titanic-dataset-my-first-submission

August 12, 2022

0.1 Titanic Dataset

My goal is build a new version [1] of predictive model that predicts which passengers survived the Titanic shipwreck; and try to use some ”advanced” (for me :smile:) techniques such as cross validation, grid search and an ensemble algorithm like ”Random Forest Classifier”. Let’s see what happen :smile:

1 Import data set

[ ]:
```
import pandas as pd

# We'll use a dataset taken from: https://www.kaggle.com/competitions/titanic
dfTrain = pd.read_csv("./data/train.csv", sep=',')
dfTest = pd.read_csv("./data/test.csv", sep=",")
```

2 Basic EDA and cleaning data

[ ]:
```
from basic_exploration import *
basicEDA(dfTrain, "Titanic Train")
```

Just first five rows

```
   PassengerId  Survived  Pclass  \
0            1         0       3
1            2         1       1
2            3         1       3
3            4         1       1
4            5         0       3

                                                Name     Sex   Age  SibSp  \
0                            Braund, Mr. Owen Harris    male  22.0      1
1  Cumings, Mrs. John Bradley (Florence Briggs Th...  female  38.0      1
2                             Heikkinen, Miss. Laina  female  26.0      0
```

[1] See https://www.kaggle.com/code/francescopaolol/logisticregression-on-complete-titanic-dataset

```
3       Futrelle, Mrs. Jacques Heath (Lily May Peel)  female  35.0      1
4                           Allen, Mr. William Henry    male  35.0      0

   Parch            Ticket     Fare Cabin Embarked
0      0         A/5 21171   7.2500   NaN        S
1      0          PC 17599  71.2833   C85        C
2      0  STON/O2. 3101282   7.9250   NaN        S
3      0            113803  53.1000  C123        S
4      0            373450   8.0500   NaN        S
```

```
'The Titanic Train data set consists of 12 different features which for 891 samples.'
```

Info about the index dtype and columns, non-null values and memory usage.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 891 entries, 0 to 890
Data columns (total 12 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  891 non-null    int64
 1   Survived     891 non-null    int64
 2   Pclass       891 non-null    int64
 3   Name         891 non-null    object
 4   Sex          891 non-null    object
 5   Age          714 non-null    float64
 6   SibSp        891 non-null    int64
 7   Parch        891 non-null    int64
 8   Ticket       891 non-null    object
 9   Fare         891 non-null    float64
 10  Cabin        204 non-null    object
 11  Embarked     889 non-null    object
dtypes: float64(2), int64(5), object(5)
memory usage: 83.7+ KB
```

```
None
```

Check all of the values in the data frame which holds my data.

```
       PassengerId       Survived         Pclass           Name  \
0    <class 'int'>  <class 'int'>  <class 'int'>  <class 'str'>
1    <class 'int'>  <class 'int'>  <class 'int'>  <class 'str'>
2    <class 'int'>  <class 'int'>  <class 'int'>  <class 'str'>
3    <class 'int'>  <class 'int'>  <class 'int'>  <class 'str'>
4    <class 'int'>  <class 'int'>  <class 'int'>  <class 'str'>
..             ...            ...            ...            ...
886  <class 'int'>  <class 'int'>  <class 'int'>  <class 'str'>
887  <class 'int'>  <class 'int'>  <class 'int'>  <class 'str'>
888  <class 'int'>  <class 'int'>  <class 'int'>  <class 'str'>
```

```
889  <class 'int'>  <class 'int'>  <class 'int'>  <class 'str'>
890  <class 'int'>  <class 'int'>  <class 'int'>  <class 'str'>

               Sex            Age          SibSp          Parch  \
0    <class 'str'>  <class 'float'>  <class 'int'>  <class 'int'>
1    <class 'str'>  <class 'float'>  <class 'int'>  <class 'int'>
2    <class 'str'>  <class 'float'>  <class 'int'>  <class 'int'>
3    <class 'str'>  <class 'float'>  <class 'int'>  <class 'int'>
4    <class 'str'>  <class 'float'>  <class 'int'>  <class 'int'>
..             ...              ...            ...            ...
886  <class 'str'>  <class 'float'>  <class 'int'>  <class 'int'>
887  <class 'str'>  <class 'float'>  <class 'int'>  <class 'int'>
888  <class 'str'>  <class 'float'>  <class 'int'>  <class 'int'>
889  <class 'str'>  <class 'float'>  <class 'int'>  <class 'int'>
890  <class 'str'>  <class 'float'>  <class 'int'>  <class 'int'>

            Ticket             Fare            Cabin       Embarked
0    <class 'str'>  <class 'float'>  <class 'float'>  <class 'str'>
1    <class 'str'>  <class 'float'>    <class 'str'>  <class 'str'>
2    <class 'str'>  <class 'float'>  <class 'float'>  <class 'str'>
3    <class 'str'>  <class 'float'>    <class 'str'>  <class 'str'>
4    <class 'str'>  <class 'float'>  <class 'float'>  <class 'str'>
..             ...              ...              ...            ...
886  <class 'str'>  <class 'float'>  <class 'float'>  <class 'str'>
887  <class 'str'>  <class 'float'>    <class 'str'>  <class 'str'>
888  <class 'str'>  <class 'float'>  <class 'float'>  <class 'str'>
889  <class 'str'>  <class 'float'>    <class 'str'>  <class 'str'>
890  <class 'str'>  <class 'float'>  <class 'float'>  <class 'str'>

[891 rows x 12 columns]
```

Count na values

```
PassengerId      0
Survived         0
Pclass           0
Name             0
Sex              0
Age            177
SibSp            0
Parch            0
Ticket           0
Fare             0
Cabin          687
Embarked         2
dtype: int64
```

```
"The columns with missing data are: ['Age', 'Cabin', 'Embarked']"


Percent of missing 'Age' records is 19.865 %
Percent of missing 'Cabin' records is 77.104 %
Percent of missing 'Embarked' records is 0.224 %
```

Show the statistic report of the numeric features of the dataset

```
             count        mean         std   min       25%       50%    75%  \
PassengerId  891.0  446.000000  257.353842  1.00  223.5000  446.0000  668.5
Survived     891.0    0.383838    0.486592  0.00    0.0000    0.0000    1.0
Pclass       891.0    2.308642    0.836071  1.00    2.0000    3.0000    3.0
Age          714.0   29.699118   14.526497  0.42   20.1250   28.0000   38.0
SibSp        891.0    0.523008    1.102743  0.00    0.0000    0.0000    1.0
Parch        891.0    0.381594    0.806057  0.00    0.0000    0.0000    0.0
Fare         891.0   32.204208   49.693429  0.00    7.9104   14.4542   31.0

                  max
PassengerId  891.0000
Survived       1.0000
Pclass         3.0000
Age           80.0000
SibSp          8.0000
Parch          6.0000
Fare         512.3292
```

Show the statistic report of the categorical features of the dataset

```
         count unique                 top freq
Name       891    891  Turcin, Mr. Stjepan    1
Sex        891      2                male  577
Ticket     891    681              347082    7
Cabin      204    147             B96 B98    4
Embarked   889      3                   S  644
```

[ ]:
```
basicEDA(dfTest, "Titanic Test")
```

Just first five rows

```
   PassengerId  Pclass                                          Name     Sex  \
0          892       3                              Kelly, Mr. James    male
1          893       3              Wilkes, Mrs. James (Ellen Needs)  female
2          894       2                     Myles, Mr. Thomas Francis    male
3          895       3                              Wirz, Mr. Albert    male
4          896       3  Hirvonen, Mrs. Alexander (Helga E Lindqvist)  female

    Age  SibSp  Parch   Ticket     Fare Cabin Embarked
0  34.5      0      0   330911   7.8292   NaN        Q
```

```
1  47.0      1      0   363272   7.0000   NaN        S
2  62.0      0      0   240276   9.6875   NaN        Q
3  27.0      0      0   315154   8.6625   NaN        S
4  22.0      1      1  3101298  12.2875   NaN        S
```

```
'The Titanic Test data set consists of 11 different features which for 418 samples.'
```

Info about the index dtype and columns, non-null values and memory usage.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 418 entries, 0 to 417
Data columns (total 11 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   PassengerId  418 non-null    int64
 1   Pclass       418 non-null    int64
 2   Name         418 non-null    object
 3   Sex          418 non-null    object
 4   Age          332 non-null    float64
 5   SibSp        418 non-null    int64
 6   Parch        418 non-null    int64
 7   Ticket       418 non-null    object
 8   Fare         417 non-null    float64
 9   Cabin        91 non-null     object
 10  Embarked     418 non-null    object
dtypes: float64(2), int64(4), object(5)
memory usage: 36.0+ KB
```

```
None
```

Check all of the values in the data frame which holds my data.

```
       PassengerId         Pclass           Name            Sex  \
0    <class 'int'>  <class 'int'>  <class 'str'>  <class 'str'>
1    <class 'int'>  <class 'int'>  <class 'str'>  <class 'str'>
2    <class 'int'>  <class 'int'>  <class 'str'>  <class 'str'>
3    <class 'int'>  <class 'int'>  <class 'str'>  <class 'str'>
4    <class 'int'>  <class 'int'>  <class 'str'>  <class 'str'>
..             ...            ...            ...            ...
413  <class 'int'>  <class 'int'>  <class 'str'>  <class 'str'>
414  <class 'int'>  <class 'int'>  <class 'str'>  <class 'str'>
415  <class 'int'>  <class 'int'>  <class 'str'>  <class 'str'>
416  <class 'int'>  <class 'int'>  <class 'str'>  <class 'str'>
417  <class 'int'>  <class 'int'>  <class 'str'>  <class 'str'>

                 Age          SibSp          Parch         Ticket  \
0    <class 'float'>  <class 'int'>  <class 'int'>  <class 'str'>
1    <class 'float'>  <class 'int'>  <class 'int'>  <class 'str'>
```

```
2    <class 'float'>  <class 'int'>  <class 'int'>  <class 'str'>
3    <class 'float'>  <class 'int'>  <class 'int'>  <class 'str'>
4    <class 'float'>  <class 'int'>  <class 'int'>  <class 'str'>
..               ...            ...            ...            ...
413  <class 'float'>  <class 'int'>  <class 'int'>  <class 'str'>
414  <class 'float'>  <class 'int'>  <class 'int'>  <class 'str'>
415  <class 'float'>  <class 'int'>  <class 'int'>  <class 'str'>
416  <class 'float'>  <class 'int'>  <class 'int'>  <class 'str'>
417  <class 'float'>  <class 'int'>  <class 'int'>  <class 'str'>

                Fare            Cabin       Embarked
0    <class 'float'>  <class 'float'>  <class 'str'>
1    <class 'float'>  <class 'float'>  <class 'str'>
2    <class 'float'>  <class 'float'>  <class 'str'>
3    <class 'float'>  <class 'float'>  <class 'str'>
4    <class 'float'>  <class 'float'>  <class 'str'>
..               ...              ...            ...
413  <class 'float'>  <class 'float'>  <class 'str'>
414  <class 'float'>    <class 'str'>  <class 'str'>
415  <class 'float'>  <class 'float'>  <class 'str'>
416  <class 'float'>  <class 'float'>  <class 'str'>
417  <class 'float'>  <class 'float'>  <class 'str'>

[418 rows x 11 columns]
```

Count na values

```
PassengerId      0
Pclass           0
Name             0
Sex              0
Age             86
SibSp            0
Parch            0
Ticket           0
Fare             1
Cabin          327
Embarked         0
dtype: int64
```

```
"The columns with missing data are: ['Age', 'Fare', 'Cabin']"
```

```
Percent of missing 'Age' records is 20.574 %
Percent of missing 'Fare' records is 0.239 %
Percent of missing 'Cabin' records is 78.23 %
```

Show the statistic report of the numeric features of the dataset

	count	mean	std	min	25%	50%	\
PassengerId	418.0	1100.500000	120.810458	892.00	996.2500	1100.5000	
Pclass	418.0	2.265550	0.841838	1.00	1.0000	3.0000	
Age	332.0	30.272590	14.181209	0.17	21.0000	27.0000	
SibSp	418.0	0.447368	0.896760	0.00	0.0000	0.0000	
Parch	418.0	0.392344	0.981429	0.00	0.0000	0.0000	
Fare	417.0	35.627188	55.907576	0.00	7.8958	14.4542	

	75%	max
PassengerId	1204.75	1309.0000
Pclass	3.00	3.0000
Age	39.00	76.0000
SibSp	1.00	8.0000
Parch	0.00	9.0000
Fare	31.50	512.3292

Show the statistic report of the categorical features of the dataset

	count	unique	top	freq
Name	418	418	Gracie, Col. Archibald IV	1
Sex	418	2	male	266
Ticket	418	363	PC 17608	5
Cabin	91	76	B57 B59 B63 B66	3
Embarked	418	3	S	270

Some considerations: first of all, we can see how ”Survived” is our target: - Survived 0 = no, 1 = yes

As regards the other features we have: - PassengerID: - Pclass: 1 = 1st, 2 = 2nd, 3 = 3rd - Name: self explanatory - Sex: self explanatory - Age: self explanatory - SibSp = nr of sibilings / spouses abroad - Parch = nr of parents / children abroad - Ticket = self explanatory - Fare = passenger fare - Cabin = self explanatory - Embarked = port of embrarkation --> C = Chernourg, Q = Queenstown, S = Southhampton

I think I can do something in order to slim down this dataset.

We can see that the two dataset are similar.

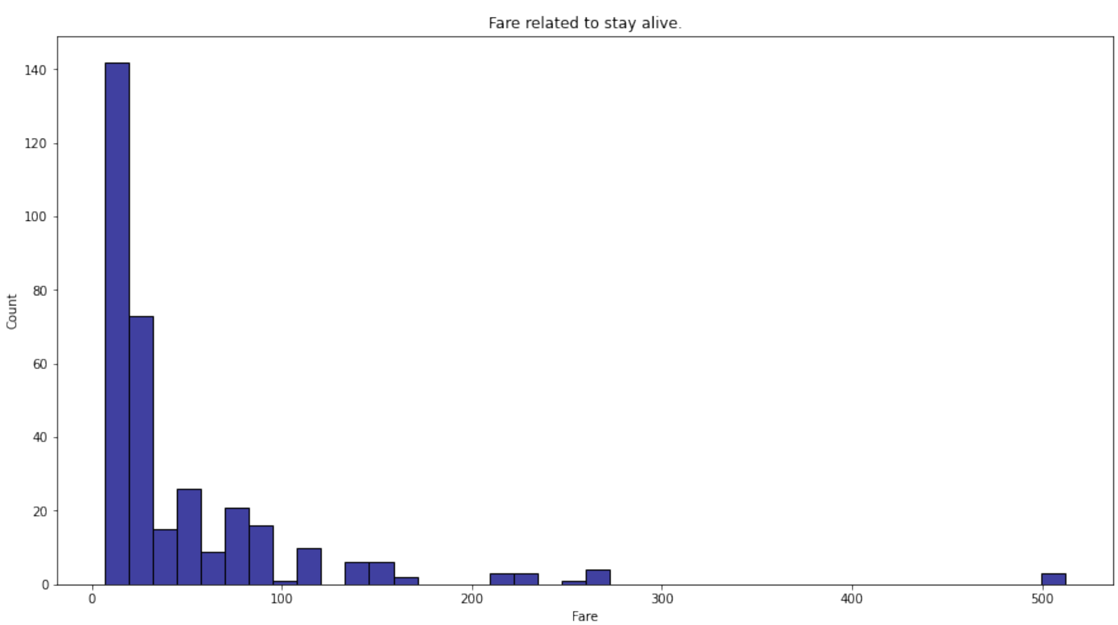
2.1 Feature engineering

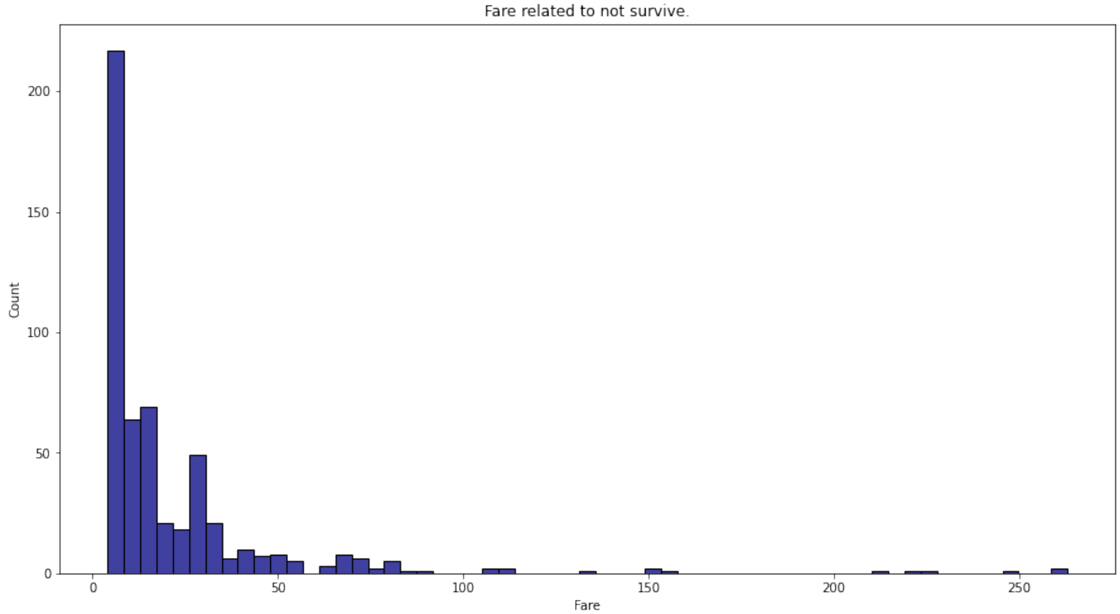
So, I think that ’Name’, ’Embarked’, ’Cabin’ and ’Ticket’ features can be dropped because I don’t believe that be called ”Nicholas” or ”Augusta” increases the possibility to survive. Same reasoning for the others features. Then, let’s start to drop useless features.

[ ]:
```
delColumn(dfTrain, "Name")
delColumn(dfTrain, "Ticket")
delColumn(dfTrain, "Cabin")
delColumn(dfTrain, "Embarked")
```

As regards ”Fare”, let’s check out if the fare is related to the better chances to be survive.

[ ]:
```
import matplotlib.pyplot as plt
import seaborn as sns

if "Fare" in dfTrain.columns:
    dfTmp = dfTrain[["Fare", "Survived"]]
    dfSurvivedYes = dfTmp[dfTmp.Survived == 1]
    dfSurvivedNo = dfTmp[dfTmp.Survived == 0]

    plt.figure(figsize = (15,8))
    plt.title("Fare related to stay alive.")
    sns.histplot(data = dfSurvivedYes[dfSurvivedYes.Fare > 0],
                 x = 'Fare',
                 color = 'navy'
                )

    plt.figure(figsize = (15,8))
    plt.title("Fare related to not survive.")
    sns.histplot(data = dfSurvivedNo[dfSurvivedNo.Fare > 0],
                 x = 'Fare',
                 color = 'navy'
                )



    plt.show()
```





So one can be dead or alive, no matter how much he paid as fare: we can also drop this feature.

[ ]:
```
delColumn(dfTrain, "Fare")
```

At this point we have few but good features. Remains to resolve the missing 'Age' records (that is 19.865 %). So, let's find out correlations with "Age" feature.

[ ]:
```
dfTrain.corr()
```

[ ]:

	PassengerId	Survived	Pclass	Age	SibSp	Parch
PassengerId	1.000000	-0.005007	-0.035144	0.036847	-0.057527	-0.001652
Survived	-0.005007	1.000000	-0.338481	-0.077221	-0.035322	0.081629
Pclass	-0.035144	-0.338481	1.000000	-0.369226	0.083081	0.018443
Age	0.036847	-0.077221	-0.369226	1.000000	-0.308247	-0.189119
SibSp	-0.057527	-0.035322	0.083081	-0.308247	1.000000	0.414838
Parch	-0.001652	0.081629	0.018443	-0.189119	0.414838	1.000000

We have three feature correlate with "Age": Pclass (PCC: -0.369226), SibSp (PCC: -0.308247), Parch (PCC: -0.189119). I'm going to use "IterativeImputer" (which is a multivariate imputer that estimates each feature from all the others) with RandomForestRegressor...

[ ]:
```
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

from sklearn.ensemble import RandomForestRegressor
import pandas as pd

dftmp = dfTrain.loc[:, ["Age"]]
```

```
imp = IterativeImputer(RandomForestRegressor(),
                       max_iter=10,
                       tol=0.001,
                       random_state=0,
                       sample_posterior=False,
                       verbose=True)
dftmp = pd.DataFrame(imp.fit_transform(dftmp),
                     columns=dftmp.columns)
```

...check if all data are property filled...

[ ]:
```
print("\nNumber of rows where 'Age' are null or empty")
print(dftmp.isnull().sum())
```

```
Number of rows where 'Age' are null or empty
Age    0
dtype: int64
```

...and finally refill the missing Age values.

[ ]:
```
delColumn(dfTrain, "Age")
dfTrain = dfTrain.join(dftmp)
```

Remains to encode the "Sex" feature.

[ ]:
```
from sklearn.preprocessing import LabelEncoder

labelencoder_X = LabelEncoder()

dfTrain["Sex"] = labelencoder_X.fit_transform(dfTrain["Sex"])
```

So this is our starting dataset.

[ ]:
```
dfTrain.head()
```

[ ]:

	PassengerId	Survived	Pclass	Sex	SibSp	Parch	Age
0	1	0	3	1	1	0	22.0
1	2	1	1	0	1	0	38.0
2	3	1	3	0	0	0	26.0
3	4	1	1	0	1	0	35.0
4	5	0	3	1	0	0	35.0

Same things to test dataset

[ ]:
```
delColumn(dfTest, "Name")
delColumn(dfTest, "Ticket")
delColumn(dfTest, "Cabin")
```

```
delColumn(dfTest, "Embarked")
delColumn(dfTest, "Fare")

dftmp = dfTest.loc[:, ["Age"]]

imp = IterativeImputer(RandomForestRegressor(),
                       max_iter=10,
                       tol=0.001,
                       random_state=0,
                       sample_posterior=False,
                       verbose=True)
dftmp = pd.DataFrame(imp.fit_transform(dftmp),
                     columns=dftmp.columns)
dfTest.drop("Age", axis=1, inplace=True)
dfTest = dfTest.join(dftmp)

dfTest["Sex"] = labelencoder_X.fit_transform(dfTest["Sex"])

dfTest.head()
```

[ ]:

	PassengerId	Pclass	Sex	SibSp	Parch	Age
0	892	3	1	0	0	34.5
1	893	3	0	1	0	47.0
2	894	2	1	0	0	62.0
3	895	3	1	0	0	27.0
4	896	3	0	1	1	22.0

3 Train and test the model

Once prepared data, we can split data in train and test, as usual.

[ ]:
```
from sklearn.model_selection import train_test_split

X = dfTrain.drop("Survived", axis=1)
y = dfTrain["Survived"]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4)
```

And prepare the grid search with RandomForestClassifier model, and return

[ ]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV, KFold

rndForestParams = {
    "criterion" : ["gini", "entropy"],                  # {"gini", "entropy",␣
 ↪"log_loss"},
```

```
    "min_samples_leaf" : [1, 5, 10],                        # The minimum number of␣
↪samples required to be at a leaf node.
    "min_samples_split" : [4, 10, 14, 16],                  # The minimum number of␣
↪samples required to split an internal node
    "n_estimators": [150, 300, 700, 1000]                   # The number of trees␣
↪in the forest.
}

rfModel = RandomForestClassifier(
    max_features = "sqrt",                                  # The number of␣
↪features to consider when looking for the best split
    oob_score = True,                                       # Whether to use␣
↪out-of-bag samples to estimate the generalization score.
                                                            # Only available if␣
↪'bootstrap = True' (that's default value!)
    random_state = 1,                                       # Controls both the␣
↪randomness of the bootstrapping of the samples used when building trees
    n_jobs = -1                                             # '-1' means using all␣
↪processors.
)

cv_method = KFold(n_splits = 10, shuffle = True)


gs = GridSearchCV(
    estimator = rfModel,
    param_grid = rndForestParams,
    scoring='accuracy',
    cv = cv_method,
    n_jobs=-1
)
```

Now we can fit the gridsearch object (it will take a while…).

[ ]:
```
gs.fit(X_train, y_train)
```

[ ]:
```
GridSearchCV(cv=KFold(n_splits=10, random_state=None, shuffle=True),
             estimator=RandomForestClassifier(max_features='sqrt', n_jobs=-1,
                                              oob_score=True, random_state=1),
             n_jobs=-1,
             param_grid={'criterion': ['gini', 'entropy'],
                         'min_samples_leaf': [1, 5, 10],
                         'min_samples_split': [4, 10, 14, 16],
                         'n_estimators': [150, 300, 700, 1000]},
             scoring='accuracy')
```

We can see the parameter setting that gave the best results on the hold out data…

[ ]:
```
gs.best_params_
```

[ ]:
```
{'criterion': 'entropy',
 'min_samples_leaf': 1,
 'min_samples_split': 10,
 'n_estimators': 150}
```

...and set up a model with the estimator that was chosen by the search.

[ ]:
```
RFC_Model = gs.best_estimator_
```

And show what is the average of all cv folds for a single combination of the parameters you specify in the tuned_params.

[ ]:
```
gs.best_score_                                          #Mean cross-validated score of␣
 ↪the best_estimator
```

[ ]:
```
0.8221872816212439
```

Let's predict on train data.

[ ]:
```
RFC_Model.predict(X_train)
```

[ ]:
```
array([0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0,
       0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 1, 1,
       1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1,
       1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 1,
       0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0,
       0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0,
       1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0,
       0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 0, 1, 0,
       1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0,
       0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 1,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0, 0,
       1, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 0, 0])
```

And make the prediction on test data.

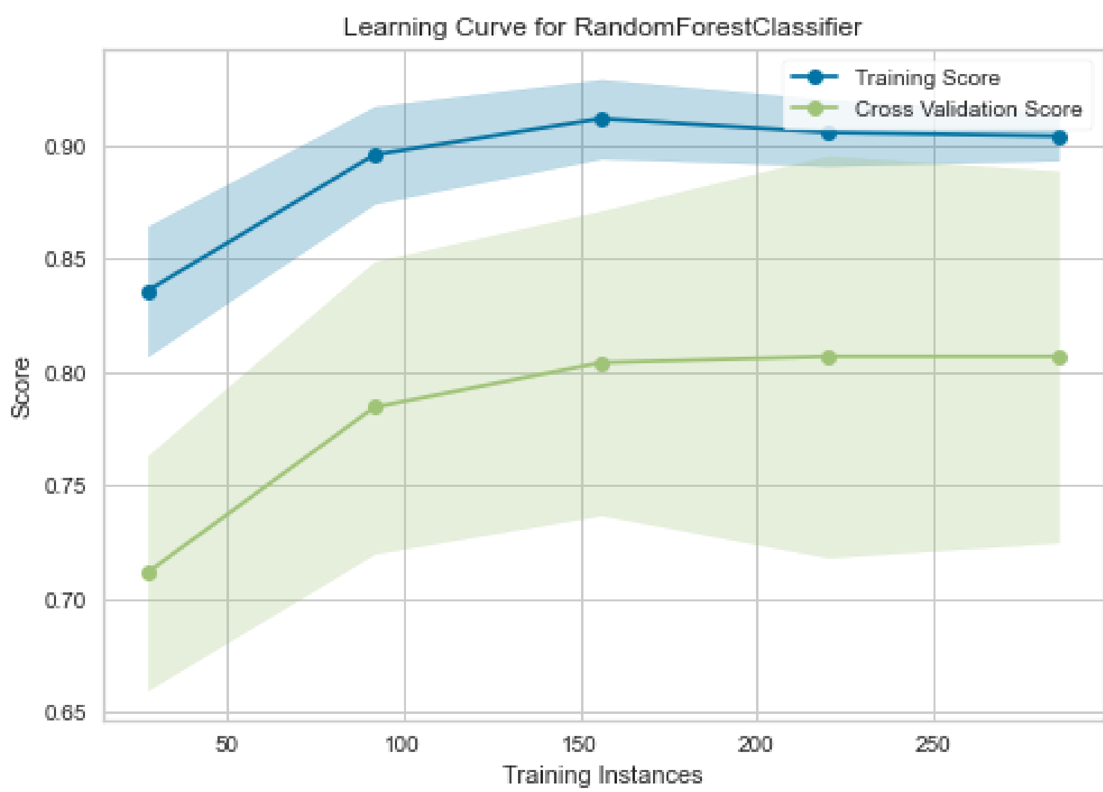
```
[ ]: RFC_Model.score(X_test, y_test)                     #Return the mean accuracy on␣
      ↪the given test data and labels
```

```
[ ]: 0.8067226890756303
```

3.1 Model Performance Analysis

Displaying the learning curve, we note that we have enough data to try to make a model.

