Lab 3.3 - Student Notebook

Overview

This lab does not continue the healthcare-provider scenario. Instead, you will work with data from an automobile dataset.

In this lab, you will:

- Encode ordinal categorical data
- Encode non-ordinal categorical data

About this dataset

This dataset consists of three types of entities:

1. The specification of an automobile in terms of various characteristics
2. Its assigned insurance risk rating
3. Its normalized losses in use compared to other cars

The second rating corresponds to the degree to which the automobile is riskier than its price indicates. Cars are initially assigned a risk factor symbol that's associated with its price. Then, if it's riskier (or less risky), this symbol is adjusted by moving it up (or down) the scale. Actuarians call this process *symboling*. A value of *+3* indicates that the car is risky. A value of *-3* indicates that the car is probably safe.

The third factor is the relative average loss payment per insured vehicle year. This value is normalized for all cars within a particular size classification (two-door small, station wagons, sports or speciality, and others). It represents the average loss per car per year.

Note: Several attributes in the database could be used as a *class* attribute.

Attribute information

Attribute: Attribute Range

1. symboling: -3, -2, -1, 0, 1, 2, 3.
2. normalized-losses: continuous from 65 to 256.
3. fuel-type: diesel, gas.

4. aspiration: std, turbo.
5. num-of-doors: four, two.
6. body-style: hardtop, wagon, sedan, hatchback, convertible.
7. drive-wheels: 4wd, fwd, rwd.
8. engine-location: front, rear.
9. wheel-base: continuous from 86.6 120.9.
10. length: continuous from 141.1 to 208.1.
11. width: continuous from 60.3 to 72.3.
12. height: continuous from 47.8 to 59.8.
13. curb-weight: continuous from 1488 to 4066.
14. engine-type: dohc, dohcv, l, ohc, ohcf, ohcv, rotor.
15. num-of-cylinders: eight, five, four, six, three, twelve, two.
16. engine-size: continuous from 61 to 326.
17. fuel-system: 1bbl, 2bbl, 4bbl, idi, mfi, mpfi, spdi, spfi.
18. bore: continuous from 2.54 to 3.94.
19. stroke: continuous from 2.07 to 4.17.
20. compression-ratio: continuous from 7 to 23.
21. horsepower: continuous from 48 to 288.
22. peak-rpm: continuous from 4150 to 6600.
23. city-mpg: continuous from 13 to 49.
24. highway-mpg: continuous from 16 to 54.
25. price: continuous from 5118 to 45400.

Dataset attributions

This dataset was obtained from: Dua, D. and Graff, C. (2019). UCI Machine Learning Repository (http://archive.ics.uci.edu/ml). Irvine, CA: University of California, School of Information and Computer Science.

Step 1: Importing and exploring the data

You will start by examining the data in the dataset.

To get the most out of this lab, read the instructions and code before you run the cells. Take time to experiment!

Start by importing the pandas package and setting some default display options.

In [51]:

```
!conda update -y numexpr
import pandas as pd

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

```
Retrieving notices: done
Channels:
 - conda-forge
 - nvidia
 - pytorch
Platform: linux-64
Collecting package metadata (repodata.json): done
Solving environment: done


==> WARNING: A newer version of conda exists. <==
    current version: 25.3.0
    latest version: 25.5.1

Please update conda by running

    $ conda update -n base -c conda-forge conda



## Package Plan ##

  environment location: /home/ec2-user/anaconda3/envs/python3

  added / updated specs:
    - numexpr


The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    ca-certificates-2025.7.14  |       hbd8a1cb_0         152 KB  conda-forge
    certifi-2025.7.14          |     pyhd8ed1ab_0         156 KB  conda-forge
    openssl-3.5.1              |       h7b32b05_0         3.0 MB  conda-forge
    ------------------------------------------------------------
                                           Total:         3.3 MB

The following packages will be UPDATED:

  ca-certificates                      2025.6.15-hbd8a1cb_0 --> 2025.7.14-hbd8a
1cb_0
  certifi                            2025.6.15-pyhd8ed1ab_0 --> 2025.7.14-pyhd8
ed1ab_0
  openssl                                  3.5.0-h7b32b05_1 --> 3.5.1-h7b32b0
5_0



Downloading and Extracting Packages:
openssl-3.5.1        | 3.0 MB    |                                        |   0%
certifi-2025.7.14    | 156 KB    |                                        |   0%

ca-certificates-2025 | 152 KB    |                                        |   0%
```

```
ca-certificates-2025 | 152 KB    | ##################################### | 100%
openssl-3.5.1        | 3.0 MB    | ###########4                          |  31%

ca-certificates-2025 | 152 KB    | ##################################### | 100%

ca-certificates-2025 | 152 KB    | ##################################### | 100%
certifi-2025.7.14    | 156 KB    | ##################################### | 100%



Preparing transaction: done
Verifying transaction: done
Executing transaction: done
```

Next, load the dataset into a pandas DataFrame.

The data doesn't contain a header, so you will define those column names in a variable that's named `col_names` to the attributes listed in the dataset description.

In [53]:

```
url = "imports-85.csv"
col_names=['symboling','normalized-losses','fuel-type','aspiration','num-of-dc
                                    'length','width','height','curb-weight','e
                                    'fuel-system','bore','stroke','compression

df_car = pd.read_csv(url,sep=',',names = col_names ,na_values="?",  header=Non
```

In [27]:

```
df_car.dropna(subset=['num-of-doors', 'drive-wheels', 'num-of-cylinders', 'asp
df_challenge = df_car[['num-of-doors', 'drive-wheels', 'num-of-cylinders', 'as
doors_map = {'two': 2, 'four': 4}
cylinders_map = {'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6, 'eight'
aspiration_map = {'std': 0, 'turbo': 1}
df_challenge['num-of-doors'] = df_challenge['num-of-doors'].map(doors_map)
df_challenge['num-of-cylinders'] = df_challenge['num-of-cylinders'].map(cylinc
df_challenge['aspiration'] = df_challenge['aspiration'].map(aspiration_map)
df_challenge = pd.get_dummies(df_challenge, columns=['drive-wheels', 'body-sty
print(df_challenge.head())
```

```
   num-of-doors  num-of-cylinders  aspiration  drive-wheels_fwd  drive-wheels_r
wd  body-style_hardtop  body-style_hatchback  body-style_sedan  body-style_wago
n  fuel-type_gas
0             2                 4           0             False             Tr
ue                False                  False             False             Fals
e            True
1             2                 4           0             False             Tr
ue                False                  False             False             Fals
e            True
2             2                 6           0             False             Tr
ue                False                   True             False             Fals
e            True
3             4                 4           0              True             Fal
se                False                  False              True             Fals
e            True
4             4                 5           0             False             Fal
se                False                  False              True             Fals
e            True
```

First, to see the number of rows (instances) and columns (features), you will use `shape`.

In [28]:
```python
df_car.shape
```

Out[28]:
```
(203, 25)
```

Next, examine the data by using the `head` method.

In [29]:
```python
df_car.head(5)
```

Out[29]:

	symboling	normalized-losses	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location
0	3	NaN	gas	std	two	convertible	rwd	front
1	3	NaN	gas	std	two	convertible	rwd	front
2	1	NaN	gas	std	two	hatchback	rwd	front
3	2	164.0	gas	std	four	sedan	fwd	front
4	2	164.0	gas	std	four	sedan	4wd	front

There are 25 columns. Some of the columns have numerical values, but many of them contain text.

To display information about the columns, use the `info` method.

In [30]:
```python
df_car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 203 entries, 0 to 204
Data columns (total 25 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   symboling          203 non-null    int64  
 1   normalized-losses  163 non-null    float64
 2   fuel-type          203 non-null    object 
 3   aspiration         203 non-null    object 
 4   num-of-doors       203 non-null    object 
 5   body-style         203 non-null    object 
 6   drive-wheels       203 non-null    object 
 7   engine-location    203 non-null    object 
 8   wheel-base         203 non-null    float64
 9   length             203 non-null    float64
 10  width              203 non-null    float64
 11  height             203 non-null    float64
 12  curb-weight        203 non-null    int64  
 13  engine-type        203 non-null    object 
 14  num-of-cylinders   203 non-null    object 
 15  engine-size        203 non-null    int64  
 16  fuel-system        203 non-null    object 
 17  bore               199 non-null    float64
 18  stroke             199 non-null    float64
 19  compression-ratio  203 non-null    float64
 20  horsepower         201 non-null    float64
 21  peak-rpm           201 non-null    float64
 22  city-mpg           203 non-null    int64  
 23  highway-mpg        203 non-null    int64  
 24  price              199 non-null    float64
dtypes: float64(11), int64(5), object(9)
memory usage: 41.2+ KB
```

To make it easier to view the dataset when you start encoding, drop the columns that you won't use.

In [31]:
```
df_car.columns
```

Out[31]:
```
Index(['symboling', 'normalized-losses', 'fuel-type', 'aspiration', 'num-of-d
oors', 'body-style', 'drive-wheels', 'engine-location', 'wheel-base', 'lengt
h', 'width', 'height', 'curb-weight', 'engine-type', 'num-of-cylinders', 'eng
ine-size', 'fuel-system', 'bore', 'stroke', 'compression-ratio', 'horsepowe
r', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price'], dtype='object')
```

In [32]:
```
df_car = df_car[[ 'aspiration', 'num-of-doors',  'drive-wheels',  'num-of-cyli
```

You now have four columns. These columns all contain text values.

In [33]:
```
df_car.head()
```

Out[33]:

	aspiration	num-of-doors	drive-wheels	num-of-cylinders
0	std	two	rwd	four
1	std	two	rwd	four
2	std	two	rwd	six
3	std	four	fwd	four
4	std	four	4wd	five

Most machine learning algorithms require inputs that are numerical values.

- The **num-of-cylinders** and **num-of-doors** features have an ordinal value. You could convert the values of these features into their numerical counterparts.
- However, **aspiration** and **drive-wheels** don't have an ordinal value. These features must be converted differently.

You will explore the ordinal features first.

Step 2: Encoding ordinal features

In this step, you will use a mapper function to convert the ordinal features into ordered numerical values.

Start by getting the new column types from the DataFrame:

In [34]:
```
df_car.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 203 entries, 0 to 204
Data columns (total 4 columns):
 #   Column            Non-Null Count  Dtype
---  ------            --------------  -----
 0   aspiration        203 non-null    object
 1   num-of-doors      203 non-null    object
 2   drive-wheels      203 non-null    object
 3   num-of-cylinders  203 non-null    object
dtypes: object(4)
memory usage: 7.9+ KB
```

First, determine what values the ordinal columns contain.

Starting with the **num-of-doors** feature, you can use `value_counts` to discover the values.

In [35]:
```
df_car['num-of-doors'].value_counts()
```

Out[35]:
```
num-of-doors
four    114
two      89
Name: count, dtype: int64
```

This feature only has two values: *four* and *two*. You can create a simple mapper that contains a dictionary:

In [36]:
```
door_mapper = {"two": 2,
              "four": 4}
```

You can then use the `replace` method from pandas to generate a new numerical column based on the **num-of-doors** column.

In [38]:
```
door_mapper = {'two': 2, 'four': 4}
df_car['doors'] = df_car['num-of-doors'].map(door_mapper)
```

When you display the DataFrame, you should see the new column on the right. It contains a numerical representation of the number of doors.

In [39]:
```
df_car.head()
```

Out[39]:

	aspiration	num-of-doors	drive-wheels	num-of-cylinders	doors
0	std	two	rwd	four	2
1	std	two	rwd	four	2
2	std	two	rwd	six	2
3	std	four	fwd	four	4
4	std	four	4wd	five	4

Repeat the process with the **num-of-cylinders** column.

First, get the values.

In [40]:
```
df_car['num-of-cylinders'].value_counts()
```

Out[40]:
```
num-of-cylinders
four      157
six        24
five       11
eight       5
two         4
three       1
twelve      1
Name: count, dtype: int64
```

Next, create the mapper.

In [41]:
```
cylinder_mapper = {"two":2,
                   "three":3,
                   "four":4,
                   "five":5,
                   "six":6,
                   "eight":8,
                   "twelve":12}
```

Apply the mapper by using the `replace` method.

In [42]:
```
cylinder_mapper = {'two': 2, 'three': 3, 'four': 4, 'five': 5, 'six': 6, 'eigh
df_car['cylinders'] = df_car['num-of-cylinders'].map(cylinder_mapper)
```

In [43]:
```
df_car.head()
```

Out[43]:

	aspiration	num-of-doors	drive-wheels	num-of-cylinders	doors	cylinders
0	std	two	rwd	four	2	4
1	std	two	rwd	four	2	4
2	std	two	rwd	six	2	6
3	std	four	fwd	four	4	4
4	std	four	4wd	five	4	5

For more information about the `replace` method, see pandas.DataFrame.replace in the pandas documentation.

Step 3: Encoding non-ordinal categorical data

In this step, you will encode non-ordinal data by using the `get_dummies` method from pandas.

The two remaining features are not ordinal.

According to the attribute description, the following values are possible:

- aspiration: std, turbo.
- drive-wheels: 4wd, fwd, rwd.

You might think that the correct strategy is to convert these values into numerical values. For example, consider the **drive-wheels** feature. You could use *4wd = 1*, *fwd = 2*, and *rwd = 3*. However, *fwd* isn't less than *rwd*. These values don't have an order, but you just introduced an order to them by assigning these numerical values.

The correct strategy is to convert these values into *binary features* for each value in the original feature. This process is often called *one-hot encoding* in machine learning, or *dummying* in statistics.

pandas provides a `get_dummies` method, which converts the data into binary features. For more information, see pandas.get_dummies in the pandas documentation.

According to the attribute description, **drive-wheels** has three possible values.

In [44]:
```
df_car['drive-wheels'].value_counts()
```

Out[44]:
```
drive-wheels
fwd    118
rwd     76
4wd      9
Name: count, dtype: int64
```

Use the `get_dummies` method to add new binary features to the DataFrame.

In [45]:
```
df_car = pd.get_dummies(df_car,columns=['drive-wheels'])
```

In [46]:
```
df_car.head()
```

Out[46]:

	aspiration	num-of-doors	num-of-cylinders	doors	cylinders	drive-wheels_4wd	drive-wheels_fwd	whe
0	std	two	four	2	4	False	False	
1	std	two	four	2	4	False	False	
2	std	two	six	2	6	False	False	
3	std	four	four	4	4	False	True	
4	std	four	five	4	5	True	False	

When you examine the dataset, you should see three new columns on the right:

- **drive-wheels_4wd**
- **drive-wheels_fwd**
- **drive-wheels_rwd**

The encoding was straightforward. If the value in the **drive-wheels** column is *4wd*, then a *1* is the value in the **drive-wheels_4wd** column. A *0* is the value for the other columns that were generated. If the value in the **drive-wheels** column is *fwd*, then a *1* is the value in the **drive-wheels_fwd** column, and so on.

These binary features enable you to express the information in a numerical way, without implying any order.

Examine the final column that you will encode.

The data in the **aspiration** column only has two values: *std* and *turbo*. You could encode this column into two binary features. However, you could also ignore the *std* value and record whether it's *turbo* or not. To do this, you would still use the `get_dummies` method, but specify `drop_first` as *True*.

In [47]:
```
df_car['aspiration'].value_counts()
```

Out[47]:
```
aspiration
std      167
turbo     36
Name: count, dtype: int64
```

In [48]:
```
df_car = pd.get_dummies(df_car,columns=['aspiration'], drop_first=True)
```

In [49]:
```
df_car.head()
```

Out[49]:

	num-of-doors	num-of-cylinders	doors	cylinders	drive-wheels_4wd	drive-wheels_fwd	drive-wheels_rwd	asp
0	two	four	2	4	False	False	True	
1	two	four	2	4	False	False	True	
2	two	six	2	6	False	False	True	
3	four	four	4	4	False	True	False	
4	four	five	4	5	True	False	False	

Challenge task: Go back to the beginning of this lab, and add other columns to the dataset. How would you encode the values of each column? Update the code to include some of the other features.

Congratulations!

You have completed this lab, and you can now end the lab by following the lab guide instructions.

In [ ]:
```

```