1: firstly, we import what we need, and observe the data

In [1]:
```
import numpy as np
import pandas as pd
import xgboost as xgb
from xgboost import XGBRegressor as XGBR
from sklearn.ensemble import RandomForestRegressor as RFR
from sklearn.linear_model import LinearRegression as LinearR
from sklearn.datasets import load_boston
from sklearn.model_selection import KFold, cross_val_score as CVS, train_te
from sklearn.metrics import mean_squared_error as MSE
import matplotlib.pyplot as plt
from time import time
import datetime
```

In [2]:
```
data = pd.read_csv("/Users/guangxinsu/Desktop/penguins_raw.csv")
```

In [3]:
```
data.head()
```

Out[3]:

	studyName	Sample Number	Species	Region	Island	Stage	Individual ID	Clutch Completion	Date Egg	Culm Leng (mı
0	PAL0708	1	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A1	Yes	2007-11-11	39
1	PAL0708	2	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N1A2	Yes	2007-11-11	39
2	PAL0708	3	Adelie Penguin (Pygoscelis adeliae)	Anvers	Torgersen	Adult, 1 Egg Stage	N2A1	Yes	2007-11-16	40
			Adelie			Adult,				

2: Data preprocess：drop useless features, Turn letters, genders and other words into numbers，Fill the nan values with proper values like mean number.

```
data.drop(["Comments","Stage","studyName","Sample Number","Region","Individual ID","Date Egg"],inplace=True,axis=1)
data["Culmen Length (mm)"] = data["Culmen Length (mm)"].fillna(data["Culmen Length (mm)"].mean())
data["Culmen Depth (mm)"] = data["Culmen Depth (mm)"].fillna(data["Culmen Depth (mm)"].mean())
data["Flipper Length (mm)"] = data["Flipper Length (mm)"].fillna(data["Flipper Length (mm)"].mean())
data["Body Mass (g)"] = data["Body Mass (g)"].fillna(data["Body Mass (g)"].mean())

data["Delta 15 N (o/oo)"] = data["Delta 15 N (o/oo)"].fillna(data["Delta 15 N (o/oo)"].mean())
data["Delta 13 C (o/oo)"] = data["Body Mass (g)"].fillna(data["Body Mass (g)"].mean())
data.loc[:,"Sex"]=(data["Sex"]=="male").astype("int")
data.loc[:,"Clutch Completion"]=(data["Clutch Completion"]=="Yes").astype("int")
#Check how many values are turned into lists
labels=data["Island"].unique().tolist()
#Convert letters to number
data["Island"]=data["Island"].apply(lambda x: labels.index(x))
labels = data["Island"].unique().tolist()
#Check how many values are turned into lists
labels=data["Species"].unique().tolist()
print(labels)
#Convert letters to number
data["Species"]=data["Species"].apply(lambda x: labels.index(x))
labels = data["Species"].unique().tolist()
data.head(300)
```

```
data.head(300)
```

```
['Adelie Penguin (Pygoscelis adeliae)', 'Gentoo penguin (Pygoscelis papu
a)', 'Chinstrap penguin (Pygoscelis antarctica)']
```

ipecies	Island	Clutch Completion	Culmen Length (mm)	Culmen Depth (mm)	Flipper Length (mm)	Body Mass (g)	Sex	Delta 15 N (o/oo)	Delta 13 ((o/oc
0	0	1	39.10000	18.70000	181.000000	3750.000000	0	8.733382	3750.00000
0	0	1	39.50000	17.40000	186.000000	3800.000000	0	8.949560	3800.00000
0	0	1	40.30000	18.00000	195.000000	3250.000000	0	8.368210	3250.00000
0	0	1	43.92193	17.15117	200.915205	4201.754386	0	8.733382	4201.75438
0	0	1	36.70000	19.30000	193.000000	3450.000000	0	8.766510	3450.00000
...	...	...	...	...	...	...	...	...	.
2	2	1	49.20000	18.20000	195.000000	4400.000000	0	9.271580	4400.00000
2	2	1	42.40000	17.30000	181.000000	3600.000000	0	9.351380	3600.00000
2	2	1	48.50000	17.50000	191.000000	3400.000000	0	9.426660	3400.00000
2	2	0	43.20000	16.60000	187.000000	2900.000000	0	9.354160	2900.00000
2	2	0	50.60000	19.40000	193.000000	3800.000000	0	9.281530	3800.00000

vs × 10 columns

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 344 entries, 0 to 343
Data columns (total 10 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Species              344 non-null    int64  
 1   Island               344 non-null    int64  
 2   Clutch Completion    344 non-null    int64  
 3   Culmen Length (mm)   344 non-null    float64
 4   Culmen Depth (mm)    344 non-null    float64
 5   Flipper Length (mm)  344 non-null    float64
 6   Body Mass (g)        344 non-null    float64
 7   Sex                  344 non-null    int64  
 8   Delta 15 N (o/oo)    344 non-null    float64
 9   Delta 13 C (o/oo)    344 non-null    float64
dtypes: float64(6), int64(4)
memory usage: 27.0 KB
```

3: divide train set and test set

In [15]:
```
#data and target
x = data.iloc[:,data.columns != "Species"]
y = data.iloc[:,data.columns == "Species"]
```

In [16]:
```
#train and test set
Xtrain,Xtest,Ytrain,Ytest = TTS(x,y,test_size=0.3,random_state=420)
```

4: Xgboost Preliminary training & cross-validation & random forest & linear-regression
Compare with each other

```
In [17]: reg = XGBR(n_estimators=100).fit(Xtrain,Ytrain)
         reg.predict(Xtest)
         reg.score(Xtest,Ytest)

Out[17]: 0.9175140971181192
```

```
In [19]: #One of the advantages of the tree model: being able to view the importance
         #the feature importances
         reg.feature_importances_

Out[19]: array([4.5520934e-01, 1.1017058e-03, 4.0963203e-01, 9.0594076e-02,
                4.1496810e-02, 1.8036369e-03, 0.0000000e+00, 1.6241733e-04,
                0.0000000e+00], dtype=float32)
```

```
In [20]: #Cross-validation
         reg = XGBR(n_estimators=100)

In [21]: CVS(reg,Xtrain,Ytrain,cv=5)

Out[21]: array([0.96334981, 0.93413385, 0.964922  , 0.94453331, 0.93087006])

In [22]: CVS(reg,Xtrain,Ytrain,cv=5,scoring='neg_mean_squared_error').mean()

Out[22]: -0.03033896242461126
```

```
]: #Let's take a look at all the model evaluation indicators in sklearn
   import sklearn
   sorted(sklearn.metrics.SCORERS.keys())
   #random forest
   rfr = RFR(n_estimators=100)
   CVS(rfr,Xtrain,Ytrain,cv=5).mean()
```

```
                ...
         estimator.fit(X_train, y_train, **fit_params)

Out[23]: 0.929550818059646
```

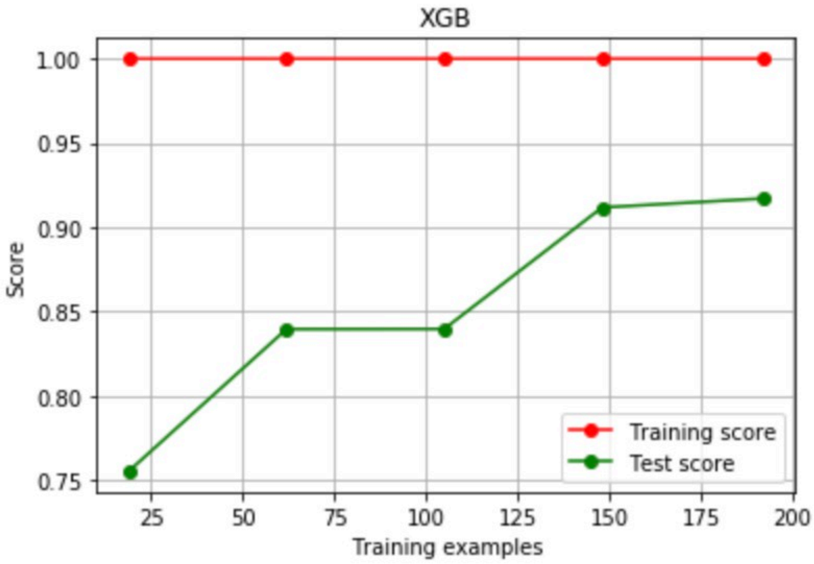
```
[25]: #linear regression
      lr = LinearR()
      CVS(lr,Xtrain,Ytrain,cv=5).mean()

[25]: 0.7479632602108952

[26]: CVS(lr,Xtrain,Ytrain,cv=5,scoring='neg_mean_squared_error').mean()

[26]: -0.14709472237947832
```

```
In [27]: reg = XGBR(n_estimators=10,silent=False)
         CVS(reg,Xtrain,Ytrain,cv=5,scoring='neg_mean_squared_error').mean()
```

5: plot learning_curve figure of xgboost and analysis, it’s overfitting now!

```
def plot_learning_curve(estimator,title, X, y,
                        ax=None, #Select subgraph
                        ylim=None, #Set the value range of the ordinate
                        cv=None, #cross validation
                        n_jobs=None #Set the thread to use
                        ):
    from sklearn.model_selection import learning_curve
    import matplotlib.pyplot as plt
    import numpy as np
    train_sizes, train_scores, test_scores = learning_curve(estimator, X, y
                                                            ,shuffle=True
                                                            ,cv = cv
                                                            ,random_state = 420
                                                            ,n_jobs = n_jobs)
    if ax == None:
        ax = plt.gca()
    else:
        ax = plt.figure()
    ax.set_title(title)
    if ylim is not None:

        ax.set_ylim(*ylim)
    ax.set_xlabel("Training examples")
    ax.set_ylabel("Score")
    ax.grid() #grid
    ax.plot(train_sizes, np.mean(train_scores, axis=1), 'o-'
            , color="r",label="Training score")
    ax.plot(train_sizes, np.mean(test_scores, axis=1), 'o-'
            , color="g",label="Test score")
    ax.legend(loc="best")
    return ax
```

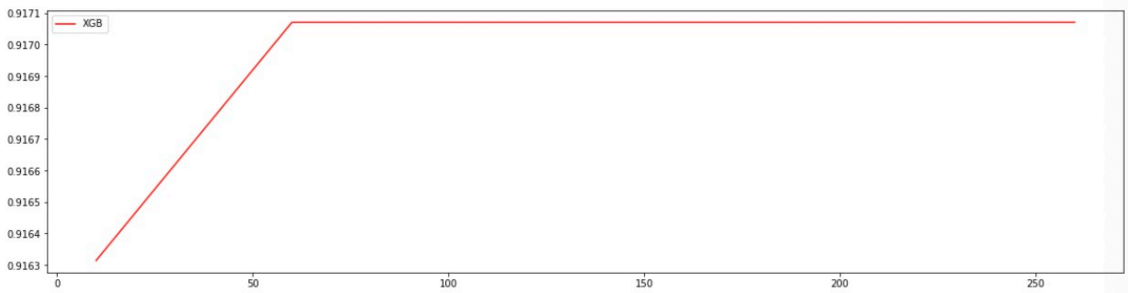
```
29]: cv = KFold(n_splits=5, shuffle = True, random_state=42)
     plot_learning_curve(XGBR(n_estimators=500,random_state=420)
                         ,"XGB",Xtrain,Ytrain,ax=None,cv=cv)
     plt.show()
```



6: find the best n_estimators/the number of trees

```python
#Observe the influence of n_eatimators on the model by using a parameter learning curve
axisx = range(10,310,50)
rs = []
for i in axisx:
    reg = XGBR(n_estimators=i,random_state=420)
    rs.append(CVS(reg,Xtrain,Ytrain,cv=cv).mean())
print(axisx[rs.index(max(rs))],max(rs))
plt.figure(figsize=(20,5))
plt.plot(axisx,rs,c="red",label="XGB")
plt.legend()
plt.show()
```

```
110 0.917070470188386
```



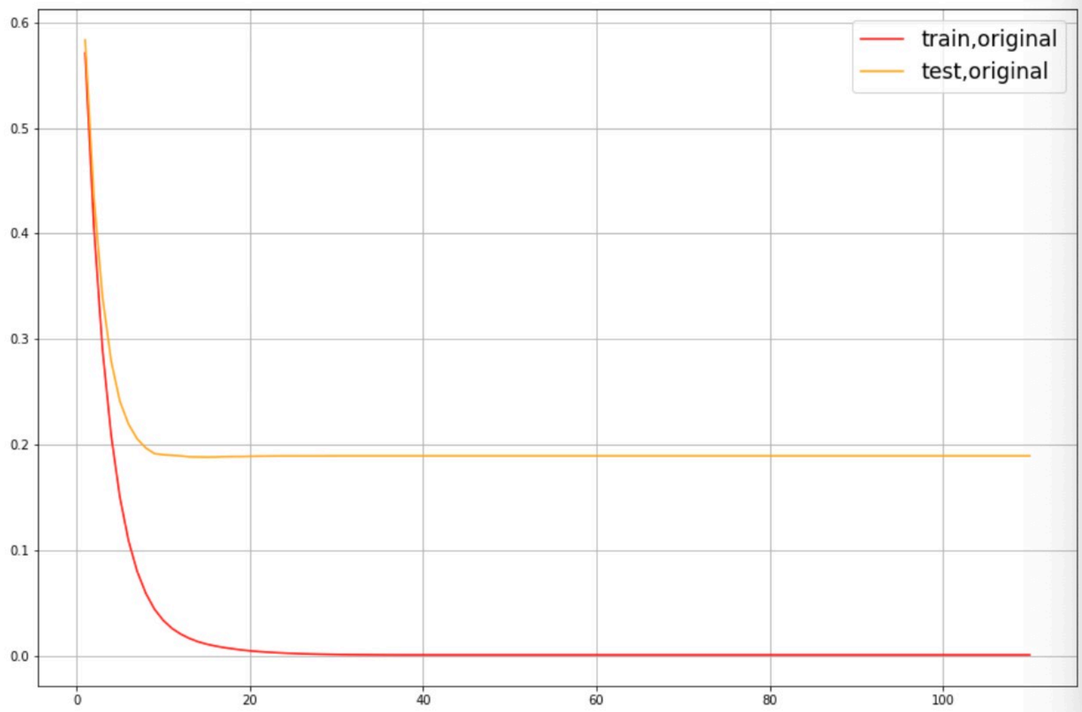

```python
#Verify that the model effect has improved?
time0 = time()
print(XGBR(n_estimators=1,random_state=420).fit(Xtrain,Ytrain).score(Xtest,Ytest))

print(time()-time0)
time0 = time()
print(XGBR(n_estimators=10,random_state=420).fit(Xtrain,Ytrain).score(Xtest,Ytest))

print(time()-time0)

time0 = time()
print(XGBR(n_estimators=40,random_state=420).fit(Xtrain,Ytrain).score(Xtest,Ytest))

print(time()-time0)
time0 = time()
print(XGBR(n_estimators=110,random_state=420).fit(Xtrain,Ytrain).score(Xtest,Ytest))
print(time()-time0)

time0 = time()
print(XGBR(n_estimators=300,random_state=420).fit(Xtrain,Ytrain).score(Xtest,Ytest))
print(time()-time0)
```

```
0.4343177778791587
0.0075042247772216 8
0.9160991184463382
0.018697023391723633
0.9175139811105371
0.0110840797424316 4
0.9175140971181192
0.013628005981445312
0.9175140971181192
0.022681236267089844
```

7: use xgboost and solve the over fitting problems(number of trees :110)

```
dfull = xgb.DMatrix(x,y)
param1 = {'silent':True
          ,'obj':'reg:linear'
,"subsample":1 ,"max_depth":6 ,"eta":0.3
,"gamma":0
,"lambda":1
,"alpha":0 ,"colsample_bytree":1 ,"colsample_bylevel":1 ,"colsample_bynode":1 ,"nfold":5}
num_round = 110
time0 = time()
cvresult1 = xgb.cv(param1, dfull, num_round)
print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))
fig,ax = plt.subplots(1,figsize=(15,10))
#ax.set_ylim(top=5)
ax.grid()
ax.plot(range(1,111),cvresult1.iloc[:,0],c="red",label="train,original")
ax.plot(range(1,111),cvresult1.iloc[:,2],c="orange",label="test,original")
ax.legend(fontsize="xx-large")
plt.show()
```

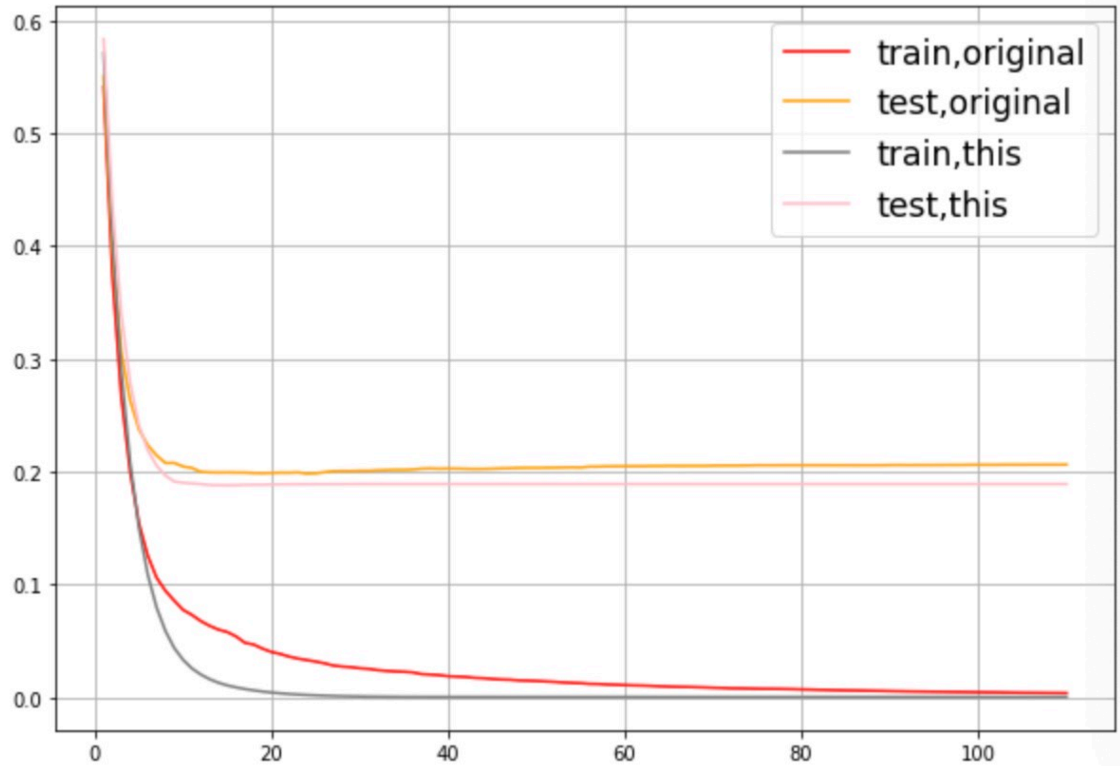
```
00:00:082927
```



Change the parameters and solve the over-fitting(many times)

```
param1 = {'silent':True
          ,'obj':'reg:linear'
,"subsample":1 ,"max_depth":3 ,"eta":0.35
,"gamma":0
,"lambda":1
,"alpha":0 ,"colsample_bytree":1 ,"colsample_bylevel":1 ,"colsample_bynode":1 ,"nfold":5}
num_round = 110
time0 = time()
cvresult1 = xgb.cv(param1, dfull, num_round)
print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))
fig,ax = plt.subplots(1,figsize=(15,10))
#ax.set_ylim(top=5)
ax.grid()
ax.plot(range(1,111),cvresult1.iloc[:,0],c="red",label="train,original")
ax.plot(range(1,111),cvresult1.iloc[:,2],c="orange",label="test,original")
param2 = {'silent':True
          ,'obj':'reg:linear'
          ,"nfold":5}
param3 = {'silent':True
          ,'obj':'reg:linear'
          ,"nfold":5}
time0 = time()
cvresult2 = xgb.cv(param2, dfull, num_round)
print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))
time0 = time()
cvresult3 = xgb.cv(param3, dfull, num_round)
print(datetime.datetime.fromtimestamp(time()-time0).strftime("%M:%S:%f"))
ax.plot(range(1,111),cvresult3.iloc[:,0],c="gray",label="train,this")
ax.plot(range(1,111),cvresult3.iloc[:,2],c="pink",label="test,this")
ax.legend(fontsize="xx-large")
plt.show()
```

```
00:00:095722
```



0.6
0.5
0.4
0.3
0.2
0.1
0.0
0
20
40
60
80
100
train,original
test,original
train,this
test,this

8: we get satisfying parameters (tree size:110)

```
param1 = {'silent':True
          ,'obj':'reg:linear'
          ,"subsample":1
          ,"max_depth":3
          ,"eta":0.35
          ,"gamma":0
          ,"lambda":1
          ,"alpha":0
          ,"colsample_bytree":1
          ,"colsample_bylevel":1
          ,"colsample_bynode":1
          ,"nfold":5}
num_round = 110
```

9: save the data as a file and train the model with parameters we processed

```
[31]: ['/Users/guangxinsu/Desktop/ml',
 '/opt/anaconda3/lib/python37.zip',
 '/opt/anaconda3/lib/python3.7',
 '/opt/anaconda3/lib/python3.7/lib-dynload',
 '',
 '/opt/anaconda3/lib/python3.7/site-packages',
 '/opt/anaconda3/lib/python3.7/site-packages/aeosa',
 '/opt/anaconda3/lib/python3.7/site-packages/IPython/extensions',
 '/Users/guangxinsu/.ipython']
```

copyy

```
import pickle
dtrain = xgb.DMatrix(Xtrain,Ytrain)
param = {'silent':True
              ,'obj':'reg:linear'
              ,"subsample":1
              ,"max_depth":3
              ,"eta":0.35
              ,"gamma":0
              ,"lambda":1
              ,"alpha":0
              ,"colsample_bytree":1
              ,"colsample_bylevel":1
              ,"colsample_bynode":1
              ,"nfold":5}
num_round = 110
bst = xgb.train(param,dtrain,num_round)
pickle.dump(bst,open("xgboostonboston.dat","wb"))
import sys
sys.path
```

```
[23:44:59] WARNING: /Users/travis/build/dmlc/xgboost/src/learner.cc:480:
Parameters: { nfold, obj, silent } might not be used.

  This may not be accurate due to some parameters are only used in langua
ge bindings but
  passed down to XGBoost core.  Or some parameters are not used but slip
through this
  verification. Please open an issue if you find above cases.
```

Copy the code above (from beginning) and train the model

```
from sklearn.model_selection import train_test_split as TTS
from sklearn.metrics import mean_squared_error as MSE
import pickle
import xgboost as xgb
import pandas as pd
```

```python
data = pd.read_csv("/Users/guangxinsu/Desktop/penguins_raw.csv")
data.drop(["Comments","Stage","studyName","Sample Number","Region","Individual ID","Date Egg"],inplace=True,axis=1)
data["Culmen Length (mm)"] = data["Culmen Length (mm)"].fillna(data["Culmen Length (mm)"].mean())
data["Culmen Depth (mm)"] = data["Culmen Depth (mm)"].fillna(data["Culmen Depth (mm)"].mean())
data["Flipper Length (mm)"] = data["Flipper Length (mm)"].fillna(data["Flipper Length (mm)"].mean())
data["Body Mass (g)"] = data["Body Mass (g)"].fillna(data["Body Mass (g)"].mean())

data["Delta 15 N (o/oo)"] = data["Delta 15 N (o/oo)"].fillna(data["Delta 15 N (o/oo)"].mean())
data["Delta 13 C (o/oo)"] = data["Body Mass (g)"].fillna(data["Body Mass (g)"].mean())
data.loc[:,"Sex"]=(data["Sex"]=="male").astype("int")
data.loc[:,"Clutch Completion"]=(data["Clutch Completion"]=="Yes").astype("int")
#Check how many values are turned into lists
labels=data["Species"].unique().tolist()
print(labels)
#Convert letters to number
data["Species"]=data["Species"].apply(lambda x: labels.index(x))
labels = data["Species"].unique().tolist()
data.head(300)
#Check how many values are turned into lists
labels=data["Island"].unique().tolist()
#Convert letters to number
data["Island"]=data["Island"].apply(lambda x: labels.index(x))
labels = data["Island"].unique().tolist()
data
x = data.iloc[:,data.columns != "Species"]
y = data.iloc[:,data.columns == "Species"]
Xtrain,Xtest,Ytrain,Ytest = TTS(x,y,test_size=0.3,random_state=420)
dtest = xgb.DMatrix(Xtest,Ytest)
loaded_model = pickle.load(open("xgboostonboston.dat", "rb"))
print("Loaded model from: xgboostonboston.dat")
ypreds = loaded_model.predict(dtest)
ypreds
```

```
In [9]: from sklearn.metrics import mean_squared_error as MSE, r2_score
        MSE(Ytest,ypreds)

Out[9]: 0.054973624965083075

In [10]: r2_score(Ytest,ypreds)

Out[10]: 0.9010163596433597

In [ ]:
```

Actually 0.9010<0.9175, But the generalization performance of the model has improved!

Thanks ☺