In [1]:

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python Docker image: https://github.com/kaggle/docke
r-python
# For example, here's several helpful packages to load

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list al
l files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets p
reserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved out
side of the current session
```

Titanic EDA and Prediction

Importing all necessary libraries

In [2]:

```
import pandas as pd
import numpy as np
import xgboost as xgb
from scipy import stats
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC, LinearSVC
from sklearn.tree import DecisionTreeClassifier
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.svm import SVR
from lightgbm import LGBMRegressor
from xgboost import XGBRegressor
from mlxtend.regressor import StackingRegressor
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import GridSearchCV
from sklearn.preprocessing import RobustScaler
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import Lasso, Ridge
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import accuracy_score
pd.pandas.set_option("display.max_columns",None)
print("all necessary libraries are imported")
```

In [3]:

```
train=pd.read_csv("../input/titanic/train.csv")
train.head()
```

Out[3]:

	PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	F
0	1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	7
2	3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	5
4	5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8

In [4]:

```
test=pd.read_csv("../input/titanic/test.csv")
test.head()
```

Out[4]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	N
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	N
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	N
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	N
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	N

In [5]:

```
train.shape,test.shape
```

Out[5]:

In [6]:

```
train_cat=list(train.select_dtypes(include='object'))
train_cat
```

Out[6]:

In [7]:

```
train_num=list(train.select_dtypes(exclude='object'))
train_num
```

Out[7]:

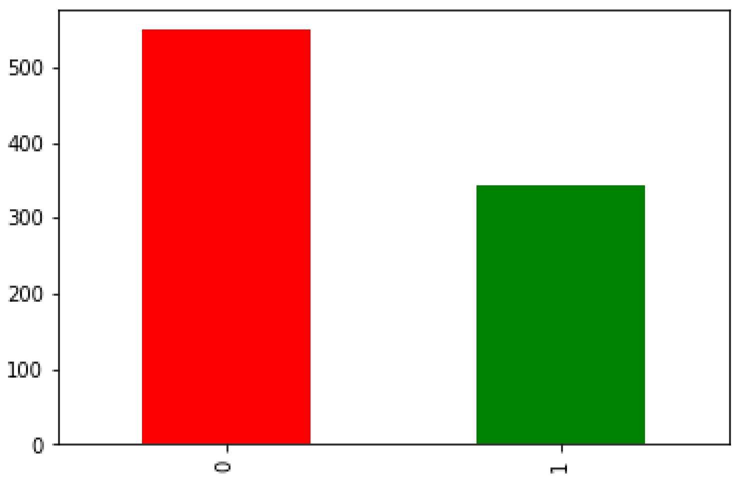
Univariate Analysis

Target Feature

In [8]:

```
train['Survived'].value_counts().plot.bar(color=['r','g'])
```

Out[8]:



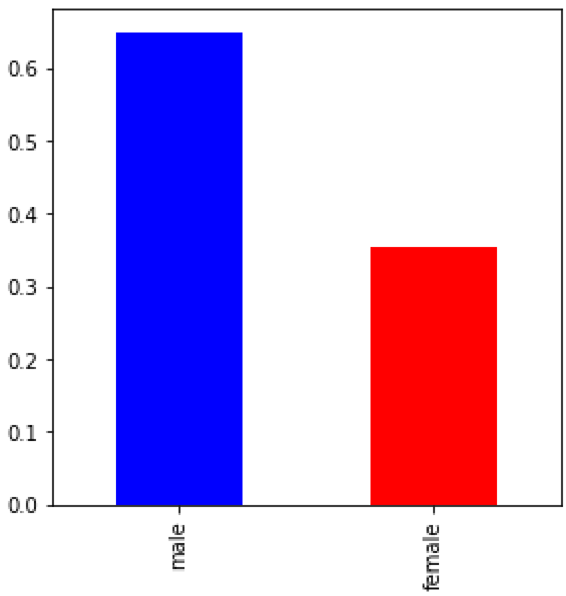
it says that most of the people have dead

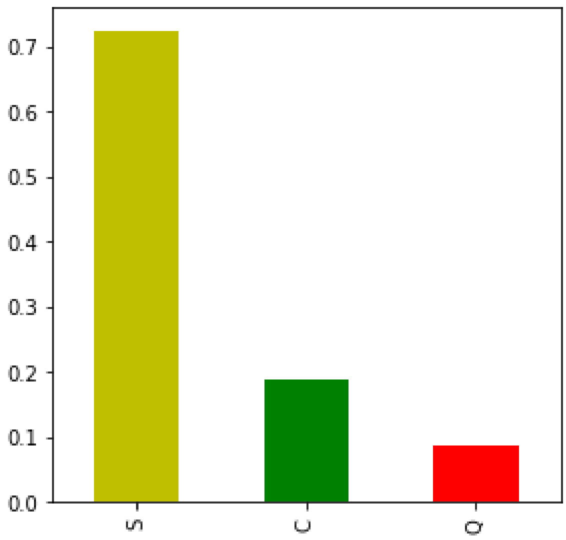
Categorical Feature

In [9]:

```
plt.figure(1)
plt.subplot(221)
train['Sex'].value_counts(normalize=True).plot.bar(figsize=(10,10),color=['b',
'r'])
plt.subplot(223)
train['Embarked'].value_counts(normalize=True).plot.bar(figsize=(10,10),color=[
'y','g','r'])
```

Out[9]:





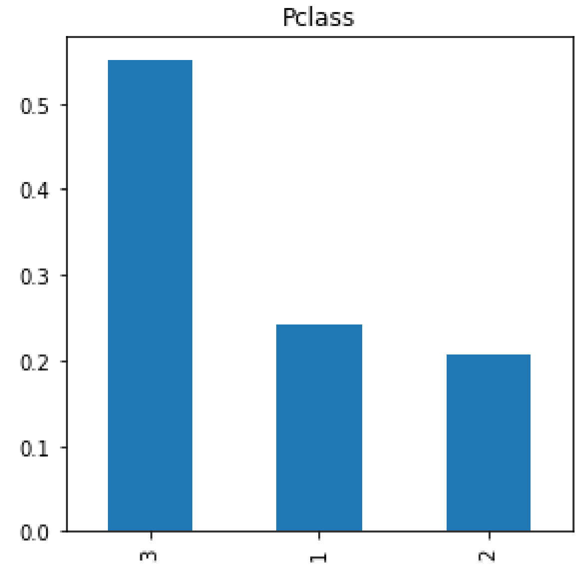
- Around 80% of the passangers are male
- Around 75% of passangers have embarked as S

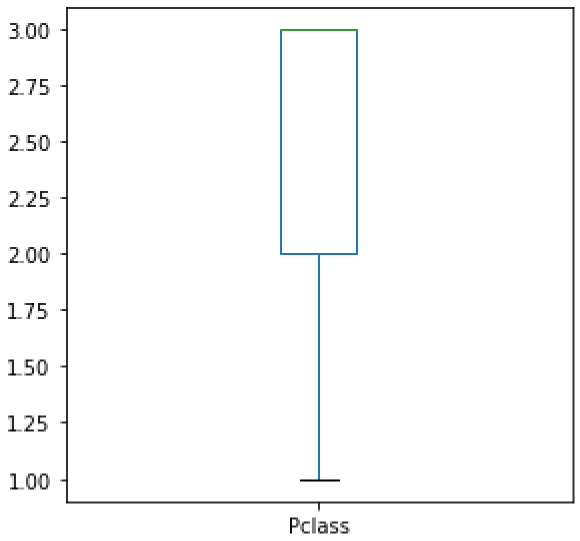
Numerical Feature

In [10]:

```
plt.figure(1)
plt.subplot(221)
train['Pclass'].value_counts(normalize=True).plot.bar(figsize=(10,10),title="Pcl
ass")
plt.subplot(223)
train['Pclass'].plot.box(figsize=(10,10))
```

Out[10]:

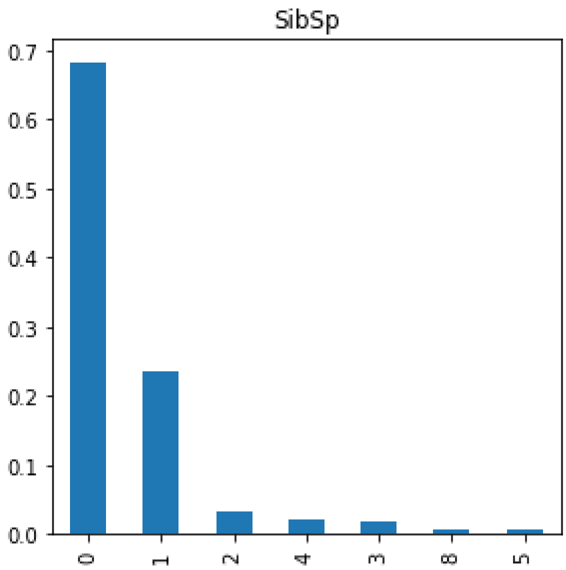

Pclass
0.0
0.1
0.2
0.3
0.4
0.5
3
1
2

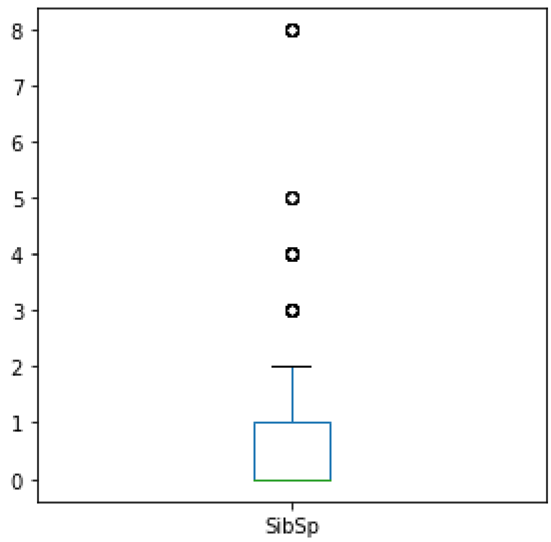

1.00
1.25
1.50
1.75
2.00
2.25
2.50
2.75
3.00
Pclass

In [11]:

```
plt.figure(1)
plt.subplot(221)
train['SibSp'].value_counts(normalize=True).plot.bar(figsize=(10,10),title="SibS
p")
plt.subplot(223)
train['SibSp'].plot.box(figsize=(10,10))
```

Out[11]:

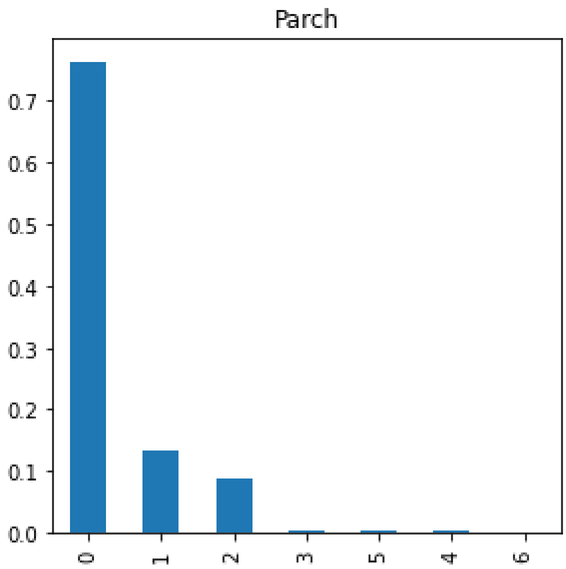


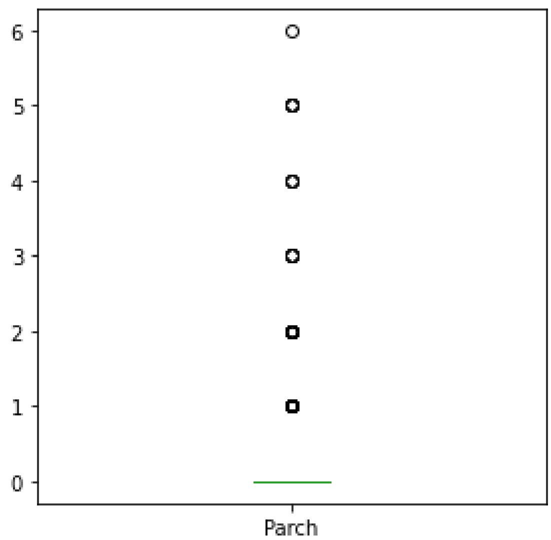


In [12]:

```
plt.figure(1)
plt.subplot(221)
train['Parch'].value_counts(normalize=True).plot.bar(figsize=(10,10),title="Parc
h")
plt.subplot(223)
train['Parch'].plot.box(figsize=(10,10))
```

Out[12]:





Bivariate Analysis

Categorical vs Target Feature

In [13]:

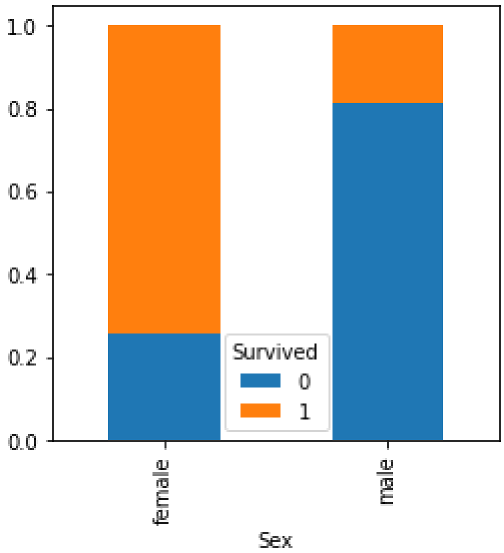
```
train_cat
```

Out[13]:

In [14]:

```
sex=pd.crosstab(train['Sex'],train['Survived'])
sex.div(sex.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,figsize=(4
,4))
```

Out[14]:

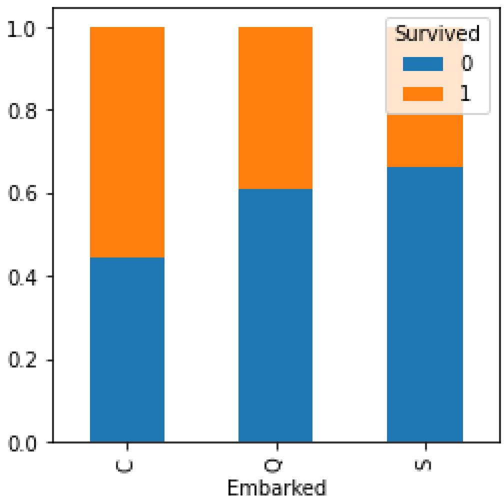


More males have dead comapre to females as if 1 shows alive and 0 shows death

In [15]:

```
embarked=pd.crosstab(train['Embarked'],train['Survived'])
embarked.div(embarked.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,
figsize=(4,4))
```

Out[15]:



The survival rate of embarked 'C' is greater than the rest two and the lowest survival rate is associated with embarked 'S'

Numerical vs Target Feature

In [16]:

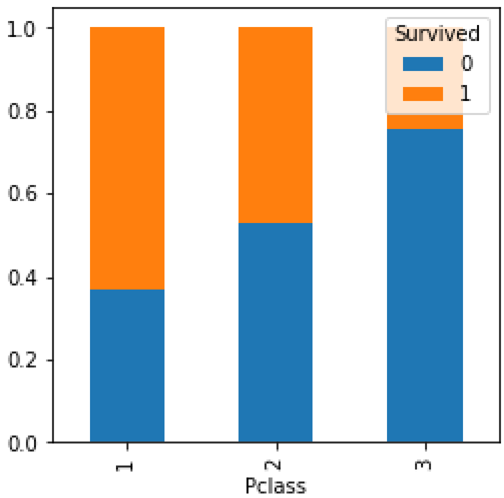
```
train_num
```

Out[16]:

In [17]:

```
pclass=pd.crosstab(train['Pclass'],train['Survived'])
pclass.div(pclass.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,figsize=(4,4))
```

Out[17]:

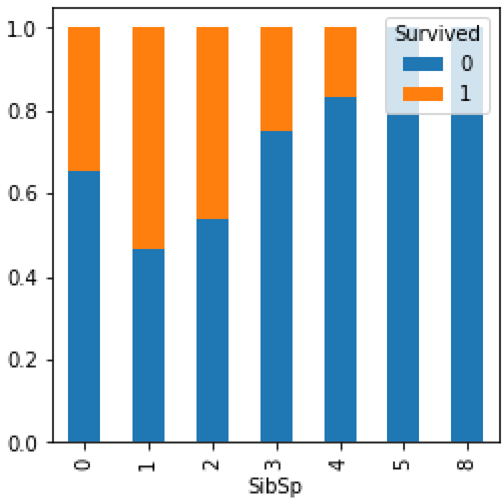


class 1 has most survival rate and class 3 has least survival rate

In [18]:

```
sibsp=pd.crosstab(train['SibSp'],train['Survived'])
sibsp.div(sibsp.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,figsiz
e=(4,4))
```

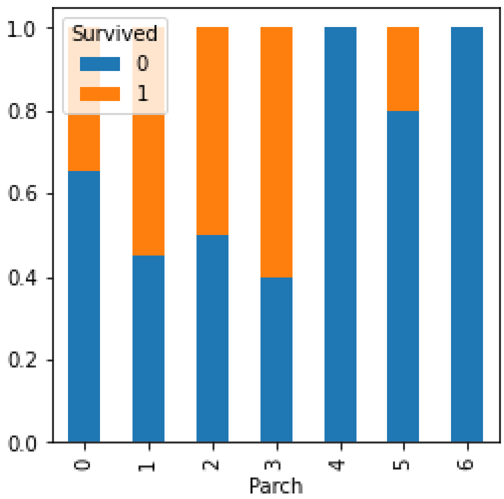
Out[18]:



In [19]:

```
parch=pd.crosstab(train['Parch'],train['Survived'])
parch.div(parch.sum(1).astype(float),axis=0).plot(kind='bar',stacked=True,figsiz
e=(4,4))
```

Out[19]:



In [20]:

```
train.isnull().any()
```

Out[20]:

In [21]:

```
train_missing_obj_col=[]
for col in train_cat:
    if train[col].isnull().any():
        train_missing_obj_col.append(col)
train_missing_obj_col
```

Out[21]:

Missing Values Imputation In Train Dataset

Imputation In Numerical Col

In [22]:

```
train_missing_num_col=[]
for col in train_num:
    if train[col].isnull().any():
        train_missing_num_col.append(col)
train_missing_num_col
```

Out[22]:

In [23]:

```
temp=train[['Age']]
temp
```

Out[23]:

	Age
0	22.0
1	38.0
2	26.0
3	35.0
4	35.0
...	...
886	27.0
887	19.0
888	NaN
889	26.0
890	32.0

891 rows × 1 columns

In [24]:

```
train.drop(['Age'],inplace=True,axis=1)
```

In [25]:

```
from sklearn.impute import SimpleImputer
my_imputer=SimpleImputer()
imputed_temp = pd.DataFrame(my_imputer.fit_transform(temp))
imputed_temp.columns = temp.columns
```

In [26]:

```
train=pd.concat([train,imputed_temp],axis=1)
```

In [27]:

```
train
```

Out[27]:

	PassengerId	Survived	Pclass	Name	Sex	SibSp	Parch	Ticket	Fare
0	1	0	3	Braund, Mr. Owen Harris	male	1	0	A/5 21171	7.250
1	2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	1	0	PC 17599	71.28
2	3	1	3	Heikkinen, Miss. Laina	female	0	0	STON/O2. 3101282	7.925
3	4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	1	0	113803	53.10
4	5	0	3	Allen, Mr. William Henry	male	0	0	373450	8.050
...	...	...	...	...	...	...	...	...	...
886	887	0	2	Montvila, Rev. Juozas	male	0	0	211536	13.00
887	888	1	1	Graham, Miss. Margaret Edith	female	0	0	112053	30.00
888	889	0	3	Johnston, Miss. Catherine Helen "Carrie"	female	1	2	W./C. 6607	23.45
889	890	1	1	Behr, Mr. Karl Howell	male	0	0	111369	30.00
890	891	0	3	Dooley, Mr. Patrick	male	0	0	370376	7.750

891 rows × 12 columns

Imputation In Categorical Col

In [28]:

```
train['Embarked'].fillna(train['Embarked'].mode(),inplace=True)
```

In [29]:

```
dummy1=pd.get_dummies(train[['Sex','Embarked']],drop_first=True)
dummy1
```

Out[29]:

	Sex_male	Embarked_Q	Embarked_S
0	1	0	1
1	0	0	0
2	0	0	1
3	0	0	1
4	1	0	1
...	...	...	...
886	1	0	1
887	0	0	1
888	0	0	1
889	1	0	0
890	1	1	0

891 rows × 3 columns

In [30]:

```
train.drop(train_missing_obj_col ,axis=1,inplace=True)
```

In [31]:

```
train=pd.concat([train,dummy1],axis=1)
```

In [32]:

```
train.drop(['Name','PassengerId','Sex','Ticket'],axis=1,inplace=True)
```

In [33]:

```
train
```

Out[33]:

	Survived	Pclass	SibSp	Parch	Fare	Age	Sex_male	Embarked_Q	Em
0	0	3	1	0	7.2500	22.000000	1	0	1
1	1	1	1	0	71.2833	38.000000	0	0	0
2	1	3	0	0	7.9250	26.000000	0	0	1
3	1	1	1	0	53.1000	35.000000	0	0	1
4	0	3	0	0	8.0500	35.000000	1	0	1
...	...	...	...	...	...	...	...	...	...
886	0	2	0	0	13.0000	27.000000	1	0	1
887	1	1	0	0	30.0000	19.000000	0	0	1
888	0	3	1	2	23.4500	29.699118	0	0	1
889	1	1	0	0	30.0000	26.000000	1	0	0
890	0	3	0	0	7.7500	32.000000	1	1	0

891 rows × 9 columns

In [34]:

```
train.shape
```

Out[34]:

Removing the Skewness In the Train Data Col

In [35]:

```
train['Age']=np.log(train['Age']+1)
```

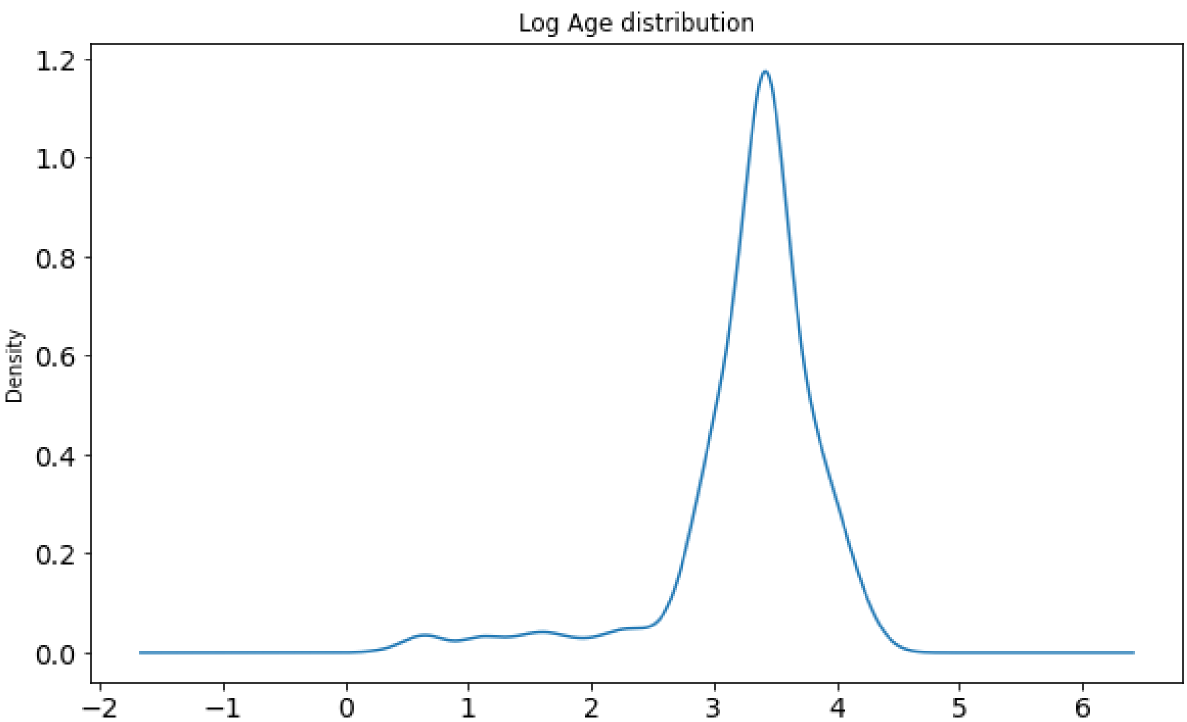
In [36]:

```
train['Fare']=np.log(train['Fare']+1)
```

In [37]:

```
 (train['Age']).plot(kind = 'density', title = 'Log Age distribution', fontsize=
14, figsize=(10, 6))
```

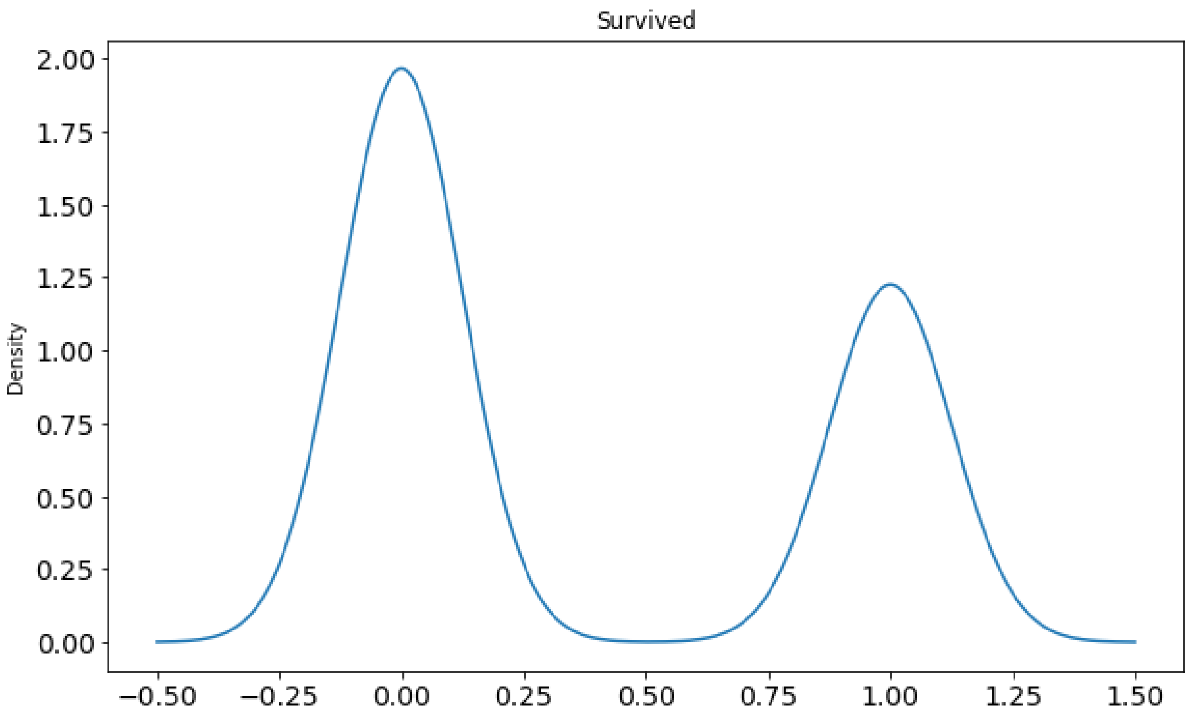
Out[37]:



In [38]:

```
(train['Survived']).plot(kind = 'density', title = 'Survived', fontsize=14, fig
size=(10, 6))
```

Out[38]:



Feature Engineering on Test Dataset

In [39]:

```
test.head()
```

Out[39]:

	PassengerId	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	C
0	892	3	Kelly, Mr. James	male	34.5	0	0	330911	7.8292	N
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female	47.0	1	0	363272	7.0000	N
2	894	2	Myles, Mr. Thomas Francis	male	62.0	0	0	240276	9.6875	N
3	895	3	Wirz, Mr. Albert	male	27.0	0	0	315154	8.6625	N
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female	22.0	1	1	3101298	12.2875	N

In [40]:

```
test_cat_col=list(test.select_dtypes(include='object'))
test_cat_col
```

Out[40]:

In [41]:

```
test_num_col=list(test.select_dtypes(exclude='object'))
test_num_col
```

Out[41]:

In [42]:

```
test_missing_cat_col=[col for col in test_cat_col if test[col].isnull().any()]
test_missing_cat_col
```

Out[42]:

In [43]:

```
test_missing_num_col=[col for col in test_num_col if test[col].isnull().any()]
test_missing_num_col
```

Out[43]:

Missing Values Imputation In Test Dataset

Imputation In Numerical Col

In [44]:

```
temp1=test[['Age','Fare']]
```

In [45]:

```
imputer=SimpleImputer()
imputed_temp1=pd.DataFrame(imputer.fit_transform(temp1))
imputed_temp1.columns=temp1.columns
```

In [46]:

```
test.drop(['Age','Cabin','Name','Fare'],axis=1,inplace=True)
```

In [47]:

```
test=pd.concat([test,imputed_temp1],axis=1)
test
```

Out[47]:

	PassengerId	Pclass	Sex	SibSp	Parch	Ticket	Embarked	Age	Fa
0	892	3	male	0	0	330911	Q	34.50000	7.
1	893	3	female	1	0	363272	S	47.00000	7.
2	894	2	male	0	0	240276	Q	62.00000	9.
3	895	3	male	0	0	315154	S	27.00000	8.
4	896	3	female	1	1	3101298	S	22.00000	12
...	...	...	...	...	...	...	...	...	...
413	1305	3	male	0	0	A.5. 3236	S	30.27259	8.
414	1306	1	female	0	0	PC 17758	C	39.00000	10
415	1307	3	male	0	0	SOTON/O.Q. 3101262	S	38.50000	7.
416	1308	3	male	0	0	359309	S	30.27259	8.
417	1309	3	male	1	1	2668	C	30.27259	22

418 rows × 9 columns

Imputation In Categorical Col

In [48]:

```
dummy2=pd.get_dummies(test[['Sex','Embarked']],drop_first=True)
dummy2
```

Out[48]:

	Sex_male	Embarked_Q	Embarked_S
0	1	1	0
1	0	0	1
2	1	1	0
3	1	0	1
4	0	0	1
...	...	...	...
413	1	0	1
414	0	0	0
415	1	0	1
416	1	0	1
417	1	0	0

418 rows × 3 columns

In [49]:

```
test.drop(['PassengerId','Sex','Embarked','Ticket'],axis=1,inplace=True)
```

In [50]:

```
test=pd.concat([test,dummy2],axis=1)
test
```

Out[50]:

	Pclass	SibSp	Parch	Age	Fare	Sex_male	Embarked_Q	Embarked_S
0	3	0	0	34.50000	7.8292	1	1	0
1	3	1	0	47.00000	7.0000	0	0	1
2	2	0	0	62.00000	9.6875	1	1	0
3	3	0	0	27.00000	8.6625	1	0	1
4	3	1	1	22.00000	12.2875	0	0	1
...	...	...	...	...	...	...	...	...
413	3	0	0	30.27259	8.0500	1	0	1
414	1	0	0	39.00000	108.9000	0	0	0
415	3	0	0	38.50000	7.2500	1	0	1
416	3	0	0	30.27259	8.0500	1	0	1
417	3	1	1	30.27259	22.3583	1	0	0

418 rows × 8 columns

In [51]:

```
test.shape
```

Out[51]:

Removing the Skewness In the Test Data Col

In [52]:

```
test['Age']=np.log(test['Age'])
```

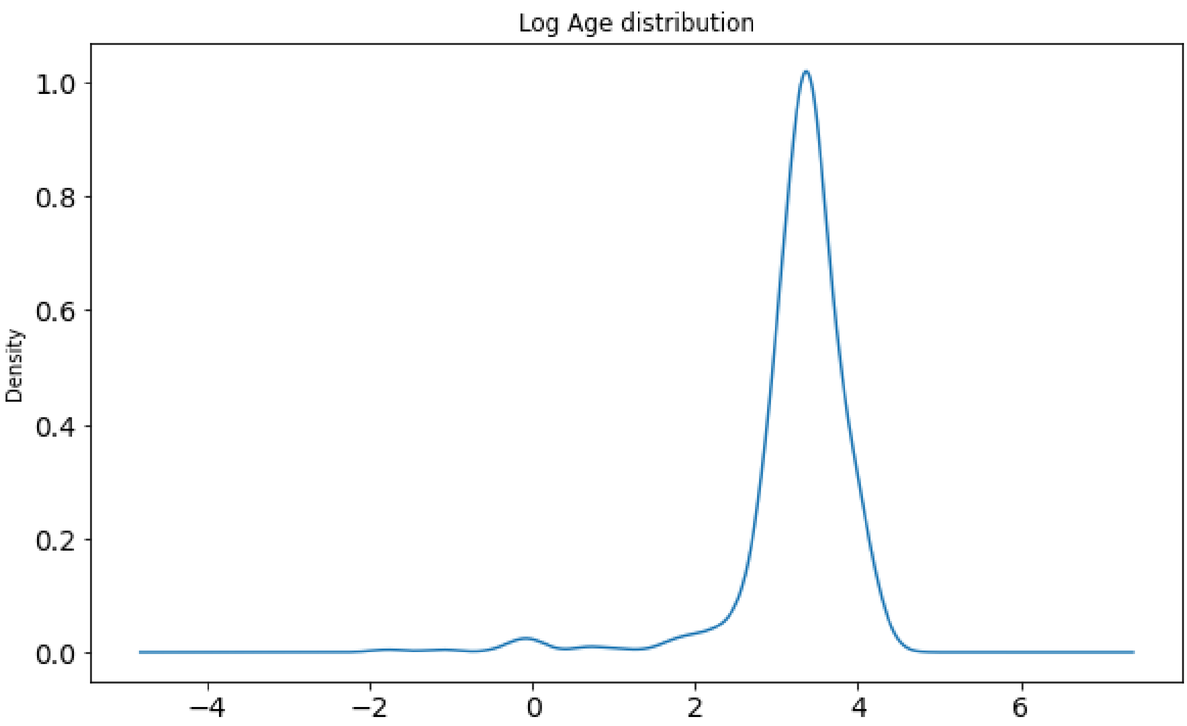
In [53]:

```
test['Fare']=np.log(test['Fare']+1)
```

In [54]:

```
 (test['Age']).plot(kind = 'density', title = 'Log Age distribution', fontsize=1
4, figsize=(10, 6))
```

Out[54]:



Removing Target Variable from the Train Dataset

In [55]:

```
y=train['Survived']
```

In [56]:

```
train.drop(['Survived'],axis=1,inplace=True)
```

In [57]:

```
train.head()
```

Out[57]:

	Pclass	SibSp	Parch	Fare	Age	Sex_male	Embarked_Q	Embarked_S
0	3	1	0	2.110213	3.135494	1	0	1
1	1	1	0	4.280593	3.663562	0	0	0
2	3	0	0	2.188856	3.295837	0	0	1
3	1	1	0	3.990834	3.583519	0	0	1
4	3	0	0	2.202765	3.583519	1	0	1

In [58]:

```
test.head()
```

Out[58]:

	Pclass	SibSp	Parch	Age	Fare	Sex_male	Embarked_Q	Embarked_S
0	3	0	0	3.540959	2.178064	1	1	0
1	3	1	0	3.850148	2.079442	0	0	1
2	2	0	0	4.127134	2.369075	1	1	0
3	3	0	0	3.295837	2.268252	1	0	1
4	3	1	1	3.091042	2.586824	0	0	1

In [59]:

```
train.shape,test.shape
```

Out[59]:

Model Building

Logistic Regression

In [60]:

```
x_train,x_test,y_train,y_test=train_test_split(train,y,random_state=42)
```

In [61]:

```
logistic=LogisticRegression(max_iter=100,random_state=1,n_jobs=-1)
logistic.fit(x_train,y_train)
pred1=logistic.predict(x_test)
pred1
```

Out[61]:

In [62]:

```
print(f'The train accuracy is {logistic.score(x_train,y_train)}')
from sklearn.metrics import accuracy_score
print('Logistic Regresson model accuracy score: {0:0.4f}'. format(accuracy_score
(y_test, pred1)))
```

Random Forest Classifier

Hyperparameter tuning

In [63]:

```
# param_grid = { "criterion" : ["gini","entropy"], "min_samples_leaf" : [1, 5, 10,
25, 50, 70],
#               "min_samples_split" : [2, 4, 10, 12, 16, 18, 25, 35], "n_estimator
s": [100, 400, 700, 1000, 1500]}
# rf = RandomForestClassifier(n_estimators=100, max_features='auto', oob_score=Tru
e, random_state=1, n_jobs=-1)
# clf = GridSearchCV(estimator=rf, param_grid=param_grid, n_jobs=-1)
# clf.fit(x_train, y_train)
# clf.best_params_
```

Now buliding the model with best parameters

In [64]:

```
random_forest = RandomForestClassifier(criterion = "entropy",
                                       min_samples_leaf = 1,
                                       min_samples_split = 12,
                                       n_estimators=100,
                                       max_features='auto',
                                       oob_score=True,
                                       random_state=42,
                                       n_jobs=-1)

random_forest.fit(x_train, y_train)
pred2 = random_forest.predict(x_test)

random_forest.score(x_train, y_train)

print("oob score:", round(random_forest.oob_score_, 4)*100, "%")
```

In [65]:

```
print('Random Forest Classifier model accuracy score: {0:0.4f}'. format(accuracy
_score(y_test, pred2)))
```

Stochastic Gradient Descent (SGD)

In [66]:

```
sgd = SGDClassifier(random_state=42)
sgd.fit(x_train, y_train)
pred3 = sgd.predict(x_test)

sgd.score(x_train, y_train)

acc_sgd = round(sgd.score(x_train, y_train) * 100, 2)
acc_sgd
```

Out[66]:

In [67]:

```
print('Stochastic Gradient Descent (SGD) model accuracy score: {0:0.4f}'. format
(accuracy_score(y_test, pred3)))
```

Hyperparameter tuning

In [68]:

```
sgdc=SGDClassifier(random_state=0)

parameters = {
            'alpha':[0.00001, 0.0001, 0.001, 0.01, 0.1, 1],
             'loss':['hinge','log'], 'penalty':['l1','l2']}
searcher = GridSearchCV(sgdc, parameters, cv=10)
searcher.fit(x_train, y_train)
print(searcher.best_params_)
print(searcher.best_score_)
```

Now buliding the model with best parameters

In [69]:

```
sgd = SGDClassifier(random_state=0,alpha=0.01,loss='log',penalty='l2')
sgd.fit(x_train, y_train)
pred3 = sgd.predict(x_test)
pred3
```

Out[69]:

In [70]:

```
print('Stochastic Gradient Descent (SGD) tuned model accuracy score: {0:0.4f}'.
format(accuracy_score(y_test, pred3)))
```

K Nearest Neighbor (KNN)

In [71]:

```
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(x_train, y_train)
pred4 = knn.predict(x_test)
acc_knn = round(knn.score(x_train, y_train) * 100, 2)
print('Stochastic Gradient Descent (SGD) model train accuracy score: {0:0.4f}'.
format(acc_knn))
print('Stochastic Gradient Descent (SGD) model accuracy score: {0:0.4f}'. format
(accuracy_score(y_test, pred4)))
```

Linear Support Vector Machine

Linear Support Vector Machine

In [72]:

```
linear_svc = LinearSVC(max_iter=100000,dual=True)
linear_svc.fit(x_train, y_train)

pred5 = linear_svc.predict(x_test)

acc_linear_svc = round(linear_svc.score(x_train, y_train) * 100, 2)
print('Linear Support Vector Machine model train accuracy score: {0:0.4f}'. form
at(acc_linear_svc))
print('Linear Support Vector Machine model accuracy score: {0:0.4f}'. format(acc
uracy_score(y_test, pred5)))
```

Decision Tree

In [73]:

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(x_train, y_train)
pred6 = decision_tree.predict(x_test)
acc_decision_tree = round(decision_tree.score(x_train,y_train) * 100, 2)
print('Decision Tree model train accuracy score: {0:0.4f}'. format(acc_decision_tree))
print('Decision Tree model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, pred6)))
```

Hyperparameter tuning

In [74]:

```
grid_param = {
    'criterion' : ['gini', 'entropy'],
    'max_depth' : [3, 5, 7, 10],
    'splitter' : ['best', 'random'],
    'min_samples_leaf' : [1, 2, 3, 5, 7],
    'min_samples_split' : [1, 2, 3, 5, 7],
    'max_features' : ['auto', 'sqrt', 'log2']
}

decision = GridSearchCV(decision_tree, grid_param, cv = 5, n_jobs = -1, verbose
= 1)
decision.fit(x_train, y_train)
print(decision.best_params_)
print(decision.best_score_)
```

Now buliding the model with best parameters

In [75]:

```
decision_tree = DecisionTreeClassifier(criterion='entropy',
                                       max_depth=7,
                                       splitter='best',
                                       min_samples_leaf=5,
                                       min_samples_split=3,
                                       max_features='log2')
decision_tree.fit(x_train, y_train)
pred_6 = decision_tree.predict(x_test)
print('Decision Tree model accuracy score: {0:0.4f}'. format(accuracy_score(y_te
st, pred_6)))
```

Adaboost Classifier

In [76]:

```
from sklearn.ensemble import AdaBoostClassifier
adb = AdaBoostClassifier(base_estimator = decision_tree)
adb.fit(x_train,y_train)
adb_pred=adb.predict(x_test)
acc_adb=adb.score(x_train,y_train)
print('Adaboost Classifier model train accuracy score: {0:0.4f}'. format(acc_adb
))
print('Adaboost Classifier model accuracy score: {0:0.4f}'. format(accuracy_scor
e(y_test, adb_pred)))
```

Hyperparameter Tuning

In [77]:

```
grid_param = {
    'n_estimators' : [100, 120, 150, 180, 200],
    'learning_rate' : [0.01, 0.1, 1, 10],
    'algorithm' : ['SAMME', 'SAMME.R']
}

ada = GridSearchCV(adb, grid_param, cv = 5, n_jobs = -1, verbose = True)
ada.fit(x_train, y_train)
print(ada.best_params_)
print(ada.best_score_)
```

In [78]:

```
adb = AdaBoostClassifier(n_estimators=200,
                         learning_rate=0.01,
                         algorithm='SAMME.R')
adb.fit(x_train,y_train)
pred7=adb.predict(x_test)
print('Adaboost Classifier model accuracy score: {0:0.4f}'. format(accuracy_scor
e(y_test, pred7)))
```

Gradient Boosting

In [79]:

```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier()
gb.fit(x_train, y_train)
pred8=gb.predict(x_test)
acc_gb=gb.score(x_train,y_train)
print('Gradient Boosting model train accuracy score: {0:0.4f}'. format(acc_gb))
print('Gradient Boosting model accuracy score: {0:0.4f}'. format(accuracy_score(
y_test, pred8)))
```

LightGBM

In [80]:

```
import lightgbm as lgb
lgbm= lgb.LGBMClassifier()
lgbm.fit(x_train,y_train)
lgbm_pred=lgbm.predict(x_test)
acc_lgbm=lgbm.score(x_train,y_train)
print('lgbm model train accuracy score: {0:0.4f}'. format(acc_lgbm))
print('lgbm model accuracy score: {0:0.4f}'. format(accuracy_score(y_test,lgbm_p
red)))
```

Hyperparameter Tuning

In [81]:

```
fit_params={"early_stopping_rounds":30,
            "eval_metric" : 'auc',
            "eval_set" : [(x_test,y_test)],
            'eval_names': ['valid'],
            'verbose': 100,
            'categorical_feature': 'auto'}
```

In [82]:

```
from scipy.stats import randint as sp_randint
from scipy.stats import uniform as sp_uniform
param_test ={'num_leaves': sp_randint(6, 50),
             'min_child_samples': sp_randint(50, 100),
             'min_child_weight': [1e-5, 1e-3, 1e-2, 1e-1, 1, 1e1, 1e2, 1e3, 1e4
],
             'subsample': sp_uniform(loc=0.2, scale=0.8),
             'colsample_bytree': sp_uniform(loc=0.4, scale=0.6),
             'reg_alpha': [0, 1e-1, 1, 2, 5, 7, 10, 50, 100],
             'reg_lambda': [0, 1e-1, 1, 5, 10, 20, 50, 100]}
```

In [83]:

```
from sklearn.model_selection import RandomizedSearchCV
clf = lgb.LGBMClassifier(max_depth=-1, random_state=42, silent=True, metric='None', n_jobs=4, n_estimators=100)
gs = RandomizedSearchCV(
    estimator=clf, param_distributions=param_test,
    n_iter=100,
    scoring='roc_auc',
    cv=3,
    refit=True,
    random_state=314,
    verbose=True)
```

In [84]:

```
gs.fit(x_train, y_train, **fit_params)
print('Best score reached: {} with params: {} '.format(gs.best_score_, gs.best_params_))
```

In [85]:

```
opt_params={'colsample_bytree':  0.952164731370897, 'min_child_samples': 61,
            'min_child_weight': 0.01, 'num_leaves': 38, 'reg_alpha': 0,
            'reg_lambda': 0.1, 'subsample':0.3029313662262354}
```

In [86]:

```
clf_sw = lgb.LGBMClassifier(**clf.get_params())
#set optimal parameters
clf_sw.set_params(**opt_params)
```

Out[86]:

In [87]:

```
clf_sw.fit(x_train,y_train)
pred9=clf_sw.predict(x_test)
print('lgbm model accuracy score: {0:0.4f}'. format(accuracy_score(y_test,pred9
)))
```

CatBoost Classifier

In [88]:

```
from catboost import CatBoostClassifier

cat = CatBoostClassifier(iterations=10)
cat.fit(x_train, y_train)
pred10=cat.predict(x_test)
acc_cat=cat.score(x_train,y_train)
print('CatBoostClassifier model accuracy score: {0:0.4f}'. format(accuracy_score
(y_test,pred10)))
```

Voting Classifier

In [89]:

```
from sklearn.ensemble import VotingClassifier
classifiers = [('Gradient Boosting Classifier', gb), ('Stochastic Gradient Boost
ing', sgd),  ('Cat Boost Classifier', cat),
                ('Decision Tree', decision_tree), ('Light Gradient', clf_sw),
               ('Random Forest', random_forest), ('Ada Boost', adb), ('Logistic'
, logistic)]
vc = VotingClassifier(estimators = classifiers)
vc.fit(x_train, y_train)
voting_pred = vc.predict(x_test)
accuracy = accuracy_score(y_test,voting_pred)
print('Voting Classifier: {:.3f}'.format(accuracy))
```

In [90]:

```
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
print(f"{confusion_matrix(y_test, vc.predict(x_test))}\n")
print(classification_report(y_test, vc.predict(x_test)))
```

Important Features

In [91]:

```
importances = pd.DataFrame({'feature':train.columns,'importance':np.round(random_forest.feature_importances_,3)})
importances = importances.sort_values('importance',ascending=False).set_index('feature')
importances.head(15)
```

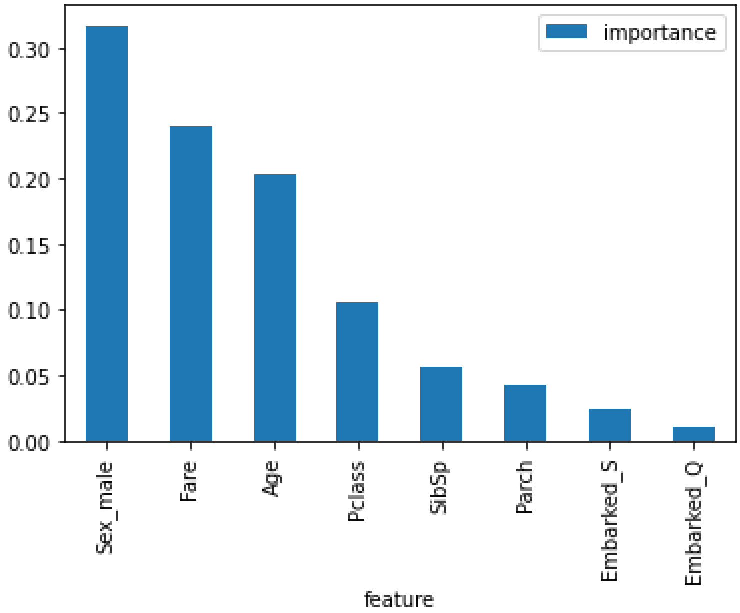
Out[91]:

	importance
feature	
Sex_male	0.317
Fare	0.240
Age	0.204
Pclass	0.106
SibSp	0.056
Parch	0.043
Embarked_S	0.024
Embarked_Q	0.010

In [92]:

```
importances.plot.bar()
```

Out[92]:



Model Comaprison

In [93]:

```
models = pd.DataFrame({
    'Model' : ['Logistic Regression','Random Forest Classifier','Stochastic Grad
ient Boosting', 'KNN','LSVM', 'Decision Tree Classifier','Ada Boost Classifier',
             'Gradient Boosting Classifier','Lightgbm', 'Cat Boost',  'Voting Cl
assifier'],
    'Score' : [accuracy_score(y_test,pred1), accuracy_score(y_test,pred2), accur
acy_score(y_test,pred3),
               accuracy_score(y_test,pred4),accuracy_score(y_test,pred5), accura
cy_score(y_test,pred_6),
               accuracy_score(y_test,pred7), accuracy_score(y_test,pred8), accur
acy_score(y_test,pred9)
               ,accuracy_score(y_test,pred10),accuracy]
})


models.sort_values(by = 'Score', ascending = False)
```

Out[93]:

	Model	Score
8	Lightgbm	0.834081
1	Random Forest Classifier	0.825112
10	Voting Classifier	0.820628
7	Gradient Boosting Classifier	0.816143
9	Cat Boost	0.816143
3	KNN	0.811659
0	Logistic Regression	0.807175
5	Decision Tree Classifier	0.802691
4	LSVM	0.798206
6	Ada Boost Classifier	0.784753
2	Stochastic Gradient Boosting	0.757848

Final Prediction

In [94]:

```
final_pred=vc.predict(test)
```

In [95]:

```
submission=pd.read_csv('../input/titanic/gender_submission.csv')
```

In [96]:

```
submission
```

Out[96]:

	PassengerId	Survived
0	892	0
1	893	1
2	894	0
3	895	0
4	896	1
...	...	...
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

418 rows × 2 columns

In [97]:

```
submission['Survived']=final_pred
```

In [98]:

```
submission
```

Out[98]:

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0
...	...	...
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

418 rows × 2 columns

In [99]:

```
pd.DataFrame(submission,columns=['PassengerId','Survived']).to_csv('titanic_new.csv',index=False)
```

In [100]:

```
pd.read_csv('titanic_new.csv')
```

Out[100]:

	PassengerId	Survived
0	892	0
1	893	0
2	894	0
3	895	0
4	896	0
...	...	...
413	1305	0
414	1306	1
415	1307	0
416	1308	0
417	1309	0

418 rows × 2 columns

In [ ]:

In [ ]: