Data Pre-processing

Objectives

After completing this lab you will be able to:

- Handle missing values
- Correct data formatting
- Standardize and normalize data
- Binning
- Indicator Variables (or dummy variable)/category into numeric values

Table of Contents

What is the purpose of data wrangling or Data Pre-Processing ?

You use data wrangling to convert data from an 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

We can find the "Automobile Dataset" from the following link:
https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data (https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data).

Import pandas

In [114]:
```
import pandas as pd
import matplotlib.pylab as plt
```

Reading the dataset and adding the related headers

Create a Python list **headers** containing name of headers.

In [115]:
```
headers = ["symboling","normalized-losses","make","fuel-type","aspiration",
           "drive-wheels","engine-location","wheel-base", "length","width","he
           "num-of-cylinders", "engine-size","fuel-system","bore","stroke","c
           "peak-rpm","city-mpg","highway-mpg","price"]
```

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

In [116]:
```
# In This case I will download the file and save it in local file.
df = pd.read_csv("auto.csv", names = headers)
```

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

In [117]:
```
# To see what the data set looks like, we'll use the head() method.
df.head()
```

Out[117]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	whe ba
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	8
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	8
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	9
3	2	164	audi	gas	std	four	sedan	fwd	front	9
4	2	164	audi	gas	std	four	sedan	4wd	front	9

5 rows × 26 columns

As you can see, several question marks appeared in the data frame; those missing values may hinder further analysis.

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 data set, 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. Use the function:

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

to replace A by B.

In [118]:
```
import numpy as np

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

Out[118]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	whe ba
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	8
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	8
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	9
3	2	164	audi	gas	std	four	sedan	fwd	front	9
4	2	164	audi	gas	std	four	sedan	4wd	front	9

5 rows × 26 columns

Evaluating for Missing Data

The missing values are converted by default. Use the following functions to identify these missing values. 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.

In [119]:
```
missing_data = df.isnull()
missing_data.head(5)
```

Out[119]:

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

5 rows × 26 columns

"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, you 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 data set. In the body of the for loop the method ".value_counts()" counts the number of "True" values.

In [120]:
```
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
```

```
symboling
symboling
False    205
Name: count, dtype: int64

normalized-losses
normalized-losses
False    164
True      41
Name: count, dtype: int64

make
make
False    205
Name: count, dtype: int64

fuel-type
fuel-type
False    205
```

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 should you 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

We should only drop whole columns if most entries in the column are empty. In the data set, 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. Apply each method to 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 are 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: You want to predict price. You cannot use any data entry without price data for prediction; therefore any row now without price data is not useful to you.

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

In [121]:
```
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
```

In [122]:
```
#axis=0: This refers to calculating the mean column-wise, meaning the operat
#axis=1: This refers to calculating the mean row-wise, meaning the operatior
```

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

In [123]:
```
df["normalized-losses"] = df["normalized-losses"].replace(np.nan, avg_norm_
```

Calculate the mean value for the "bore" column

In [124]:
```
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

In [125]:
```
df["bore"] = df["bore"].replace(np.nan, avg_bore)
```

In [126]:
```
# Replace NaN in "stroke" column with the mean value.
avg_stroke = df['stroke'].astype('float').mean(axis=0)
#print("Average of stroke:", avg_stroke)
df["stroke"] = df["stroke"].replace(np.nan, avg_stroke)
```

Calculate the mean value for the "horsepower" column

In [127]:
```
avg_horsepower = df['horsepower'].astype('float').mean(axis=0)
print("Average horsepower:", avg_horsepower)
```

```
Average horsepower: 104.25615763546799
```

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

In [128]:
```
df['horsepower'] = df['horsepower'].replace(np.nan, avg_horsepower)
```

Calculate the mean value for "peak-rpm" column

In [129]:
```
avg_peakrpm=df['peak-rpm'].astype('float').mean(axis=0)
print("Average peak rpm:", avg_peakrpm)
```

```
Average peak rpm: 5125.369458128079
```

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

In [130]:
```
df['peak-rpm'] = df['peak-rpm'].replace(np.nan, avg_peakrpm)
```

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

In [131]:
```
df['num-of-doors'].value_counts()
```

Out[131]:
```
num-of-doors
four    114
two      89
Name: count, dtype: int64
```

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

In [132]:
```
df['num-of-doors'].value_counts().idxmax()
```

Out[132]:
```
'four'
```

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

In [133]:
```
#replace the missing 'num-of-doors' values by the most frequent
df["num-of-doors"] = df["num-of-doors"].replace(np.nan, "four")
```

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

In [134]:
```
# 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)
```

In [135]:
```
df.head()
```

Out[135]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	whe ba
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	8
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	8
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	9
3	2	164	audi	gas	std	four	sedan	fwd	front	9
4	2	164	audi	gas	std	four	sedan	4wd	front	9

5 rows × 26 columns

Good! Now, we have a data set 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

In [136]:
```
df.dtypes
```

Out[136]:
```
symboling              int64
normalized-losses     object
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                  object
stroke                object
compression-ratio    float64
horsepower            object
peak-rpm              object
city-mpg               int64
highway-mpg            int64
price                 object
dtype: object
```

As you 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, the numerical values 'bore' and 'stroke' describe the engines, so you should expect them to be of the type 'float' or 'int'; however, they are shown as type 'object'. You have to convert data types into a proper format for each column using the "astype()" method.

Convert data types to proper format

In [137]:
```
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

In [138]:
```
df.dtypes
```

Out[138]:
```
symboling              int64
normalized-losses      int32
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
dtype: object
```

Wonderful!

Now we finally obtained the cleansed data set with no missing values and with all data in its proper format.

Data Standardization

We usually collect data 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 data set, 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.

We use this formula for unit conversion:

L/100km = 235 / mpg

we can do many mathematical operations directly using Pandas.

In [139]:
```
df.head()
```

Out[139]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	whe ba
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	8
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	8
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	9
3	2	164	audi	gas	std	four	sedan	fwd	front	9
4	2	164	audi	gas	std	four	sedan	4wd	front	9

5 rows × 26 columns

In [140]:
```
# 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()
```

Out[140]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	whe ba
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	8
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	8
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	9
3	2	164	audi	gas	std	four	sedan	fwd	front	9
4	2	164	audi	gas	std	four	sedan	4wd	front	9

5 rows × 27 columns

Transform mpg to L/100km in the column of "highway-mpg" and change the name of

In [141]:
```
# 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()
```

Out[141]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	whe ba
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	8
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	8
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	9
3	2	164	audi	gas	std	four	sedan	fwd	front	9
4	2	164	audi	gas	std	four	sedan	4wd	front	9

5 rows × 27 columns

Data Normalization

Why normalization?

Normalization is the process of transforming values of several variables into a similar range. Typical normalizations include

1. scaling the variable so the variable average is 0
2. scaling the variable so the variance is 1
3. scaling the variable so the variable values range from 0 to 1

Example

To demonstrate normalization, say you want to scale the columns "length", "width" and "height".

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

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

In [142]:
```
# replace (original value) by (original value)/(maximum value)
df['length'] = df['length']/df['length'].max()
df['width'] = df['width']/df['width'].max()
```

Normalize the column "height".

In [143]:
```
# replace (original value) by (original value)/(maximum value)
df['height'] = df['height']/df['height'].max()
# show the scaled columns
df[["length","width","height"]].head()
```

Out[143]:

	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

We've normalized "length", "width" and "height" to fall 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 your data set, "horsepower" is a real valued variable ranging from 48 to 288 and it has 59 unique values. What if you only care about the price difference between cars with high horsepower, medium horsepower, and little horsepower (3 types)? You can rearrange them into three ‘bins' to simplify analysis.

Use the Pandas method 'cut' to segment the 'horsepower' column into 3 bins.

Example of Binning Data In Pandas

Convert data to correct format:

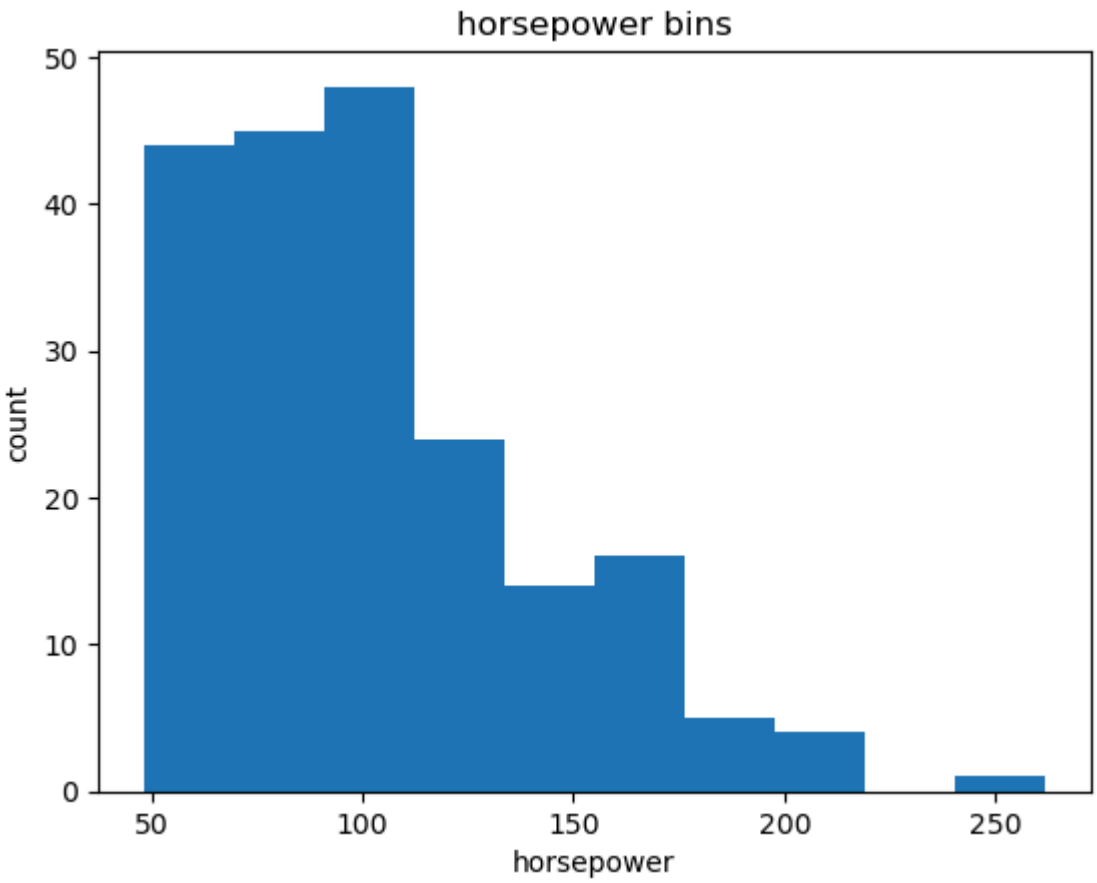
In [144]:
```
df["horsepower"]=df["horsepower"].astype(int, copy=True)
```

Plot the histogram of horsepower to see the distribution of horsepower.

In [145]:
```
%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")
```

Out[145]:
```
Text(0.5, 1.0, 'horsepower bins')
```



Find 3 bins of equal size bandwidth by using Numpy's `linspace(start_value, end_value, numbers_generated` function.

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

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

Since you are building 3 bins of equal length, you need 4 dividers, so numbers_generated = 4.

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.

In [146]:
```
bins = np.linspace(min(df["horsepower"]), max(df["horsepower"]), 4)
bins
```

Out[146]:
```
array([ 48.        , 119.33333333, 190.66666667, 262.        ])
```

Set group names:

In [147]:
```
group_names = ['Low', 'Medium', 'High']
```

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

In [148]:
```
df['horsepower-binned'] = pd.cut(df['horsepower'], bins, labels=group_names,
df[['horsepower','horsepower-binned']].head(20)
```

Out[148]:

	horsepower	horsepower-binned
0	111	Low
1	111	Low
2	154	Medium
3	102	Low
4	115	Low
5	110	Low
6	110	Low
7	110	Low
8	140	Medium
9	101	Low
10	101	Low
11	121	Medium
12	121	Medium
13	121	Medium
14	182	Medium
15	182	Medium
16	182	Medium
17	48	Low
18	70	Low
19	70	Low

See the number of vehicles in each bin:

In [149]:
```
df["horsepower-binned"].value_counts()
```

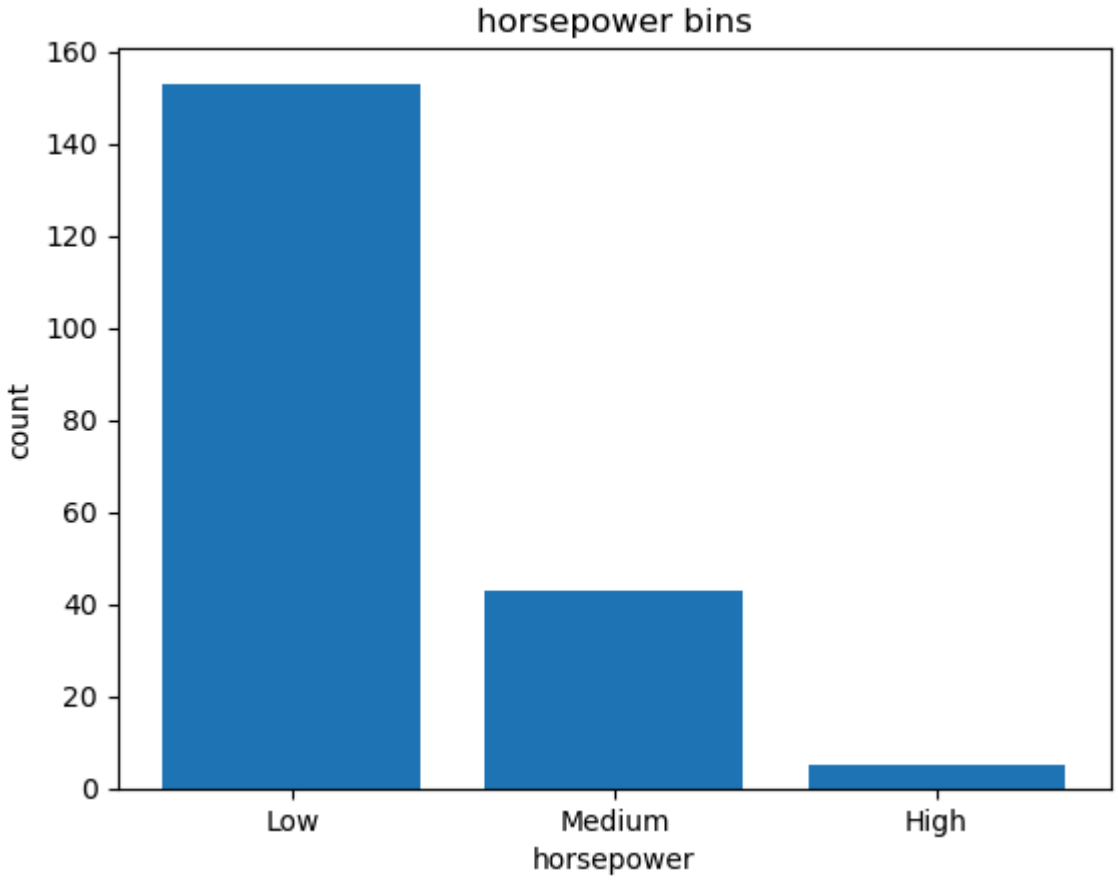
Out[149]:
```
horsepower-binned
Low       153
Medium     43
High        5
Name: count, dtype: int64
```

Plot the distribution of each bin:

In [150]:
```
%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")
```

Out[150]:
```
Text(0.5, 1.0, 'horsepower bins')
```



Look at the data frame 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

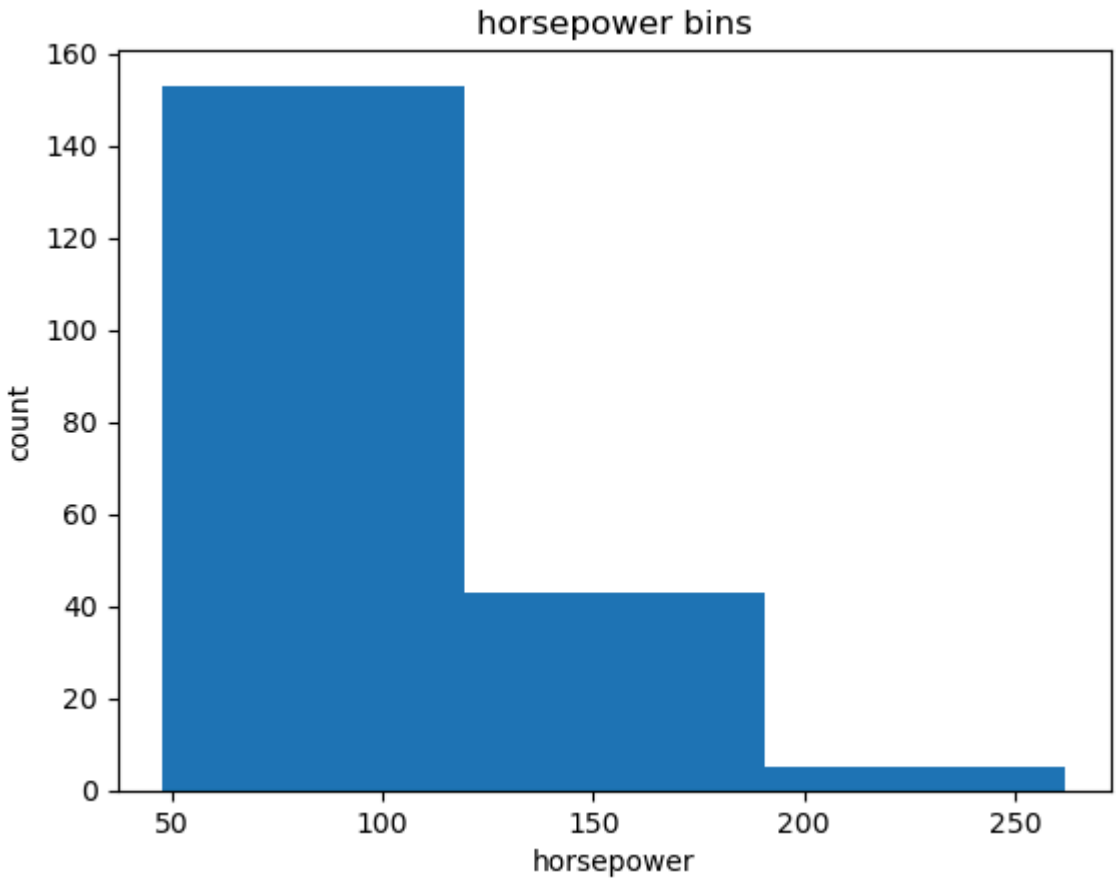
Normally, we use a histogram to visualize the distribution of bins we created above.

In [151]:
```
%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")
```

Out[151]:
```
Text(0.5, 1.0, 'horsepower bins')
```



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

Indicator 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 use indicator variables?

You use indicator variables so you can use categorical variables for regression analysis in the later modules.
Example

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, you can convert "fuel-type" to indicator variables.

Use the Panda method 'get_dummies' to assign numerical values to different categories of fuel type.

In [152]:
```
df.columns
```

Out[152]:
```
Index(['symboling', 'normalized-losses', 'make', 'fuel-type', 'aspiratio
n',
       'num-of-doors', 'body-style', 'drive-wheels', 'engine-location',
       'wheel-base', 'length', 'width', 'height', 'curb-weight', 'engine-t
ype',
       'num-of-cylinders', 'engine-size', 'fuel-system', 'bore', 'stroke',
       'compression-ratio', 'horsepower', 'peak-rpm', 'city-mpg',
       'highway-mpg', 'price', 'city-L/100km', 'horsepower-binned'],
      dtype='object')
```

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

In [153]:
```
dummy_variable_1 = pd.get_dummies(df["fuel-type"])
dummy_variable_1.head()
```

Out[153]:

	diesel	gas
0	False	True
1	False	True
2	False	True
3	False	True
4	False	True

Change the column names for clarity:

In [154]:
```
dummy_variable_1.rename(columns={'gas':'fuel-type-gas', 'diesel':'fuel-type-
dummy_variable_1.head()
```

Out[154]:

	fuel-type-diesel	fuel-type-gas
0	False	True
1	False	True
2	False	True
3	False	True
4	False	True

In [155]:
```
type(dummy_variable_1)
```

Out[155]:
```
pandas.core.frame.DataFrame
```

In [156]:
```
dummy_variable_1 = dummy_variable_1.astype(int)
dummy_variable_1.head()
```

Out[156]:

	fuel-type-diesel	fuel-type-gas
0	0	1
1	0	1
2	0	1
3	0	1
4	0	1

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

In [157]:
```
# 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)
```

In [158]:
```
df.head(100)
```

Out[158]:

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	(
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	(
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	(
3	2	164	audi	std	four	sedan	fwd	front	99.8	(
4	2	164	audi	std	four	sedan	4wd	front	99.4	(
...	...	...	...	...	...	...	...	...	...	
95	2	168	nissan	std	two	hardtop	fwd	front	95.1	(
96	0	106	nissan	std	four	hatchback	fwd	front	97.2	(
97	0	106	nissan	std	four	sedan	fwd	front	97.2	(
98	0	128	nissan	std	four	sedan	fwd	front	100.4	(
99	0	108	nissan	std	four	wagon	fwd	front	100.4	(

100 rows × 29 columns

In [159]:
```
# Count how many gas and how many diesel
print(f"Number of fuel-type-gas: {dummy_variable_1['fuel-type-gas'].sum()}")
print(f"Number of fuel-type-diesel: {dummy_variable_1['fuel-type-diesel'].su
```

```
Number of fuel-type-gas: 181
Number of fuel-type-diesel: 20
```

In [160]:
```
df['fuel-type-gas'].value_counts()
```

Out[160]:
```
fuel-type-gas
1    181
0     20
Name: count, dtype: int64
```

In [161]:
```
df['fuel-type-diesel'].value_counts()
```

Out[161]:
```
fuel-type-diesel
0    181
1     20
Name: count, dtype: int64
```

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

In [162]:
```
df = df.drop('fuel-type-diesel', axis=1)
```

In [163]:
```
df.head()
```

Out[163]:

	symboling	normalized-losses	make	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	
0	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.
1	3	122	alfa-romero	std	two	convertible	rwd	front	88.6	0.
2	1	122	alfa-romero	std	two	hatchback	rwd	front	94.5	0.
3	2	164	audi	std	four	sedan	fwd	front	99.8	0.
4	2	164	audi	std	four	sedan	4wd	front	99.4	0.

5 rows × 28 columns

Create an indicator variable for the column "aspiration"

In [164]:
```
# get indicator variables of aspiration and assign it to data frame "dummy_v
dummy_variable_2 = pd.get_dummies(df["aspiration"])
# change column names for clarity
dummy_variable_2.rename(columns= {'std':'aspiration-std', 'turbo': 'aspirati
# cast dummy-variable to type int
dummy_variable_2 = dummy_variable_2.astype(int)
dummy_variable_2.head()
```

Out[164]:

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

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

In [165]:
```
# merge
df = pd.concat([df,dummy_variable_2], axis = 1)

# drop column aspiration
df.drop("aspiration", axis = 1, inplace = True)
df.head()
```

In [172]:
```
# Count how many std and how many turbo to choose which dummy variable to dr
print(f"Number of aspiration-std: {dummy_variable_2['aspiration-std'].sum()}
print(f"Number of aspiration-turbo: {dummy_variable_2['aspiration-turbo'].su
```

```
Number of aspiration-std: 165
Number of aspiration-turbo: 36
```

In [173]:
```
# drop aspiration-turbo to avoid confusion later
df.drop('aspiration-turbo', axis = 1, inplace =True)
```

In [174]:
```
df.head()
```

Out[174]:

	symboling	normalized-losses	make	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	length	
0	3	122	alfa-romero	two	convertible	rwd	front	88.6	0.811148	0.8
1	3	122	alfa-romero	two	convertible	rwd	front	88.6	0.811148	0.8
2	1	122	alfa-romero	two	hatchback	rwd	front	94.5	0.822681	0.9
3	2	164	audi	four	sedan	fwd	front	99.8	0.848630	0.9
4	2	164	audi	four	sedan	4wd	front	99.4	0.848630	0.9

5 rows × 28 columns

Save the new csv:

In [175]:
```
df.to_csv('clean_auto_df.csv')
```