Space X Falcon 9 First Stage Landing Prediction

Space X advertises Falcon 9 rocket launches on its website with a cost of 62 million dollars; other providers cost upward of 165 million dollars each, much of the savings is because Space X can reuse the first stage. Therefore if we can determine if the first stage will land, we can determine the cost of a launch. This information can be used if an alternate company wants to bid against space X for a rocket launch. In this lab, you will create a machine learning pipeline to predict if the first stage will land given the data from the preceding labs.

Several examples of an unsuccessful landing are shown here:

Most unsuccessful landings are planed. Space X; performs a controlled landing in the oceans.

Objectives

Perform exploratory Data Analysis and determine Training Labels

- create a column for the class
- Standardize the data
- Split into training data and test data

-Find best Hyperparameter for SVM, Classification Trees and Logistic Regression

- Find the method performs best using test data

Import Libraries and Define Auxiliary Functions

In [1]:
```
import piplite
```

```
await piplite.install(['numpy'])
await piplite.install(['pandas'])
await piplite.install(['seaborn'])
```

We will import the following libraries for the lab

In [2]:

```
# Pandas is a software library written for the Python programming language for data
import pandas as pd
# NumPy is a library for the Python programming language, adding support for large,
import numpy as np
# Matplotlib is a plotting library for python and pyplot gives us a MatLab like plo
import matplotlib.pyplot as plt
#Seaborn is a Python data visualization library based on matplotlib. It provides a
import seaborn as sns
# Preprocessing allows us to standarsize our data
from sklearn import preprocessing
# Allows us to split our data into training and testing data
from sklearn.model_selection import train_test_split
# Allows us to test parameters of classification algorithms and find the best one
from sklearn.model_selection import GridSearchCV
# Logistic Regression classification algorithm
from sklearn.linear_model import LogisticRegression
# Support Vector Machine classification algorithm
from sklearn.svm import SVC
# Decision Tree classification algorithm
from sklearn.tree import DecisionTreeClassifier
# K Nearest Neighbors classification algorithm
from sklearn.neighbors import KNeighborsClassifier
```

This function is to plot the confusion matrix.

In [4]:

```
def plot_confusion_matrix(y,y_predict):
    "this function plots the confusion matrix"
    from sklearn.metrics import confusion_matrix

    cm = confusion_matrix(y, y_predict)
    ax= plt.subplot()
    sns.heatmap(cm, annot=True, ax = ax); #annot=True to annotate cells
    ax.set_xlabel('Predicted labels')
    ax.set_ylabel('True labels')
    ax.set_title('Confusion Matrix');
    ax.xaxis.set_ticklabels(['did not land', 'land']); ax.yaxis.set_ticklabels(['di
    plt.show()
```

Load the dataframe

Load the data

In [5]:

```
from js import fetch
import io

URL1 = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0
resp1 = await fetch(URL1)
text1 = io.BytesIO((await resp1.arrayBuffer()).to_py())
data = pd.read_csv(text1)
```

In [6]:

```
data.head()
```

Out[6]:

	FlightNumber	Date	BoosterVersion	PayloadMass	Orbit	LaunchSite	Outcome	Flights	Gric
0	1	2010-06-04	Falcon 9	6104.959412	LEO	CCAFS SLC 40	None None	1	
1	2	2012-05-22	Falcon 9	525.000000	LEO	CCAFS SLC 40	None None	1	
2	3	2013-03-01	Falcon 9	677.000000	ISS	CCAFS SLC 40	None None	1	
3	4	2013-09-29	Falcon 9	500.000000	PO	VAFB SLC 4E	False Ocean	1	
4	5	2013-12-03	Falcon 9	3170.000000	GTO	CCAFS SLC 40	None None	1	

In [7]:
```
URL2 = 'https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBM-DS0:
resp2 = await fetch(URL2)
text2 = io.BytesIO((await resp2.arrayBuffer()).to_py())
X = pd.read_csv(text2)
```

In [8]:
```
X.head(100)
```

Out[8]:

	FlightNumber	PayloadMass	Flights	Block	ReusedCount	Orbit_ES-L1	Orbit_GEO	Orbit_GTO
0	1.0	6104.959412	1.0	1.0	0.0	0.0	0.0	0.0
1	2.0	525.000000	1.0	1.0	0.0	0.0	0.0	0.0
2	3.0	677.000000	1.0	1.0	0.0	0.0	0.0	0.0
3	4.0	500.000000	1.0	1.0	0.0	0.0	0.0	0.0
4	5.0	3170.000000	1.0	1.0	0.0	0.0	0.0	1.0
...	...	...	...	...	...	...	...	...
85	86.0	15400.000000	2.0	5.0	2.0	0.0	0.0	0.0
86	87.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0
87	88.0	15400.000000	6.0	5.0	5.0	0.0	0.0	0.0
88	89.0	15400.000000	3.0	5.0	2.0	0.0	0.0	0.0
89	90.0	3681.000000	1.0	5.0	0.0	0.0	0.0	0.0

90 rows × 83 columns

TASK 1

Create a NumPy array from the column `Class` in `data`, by applying the method `to_numpy()` then assign to the variable `Y`

In [10]:
```
Y = data['Class'].to_numpy()
```

In [13]:
```
type(Y)
```

Out[13]:
```
numpy.ndarray
```

TASK 2

Standardize the data in `X` then reassign it to the variable `X` using the transform provided below.

In [15]:
```
# students get this
transform = preprocessing.StandardScaler()
X = transform.fit_transform(X)
```

We split the data into training and testing data using the function `train_test_split`. The training data is divided into validation data, a second set used for training data; then the models are trained and hyperparameters are selected using the function `GridSearchCV`.

TASK 3

Use the function train_test_split to split the data X and Y into training and test data. Set the parameter test_size to 0.2 and random_state to 2. The training data and test data should be assigned to the following labels.

`X_train, X_test, Y_train, Y_test`

In [41]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
```

we can see we only have 18 test samples.

In [18]:
```
Y_test.shape
```

Out[18]:
```
(18,)
```

TASK 4

Create a logistic regression object then create a GridSearchCV object `logreg_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

In [19]:
```
parameters ={'C':[0.01,0.1,1],
             'penalty':['l2'],
             'solver':['lbfgs']}
```

In [33]:
```
parameters ={"C":[0.01,0.1,1],'penalty':['l2'], 'solver':['lbfgs']}# l1 lasso l2 r
lr=LogisticRegression()
logreg_cv = GridSearchCV(cv=10, estimator=lr, param_grid=parameters, refit=True)
```

We output the `GridSearchCV` object for logistic regression. We display the best parameters using the data attribute `best_params\_` and the accuracy on the validation data using the data attribute `best_score\_`.

In [37]:
```
logreg_cv.fit(X_train, Y_train)
```

Out[37]:
```
GridSearchCV(cv=10, estimator=LogisticRegression(),
             param_grid={'C': [0.01, 0.1, 1], 'penalty': ['l2'],
                         'solver': ['lbfgs']})
```

In [38]:
```
print("tuned hpyerparameters :(best parameters) ",logreg_cv.best_params_)
print("accuracy :",logreg_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 0.1, 'penalty': 'l2', 'solver': 'l
bfgs'}
accuracy : 0.8446428571428571
```

TASK 5

Calculate the accuracy on the test data using the method `score` :

In [47]:
```
# testing accuracy
logreg_cv.score(X_test, Y_test)
```

Out[47]:
```
0.9444444444444444
```

Lets look at the confusion matrix:

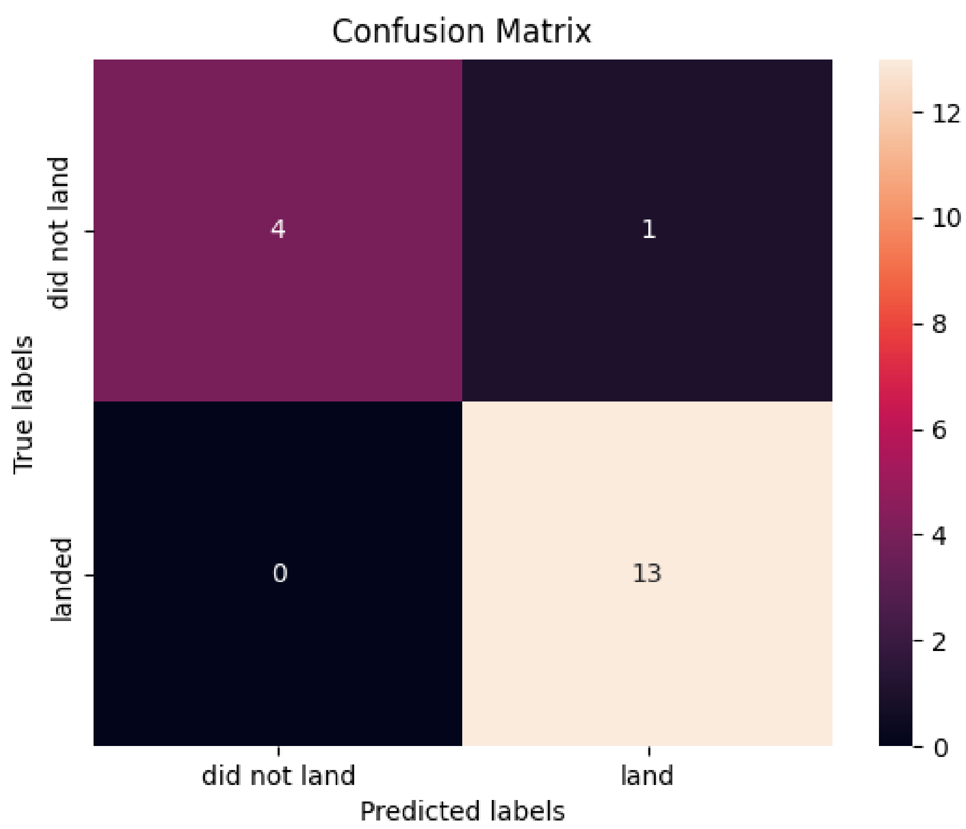
In [48]:
```
yhat
```

Out[48]:
```
array([1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1], dtype=int64)
```

In [49]:
```
Y_test
```

Out[49]:
```
array([1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1], dtype=int64)
```

In [45]:
```
yhat=logreg_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



In [ ]:
```
Examining the confusion matrix, we see that logistic regression can distinguish be
```

TASK 6

Create a support vector machine object then create a `GridSearchCV` object `svm_cv` with cv - 10. Fit the object to find the best parameters from the dictionary `parameters`.

In [52]:
```
parameters = {'kernel':('linear', 'rbf','poly','rbf', 'sigmoid'),
              'C': np.logspace(-3, 3, 5),
              'gamma':np.logspace(-3, 3, 5)}
svm = SVC()
```

In [58]:
```
# GridSearch(Model, ParametersToChooseFrom, CV-Cross validation-)
svm_cv = GridSearchCV(svm, parameters, cv=10)
```

In [62]:
```
svm_cv.fit(X_train, Y_train)
```

Out[62]:
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'C': array([1.00000000e-03, 3.16227766e-02, 1.00000000e+0
0, 3.16227766e+01,
       1.00000000e+03]),
                         'gamma': array([1.00000000e-03, 3.16227766e-02, 1.0000000
0e+00, 3.16227766e+01,
       1.00000000e+03]),
                         'kernel': ('linear', 'rbf', 'poly', 'rbf', 'sigmoid')})
```

In [57]:
```
print("tuned hpyerparameters :(best parameters) ",svm_cv.best_params_)
print("accuracy :",svm_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'C': 1.0, 'gamma': 0.03162277660168379,
'kernel': 'sigmoid'}
accuracy : 0.8464285714285713
```

TASK 7

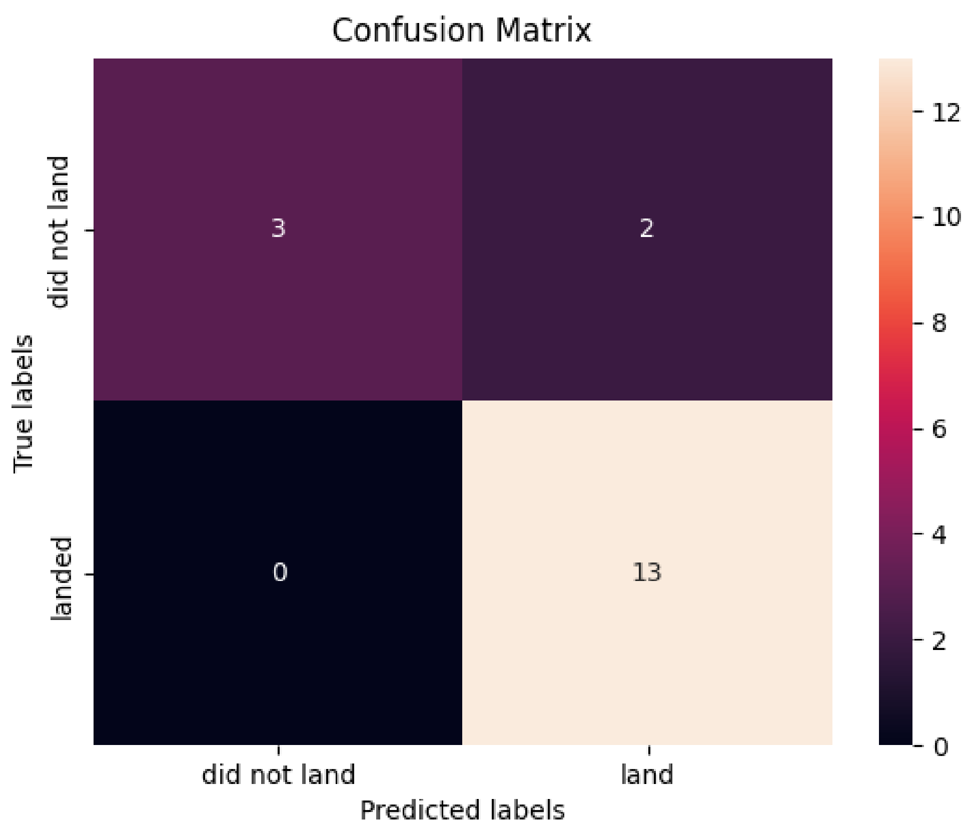
Calculate the accuracy on the test data using the method `score`:

In [65]:
```
####  R squared ranges from - infinity to One
svm_cv.score(X_test, Y_test)
```

Out[65]:
```
0.8888888888888888
```

We can plot the confusion matrix

In [66]:
```
yhat=svm_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 8

Create a decision tree classifier object then create a `GridSearchCV` object `tree_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters`.

In [67]:
```
parameters = {'criterion': ['gini', 'entropy'],
     'splitter': ['best', 'random'],
     'max_depth': [2*n for n in range(1,10)],
     'max_features': ['auto', 'sqrt'],
     'min_samples_leaf': [1, 2, 4],
     'min_samples_split': [2, 5, 10]}

tree = DecisionTreeClassifier()
```

In [68]:
```
tree_cv = GridSearchCV(estimator=tree, param_grid=parameters, cv=10)
```

In [69]:
```
tree_cv.fit(X_train, Y_train)
```

Out[69]:
```
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [2, 4, 6, 8, 10, 12, 14, 16, 18],
                         'max_features': ['auto', 'sqrt'],
                         'min_samples_leaf': [1, 2, 4],
                         'min_samples_split': [2, 5, 10],
                         'splitter': ['best', 'random']})
```

In [70]:
```
print("tuned hpyerparameters :(best parameters) ",tree_cv.best_params_)
print("accuracy :",tree_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'criterion': 'gini', 'max_depth': 4, 'm
ax_features': 'auto', 'min_samples_leaf': 1, 'min_samples_split': 2, 'splitter':
'best'}
accuracy : 0.9160714285714286
```

TASK 9

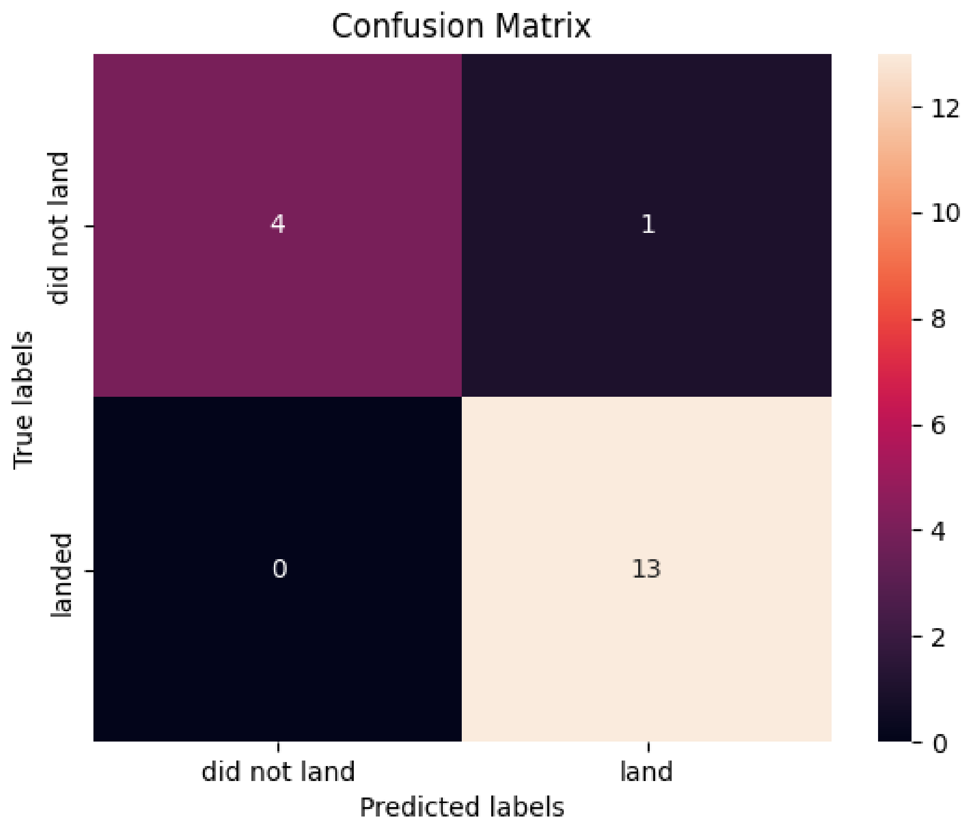
Calculate the accuracy of tree_cv on the test data using the method `score` :

In [71]:
```
tree_cv.score(X_test, Y_test)
```

Out[71]:
```
0.9444444444444444
```

We can plot the confusion matrix

In [73]:
```
yhat = tree_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 10

Create a k nearest neighbors object then create a `GridSearchCV` object `knn_cv` with cv = 10. Fit the object to find the best parameters from the dictionary `parameters` .

In [74]:
```
parameters = {'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
              'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
              'p': [1,2]}

KNN = KNeighborsClassifier()
```

In [75]:
```
knn_cv = GridSearchCV(estimator=KNN, param_grid=parameters, cv=10)
```

In [76]:
```
knn_cv.fit(X_train, Y_train)
```

Out[76]:
```
GridSearchCV(cv=10, estimator=KNeighborsClassifier(),
             param_grid={'algorithm': ['auto', 'ball_tree', 'kd_tree', 'brute'],
                         'n_neighbors': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],
                         'p': [1, 2]})
```

In [77]:
```
print("tuned hpyerparameters :(best parameters) ",knn_cv.best_params_)
print("accuracy :",knn_cv.best_score_)
```

```
tuned hpyerparameters :(best parameters)  {'algorithm': 'auto', 'n_neighbors': 3,
'p': 1}
accuracy : 0.8482142857142858
```

TASK 11

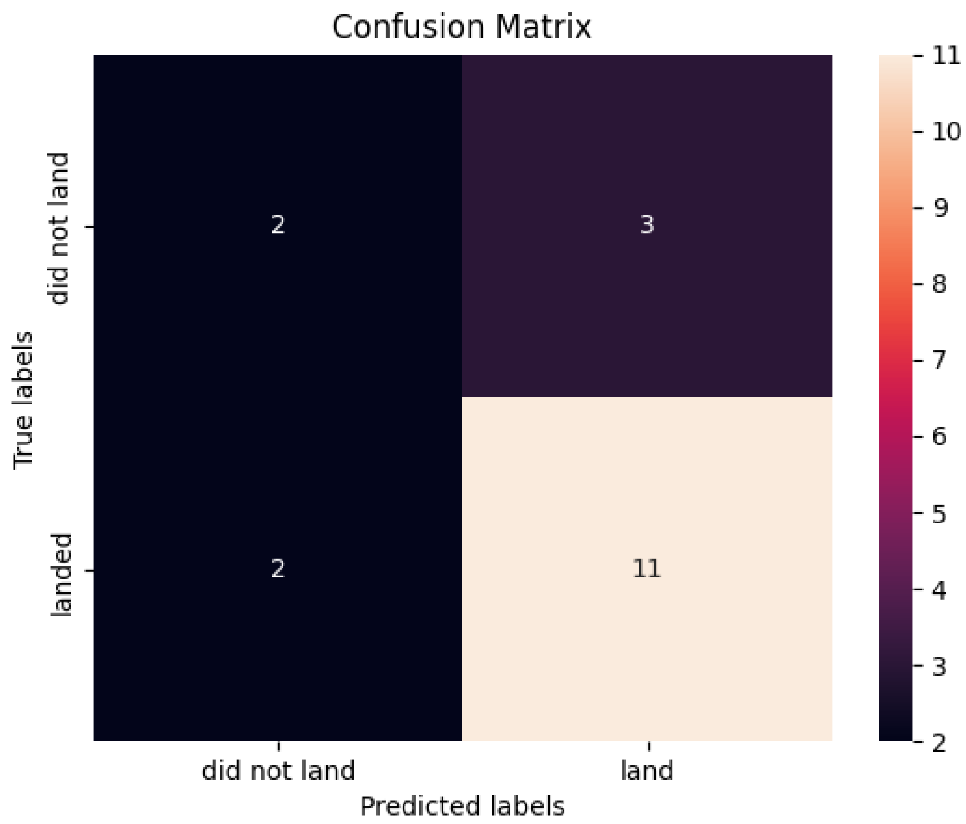
Calculate the accuracy of knn_cv on the test data using the method `score` :

In [78]:
```
knn_cv.score(X_test, Y_test)
```

Out[78]:
```
0.7222222222222222
```

We can plot the confusion matrix

In [79]:
```
yhat = knn_cv.predict(X_test)
plot_confusion_matrix(Y_test,yhat)
```



TASK 12

Find the method performs best:

In [107…
```
predictors = [knn_cv, svm_cv, logreg_cv, tree_cv]
best_predictor = ""
```

```python
best_result = 0
values_ = []
for i, predictor in enumerate(predictors):
    #print(predictors[i], end='')
    j = predictor.score(X_test, Y_test)
    values_.append(j)
    print(j)
```

```
0.7222222222222222
0.8888888888888888
0.9444444444444444
0.9444444444444444
```

In [108…

```python
values_
```

Out[108]:

```
[0.7222222222222222,
 0.8888888888888888,
 0.9444444444444444,
 0.9444444444444444]
```

In [109…

```python
predictors = ['knn_cv', 'svm_cv', 'logreg_cv', 'tree_cv']
```

In [110…

```python
plt.bar(np.array(predictors), values_);
plt.yticks(np.arange(0, max(values_)+ 0.2, 0.1))
plt.grid();
plt.show();
```