

Skills Network Labs

File Edit View Run Kernel Git Tabs Settings Help

Launcher data-wrangling.ipynb git Run as Pipeline Python

 IBM Developer SKILLS NETWORK

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 content

- 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 Data Set" from the following link: <https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data>. We will be using this data set throughout this course.

Import pandas

```
[ ]: import pandas as pd
      import matplotlib.pyplot as plt
```

Reading the data set from the URL and adding the related headers.

URL of the dataset

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/IBMDeveloperSkillsNetwork-DA0101EN-SkillsNetwork/labs/Data%20files/auto.csv."
```

Python list headers containing name of headers

```
[ ]: headers = ["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"]
```

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()
```

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), which is 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)
```

Identify_missing_values

Evaluating for Missing Data

The missing values are converted to 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)
```

"True" stands for missing value, while "False" stands for not 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, "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("").....
```

Based on the summary above, each column has 205 rows of data, seven 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 average of the column

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

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

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

Calculate the mean value for 'bore' column

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

Replace NaN by mean value

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

Question #1:

According to the example above, replace NaN in "stroke" column by mean.

```
[ ]: # Write your code below and press Shift+Enter to execute..
```

▼ Click here for the solution

```
#Calculate the mean value for "stroke" column
avg_stroke = df["stroke"].astype("float").mean(axis = 0)
print("Average of stroke:", avg_stroke)

# replace NaN by mean value in "stroke" column
df["stroke"].replace(np.nan, avg_stroke, inplace = True)
```

Calculate the mean value for the 'horsepower' column:

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

Replace "NaN" by mean value:

```
[ ]: 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)
```

Replace NaN by mean value:

```
[ ]: 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()
```

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

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

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 dropped two rows
df.reset_index(drop=True, inplace=True)
```

```
[ ]: df.head()
```

Good! Now, we obtain the 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

Lets list the data types for each column

```
[ ]: df.dtypes
```

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

Did you know? IBM Watson Studio lets you build and deploy an AI solution, using the best of open source and IBM software and giving your team a single environment to work in. [Learn more here.](#)

```
[ ]: 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
```

Wonderful!

Now, we finally obtain the cleaned dataset with no missing values and all data in its proper format.

Data Standardization

Data is usually collected from different agencies with 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 which allows 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 accept 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()
[ ]: # 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()
```

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..
```

▼ Click here for the solution
transform mpg to L/100km by mathematical operation (235 divided by mpg)
df["highway-mpg"] = 235/df["highway-mpg"]

rename column name from "highway-mpg" to "highway-L/100km"
df.rename(columns={"highway-mpg":'highway-L/100km'}, inplace=True)

check your transformed data
df.head()

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 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..
```

▼ Click here for the solution
df['height'] = df['height']/df['height'].max()

show the scaled columns
df[['length','width','height']].head()

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, it has 57 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)
```

Lets 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")
```

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, with bandwidth calculated above. The bins will be values used to determine when one bin ends and another begins.

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

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=True)
df[['horsepower','horsepower-binned']].head(20)
```

Lets see the number of vehicles in each bin.

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

Lets plot the distribution of each bin.

```
[ ]: %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")
```

Check the dataframe above carefully, you will find the last column provides the bins for "horsepower" with 3 categories ("Low", "Medium" and "High").

We successfully narrow the intervals from 57 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 histogram 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")
```

The plot above shows the binning result for 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?

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" into indicator variables.

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

```
[ ]: df.columns
```

get indicator variables and assign it to data frame "dummy_variable_1"

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

change column names for clarity

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

In the dataframe, column fuel-type has a value 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()
```

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

Question #4:

As above, create indicator variable to the column of "aspiration"

[]: # Write your code below and press Shift+Enter to execute..

▼ Click here for the solution
get indicator variables of aspiration and assign it to data frame "dummy_variable_2"
dummy_variable_2 = pd.get_dummies(df['aspiration'])

change column names for clarity
dummy_variable_2.rename(columns={'std':'aspiration-std', 'turbo': 'aspiration-turbo'}, inplace=True)

show first 5 instances of data frame "dummy_variable_1"
dummy_variable_2.head()

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..

▼ Click here for the solution
merge the new dataframe to the original datafram
df = pd.concat([df, dummy_variable_2], axis=1)

drop original column "aspiration" from "df"
df.drop('aspiration', axis = 1, inplace=True)

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

Simple 0 20 Fully initialized Python | Idle Mem: 1.16 / 6.00 GB

Mode: Command Ln 1, Col 1 English (American) data-wrangling.ipynb