1-ExploringTheAutomobileMpgDataset

October 16, 2021

[1]:
```
import sklearn

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import datetime
```

[3]:
```
print(sklearn.__version__)
```

```
0.24.1
```

[4]:
```
print(np.__version__)
```

```
1.20.1
```

[5]:
```
print(pd.__version__)
```

```
1.2.4
```

[9]:
```
'''
This is a dataset that contains a number of different automobile features,␣
 ↪which we use to predict how many
miles that automobile runs per gallon of fuel.
'''
automobile_df=pd.read_csv('data/auto-mpg.csv')
```

[11]:
```
'''
 If you want to view a sample of records in your data frame so that you can␣
 ↪explore the dataset,
 you can call the df.sample function. The parameter 5 indicates that five␣
 ↪records should be displayed.
 And here are five records chosen at random from our dataset.
'''
automobile_df.sample(5)
```

[11]:
```
      mpg  cylinders  displacement horsepower  weight  acceleration  \
167  29.0          4          97.0         75    2171          16.0
177  23.0          4         115.0         95    2694          15.0
339  26.6          4         151.0         84    2635          16.4
225  17.5          6         250.0        110    3520          16.4
121  15.0          8         318.0        150    3399          11.0

     model year  origin            car name
167          75       3      toyota corolla
177          75       2          audi 100ls
339          81       1       buick skylark
225          77       1  chevrolet concours
121          73       1    dodge dart custom
```

[12]:
```
'''
The columns at the very right make up the features of our machine learning␣
↪model.

The regression models that we're going to build will use these columns in order␣
↪to make predictions
about the miles per gallon for that car.

There are features such as the number of cylinders the car has, the␣
↪displacement of the car from the bottom,
the horsepower, the weight, the acceleration, model, year, the origin of the␣
↪car, and the name of the car.

The first column off to the left, the mpg column, gives us the miles per gallon␣
↪for that particular car,
and this is what we'll try and predict using regression.
'''
automobile_df.sample(5)
```

[12]:
```
      mpg  cylinders  displacement horsepower  weight  acceleration  \
289  16.9          8         350.0        155    4360          14.9
339  26.6          4         151.0         84    2635          16.4
4    17.0          8         302.0        140    3449          10.5
1    15.0          8         350.0        165    3693          11.5
276  21.6          4         121.0        115    2795          15.7

     model year  origin                 car name
289          79       1  buick estate wagon (sw)
339          81       1            buick skylark
4            70       1              ford torino
1            70       1        buick skylark 320
276          78       2               saab 99gle
```

```
[13]: '''

The shape variable for any dataset gives us how many records are in the dataset␣
 ↪and how many columns.
So we have 398 records and 9 columns of data.


These 9 columns include 8 columns of features and 1 column that forms our␣
 ↪machine learning target,
the value we are trying to predict, the mpg.
'''
automobile_df.shape
```

```
[13]: (398, 9)
```

```
[14]: '''

Now, datasets that we work with in the real world often contain missing fields␣
 ↪or values, and these records need to be handled and cleaned in some way.
This is part of the data wrangling or preprocessing that will apply to our data.


Now this particular dataset contains question marks(?) in place of missing␣
 ↪fields;
we'll replace all of those question marks with NaNs, or not a numbers.
Call the automobile_df.replace function in order to perform this replacement.
'''

automobile_df=automobile_df.replace("?",np.nan)
```

```
[15]: '''

And once you have NaNs in place of missing values, it's very easy to clean your␣
 ↪data frame.
The drop any function on your pandas DataFrame will simply drop all of those␣
 ↪records which have any fields missing.
'''

automobile_df=automobile_df.dropna()
```

```
[16]: '''
And if you take a look at the shape of your data frame now, you see that we␣
 ↪have 392 records.
We originally had 398 records, and now it's 392.
6 records had missing fields, they were dropped.
'''
```

```
automobile_df.shape
```

[16]:
```
(392, 9)
```

[17]:
```
'''
While we are building up the features for our linear regression model,
it's pretty clear that the origin of the car and the name of the car has no␣
↪impact on its mileage.
This is something that we can determine just by a cursory look at the columns␣
↪in our data frame,
so go ahead and drop the origin and car name columns in place.

These features, we know by using our common sense and logic, have no predictive␣
↪powers.
'''
automobile_df.drop(['origin','car name'],axis=1,inplace=True)
```

```
/Users/subhasish/opt/anaconda3/envs/ML/lib/python3.8/site-
packages/pandas/core/frame.py:4308: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  return super().drop(
```

[20]:
```
'''

  I'm going to call automobile_df.sample to sample five records from our data␣
↪frames.
  And here are the features that we're going to work with: cylinders,␣
↪displacement, horsepower, weight,
  acceleration, and model year, and the miles per gallon is our target, what␣
↪you're going to try and predict.
'''
automobile_df.sample(5)
from IPython.display import Image
Image('/Users/subhasish/Documents/APPLE/SUBHASISH/Development/GIT/Interstellar/
↪SB-AI-DEV/ML/SB/LinerRegression/Images/2021-10-16_01-21-45.jpg')
```

[20]:

```
automobile_df.sample(5)
```

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year
104	12.0	8	400.0	167	4906	12.5	73
270	21.1	4	134.0	95	2515	14.8	78
190	14.5	8	351.0	152	4215	12.8	76
373	24.0	4	140.0	92	2865	16.4	82
318	29.8	4	134.0	90	2711	15.5	80

[21]:
```
'''

 Now this dataset is from the '90s, and you can see that all of the model years␣
↪are basically 1973, 78, 82, and so on.
 Now the model year by itself is just an object. Let's make this useful by␣
↪converting this to be the age of the car.

 It's quite possible that we don't know for sure that the age of the car might␣
↪have some impact on its mileage.
 Before we get to the age, let's convert the year to its full form, 1973, 1980,␣
↪and so on,

 so I'm going to prepend the string 19 to the model year. So 19 + model year as␣
↪string,
 will give us the resultant model year.

 Assign this new format to the model year column and let's sample our data␣
↪frame and take a look at the result.
 The model year now has the full year, 1982, 1972, and so on.
'''
automobile_df['model year'] = '19' + automobile_df['model year'].astype(str)
```

```
<ipython-input-21-bbbf1278e719>:15: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  automobile_df['model year'] = '19' + automobile_df['model year'].astype(str)
```

[22]:
```
'''

 Assign this new format to the model year column and let's sample our data␣
 ↪frame and take a look at the result.
 The model year now has the full year, 1982, 1972, and so on.


 Now with this, we can calculate how old this particular car is.
'''
automobile_df.sample(5)
```

[22]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	model year
133	16.0	6	250.0	100	3781	17.0	1974
291	19.2	8	267.0	125	3605	15.0	1979
383	38.0	4	91.0	67	1965	15.0	1982
390	32.0	4	144.0	96	2665	13.9	1982
113	21.0	6	155.0	107	2472	14.0	1973

[23]:
```
'''
 You can choose any reference date to calculate the age, as long as it's later␣
 ↪than the last year that the
 car was made.

 In order to keep things simple, we'll calculate each field by subtracting from␣
 ↪the current year.

 I'll use the datetime library to access the current year we're at; this year␣
 ↪will be in numeric form.

 And I'll convert the data in the model year column to numeric form by calling␣
 ↪pd.to_numeric.

 The result will be a number that will represent the age of a particular car.

'''

automobile_df['age']=datetime.datetime.now().year-pd.
 ↪to_numeric(automobile_df['model year'])
```

[24]:
```
'''
 Go ahead and drop the original model year field, we no longer needed because␣
 ↪we have the age column.
'''
automobile_df.drop(['model year'], axis=1, inplace=True)
```

[25]:
```
'''
 Let's view a sample of this data frame.
 Once again, you can see we now have each column which tells you how old this␣
 ↪particular car is.

 The absolute values for these ages don't really matter so much.
 It is their relative values that are more significant.
 If a car is older than another, it's possible that its mileage goes down.
'''

automobile_df.sample(5)
```

[25]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	age
234	24.5	4	151.0	88	2740	16.0	44
238	33.5	4	98.0	83	2075	15.9	44
199	20.0	6	225.0	100	3651	17.7	45
100	18.0	6	250.0	88	3021	16.5	48
246	32.8	4	78.0	52	1985	19.4	43

[26]:
```
'''

If you're building and training a machine learning model, all of the inputs to␣
 ↪your model need to be numeric.

Take a look at the data types of the different columns.
You'll find that all of them are numeric except for one, that is the horsepower␣
 ↪column.

The horsepower is a numeric field, but its data type in our data frame is␣
 ↪object.
We need to fix this. This is very easily done using pandas.





'''
automobile_df.dtypes
```

[26]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight            int64
acceleration    float64
age               int64
dtype: object
```

[27]:

```
'''
Simply call pd.to_numeric to convert horsepower to a numeric field and assign␣
↪it to the horsepower column once again.

'''
automobile_df['horsepower']=pd.
↪to_numeric(automobile_df['horsepower'],errors='coerce')
```

[28]:

```
'''
Let's now call describe on our dataset in order to get a few statistical bits␣
↪of information about all of our
numerical features.

You can see that all of the features in our dataset are now numeric.

We have mean value, standard deviations, and the different percentiles␣
↪displayed here.

The describe function in pandas is an easy way for you to get a quick feel for␣
↪your numeric data.
'''

automobile_df.describe()
```

[28]:

```
              mpg   cylinders  displacement  horsepower       weight  \
count  392.000000  392.000000    392.000000  392.000000   392.000000
mean    23.445918    5.471939    194.411990  104.469388  2977.584184
std      7.805007    1.705783    104.644004   38.491160   849.402560
min      9.000000    3.000000     68.000000   46.000000  1613.000000
25%     17.000000    4.000000    105.000000   75.000000  2225.250000
50%     22.750000    4.000000    151.000000   93.500000  2803.500000
75%     29.000000    8.000000    275.750000  126.000000  3614.750000
max     46.600000    8.000000    455.000000  230.000000  5140.000000

       acceleration         age
count    392.000000  392.000000
mean      15.541327   45.020408
std        2.758864    3.683737
min        8.000000   39.000000
25%       13.775000   42.000000
50%       15.500000   45.000000
75%       17.025000   48.000000
max       24.800000   51.000000
```

[ ]: