Data Wrangling

Estimated time needed: **30** minutes

Objectives

After completing this lab you will be able to:

- Handle missing values
- Correct data format
- Standardize and normalize data

Table of Contents

- Identify and handle missing values
    - Identify missing values
    - Deal with missing values
    - Correct data format
- Data standardization
- Data normalization (centering/scaling)
- Binning
- Indicator variable

What is the purpose of data wrangling?

Data wrangling is the process of converting data from the initial format to a format that may be better for analysis.

What is the fuel consumption (L/100k) rate for the diesel car?

Import data

You can find the "Automobile Dataset" from the following link: https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data. We will be using this dataset throughout this course.

Import pandas

```
#install specific version of libraries used in lab
#! mamba install pandas==1.3.3
#! mamba install numpy=1.21.2
```

```
import pandas as pd
import matplotlib.pylab as plt
```

Reading the dataset from the URL and adding the related headers

First, we assign the URL of the dataset to "filename".

This dataset was hosted on IBM Cloud object. Click HERE for free storage.

```
filename = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMDeveloperSk
```

Then, we create a Python list **headers** containing name of headers.

```
headers = ["symboling","normalized-losses","make","fuel-type","aspiration", "num-of-doors","b
         "drive-wheels","engine-location","wheel-base", "length","width","height","curb-weigh
         "num-of-cylinders", "engine-size","fuel-system","bore","stroke","compression-ratio",
         "peak-rpm","city-mpg","highway-mpg","price"]
```

Use the Pandas method **read_csv()** to load the data from the web address. Set the parameter "names" equal to the Python list "headers".

```
df = pd.read_csv(filename, names = headers)
```

Use the method **head()** to display the first five rows of the dataframe.

```
# To see what the data set looks like, we'll use the head() method.
```

```
df.head()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front
3	2	164	audi	gas	std	four	sedan	fwd	front
4	2	164	audi	gas	std	four	sedan	4wd	front

5 rows × 26 columns

As we can see, several question marks appeared in the dataframe; those are missing values which may hinder our further analysis.

So, how do we identify all those missing values and deal with them?

How to work with missing data?

Steps for working with missing data:

1. Identify missing data
2. Deal with missing data
3. Correct data format

Identify and handle missing values

Identify missing values

Convert "?" to NaN

In the car dataset, missing data comes with the question mark "?". We replace "?" with NaN (Not a Number), Python's default missing value marker for reasons of computational speed and convenience. Here we use the function:

```
.replace(A, B, inplace = True)
```

to replace A by B.

```
import numpy as np

# replace "?" to NaN
df.replace("?", np.nan, inplace = True)
df.head(5)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front
3	2	164	audi	gas	std	four	sedan	fwd	front
4	2	164	audi	gas	std	four	sedan	4wd	front

5 rows × 26 columns

Evaluating for Missing Data

The missing values are converted by default. We use the following functions to identify these missing values. There are two methods to detect missing data:

1. **.isnull()**
2. **.notnull()**

The output is a boolean value indicating whether the value that is passed into the argument is in fact missing data.

```
missing_data = df.isnull()
missing_data.head(5)
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	whe b
0	False	True	False	False	False	False	False	False	False	Fa
1	False	True	False	False	False	False	False	False	False	Fa
2	False	True	False	False	False	False	False	False	False	Fa
3	False	False	False	False	False	False	False	False	False	Fa
4	False	False	False	False	False	False	False	False	False	Fa

"True" means the value is a missing value while "False" means the value is not a missing value.

Count missing values in each column

Using a for loop in Python, we can quickly figure out the number of missing values in each column. As mentioned above, "True" represents a missing value and "False" means the value is present in the dataset. In the body of the for loop the method ".value_counts()" counts the number of "True" values.

```
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
```

```
curb-weight
False    205
Name: curb-weight, dtype: int64

engine-type
False    205
Name: engine-type, dtype: int64

num-of-cylinders
False    205
Name: num-of-cylinders, dtype: int64

engine-size
False    205
Name: engine-size, dtype: int64

fuel-system
False    205
Name: fuel-system, dtype: int64

bore
False    201
True       4
Name: bore, dtype: int64
```

```
Name: bore, dtype: int64

stroke
False    201
True       4
Name: stroke, dtype: int64

compression-ratio
False    205
Name: compression-ratio, dtype: int64

horsepower
False    203
True       2
Name: horsepower, dtype: int64

peak-rpm
False    203
True       2
Name: peak-rpm, dtype: int64

city-mpg
False    205
Name: city-mpg, dtype: int64

highway-mpg
False    205
Name: highway-mpg, dtype: int64

price
False    201
True       4
Name: price, dtype: int64
```

Based on the summary above, each column has 205 rows of data and seven of the columns containing missing data:

1. "normalized-losses": 41 missing data
2. "num-of-doors": 2 missing data
3. "bore": 4 missing data
4. "stroke" : 4 missing data
5. "horsepower": 2 missing data
6. "peak-rpm": 2 missing data
7. "price": 4 missing data

Deal with missing data

How to deal with missing data?

1. Drop data
    a. Drop the whole row
    b. Drop the whole column
2. Replace data
    a. Replace it by mean
    b. Replace it by frequency
    c. Replace it based on other functions

Whole columns should be dropped only if most entries in the column are empty. In our dataset, none of the columns are empty enough to drop entirely. We have some freedom in choosing which method to replace data; however, some methods may seem more reasonable than others. We will apply each method to many different columns:

Replace by mean:

- "normalized-losses": 41 missing data, replace them with mean
- "stroke": 4 missing data, replace them with mean
- "bore": 4 missing data, replace them with mean
- "horsepower": 2 missing data, replace them with mean
- "peak-rpm": 2 missing data, replace them with mean

Replace by frequency:

- "num-of-doors": 2 missing data, replace them with "four".
    - Reason: 84% sedans is four doors. Since four doors is most frequent, it is most likely to occur

Drop the whole row:

- "price": 4 missing data, simply delete the whole row
    - Reason: price is what we want to predict. Any data entry without price data cannot be used for prediction; therefore any row now without price data is not useful to us

Calculate the mean value for the "normalized-losses" column

```
avg_norm_loss = df["normalized-losses"].astype("float").mean(axis=0)
print("Average of normalized-losses:", avg_norm_loss)
```

```
    Average of normalized-losses: 122.0
```

Replace "NaN" with mean value in "normalized-losses" column

```
df["normalized-losses"].replace(np.nan, avg_norm_loss, inplace=True)
```

Calculate the mean value for the "bore" column

```
avg_bore=df['bore'].astype('float').mean(axis=0)
print("Average of bore:", avg_bore)
```

```
     Average of bore: 3.3297512437810943
```

Replace "NaN" with the mean value in the "bore" column

```
df["bore"].replace(np.nan, avg_bore, inplace=True)
```

Question #1:

Based on the example above, replace NaN in "stroke" column with the mean value.

```
# Write your code below and press Shift+Enter to execute
avg_stroke = df["stroke"].astype("float").mean(axis = 0)

df["stroke"].replace(np.nan, avg_stroke, inplace = True)
```

▶ Click here for the solution

Calculate the mean value for the "horsepower" column

```
avg_horsepower = df['horsepower'].astype('float').mean(axis=0)
print("Average horsepower:", avg_horsepower)
```

```
     Average horsepower: 103.40553390682057
```

Replace "NaN" with the mean value in the "horsepower" column

```
df['horsepower'].replace(np.nan, avg_horsepower, inplace=True)
```

Calculate the mean value for "peak-rpm" column

```
avg_peakrpm=df['peak-rpm'].astype('float').mean(axis=0)
print("Average peak rpm:", avg_peakrpm)
```

```
Average peak rpm: 5117.665367742568
```

Replace "NaN" with the mean value in the "peak-rpm" column

```
df['peak-rpm'].replace(np.nan, avg_peakrpm, inplace=True)
```

To see which values are present in a particular column, we can use the ".value_counts()" method:

