Fish Market Dataset

Built for multiple linear regression and multivariate analysis, the Fish Market Dataset contains information about common fish species in market sales. The dataset includes the fish species, weight, length, height, and width.

This dataset is a record of 7 common different fish species in fish market sales. With this dataset, a predictive model can be performed using machine friendly data and estimate the weight of fish can be predicted.

Acknowledgements

Thanks to all who make Kernels using this dataset and also people viewed or download this data.

Inspiration

Multiple linear regression is a fundamental practice for this dataset. Multivariate analysis can also be performed.

Data Set of fishes:

Predict the Weight of Fish

In [ ]:

```
import copy
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, PolynomialFeatures
from sklearn.linear_model import LinearRegression, RidgeCV
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
```

Data is loaded

In [ ]:

```
#The original data can never be added or deleted columns
```

```
original_data = pd.read_csv("fish.csv")
#The data variable is used to make modifications on it
data = copy.deepcopy(original_data)
data.head()
```
linkcode

Check null data items¶

In [ ]:

```
np.sum(data.isnull())
```
There aren't null items

Data is treated (Strings converted to numerical data)

The only non numerical column is the 'Species', so this one is encoded to an integer

In [ ]:

```
original_data["Species"] = 
pd.DataFrame(original_data["Species"]).apply(LabelEncoder().fit_transform)
```
Let's see the linear correlation of the different features

In [ ]:

```
sns.heatmap(data.corr(), annot=True)
```
In [ ]:

```
corr = data.corr()["Weight"].drop("Weight")
print(corr)
```
Error is studied according to the number of degree of the regression

The aim of this cell is to choose the best degree for the regression. So as to achive this goal:

- It iterates over the diferent degrees.
- For each one, the model is trained several times (50 for example). For each training the training and test error is recorded. For each training, is randomly shuffled between training and test data. The purpose of this strategy is to getting a non random error, by averaging the errors. A conclusion can be drawn from the resulting plots:
- 2 may be the most suitable degree for the regression because:
  - It gets the lowest test error.
  - It gets the closest test error to the training one.
  - That's why a balance between bias and variance is found

In [ ]:
```
#Variables for keeping track of errors are initialized
e_train = []
e_test = []
e_train_hist = []
e_test_hist = []
alpha_hist = []
alpha = []
```

```
#Max degree of the regression
max_degree = 5

#No. of training times
training_times = 50

#Iterate over the different degrees
for degree in range(1,max_degree):
    poly = PolynomialFeatures(degree)
    data = copy.deepcopy(original_data)
    y = pd.DataFrame(data["Weight"])
    data = data.drop("Weight", axis = 1)
    x = poly.fit_transform(data)
    for i in range(training_times):
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, 
random_state=np.random.randint(100))
        model = RidgeCV(alphas=[1e-3, 1e-2, 1e-1, 1, 2, 4, 6, 8, 16, 32, 40, 50, 80, 
100, 150, 200, 250, 300, 350, 400])
        model.fit(x_train, y_train)
        #Training error is recorded
        e = np.sqrt(mean_squared_error(y_train, model.predict(x_train)))
        e_train.append(e)
        #Test error is recorded
        e = np.sqrt(mean_squared_error(y_test, model.predict(x_test)))
        e_test.append(e)
        #The alpha hyperparameter is recorded
        alpha.append(model.alpha_)
    #The records of the current degree are saved
    e_train_hist.append(e_train)
    e_train = []
    e_test_hist.append(e_test)
    e_test = []
    alpha_hist.append(alpha)
    alpha = []

#The mean for each degree is calculated
e_train = np.mean(np.array(e_train_hist),axis=1)
e_test = np.mean(np.array(e_test_hist),axis=1)
alpha = np.mean(np.array(alpha_hist),axis=1)

#The errors and alpha record is plotted
plt.plot(range(1,max_degree), e_train, 'o-', label = "train")
plt.plot(range(1,max_degree), e_test, 'o-',label = "test")
plt.legend()
plt.figure()
plt.plot(range(1,max_degree), alpha, 'o-',label = "alpha")
plt.legend()
```

Error is studied according to amount of data

The aim of this cell is to plot the learning curve of the model.
As a result, it can be easily spotted that training a test error end up close one to each other.
In addition, the hyperparameter alpha gets bigger and bigger because overfitting is decreasing for every dataset size iteration.

