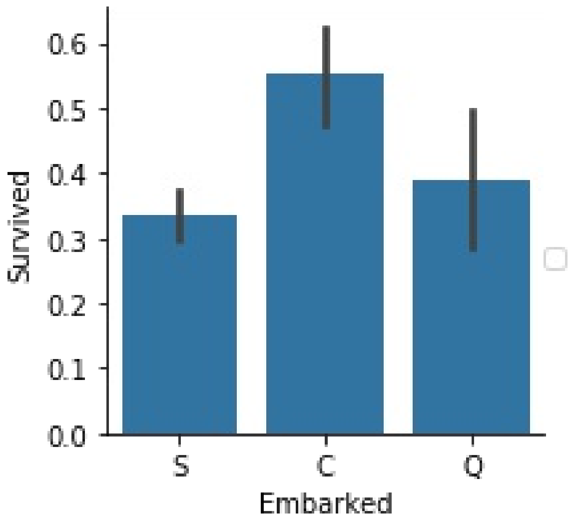



```
plot_distribution( train , var = 'Fare' , target = 'Survived' , row = 'Sex' )
```



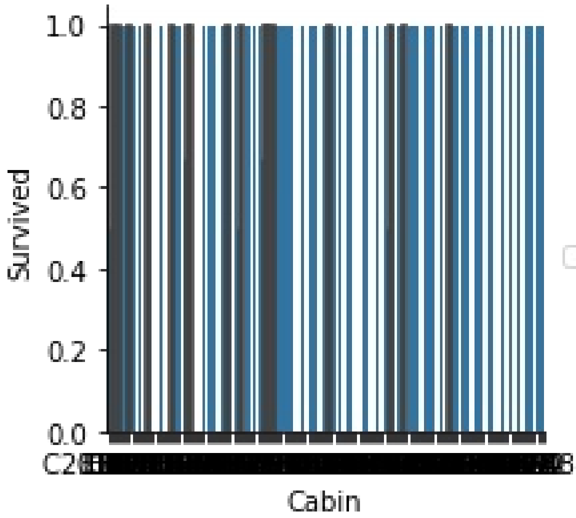

```
plot_categories( train , cat = 'Embarked' , target = 'Survived' )
```

```
/hdd/home/superdanby/Github/Kaggle/venv/lib/python3.6/site-
packages/seaborn/axisgrid.py:703: UserWarning: Using the barplot function without
specifying `order` is likely to produce an incorrect plot.
  warnings.warn(warning)
```



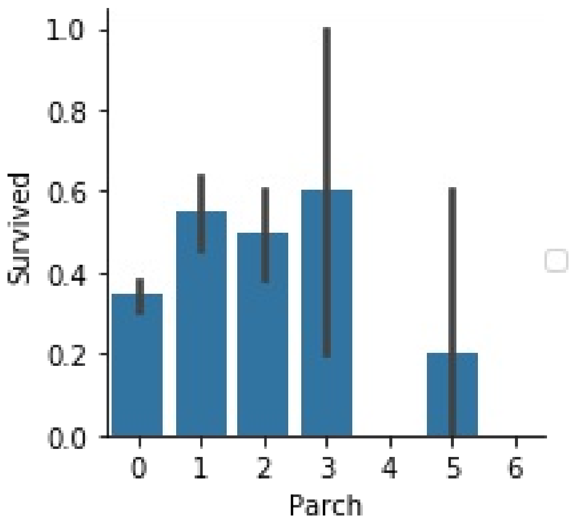

```
plot_categories( train , cat = 'Cabin' , target = 'Survived' )
```

```
/hdd/home/superdanby/Github/Kaggle/venv/lib/python3.6/site-
packages/seaborn/axisgrid.py:703: UserWarning: Using the barplot function without
specifying `order` is likely to produce an incorrect plot.
  warnings.warn(warning)
```



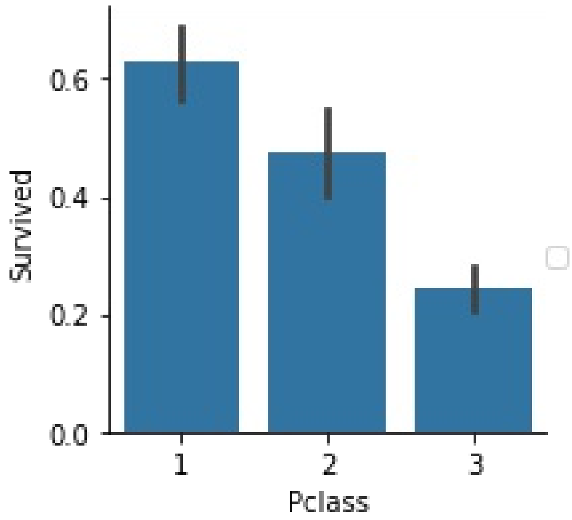

```
plot_categories( train , cat = 'Parch' , target = 'Survived' )
```

```
/hdd/home/superdanby/Github/Kaggle/venv/lib/python3.6/site-
packages/seaborn/axisgrid.py:703: UserWarning: Using the barplot function without
specifying `order` is likely to produce an incorrect plot.
  warnings.warn(warning)
```



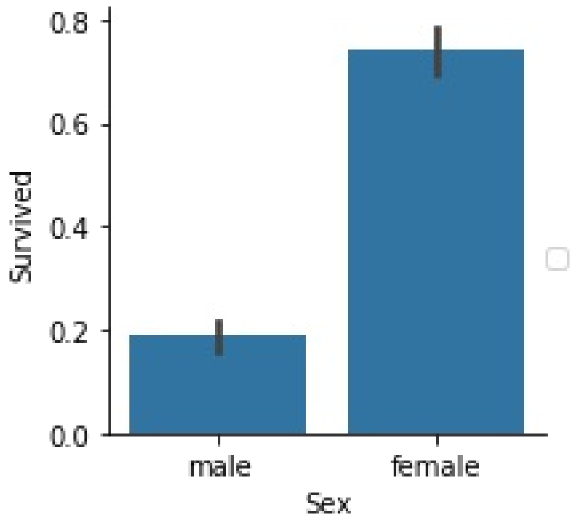

```
plot_categories( train , cat = 'Pclass' , target = 'Survived' )
```

```
/hdd/home/superdanby/Github/Kaggle/venv/lib/python3.6/site-
packages/seaborn/axisgrid.py:703: UserWarning: Using the barplot function without
specifying `order` is likely to produce an incorrect plot.
  warnings.warn(warning)
```



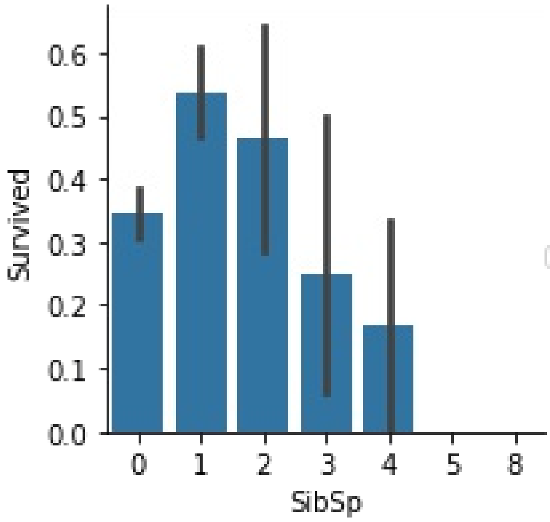

```
plot_categories( train , cat = 'Sex' , target = 'Survived' )
```

```
/hdd/home/superdanby/Github/Kaggle/venv/lib/python3.6/site-
packages/seaborn/axisgrid.py:703: UserWarning: Using the barplot function without
specifying `order` is likely to produce an incorrect plot.
  warnings.warn(warning)
```




```
plot_categories( train , cat = 'SibSp' , target = 'Survived' )
```

```
/hdd/home/superdanby/Github/Kaggle/venv/lib/python3.6/site-
packages/seaborn/axisgrid.py:703: UserWarning: Using the barplot function without
specifying `order` is likely to produce an incorrect plot.
  warnings.warn(warning)
```




```
# combine test and train
test['Survived'] = -1
whole = train.append(test, ignore_index = True)
```

```
# label encoding
```

將字串轉成數字型態

```
le = LabelEncoder()

# dropped = whole.drop(['Name', 'Ticket', 'PassengerId'], axis = 1).iloc[:,:]
dropped = whole.drop(['Name', 'Ticket', 'Cabin', 'PassengerId'], axis = 1).iloc[:,:]

# dropped['Cabin'] = le.fit_transform(dropped['Cabin'].astype(str))

dropped['Embarked'] = le.fit_transform(dropped['Embarked'].astype(str))

dropped['Sex'] = le.fit_transform(dropped['Sex'].astype(str))
```

```
# NAN

# Fill missing values of Age with the average of Age (mean)
dropped[ 'Age' ] = dropped.Age.fillna( dropped.Age.mean() )

# Fill missing values of Fare with the average of Fare (mean)
dropped[ 'Fare' ] = dropped.Fare.fillna( dropped.Fare.mean() )

# Fill missing values of Parch with the average of Parch (mean)
dropped[ 'Parch' ] = dropped.Parch.fillna( dropped.Parch.mean() )

# Fill missing values of SibSp with the average of SibSp (mean)
dropped[ 'SibSp' ] = dropped.SibSp.fillna( dropped.SibSp.mean() )
```

```
# separate train data and test data
trle = dropped[dropped['Survived'] != -1]

trle_X = trle.drop(['Survived'], axis = 1)

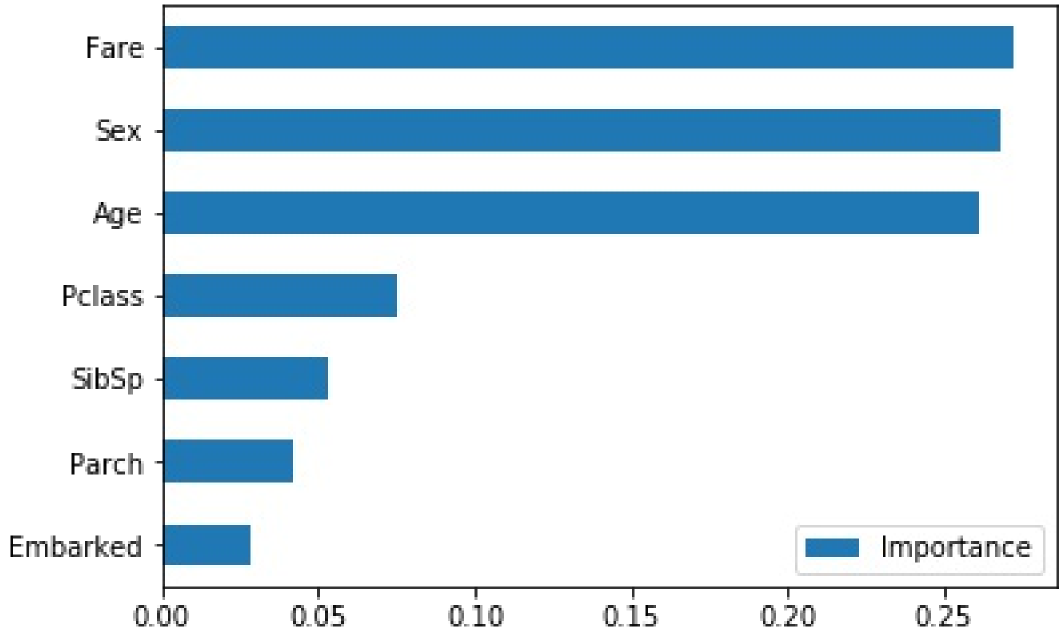
trle_Y = trle.Survived
```

```
tsle = dropped[dropped['Survived'] == -1].drop(['Survived'], axis = 1)
```

利用隨機森林選取重要特徵

```
# feature importance
plot_variable_importance(trle_X, trle_Y)
```

```
0.967452300786
```



Model: Random Forest

```
# RandomForest
param_grid = [{'n_estimators': [10, 50, 100], 'class_weight': [{0:1, 1:1},
'balanced']},
             ]
RF = GridSearchCV(RandomForestClassifier(), param_grid, verbose = 0, n_jobs = -1,
scoring = 'accuracy', cv = StratifiedKFold(5), return_train_score = True)

RF.fit(trle_X, trle_Y)
```

```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
       error_score='raise',
       estimator=RandomForestClassifier(bootstrap=True, class_weight=None,
criterion='gini',
            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, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid=[{'n_estimators': [10, 50, 100], 'class_weight': [{0: 1, 1: 1},
'balanced']}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
```

```
        scoring='accuracy', verbose=0)
```

```
print('Best Param:', RF.best_params_)

print('Best Model Score:', RF.best_estimator_.score(trle_X, trle_Y))

display(pd.DataFrame(RF.cv_results_))
```

```
Best Param: {'class_weight': {0: 1, 1: 1}, 'n_estimators': 100}
Best Model Score: 0.982042648709
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_class_wei
0	0.015140	0.001511	0.810325	0.967739	{0: 1, 1: 1}
1	0.067461	0.004365	0.808081	0.984009	{0: 1, 1: 1}
2	0.112714	0.006533	0.814815	0.984569	{0: 1, 1: 1}
3	0.014472	0.001258	0.792368	0.968856	balanced
4	0.040834	0.002599	0.809203	0.984569	balanced
5	0.078288	0.004848	0.814815	0.984569	balanced

6 rows × 22 columns

```
# output
tsrf_Y = RF.predict( tsle )

passenger_id = test.PassengerId

tsrf = pd.DataFrame( { 'PassengerId': passenger_id , 'Survived': tsrf_Y } )

tsrf.shape

tsrf.head()

tsrf.to_csv( 'RandomForest.csv' , index = False )
```

Model: Gradient Boosting

```
# GradientBoosting
param_grid = [{'loss': ['deviance', 'exponential'], 'n_estimators': [10, 100, 200],
'learning_rate': [0.1, 0.5, 0.9]},
             ]
GBC = GridSearchCV(GradientBoostingClassifier(), param_grid, verbose = 0, n_jobs =
-1, scoring = 'accuracy', cv = StratifiedKFold(5), return_train_score = True)

GBC.fit(trle_X, trle_Y)
```

```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
       error_score='raise',
       estimator=GradientBoostingClassifier(criterion='friedman_mse', init=None,
              learning_rate=0.1, loss='deviance', max_depth=3,
              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, n_estimators=100,
              presort='auto', random_state=None, subsample=1.0, verbose=0,
              warm_start=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid=[{'loss': ['deviance', 'exponential'], 'n_estimators': [10, 100,
200], 'learning_rate': [0.1, 0.5, 0.9]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='accuracy', verbose=0)
```

```
print('Best Param:', GBC.best_params_)

print('Best Model Score:', GBC.best_estimator_.score(trle_X, trle_Y))

display(pd.DataFrame(GBC.cv_results_))
```

```
Best Param: {'learning_rate': 0.1, 'loss': 'exponential', 'n_estimators': 200}
Best Model Score: 0.910213243547
```

```
.dataframe tbody tr th {
    vertical-align: top;
}
```



```
.dataframe thead th {
    text-align: right;
}
```

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_learning
0	0.008627	0.000656	0.817059	0.836137	0.1
1	0.059809	0.000877	0.827160	0.905727	0.1
2	0.113517	0.000908	0.826038	0.932384	0.1
3	0.007944	0.000501	0.811448	0.831934	0.1
4	0.049367	0.000530	0.821549	0.894502	0.1
5	0.083615	0.000708	0.828283	0.925085	0.1
6	0.004666	0.000282	0.827160	0.878507	0.5
7	0.039287	0.000497	0.820426	0.971380	0.5
8	0.083753	0.000732	0.801347	0.982887	0.5
9	0.004879	0.000266	0.806958	0.859988	0.5

10	0.042231	0.000484	0.819304	0.964369	0.5
11	0.084677	0.000713	0.808081	0.980641	0.5
12	0.004554	0.000281	0.813692	0.888887	0.9
13	0.047820	0.000604	0.801347	0.980642	0.9
14	0.079258	0.000751	0.809203	0.984569	0.9
15	0.004829	0.000265	0.826038	0.880467	0.9
16	0.042412	0.000487	0.804714	0.980361	0.9
17	0.085332	0.000712	0.810325	0.984569	0.9

18 rows × 23 columns

```
# output
tsgbc_Y = GBC.predict( tsle )

passenger_id = test.PassengerId

tsgbc = pd.DataFrame( { 'PassengerId': passenger_id , 'Survived': tsgbc_Y } )

tsgbc.shape

tsgbc.head()

tsgbc.to_csv( 'GradientBoost.csv' , index = False )
```

Model: Logistic Regression

```
# LogisticRegression
param_grid = [{'penalty': ['l1'], 'C': [0.1, 1, 10], 'solver': ['liblinear', 'saga'],
'class_weight': [{0:1, 1:1}, 'balanced']},
              {'penalty': ['l2'], 'C': [0.1, 1, 10], 'solver': ['newton-cg', 'sag',
'lbfgs'], 'class_weight': [{0:1, 1:1}, 'balanced']},
             ]
LR = GridSearchCV(LogisticRegression(), param_grid, verbose = 0, n_jobs = -1, scoring
= 'accuracy', cv = StratifiedKFold(5), return_train_score = True)

LR.fit(trle_X, trle_Y)
    GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
           error_score='raise',
           estimator=LogisticRegression(C=1.0, class_weight=None, dual=False,
fit_intercept=True,
              intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
              penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
              verbose=0, warm_start=False),
           fit_params=None, iid=True, n_jobs=-1,
           param_grid=[{'penalty': ['l1'], 'C': [0.1, 1, 10], 'solver': ['liblinear',
'saga'], 'class_weight': [{0: 1, 1: 1}, 'balanced']}, {'penalty': ['l2'], 'C': [0.1,
1, 10], 'solver': ['newton-cg', 'sag', 'lbfgs'], 'class_weight': [{0: 1, 1: 1},
'balanced']}],
           pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
           scoring='accuracy', verbose=0)



```python
print('Best Param:', LR.best_params_)

print('Best Model Score:', LR.best_estimator_.score(trle_X, trle_Y))

display(pd.DataFrame(LR.cv_results_))
```

```
Best Param: {'C': 0.1, 'class_weight': {0: 1, 1: 1}, 'penalty': 'l1', 'solver':
'liblinear'}
Best Model Score: 0.792368125701
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C
0	0.004839	0.000484	0.794613	0.791249	0.1
```

1	0.021139	0.000424	0.691358	0.690801	0.1	
2	0.004097	0.000411	0.780022	0.781985	0.1	
3	0.018132	0.000366	0.690236	0.691645	0.1	
4	0.003667	0.000255	0.786756	0.803312	1	
5	0.013572	0.000293	0.690236	0.689118	1	
6	0.003973	0.000249	0.778900	0.789845	1	
7	0.018356	0.000398	0.694725	0.696134	1	
8	0.004004	0.000258	0.784512	0.802752	10	
9	0.015304	0.000328	0.690236	0.689679	10	
10	0.005029	0.000329	0.775533	0.792089	10	
	0.017352	0.000369	0.694725	0.696133	10	

11						
12	0.009852	0.000267	0.785634	0.805838	0.1	
13	0.010501	0.000285	0.689113	0.689118	0.1	
14	0.011271	0.000252	0.786756	0.806400	0.1	
15	0.011643	0.000279	0.785634	0.793773	0.1	
16	0.013642	0.000308	0.694725	0.696415	0.1	
17	0.012742	0.000281	0.785634	0.793773	0.1	
18	0.012533	0.000293	0.785634	0.801630	1	
19	0.010670	0.000372	0.690236	0.689398	1	
20	0.012435	0.000270	0.785634	0.801630	1	
21	0.012769	0.000264	0.775533	0.794617	1	

22	0.011722	0.000321	0.694725	0.696133	1	
23	0.014138	0.000291	0.775533	0.794617	1	
24	0.011371	0.000252	0.784512	0.803032	10	
25	0.012640	0.000331	0.690236	0.689398	10	
26	0.011121	0.000267	0.784512	0.803032	10	
27	0.013278	0.000293	0.775533	0.792931	10	
28	0.014154	0.000320	0.694725	0.696695	10	
29	0.013745	0.000306	0.775533	0.792931	10	

30 rows × 24 columns

```
# output
tsLR_Y = LR.predict( tsle )

passenger_id = test.PassengerId

tsLR = pd.DataFrame( { 'PassengerId': passenger_id , 'Survived': tsLR_Y } )
```

```
tsLR.shape

tsLR.head()

tsLR.to_csv( 'LogisticRegression.csv' , index = False )
```

One Hot Encoding 解決 Categorial Features

```
# get numerical features
numerical = pd.DataFrame()

# Fill missing values of Age with the average of Age (mean)
numerical[ 'Age' ] = whole.Age.fillna( whole.Age.mean() )

# Fill missing values of Fare with the average of Fare (mean)
numerical[ 'Fare' ] = whole.Fare.fillna( whole.Fare.mean() )

# Fill missing values of Parch with the average of Parch (mean)
numerical[ 'Parch' ] = whole.Parch.fillna( whole.Parch.mean() )

# Fill missing values of SibSp with the average of SibSp (mean)
numerical[ 'SibSp' ] = whole.SibSp.fillna( whole.SibSp.mean() )

numerical.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Age	Fare	Parch	SibSp
0	22.0	7.2500	0	1
1	38.0	71.2833	0	1
2	26.0	7.9250	0	0
3	35.0	53.1000	0	1
4	35.0	8.0500	0	0

```
# get categorial features
cabin = pd.DataFrame()

# replacing missing cabins with U (for Uknown)
cabin[ 'Cabin' ] = whole.Cabin.fillna( 'Unknown' )

# dummy encoding ...
cabin = pd.get_dummies( cabin['Cabin'] , prefix = 'Cabin' )
```

```
cabin.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Cabin_A10	Cabin_A11	Cabin_A14	Cabin_A16	Cabin_A18	Cabin_A19	Cabin
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0

5 rows × 187 columns

```
# get categorial features
embarked = pd.DataFrame()

# replacing missing cabins with U (for Uknown)
embarked[ 'Embarked' ] = whole.Embarked.fillna( 'Unknown' )

# dummy encoding ...
embarked = pd.get_dummies( embarked['Embarked'] , prefix = 'Embarked' )

embarked.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Embarked_C	Embarked_Q	Embarked_S	Embarked_Unknown
0	0	0	1	0
1	1	0	0	0

2	0	0	1	0
3	0	0	1	0
4	0	0	1	0

```
# get categorial features
pclass = pd.DataFrame()

# replacing missing cabins with U (for Uknown)
pclass[ 'Pclass' ] = whole.Pclass.fillna( 'Unknown' )

# dummy encoding ...
pclass = pd.get_dummies( pclass['Pclass'] , prefix = 'Pclass' )

pclass.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Pclass_1	Pclass_2	Pclass_3
0	0	0	1
1	1	0	0
2	0	0	1
3	1	0	0
4	0	0	1

```
# get categorial features
sex = pd.DataFrame()

# replacing missing cabins with U (for Uknown)
sex[ 'Sex' ] = whole.Sex.fillna( 'Unknown' )

# dummy encoding ...
sex = pd.get_dummies( sex['Sex'] , prefix = 'Sex' )

sex.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
```

```
}

.dataframe thead th {
    text-align: right;
}
```

	Sex_female	Sex_male
0	0	1
1	1	0
2	1	0
3	1	0
4	0	1

```
# combine all features
ready = pd.concat([numerical, cabin, pclass, sex, whole.Survived], axis = 1)

ready.head()
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	Age	Fare	Parch	SibSp	Cabin_A10	Cabin_A11	Cabin_A14	Cabin_
0	22.0	7.2500	0	1	0	0	0	0
1	38.0	71.2833	0	1	0	0	0	0
2	26.0	7.9250	0	0	0	0	0	0
3	35.0	53.1000	0	1	0	0	0	0
4	35.0	8.0500	0	0	0	0	0	0

5 rows × 197 columns

```
# separate train data and test data
trohe = ready[ready['Survived'] != -1]
```

```
trohe_X = trohe.drop(['Survived'], axis = 1)

trohe_Y = trohe.Survived

tsohe = ready[ready['Survived'] == -1].drop(['Survived'], axis = 1)
```

資料正規化

```
# standardize
sc = StandardScaler()

sc.fit(trohe_X)

trstd_X = sc.transform(trohe_X)

tsstd = sc.transform(tsohe)
```

主成份分析

```
# PCA
pca = PCA(n_components = 0.95)

trpca_X = pca.fit_transform(trstd_X)

tspca = pca.transform(tsstd)

pca.explained_variance_ratio_
```

```
array([ 0.02328329,  0.01583371,  0.01158622,  0.00983334,  0.0083634 ,
        0.00803865,  0.00721866,  0.00684796,  0.00639709,  0.00639266,
        0.00639178,  0.00639094,  0.00639093,  0.00639014,  0.00638747,
        0.00638459,  0.00638376,  0.00638376,  0.00638376,  0.00638376,
        0.00638376,  0.00638376,  0.00638376,  0.00638376,  0.00638376,
        0.00638376,  0.00638376,  0.00638376,  0.00638376,  0.00638376,
        0.00638376,  0.00638376,  0.00638376,  0.00638376,  0.00638376,
        0.00638376,  0.00638376,  0.00638376,  0.00638376,  0.00638376,
        0.00638376,  0.00638376,  0.00638376,  0.00638376,  0.00638376,
        0.00638376,  0.00638236,  0.00638215,  0.00638139,  0.00638067,
        0.00638033,  0.00637967,  0.00637867,  0.00637782,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658,  0.00637658,  0.00637658,
        0.00637658,  0.00637658,  0.00637658])
```

```
cov_mat = np.cov(trstd_X.T)

eigen_vals, eigen_vecs = np.linalg.eig(cov_mat)

print('\nEigenvalues \n%s' % eigen_vals)
```

```
Eigenvalues
[  3.65958401e+00   2.48868588e+00   1.82107991e+00   1.54556867e+00
   1.20120770e-01   2.05148278e-01   3.76339613e-01   4.57377897e-01
   1.31452874e+00   6.80901036e-01   1.26348610e+00   7.93501504e-01
   8.70258505e-01   1.13460346e+00   1.07633704e+00  -1.53652359e-15
   2.87021781e-16  -1.50181553e-16   1.00547180e+00   1.00477483e+00
   1.00463734e+00   1.00395931e+00   1.00437908e+00   1.00450295e+00
   1.00450570e+00   1.00350660e+00   1.00244331e+00   1.00315648e+00
   1.00312270e+00   1.00300417e+00   1.00273391e+00   1.00257593e+00
   1.00283670e+00   1.00289087e+00   1.00224845e+00   1.00224845e+00
   1.00337584e+00   1.00337584e+00   1.00224845e+00   1.00337584e+00
   1.00337584e+00   1.00337584e+00   1.00337584e+00   1.00337584e+00
   1.00337584e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00337584e+00   1.00337584e+00   1.00337584e+00
   1.00337584e+00   1.00337584e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00337584e+00   1.00337584e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00337584e+00
   1.00337584e+00   1.00337584e+00   1.00337584e+00   1.00337584e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00337584e+00
   1.00337584e+00   1.00337584e+00   1.00337584e+00   1.00337584e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00337584e+00   1.00337584e+00   1.00337584e+00   1.00337584e+00
   1.00337584e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   1.00224845e+00   1.00224845e+00   1.00224845e+00
   1.00224845e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00
   0.00000000e+00   0.00000000e+00   0.00000000e+00   0.00000000e+00]
```

線性判別分析

```
# LDA
# lda = LinearDiscriminantAnalysis(n_components=10)

# trlda_X = lda.fit_transform(trstd_X, trohe_Y)

# tslda = lda.transform(tsstd)

# lda.explained_variance_ratio_
```

Model: Support Vector Machine

```
# SVM
param_grid = [
   {'C': [0.1, 1, 10], 'gamma': ['auto', (1 / 10), 1, 10], 'kernel': ['rbf'],
'class_weight': [{0:1, 1:1}, 'balanced']},
   {'C': [ 0.1, 1, 10], 'kernel': ['poly'], 'class_weight': [{0:1, 1:1},
'balanced']},
   {'C': [ 0.1, 1, 10], 'kernel': ['sigmoid'], 'class_weight': [{0:1, 1:1},
'balanced']},
 ]

SVM = GridSearchCV(SVC(), param_grid, verbose = 0, n_jobs = -1, scoring = 'accuracy',
cv = StratifiedKFold(5), return_train_score = True)

SVM.fit(trpca_X, trohe_Y)
```

```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
       error_score='raise',
       estimator=SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid=[{'C': [0.1, 1, 10], 'gamma': ['auto', 0.1, 1, 10], 'kernel':
['rbf'], 'class_weight': [{0: 1, 1: 1}, 'balanced']}, {'C': [0.1, 1, 10],
'kernel': ['poly'], 'class_weight': [{0: 1, 1: 1}, 'balanced']}, {'C': [0.1, 1,
10], 'kernel': ['sigmoid'], 'class_weight': [{0: 1, 1: 1}, 'balanced']}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='accuracy', verbose=0)
```

```
print('Best Param:', SVM.best_params_)

print('Best Model Score:', SVM.best_estimator_.score(trpca_X, trohe_Y))

display(pd.DataFrame(SVM.cv_results_))
```

```
Best Param: {'C': 10, 'class_weight': {0: 1, 1: 1}, 'gamma': 1, 'kernel': 'rbf'}
Best Model Score: 0.897867564534
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
```

```
    text-align: right;
}
```

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_C
0	0.067622	0.015151	0.692480	0.693609	0.1
1	0.063431	0.013673	0.774411	0.780875	0.1
2	0.067269	0.015051	0.773288	0.809177	0.1
3	0.070405	0.016087	0.618406	0.633839	0.1
4	0.077512	0.017281	0.760943	0.760663	0.1
5	0.067628	0.014051	0.766554	0.767680	0.1
6	0.072611	0.015498	0.710438	0.715216	0.1
7	0.078002	0.017635	0.628507	0.640301	0.1
8	0.057563	0.012334	0.760943	0.829128	1
	0.056031	0.011530	0.787879	0.865324	1

9						
10	0.060329	0.012620	0.796857	0.889457	1	
11	0.067974	0.014534	0.765432	0.911061	1	
12	0.063638	0.012563	0.762065	0.826042	1	
13	0.060908	0.011979	0.796857	0.857192	1	
14	0.065954	0.013123	0.796857	0.886368	1	
15	0.071702	0.014758	0.756453	0.907973	1	
16	0.053650	0.010966	0.771044	0.858872	10	
17	0.059536	0.011157	0.793490	0.879637	10	
18	0.068366	0.011943	0.800224	0.900116	10	
19	0.097121	0.013843	0.746352	0.926211	10	

20	0.059402	0.011355	0.763187	0.840355	10	
21	0.065315	0.011570	0.795735	0.881319	10	
22	0.070607	0.012455	0.792368	0.900115	10	
23	0.099861	0.014048	0.737374	0.920880	10	
24	0.058854	0.012424	0.659933	0.748606	0.1	
25	0.073353	0.015219	0.665544	0.751972	0.1	
26	0.054444	0.011420	0.661055	0.762912	1	
27	0.066622	0.013801	0.662177	0.761790	1	
28	0.053843	0.011098	0.670034	0.767681	10	
29	0.065407	0.013328	0.670034	0.767120	10	

30	0.065600	0.015402	0.722783	0.728680	0.1	
31	0.079326	0.018259	0.769921	0.784242	0.1	
32	0.052353	0.010807	0.773288	0.799936	1	
33	0.057228	0.011509	0.782267	0.807237	1	
34	0.036517	0.006478	0.711560	0.746917	10	
35	0.035237	0.006262	0.700337	0.737373	10	

36 rows × 24 columns

```
# output
tssvm_Y = SVM.predict( tspca )

passenger_id = test.PassengerId

tssvm = pd.DataFrame( { 'PassengerId': passenger_id , 'Survived': tssvm_Y } )

tssvm.shape

tssvm.head()

tssvm.to_csv( 'SVM.csv' , index = False )
```

Model: K Nearist Neighbors

```
# KNN
param_grid = [{'n_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance'],
'algorithm': ['ball_tree', 'kd_tree', 'brute'], 'leaf_size': [10, 30, 50]},]
```



```
KNN = GridSearchCV(KNeighborsClassifier(), param_grid, verbose = 0, n_jobs = -1,
scoring = 'accuracy', cv = StratifiedKFold(5), return_train_score = True)

KNN.fit(trpca_X, trohe_Y)
```

```
GridSearchCV(cv=StratifiedKFold(n_splits=5, random_state=None, shuffle=False),
       error_score='raise',
       estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform'),
       fit_params=None, iid=True, n_jobs=-1,
       param_grid=[{'n_neighbors': [3, 5, 7], 'weights': ['uniform', 'distance'],
'algorithm': ['ball_tree', 'kd_tree', 'brute'], 'leaf_size': [10, 30, 50]}],
       pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
       scoring='accuracy', verbose=0)
```

```
print('Best Param:', KNN.best_params_)

print('Best Model Score:', KNN.best_estimator_.score(trpca_X, trohe_Y))

display(pd.DataFrame(KNN.cv_results_))
```

```
Best Param: {'algorithm': 'kd_tree', 'leaf_size': 30, 'n_neighbors': 5, 'weights':
'uniform'}
Best Model Score: 0.859708193042
```

```
.dataframe tbody tr th {
    vertical-align: top;
}

.dataframe thead th {
    text-align: right;
}
```

	mean_fit_time	mean_score_time	mean_test_score	mean_train_score	param_algorithı
0	0.005140	0.022062	0.801347	0.879357	ball_tree
1	0.005068	0.022165	0.790123	0.991021	ball_tree
2	0.004310	0.021905	0.803591	0.856072	ball_tree

3	0.004046	0.021841	0.789001	0.991021	ball_tree
4	0.004148	0.022011	0.803591	0.830816	ball_tree
5	0.004125	0.021968	0.794613	0.991021	ball_tree
6	0.003257	0.021114	0.802469	0.879638	ball_tree
7	0.003366	0.021569	0.791246	0.991021	ball_tree
8	0.003495	0.021299	0.803591	0.856072	ball_tree
9	0.003189	0.021142	0.787879	0.991021	ball_tree
10	0.003156	0.021588	0.803591	0.830536	ball_tree
11	0.003484	0.021501	0.794613	0.991021	ball_tree
12	0.002832	0.021116	0.801347	0.879358	ball_tree
	0.002802	0.021013	0.790123	0.991021	ball_tree

13					
14	0.003092	0.021117	0.803591	0.856072	ball_tree
15	0.002855	0.021035	0.789001	0.991021	ball_tree
16	0.002710	0.020859	0.803591	0.830536	ball_tree
17	0.002738	0.020687	0.794613	0.991021	ball_tree
18	0.003544	0.033177	0.801347	0.879919	kd_tree
19	0.003469	0.033502	0.790123	0.991021	kd_tree
20	0.003453	0.035637	0.803591	0.856072	kd_tree
21	0.003454	0.035690	0.789001	0.991021	kd_tree
22	0.003520	0.037450	0.803591	0.830536	kd_tree
23	0.003633	0.037512	0.794613	0.991021	kd_tree

24	0.002800	0.021934	0.802469	0.879638	kd_tree
25	0.002870	0.022129	0.791246	0.991021	kd_tree
26	0.002859	0.023260	0.804714	0.856072	kd_tree
27	0.002825	0.023594	0.790123	0.991021	kd_tree
28	0.002976	0.024194	0.803591	0.831098	kd_tree
29	0.003021	0.024405	0.794613	0.991021	kd_tree
30	0.002528	0.021824	0.801347	0.879358	kd_tree
31	0.002572	0.021901	0.790123	0.991021	kd_tree
32	0.002501	0.022939	0.803591	0.856072	kd_tree
33	0.002556	0.022805	0.789001	0.991021	kd_tree

34	0.002627	0.023019	0.803591	0.830536	kd_tree
35	0.002538	0.071179	0.794613	0.991021	kd_tree
36	0.014250	0.018346	0.801347	0.878236	brute
37	0.016491	0.020921	0.789001	0.993547	brute
38	0.001151	0.030036	0.804714	0.856353	brute
39	0.005238	0.018812	0.787879	0.993547	brute
40	0.001108	0.042256	0.803591	0.831098	brute
41	0.001120	0.065218	0.793490	0.993547	brute
42	0.007890	0.013164	0.801347	0.878236	brute
43	0.006656	0.021156	0.789001	0.993547	brute

44	0.001133	0.021750	0.804714	0.856353	brute
45	0.005301	0.020289	0.787879	0.993547	brute
46	0.004540	0.061174	0.803591	0.831098	brute
47	0.005068	0.050584	0.793490	0.993547	brute
48	0.008016	0.034973	0.801347	0.878236	brute
49	0.001116	0.030357	0.789001	0.993547	brute
50	0.005894	0.085542	0.804714	0.856353	brute
51	0.007367	0.056985	0.787879	0.993547	brute
52	0.006253	0.042054	0.803591	0.831098	brute
53	0.001106	0.024750	0.793490	0.993547	brute

54 rows × 24 columns

```
# output
tsknn_Y = KNN.predict( tspca )

passenger_id = test.PassengerId

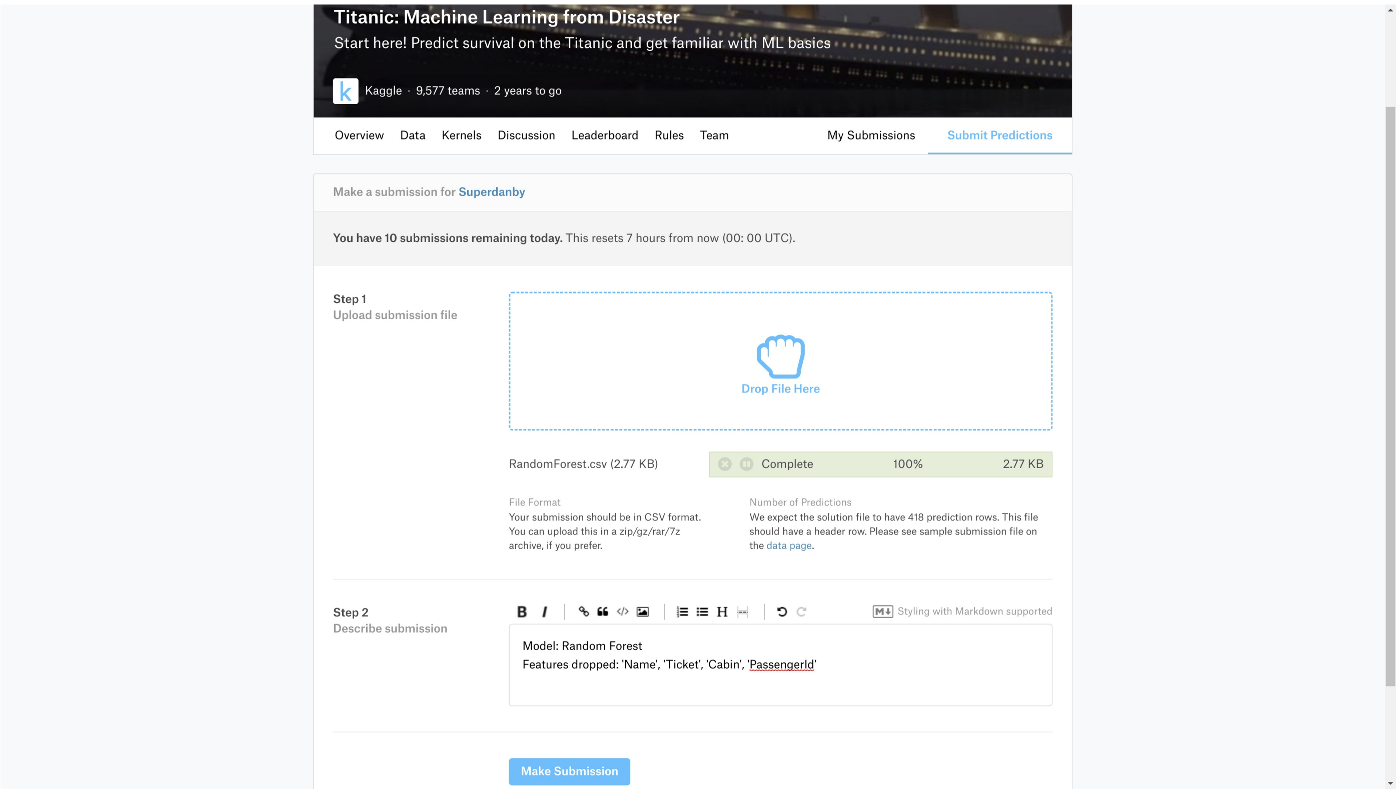
tsknn = pd.DataFrame( { 'PassengerId': passenger_id , 'Survived': tsknn_Y } )

tsknn.shape

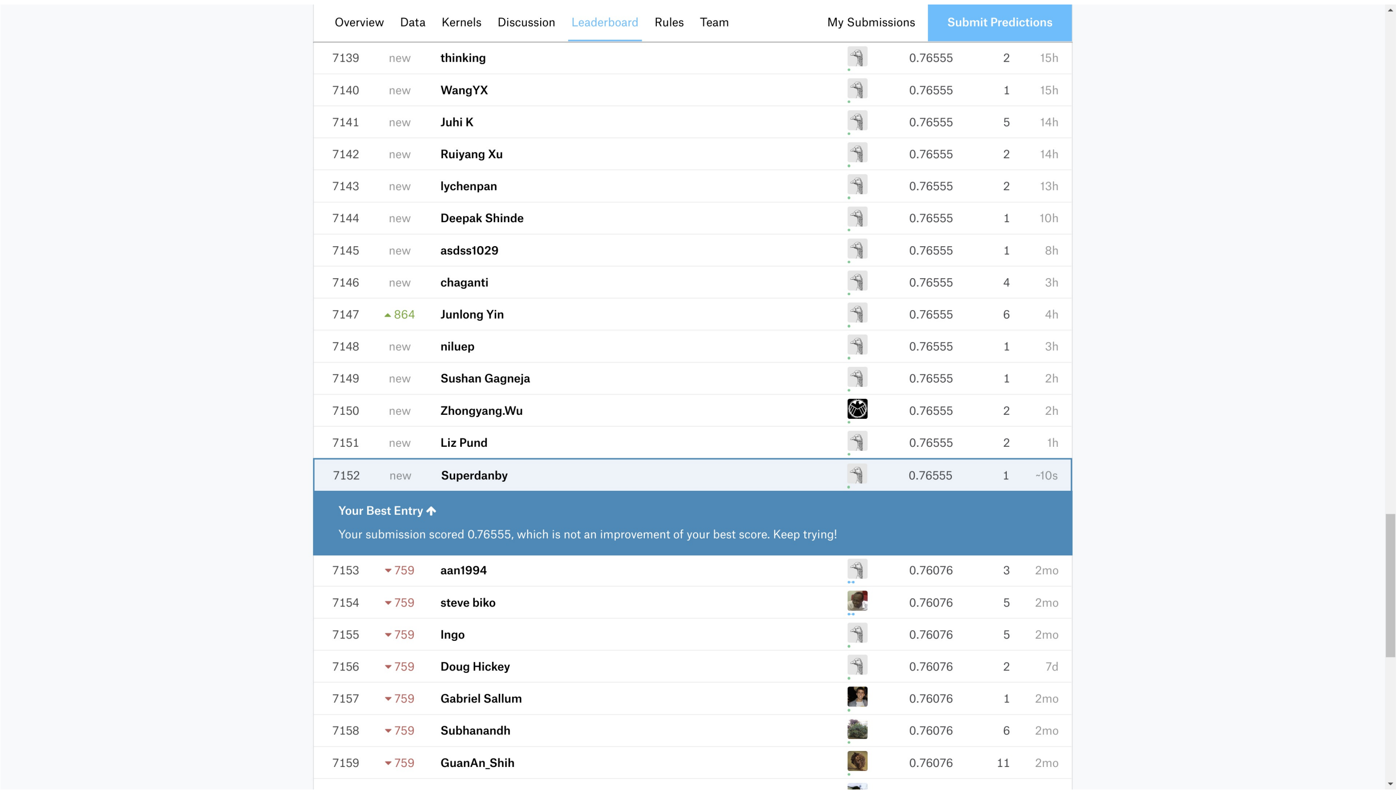
tsknn.head()

tsknn.to_csv( 'kNearistNeighbor.csv' , index = False )
```

選擇一個結果上傳



排名



Overview Data Kernels Discussion Leaderboard Rules Team My Submissions Submit Predictions

7139	new	thinking	0.76555	2	15h
7140	new	WangYX	0.76555	1	15h
7141	new	Juhi K	0.76555	5	14h
7142	new	Ruiyang Xu	0.76555	2	14h
7143	new	lychenpan	0.76555	2	13h
7144	new	Deepak Shinde	0.76555	1	10h
7145	new	asdss1029	0.76555	1	8h
7146	new	chaganti	0.76555	4	3h
7147	▲864	Junlong Yin	0.76555	6	4h
7148	new	niluep	0.76555	1	3h
7149	new	Sushan Gagneja	0.76555	1	2h
7150	new	Zhongyang.Wu	0.76555	2	2h
7151	new	Liz Pund	0.76555	2	1h
7152	new	Superdanby	0.76555	1	~10s

Your Best Entry ↑
Your submission scored 0.76555, which is not an improvement of your best score. Keep trying!

7153	▼759	aan1994	0.76076	3	2mo
7154	▼759	steve biko	0.76076	5	2mo
7155	▼759	Ingo	0.76076	5	2mo
7156	▼759	Doug Hickey	0.76076	2	7d
7157	▼759	Gabriel Sallum	0.76076	1	2mo
7158	▼759	Subhanandh	0.76076	6	2mo
7159	▼759	GuanAn_Shih	0.76076	11	2mo

目標

拿到各項比賽第一名