```
df['num-of-doors'].value_counts()
```

```
four    115
two      86
Name: num-of-doors, dtype: int64
```

We can see that four doors are the most common type. We can also use the ".idxmax()" method to calculate the most common type automatically:

```
df['num-of-doors'].value_counts().idxmax()
```

```
'four'
```

The replacement procedure is very similar to what we have seen previously:

```
#replace the missing 'num-of-doors' values by the most frequent
df["num-of-doors"].replace(np.nan, "four", inplace=True)
```

Finally, let's drop all rows that do not have price data:

```
# simply drop whole row with NaN in "price" column
df.dropna(subset=["price"], axis=0, inplace=True)

# reset index, because we droped two rows
df.reset_index(drop=True, inplace=True)
```

```
df.head()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front
3	2	164	audi	gas	std	four	sedan	fwd	front
4	2	164	audi	gas	std	four	sedan	4wd	front

5 rows × 26 columns

Good! Now, we have a dataset with no missing values.

Correct data format

We are almost there!

The last step in data cleaning is checking and making sure that all data is in the correct format (int, float, text or other).

In Pandas, we use:

.dtype() to check the data type

.astype() to change the data type

Let's list the data types for each column

```
df.dtypes
```

```
symboling              int64
normalized-losses      int64
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
```

```
height               float64
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                 float64
stroke               float64
compression-ratio    float64
horsepower            object
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
city-L/100km         float64
dtype: object
```

As we can see above, some columns are not of the correct data type. Numerical variables should have type 'float' or 'int', and variables with strings such as categories should have type 'object'. For example, 'bore' and 'stroke' variables are numerical values that describe the engines, so we should expect them to be of the type 'float' or 'int'; however, they are shown as type 'object'. We have to convert data types into a proper format for each column using the "astype()" method.

Convert data types to proper format

```
df[["bore", "stroke"]] = df[["bore", "stroke"]].astype("float")
df[["normalized-losses"]] = df[["normalized-losses"]].astype("int")
df[["price"]] = df[["price"]].astype("float")
df[["peak-rpm"]] = df[["peak-rpm"]].astype("float")
```

Let us list the columns after the conversion

```
df.dtypes
```

```
symboling              int64
normalized-losses      int64
make                  object
fuel-type             object
aspiration            object
num-of-doors          object
body-style            object
drive-wheels          object
engine-location       object
wheel-base           float64
length               float64
width                float64
height               float64
```

```
curb-weight            int64
engine-type           object
num-of-cylinders      object
engine-size            int64
fuel-system           object
bore                 float64
stroke               float64
compression-ratio    float64
horsepower            object
peak-rpm             float64
city-mpg               int64
highway-mpg            int64
price                float64
city-L/100km         float64
dtype: object
```

Wonderful!

Now we have finally obtained the cleaned dataset with no missing values with all data in its proper format.

Data Standardization

Data is usually collected from different agencies in different formats. (Data standardization is also a term for a particular type of data normalization where we subtract the mean and divide by the standard deviation.)

What is standardization?

Standardization is the process of transforming data into a common format, allowing the researcher to make the meaningful comparison.

Example

Transform mpg to L/100km:

In our dataset, the fuel consumption columns "city-mpg" and "highway-mpg" are represented by mpg (miles per gallon) unit. Assume we are developing an application in a country that accepts the fuel consumption with L/100km standard.

We will need to apply **data transformation** to transform mpg into L/100km.

The formula for unit conversion is:

L/100km = 235 / mpg

We can do many mathematical operations directly in Pandas.

```
df.head()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front
3	2	164	audi	gas	std	four	sedan	fwd	front
4	2	164	audi	gas	std	four	sedan	4wd	front

5 rows × 27 columns

```
# Convert mpg to L/100km by mathematical operation (235 divided by mpg)
df['city-L/100km'] = 235/df["city-mpg"]

# check your transformed data
df.head()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front
3	2	164	audi	gas	std	four	sedan	fwd	front
4	2	164	audi	gas	std	four	sedan	4wd	front

5 rows × 27 columns

Question #2:

According to the example above, transform mpg to L/100km in the column of "highway-mpg" and change the name of column to "highway-L/100km"

```
# Write your code below and press Shift+Enter to execute
df["highway-mpg"] = 235 / df["highway-mpg"]
df.rename(columns={"highway-mpg": "highway-L/100km"}, inplace=True)
df.head()
```

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front
3	2	164	audi	gas	std	four	sedan	fwd	front
4	2	164	audi	gas	std	four	sedan	4wd	front

5 rows × 27 columns

▶ Click here for the solution

Data Normalization

Why normalization?

Normalization is the process of transforming values of several variables into a similar range. Typical normalizations include scaling the variable so the variable average is 0, scaling the variable so the variance is 1, or scaling the variable so the variable values range from 0 to 1.

Example

To demonstrate normalization, let's say we want to scale the columns "length", "width" and "height".

Target: would like to normalize those variables so their value ranges from 0 to 1

Approach: replace original value by (original value)/(maximum value)

```
# replace (original value) by (original value)/(maximum value)
df['length'] = df['length']/df['length'].max()
df['width'] = df['width']/df['width'].max()
```

Question #3:

According to the example above, normalize the column "height".

```
# Write your code below and press Shift+Enter to execute
df["height"] = df["height"]/df["height"].max()
df[["length","width","height"]].head()
```

```
     length     width    height
0  0.811148  0.890278  0.816054
1  0.811148  0.890278  0.816054
2  0.822681  0.909722  0.876254
3  0.848630  0.919444  0.908027
4  0.848630  0.922222  0.908027
```

▶ Click here for the solution

Here we can see we've normalized "length", "width" and "height" in the range of [0,1].

Binning

Why binning?

Binning is a process of transforming continuous numerical variables into discrete categorical 'bins' for grouped analysis.

Example:

In our dataset, "horsepower" is a real valued variable ranging from 48 to 288 and it has 59 unique values. What if we only care about the price difference between cars with high horsepower, medium horsepower, and little horsepower (3 types)? Can we rearrange them into three ‘bins' to simplify analysis?

We will use the pandas method 'cut' to segment the 'horsepower' column into 3 bins.

Example of Binning Data In Pandas

Convert data to correct format:

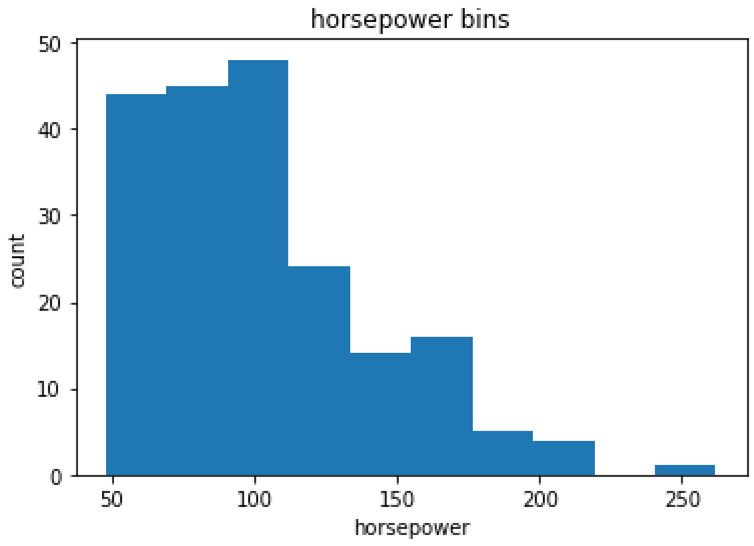
```
df["horsepower"]=df["horsepower"].astype(int, copy=True)
```

Let's plot the histogram of horsepower to see what the distribution of horsepower looks like.

```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
plt.pyplot.hist(df["horsepower"])

# set x/y labels and plot title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

```
    Text(0.5, 1.0, 'horsepower bins')
```



We would like 3 bins of equal size bandwidth so we use numpy's `linspace(start_value, end_value, numbers_generated` function.

Since we want to include the minimum value of horsepower, we want to set start_value = min(df["horsepower"]).

Since we want to include the maximum value of horsepower, we want to set end_value = max(df["horsepower"]).

Since we are building 3 bins of equal length, there should be 4 dividers, so numbers_generated = 4.

We build a bin array with a minimum value to a maximum value by using the bandwidth calculated above. The values will determine when one bin ends and another begins.

```
bins = np.linspace(min(df["horsepower"]), max(df["horsepower"]), 4)
bins
```

```
     array([ 48.        , 119.33333333, 190.66666667, 262.        ])
```

We set group names:

```
group_names = ['Low', 'Medium', 'High']
```

We apply the function "cut" to determine what each value of `df['horsepower']` belongs to.

```
df['horsepower-binned'] = pd.cut(df['horsepower'], bins, labels=group_names, include_lowest=T
df[['horsepower','horsepower-binned']].head(20)
```

	horsepower	horsepower-binned
0	111	Low
1	111	Low
2	154	Medium

Let's see the number of vehicles in each bin:

4	115	Low

```
df["horsepower-binned"].value_counts()
```

```
Low       153
Medium     43
High        5
Name: horsepower-binned, dtype: int64
```

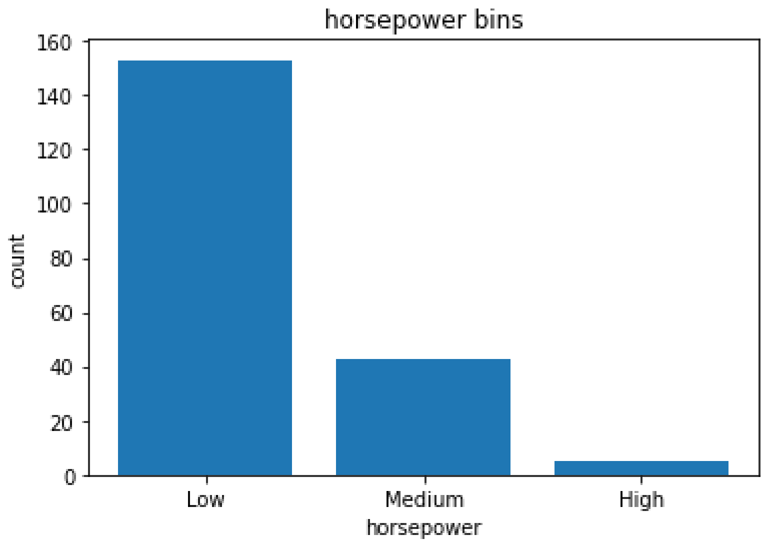
Let's plot the distribution of each bin:

10	101	Low

```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot
pyplot.bar(group_names, df["horsepower-binned"].value_counts())

# set x/y labels and plot title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

```
Text(0.5, 1.0, 'horsepower bins')
```



Look at the dataframe above carefully. You will find that the last column provides the bins for "horsepower" based on 3 categories ("Low", "Medium" and "High").

We successfully narrowed down the intervals from 59 to 3!

Bins Visualization

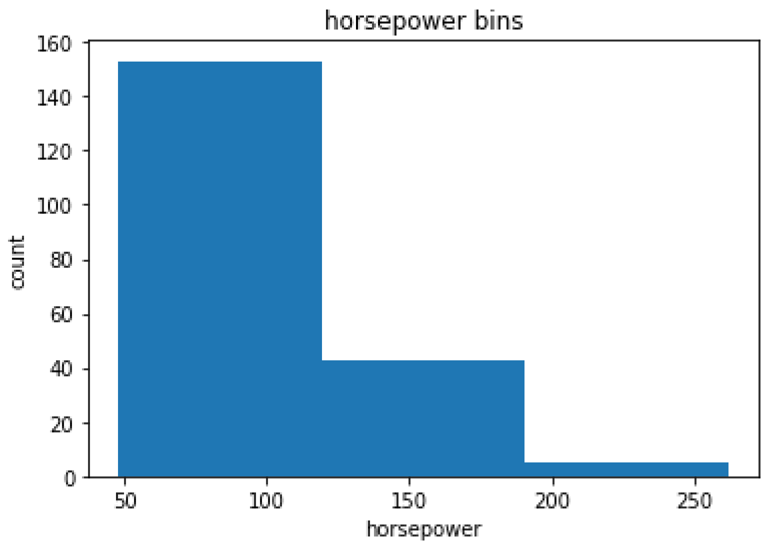
Normally, a histogram is used to visualize the distribution of bins we created above.

```
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot


# draw historgram of attribute "horsepower" with bins = 3
plt.pyplot.hist(df["horsepower"], bins = 3)

# set x/y labels and plot title
plt.pyplot.xlabel("horsepower")
plt.pyplot.ylabel("count")
plt.pyplot.title("horsepower bins")
```

```
    Text(0.5, 1.0, 'horsepower bins')
```



The plot above shows the binning result for the attribute "horsepower".

Indicator Variable (or Dummy Variable)

What is an indicator variable?

An indicator variable (or dummy variable) is a numerical variable used to label categories. They are called 'dummies' because the numbers themselves don't have inherent meaning.

Why we use indicator variables?

We use indicator variables so we can use categorical variables for regression analysis in the later modules.

Example

We see the column "fuel-type" has two unique values: "gas" or "diesel". Regression doesn't understand words, only numbers. To use this attribute in regression analysis, we convert "fuel-type" to indicator variables.

We will use pandas' method 'get_dummies' to assign numerical values to different categories of fuel type.

```
df.columns
```

```
    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-L/100km', 'price', 'city-L/100km', 'horsepower-binned'],
          dtype='object')
```

Get the indicator variables and assign it to data frame "dummy_variable_1":

```
dummy_variable_1 = pd.get_dummies(df["fuel-type"])
dummy_variable_1.head()
```

	diesel	gas
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1

Change the column names for clarity:

```
dummy_variable_1.rename(columns={'gas':'fuel-type-gas', 'diesel':'fuel-type-diesel'}, inplace
dummy_variable_1.head()
```

	fuel-type-diesel	fuel-type-gas
0	0	1

In the dataframe, column 'fuel-type' has values for 'gas' and 'diesel' as 0s and 1s now.

```
# merge data frame "df" and "dummy_variable_1"
df = pd.concat([df, dummy_variable_1], axis=1)

# drop original column "fuel-type" from "df"
df.drop("fuel-type", axis = 1, inplace=True)
```

```
df.head()
```

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel bas
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.
3	2	164	audi	std	four	sedan	fwd	front	99.
4	2	164	audi	std	four	sedan	4wd	front	99.

5 rows × 29 columns

The last two columns are now the indicator variable representation of the fuel-type variable. They're all 0s and 1s now.

Question #4:

Similar to before, create an indicator variable for the column "aspiration"

```
# Write your code below and press Shift+Enter to execute
dummy_variable_2 = pd.get_dummies(df['aspiration'])
dummy_variable_2.rename(columns={'std':'aspiration-std', 'turbo': 'aspiration-turbo'}, inplac
dummy_variable_2.head()
```

	aspiration-std	aspiration-turbo
0	1	0
1	1	0
2	1	0
3	1	0
4	1	0

▶ Click here for the solution

Question #5:

Merge the new dataframe to the original dataframe, then drop the column 'aspiration'.

```
# Write your code below and press Shift+Enter to execute
df = pd.concat([df, dummy_variable_2], axis=1)
df.drop('aspiration', axis = 1, inplace=True)
```

▶ Click here for the solution

Save the new csv:

```
df.to_csv('clean_df.csv')
```

Thank you for completing this lab!

Author

[Joseph Santarcangelo]()

Other Contributors

[Mahdi Noorian PhD]()

Bahare Talayian

Eric Xiao

Steven Dong

Parizad

Hima Vasudevan

Fiorella Wenver

Yi Yao.

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2020-10-30	2.2	Lakshmi	Changed URL of csv
2020-09-09	2.1	Lakshmi	Updated Indicator Variables section
2020-08-27	2.0	Lavanya	Moved lab to course repo in GitLab

✓ 0 s completado a las 20:15