In [ ]:

```
#Variables for keeping track of errors are initialized
e_train = []
e_test = []
e_train_hist = []
e_test_hist = []
alpha_hist = []
alpha = []

#Max degree of the regression
max_degree = 5

#No. of training times
training_times = 50

#No. of training examples
m = original_data.shape[0]

step = 1

degree = 2

#For every iteration diferent amounts of data are selected
for n_data in range(20, m, step):
    poly = PolynomialFeatures(degree)
    # The model is trained several times with diferent data so as to get a non-random
and more precise error.
    for i in range(training_times):
        data = copy.deepcopy(original_data)
        data = data.iloc[np.random.permutation(np.arange(0,m)),:] #Data is shuffled
        data = data.iloc[1:n_data,:]
        y = pd.DataFrame(data["Weight"])
        data = data.drop("Weight", axis = 1)
        x = poly.fit_transform(data)
        x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3,
random_state=np.random.randint(100))
        model = RidgeCV(alphas=[1e-3, 1e-2, 1e-1, 1, 2, 4, 6, 8, 16, 32, 40, 50, 80,
100, 150, 200, 250, 300, 350, 400])
        model.fit(x_train, y_train)
        #Training error is recorded
        e = np.sqrt(mean_squared_error(y_train, model.predict(x_train)))
        e_train.append(e)
        #Test error is recorded
        e = np.sqrt(mean_squared_error(y_test, model.predict(x_test)))
        e_test.append(e)
        #The alpha hyperparameter is recorded
        alpha.append(model.alpha_)
    #The records of the current degree are saved
    e_train_hist.append(e_train)
    e_train = []
    e_test_hist.append(e_test)
    e_test = []
    alpha_hist.append(alpha)
    alpha = []
```

```python
#The mean for every training examples amount is calculated
e_train = np.mean(np.array(e_train_hist),axis=1)
e_test = np.mean(np.array(e_test_hist),axis=1)
alpha = np.mean(np.array(alpha_hist),axis=1)

#The errors and alpha record are plotted
plt.plot(range(20, m, step), e_train, 'o-', label = "train")
plt.plot(range(20, m, step), e_test, 'o-',label = "test")
plt.legend()
plt.figure()
plt.plot(range(20, m, step), alpha, 'o-',label = "alpha")
plt.legend()
```

Species	Weight	Length1	Length2	Length3	Height	Width
Bream	242	23.2	25.4	30	11.52	4.02
Bream	290	24	26.3	31.2	12.48	4.3056
Bream	340	23.9	26.5	31.1	12.3778	4.6961
Bream	363	26.3	29	33.5	12.73	4.4555
Bream	430	26.5	29	34	12.444	5.134
Bream	450	26.8	29.7	34.7	13.6024	4.9274
Bream	500	26.8	29.7	34.5	14.1795	5.2785
Bream	390	27.6	30	35	12.67	4.69
Bream	450	27.6	30	35.1	14.0049	4.8438
Bream	500	28.5	30.7	36.2	14.2266	4.9594
Bream	475	28.4	31	36.2	14.2628	5.1042
Bream	500	28.7	31	36.2	14.3714	4.8146
Bream	500	29.1	31.5	36.4	13.7592	4.368
Bream	340	29.5	32	37.3	13.9129	5.0728
Bream	600	29.4	32	37.2	14.9544	5.1708
Bream	600	29.4	32	37.2	15.438	5.58
Bream	700	30.4	33	38.3	14.8604	5.2854
Bream	700	30.4	33	38.5	14.938	5.1975
Bream	610	30.9	33.5	38.6	15.633	5.1338
Bream	650	31	33.5	38.7	14.4738	5.7276
Bream	575	31.3	34	39.5	15.1285	5.5695
Bream	685	31.4	34	39.2	15.9936	5.3704
Bream	620	31.5	34.5	39.7	15.5227	5.2801
Bream	680	31.8	35	40.6	15.4686	6.1306
Bream	700	31.9	35	40.5	16.2405	5.589
Bream	725	31.8	35	40.9	16.36	6.0532
Bream	720	32	35	40.6	16.3618	6.09
Bream	714	32.7	36	41.5	16.517	5.8515
Bream	850	32.8	36	41.6	16.8896	6.1984
Bream	1000	33.5	37	42.6	18.957	6.603
Bream	920	35	38.5	44.1	18.0369	6.3063
Bream	955	35	38.5	44	18.084	6.292
Bream	925	36.2	39.5	45.3	18.7542	6.7497
Bream	975	37.4	41	45.9	18.6354	6.7473
Bream	950	38	41	46.5	17.6235	6.3705
Roach	40	12.9	14.1	16.2	4.1472	2.268
Roach	69	16.5	18.2	20.3	5.2983	2.8217
Roach	78	17.5	18.8	21.2	5.5756	2.9044
Roach	87	18.2	19.8	22.2	5.6166	3.1746
Roach	120	18.6	20	22.2	6.216	3.5742
Roach	0	19	20.5	22.8	6.4752	3.3516
Roach	110	19.1	20.8	23.1	6.1677	3.3957
Roach	120	19.4	21	23.7	6.1146	3.2943
Roach	150	20.4	22	24.7	5.8045	3.7544
Roach	145	20.5	22	24.3	6.6339	3.5478
Roach	160	20.5	22.5	25.3	7.0334	3.8203
Roach	140	21	22.5	25	6.55	3.325
Roach	160	21.1	22.5	25	6.4	3.8
Roach	169	22	24	27.2	7.5344	3.8352
Roach	161	22	23.4	26.7	6.9153	3.6312
Roach	200	22.1	23.5	26.8	7.3968	4.1272
Roach	180	23.6	25.2	27.9	7.0866	3.906
Roach	290	24	26	29.2	8.8768	4.4968
Roach	272	25	27	30.6	8.568	4.7736
Roach	390	29.5	31.7	35	9.485	5.355
Whitefish	270	23.6	26	28.7	8.3804	4.2476
Whitefish	270	24.1	26.5	29.3	8.1454	4.2485
Whitefish	306	25.6	28	30.8	8.778	4.6816
Whitefish	540	28.5	31	34	10.744	6.562
Whitefish	800	33.7	36.4	39.6	11.7612	6.5736
Whitefish	1000	37.3	40	43.5	12.354	6.525
Parkki	55	13.5	14.7	16.5	6.8475	2.3265
Parkki	60	14.3	15.5	17.4	6.5772	2.3142
Parkki	90	16.3	17.7	19.8	7.4052	2.673
Parkki	120	17.5	19	21.3	8.3922	2.9181
Parkki	150	18.4	20	22.4	8.8928	3.2928
Parkki	140	19	20.7	23.2	8.5376	3.2944
Parkki	170	19	20.7	23.2	9.396	3.4104
Parkki	145	19.8	21.5	24.1	9.7364	3.1571
Parkki	200	21.2	23	25.8	10.3458	3.6636
Parkki	273	23	25	28	11.088	4.144
Parkki	300	24	26	29	11.368	4.234
Perch	5.9	7.5	8.4	8.8	2.112	1.408
Perch	32	12.5	13.7	14.7	3.528	1.9992
Perch	40	13.8	15	16	3.824	2.432
Perch	51.5	15	16.2	17.2	4.5924	2.6316
Perch	70	15.7	17.4	18.5	4.588	2.9415
Perch	100	16.2	18	19.2	5.2224	3.3216
Perch	78	16.8	18.7	19.4	5.1992	3.1234
Perch	80	17.2	19	20.2	5.6358	3.0502
Perch	85	17.8	19.6	20.8	5.1376	3.0368
Perch	85	18.2	20	21	5.082	2.772
Perch	110	19	21	22.5	5.6925	3.555
Perch	115	19	21	22.5	5.9175	3.3075
Perch	125	19	21	22.5	5.6925	3.6675
Perch	130	19.3	21.3	22.8	6.384	3.534
Perch	120	20	22	23.5	6.11	3.4075
Perch	120	20	22	23.5	5.64	3.525
Perch	130	20	22	23.5	6.11	3.525
Perch	135	20	22	23.5	5.875	3.525
Perch	110	20	22	23.5	5.5225	3.995
Perch	130	20.5	22.5	24	5.856	3.624
Perch	150	20.5	22.5	24	6.792	3.624
Perch	145	20.7	22.7	24.2	5.9532	3.63
Perch	150	21	23	24.5	5.2185	3.626
Perch	170	21.5	23.5	25	6.275	3.725
Perch	225	22	24	25.5	7.293	3.723
Perch	145	22	24	25.5	6.375	3.825

LOGS:

Notebook | Data | Logs | Comments (0)

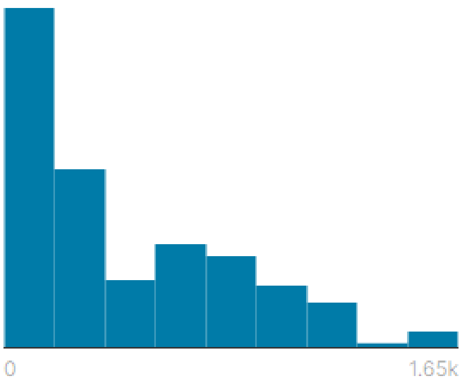
Other (68) 43%

Unique 7

Most Common Perch 35%

Weight

weight of fish in Gram g



Valid	159	100%
Mismatched	0	0%
Missing	0	0%
Mean	398	
Std. Deviation	357	
Quantiles	0	Min
	120	25%
	273	50%
	650	75%
	1.65k	Max