

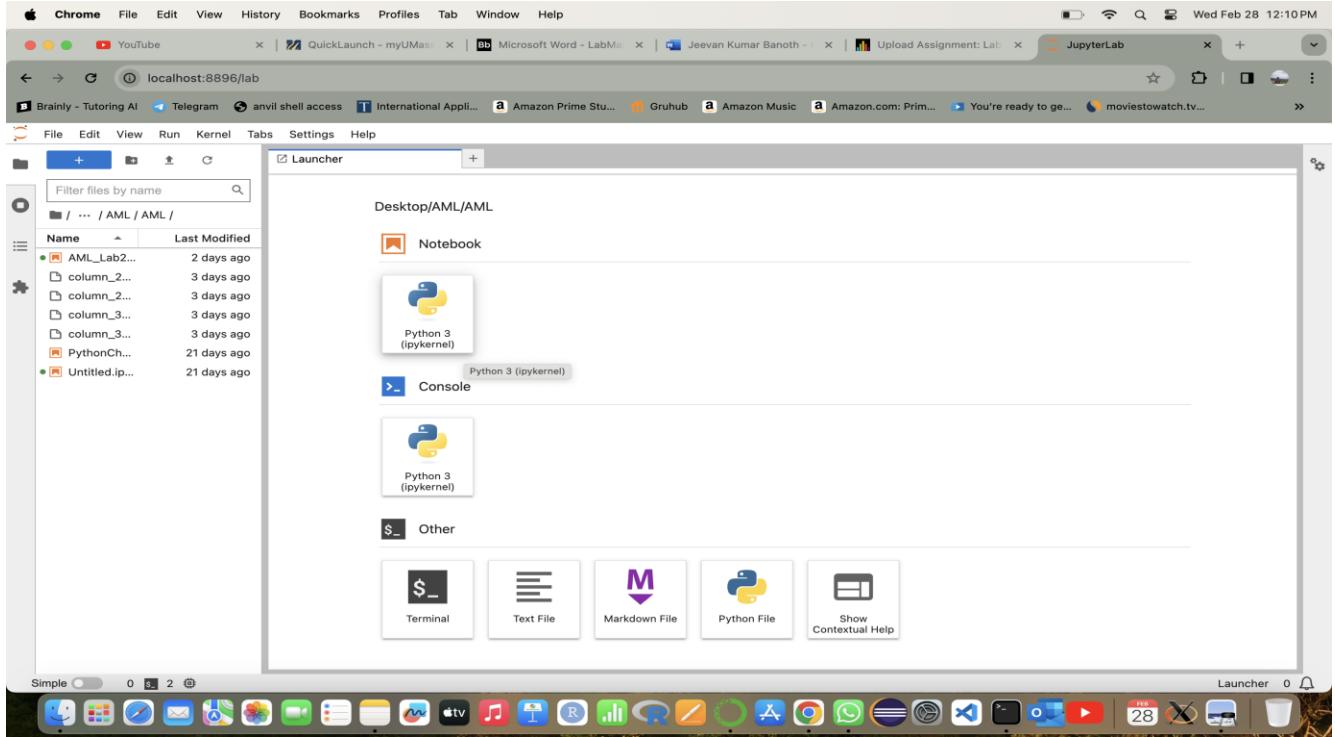
**Jeevan Kumar Banoth - 02105145**

**Advanced machine learning**

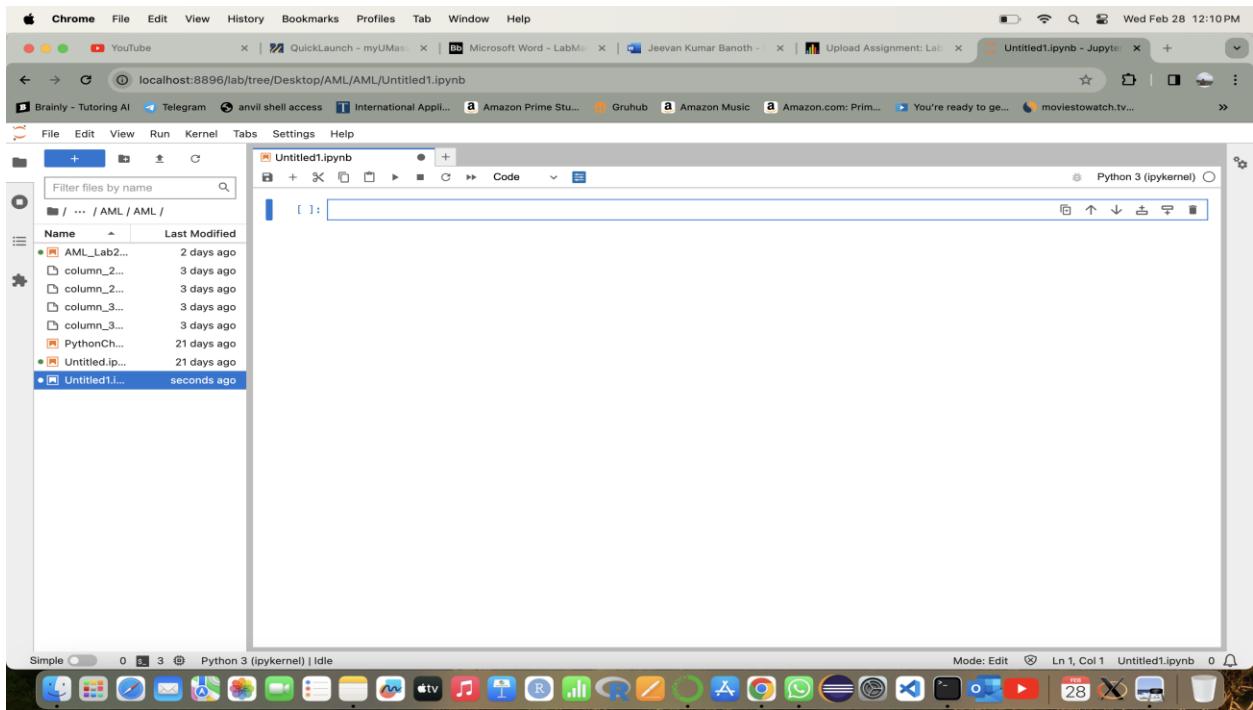
**Homework –3:**

**Lab\_3\_Manual\_Encoding Categorical Data**

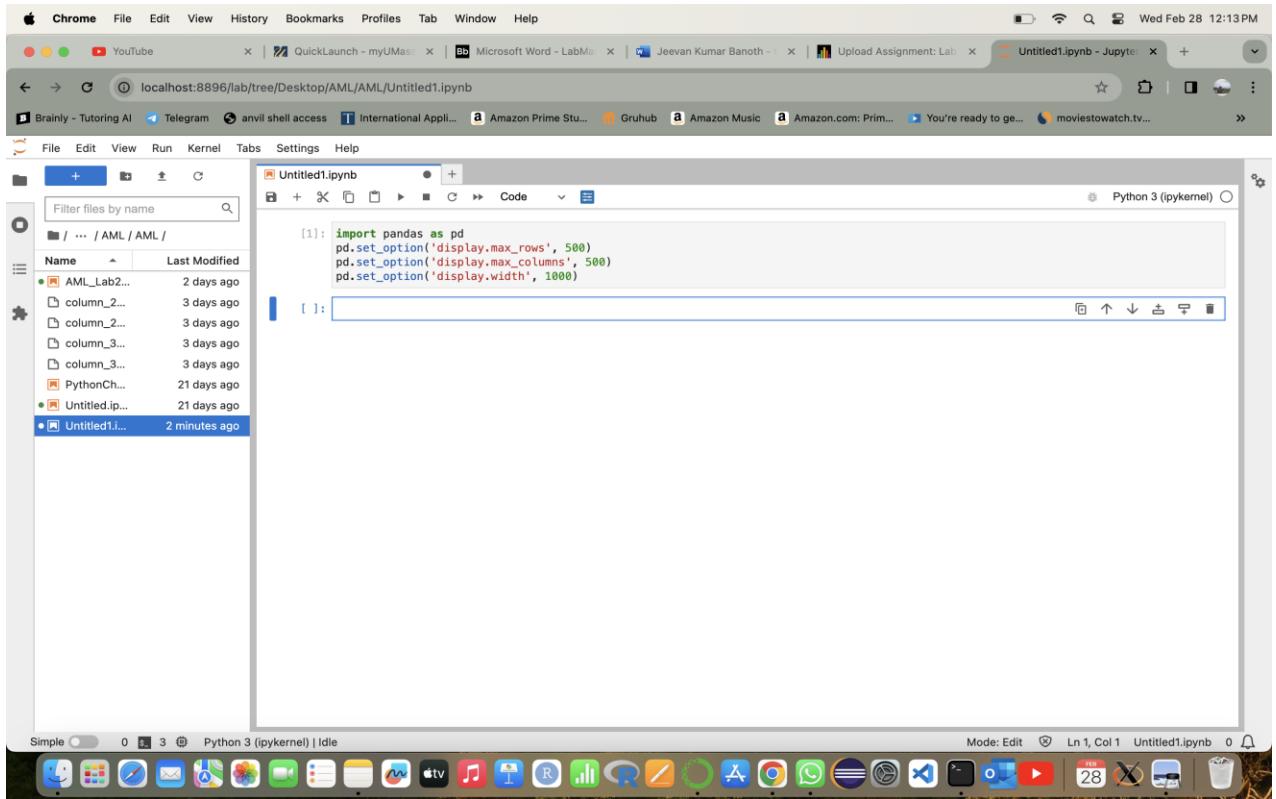
- First, we need to open the anaconda application and then open Jupyter lab, It looks like below.



- Then click on the python 3 ipykernel in notebook, so it will redirect to chrome and will open a notebook as follows.



## Step 1: Importing and exploring the data:



The screenshot shows a Jupyter Notebook interface running in a Chrome browser window. The notebook is titled 'Untitled1.ipynb'. In the code cell [1], the following Python code is written:

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

The file tree on the left shows several files in the 'AML' directory, including 'AML\_Lab2.ipynb', 'column\_2.ipynb', 'column\_3.ipynb', 'PythonCh... .ipynb', 'Untitled.ipynb', and 'Untitled1.ipynb'. The status bar at the bottom indicates 'Mode: Edit' and 'Ln 1, Col 1 Untitled1.ipynb'.

- By using the keyboard, we have written the code for importing the panda's package and setting up some default package options for displaying.
- Now we are loading the dataset into the Panda's Data frame by assigning the dataset containing file location to it in the place of url.
- And we are defining the col\_names that exist in the dataset by looking at attributes listed in the dataset description.

- Now, we will see the number of rows (instances) and columns (features) in the dataset by using the shape () command.
- Next, we are using the head command to examine the data.
- There are 5 columns (0-5), some of them contain numerical data, and some of them are text, Nan values too.
- Below is the image of using shape and head commands.

The screenshot shows a Jupyter Notebook interface running on a Mac OS X desktop. The notebook file is titled 'Untitled1.ipynb'. In the code cell [12], the user has run the following Python code:

```
[12]: url = "http://www.automobile.com/imports-85.data"
df_car = pd.read_csv(url, names=col_names, na_values="?", header=None)
```

In the code cell [13], the user has run:

```
[13]: df_car.shape
```

The output is:

```
[13]: (205, 26)
```

In the code cell [14], the user has run:

```
[14]: df_car.head(5)
```

The output is a DataFrame showing the first 5 rows of the dataset:

|   | symboling | normalized-losses | make        | fuel-type | aspiration | num-of-doors | body-style  | drive-wheels | engine-location | wheel-base | length | width | height | curb-weight | engine-type | num-of-cylinders |
|---|-----------|-------------------|-------------|-----------|------------|--------------|-------------|--------------|-----------------|------------|--------|-------|--------|-------------|-------------|------------------|
| 0 | 3         | NaN               | alfa-romero | gas       | std        | two          | convertible | rwd          | front           | 88.6       | 168.8  | 64.1  | 48.8   | 2548        | dohc        | four             |
| 1 | 3         | NaN               | alfa-romero | gas       | std        | two          | convertible | rwd          | front           | 88.6       | 168.8  | 64.1  | 48.8   | 2548        | dohc        | four             |
| 2 | 1         | NaN               | alfa-romero | gas       | std        | two          | hatchback   | rwd          | front           | 94.5       | 171.2  | 65.5  | 52.4   | 2823        | ohcv        | six              |
| 3 | 2         | 164.0             | audi        | gas       | std        | four         | sedan       | fwd          | front           | 99.8       | 176.6  | 66.2  | 54.3   | 2337        | ohc         | four             |
| 4 | 2         | 164.0             | audi        | gas       | std        | four         | sedan       | 4wd          | front           | 99.4       | 176.6  | 66.4  | 54.3   | 2824        | ohc         | five             |

- To display the information about columns we are using info () command here.
- This command gives the information about column names, Dtypes, etc

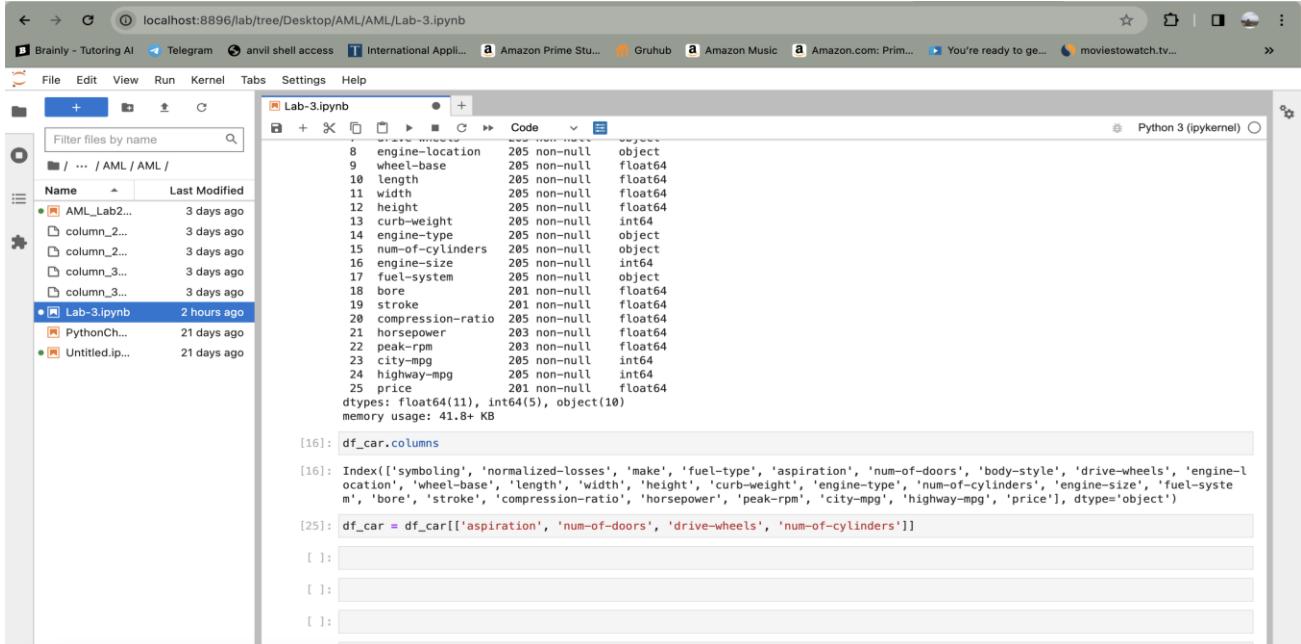
The screenshot shows a Jupyter Notebook interface running on a Mac OS X desktop. The notebook file is titled 'Untitled1.ipynb'. In the code cell [15], the user has run:

```
[15]: df_car.info()
```

The output is:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 26 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   symboling        205 non-null    int64  
 1   normalized-losses 164 non-null    float64 
 2   make             205 non-null    object  
 3   fuel-type        205 non-null    object  
 4   aspiration       205 non-null    object  
 5   num-of-doors     203 non-null    object  
 6   body-style       205 non-null    object  
 7   drive-wheels     205 non-null    object  
 8   engine-location  205 non-null    object  
 9   wheel-base       205 non-null    float64 
 10  length           205 non-null    float64 
 11  width            205 non-null    float64 
 12  height           205 non-null    float64 
 13  curb-weight      205 non-null    int64  
 14  engine-type      205 non-null    object  
 15  num-of-cylinders 205 non-null    object  
 16  engine-size      205 non-null    int64  
 17  fuel-system      205 non-null    object  
 18  bore             205 non-null    float64 
 19  stroke           201 non-null    float64 
 20  compression-ratio 205 non-null    float64 
 21  horsepower        203 non-null    float64 
 22  peak-rpm          203 non-null    float64 
 23  city-mpg          205 non-null    int64  
 24  highway-mpg        205 non-null    int64  
 25  price             201 non-null    float64 
dtypes: float64(11), int64(5), object(10)
memory usage: 41.8+ KB
```

- So, now we are dropping the columns that we won't use for coding to make the view of dataset simpler.



The screenshot shows a Jupyter Notebook interface with the file `Lab-3.ipynb` open. The code cell [16] contains the command `df_car.columns`, which outputs the column names and their data types. The code cell [25] contains the command `df_car = df_car[['aspiration', 'num-of-doors', 'drive-wheels', 'num-of-cylinders']]`. The code cell [26] contains the command `df_car.head()`, which displays the first five rows of the DataFrame.

```

8 engine-location 285 non-null object
9 wheel-base 285 non-null float64
10 length 285 non-null float64
11 width 285 non-null float64
12 height 285 non-null float64
13 curb-weight 285 non-null int64
14 engine-type 285 non-null object
15 num-of-cylinders 285 non-null object
16 engine-size 285 non-null int64
17 fuel-system 285 non-null object
18 bore 281 non-null float64
19 stroke 281 non-null float64
20 compression-ratio 285 non-null float64
21 horsepower 283 non-null float64
22 peak-rpm 283 non-null float64
23 city-mpg 285 non-null int64
24 highway-mpg 285 non-null int64
25 price 281 non-null float64
dtypes: float64(11), int64(5), object(10)
memory usage: 41.8+ KB

[16]: df_car.columns
[16]: Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-of-doors', 'body-style', 'drive-wheels', 'engine-location', 'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-type', 'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke', 'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg', 'highway-mpg', 'price'], dtype='object')

[25]: df_car = df_car[['aspiration', 'num-of-doors', 'drive-wheels', 'num-of-cylinders']]

[ ]:

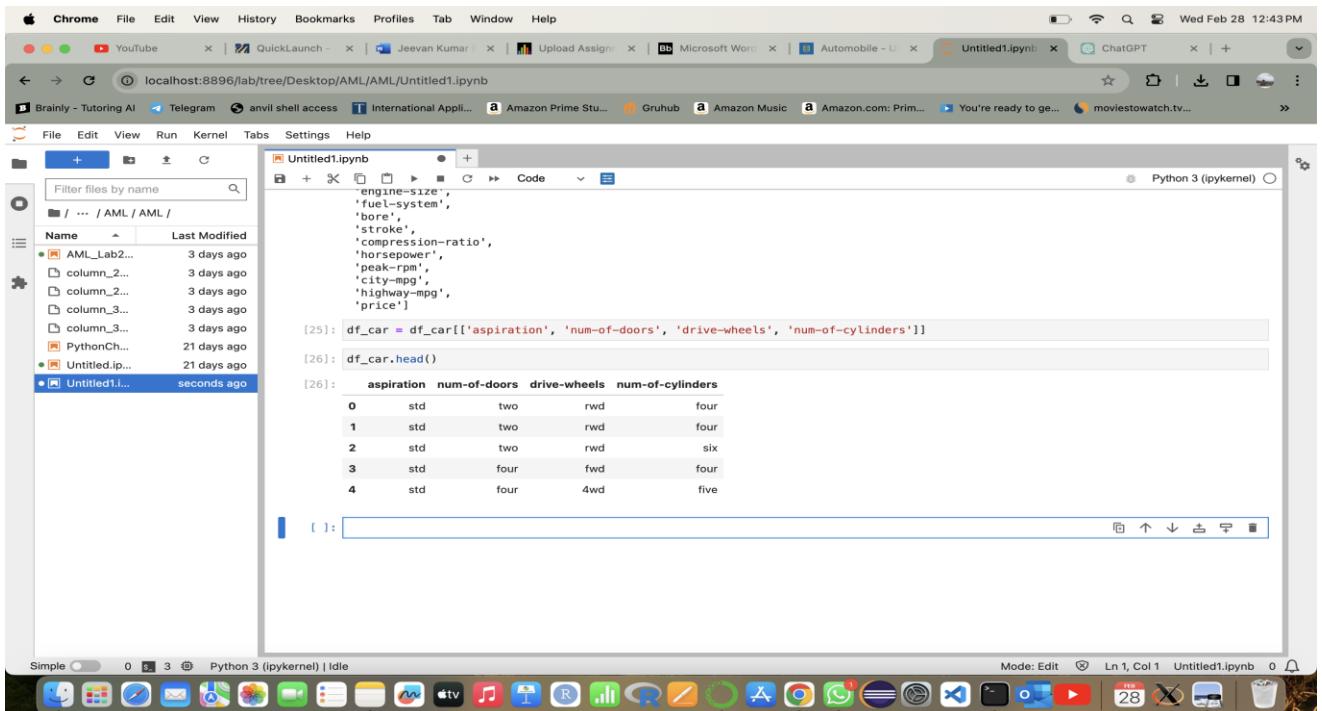
[ ]:

[ ]:

[26]: df_car.head()
[26]:
   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

```

- Now we have four columns that all contain text values, we are using head () command to view those values.



The screenshot shows a Jupyter Notebook interface with the file `Untitled1.ipynb` open. The code cell [25] contains the command `df_car = df_car[['aspiration', 'num-of-doors', 'drive-wheels', 'num-of-cylinders']]`. The code cell [26] contains the command `df_car.head()`, which displays the first five rows of the DataFrame.

```

[25]: df_car = df_car[['aspiration', 'num-of-doors', 'drive-wheels', 'num-of-cylinders']]

[26]: df_car.head()
[26]:
   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

```

## Step 2: Encoding ordinal features:

- Most of the ML algorithms require the values which are numerical. But here, num-of-cylinders and num-of-doors features have an ordinal value so we can convert these features into numerical counterparts.
- Aspiration and drive-wheels don't have an ordinal value, so we need to convert them differently.
- Now, in this step we use mapper function to convert ordinal functions into ordered numerical values.

```
[27]: df_car.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
--- 
 0   aspiration    205 non-null   object  
 1   num-of-doors  203 non-null   object  
 2   drive-wheels  205 non-null   object  
 3   num-of-cylinders  205 non-null  object  
dtypes: object(4)
memory usage: 6.5+ KB
```

- First, we start by what values the ordinal function contain, starting with the num-of-doors feature, we can use **value\_counts** to discover the values.
- The ordinal values of value\_counts feature is shown below.

```

dtypes: object(4)
memory usage: 6.5+ KB

[29]: df_car['num-of-doors'].value_counts()

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

[30]: door_mapper = {"two": 2,
        "four": 4}

```

- It has two values, four and two, for this, we need to create a dictionary and map the variable names to their numerical values as above.
- We can use the replace method from pandas to generate new numerical column based on num-of-doors column.

```

[29]: df_car['num-of-doors'].value_counts()

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

[34]: door_mapper = {
        "two": 2,
        "four": 4

        df_car.loc[:, 'doors'] = df_car["num-of-doors"].replace(door_mapper)

[35]: df_car.head()

```

|   | aspiration | num-of-doors | drive-wheels | num-of-cylinders | doors |
|---|------------|--------------|--------------|------------------|-------|
| 0 | std        | two          | rwd          | four             | 2.0   |
| 1 | std        | two          | rwd          | four             | 2.0   |
| 2 | std        | two          | rwd          | six              | 2.0   |
| 3 | std        | four         | fwd          | four             | 4.0   |
| 4 | std        | four         | 4wd          | five             | 4.0   |

- As shown in the above image, we are using the head function for generating detailed information about it.

- Now we are replacing the above num-of-doors with num-of-cylinders as below.

```

[37]: df_car['num-of-cylinders'].value_counts()
[37]: num-of-cylinders
        four    159
        six     24
        five    11
        eight   5
        two     4
        three   1
       twelve   1
Name: count, dtype: int64

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

[43]: df_car.loc[:, 'cylinders'] = df_car['num-of-cylinders'].replace(cylinder_mapper)

[44]: df_car.head()

```

|   | aspiration | num-of-doors | drive-wheels | num-of-cylinders | doors | cylinders |
|---|------------|--------------|--------------|------------------|-------|-----------|
| 0 | std        | two          | rwd          | four             | 2.0   | 4         |
| 1 | std        | two          | rwd          | four             | 2.0   | 4         |
| 2 | std        | two          | rwd          | six              | 2.0   | 6         |
| 3 | std        | four         | fwd          | four             | 4.0   | 4         |
| 4 | std        | four         | 4wd          | five             | 4.0   | 5         |

- Similarly for the Cylinders column we can create a mapper dictionary and replace the values in the dataset with the respective values as above.

### Step 3: Encoding non-ordinal categorical data:

- In this step, we are using a library from pandas called “get\_dummies”, we will encode non-ordinal data by using it.
- Here for the non-ordinal data, we can't create a mapper dictionary and replace it.
- The ‘get\_dummies’ method adds the separate columns for separate values and then converts the data into Boolean Data types.

```

[46]: df_car['drive-wheels'].value_counts()
[46]: drive-wheels
      fwd    120
      rwd    76
      4wd     9
      Name: count, dtype: int64

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

[49]: df_car.head()
[49]:
   aspiration  num-of-doors  num-of-cylinders  doors  cylinders  drive-wheels_4wd  drive-wheels_fwd  drive-wheels_rwd
0          std           two              four   2.0        4       False      False       True
1          std           two              four   2.0        4       False      False       True
2          std           two              six   2.0        6       False      False       True
3          std           four              four   4.0        4       False      True      False
4          std           four              five   4.0        5       True      False      False

```

- As we examine the dataset, we can see there are three new columns in the dataset.
- Here, the binary variables enable us to express the information in the numerical way without any order to it.
- Now we are examining the final column which is “aspiration”, it has only two values std and turbo.
- We will use here, get\_dummies method but will specify drop\_first as true.

The screenshot shows a Jupyter Notebook interface running in a Chrome browser on a Mac OS X desktop. The browser window title is "localhost:8896/lab/tree/Desktop/AML/AML/Untitled1.ipynb". The notebook sidebar lists files like "AML\_Lab2.ipynb", "column\_2.ipynb", etc. The main area displays a Python code cell and its output:

```
[50]: df_car['aspiration'].value_counts()
[50]: aspiration
      std    168
      turbo   37
      Name: count, dtype: int64
[51]: df_car = pd.get_dummies(df_car, columns=['aspiration'], drop_first=True)
[52]: df_car.head()
```

The output shows the first five rows of the DataFrame "df\_car" with the "aspiration" column converted to dummy variables:

|   | num-of-doors | num-of-cylinders | doors | cylinders | drive-wheels_4wd | drive-wheels_fwd | drive-wheels_rwd | aspiration_turbo |
|---|--------------|------------------|-------|-----------|------------------|------------------|------------------|------------------|
| 0 | two          | four             | 2.0   | 4         | False            | False            | True             | False            |
| 1 | two          | four             | 2.0   | 4         | False            | False            | True             | False            |
| 2 | two          | six              | 2.0   | 6         | False            | False            | True             | False            |
| 3 | four         | four             | 4.0   | 4         | False            | True             | False            | False            |
| 4 | four         | five             | 4.0   | 5         | True             | False            | False            | False            |

## Challenge task:

Go back to the beginning of this lab and add other columns to the dataset. Add at least two columns. How would you encode the values of for that column? Update the code to include some of the other features. Do it for at least one more column.

- So, here I have used the city-mpg and peak-rpm from the dataset.
- To do this, we must load the dataset again as follows.

The screenshot shows a Jupyter Notebook interface running on Python 3 (ipykernel). The code cell [57] contains the following command:

```
[57]: url = "Users/jeevankumarbanoth/Downloads/automobile/imports-85.data"
df_car = pd.read_csv(url, names=col_names, na_values=?, header=None)
```

The code cell [58] contains:

```
[58]: df_car = df_car[['city-mpg', 'peak-rpm']]
```

The code cell [59] contains:

```
[59]: df_car.info()
```

The output of cell [59] is:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 205 entries, 0 to 204
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
 --- 
 0   city-mpg    205 non-null    int64  
 1   peak-rpm    203 non-null    float64 
dtypes: float64(1), int64(1)
memory usage: 3.3 KB
```

The code cell [60] contains:

```
[60]: df_car['city-mpg'].value_counts()
```

The output of cell [60] is:

```
[60]: city-mpg
31    28
19    27
24    22
27    14
17    13
26    12
23    12
21     8
25     8
30     8
38     7
28     7
16     6
37     6
```

- So, we simply must use the pandas ‘get\_dummies’ method to create new columns and with the value names and convert them into binary values.
  - The same method is used for the engine location column too.

```

[64]: df_car['peak-rpm'].value_counts()
[64]:
      Name
    5500.0    37
    4800.0    36
    5000.0    27
    5200.0    23
    5400.0    13
    6000.0     9
    5800.0     7
    5200.0     7
    4500.0     7
    4150.0     5
    4200.0     5
    4350.0     4
    4750.0     4
    5100.0     3
    5900.0     3
    4250.0     3
    4400.0     3
    6600.0     2
    4650.0     1
    5000.0     1
    5750.0     1
    4900.0     1
    5300.0     1
Name: count, dtype: int64

[65]: df_car = pd.get_dummies(df_car,columns=['peak-rpm'], drop_first=True)
[66]: df_car.replace({False: 0, True: 1}, inplace=True)
[67]: df_car.head()

```

## Conclusion:

- We started by loading the dataset to get an idea of its structure, size, and variables.
- We converted the raw data from a CSV file into a Data Frame to make it easier to work with.
- We identified which columns held object data types, which usually represent categories.
- We took care to handle categorical data properly, classifying them as ordinal or non-ordinal.
- For ordinal features, we created a mapper dictionary to convert categorical values into numerical equivalents while keeping the order of the categories.
- We replaced the original ordinal values in the dataset with their numerical equivalents.

To summarize, our data preprocessing steps lay the groundwork for accurate and efficient training of machine learning models. We ensure our dataset is in a usable format for advanced data analysis and modeling techniques. This involves handling data import, examining the data, segregating it, and encoding it. Data preprocessing is an essential and foundational stage in our data analysis and modeling pipeline.