```
[ ]: from yellowbrick.model_selection import learning_curve

     learning_curve(RFC_Model, X_test, y_test, scoring='accuracy')
     plt.show()
```

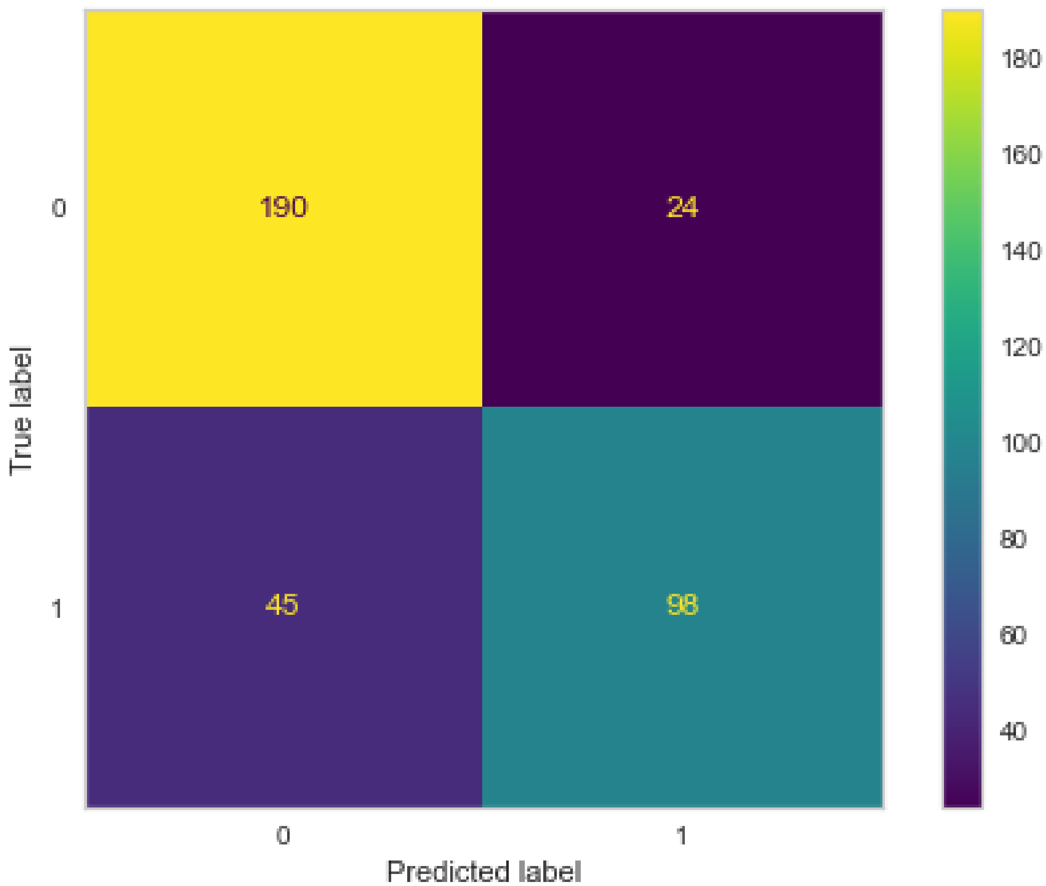



```
[ ]: from sklearn.metrics import classification_report

     dfReport = pd.DataFrame(classification_report(y_test, RFC_Model.
      ↪predict(X_test), output_dict=True))
     dfReport
```

[ ]:

	0	1	accuracy	macro avg	weighted avg
precision	0.808511	0.803279	0.806723	0.805895	0.806415
recall	0.887850	0.685315	0.806723	0.786583	0.806723
f1-score	0.846325	0.739623	0.806723	0.792974	0.803584
support	214.000000	143.000000	0.806723	357.000000	357.000000

[ ]:
```
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

predictions = RFC_Model.predict(X_test)
cm = confusion_matrix(y_test, predictions, labels = RFC_Model.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix = cm,
                              display_labels = RFC_Model.classes_)
disp.plot()
plt.grid(visible=None)
plt.show()
```



4 Submission

[ ]:
```python
predictions = RFC_Model.predict(dfTest)
predictions
```

[ ]:
```
array([0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1,
       1, 1, 1, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1,
       1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1,
       1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 1,
       0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1,
       1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1,
       0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 1, 0,
       1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 1, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 0, 0,
       1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0,
       1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0, 0, 1,
       0, 1, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0])
```

[ ]:
```python
PassengerId = dfTest['PassengerId']
submission = pd.DataFrame({"PassengerId": PassengerId,"Survived": predictions})
submission.to_csv('submission.csv', index=False)
```