Navigation

I want to write a few words about the goals and objectives of this notebook..

of Course, the main goal is to achieve the maximum result in this competition, but not only.. I started my DS / ML career not so long ago and a lot of questions arise in the course of my work. In this notebook, I will try to give answers to the questions of beginners that I think they can ask, or that I myself would have asked at the very beginning.. So.. the result of this work will be submission. but let's understand in order what is needed for this I would conditionally split the whole process into 6 parts:

1. Carefully read the terms of the contest!!! It is important to understand what needs to be done, and most importantly how it will be evaluated!

2. Upload the submitted data: carefully study it, check for completeness and validity. You should understand the meaning of this data. If it is not clear, then look for an opportunity to solve this problem-search on the Internet, ask the organizers, ask your colleagues on kaggle. The point is that it is not possible to build a good model if you do not understand the logic of the process or the purpose of certain data.

3. Processing and preparing data: You need to study the data very carefully, select the fields that you intend to use in the model, and get rid of those that you don't think you need (you can always play this back). Put the data itself in order-delete or fill in the missing values, bring everything to the same view. All categorical data must be converted to a numeric form - for example, we have A, B, C-it must be 0,1,2 or 1,2,3, or male/female turns into 1/2.. and so on.

4. Identifying features: then the creative process begins:) next, we will consider how you can get a valuable feature from information that would seem completely useless at first glance.

5. there Is a set of features for the first run, you can start..

6. Debugging and calibrating models - another one creative process:)) although it is more formalized and automated than the selection of features:)

I also want to say right away that you will not find in this work a mega super cool model that gives 100500% accuracy. Maximum, I managed to get 0.80861, and it was just once. And so it gives an average of 0.795-0.805 if you play around with the settings.

Хочу написать несоклько слов о целях и задачах этого ноутбука..

Конечно, основная цель добиться максимального результате в этом соревновании, но не только.. я сам не так давно начал карьеру DS/ML и по ходу работы возникло много вопросов. В этом ноутбуке я попытаюсь дать ответы на вопросы новичков, которые, как мне кажется они могут задать, ну или которые я сам бы задал в самом начале.. Итак.. итогом проведенной работы будет submission. но давайте по-порядку разбираться , что для этого нужно я бы условно разбил весь процесс на 6 частей:

1. Внимательно прочитать условия конкурса!!! Важно понять, что нужно сделать, и самое главное как это будет оцениваться!
2. Загрузить представленные данные: Внимательно изучить их, проверить на полноту, валидность. Вам должен быть понятен смысл этих данных. Если не понятен, решайте эту проблему - ищите в интеренете, спрашивайте организаторов, спрашивайте у коллег на kaggle. Смысл в том, что не возможно построить хорошую модель, если вы не понимаете логику процесса или назначение тех или иных данных.
3. Обработка и подготовка данных: Нужно уже очень внимательно изучить данные, выбрать те поля, которые вы предполагаете использовать в модели, и избавиться от тех, которые, как вам кажется, не нужны (всегда можно будет это отыграть назад). Приведите сами данные в порядок - удалите или заполните пропущенные значения, приведите все к одному виду. Все категориальные данные

нужно привести к числовому виду - например имеем A, B, C - должно быть 0,1,2 или 1,2,3 или male/female превращается в 1/2.. ну и так далее.

4. Выявление фичей: дальше начинется творчсекий процесс:) дальше будем рассматривать как из, казалось бы совершенно бесполезной на первый взгляд информации, можно получить ценную фичу
5. Есть набор фичей для первого прогона, можно стартовать..
6. Отладка и колибровка моделей - еще один творческий процесс:)) хотя и в большей степени формализован и автоматизирован, чем выбор фичей:)

Так же сразу хочу сказать, что вы не найдете в этой работе мега супер крутой модели, которая выдает 100500 % accuracy. Максимум, мне удалось получить 0.80861, и то 1 раз. А так выдает в среднем 79.5 - 80.5 если поиграться настройками.

Libraries import

In [1]:
```
import numpy as np
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import colors
```

In [2]:
```
print ('numpy   ver: ', np.__version__)
print ('pandas  ver: ', pd.__version__)
print ('seaborn ver: ', sns.__version__)
```

```
numpy   ver:  1.19.2
pandas  ver:  1.1.3
seaborn ver:  0.11.0
```

In [3]:
```
from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV, cross_validate
from sklearn import metrics
from sklearn.metrics import f1_score, confusion_matrix
```

In [4]:
```
from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier, VotingClassifier, BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier, GradientBoostingClassifier, StackingClassifier
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.svm import SVC, LinearSVC, NuSVC
from sklearn.neighbors import KNeighborsClassifier, RadiusNeighborsClassifier
from sklearn.naive_bayes import GaussianNB, CategoricalNB, ComplementNB
from sklearn.tree import DecisionTreeClassifier, ExtraTreeClassifier
from sklearn.neural_network import MLPClassifier
```

```
from sklearn.linear_model import PassiveAggressiveClassifier, LogisticRegressionCV, LarsCV, LassoCV, LassoLarsCV
from sklearn.linear_model import LogisticRegression, Perceptron, SGDClassifier, RidgeClassifierCV
from sklearn.linear_model import RidgeClassifier, ElasticNetCV, OrthogonalMatchingPursuit
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis
from sklearn.semi_supervised import LabelPropagation

from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn import preprocessing
from sklearn.pipeline import make_pipeline
from sklearn.base import is_classifier, is_regressor
```

In [5]:

```
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
```

In [6]:

```
import warnings
warnings.filterwarnings('ignore')
warnings.filterwarnings('ignore', category=DeprecationWarning)
```

Data loading

In [7]:

```
# Loading data files into pandas DataFrames
# train = pd.read_csv('/kaggle/input/titanic/train.csv')
# test = pd.read_csv('/kaggle/input/titanic/test.csv')
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

In [8]:

```
# lets merge dataframes before cleaning data
full = pd.merge(train, test, how = 'outer')
```

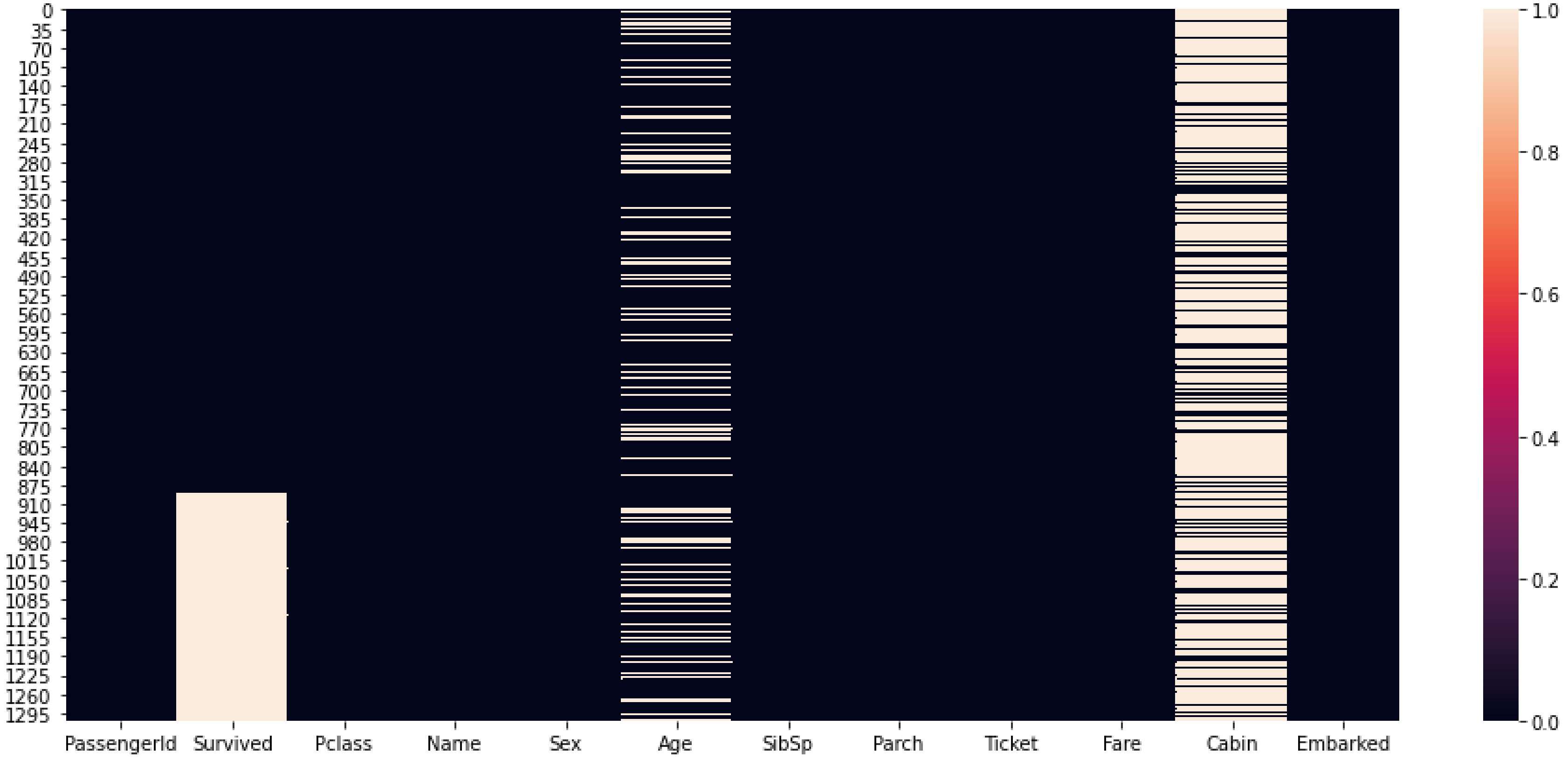
Vizualization

In [9]:

```
# lets see on our data
plt.figure(figsize=(16,7))
sns.heatmap(full.isnull())
```

Out[9]:

```
<AxesSubplot:>
```



In [10]:
```
full.isnull().sum()
```

Out[10]:
```
PassengerId       0
Survived        418
Pclass            0
Name              0
Sex               0
Age             263
SibSp             0
Parch             0
Ticket            0
Fare              1
Cabin          1014
Embarked          2
dtype: int64
```

Age

In [11]:
```
# we can use median to fill missed data on 'Age'
full.groupby(['Pclass', 'Sex'])['Age'].median()
```

Out[11]:
```
Pclass  Sex
1       female    36.0
        male      42.0
2       female    28.0
        male      29.5
3       female    22.0
        male      25.0
Name: Age, dtype: float64
```

In [12]:
```
def set_null_age(cols):
    Age,Pclass,Sex = cols
    if pd.isnull(Age):
        if Pclass == 1:
            if Sex == 'female':
                return 36
            else:
                return 42
        elif Pclass == 2:
            if Sex == 'female':
                return 28
            else:
                return 29.5
        elif Pclass == 3:
            if Sex == 'female':
                return 22
            else:
                return 25
    else:
        return Age
```

In [13]:
```
full['Age']=full[['Age', 'Pclass', 'Sex']].apply(set_null_age, axis = 1)
```

In [14]:
```
#lets round all ages
full.Age = full.Age.apply('ceil').astype(int)
```

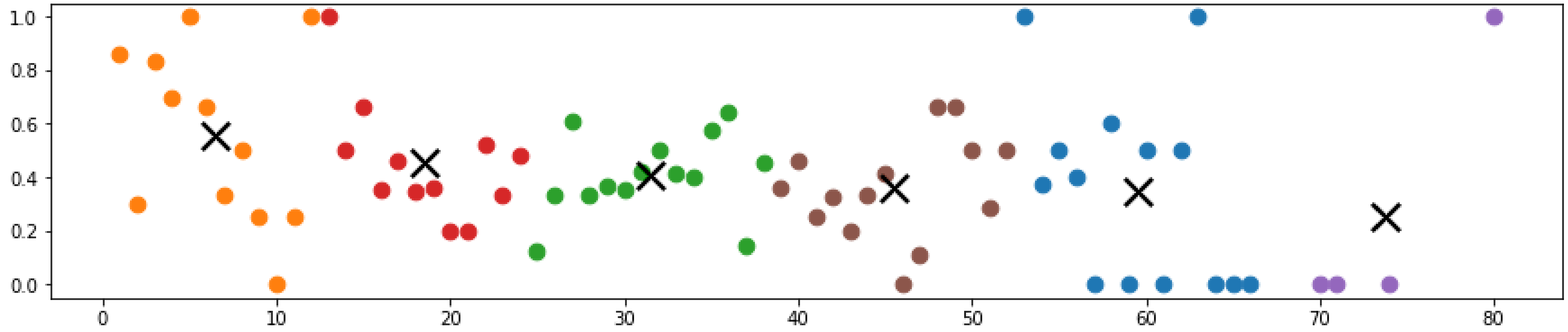
In [15]:
```
# im goint ot use ML to split ages for some clusters:)

N = 6
age = full.pivot_table(values = 'Survived', index = 'Age').sort_values('Age').reset_index()
X = age[['Age', 'Survived']]
```

```
clust = KMeans(n_clusters=N).fit(X)
c = clust.cluster_centers_

# we can take colors fro mathplotlib (10 colors - max 10 clusters)
clrs = list(colors.TABLEAU_COLORS.keys())

fig = plt.figure(figsize=(15, 3))
for x, y in zip(age.Age, age.Survived):
    cl = clust.predict(np.array([x,y]).reshape(1, -1))
    plt.scatter(x, y, s=75, c = clrs[cl[0]])
    for i in range(len(c)):
        plt.scatter(c[i][0], c[i][1], s=200, marker="x", c="black")
plt.show()
```



In [16]:

```
# Another algorithm for clustering
# N = 12
# age = full.pivot_table(values = 'Survived', index = 'Age').sort_values('Age').reset_index()
# X = age[['Age', 'Survived']]

# clust = AgglomerativeClustering(n_clusters=N).fit(X)

# lab =  clust.labels_
# colors = ['g', 'y', 'm', 'r', 'c', 'b', 'g', 'y', 'm', 'r', 'c', 'b']

# fig = plt.figure(figsize=(15, 3))
# for x, y, z in zip(age.Age, age.Survived, lab):
#     plt.scatter(x, y, s=75, c = colors[z])
# plt.show()
```

In [17]:

```
age['age_grouped'] = clust.labels_
```

```
age = age.drop('Survived', axis = 1)
```

In [18]:

```
full = pd.merge(full, age, on = 'Age', how = 'left')
```

In [19]:

```
full[pd.isnull(full['age_grouped'])]
```

Out[19]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	age_grouped
972	973	NaN	1	Straus, Mr. Isidor	male	67	1	0	PC 17483	221.7792	C55 C57	S	NaN
987	988	NaN	1	Cavendish, Mrs. Tyrell William (Julia Florence...	female	76	1	0	19877	78.8500	C46	S	NaN

In [20]:

```
# we fill set nearest clusters numbers for those rows
full.loc[972, 'age_grouped'] = full['age_grouped'][full.Age == 66].min()
full.loc[987, 'age_grouped'] = full['age_grouped'][full.Age == 74].min()
```

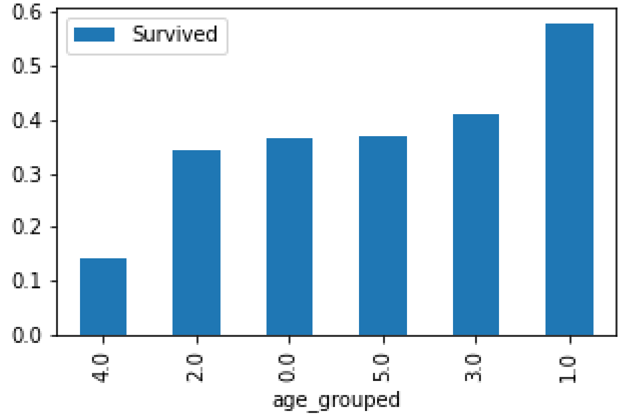
In [21]:

```
col = 'age_grouped'
target = 'Survived'
sort = target
print(full[:891].pivot_table(values = target, index = col).sort_values(sort))
full[:891].pivot_table(values = target, index = col).sort_values(sort).plot(kind = 'bar', figsize=(5,3))
```

```
             Survived
age_grouped
4.0          0.142857
2.0          0.342246
0.0          0.363636
5.0          0.367347
3.0          0.412000
1.0          0.579710
```

Out[21]:

```
<AxesSubplot:xlabel='age_grouped'>
```



Name

In [22]:
```
# we have some duplicates on 'Name'
full.Name.nunique()
```

Out[22]: 1307

In [23]:
```
full[full.duplicated('Name')]
```

Out[23]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	age_grouped
891	892	NaN	3	Kelly, Mr. James	male	35	0	0	330911	7.8292	NaN	Q	2.0
897	898	NaN	3	Connolly, Miss. Kate	female	30	0	0	330972	7.6292	NaN	Q	2.0

In [24]:
```
# its ok - they are different ppl
full[(full.Name == 'Kelly, Mr. James') | (full.Name == 'Connolly, Miss. Kate')].sort_values('Name')
```

Out[24]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	age_grouped
289	290	1.0	3	Connolly, Miss. Kate	female	22	0	0	370373	7.7500	NaN	Q	3.0
897	898	NaN	3	Connolly, Miss. Kate	female	30	0	0	330972	7.6292	NaN	Q	2.0
696	697	0.0	3	Kelly, Mr. James	male	44	0	0	363592	8.0500	NaN	S	5.0
891	892	NaN	3	Kelly, Mr. James	male	35	0	0	330911	7.8292	NaN	Q	2.0

In [25]:
```
# we dont really need names, but we can try to use Titles
```

In [26]:
```
full['Title'] = full['Name'].str.extract(' ([A-Za-z]+)\.', expand=False)
```

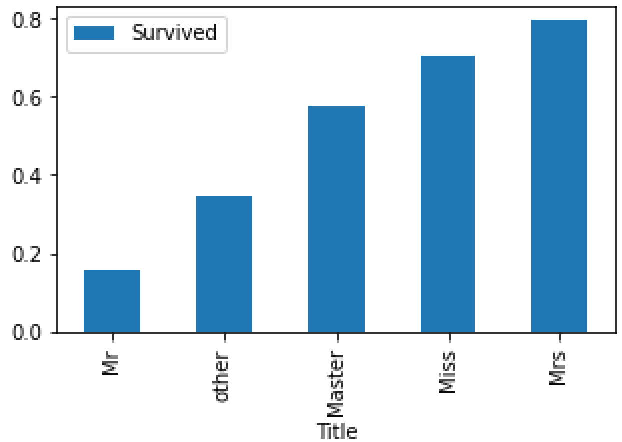
In [27]:
```
full['Title'] = full['Title'].replace(['Lady', 'Countess','Capt', 'Col',\
'Don', 'Dr', 'Major', 'Rev', 'Sir', 'Jonkheer', 'Dona'], 'other')

full['Title'] = full['Title'].replace(['Mlle', 'Ms'], 'Miss')
full['Title'] = full['Title'].replace('Mme', 'Mrs')
```

In [28]:
```
# seems like a good feature
col = 'Title'
target = 'Survived'
sort = target
print(full[:891].pivot_table(values = target, index = col).sort_values(sort))
full[:891].pivot_table(values = target, index = col).sort_values(sort).plot(kind = 'bar', figsize=(5,3))
```

```
        Survived
Title
Mr      0.156673
other   0.347826
Master  0.575000
Miss    0.702703
Mrs     0.793651
```

Out[28]: <AxesSubplot:xlabel='Title'>



In [29]:
```
full['Title'] = full['Title'].map({"Mr": 1, "other": 2, "Master": 3, "Miss": 4, "Mrs": 5})
```

Ticket

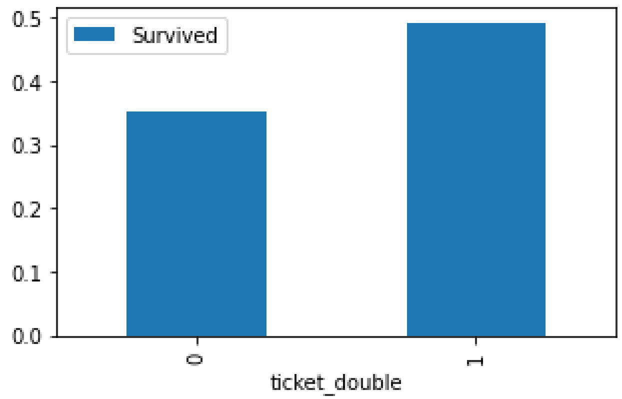
In [30]:
```
# some tickets have diplicates in number.. we can try to use it
full['ticket_double'] = 0
```

In [31]:
```
full['ticket_double'][full.duplicated('Ticket')] = 1
```

In [32]:
```
# almost 50% survuve rate against 36% basic.. it can be the feature
col = 'ticket_double'
target = 'Survived'
sort = col
print(full[:891].pivot_table(values = target, index = col).sort_values(sort))
full[:891].pivot_table(values = target, index = col).sort_values(sort).plot(kind = 'bar', figsize=(5,3))
```

```
               Survived
ticket_double
0              0.350954
1              0.490476
```

Out[32]:
```
<AxesSubplot:xlabel='ticket_double'>
```



Cabin

In [33]:
```
# we can try to extract cabin name and use it later as feature
full.Cabin.unique()
```

Out[33]:
```
array([nan, 'C85', 'C123', 'E46', 'G6', 'C103', 'D56', 'A6',
       'C23 C25 C27', 'B78', 'D33', 'B30', 'C52', 'B28', 'C83', 'F33',
       'F G73', 'E31', 'A5', 'D10 D12', 'D26', 'C110', 'B58 B60', 'E101',
       'F E69', 'D47', 'B86', 'F2', 'C2', 'E33', 'B19', 'A7', 'C49', 'F4',
       'A32', 'B4', 'B80', 'A31', 'D36', 'D15', 'C93', 'C78', 'D35',
```

```
       'C87', 'B77', 'E67', 'B94', 'C125', 'C99', 'C118', 'D7', 'A19',
       'B49', 'D', 'C22 C26', 'C106', 'C65', 'E36', 'C54',
       'B57 B59 B63 B66', 'C7', 'E34', 'C32', 'B18', 'C124', 'C91', 'E40',
       'T', 'C128', 'D37', 'B35', 'E50', 'C82', 'B96 B98', 'E10', 'E44',
       'A34', 'C104', 'C111', 'C92', 'E38', 'D21', 'E12', 'E63', 'A14',
       'B37', 'C30', 'D20', 'B79', 'E25', 'D46', 'B73', 'C95', 'B38',
       'B39', 'B22', 'C86', 'C70', 'A16', 'C101', 'C68', 'A10', 'E68',
       'B41', 'A20', 'D19', 'D50', 'D9', 'A23', 'B50', 'A26', 'D48',
       'E58', 'C126', 'B71', 'B51 B53 B55', 'D49', 'B5', 'B20', 'F G63',
       'C62 C64', 'E24', 'C90', 'C45', 'E8', 'B101', 'D45', 'C46', 'D30',
       'E121', 'D11', 'E77', 'F38', 'B3', 'D6', 'B82 B84', 'D17', 'A36',
       'B102', 'B69', 'E49', 'C47', 'D28', 'E17', 'A24', 'C50', 'B42',
       'C148', 'B45', 'B36', 'A21', 'D34', 'A9', 'C31', 'B61', 'C53',
       'D43', 'C130', 'C132', 'C55 C57', 'C116', 'F', 'A29', 'C6', 'C28',
       'C51', 'C97', 'D22', 'B10', 'E45', 'E52', 'A11', 'B11', 'C80',
       'C89', 'F E46', 'B26', 'F E57', 'A18', 'E60', 'E39 E41',
       'B52 B54 B56', 'C39', 'B24', 'D40', 'D38', 'C105'], dtype=object)
```

In [34]:
```
full['Cabin'] = full['Cabin'].str.extract('([A-Za-z]+)', expand = False)
```

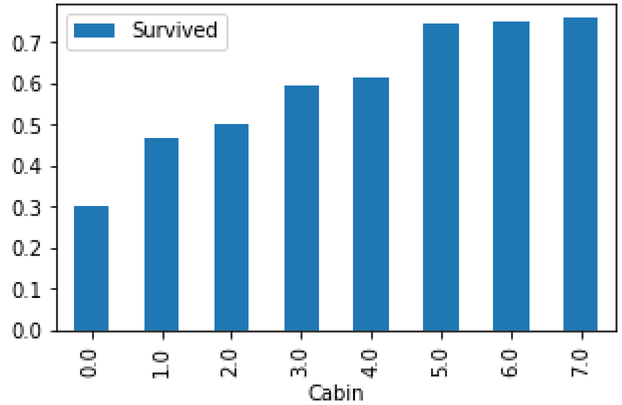
In [35]:
```
full['Cabin'] = full['Cabin'].map({'A':1, 'G':2, 'C':3, 'F':4, 'B':5, 'E':6, 'D':7, 'T':0})
```

In [36]:
```
full['Cabin'] = full['Cabin'].fillna(0)
```

In [37]:
```
# also seems good for feature
col = 'Cabin'
target = 'Survived'
sort = col
print(full[:891].pivot_table(values = target, index = col).sort_values(sort))
full[:891].pivot_table(values = target, index = col).sort_values(sort).plot(kind = 'bar', figsize=(5,3))
```

```
       Survived
Cabin
0.0    0.299419
1.0    0.466667
2.0    0.500000
3.0    0.593220
4.0    0.615385
5.0    0.744681
6.0    0.750000
7.0    0.757576
```

Out[37]:
```
<AxesSubplot:xlabel='Cabin'>
```



Fare

In [38]:
```
full[pd.isnull(full.Fare) | (full.Fare == 0)].Fare.count()
```

Out[38]:
```
17
```

In [39]:
```
full.groupby(['Pclass'])['Fare'].median().round(2)
```

Out[39]:
```
Pclass
1    60.00
2    15.05
3     8.05
Name: Fare, dtype: float64
```

In [40]:
```
def set_null_fare(cols):
    Fare, Pclass = cols
    if Fare == 0 or pd.isnull(Fare):
        if Pclass == 1:
            return 63.36
        elif Pclass == 2:
            return 15.75
        elif Pclass == 3:
            return 8.05
    else:
        return Fare
```

In [41]:
```
full['Fare']=full[['Fare', 'Pclass']].apply(set_null_fare, axis = 1).round(2)
```

In [42]:
```
# we can try different classes for 'Fare'
```

```python
def set_gr_fare(col):
    Fare = col
    if Fare < 8.05:
        return 1
    elif Fare < 15.75:
        return 2
    elif Fare < 32:
        return 3
    else:
        return 4
```

In [43]:

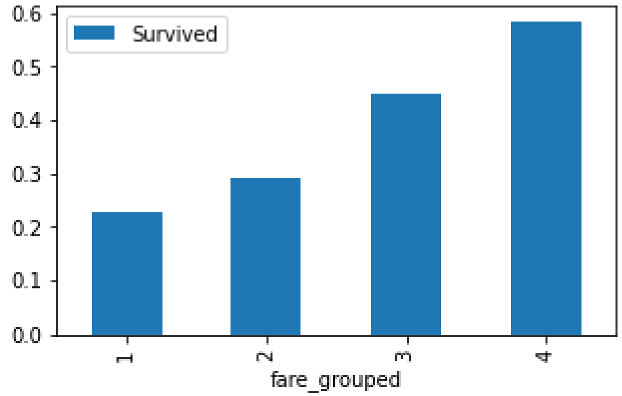
```python
full['fare_grouped']=full['Fare'].apply(set_gr_fare)
```

In [44]:

```python
# also seems good for feature
col = 'fare_grouped'
target = 'Survived'
sort = col
print(full[:891].pivot_table(values = target, index = col).sort_values(sort))
full[:891].pivot_table(values = target, index = col).sort_values(sort).plot(kind = 'bar', figsize=(5,3))
```

```
              Survived
fare_grouped
1             0.229075
2             0.292887
3             0.449761
4             0.583333
```

Out[44]: <AxesSubplot:xlabel='fare_grouped'>

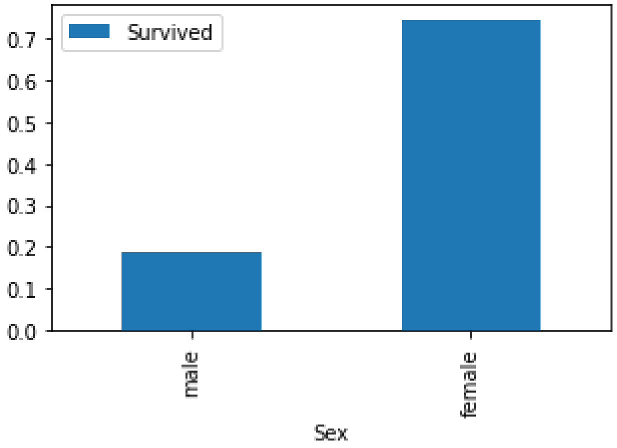


Sex

In [45]:
```
#very good feature
col = 'Sex'
target = 'Survived'
sort = target
print(full[:891].pivot_table(values = target, index = col).sort_values(sort))
full[:891].pivot_table(values = target, index = col).sort_values(sort).plot(kind = 'bar', figsize=(5,3))
```

```
        Survived
Sex
male    0.188908
female  0.742038
```

Out[45]: <AxesSubplot:xlabel='Sex'>



In [46]:
```
full['Sex'] = full['Sex'].map({'male':1, 'female':2})
```

Embarked

In [47]:
```
full[pd.isnull(full.Embarked)]
```

Out[47]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	age_grouped	Title	ticket_double	fare_grouped
61	62	1.0	1	Icard, Miss. Amelie	2	38	0	0	113572	80.0	5.0	NaN	2.0	4	0	4

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked	age_grouped	Title	ticket_double	fare_grouped
829	830	1.0	1	Stone, Mrs. George Nelson (Martha Evelyn)	2	62	0	0	113572	80.0	5.0	NaN	0.0	5	1	4

"..Mrs Stone boarded the Titanic in Southampton on 10 April 1912 and was travelling in first class with her maid Amelie Icard. She occupied cabin B-28.."

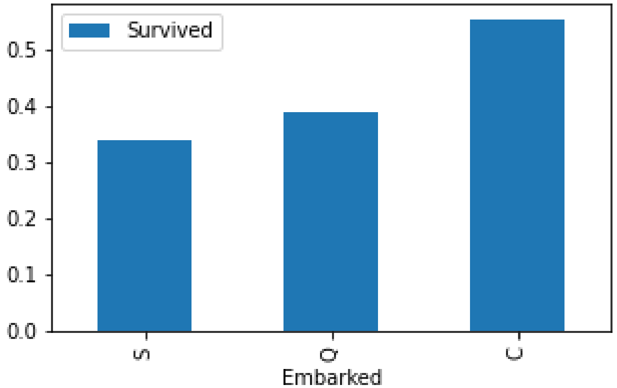
https://www.encyclopedia-titanica.org/titanic-survivor/martha-evelyn-stone.html

In [48]:
```
full.loc[(61,829), 'Embarked'] = 'S'
```

In [49]:
```
col = 'Embarked'
target = 'Survived'
sort = target
print(full[:891].pivot_table(values = target, index = col).sort_values(sort))
full[:891].pivot_table(values = target, index = col).sort_values(sort).plot(kind = 'bar', figsize=(5,3))
```

```
          Survived
Embarked
S         0.339009
Q         0.389610
C         0.553571
```

Out[49]: <AxesSubplot:xlabel='Embarked'>



In [50]:
```
# we will use 1, 2, 3 insteed of S, Q, C
full['Embarked'] = full['Embarked'].map({"S": 1, "Q": 2, "C": 3})
```

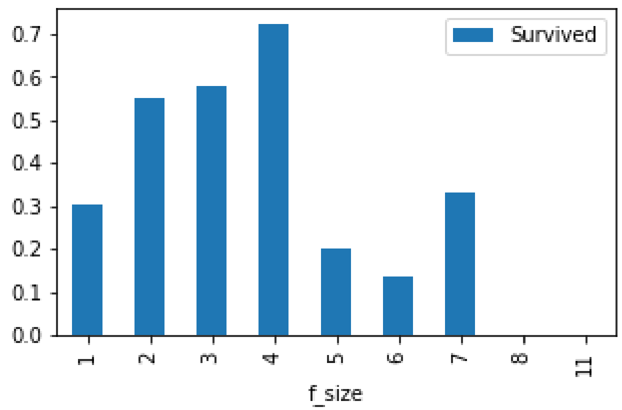
SibSp Parch

In [51]:
```
# we can count family size
full['f_size'] = full.SibSp + full.Parch + 1
full['age_class'] = full.age_grouped * full.Pclass
full['is_alone'] = full['f_size'].apply(lambda x: 0 if x == 1 else 1)
```

In [52]:
```
col = 'f_size'
target = 'Survived'
sort = col
print(full[:891].pivot_table(values = target, index = col).sort_values(sort))
full[:891].pivot_table(values = target, index = col).sort_values(sort).plot(kind = 'bar', figsize=(5,3))
```

```
        Survived
f_size
1       0.303538
2       0.552795
3       0.578431
4       0.724138
5       0.200000
6       0.136364
7       0.333333
8       0.000000
11      0.000000
```

Out[52]: <AxesSubplot:xlabel='f_size'>



In [53]:
```
# we dony anymore need those columns
```

```
full = full.drop(['PassengerId', 'Name', 'Ticket'], axis = 1)
```

In [ ]:

```

```

In [54]:

```
# We can use features as is.. but i want ot test dummies on all columns.. sometimes it gives better results.
# But we can try both ways later

bfull = full.drop(['Age', 'Fare'], axis = 1)
bfull = bfull.fillna(0).astype(int)

dummy_col=[ 'Pclass',
            'Sex',
            'Cabin',
            'Embarked',
            'age_grouped',
            'Title',
            'ticket_double',
            'fare_grouped',
            'f_size',
            'is_alone']
dummy = pd.get_dummies(bfull[dummy_col], columns=dummy_col)
bfull = pd.concat([dummy, bfull], axis = 1)
bfull.drop([
            'Pclass',
            'Sex',
            'SibSp',
            'Parch',
            'Cabin',
            'Embarked',
            'age_grouped',
            'Title',
            'ticket_double',
            'fare_grouped',
            'f_size',
            'age_class',
            'is_alone'], inplace = True, axis = 1)


# X_train = bfull[:891].drop('Survived', axis = 1).astype(int)
# y_train = bfull[:891].Survived.astype(int)
# X_test = bfull[891:].drop('Survived', axis = 1).astype(int)
```

In [55]:

```
# X_train = full[:891].drop('Survived', axis = 1).astype(int)
```

```
# y_train = full[:891].Survived.astype(int)
# X_test = full[891:].drop('Survived', axis = 1).astype(int)
```

Features choosing import

In [56]:
```
# list of all features we have.. we will reduce that number later
list(bfull.columns)
```

Out[56]:
```
['Pclass_1',
 'Pclass_2',
 'Pclass_3',
 'Sex_1',
 'Sex_2',
 'Cabin_0',
 'Cabin_1',
 'Cabin_2',
 'Cabin_3',
 'Cabin_4',
 'Cabin_5',
 'Cabin_6',
 'Cabin_7',
 'Embarked_1',
 'Embarked_2',
 'Embarked_3',
 'age_grouped_0',
 'age_grouped_1',
 'age_grouped_2',
 'age_grouped_3',
 'age_grouped_4',
 'age_grouped_5',
 'Title_1',
 'Title_2',
 'Title_3',
 'Title_4',
 'Title_5',
 'ticket_double_0',
 'ticket_double_1',
 'fare_grouped_1',
 'fare_grouped_2',
 'fare_grouped_3',
 'fare_grouped_4',
 'f_size_1',
 'f_size_2',
 'f_size_3',
 'f_size_4',
 'f_size_5',
```

```
 'f_size_6',
 'f_size_7',
 'f_size_8',
 'f_size_11',
 'is_alone_0',
 'is_alone_1',
 'Survived']
```

In [57]:

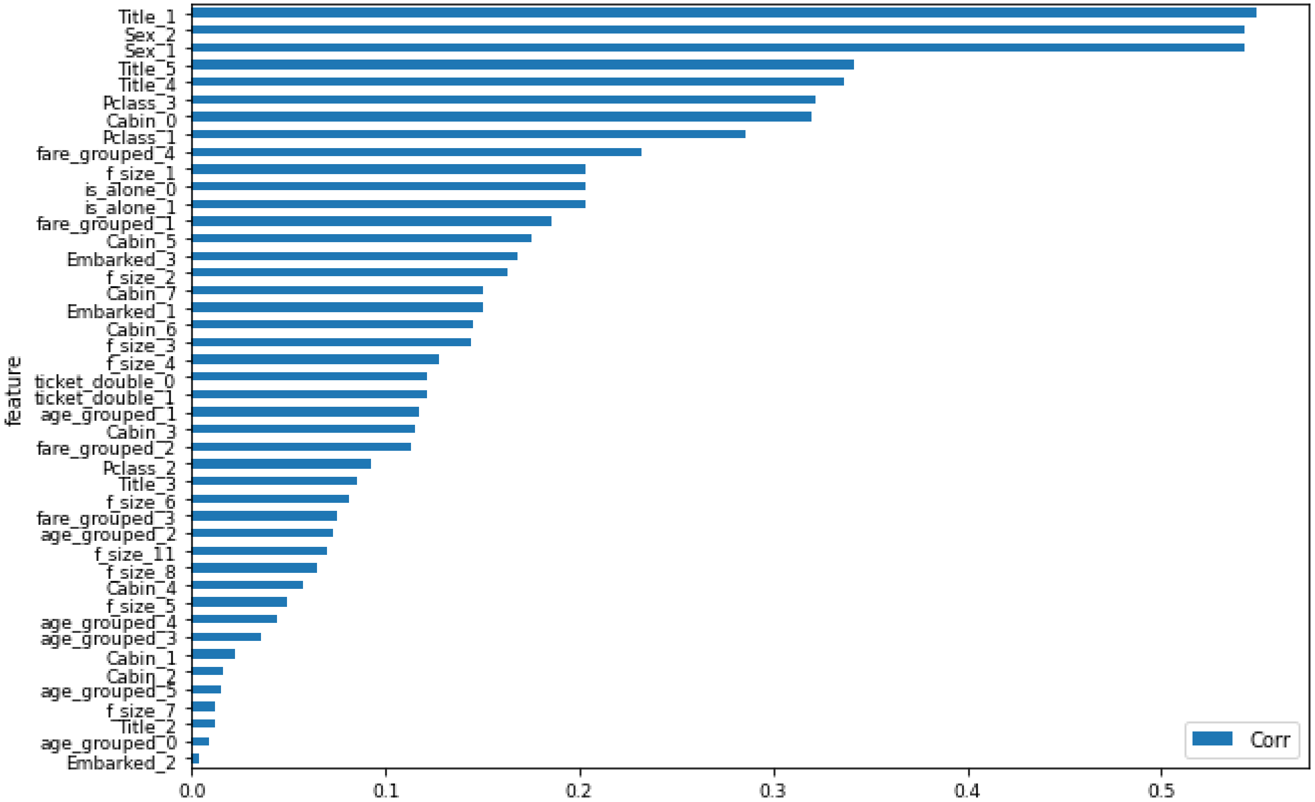
```
# let see correlation rates
df = bfull[:891].astype(int)
df_all_corr = df.corr().abs().unstack().sort_values(kind="quicksort", ascending=False).reset_index()
df_all_corr.rename(columns={"level_0": "F1", "level_1": "feature", 0: 'Corr'}, inplace=True)
final_corr = df_all_corr[df_all_corr['F1'] == 'Survived']
final_corr.Corr = final_corr.Corr.apply('abs').round(3)
final_corr = final_corr.sort_values('Corr', ascending=False)

col = 'feature'
target = 'Corr'
sort = target
print(final_corr[1:].pivot_table(values = target, index = col).sort_values(sort, ascending=False))
final_corr[1:].pivot_table(values = target, index = col).sort_values(sort).plot(kind = 'barh', figsize=(10,7), fontsize = 9)
```

```
                  Corr
feature
Title_1          0.549
Sex_2            0.543
Sex_1            0.543
Title_5          0.342
Title_4          0.336
Pclass_3         0.322
Cabin_0          0.320
Pclass_1         0.286
fare_grouped_4   0.232
f_size_1         0.203
is_alone_0       0.203
is_alone_1       0.203
fare_grouped_1   0.186
Cabin_5          0.175
Embarked_3       0.168
f_size_2         0.163
Cabin_7          0.151
Embarked_1       0.150
Cabin_6          0.145
f_size_3         0.144
f_size_4         0.128
ticket_double_0  0.122
ticket_double_1  0.122
```

```
age_grouped_1    0.117
Cabin_3          0.115
fare_grouped_2   0.113
Pclass_2         0.093
Title_3          0.085
f_size_6         0.081
fare_grouped_3   0.075
age_grouped_2    0.073
f_size_11        0.070
f_size_8         0.065
Cabin_4          0.058
f_size_5         0.049
age_grouped_4    0.044
age_grouped_3    0.036
Cabin_1          0.022
Cabin_2          0.016
age_grouped_5    0.015
Title_2          0.012
f_size_7         0.012
age_grouped_0    0.009
Embarked_2       0.004
```

Out[57]:
```
<AxesSubplot:ylabel='feature'>
```



In [58]:
```
# lets have dataset with features rates
liverate = pd.DataFrame(list(bfull.columns)[:-1], columns = ['feature'])
```

In [59]:
```
liverate[[ 'nSurv', 'Surv', 'count']] = 0
```

In [60]:
```
for i, feat in enumerate(liverate.feature):
    col = feat
    target = 'Survived'
    sort = col
    r = bfull[:891].pivot_table(values = target, index = col).sort_values(sort)
    q = bfull[:891].groupby(feat).count()
    liverate.loc[i, 'count'] = q.iloc[1,0]
    liverate.loc[i, 'nSurv'] = r.loc[0,'Survived'].round(3)
    liverate.loc[i, 'Surv'] = r.loc[1,'Survived'].round(3)
```

In [61]:
```
liverate = pd.merge(liverate, final_corr[['feature', 'Corr']], on = 'feature',
```

```
                              how = 'outer').sort_values('feature').reset_index().drop('index', axis=1)
```

In [62]:

```
liverate.sort_values('feature')
```

Out[62]:

	feature	nSurv	Surv	count	Corr
0	Cabin_0	0.670	0.299	688.0	0.320
1	Cabin_1	0.382	0.467	15.0	0.022
2	Cabin_2	0.383	0.500	4.0	0.016
3	Cabin_3	0.369	0.593	59.0	0.115
4	Cabin_4	0.380	0.615	13.0	0.058
5	Cabin_5	0.364	0.745	47.0	0.175
6	Cabin_6	0.370	0.750	32.0	0.145
7	Cabin_7	0.369	0.758	33.0	0.151
8	Embarked_1	0.502	0.339	646.0	0.150
9	Embarked_2	0.383	0.390	77.0	0.004
10	Embarked_3	0.344	0.554	168.0	0.168
11	Pclass_1	0.305	0.630	216.0	0.286
12	Pclass_2	0.361	0.473	184.0	0.093
13	Pclass_3	0.558	0.242	491.0	0.322
14	Sex_1	0.742	0.189	577.0	0.543
15	Sex_2	0.189	0.742	314.0	0.543
16	Survived	NaN	NaN	NaN	1.000
17	Title_1	0.698	0.157	517.0	0.549
18	Title_2	0.385	0.348	23.0	0.012
19	Title_3	0.375	0.575	40.0	0.085
20	Title_4	0.300	0.703	185.0	0.336
21	Title_5	0.316	0.794	126.0	0.342

	feature	nSurv	Surv	count	Corr
22	age_grouped_0	0.385	0.364	44.0	0.009
23	age_grouped_1	0.367	0.580	69.0	0.117
24	age_grouped_2	0.414	0.342	374.0	0.073
25	age_grouped_3	0.373	0.412	250.0	0.036
26	age_grouped_4	0.386	0.143	7.0	0.044
27	age_grouped_5	0.387	0.367	147.0	0.015
28	f_size_1	0.506	0.304	537.0	0.203
29	f_size_11	0.387	0.000	7.0	0.070
30	f_size_2	0.347	0.553	161.0	0.163
31	f_size_3	0.359	0.578	102.0	0.144
32	f_size_4	0.372	0.724	29.0	0.128
33	f_size_5	0.387	0.200	15.0	0.049
34	f_size_6	0.390	0.136	22.0	0.081
35	f_size_7	0.385	0.333	12.0	0.012
36	f_size_8	0.386	0.000	6.0	0.065
37	fare_grouped_1	0.437	0.229	227.0	0.186
38	fare_grouped_2	0.417	0.293	239.0	0.113
39	fare_grouped_3	0.364	0.450	209.0	0.075
40	fare_grouped_4	0.320	0.583	216.0	0.232
41	is_alone_0	0.506	0.304	537.0	0.203
42	is_alone_1	0.304	0.506	354.0	0.203
43	ticket_double_0	0.490	0.351	681.0	0.122
44	ticket_double_1	0.351	0.490	210.0	0.122

In [63]:
```
# we have different ways to chose starting fetures from whole fetures list to test the difference
# (there is almost no difference what features you start with, because we will reduce the number of them later)
df_nSurv = sorted(list(liverate.sort_values(['nSurv'],
                                            ascending = False).head(20).feature) + ['Survived'])
df_Surv = sorted(list(liverate.sort_values(['Surv'],
                                           ascending = False).head(20).feature) + ['Survived'])
df_count = sorted(list(liverate.sort_values(['count'],
                                            ascending = False).head(20).feature) + ['Survived'])
df_corr = sorted(list(liverate.sort_values(['Corr'], ascending = False).head(21).feature))

df_mix = sorted(list(set(pd.concat([liverate.sort_values(['nSurv'], ascending = False).head(10).feature,
                    liverate.sort_values(['Surv'], ascending = False).head(25).feature,
                        liverate.sort_values(['count'], ascending = False).head(7).feature,
                        liverate.sort_values(['Corr'], ascending = False).head(8).feature]))))
df_max = list(liverate.feature)
df_mix
```

Out[63]:
```
['Cabin_0',
 'Cabin_1',
 'Cabin_2',
 'Cabin_3',
 'Cabin_4',
 'Cabin_5',
 'Cabin_6',
 'Cabin_7',
 'Embarked_1',
 'Embarked_2',
 'Embarked_3',
 'Pclass_1',
 'Pclass_2',
 'Pclass_3',
 'Sex_1',
 'Sex_2',
 'Survived',
 'Title_1',
 'Title_3',
 'Title_4',
 'Title_5',
 'age_grouped_1',
 'age_grouped_3',
 'age_grouped_5',
 'f_size_1',
 'f_size_2',
 'f_size_3',
 'f_size_4',
 'fare_grouped_1',
```

```
 'fare_grouped_2',
 'fare_grouped_3',
 'fare_grouped_4',
 'is_alone_0',
 'is_alone_1',
 'ticket_double_0',
 'ticket_double_1']
```

In [64]:

```
start = df_mix
```

Next try array

In [65]:

```
# we will put here best features after first models testing

df_next_try = ['Cabin_0',
 'Embarked_3',
 'Pclass_1',
 'Pclass_2',
 'Pclass_3',
 'Sex_1',
 'Sex_2',
 'Title_1',
 'Title_4',
 'Title_5',
 'age_grouped_2',
 'age_grouped_4',
 'f_size_1',
 'f_size_3',
 'is_alone_0'] + ['Survived']
```

In [66]:

```
start = df_next_try
```

In [67]:

```
bfull_test = bfull[start]
```

In [68]:

```
X_train0 = X_train = bfull_test[:891].drop('Survived', axis = 1).astype(int)
X_test0 = bfull_test[891:].drop('Survived', axis = 1).astype(int)
y_train0 = bfull_test[:891].Survived.astype(int)
# y_test0 = pd.read_csv('test.csv').drop('PassengerId', axis  = 1)
```

In [69]:

```
X_train, X_test, y_train, y_test = train_test_split(X_train0, y_train0, test_size=0.25, random_state=0)
```

In [70]:

```
X_train.describe()
```

Out[70]:

	Cabin_0	Embarked_3	Pclass_1	Pclass_2	Pclass_3	Sex_1	Sex_2	Title_1	Title_4	Title_5	age_grouped_2	age_grou
count	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668.0
mean	0.766467	0.173653	0.244012	0.206587	0.549401	0.654192	0.345808	0.589820	0.202096	0.140719	0.428144	0.0
std	0.423395	0.379094	0.429822	0.405160	0.497926	0.475988	0.475988	0.492235	0.401864	0.347992	0.495181	0.0
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
25%	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.0
50%	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	0.000000	1.000000	0.000000	0.000000	0.000000	0.0
75%	1.000000	0.000000	0.000000	0.000000	1.000000	1.000000	1.000000	1.000000	0.000000	0.000000	1.000000	0.0
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.0

Normalization

In [71]:

```
# sometimes we need to make some preprocessing on our data
# some algorithms show better results

columns = list(X_train.columns)

NZ = preprocessing.Normalizer()
X_NZ = pd.DataFrame(NZ.fit_transform(X_train), columns = columns)
RS = preprocessing.RobustScaler()
X_RS = pd.DataFrame(RS.fit_transform(X_train), columns = columns)
MM = preprocessing.MinMaxScaler()
X_MM = pd.DataFrame(MM.fit_transform(X_train), columns = columns)
SS = preprocessing.StandardScaler()
X_SS = pd.DataFrame(SS.fit_transform(X_train), columns = columns)
```

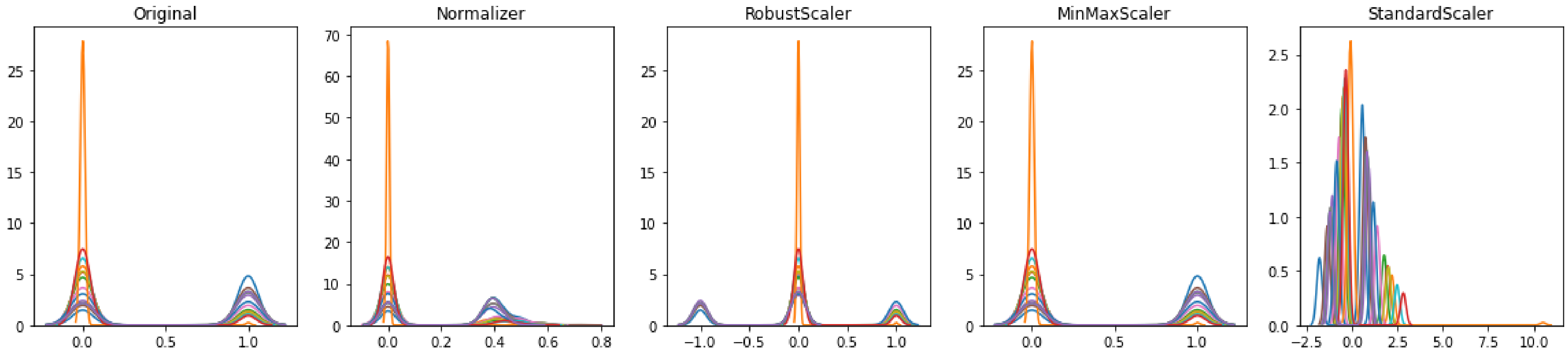
In [72]:

```
plots = 5
fig, ax = plt.subplots(ncols=plots, figsize=(20, 4))

titles  = ['Original', 'Normalizer', 'RobustScaler', 'MinMaxScaler', 'StandardScaler']
dfs = [X_train, X_NZ, X_RS, X_MM, X_SS]

for i in range(plots):
    ax[i].set_title(titles[i])
    for col in columns:
```

```
        sns.kdeplot(dfs[i][col], ax=ax[i], bw=0.15, legend = None)
        ax[i].set_xlabel(None)
        ax[i].set_ylabel(None)
```



RandomizedSearch

In [73]:
```
# we can use 2 steps pretraing for models
# 1st step - random test for some model parameters, its good, because RandomSearch works musch faster then GridSearch

rfc = RandomForestClassifier()
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 800, num = 10)]
max_features = ['log2', 'sqrt']
max_depth = [int(x) for x in np.linspace(start = 3, stop = 15, num = 15)]
min_samples_split = [int(x) for x in np.linspace(start = 2, stop = 50, num = 15)]
min_samples_leaf = [int(x) for x in np.linspace(start = 2, stop = 50, num = 15)]
bootstrap = [True, False]
param_dist = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
rs = RandomizedSearchCV(rfc,
                        param_dist,
                        n_iter = 200,
                        cv = 3,
                        verbose = 1,
                        n_jobs=-1,
                        random_state=0)
rs.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 200 candidates, totalling 600 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   22.7s
[Parallel(n_jobs=-1)]: Done 192 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done 442 tasks      | elapsed:  3.1min
[Parallel(n_jobs=-1)]: Done 600 out of 600 | elapsed:  4.1min finished
```

Out[73]:

```
RandomizedSearchCV(cv=3, estimator=RandomForestClassifier(), n_iter=200,
                   n_jobs=-1,
                   param_distributions={'bootstrap': [True, False],
                                        'max_depth': [3, 3, 4, 5, 6, 7, 8, 9, 9,
                                                      10, 11, 12, 13, 14, 15],
                                        'max_features': ['log2', 'sqrt'],
                                        'min_samples_leaf': [2, 5, 8, 12, 15,
                                                             19, 22, 26, 29, 32,
                                                             36, 39, 43, 46,
                                                             50],
                                        'min_samples_split': [2, 5, 8, 12, 15,
                                                              19, 22, 26, 29,
                                                              32, 36, 39, 43,
                                                              46, 50],
                                        'n_estimators': [100, 177, 255, 333,
                                                         411, 488, 566, 644,
                                                         722, 800]},
                   random_state=0, verbose=1)
```

In [74]:

```python
rs_df = pd.DataFrame(rs.cv_results_).sort_values('rank_test_score').reset_index(drop=True)
rs_df = rs_df.drop([
            'mean_fit_time',
            'std_fit_time',
            'mean_score_time',
            'std_score_time',
            'params',
            'split0_test_score',
            'split1_test_score',
            'split2_test_score',
            'std_test_score'],
            axis=1)
rs_df.sort_values('mean_test_score', ascending = False)
```

Out[74]:

	param_n_estimators	param_min_samples_split	param_min_samples_leaf	param_max_features	param_max_depth	param_bootstrap	mean_test_score	ra
0	100	12	5	sqrt	9	True	0.815908	
1	333	50	8	sqrt	15	False	0.814420	

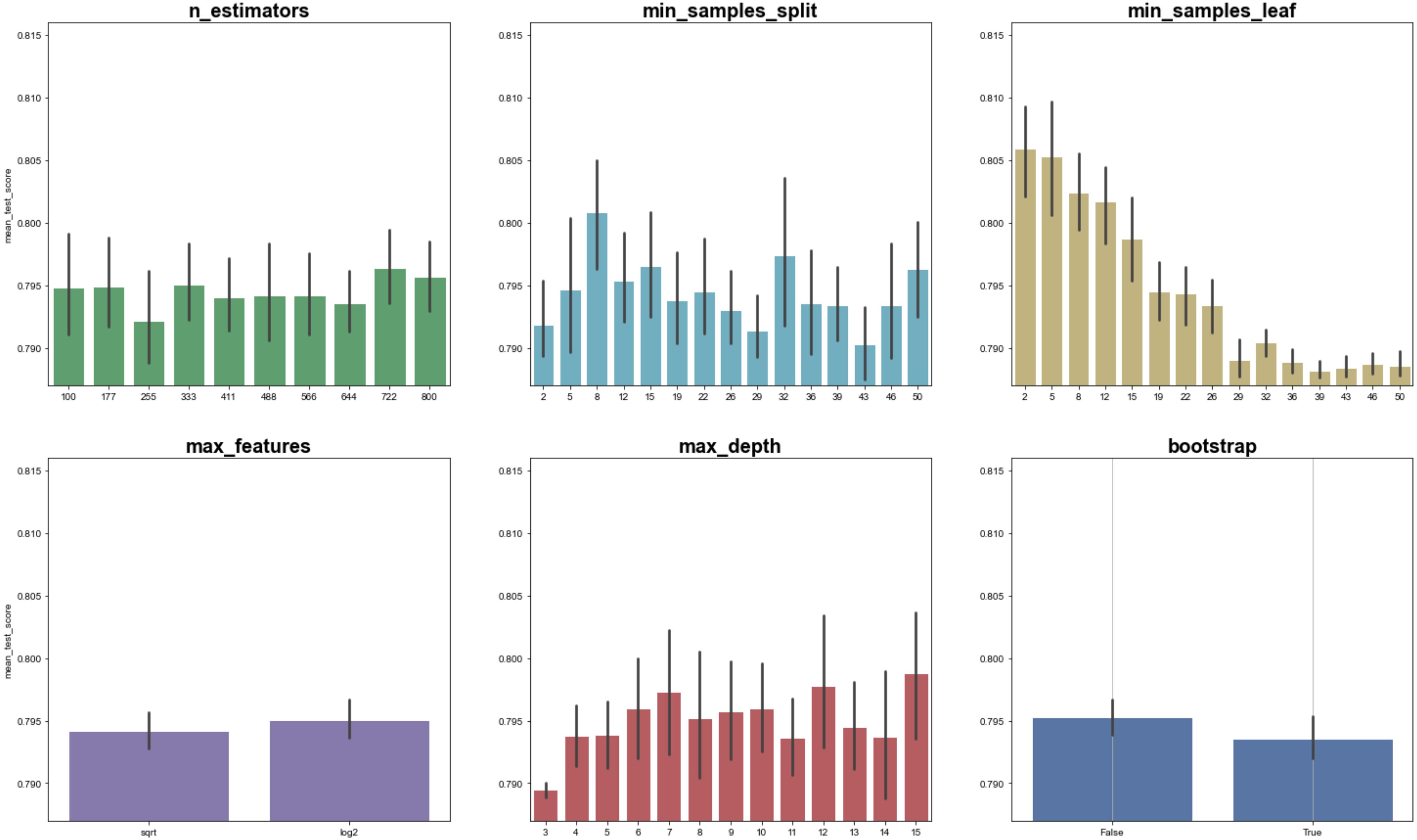
	param_n_estimators	param_min_samples_split	param_min_samples_leaf	param_max_features	param_max_depth	param_bootstrap	mean_test_score	ra
2	100	32	5	sqrt	8	False	0.814413	
3	177	36	2	log2	10	False	0.814400	
4	177	5	2	sqrt	9	False	0.812912	
...	...	...	...	...	...	...	...	
167	488	5	29	log2	14	True	0.787453	
168	411	26	26	log2	12	True	0.787453	
169	644	39	46	sqrt	14	False	0.787453	
170	333	29	43	sqrt	9	True	0.787453	
199	177	19	26	log2	14	True	0.787453	

200 rows × 8 columns

In [75]:

```
colors = ['g', 'c', 'y',  'm', 'r', 'b']
columns = list(rs_df.columns)
y_min = round(rs_df['mean_test_score'].min(), 3)
y_max = round(rs_df['mean_test_score'].max(), 3)
fig, axs = plt.subplots(ncols=3, nrows=2, figsize=(25,15), )
sns.set(style="whitegrid", font_scale = 2)
# fig.set_size_inches(25,18)

for i,k in enumerate(columns[:6]):
    sns.barplot(x=k, y=columns[6], data=rs_df, color=colors[i], ax = axs[i//3,i%3])
    axs[i//3,i%3].set_ylim((y_min, y_max))
    axs[i//3,i%3].set_title(label = k[6:], weight='bold', fontsize = 20)
    axs[i//3,i%3].set_xlabel(None,fontsize = 18)
    axs[i//3,1].set_ylabel(None)
    axs[i//3,2].set_ylabel(None)
    plt.grid()
plt.show()
```



GridSearch

In [76]:

```
# now we can use some of the nest params in GridSearch to tune the model

rfc_2 = RandomForestClassifier()
n_estimators = [722]
max_features = ['log2']
max_depth = [6, 7, 12, 15]
```

```
min_samples_split = [8, 32]
min_samples_leaf = [2,3,4,5,6,7,8]
bootstrap = [False]
param_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf,
               'bootstrap': bootstrap}
gs = GridSearchCV(rfc_2, param_grid, cv = 3, verbose = 1, n_jobs=-1)
gs.fit(X_train, y_train)
rfc_3 = gs.best_estimator_
best_params = gs.best_params_
best_params
```

```
Fitting 3 folds for each of 56 candidates, totalling 168 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 4 concurrent workers.
[Parallel(n_jobs=-1)]: Done  42 tasks      | elapsed:   22.8s
[Parallel(n_jobs=-1)]: Done 168 out of 168 | elapsed:  1.5min finished
```

Out[76]:
```
{'bootstrap': False,
 'max_depth': 15,
 'max_features': 'log2',
 'min_samples_leaf': 6,
 'min_samples_split': 32,
 'n_estimators': 722}
```

In [77]:
```
model_rfc = RandomForestClassifier(**(best_params)).fit(X_train, y_train)
Y_pred_RF = model_rfc.predict(X_test)
RF = round(metrics.accuracy_score(y_test, Y_pred_RF), 5)
print(RF)
```

```
0.80269
```

Algorithms combine

In [78]:
```
# im going to start algorithm factory:) want it try all algorithms and compair it
# but im too lazy to do it, so this ive automatised it:)
s_names = ['RFC', 'LR', 'ENCV', 'OMP', 'LDA', 'QDA', 'RC', 'KNC', 'SVC', 'GNB', 'PCT', 'LSVC', 'SGDC', 'DTC',
           'MLPC', 'LGBM', 'CBC', 'RCCV', 'ABC', 'PAC', 'LRCV', 'ETC', 'LLCV', 'ETsC', 'GBC', 'HGBC', 'NSVC',
           'LPG', 'BC', 'CNB', 'LrCV', 'LsCV', 'XGBC']


alg = [RandomForestClassifier(), LogisticRegression(), ElasticNetCV(), OrthogonalMatchingPursuit(),
       LinearDiscriminantAnalysis(), QuadraticDiscriminantAnalysis(), RidgeClassifier(), KNeighborsClassifier(),
       SVC(), GaussianNB(), Perceptron(), LinearSVC(), SGDClassifier(), DecisionTreeClassifier(), MLPClassifier(),
```

```python
        LGBMClassifier(), CatBoostClassifier(verbose=False), RidgeClassifierCV(), AdaBoostClassifier(),
        PassiveAggressiveClassifier(), LogisticRegressionCV(), ExtraTreeClassifier(), LassoLarsCV(),
        ExtraTreesClassifier(), GradientBoostingClassifier(), HistGradientBoostingClassifier(), NuSVC(),
        LabelPropagation(), BaggingClassifier(), ComplementNB(), LarsCV(), LassoCV(), XGBClassifier()]

algs = pd.DataFrame(columns = ['name', 'algorithm', 's_name',
                               'acc', 'pres_1', 'pres_2', 'rec_1', 'rec_2', 'fsc_1', 'fsc_2', 'matrix', 'nS_S', 'model'])

algs['algorithm'] = alg
algs['s_name'] = s_names


def add_column(col_name):
    list_temp = []
    for i in range(len(alg)):
        list_temp.append({})
    algs[col_name] = list_temp


# we can use those params for Random or Grid testing in aotmat mode
param_grid =  {
                'AdaBoostClassifier': {},
                'BaggingClassifier': {},
                'CatBoostClassifier': {},
                'ComplementNB': {},
                'DecisionTreeClassifier': {},
                'ElasticNetCV': {},
                'ExtraTreeClassifier': {},
                'ExtraTreesClassifier': {},
                'GaussianNB': {},
                'GradientBoostingClassifier': {},
                'HistGradientBoostingClassifier': {},
                'KNeighborsClassifier': {},
                'LGBMClassifier':
                    {
                        'iterations': [10, 20, 50, 100, 300, 500],
                        'learning_rate': [0.01, 0.05, 0.1],
                        'depth': [5, 7, 9, 11],
                        'l2_leaf_reg': [1, 3, 5, 7, 9]
                    },
                'LabelPropagation': {},
                'LarsCV': {},
                'LassoCV': {},
                'LassoLarsCV': {},
```

```
                'LinearDiscriminantAnalysis': {},
                'LinearSVC': {},
                'LogisticRegression': {},
                'LogisticRegressionCV': {},
                'MLPClassifier':
                    {
                        'n_estimators': [25, 50, 100],
                        'learning_rate': [0.01, 0.05, 0.1],
                        'num_leaves': [7, 15, 31]
                    },
                'NuSVC': {},
                'OrthogonalMatchingPursuit': {},
                'PassiveAggressiveClassifier': {},
                'Perceptron': {},
                'QuadraticDiscriminantAnalysis': {},
                'RandomForestClassifier':
                    {
                        'n_estimators': [20, 50, 200, 500, 800],
                        'max_features': ['log2', 'sqrt'],
                        'max_depth': [5, 7, 10],
                        'min_samples_split': [1, 5, 10, 20],
                        'min_samples_leaf': [2,3,4,5],
                        'bootstrap': [False, True]
                    },
                'RidgeClassifier': {},
                'RidgeClassifierCV': {},
                'SGDClassifier': {},
                'SVC': {},
                'XGBClassifier': {}
            }

# we can use those params for algorithms in automode
best_params =  {
                'AdaBoostClassifier': {},
                'BaggingClassifier': {},
                'CatBoostClassifier': {},
                'ComplementNB': {},
                'DecisionTreeClassifier': {},
                'ElasticNetCV': {},
                'ExtraTreeClassifier': {},
                'ExtraTreesClassifier': {},
                'GaussianNB': {},
                'GradientBoostingClassifier': {},
                'HistGradientBoostingClassifier': {},
```

```
                    'KNeighborsClassifier':
                        {
                            'n_neighbors': 4,
                            'weights': 'uniform',
                            'algorithm': 'brute'
                        },
                    'LGBMClassifier':
                        {
                            'random_state': 0,
                            'learning_rate': 0.01,
                            'num_leaves': 3,
                            'n_estimators': 100,
                            'class_weight': 'balanced',
                            'n_jobs': -1
                        },
                    'LabelPropagation': {},
                    'LarsCV': {},
                    'LassoCV': {},
                    'LassoLarsCV': {},
                    'LinearDiscriminantAnalysis': {},
                    'LinearSVC': {},
                    'LogisticRegression':
                        {
                            'random_state': 1,
                            'solver': 'liblinear',
                            'penalty': 'l1'
                        },
                    'LogisticRegressionCV':
                        {'cv': 5,
                            'random_state': 0,
                            'dual': False,
                            'penalty': 'l2',
                            'solver': 'lbfgs',
                            'max_iter': 100,
                            'n_jobs': -1
                        },
                    'MLPClassifier':
                        {
                            'solver': 'lbfgs',
                            'alpha': 1e-05,
                            'hidden_layer_sizes': (4, 3),
                            'random_state': 0,
                            'max_iter': 10000,
                            'learning_rate': 'adaptive'
```

```
                 },
               'NuSVC': {},
               'OrthogonalMatchingPursuit': {},
               'PassiveAggressiveClassifier': {},
               'Perceptron':
                   {
                       'random_state': 0,
                       'penalty': 'l2',
                       'max_iter': 5000
                   },
               'QuadraticDiscriminantAnalysis': {},
               'RandomForestClassifier':
                   {
                       'n_estimators': 20,
                       'criterion': 'gini',
                       'max_features': 'log2',
                       'max_depth': 8,
                       'min_samples_split': 3,
                       'min_samples_leaf': 13,
                       'bootstrap': True,
                       'random_state': 0
                   },
               'RidgeClassifier': {},
               'RidgeClassifierCV': {},
               'SGDClassifier': {},
               'SVC': {},
               'XGBClassifier':
                   {
                       'random_state': 0,
                       'learning_rate': 0.01,
                       'max_depth': 2,
                       'n_estimators': 20,
                       'n_jobs': -1
                   }
               }

names = []
for i in alg:
    names.append(str(i))

algs['names'] = names
# algs['names'] = algs['algorithm'].apply('str')
algs['name'] = algs['names'].str.extract('([A-Za-z]+)', expand = False)
algs[algs.name == 'catboost']
```

```
algs.loc[16, 'name'] = 'CatBoostClassifier'
algs = algs.sort_values('name').reset_index().drop(['names', 'index'], axis  =1)
algs['param_grid'] = dict.fromkeys(algs['name'], {})
add_column('best_params')
add_column('best_model')
algs['type'] = algs['algorithm'].apply(lambda x: 'classifier'if is_classifier(x) else 'regressor'if is_regressor(x) else 'NA')
algs['param_grid'] = algs['name'].apply(lambda x: param_grid[x])
algs['best_params'] = algs['name'].apply(lambda x: best_params[x])
# algs.to_csv ('algorithms_sklearn.csv', index = False, encoding = 'utf-8')
```

In [79]:

```
# rfc_2 = RandomForestClassifier()
# n_estimators = [720,800]
# max_features = ['log2']
# max_depth = [5, 7, 10]
# min_samples_split = [8, 22]
# min_samples_leaf = [2,3,4,5]
# bootstrap = [False]
# param_grid = {'n_estimators': n_estimators,
#                'max_features': max_features,
#                'max_depth': max_depth,
#                'min_samples_split': min_samples_split,
#                'min_samples_leaf': min_samples_leaf,
#                'bootstrap': bootstrap}
# gs = GridSearchCV(rfc_2, param_grid, cv = 3, verbose = 1, n_jobs=-1)
# gs.fit(X_train, y_train)
# rfc_3 = gs.best_estimator_
# gs.best_params_
```

In [80]:

```
# Algorithm factory here. All target scores saved to algs DataFrame
best_score = 0
best_model = ''
for i, alg in enumerate(algs.algorithm):
    print(alg)
#     param_grid = algs.loc[i, 'param_grid']
#     param_grid = {}
    best_params  = algs.loc[i, 'best_params']
    model = alg.set_params(**best_params)
#     try:
#         model = GridSearchCV(model, param_grid, cv = 5, scoring='accuracy', verbose = 0, n_jobs=-1).fit(X_train, y_train)
#     except:
#         continue
    try:
        model.fit(X_train, y_train)
```

```
    except:
        continue
    Y_pred = model.predict(X_test).round()
    accuracy_score = metrics.accuracy_score(y_test, Y_pred).round(5)
    f1_score = metrics.f1_score(y_test, Y_pred, average=None).round(5)
    pr_re_fscore  = metrics.precision_recall_fscore_support(y_test, Y_pred, average=None)
    confusion_matrix = metrics.confusion_matrix(y_test, Y_pred)
    if i == 27:
        RFC_score = accuracy_score
        RFC_model = model
    algs.at[i, 'acc'] = accuracy_score
    algs.at[i, 'pres_1'] = pr_re_fscore[0][0].round(5)
    algs.at[i, 'pres_2'] = pr_re_fscore[0][1].round(5)
    algs.at[i, 'rec_1'] = pr_re_fscore[1][0].round(5)
    algs.at[i, 'rec_2'] = pr_re_fscore[1][1].round(5)
    algs.at[i, 'fsc_1'] = pr_re_fscore[2][0].round(5)
    algs.at[i, 'fsc_2'] = pr_re_fscore[2][1].round(5)
    algs.at[i, 'matrix'] = confusion_matrix.round(5)
    algs.at[i, 'nS_S'] = pr_re_fscore[3].round(5)
    algs.at[i, 'model'] = model


#     algs.loc[i, 'best_params'] = model.best_params_
#     algs.loc[i, 'best_model'] = model.best_estimator_
# algs.to_csv ('class.csv', columns =['name', 'algorithm', 's_name', 'score'], index=False, encoding = 'utf-8')
```

```
AdaBoostClassifier()
BaggingClassifier()
<catboost.core.CatBoostClassifier object at 0x00000175DF028DC0>
ComplementNB()
DecisionTreeClassifier()
ElasticNetCV()
ExtraTreeClassifier()
ExtraTreesClassifier()
GaussianNB()
GradientBoostingClassifier()
HistGradientBoostingClassifier()
KNeighborsClassifier()
LGBMClassifier()
LabelPropagation()
LarsCV()
LassoCV()
LassoLarsCV()
LinearDiscriminantAnalysis()
LinearSVC()
LogisticRegression()
```

```
LogisticRegressionCV()
MLPClassifier()
NuSVC()
OrthogonalMatchingPursuit()
PassiveAggressiveClassifier()
Perceptron()
QuadraticDiscriminantAnalysis()
RandomForestClassifier()
RidgeClassifier()
RidgeClassifierCV(alphas=array([ 0.1,  1. , 10. ]))
SGDClassifier()
SVC()
XGBClassifier(base_score=None, booster=None, colsample_bylevel=None,
              colsample_bynode=None, colsample_bytree=None, gamma=None,
              gpu_id=None, importance_type='gain', interaction_constraints=None,
              learning_rate=None, max_delta_step=None, max_depth=None,
              min_child_weight=None, missing=nan, monotone_constraints=None,
              n_estimators=100, n_jobs=None, num_parallel_tree=None,
              random_state=None, reg_alpha=None, reg_lambda=None,
              scale_pos_weight=None, subsample=None, tree_method=None,
              validate_parameters=None, verbosity=None)
```

Final result voting

In [81]:
```
# dataset with various csores from all algoritys finished
algs[['name', 's_name', 'acc', 'pres_1', 'pres_2', 'rec_1', 'rec_2',
      'fsc_1', 'fsc_2', 'matrix', 'nS_S']].sort_values('acc', ascending = False)
```

Out[81]:

	name	s_name	acc	pres_1	pres_2	rec_1	rec_2	fsc_1	fsc_2	matrix	nS_S
9	GradientBoostingClassifier	GBC	0.82511	0.84722	0.78481	0.8777	0.7381	0.86219	0.76074	[[122, 17], [22, 62]]	[139, 84]
31	SVC	SVC	0.81166	0.82119	0.79167	0.89209	0.67857	0.85517	0.73077	[[124, 15], [27, 57]]	[139, 84]
10	HistGradientBoostingClassifier	HGBC	0.81166	0.84397	0.7561	0.85612	0.7381	0.85	0.74699	[[119, 20], [22, 62]]	[139, 84]
22	NuSVC	NSVC	0.80269	0.82759	0.75641	0.86331	0.70238	0.84507	0.7284	[[120, 19], [25, 59]]	[139, 84]
27	RandomForestClassifier	RFC	0.80269	0.82313	0.76316	0.8705	0.69048	0.84615	0.725	[[121, 18], [26, 58]]	[139, 84]
2	CatBoostClassifier	CBC	0.79821	0.81757	0.76	0.8705	0.67857	0.84321	0.71698	[[121, 18], [27, 57]]	[139, 84]
7	ExtraTreesClassifier	ETsC	0.79821	0.81333	0.76712	0.8777	0.66667	0.84429	0.71338	[[122, 17], [28, 56]]	[139, 84]
11	KNeighborsClassifier	KNC	0.79372	0.78882	0.80645	0.91367	0.59524	0.84667	0.68493	[[127, 12], [34, 50]]	[139, 84]
1	BaggingClassifier	BC	0.79372	0.82979	0.73171	0.84173	0.71429	0.83571	0.72289	[[117, 22], [24, 60]]	[139, 84]

	name	s_name	acc	pres_1	pres_2	rec_1	rec_2	fsc_1	fsc_2	matrix	nS_S
3	ComplementNB	CNB	0.78924	0.84328	0.70787	0.81295	0.75	0.82784	0.72832	[[113, 26], [21, 63]]	[139, 84]
8	GaussianNB	GNB	0.78924	0.85385	0.69892	0.79856	0.77381	0.82528	0.73446	[[111, 28], [19, 65]]	[139, 84]
13	LabelPropagation	LPG	0.78924	0.80263	0.76056	0.8777	0.64286	0.83849	0.69677	[[122, 17], [30, 54]]	[139, 84]
20	LogisticRegressionCV	LRCV	0.78475	0.83704	0.70455	0.81295	0.7381	0.82482	0.72093	[[113, 26], [22, 62]]	[139, 84]
19	LogisticRegression	LR	0.78475	0.83704	0.70455	0.81295	0.7381	0.82482	0.72093	[[113, 26], [22, 62]]	[139, 84]
18	LinearSVC	LSVC	0.78475	0.83704	0.70455	0.81295	0.7381	0.82482	0.72093	[[113, 26], [22, 62]]	[139, 84]
17	LinearDiscriminantAnalysis	LDA	0.78475	0.83704	0.70455	0.81295	0.7381	0.82482	0.72093	[[113, 26], [22, 62]]	[139, 84]
0	AdaBoostClassifier	ABC	0.78475	0.83704	0.70455	0.81295	0.7381	0.82482	0.72093	[[113, 26], [22, 62]]	[139, 84]
25	Perceptron	PCT	0.78475	0.79355	0.76471	0.88489	0.61905	0.83673	0.68421	[[123, 16], [32, 52]]	[139, 84]
28	RidgeClassifier	RC	0.78475	0.83704	0.70455	0.81295	0.7381	0.82482	0.72093	[[113, 26], [22, 62]]	[139, 84]
21	MLPClassifier	MLPC	0.78475	0.80537	0.74324	0.86331	0.65476	0.83333	0.6962	[[120, 19], [29, 55]]	[139, 84]
6	ExtraTreeClassifier	ETC	0.78027	0.8	0.73973	0.86331	0.64286	0.83045	0.6879	[[120, 19], [30, 54]]	[139, 84]
4	DecisionTreeClassifier	DTC	0.78027	0.79605	0.74648	0.8705	0.63095	0.83162	0.68387	[[121, 18], [31, 53]]	[139, 84]
32	XGBClassifier	XGBC	0.77578	0.76023	0.82692	0.93525	0.5119	0.83871	0.63235	[[130, 9], [41, 43]]	[139, 84]
29	RidgeClassifierCV	RCCV	0.76682	0.83206	0.67391	0.78417	0.7381	0.80741	0.70455	[[109, 30], [22, 62]]	[139, 84]
24	PassiveAggressiveClassifier	PAC	0.76233	0.84127	0.65979	0.76259	0.7619	0.8	0.70718	[[106, 33], [20, 64]]	[139, 84]
15	LassoCV	LsCV	0.76233	0.83077	0.66667	0.77698	0.7381	0.80297	0.70056	[[108, 31], [22, 62]]	[139, 84]
5	ElasticNetCV	ENCV	0.76233	0.83077	0.66667	0.77698	0.7381	0.80297	0.70056	[[108, 31], [22, 62]]	[139, 84]
23	OrthogonalMatchingPursuit	OMP	0.75785	0.84553	0.65	0.7482	0.77381	0.79389	0.70652	[[104, 35], [19, 65]]	[139, 84]
14	LarsCV	LrCV	0.75785	0.83077	0.66304	0.77698	0.72619	0.80297	0.69318	[[108, 31, 0], [22, 61, 1], [0, 0, 0]]	[139, 84, 0]
12	LGBMClassifier	LGBM	0.75785	0.84553	0.65	0.7482	0.77381	0.79389	0.70652	[[104, 35], [19, 65]]	[139, 84]
16	LassoLarsCV	LLCV	0.75785	0.82946	0.65957	0.76978	0.7381	0.79851	0.69663	[[107, 32], [22, 62]]	[139, 84]
30	SGDClassifier	SGDC	0.74439	0.86607	0.62162	0.69784	0.82143	0.77291	0.70769	[[97, 42], [15, 69]]	[139, 84]
26	QuadraticDiscriminantAnalysis	QDA	0.69955	0.71951	0.64407	0.84892	0.45238	0.77888	0.53147	[[118, 21], [46, 38]]	[139, 84]

In [82]:
```
# now ill try to chose better combination of algorithm results to reach max accuracy
```

In [83]:
```
# those give max acuracy (top4)
t1 = algs[['name', 'algorithm', 'acc', 'rec_1', 'rec_2','fsc_1', 'fsc_2']][algs.type == 'classifier']\
                    .sort_values('acc', ascending = False)[0:4].reset_index().drop('index', axis  =1)
t1
```

Out[83]:

	name	algorithm	acc	rec_1	rec_2	fsc_1	fsc_2
0	GradientBoostingClassifier	([DecisionTreeRegressor(criterion='friedman_ms...	0.82511	0.8777	0.7381	0.86219	0.76074
1	SVC	SVC()	0.81166	0.89209	0.67857	0.85517	0.73077
2	HistGradientBoostingClassifier	HistGradientBoostingClassifier()	0.81166	0.85612	0.7381	0.85	0.74699
3	RandomForestClassifier	(DecisionTreeClassifier(max_depth=8, max_featu...	0.80269	0.8705	0.69048	0.84615	0.725

In [84]:
```
#max pression not Survived (top1)
t2 = algs[['name', 'algorithm', 'acc', 'pres_1', 'pres_2', 'rec_1', 'rec_2','fsc_1', 'fsc_2']][algs.type == 'classifier']\
                    .sort_values('pres_1', ascending = False)[0:1].reset_index().drop('index', axis  =1)
t2
```

Out[84]:

	name	algorithm	acc	pres_1	pres_2	rec_1	rec_2	fsc_1	fsc_2
0	SGDClassifier	SGDClassifier()	0.74439	0.86607	0.62162	0.69784	0.82143	0.77291	0.70769

In [85]:
```
#max pression Survived (top1)
t3 = algs[['name', 'algorithm', 'acc', 'pres_1', 'pres_2', 'rec_1', 'rec_2','fsc_1', 'fsc_2']][algs.type == 'classifier']\
                    .sort_values('pres_2', ascending = False)[0:1].reset_index().drop('index', axis  =1)
t3
```

Out[85]:

	name	algorithm	acc	pres_1	pres_2	rec_1	rec_2	fsc_1	fsc_2
0	XGBClassifier	XGBClassifier(base_score=0.5, booster='gbtree'...	0.77578	0.76023	0.82692	0.93525	0.5119	0.83871	0.63235

In [86]:
```
# max recall for not Suvived here (top1)
t4 = algs[['name', 'algorithm', 'acc', 'pres_1', 'pres_2', 'rec_1', 'rec_2','fsc_1', 'fsc_2']][algs.type == 'classifier']\
                    .sort_values('rec_1', ascending = False)[0:1].reset_index().drop('index', axis  =1)
t4
```

Out[86]:

	name	algorithm	acc	pres_1	pres_2	rec_1	rec_2	fsc_1	fsc_2
0	XGBClassifier	XGBClassifier(base_score=0.5, booster='gbtree'...	0.77578	0.76023	0.82692	0.93525	0.5119	0.83871	0.63235

In [87]:

```
# max recall for 'Suvived' here (top1)
t5 = algs[['name', 'algorithm', 'acc', 'pres_1', 'pres_2', 'rec_1', 'rec_2','fsc_1', 'fsc_2']][algs.type == 'classifier']\
                    .sort_values('rec_2', ascending = False)[0:1].reset_index().drop('index', axis  =1)
t5
```

Out[87]:

	name	algorithm	acc	pres_1	pres_2	rec_1	rec_2	fsc_1	fsc_2
0	SGDClassifier	SGDClassifier()	0.74439	0.86607	0.62162	0.69784	0.82143	0.77291	0.70769

In [88]:

```
#F-score for not Survived/Survived (top2 each)
tf1  = algs[['name', 'algorithm', 'acc', 'pres_1', 'pres_2', 'rec_1', 'rec_2','fsc_1', 'fsc_2']][algs.type == 'classifier']\
                    .sort_values('fsc_1', ascending = False)[0:2].reset_index().drop('index', axis  =1)
tf2  = algs[['name', 'algorithm', 'acc', 'pres_1', 'pres_2', 'rec_1', 'rec_2','fsc_1', 'fsc_2']][algs.type == 'classifier']\
                    .sort_values('fsc_2', ascending = False)[0:2].reset_index().drop('index', axis  =1)

tf = pd.concat([tf1, tf2], axis=0).drop_duplicates('name')
tf
```

Out[88]:

	name	algorithm	acc	pres_1	pres_2	rec_1	rec_2	fsc_1	fsc_2
0	GradientBoostingClassifier	([DecisionTreeRegressor(criterion='friedman_ms...	0.82511	0.84722	0.78481	0.8777	0.7381	0.86219	0.76074
1	SVC	SVC()	0.81166	0.82119	0.79167	0.89209	0.67857	0.85517	0.73077
1	HistGradientBoostingClassifier	HistGradientBoostingClassifier()	0.81166	0.84397	0.7561	0.85612	0.7381	0.85	0.74699

In [89]:

```
#top TN/TP scores (top1)
tm1 = algs[['name', 'algorithm', 'acc', 'pres_1', 'pres_2', 'rec_1', 'rec_2','fsc_1', 'fsc_2', 'matrix']][algs.type == 'classifier
                    .sort_values('matrix', key = lambda x: pd.Series(y[0][0] for y in x),
                                 ascending = False)[0:1].reset_index().drop('index', axis  =1)
tm2 = algs[['name', 'algorithm', 'acc', 'pres_1', 'pres_2', 'rec_1', 'rec_2','fsc_1', 'fsc_2', 'matrix']][algs.type == 'classifier
                    .sort_values('matrix', key = lambda x: pd.Series(y[0][1] for y in x),
                                 ascending = False)[0:1].reset_index().drop('index', axis  =1)

tm = pd.concat([tm1, tm2], axis=0).drop_duplicates('name')
tm
```

Out[89]:

	name	algorithm	acc	pres_1	pres_2	rec_1	rec_2	fsc_1	fsc_2	matrix
0	XGBClassifier	XGBClassifier(base_score=0.5, booster='gbtree'...	0.77578	0.76023	0.82692	0.93525	0.5119	0.83871	0.63235	[[130, 9], [41, 43]]
0	SGDClassifier	SGDClassifier()	0.74439	0.86607	0.62162	0.69784	0.82143	0.77291	0.70769	[[97, 42], [15, 69]]

In [90]:

```
t = pd.concat([t1, t2, t3, t4, t5], axis=0).drop_duplicates('name')
t
```

Out[90]:

	name	algorithm	acc	rec_1	rec_2	fsc_1	fsc_2	pres_1	pres_2
0	GradientBoostingClassifier	([DecisionTreeRegressor(criterion='friedman_ms...	0.82511	0.8777	0.7381	0.86219	0.76074	NaN	NaN
1	SVC	SVC()	0.81166	0.89209	0.67857	0.85517	0.73077	NaN	NaN
2	HistGradientBoostingClassifier	HistGradientBoostingClassifier()	0.81166	0.85612	0.7381	0.85	0.74699	NaN	NaN
3	RandomForestClassifier	(DecisionTreeClassifier(max_depth=8, max_featu...	0.80269	0.8705	0.69048	0.84615	0.725	NaN	NaN
0	SGDClassifier	SGDClassifier()	0.74439	0.69784	0.82143	0.77291	0.70769	0.86607	0.62162
0	XGBClassifier	XGBClassifier(base_score=0.5, booster='gbtree'...	0.77578	0.93525	0.5119	0.83871	0.63235	0.76023	0.82692

In [91]:

```
# we can play with different algorithms we want to use them at finishing Voting algorithm
t_full=pd.concat([t, tf, tm], axis=0).drop_duplicates('name').reset_index().drop('index', axis=1)
t_full
```

Out[91]:

	name	algorithm	acc	rec_1	rec_2	fsc_1	fsc_2	pres_1	pres_2	matrix
0	GradientBoostingClassifier	([DecisionTreeRegressor(criterion='friedman_ms...	0.82511	0.8777	0.7381	0.86219	0.76074	NaN	NaN	NaN
1	SVC	SVC()	0.81166	0.89209	0.67857	0.85517	0.73077	NaN	NaN	NaN
2	HistGradientBoostingClassifier	HistGradientBoostingClassifier()	0.81166	0.85612	0.7381	0.85	0.74699	NaN	NaN	NaN
3	RandomForestClassifier	(DecisionTreeClassifier(max_depth=8, max_featu...	0.80269	0.8705	0.69048	0.84615	0.725	NaN	NaN	NaN
4	SGDClassifier	SGDClassifier()	0.74439	0.69784	0.82143	0.77291	0.70769	0.86607	0.62162	NaN
5	XGBClassifier	XGBClassifier(base_score=0.5, booster='gbtree'...	0.77578	0.93525	0.5119	0.83871	0.63235	0.76023	0.82692	NaN

In [92]:

```
# t.drop([6], inplace=True)
# t.drop([8], inplace=True)
```

In [93]:
```
# So finishing Voting algorithm
estimators  = list(zip(t_full.name, t_full.algorithm))
eclf1 = VotingClassifier(estimators=estimators,
                         voting='hard').fit(X_train0, y_train0)
Y_pred_VT1 = eclf1.predict(X_test0).round()
# accuracy_score = metrics.accuracy_score(y_test, Y_pred_VT1).round(5)
# f1_score = metrics.f1_score(y_test, Y_pred_VT1, average=None).round(5)
# pr_re_fscore  = metrics.precision_recall_fscore_support(y_test, Y_pred_VT1, average=None)
# confusion_matrix = metrics.confusion_matrix(y_test, Y_pred_VT1)
# print('acc   : ' , accuracy_score)
# print('pres_1: ' , pr_re_fscore[0][0].round(5))
# print('pres_2: ' , pr_re_fscore[0][1].round(5))
# print('rec_1 : ' , pr_re_fscore[1][0].round(5))
# print('rec_2 : ' , pr_re_fscore[1][1].round(5))
# print('fsc_1 : ' , pr_re_fscore[2][0].round(5))
# print('fsc_1 : ' , pr_re_fscore[2][1].round(5))
# print('matrix: ' , confusion_matrix[0])
# print('        ' , confusion_matrix[1])
# metrics.plot_confusion_matrix(eclf1, X_test0, y_test0)
# test0 = eclf1.predict(X_test0).round()
# print(metrics.accuracy_score(y_test0, test0).round(5))
```

In [94]:
```
# example of cross-validation
cross_validate(eclf1, X_test, y_test, return_train_score=True, return_estimator=True, cv=2, n_jobs=-1)
```

Out[94]:
```
{'fit_time': array([0.35578108, 0.38776207]),
 'score_time': array([0.03098083, 0.03597593]),
 'estimator': (VotingClassifier(estimators=[('GradientBoostingClassifier',
                                GradientBoostingClassifier()),
                               ('SVC', SVC()),
                               ('HistGradientBoostingClassifier',
                                HistGradientBoostingClassifier()),
                               ('RandomForestClassifier',
                                RandomForestClassifier(max_depth=8,
                                                       max_features='log2',
                                                       min_samples_leaf=13,
                                                       min_samples_split=3,
                                                       n_estimators=20,
                                                       random_state=0)),
                               ('SGDClassifier', SGDClass...
                                              colsample_bytree=1, gamma=0,
                                              gpu_id=-1, importance_type='gain',
                                              interaction_constraints='',
                                              learning_rate=0.01,
                                              max_delta_step=0, max_depth=2,
```

```
                                             min_child_weight=1, missing=nan,
                                             monotone_constraints='()',
                                             n_estimators=20, n_jobs=-1,
                                             num_parallel_tree=1, random_state=0,
                                             reg_alpha=0, reg_lambda=1,
                                             scale_pos_weight=1, subsample=1,
                                             tree_method='exact',
                                             validate_parameters=1,
                                             verbosity=None))]),
 VotingClassifier(estimators=[('GradientBoostingClassifier',
                               GradientBoostingClassifier()),
                              ('SVC', SVC()),
                              ('HistGradientBoostingClassifier',
                               HistGradientBoostingClassifier()),
                              ('RandomForestClassifier',
                               RandomForestClassifier(max_depth=8,
                                                      max_features='log2',
                                                      min_samples_leaf=13,
                                                      min_samples_split=3,
                                                      n_estimators=20,
                                                      random_state=0)),
                              ('SGDClassifier', SGDClass...
                                             colsample_bytree=1, gamma=0,
                                             gpu_id=-1, importance_type='gain',
                                             interaction_constraints='',
                                             learning_rate=0.01,
                                             max_delta_step=0, max_depth=2,
                                             min_child_weight=1, missing=nan,
                                             monotone_constraints='()',
                                             n_estimators=20, n_jobs=-1,
                                             num_parallel_tree=1, random_state=0,
                                             reg_alpha=0, reg_lambda=1,
                                             scale_pos_weight=1, subsample=1,
                                             tree_method='exact',
                                             validate_parameters=1,
                                             verbosity=None))])),
'test_score': array([0.79464286, 0.73873874]),
'train_score': array([0.81981982, 0.88392857])}
```

Features importance

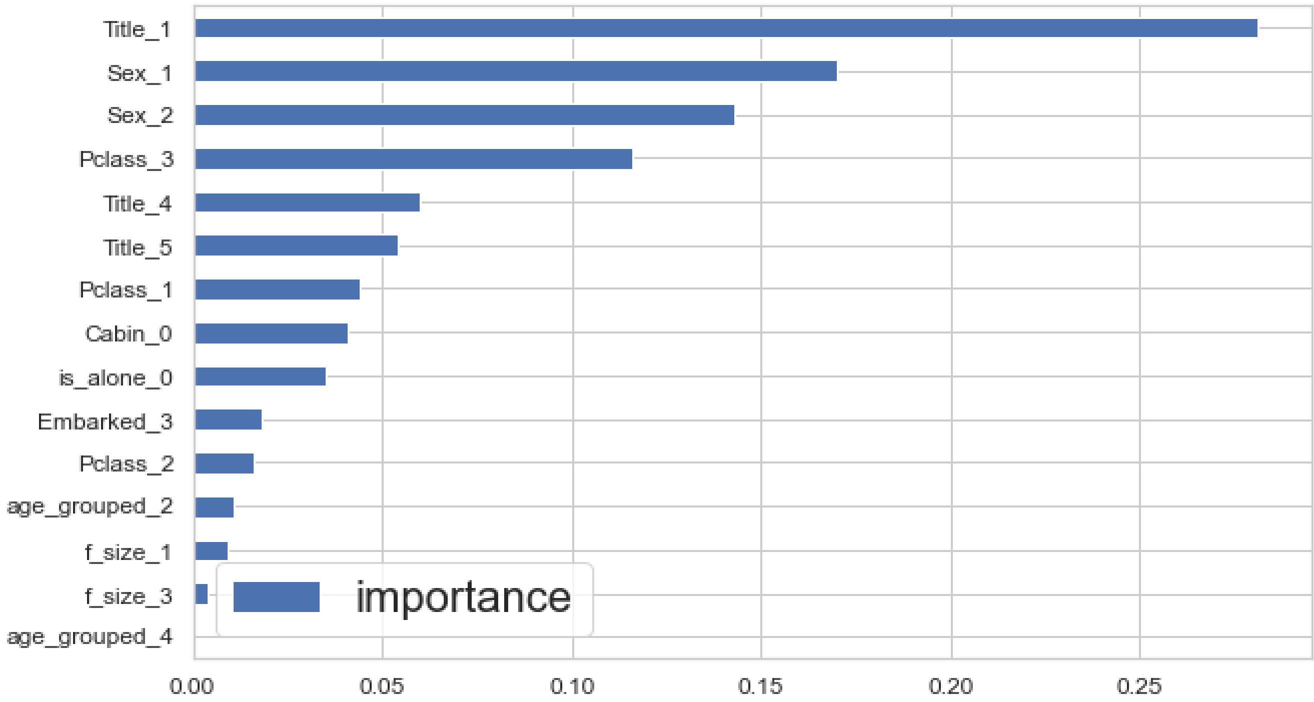
In [95]:

```
imp = pd.DataFrame(RFC_model.feature_importances_.round(3), index=X_train0.columns, columns=['importance'])
print(imp.sort_values('importance', ascending = False).to_markdown())
imp.sort_values('importance').plot(kind='barh', figsize=(10, 6), fontsize = 12)
```

```
|                |   importance |
|:---------------|-------------:|
```

```
| Title_1        |        0.281 |
| Sex_1          |        0.17  |
| Sex_2          |        0.143 |
| Pclass_3       |        0.116 |
| Title_4        |        0.06  |
| Title_5        |        0.054 |
| Pclass_1       |        0.044 |
| Cabin_0        |        0.041 |
| is_alone_0     |        0.035 |
| Embarked_3     |        0.018 |
| Pclass_2       |        0.016 |
| age_grouped_2  |        0.011 |
| f_size_1       |        0.009 |
| f_size_3       |        0.004 |
| age_grouped_4  |        0     |
```

Out[95]: <AxesSubplot:>



In [96]:
```
best_feachers = sorted(list(imp.sort_values('importance', ascending = False)[0:15].index))
```

In [97]:
```
#lets take best 15 features for next try
best_feachers
```

Out[97]:
```
['Cabin_0',
 'Embarked_3',
 'Pclass_1',
 'Pclass_2',
 'Pclass_3',
 'Sex_1',
 'Sex_2',
 'Title_1',
 'Title_4',
 'Title_5',
 'age_grouped_2',
 'age_grouped_4',
 'f_size_1',
 'f_size_3',
 'is_alone_0']
```

set Next_try

In [98]:
```
submission = pd.DataFrame(columns = ['PassengerId', 'Survived'])
```

In [99]:
```
for i in range(len(Y_pred_VT1)):
    submission.loc[i, 'PassengerId'] = i+892
    submission.loc[i, 'Survived'] = int(round(Y_pred_VT1[i]))
submission = submission.astype(int)
```

In [100…
```
submission[submission.Survived == 1].count()
```

Out[100…
```
PassengerId    147
Survived       147
dtype: int64
```

In [101…
```
# submission.to_csv ('/kaggle/working/submission.csv', columns =['PassengerId', 'Survived'], index=False, encoding = 'utf-8')
# submission.to_csv ('submission.csv', columns =['PassengerId', 'Survived'], index=False, encoding = 'utf-8')
```

In [ ]:

In [102…
```
# we can train different models separatelly to take a look how its work
```

In [103…
```
model = LogisticRegression(random_state=1, solver='liblinear', penalty = 'l1').fit(X_train, y_train)
Y_pred_LR = model.predict(X_test).round()
LR = round(metrics.accuracy_score(y_test, Y_pred_LR), 5)
print(LR)
```

```
0.78475
```

In [104…

```
model = ElasticNetCV(cv=5, random_state=0).fit(X_train, y_train)
Y_pred_EN = model.predict(X_test).round(0)
EN = round(metrics.accuracy_score(y_test, Y_pred_EN), 5)
print(EN)
```

```
0.76233
```

In [105…

```
model = OrthogonalMatchingPursuit().fit(X_train, y_train)
Y_pred_OMP = model.predict(X_test).round(0)
OMP = round(metrics.accuracy_score(y_test, Y_pred_OMP), 5)
print(OMP)
```

```
0.75785
```

In [106…

```
model = LinearDiscriminantAnalysis().fit(X_train, y_train)
Y_pred_LDA = model.predict(X_test)
LDA = round(metrics.accuracy_score(y_test, Y_pred_LDA), 5)
print(LDA)
```

```
0.78475
```

In [107…

```
model = QuadraticDiscriminantAnalysis().fit(X_train, y_train)
Y_pred_QDA = model.predict(X_test)
QDA = round(metrics.accuracy_score(y_test, Y_pred_QDA), 5)
print(QDA)
```

```
0.69955
```

In [108…

```
model = RidgeClassifier(random_state=0).fit(X_train, y_train)
Y_pred_RC = model.predict(X_test)
RC = round(metrics.accuracy_score(y_test, Y_pred_RC), 5)
print(RC)
```

```
0.78475
```

In [109…

```
model = KNeighborsClassifier(n_neighbors = 4, weights = 'uniform', algorithm = 'brute').fit(X_train, y_train)
Y_pred_KNN = model.predict(X_test)
KNN = round(metrics.accuracy_score(y_test, Y_pred_KNN), 5)
print(KNN)
```

```
0.79372
```

In [110…

```
model = GaussianNB().fit(X_train, y_train)
Y_pred_GNB = model.predict(X_test)
```

```
GNB = round(metrics.accuracy_score(y_test, Y_pred_GNB), 5)
print(GNB)
```

0.78924

In [111…
```
model = Perceptron(random_state=0, penalty = 'l2', max_iter = 5000).fit(X_train, y_train)
Y_pred_PCT = model.predict(X_test)
PCT = round(metrics.accuracy_score(y_test, Y_pred_PCT), 5)
print(PCT)
```

0.78475

In [112…
```
model = LinearSVC(random_state=0, max_iter = 10000).fit(X_train, y_train)
Y_pred_LSVC = model.predict(X_test)
LSVC = round(metrics.accuracy_score(y_test, Y_pred_LSVC), 5)
print(LSVC)
```

0.78475

In [113…
```
model = SGDClassifier().fit(X_train, y_train)
Y_pred_SGD = model.predict(X_test)
SGD = round(metrics.accuracy_score(y_test, Y_pred_SGD), 5)
print(SGD)
```

0.77578

In [114…
```
model = DecisionTreeClassifier(max_depth = 3, min_samples_leaf = 1, min_samples_split = 2).fit(X_train, y_train)
Y_pred_DTC = model.predict(X_test)
DTC = round(metrics.accuracy_score(y_test, Y_pred_DTC), 5)
print(DTC)
```

0.78475

In [115…
```
model = MLPClassifier(solver='lbfgs', alpha=1e-5, hidden_layer_sizes=(4, 3),
                   random_state=0, max_iter = 10000, learning_rate = 'adaptive').fit(X_train, y_train)
Y_pred_MLPC = model.predict(X_test)
MLPC = round(metrics.accuracy_score(y_test, Y_pred_MLPC), 5)
print(MLPC)
```

0.78475

In [116…
```
model = XGBClassifier(random_state=0, learning_rate = 0.01,
                              max_depth = 2, n_estimators = 20, n_jobs=-1).fit(X_train, y_train)
Y_pred_XGBC = model.predict(X_test)
XGBC = round(metrics.accuracy_score(y_test, Y_pred_XGBC), 5)
print(XGBC)
```

```
0.77578
```

In [117…

```
model = LGBMClassifier(random_state=0, learning_rate = 0.01, num_leaves = 3, n_estimators = 100,
                                 class_weight='balanced').fit(X_train, y_train)
Y_pred_LGBM = model.predict(X_test)
LGBM = round(metrics.accuracy_score(y_test, Y_pred_LGBM), 5)
print(LGBM)
```

```
0.75785
```

In [118…

```
model = CatBoostClassifier(random_state=0, learning_rate = 0.01, l2_leaf_reg = 5,
                                     depth = 5, iterations = 10, verbose=False).fit(X_train, y_train)
Y_pred_CBC = model.predict(X_test)
CBC = round(metrics.accuracy_score(y_test, Y_pred_CBC), 5)
print(CBC)
```

```
0.76682
```

In [119…

```
model = RidgeClassifierCV(cv=5).fit(X_train, y_train)
Y_pred_RCCV = model.predict(X_test)
RCCV = round(metrics.accuracy_score(y_test, Y_pred_RCCV), 5)
print(RCCV)
```

```
0.78475
```

In [120…

```
model = AdaBoostClassifier().fit(X_train, y_train)
Y_pred_ABC = model.predict(X_test)
ABC = round(metrics.accuracy_score(y_test, Y_pred_ABC), 5)
print(ABC)
```

```
0.78475
```

In [121…

```
model = PassiveAggressiveClassifier().fit(X_train, y_train)
Y_pred_PAC = model.predict(X_test)
PAC = round(metrics.accuracy_score(y_test, Y_pred_PAC), 5)
print(PAC)
```

```
0.78924
```

In [122…

```
model = LogisticRegressionCV().fit(X_train, y_train)
Y_pred_LRCV = model.predict(X_test)
LRCV = round(metrics.accuracy_score(y_test, Y_pred_LRCV), 5)
print(LRCV)
```

```
0.78475
```

In [123…
```
model = ExtraTreeClassifier().fit(X_train, y_train)
Y_pred_ETC = model.predict(X_test)
ETC = round(metrics.accuracy_score(y_test, Y_pred_ETC), 5)
print(ETC)
```

```
0.78475
```

In [124…
```
model = LogisticRegressionCV(cv=5, random_state=0, dual=False,
                             penalty='l2', solver='lbfgs', max_iter=100,
                             n_jobs=-1).fit(X_train, y_train)
Y_pred_LRCV = model.predict(X_test)
LRCV = round(metrics.accuracy_score(y_test, Y_pred_LRCV), 5)
print(LRCV)
```

```
0.78475
```

In [125…
```
model = LassoLarsCV().fit(X_train, y_train)
Y_pred_LLCV = model.predict(X_test).round()
LLCV = round(metrics.accuracy_score(y_test, Y_pred_LLCV), 5)
print(LLCV)
```

```
0.75785
```

In [126…
```
model = HistGradientBoostingClassifier().fit(X_train, y_train)
Y_pred_HGBC = model.predict(X_test)
HGBC = round(metrics.accuracy_score(y_test, Y_pred_HGBC), 5)
print(HGBC)
```

```
0.81166
```

In [127…
```
model = ExtraTreesClassifier().fit(X_train, y_train)
Y_pred_ETC = model.predict(X_test)
ETC = round(metrics.accuracy_score(y_test, Y_pred_ETC), 5)
print(ETC)
```

```
0.79372
```

In [128…
```
model = GradientBoostingClassifier().fit(X_train, y_train)
Y_pred_GBC = model.predict(X_test)
GBC = round(metrics.accuracy_score(y_test, Y_pred_GBC), 5)
print(GBC)
```

```
0.82511
```

In [129…
```
model = NuSVC().fit(X_train, y_train)
Y_pred_NSVC = model.predict(X_test)
```

```
NSVC = round(metrics.accuracy_score(y_test, Y_pred_NSVC), 5)
print(NSVC)
```

0.80269

In [130…

```
model = LabelPropagation().fit(X_train, y_train)
Y_pred_LP = model.predict(X_test)
LP = round(metrics.accuracy_score(y_test, Y_pred_LP), 5)
print(LP)
```

0.78924

Begin

Voting