In [6]:
```
#Reading the dataset from the URL and adding the related headers
# Import pandas library
import pandas as pd
import matplotlib.pylab as plt
path = "https://archive.ics.uci.edu/ml/machine-learning-databases/autos/imports-85.data?utm_medium=Exinfluencer&utm_source=Ex
# Read the online file by the URL provides above, and assign it to variable "df"

df = pd.read_csv(path , header=None)
df.head()
```

Out[6]:

	0	1	2	3	4	5	6	7	8	9	...	16	17	18	19	20	21	22	23	24	25
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	13495
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	111	5000	21	27	16500
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	154	5000	19	26	16500
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	102	5500	24	30	13950
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	115	5500	18	22	17450

5 rows × 26 columns

In [3]:
```
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"]
```

In [29]:
```
df.columns = headers
df.head(4)
```

Out[29]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepo
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	

4 rows × 26 columns

In [30]:
```
df.head(5)
```

Out[30]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepo
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	

5 rows × 26 columns

In [45]:
```
import numpy as np

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

Out[45]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepo
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	

5 rows × 26 columns

In [32]:
```
#Evaluating for Missing Data
#The missing values are converted by default. We use the following functions to identify these missing values. There are two

#.isnull()
#.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()
type(missing_data)
missing_data.head(5)
```

Out[32]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepower
0	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
1	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
2	False	True	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False	...	False	False	False	False	False	False

5 rows × 26 columns

In [15]:
```
#Count missing values in each column
for column in missing_data.columns.values.tolist():
    print(column)
    print (missing_data[column].value_counts())
    print("")
```

```
0
False    205
Name: 0, dtype: int64

1
False    164
True      41
Name: 1, dtype: int64

2
False    205
Name: 2, dtype: int64

3
False    205
Name: 3, dtype: int64

4
False    205
```

In [34]:
```
#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
```

In [41]:
```
#Replace "NaN" with mean value in "normalized-losses" column

df["normalized-losses"].replace(np.nan, avg_norm_loss, inplace=True)
df.head(5)
```

Out[41]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepo
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	

5 rows × 26 columns

In [39]:
```
#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
```

In [40]:
```
#Replace "NaN" with the mean value in the "bore" column
df["bore"].replace(np.nan, avg_bore, inplace=True)
```

In [46]:
```
df.head(5)
```

Out[46]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepo
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	

5 rows × 26 columns

In [47]:
```
#Calculate the mean value for the "stroke" column
avg_stroke = df["stroke"].astype("float").mean(axis=0)
print("Average of stroke:", avg_stroke)
#Replace "NaN" with the mean value in the "stroke" column
df["stroke"].replace(np.nan, avg_stroke, inplace=True)
```

```
Average of stroke: 3.255422885572139
```

In [48]:
```
#Calculate the mean value for the "horsepower" column
avg_horsepower = df['horsepower'].astype('float').mean(axis=0)
print("Average horsepower:", avg_horsepower)
```

```
Average horsepower: 104.25615763546799
```

In [49]:
```
#Replace "NaN" with the mean value in the "horsepower" column
df['horsepower'].replace(np.nan, avg_horsepower, inplace=True)
```

In [50]:
```
avg_peakrpm=df['peak-rpm'].astype('float').mean(axis=0)
print("Average peak rpm:", avg_peakrpm)
```

```
Average peak rpm: 5125.369458128079
```

In [51]:
```
df['peak-rpm'].replace(np.nan, avg_peakrpm, inplace=True)
```

In [52]:
```
#To see which values are present in a particular column, we can use the ".value_counts()" method:
df['num-of-doors'].value_counts()
```

Out[52]:
```
four    114
two      89
Name: num-of-doors, dtype: int64
```

In [53]:
```
#****We can see that four doors are the most common type. We can also use the ".idxmax()" method to calculate the most common
df['num-of-doors'].value_counts().idxmax()
```

Out[53]:
```
'four'
```

In [55]:
```
#replace the missing 'num-of-doors' values by the most frequent
df["num-of-doors"].replace(np.nan, "four", inplace=True)
```

In [56]:
```
# 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()
```

Out[56]:

	symboling	normalized-losses	make	fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	engine-size	fuel-system	bore	stroke	compression-ratio	horsepo
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	130	mpfi	3.47	2.68	9.0	
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	152	mpfi	2.68	3.47	9.0	
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	109	mpfi	3.19	3.40	10.0	
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	136	mpfi	3.19	3.40	8.0	

5 rows × 26 columns

In [59]:
```
#Let's list the data types for each column
df.dtypes
```

Out[59]:
```
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
```

In [60]:
```
#<h4>Convert data types to proper format</h4>
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")
```

In [61]:
```
df.dtypes
```

Out[61]:
```
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
```

In [ ]:
```
#Now we have finally obtained the cleaned dataset with no missing values with all data in its proper format.
```

In [62]:
```
# 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[62]:

fuel-type	aspiration	num-of-doors	body-style	drive-wheels	engine-location	wheel-base	...	fuel-system	bore	stroke	compression-ratio	horsepower	peak-rpm	city-mpg	highway-mpg	price	c L/100
gas	std	two	convertible	rwd	front	88.6	...	mpfi	3.47	2.68	9.0	111	5000.0	21	27	13495.0	11.190
gas	std	two	convertible	rwd	front	88.6	...	mpfi	3.47	2.68	9.0	111	5000.0	21	27	16500.0	11.190
gas	std	two	hatchback	rwd	front	94.5	...	mpfi	2.68	3.47	9.0	154	5000.0	19	26	16500.0	12.368
gas	std	four	sedan	fwd	front	99.8	...	mpfi	3.19	3.40	10.0	102	5500.0	24	30	13950.0	9.791
gas	std	four	sedan	4wd	front	99.4	...	mpfi	3.19	3.40	8.0	115	5500.0	18	22	17450.0	13.055

In [63]:
```
#According to the example above, transform mpg to L/100km in the column of "highway-mpg" and change the name of column to "hi
# 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[63]:

normalized-	fuel-	num-	body-	drive-	engine-	wheel-	fuel-	compression-	pe

	symboling	normalized-losses	make	fuel-type	aspiration	of-doors	body-style	drive-wheels	engine-location	wheel-base	...	fuel-system	bore	stroke	compression-ratio	horsepower	pe rp
0	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	mpfi	3.47	2.68	9.0	111	500
1	3	122	alfa-romero	gas	std	two	convertible	rwd	front	88.6	...	mpfi	3.47	2.68	9.0	111	500
2	1	122	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	...	mpfi	2.68	3.47	9.0	154	500
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	...	mpfi	3.19	3.40	10.0	102	550
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	...	mpfi	3.19	3.40	8.0	115	550

5 rows × 27 columns

In [64]:
```
#data_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
# replace (original value) by (original value)/(maximum value)
df['length'] = df['length']/df['length'].max()
df['width'] = df['width']/df['width'].max()
```

In [65]:
```
#According to the example above, normalize the column "height"
df['height'] = df['height']/df['height'].max()

# show the scaled columns
df[["length","width","height"]].head()
```

Out[65]:

	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

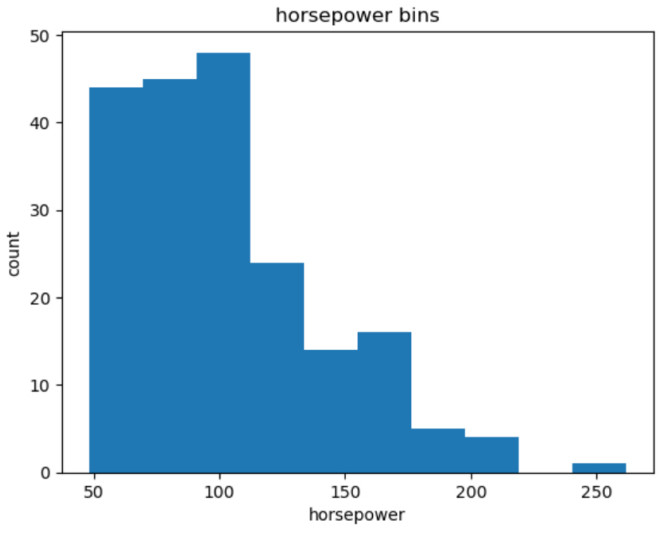
In [ ]:
```
#above we can see we've normalized "length", "width" and "height" in the range of \[0,1].
```

In [66]:
```
#Binning
# Binning is a process of transforming continuous numerical variables into discrete categorical 'bins' for grouped analysis.
#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?
#Convert data to correct format:
df["horsepower"]=df["horsepower"].astype(int, copy=True)
```

In [67]:
```
#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")
```

Out[67]:
```
Text(0.5, 1.0, 'horsepower bins')
```



In [68]:
```
#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.
bins = np.linspace(min(df["horsepower"]), max(df["horsepower"]), 4)
bins
```

Out[68]:
```
array([ 48.        , 119.33333333, 190.66666667, 262.        ])
```

In [69]:
```
#We set group names:
group_names = ['Low', 'Medium', 'High']
```

In [70]:
```
#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)
```

Out[70]:

	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

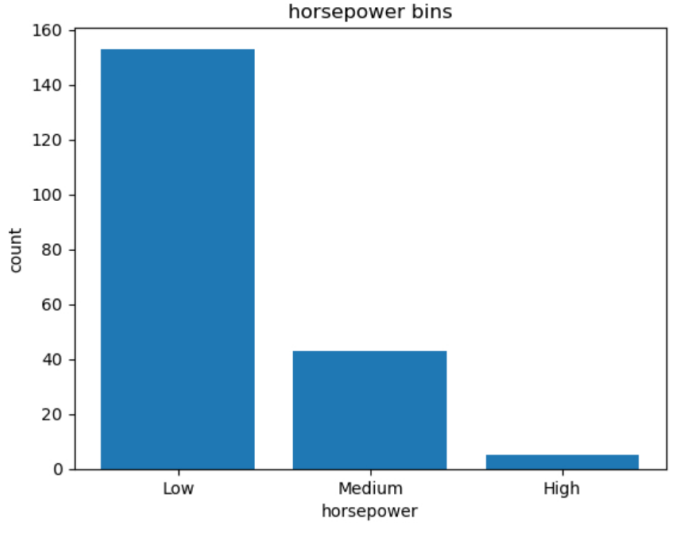
In [72]:
```
#Let's see the number of vehicles in each bin:
df["horsepower-binned"].value_counts()
```

Out[72]:
```
Low       153
Medium     43
High        5
Name: horsepower-binned, dtype: int64
```

In [74]:
```
#et's 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")
```

Out[74]:
```
Text(0.5, 1.0, 'horsepower bins')
```

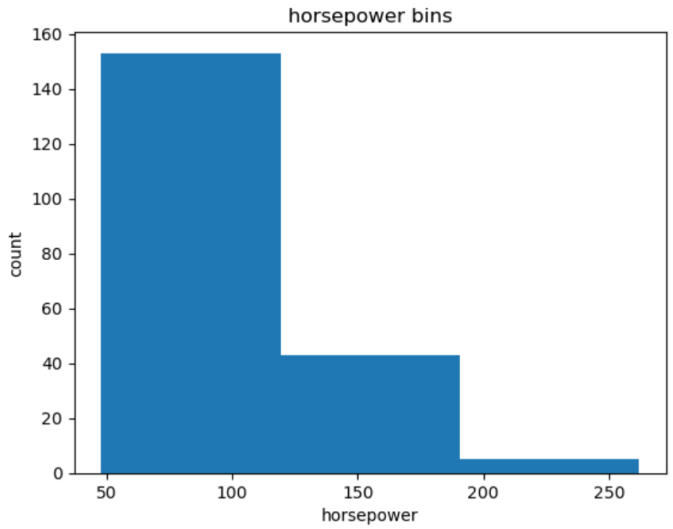


In [76]:
```
#he plot above shows the binning result for the attribute "horsepower".
%matplotlib inline
import matplotlib as plt
from matplotlib import pyplot


# 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[76]:
```
Text(0.5, 1.0, 'horsepower bins')
```



In [ ]:
```
#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' becaus

#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. T

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

In [77]:
```
df.columns
```

Out[77]:
```
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')
```

In [78]:
```
#Similar to before, create an indicator variable for the column "aspiration
# 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()
```

Out[78]:

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

In [79]:
```
#Merge the new dataframe to the original dataframe, then drop the column 'aspiration'.
# 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)
```

In [80]:
```
df.to_csv('clean_df.csv')
```

In [ ]: