PG2104 Exam, Knr: 1021

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
```

Problem 1: Conceptual Questions (10 points)

Use Markdown cell in your Jupyter Notebook and explain the following:

1. What are escape sequences? Explain any 3 escape sequences.? (2 points)

```
Escape sequences are ways to get around certain characters that would otherwise be illegal to put in strings.
'\t' - creates tab in text.
'\n' - creates a new line.
'\\' - backslash
```

2. What are data types? What are Python's built-in core data types. (2 points)

```
Variables in a program can store different types of data and different types of data can do different things.
Adding two numbers which are in the String data type will not perform arithmatics on them, thus you don't get the sum of
the two numbers, rather the strings get concatinated together. Example: a = str(5) and b = str(3) a + b = 53.

In python the built-in default data types are:

str, int, float, complex, list, tuple, range, dict, set, frozenset, bool, bytes, bytearray, memoryview and NoneType.
```

3. What do you mean by positive correlation and negative correlation? Give one example of each. (2 points)

Correlation is the relationship between two variables. It measures the strength between the variables. When studying correlation it is important to keep in mind;

```
Independant variables; The factor that causes the dependant variable to change.
Dependant variables; Are influenced by the independant variables.
```

By positive correlation one means that if there is an increase in the independant variable, it causes an increase in the dependant variable. For example there could be a positive correlation between good weather and the amount of ice cream sold at a shop.

By negative correlation means that if there is an increase in the independant variable there will be a decrease/fall in the dependant variable. For example there could be a negative correlation between the sales of umbrellas and sunny weather. As we get more sunny weather, less umbrellas are bought.

4. What do you mean by five-point summary of data? Which plot is used to reflect five-point summary of data? (2 points)

Five-point summary means showing a standard summary of the data. It involves calculating the following statistical quantities;

```
Median: The middle value of the data. (Also known as the 2nd quartile or 50th percentile)
1st quartile: The 25th percentile.
3rd quartile: The 75th percentile.
Minimum: The smalles observation in the data.
Maximum: The largest obeservation in the data.
```

A percentile indicates the value below which a percentage of observations in a group of observation fall into. (Geron, 2017, pg 52)

To reflect on a five-point summary it is standard to plot the data with a box-and-whisker plot.

5. What is difference between univariate, bivariate and multivariate analysis? Give one example each of the plots used for these analysis. (2 points)

Univariate analysis:

```
Uni(one)variate(variable) analysis refers to analysis involving a one variable. The data must contain one and only one variable. Examples
of such analysis is for example the five-point summary. One plot used for univariate analsysis is a pie chart.
```

Bivariate analysis:

```
Bi(two)variate(variable) analysis refers to analsysis between two variables. Such analysis helps study/reveal the relationship (or lack t
here of) between two variables. One plot used for bivariate analysis is a scatter plot where the two variables are plotted against eachot
her.
```

Multivariate analysis:

```
Multi(many)variate(variable) analysis refers to analysis involving many variables(more than two). One such analysis method called cluster
ing analysis, where one wants to split the datapoints into groups known as clusters, a scatter plot is used to  show the groups of points
in the data.
```

Problem 2: Programming Problem (20 points)

1. Write a Python program to print the following pattern (10 points):

```
#######
#     #
#     #
#     #
#     #
#     #
#######
```

In [2]:
```
def printPattern(size):
    print("Pattern of size", size)
    pattern = ""
    for i in range (0, size):
        if(i == 0 or i == size-1):
            pattern += "\t"
            pattern += "#"*size + "\n"
        else:
            pattern += "\t#" + " "*(size-3) + " #\n"
    print(pattern)
```

In [3]:
```
printPattern(7)
printPattern(10)
```

```
Pattern of size 7
        #######
        #     #
        #     #
        #     #
        #     #
        #     #
        #######

Pattern of size 10
        ##########
        #        #
        #        #
        #        #
        #        #
        #        #
        #        #
        #        #
        #        #
        ##########
```

2. Write a Python program to input marks in 3 subjects; compute average and then calculate grade as per following guidelines (10 points):

For this task I create multiple functions to validate input before taking more input. I then calculate the average of the grades that the user inputs before chaning the grades to A-F

equivalents according to the exam text.

In [4]:
```python
# Function to get student id while validating that we only get integers.
def getStudentId(text):
    while True:
        try:
            userInput = int(input(text))
            print()
            return userInput             # Only returns int values
        except ValueError:
            print("Input must be integer.")
            print()
            continue
```

In [5]:
```python
# Function to set grades and validate that the grades are between 0 and 100.
def getGrade(text):
    while True:
        try:
            userInput = int(input(text))
            if(userInput < 0 or userInput > 100):
                print("Grade must be a number between 0 and 100.")
                continue
            print()
            return userInput            # Only returns int values.
        except ValueError:
            print("Input must be integer.")
            print()
            continue
```

In [6]:
```python
# Function to start the functions to get ID and grades. Passes text to functions which is printed to the user.
def askUser():
    studentId = getStudentId("What is the students' id? \n")
    mathGrade = getGrade("What is their math grade? \n")
    pgrGrade = getGrade("What is their programming grade? \n")
    histGrade = getGrade("What is their history grade? \n")
    printStudentInfo(studentId, mathGrade, pgrGrade, histGrade)
```

In [7]:
```
def calcGrade(grade):
    if(grade <= 39):
        return 'F'
    elif(grade <= 49):
        return 'E'
    elif(grade <= 59):
        return 'D'
    elif(grade <= 79):
        return 'C'
    elif(grade <= 89):
        return 'B'
    else:
        return 'A'
```

In [8]:
```
# Printing student info before calculating the average and the A-F equivalent of the grade.
def printStudentInfo(studentId, mathGrade, pgrGrade, histGrade):
    avg = 0
    avg = (mathGrade+pgrGrade+histGrade)/3
    math = calcGrade(mathGrade)
    pgr = calcGrade(pgrGrade)
    hist = calcGrade(histGrade)
    print('Student with ID:',studentId,
    '\nTheir average grade is:', round(avg, 2),
    '\nMath grade:', math,
    '\nProgramming grade:', pgr,
    '\nHistory grade:', hist)
```

In [9]:
```
askUser()
```

```
What is the students' id?
NaN
Input must be integer.

What is the students' id?
1234

What is their math grade?
-1
Grade must be a number between 0 and 100.
What is their math grade?
101
Grade must be a number between 0 and 100.
What is their math grade?
70

What is their programming grade?
95

What is their history grade?
30

Student with ID: 1234
Their average grade is: 65.0
Math grade: C
Programming grade: A
History grade: F
```

Problem 3: Data Wrangling and Plotting Problem (50 points)

For this exercise, we will use Automobile Dataset which has been provided to you in the file auto_data.csv. Perform the following tasks:

1. Read the csv file in Pandas and create a DataFrame named Auto_df. What is the shape of Auto_df. Print first 7 and last 7 rows of Auto_df. (2 points)

In [10]:
```
# Reading the csv file and creating dataframe.
Auto_df = pd.read_csv('auto_data.csv')
```

In [11]:
```
# I want to see all the columns so I change the options of pandas with set_options().
pd.set_option('display.max_columns', None)
```

In [12]:
```
# Shape of the df, rows, columns
Auto_df.shape
```

Out[12]:
```
(205, 26)
```

In [13]:
```
# Auto_df.head(7) works fine too.
Auto_df[:7]
```

Out[13]:

	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
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	3.47	2.68
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	3.47	2.68
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823	ohcv	six	152	mpfi	2.68	3.47
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337	ohc	four	109	mpfi	3.19	3.4
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824	ohc	five	136	mpfi	3.19	3.4
5	2	?	audi	gas	std	two	sedan	fwd	front	99.8	177.3	66.3	53.1	2507	ohc	five	136	mpfi	3.19	3.4
6	1	158	audi	gas	std	four	sedan	fwd	front	105.8	192.7	71.4	55.7	2844	ohc	five	136	mpfi	3.19	3.4

In [14]:
```
# Could just solve this with Auto_df.tail(7) but slicing is cooler
# flip the dataframe around and slice the first 7 entries.
Auto_df[::-1][:7]
```

Out[14]:

	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	c
204	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	188.8	68.9	55.5	3062	ohc	four	141	mpfi	3.78	3.15	
203	-1	95	volvo	diesel	turbo	four	sedan	rwd	front	109.1	188.8	68.9	55.5	3217	ohc	six	145	idi	3.01	3.4	
202	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	188.8	68.9	55.5	3012	ohcv	six	173	mpfi	3.58	2.87	
201	-1	95	volvo	gas	turbo	four	sedan	rwd	front	109.1	188.8	68.8	55.5	3049	ohc	four	141	mpfi	3.78	3.15	
200	-1	95	volvo	gas	std	four	sedan	rwd	front	109.1	188.8	68.9	55.5	2952	ohc	four	141	mpfi	3.78	3.15	
199	-1	74	volvo	gas	turbo	four	wagon	rwd	front	104.3	188.8	67.2	57.5	3157	ohc	four	130	mpfi	3.62	3.15	
198	-2	103	volvo	gas	turbo	four	sedan	rwd	front	104.3	188.8	67.2	56.2	3045	ohc	four	130	mpfi	3.62	3.15	

2. Do you find anything unusual in the dataset? If yes, then replace these unusual values with NaN values. (3 points)

The datatypes in this dataset are all over the place. Many columns are objects indicating they should have mixed values but when looking further into it they don't include mixed datatypes. There are some value set to '?' and 0 null values.

For this question I assume that you want me to replace the '?' values with NaN values.

In [15]:
```
Auto_df.info()
```

```
<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  205 non-null    object 
 2   make               205 non-null    object 
 3   fuel-type          205 non-null    object 
 4   aspiration         205 non-null    object 
 5   num-of-doors       205 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    object 
 19  stroke             205 non-null    object 
 20  compression-ratio  205 non-null    float64
 21  horsepower         205 non-null    object 
 22  peak-rpm           205 non-null    object 
 23  city-mpg           205 non-null    int64  
 24  highway-mpg        205 non-null    int64  
 25  price              205 non-null    object 
dtypes: float64(5), int64(5), object(16)
memory usage: 41.8+ KB
```

In [16]:
```
# The datatypes in this dataset seem to be wrong.
# Columns: 'bore', 'stroke' should be float64.
# Columns: 'normalized-losses', 'horsepower', 'peak-rpm', 'price' should be int64
# Columns: 'make', 'fuel-type', 'aspiration', 'num-of-doors', 'body-style', 'drive-wheels', 'engine-location', 'engine-type'
# 'num-of-cylinders', 'fuel-system' should be string
Auto_df.dtypes
```

Out[16]:
```
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 [17]:
```
Auto_df.head()
```

Out[17]:

	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
0	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	3.47	2.68
1	3	?	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	3.47	2.68
2	1	?	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823	ohcv	six	152	mpfi	2.68	3.47
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337	ohc	four	109	mpfi	3.19	3.4
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824	ohc	five	136	mpfi	3.19	3.4

In [18]:
```
Auto_df = Auto_df.replace('?', np.nan)
```

3. Which columns have NaN values and how many? Finally, handle these null values using any strategy taught in the class. (5 points)

In [19]:
```
# Showing columns with NaN values and how many there are.
Auto_df.isna().sum()
```

Out[19]:
```
symboling             0
normalized-losses    41
make                  0
fuel-type             0
aspiration            0
num-of-doors          2
body-style            0
drive-wheels          0
engine-location       0
wheel-base            0
length                0
width                 0
height                0
curb-weight           0
engine-type           0
num-of-cylinders      0
engine-size           0
fuel-system           0
bore                  4
stroke                4
compression-ratio     0
horsepower            2
peak-rpm              2
city-mpg              0
highway-mpg           0
price                 4
dtype: int64
```

I will go through these columns one by one, take a look at the data and make descisions on what to replace the NaN values with below:

Column: 'normalized-losses'

In [20]:
```
# Finding min, max and mean value. Have to read the values as float64 even though they are integer values because
# np.nan is classified as a float, not an int.
print("Min value in 'normalized-losses' column :", np.nanmin(Auto_df['normalized-losses'].astype('float64')))
print("Max value in 'normalized-losses' column :", np.nanmax(Auto_df['normalized-losses'].astype('float64')))
print("Mean value in 'normalized-losses' column :", Auto_df['normalized-losses'].astype('float64').mean())
```

```
Min value in 'normalized-losses' column : 65.0
Max value in 'normalized-losses' column : 256.0
Mean value in 'normalized-losses' column : 122.0
```

In [22]:
```
Auto_df['normalized-losses'].value_counts()
```

Out[22]:

161	11
91	8
150	7
128	6
134	6
104	6
95	5
102	5
103	5
74	5
85	5
168	5
94	5
65	5
106	4
122	4
148	4
118	4
93	4
101	3
125	3
137	3
154	3
83	3
115	3
119	2
87	2
194	2
197	2
108	2
89	2
164	2
158	2
145	2
192	2
188	2
81	2
110	2
113	2
129	2
153	2
107	1
78	1
186	1
231	1
77	1
98	1
121	1
90	1
142	1

```
256     1
Name: normalized-losses, dtype: int64
```

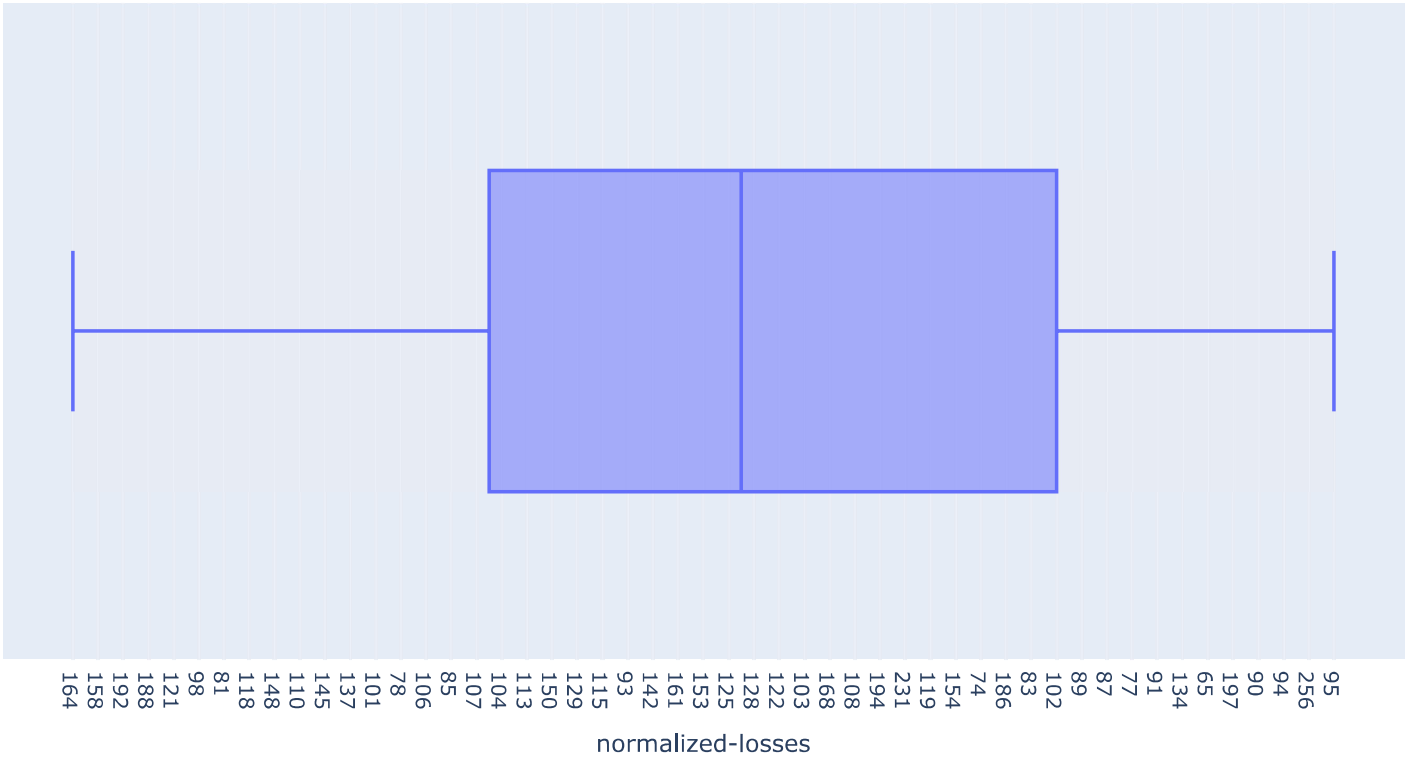
In [23]:
```
# Finding the rows where 'normalized-losses' = NaN
Auto_df.loc[pd.isna(Auto_df['normalized-losses'])]
```

Out[23]:

	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	s
0	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	3.47	
1	3	NaN	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	3.47	
2	1	NaN	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823	ohcv	six	152	mpfi	2.68	
5	2	NaN	audi	gas	std	two	sedan	fwd	front	99.8	177.3	66.3	53.1	2507	ohc	five	136	mpfi	3.19	
7	1	NaN	audi	gas	std	four	wagon	fwd	front	105.8	192.7	71.4	55.7	2954	ohc	five	136	mpfi	3.19	
9	0	NaN	audi	gas	turbo	two	hatchback	4wd	front	99.5	178.2	67.9	52.0	3053	ohc	five	131	mpfi	3.13	
14	1	NaN	bmw	gas	std	four	sedan	rwd	front	103.5	189.0	66.9	55.7	3055	ohc	six	164	mpfi	3.31	
15	0	NaN	bmw	gas	std	four	sedan	rwd	front	103.5	189.0	66.9	55.7	3230	ohc	six	209	mpfi	3.62	
16	0	NaN	bmw	gas	std	two	sedan	rwd	front	103.5	193.8	67.9	53.7	3380	ohc	six	209	mpfi	3.62	
17	0	NaN	bmw	gas	std	four	sedan	rwd	front	110.0	197.0	70.9	56.3	3505	ohc	six	209	mpfi	3.62	
43	0	NaN	isuzu	gas	std	four	sedan	rwd	front	94.3	170.7	61.8	53.5	2337	ohc	four	111	2bbl	3.31	
44	1	NaN	isuzu	gas	std	two	sedan	fwd	front	94.5	155.9	63.6	52.0	1874	ohc	four	90	2bbl	3.03	
45	0	NaN	isuzu	gas	std	four	sedan	fwd	front	94.5	155.9	63.6	52.0	1909	ohc	four	90	2bbl	3.03	
46	2	NaN	isuzu	gas	std	two	hatchback	rwd	front	96.0	172.6	65.2	51.4	2734	ohc	four	119	spfi	3.43	
48	0	NaN	jaguar	gas	std	four	sedan	rwd	front	113.0	199.6	69.6	52.8	4066	dohc	six	258	mpfi	3.63	
49	0	NaN	jaguar	gas	std	two	sedan	rwd	front	102.0	191.7	70.6	47.8	3950	ohcv	twelve	326	mpfi	3.54	
63	0	NaN	mazda	diesel	std	NaN	sedan	fwd	front	98.8	177.8	66.5	55.5	2443	ohc	four	122	idi	3.39	
66	0	NaN	mazda	diesel	std	four	sedan	rwd	front	104.9	175.0	66.1	54.4	2700	ohc	four	134	idi	3.43	
71	-1	NaN	mercedes-benz	gas	std	four	sedan	rwd	front	115.6	202.6	71.7	56.5	3740	ohcv	eight	234	mpfi	3.46	
73	0	NaN	mercedes-benz	gas	std	four	sedan	rwd	front	120.9	208.1	71.7	56.7	3900	ohcv	eight	308	mpfi	3.8	
74	1	NaN	mercedes-benz	gas	std	two	hardtop	rwd	front	112.0	199.2	72.0	55.4	3715	ohcv	eight	304	mpfi	3.8	
75	1	NaN	mercury	gas	turbo	two	hatchback	rwd	front	102.7	178.4	68.0	54.8	2910	ohc	four	140	mpfi	3.78	
82	3	NaN	mitsubishi	gas	turbo	two	hatchback	fwd	front	95.9	173.2	66.3	50.2	2833	ohc	four	156	spdi	3.58	
83	3	NaN	mitsubishi	gas	turbo	two	hatchback	fwd	front	95.9	173.2	66.3	50.2	2921	ohc	four	156	spdi	3.59	
84	3	NaN	mitsubishi	gas	turbo	two	hatchback	fwd	front	95.9	173.2	66.3	50.2	2926	ohc	four	156	spdi	3.59	
109	0	NaN	peugot	gas	std	four	wagon	rwd	front	114.2	198.9	68.4	58.7	3230	l	four	120	mpfi	3.46	
110	0	NaN	peugot	diesel	turbo	four	wagon	rwd	front	114.2	198.9	68.4	58.7	3430	l	four	152	idi	3.7	
113	0	NaN	peugot	gas	std	four	wagon	rwd	front	114.2	198.9	68.4	56.7	3285	l	four	120	mpfi	3.46	

	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	s
114	0	NaN	peugot	diesel	turbo	four	wagon	rwd	front	114.2	198.9	68.4	58.7	3485	l	four	152	idi	3.7	
124	3	NaN	plymouth	gas	turbo	two	hatchback	rwd	front	95.9	173.2	66.3	50.2	2818	ohc	four	156	spdi	3.59	
126	3	NaN	porsche	gas	std	two	hardtop	rwd	rear	89.5	168.9	65.0	51.6	2756	ohcf	six	194	mpfi	3.74	
127	3	NaN	porsche	gas	std	two	hardtop	rwd	rear	89.5	168.9	65.0	51.6	2756	ohcf	six	194	mpfi	3.74	
128	3	NaN	porsche	gas	std	two	convertible	rwd	rear	89.5	168.9	65.0	51.6	2800	ohcf	six	194	mpfi	3.74	
129	1	NaN	porsche	gas	std	two	hatchback	rwd	front	98.4	175.7	72.3	50.5	3366	dohcv	eight	203	mpfi	3.94	
130	0	NaN	renault	gas	std	four	wagon	fwd	front	96.1	181.5	66.5	55.2	2579	ohc	four	132	mpfi	3.46	
131	2	NaN	renault	gas	std	two	hatchback	fwd	front	96.1	176.8	66.6	50.5	2460	ohc	four	132	mpfi	3.46	
181	-1	NaN	toyota	gas	std	four	wagon	rwd	front	104.5	187.8	66.5	54.1	3151	dohc	six	161	mpfi	3.27	
189	3	NaN	volkswagen	gas	std	two	convertible	fwd	front	94.5	159.3	64.2	55.6	2254	ohc	four	109	mpfi	3.19	
191	0	NaN	volkswagen	gas	std	four	sedan	fwd	front	100.4	180.2	66.9	55.1	2661	ohc	five	136	mpfi	3.19	
192	0	NaN	volkswagen	diesel	turbo	four	sedan	fwd	front	100.4	180.2	66.9	55.1	2579	ohc	four	97	idi	3.01	
193	0	NaN	volkswagen	gas	std	four	wagon	fwd	front	100.4	183.1	66.9	55.1	2563	ohc	four	109	mpfi	3.19	

In [21]:
```
fig = px.box(Auto_df, x="normalized-losses")
fig.show()
```



In [24]:
```
# For this column I can use the mean since the data isn't very skewed
Auto_df['normalized-losses'].fillna(Auto_df['normalized-losses'].astype('float64').mean(), inplace = True)
```

Column: 'num-of-doors'

In [25]:
```
# Counts in the 'num-of-doors' column
Auto_df['num-of-doors'].value_counts()
```

Out[25]:
```
four    114
two      89
Name: num-of-doors, dtype: int64
```

In [26]:
```
# Finding the rows where 'num-of-doors' = NaN
Auto_df.loc[pd.isna(Auto_df['num-of-doors'])]
```

Out[26]:

	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	co
27	1	148	dodge	gas	turbo	NaN	sedan	fwd	front	93.7	157.3	63.8	50.6	2191	ohc	four	98	mpfi	3.03	3.39	
63	0	122.0	mazda	diesel	std	NaN	sedan	fwd	front	98.8	177.8	66.5	55.5	2443	ohc	four	122	idi	3.39	3.39	

Looking at the other columns does not give us an decisive answer whether this should be a car with two or four doors so it's filled with ffill. Alternatively I could've just dropped these.

In [27]:
```
Auto_df['num-of-doors'].fillna(method='ffill', inplace = True)
```

Column: 'bore'

In [28]:
```
# Min, max and mean values
print("Min value in 'bore' column :", min(Auto_df['bore'].astype('float64')))
print("Max value in 'bore' column :", max(Auto_df['bore'].astype('float64')))
print("Mean value in 'bore' column :", round(Auto_df['bore'].astype('float64').mean(), 2))
```

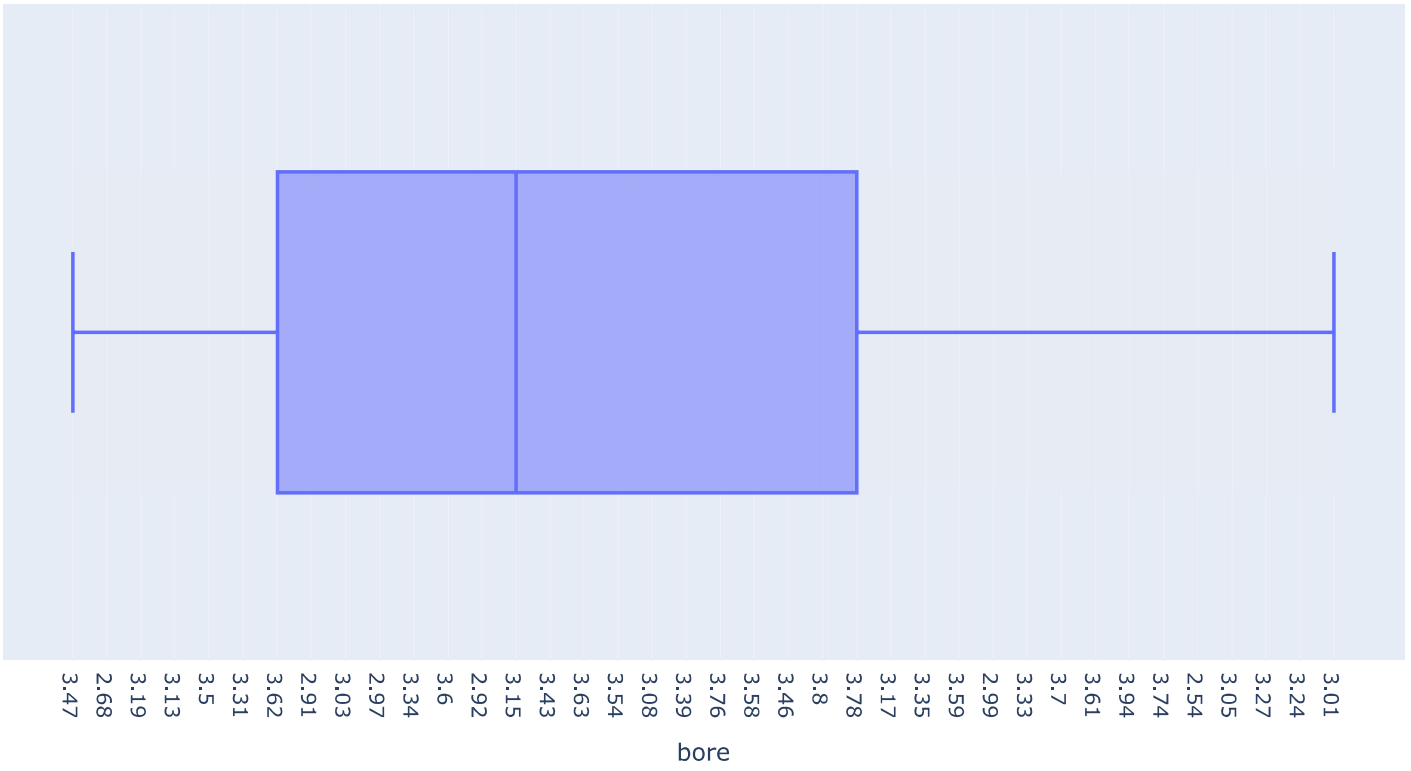
```
Min value in 'bore' column : 2.54
Max value in 'bore' column : 3.94
Mean value in 'bore' column : 3.33
```

In [29]:
```
# Locating NaN values in 'bore' column.
Auto_df.loc[pd.isna(Auto_df['bore'])]
```

Out[29]:

	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
55	3	150	mazda	gas	std	two	hatchback	rwd	front	95.3	169.0	65.7	49.6	2380	rotor	two	70	4bbl	NaN	NaN
56	3	150	mazda	gas	std	two	hatchback	rwd	front	95.3	169.0	65.7	49.6	2380	rotor	two	70	4bbl	NaN	NaN
57	3	150	mazda	gas	std	two	hatchback	rwd	front	95.3	169.0	65.7	49.6	2385	rotor	two	70	4bbl	NaN	NaN
58	3	150	mazda	gas	std	two	hatchback	rwd	front	95.3	169.0	65.7	49.6	2500	rotor	two	80	mpfi	NaN	NaN

In [30]:
```
fig = px.box(Auto_df, x="bore")
fig.show()
```



In [31]:
```
# For 'bore' column I can use the median since the data is skewed
Auto_df['bore'].fillna(round(Auto_df['bore'].astype('float64').median(), 2), inplace = True)
```

Column: 'stroke'

In [32]:
```
# Min, max and mean values
print("Min value in 'stroke' column :", min(Auto_df['stroke'].astype('float64')))
print("Max value in 'stroke' column :", max(Auto_df['stroke'].astype('float64')))
print("Mean value in 'stroke' column :", round(Auto_df['stroke'].astype('float64').mean(), 2))
```

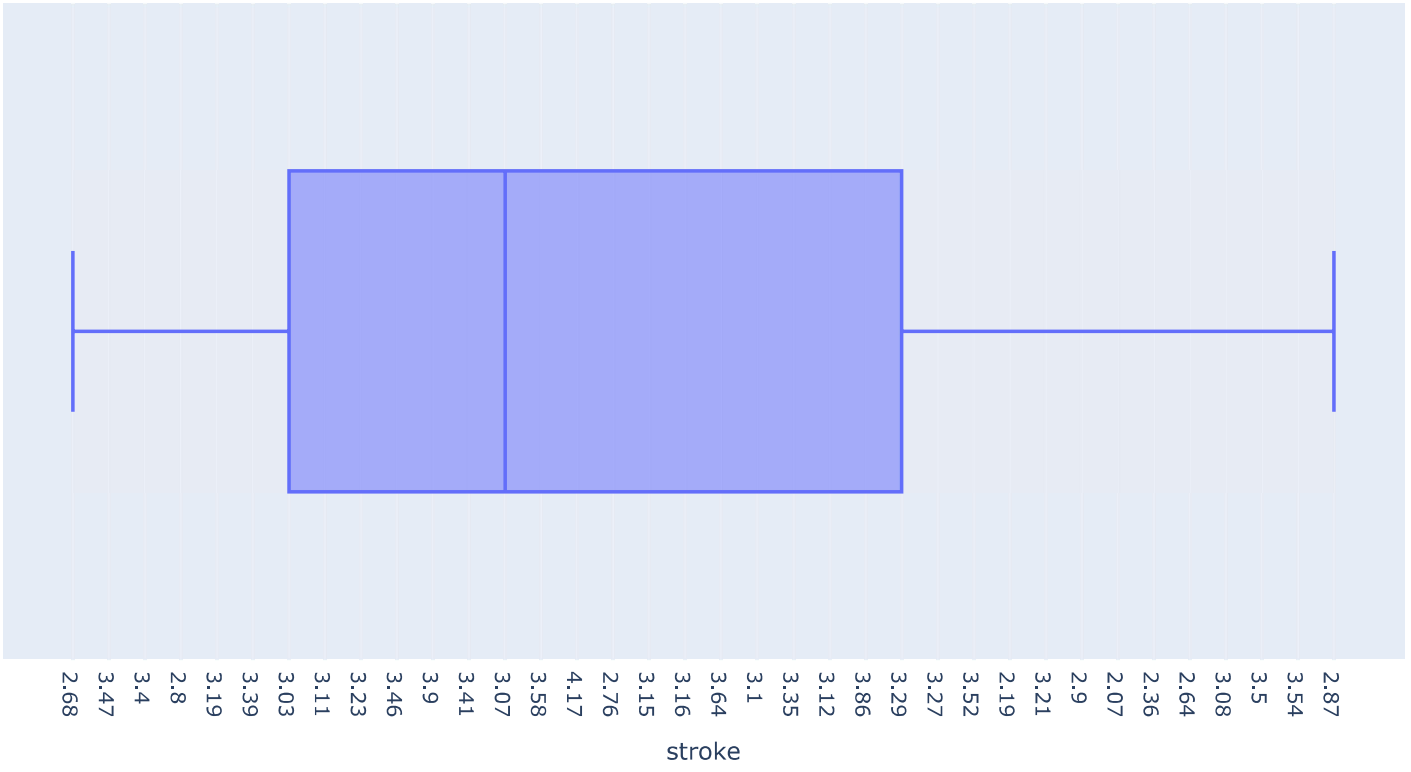
```
Min value in 'stroke' column : 2.07
Max value in 'stroke' column : 4.17
Mean value in 'stroke' column : 3.26
```

In [33]:
```
# Locating NaN values in 'bore' column.
Auto_df.loc[pd.isna(Auto_df['stroke'])]
```

Out[33]:

	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
55	3	150	mazda	gas	std	two	hatchback	rwd	front	95.3	169.0	65.7	49.6	2380	rotor	two	70	4bbl	3.31	NaN
56	3	150	mazda	gas	std	two	hatchback	rwd	front	95.3	169.0	65.7	49.6	2380	rotor	two	70	4bbl	3.31	NaN
57	3	150	mazda	gas	std	two	hatchback	rwd	front	95.3	169.0	65.7	49.6	2385	rotor	two	70	4bbl	3.31	NaN
58	3	150	mazda	gas	std	two	hatchback	rwd	front	95.3	169.0	65.7	49.6	2500	rotor	two	80	mpfi	3.31	NaN

In [34]:
```
fig = px.box(Auto_df, x="stroke")
fig.show()
```



In [35]:
```
# For the 'stroke' value I will also use the median value.
Auto_df['stroke'].fillna(round(Auto_df['stroke'].astype('float64').median(), 2), inplace = True)
```

Column: 'horsepower'

In [36]:
```
# min, max and mean values
print("Min value in 'horsepower' column :", np.nanmin(Auto_df['horsepower'].astype('float64')))
print("Max value in 'horsepower' column :", np.nanmax(Auto_df['horsepower'].astype('float64')))
print("Mean value in 'horsepower' column :", round(Auto_df['horsepower'].astype('float64').mean(), 2))
```

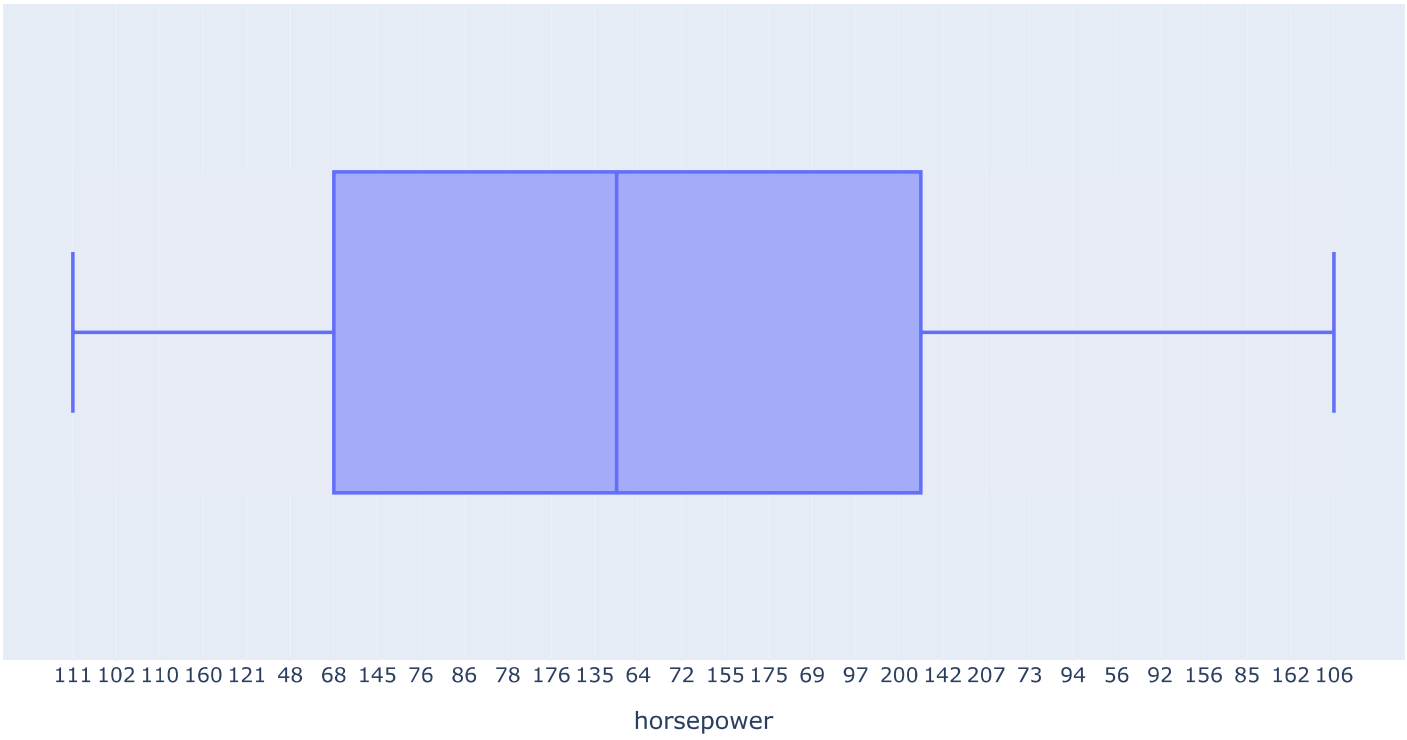
```
Min value in 'horsepower' column : 48.0
Max value in 'horsepower' column : 288.0
Mean value in 'horsepower' column : 104.26
```

In [37]:
```
# Locating NaN values in 'horsepower' column.
Auto_df.loc[pd.isna(Auto_df['horsepower'])]
```

Out[37]:

	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
130	0	122.0	renault	gas	std	four	wagon	fwd	front	96.1	181.5	66.5	55.2	2579	ohc	four	132	mpfi	3.46	3.9
131	2	122.0	renault	gas	std	two	hatchback	fwd	front	96.1	176.8	66.6	50.5	2460	ohc	four	132	mpfi	3.46	3.9

In [38]:
```
fig = px.box(Auto_df, x="horsepower")
fig.show()
```



In [39]:
```
# For the 'horsepower' value I will also use the median value.
Auto_df['horsepower'].fillna(round(Auto_df['horsepower'].astype('float64').mean(), 2), inplace = True)
```

Column: 'peak-rpm'

In [40]:
```
# min, max and mean values
print("Min value in 'peak-rpm' column :", np.nanmin(Auto_df['peak-rpm'].astype('float64')))
print("Max value in 'peak-rpm' column :", np.nanmax(Auto_df['peak-rpm'].astype('float64')))
print("Mean value in 'peak-rpm' column :", round(Auto_df['peak-rpm'].astype('float64').mean(), 2))
```

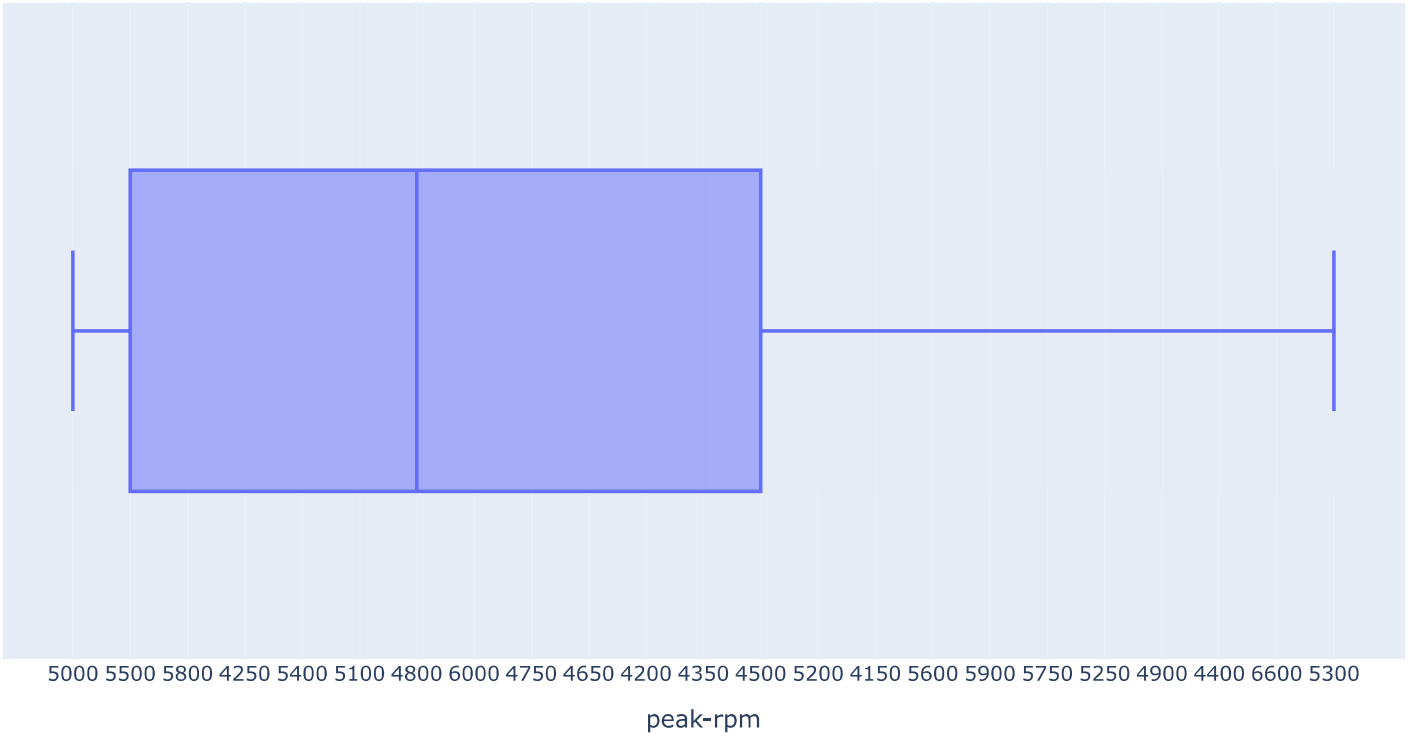
```
Min value in 'peak-rpm' column : 4150.0
Max value in 'peak-rpm' column : 6600.0
Mean value in 'peak-rpm' column : 5125.37
```

In [41]:
```
# Locating NaN values in 'peak-rpm' column.
Auto_df.loc[pd.isna(Auto_df['peak-rpm'])]
```

Out[41]:

	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
130	0	122.0	renault	gas	std	four	wagon	fwd	front	96.1	181.5	66.5	55.2	2579	ohc	four	132	mpfi	3.46	3.9
131	2	122.0	renault	gas	std	two	hatchback	fwd	front	96.1	176.8	66.6	50.5	2460	ohc	four	132	mpfi	3.46	3.9

In [42]:
```
fig = px.box(Auto_df, x="peak-rpm")
fig.show()
```



In [43]:
```
# For this column I can use the median since it is very skewed
Auto_df['peak-rpm'].fillna(Auto_df['peak-rpm'].median(), inplace = True)
```

Column: 'price'

In [44]:
```
# min, max and mean values
print("Min value in 'price' column :", np.nanmin(Auto_df['price'].astype('float64')))
print("Max value in 'price' column :", np.nanmax(Auto_df['price'].astype('float64')))
print("Mean value in 'price' column :", round(Auto_df['price'].astype('float64').mean(), 2))
```

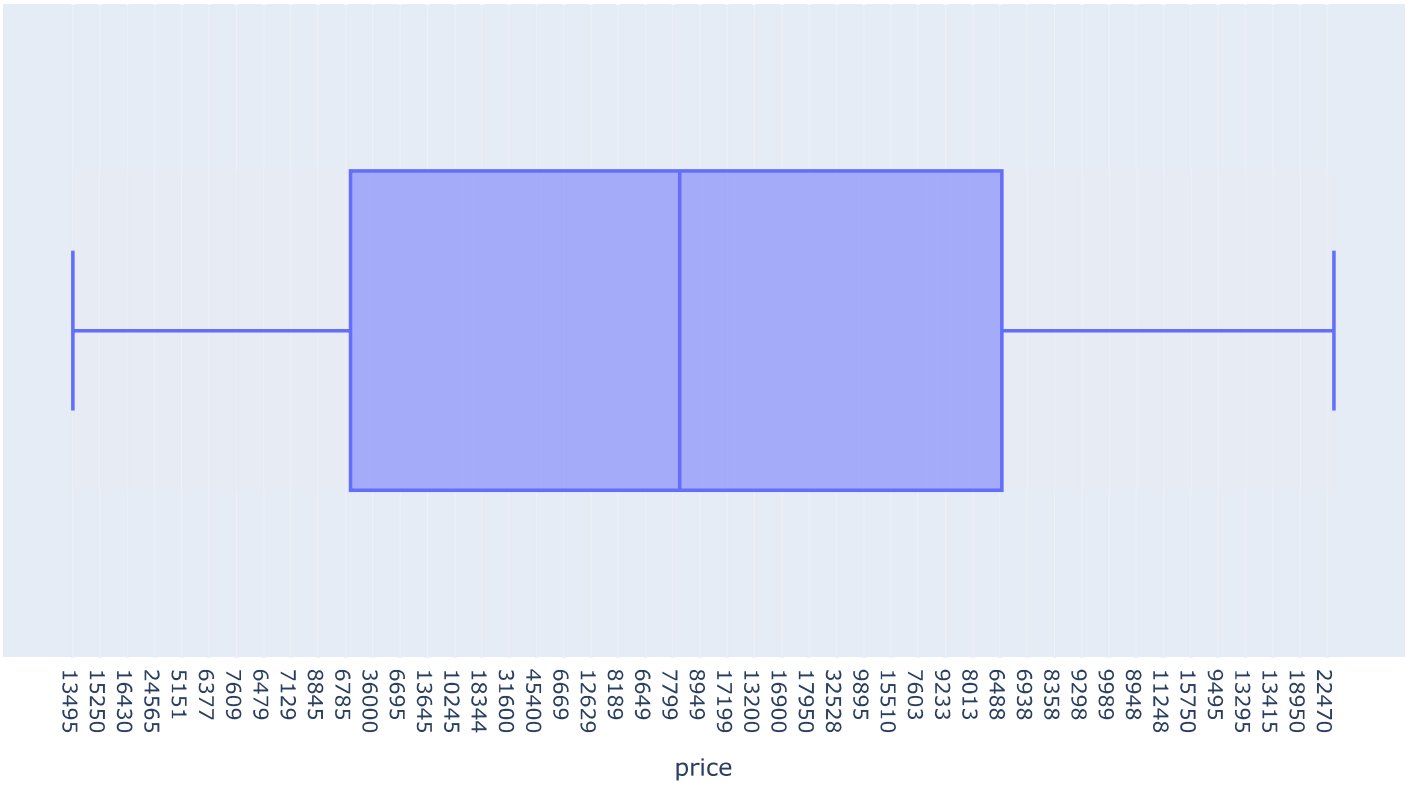
```
Min value in 'price' column : 5118.0
Max value in 'price' column : 45400.0
Mean value in 'price' column : 13207.13
```

In [45]:
```
# Locating NaN values in 'peak-rpm' column.
Auto_df.loc[pd.isna(Auto_df['price'])]
```

Out[45]:

	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
9	0	122.0	audi	gas	turbo	two	hatchback	4wd	front	99.5	178.2	67.9	52.0	3053	ohc	five	131	mpfi	3.13	3.4
44	1	122.0	isuzu	gas	std	two	sedan	fwd	front	94.5	155.9	63.6	52.0	1874	ohc	four	90	2bbl	3.03	3.11
45	0	122.0	isuzu	gas	std	four	sedan	fwd	front	94.5	155.9	63.6	52.0	1909	ohc	four	90	2bbl	3.03	3.11
129	1	122.0	porsche	gas	std	two	hatchback	rwd	front	98.4	175.7	72.3	50.5	3366	dohcv	eight	203	mpfi	3.94	3.11

In [46]:
```
fig = px.box(Auto_df, x="price")
fig.show()
```



In [47]:
```
# For the 'horsepower' value I will also use the mean value.
Auto_df['price'].fillna(round(Auto_df['price'].astype('float64').mean(), 0), inplace = True)
```

In [48]:
```
Auto_df.isna().sum()
```

Out[48]:
```
symboling           0
normalized-losses    0
make                0
fuel-type           0
aspiration          0
num-of-doors        0
body-style          0
drive-wheels        0
engine-location     0
wheel-base          0
length              0
width               0
height              0
curb-weight         0
engine-type         0
num-of-cylinders    0
engine-size         0
fuel-system         0
bore                0
stroke              0
compression-ratio   0
horsepower          0
peak-rpm            0
city-mpg            0
highway-mpg         0
price               0
dtype: int64
```

4. Name the columns that have datatype as “object”. (2 points)

In [49]:
```
Auto_df.dtypes
```

Out[49]:
```
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 [50]:
```
Auto_df.columns
```

Out[50]:
```
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')
```

Answer: 'normalized-losses', 'make', 'fuel-type', 'aspiration', 'num-of-doors', 'body-style', 'drive-wheels', 'engine-location','engine-type', 'num-of-cylinders','fuel-system', 'bore', 'stroke', 'horsepower', 'peak-rpm', 'price'

5. What is the mean and median value for the length and width column? (3 points)

In [51]:
```
Auto_df.describe()
```

Out[51]:

	symboling	wheel-base	length	width	height	curb-weight	engine-size	compression-ratio	city-mpg	highway-mpg
count	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000	205.000000
mean	0.834146	98.756585	174.049268	65.907805	53.724878	2555.565854	126.907317	10.142537	25.219512	30.751220
std	1.245307	6.021776	12.337289	2.145204	2.443522	520.680204	41.642693	3.972040	6.542142	6.886443
min	-2.000000	86.600000	141.100000	60.300000	47.800000	1488.000000	61.000000	7.000000	13.000000	16.000000
25%	0.000000	94.500000	166.300000	64.100000	52.000000	2145.000000	97.000000	8.600000	19.000000	25.000000
50%	1.000000	97.000000	173.200000	65.500000	54.100000	2414.000000	120.000000	9.000000	24.000000	30.000000
75%	2.000000	102.400000	183.100000	66.900000	55.500000	2935.000000	141.000000	9.400000	30.000000	34.000000
max	3.000000	120.900000	208.100000	72.300000	59.800000	4066.000000	326.000000	23.000000	49.000000	54.000000

Length mean and median:

In [52]:
```
print("Mean value in 'length' column :", round(Auto_df['length'].astype('float64').mean(), 2))
print("Median value in 'length' column :", round(Auto_df['length'].astype('float64').median(), 2))
```

```
Mean value in 'length' column : 174.05
Median value in 'length' column : 173.2
```

Width mean and median:

In [53]:
```
print("Mean value in 'width' column :", round(Auto_df['width'].astype('float64').mean(), 2))
print("Median value in 'width' column :", round(Auto_df['width'].astype('float64').median(), 2))
```

```
Mean value in 'width' column : 65.91
Median value in 'width' column : 65.5
```

6. The fuel consumption of the vehicles on the highway is given in miles per gallon (mpg) in the column “highway-mpg”. Transform the mpg to L/100km (using formula L/100km = 235/mpg) in the column of “highway-mpg” and change the name of column to “highway-L/100km”. (5 points)

In [54]:
```
Auto_df['highway-mpg']
```

Out[54]:
```
0      27
1      27
2      26
3      30
4      22
       ..
200    28
201    25
202    23
203    27
204    25
Name: highway-mpg, Length: 205, dtype: int64
```

In [55]:
```
Auto_df['highway-mpg'] = 235 / Auto_df['highway-mpg']
```

In [56]:
```
Auto_df['highway-mpg'] = round(Auto_df['highway-mpg'], 2)
```

In [57]:
```
Auto_df.rename(columns={'highway-mpg': 'highway-L/100km'}, inplace=True)
```

In [58]:
```
Auto_df.head()
```

Out[58]:

	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
0	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	3.47	2.68
1	3	122.0	alfa-romero	gas	std	two	convertible	rwd	front	88.6	168.8	64.1	48.8	2548	dohc	four	130	mpfi	3.47	2.68
2	1	122.0	alfa-romero	gas	std	two	hatchback	rwd	front	94.5	171.2	65.5	52.4	2823	ohcv	six	152	mpfi	2.68	3.47
3	2	164	audi	gas	std	four	sedan	fwd	front	99.8	176.6	66.2	54.3	2337	ohc	four	109	mpfi	3.19	3.4
4	2	164	audi	gas	std	four	sedan	4wd	front	99.4	176.6	66.4	54.3	2824	ohc	five	136	mpfi	3.19	3.4

7. Make a box plot for the curb-weight showing the 5-point summary. (2 points)

Before going further I will fix the numerical datatypes in the dataframe.

In [59]:
```
# The datatypes in this dataset seem to be wrong. Before going further I fix according to what kind of data we have in the dataset.
# Columns: 'bore', 'stroke' should be float64.
# Columns: 'normalized-losses', 'horsepower', 'peak-rpm', 'price' should be int64
Auto_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-L/100km      float64
price                 object
dtype: object
```

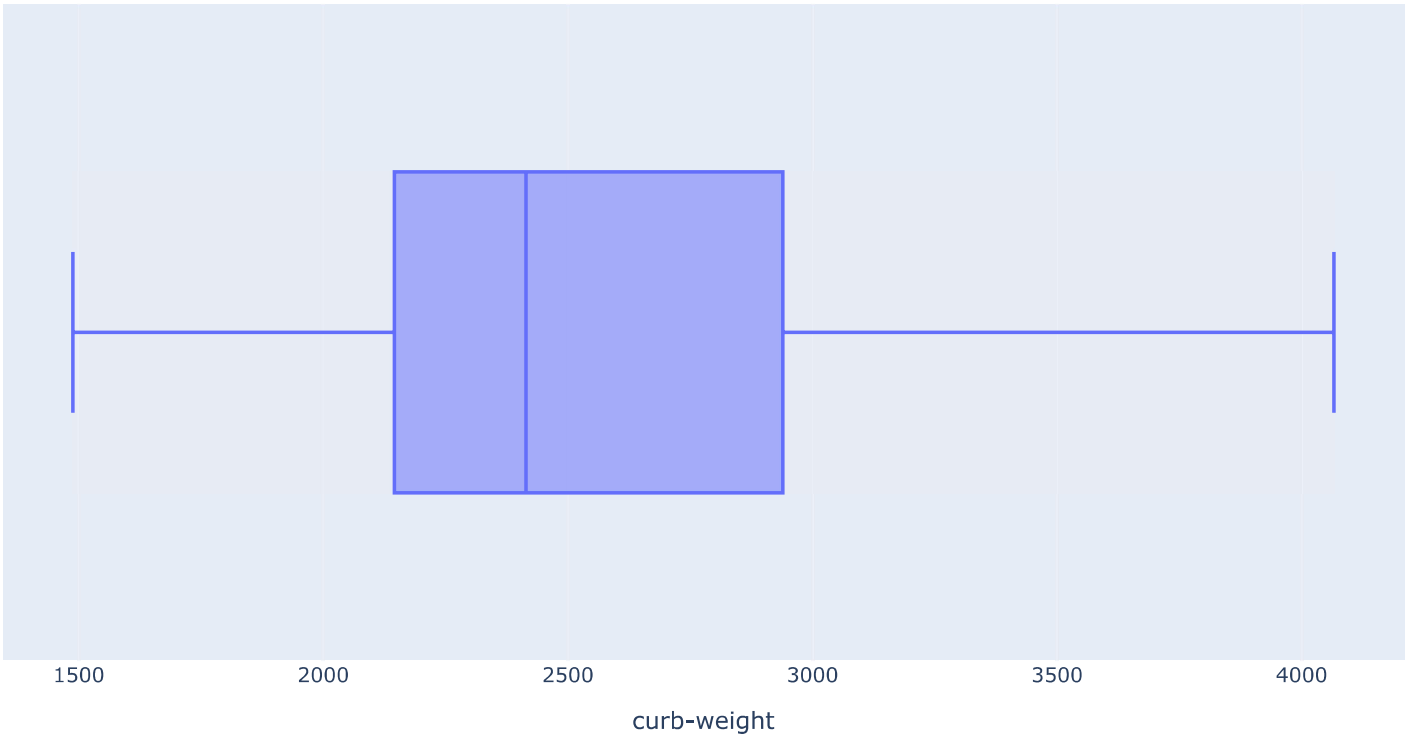
In [60]:
```
Auto_df['bore'] = Auto_df['bore'].astype('float64')
Auto_df['stroke'] = Auto_df['stroke'].astype('float64')

Auto_df['normalized-losses'] = Auto_df['normalized-losses'].astype('int64')
Auto_df['horsepower'] = Auto_df['horsepower'].astype('int64')
Auto_df['peak-rpm'] = Auto_df['peak-rpm'].astype('int64')
Auto_df['price'] = Auto_df['price'].astype('int64')
```

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

Out[61]:
```
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             int64
peak-rpm               int64
city-mpg               int64
highway-L/100km      float64
price                  int64
dtype: object
```

In [62]:
```
fig = px.box(Auto_df, x="curb-weight")
fig.show()
```



8. For the column “horsepower”, discretize the data into 5 bins and draw a histogram. (3 points)

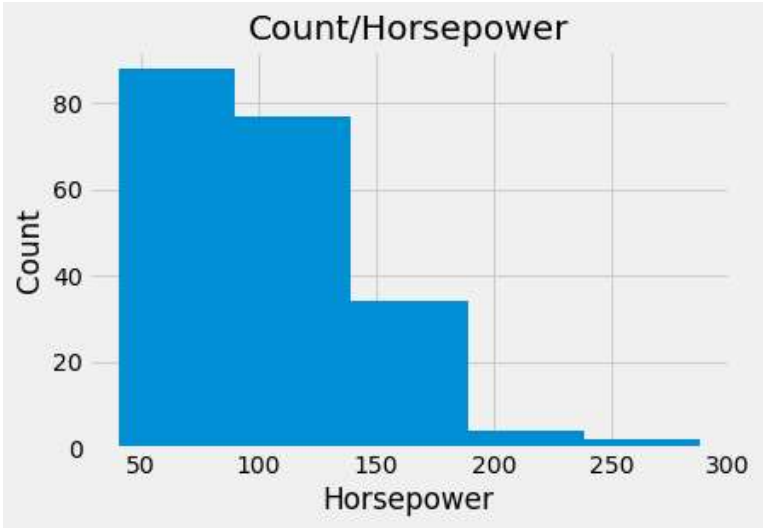
In [63]:
```
print("Min value in 'horsepower' column :", np.nanmin(Auto_df['horsepower']))
print("Max value in 'horsepower' column :", np.nanmax(Auto_df['horsepower']))
```

```
Min value in 'horsepower' column : 48
Max value in 'horsepower' column : 288
```

In [64]:
```
bins = np.linspace(40, 288, num = 6)
bins
```

Out[64]:
```
array([ 40. ,  89.6, 139.2, 188.8, 238.4, 288. ])
```

In [65]:
```
# Solution: Using plt.hist
plt.style.use("fivethirtyeight")
plt.hist(Auto_df['horsepower'], bins=bins, rwidth=1)
plt.ylabel("Count")
plt.xlabel("Horsepower")
plt.title("Count/Horsepower")
plt.show()
```



An alternative solution to this would be using plotly. However when passing nbins to plotly.histogram it doesn't want to split this data into 5 bins and defaults to 6. I couldn't figure out how to fix this but I will add the code under in comments.

In [66]:
```
# fig = px.histogram(Auto_df['horsepower'].astype('int64'), nbins=5)
# fig.show()
```

9. Find the Spearman correlation between the following columns: bore, stroke, compression-ratio and horsepower. (2 points)

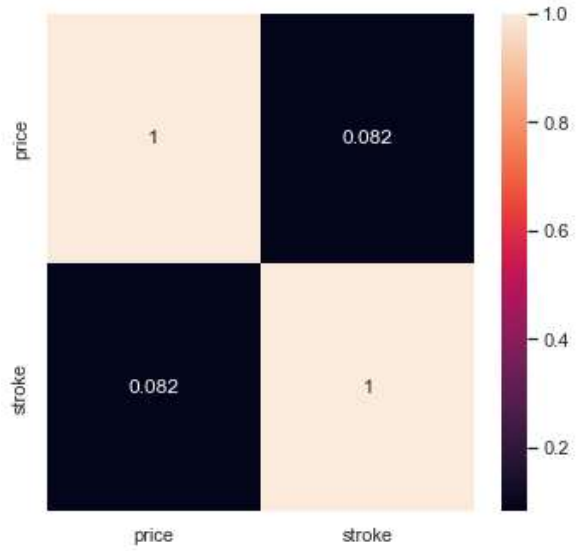
In [67]:
```
Auto_df[['bore', 'stroke', 'compression-ratio', 'horsepower']].corr(method = 'spearman')
```

Out[67]:

	bore	stroke	compression-ratio	horsepower
bore	1.000000	-0.081167	-0.162758	0.640786
stroke	-0.081167	1.000000	-0.067989	0.137894
compression-ratio	-0.162758	-0.067989	1.000000	-0.355908
horsepower	0.640786	0.137894	-0.355908	1.000000

10. Given the correlation results between "price" and "stroke", do you expect a linear relationship? Verify your results using the function "regplot()" (3 points)

In [68]:
```
# Plotting a correlation matrix for Auto_df
corr_matrix = Auto_df[['price', 'stroke']].corr()
sns.set(rc={'figure.figsize':(5, 5)})
sns.heatmap(corr_matrix, annot=True)
plt.show()
```



There is a small positive correlation og 0,082, which indicates a weak linear relationship between the two variables.

Correlation coefficient confirms a weak positive linear relationship between the variables.

In [69]:
```
np.corrcoef(Auto_df['price'], Auto_df['stroke'])
```

Out[69]:
```
array([[1.        , 0.08203437],
       [0.08203437, 1.        ]])
```

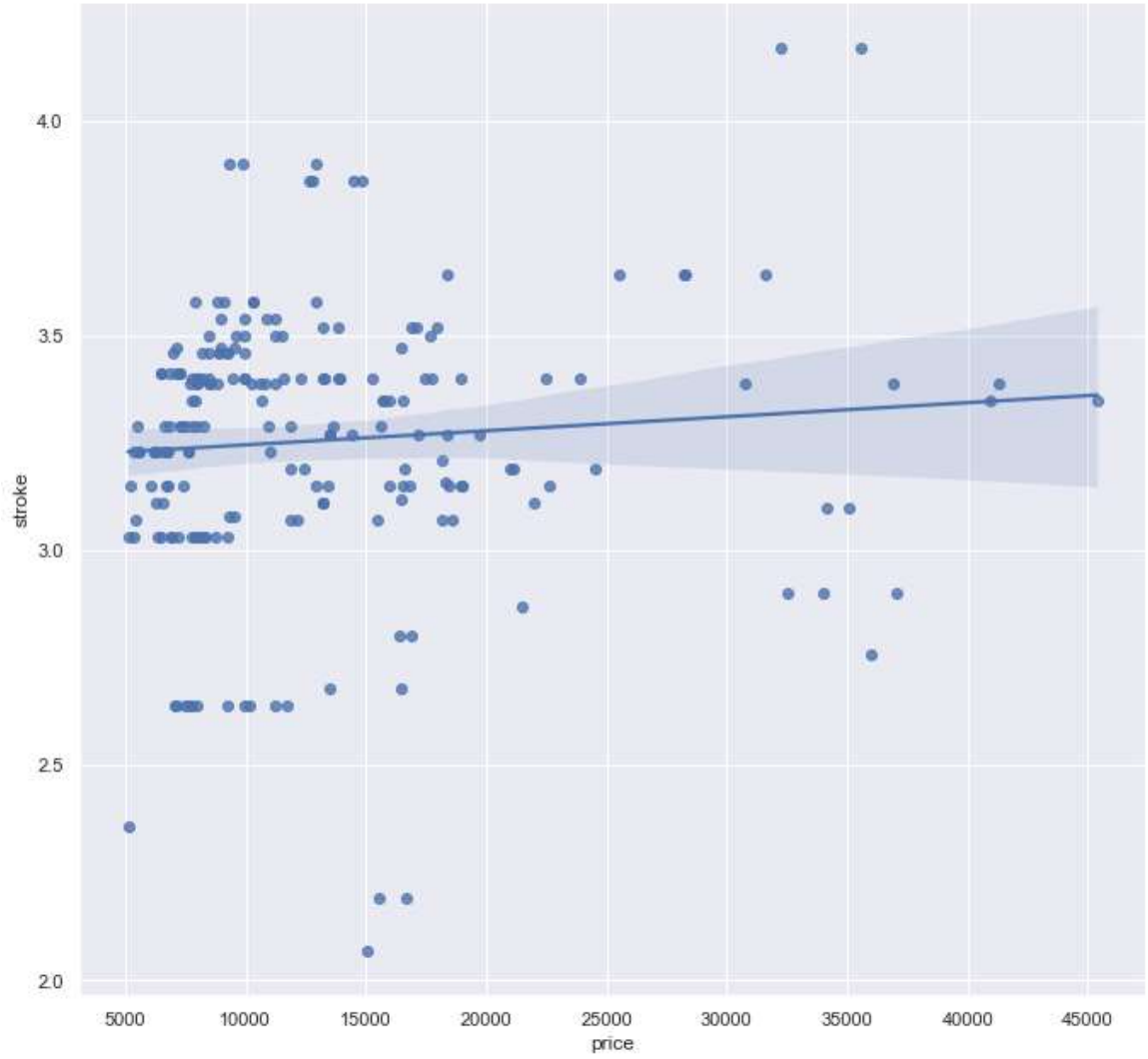
Using linear regression to confirm whether they have a relationship.

In [70]:
```
from scipy.stats import linregress
linregress(Auto_df['price'], Auto_df['stroke'])
```

Out[70]:
```
LinregressResult(slope=3.2697289856827473e-06, intercept=3.212913835564363, rvalue=0.0820343709439927, pvalue=0.24226532664592337, stderr=2.7880
56847000456e-06, intercept_stderr=0.04283490508601776)
```

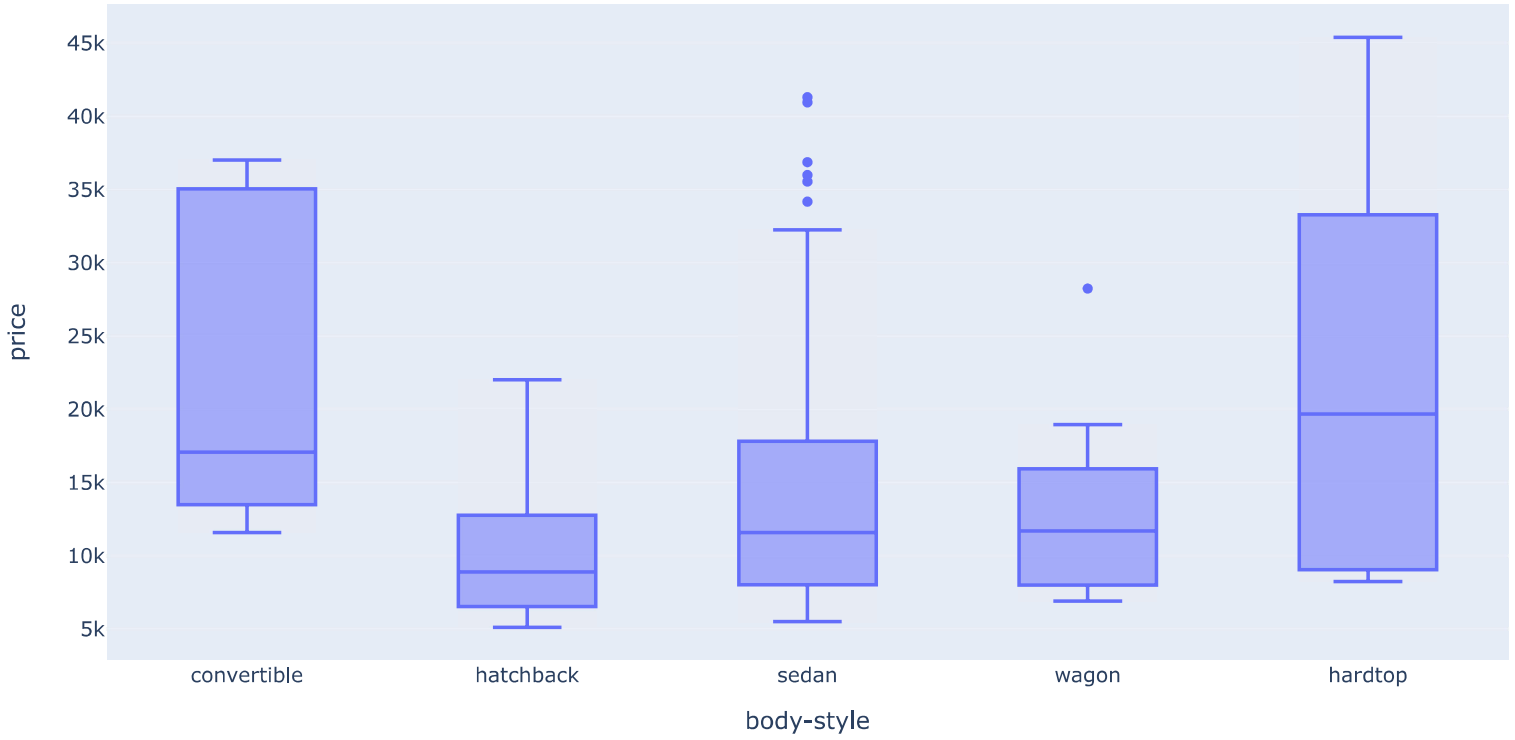
The pvalue, 0.24, is above 0,05 indicating that the relationship between price and stroke is indeed significant.

In [71]:
```
sns.set(rc={'figure.figsize':(10, 10)})
sns.regplot(data = Auto_df[['price', 'stroke']], x = 'price', y = 'stroke')
plt.show()
```



11. Make a box plot between “body-style” and “price”. Which body-style has the maximum outliers in price? (5 points)

In [72]:
```
fig = px.box(Auto_df, x = 'body-style', y = 'price', points = 'outliers')
fig.show()
```



Answer: The sedan body-style has the most outliers in price.

12. Into how many categories can vehicles be categorized based on drive-wheels? How many vehicles are four-wheel drive (i.e. 4wd)? (3 points)

In [73]:
```
Auto_df['drive-wheels'].nunique()
```

Out[73]:
```
3
```

There are 3 categories in 'drive-wheels'.

In [74]:
```
Auto_df['drive-wheels'].value_counts()
```

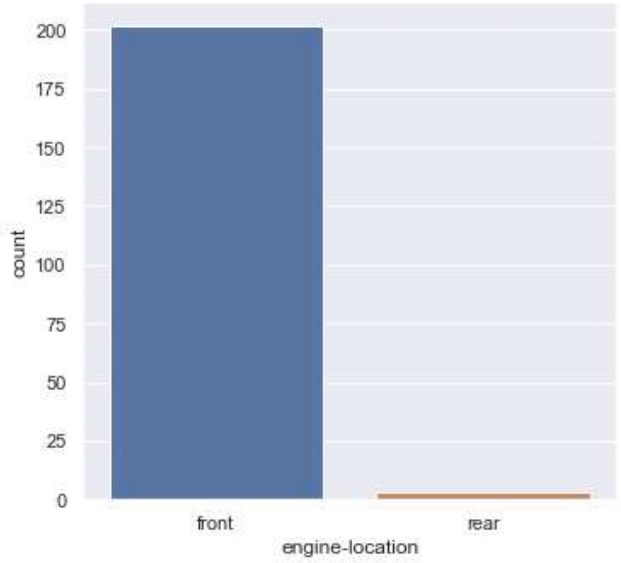
Out[74]:
```
fwd    120
rwd     76
4wd      9
Name: drive-wheels, dtype: int64
```

Answer: 9 vehicles have for-wheel drive.

13. How many vehicles have rear engine location? (2 points)

In [75]:
```
sns.set(rc={'figure.figsize':(5, 5)})
sns.countplot(x=Auto_df['engine-location'])
```

Out[75]:
```
<AxesSubplot:xlabel='engine-location', ylabel='count'>
```



In [76]:
```
Auto_df['engine-location'].value_counts()
```

Out[76]:
```
front    202
rear       3
Name: engine-location, dtype: int64
```

Answer: 3 vehicles have a engine in the rear-location

14. What is the average price of the forward wheel drive (fwd) and rear wheel drive (rwd) category of vehicles? (5 points)

In [77]:
```
Auto_df[['price', 'drive-wheels']].groupby('drive-wheels').mean()
```

Out[77]:

drive-wheels	price
4wd	10570.555556
fwd	9310.816667
rwd	19671.421053

Answer:

fwd 9310.816667

rwd 19671.421053

15. What is the average price of rwd-sedan and 4wd-hatchback? Which one is larger? (5 points)

In [78]:
```
Auto_df[['price', 'drive-wheels', 'body-style']].groupby(['body-style', 'drive-wheels']).mean().round(2)
```

Out[78]:

body-style	drive-wheels	price
convertible	fwd	11595.00
	rwd	23949.60
hardtop	fwd	8249.00
	rwd	24202.71
hatchback	4wd	10405.00
	fwd	8396.39
	rwd	14278.26
sedan	4wd	12647.33
	fwd	9930.93
	rwd	21711.83
wagon	4wd	9095.75
	fwd	9997.33
	rwd	16994.22

Answer:

Average price for rwd-sedan is 21711.83

Average price for 4wd-hatchback is 10405.00

Problem 4: Web Scraping Problem (20 points)

In this task you will webscrap the following website https://en.wikipedia.org/wiki/World_population (https://en.wikipedia.org/wiki/World_population). Perform the following tasks:

1. Scrap the table titled: “10 most populous countries” shown below and create a DataFrame named DF_Pop (15 points).

In [79]:
```
!pip install bs4
!pip install lxml
!pip install html5lib
!pip install requests
```

```
Requirement already satisfied: bs4 in c:\users\bjark\anaconda3\lib\site-packages (0.0.1)
Requirement already satisfied: beautifulsoup4 in c:\users\bjark\anaconda3\lib\site-packages (from bs4) (4.11.1)
Requirement already satisfied: soupsieve>1.2 in c:\users\bjark\anaconda3\lib\site-packages (from beautifulsoup4->bs4) (2.3.1)
Requirement already satisfied: lxml in c:\users\bjark\anaconda3\lib\site-packages (4.8.0)
Requirement already satisfied: html5lib in c:\users\bjark\anaconda3\lib\site-packages (1.1)
Requirement already satisfied: six>=1.9 in c:\users\bjark\anaconda3\lib\site-packages (from html5lib) (1.16.0)
Requirement already satisfied: webencodings in c:\users\bjark\anaconda3\lib\site-packages (from html5lib) (0.5.1)
Requirement already satisfied: requests in c:\users\bjark\anaconda3\lib\site-packages (2.27.1)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\bjark\anaconda3\lib\site-packages (from requests) (2021.10.8)
Requirement already satisfied: idna<4,>=2.5 in c:\users\bjark\anaconda3\lib\site-packages (from requests) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in c:\users\bjark\anaconda3\lib\site-packages (from requests) (2.0.4)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in c:\users\bjark\anaconda3\lib\site-packages (from requests) (1.26.9)
```

In [80]:
```
from bs4 import BeautifulSoup # this module helps in web scrapping.
import requests  # this module helps us to download a web page
```

In [81]:
```
# Url of the website with the table we need to scrape
url = "https://en.wikipedia.org/wiki/World_population"
```

In [82]:
```
# get the contents of the webpage in text format and store in a variable called data
data  = requests.get(url).text
```

In [83]:
```
soup = BeautifulSoup(data,"html.parser")
```

In [84]:
```
# find all html tables in the web page
tables = soup.find_all('table')
tables # in html table is represented by the tag <table>
```

```
 <td>867
 </td></tr>
 <tr>
 <td>4</td>
 <td align="left"><span class="flagicon"><img alt="" class="thumbborder" data-file-height="600" data-file-width="900" decoding="async" height
="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/5/59/Flag_of_Lebanon.svg/23px-Flag_of_Lebanon.svg.png" srcset="//upload.wikimedia.o
rg/wikipedia/commons/thumb/5/59/Flag_of_Lebanon.svg/35px-Flag_of_Lebanon.svg.png 1.5x, //upload.wikimedia.org/wikipedia/commons/thumb/5/59/Fl
ag_of_Lebanon.svg/45px-Flag_of_Lebanon.svg.png 2x" width="23"/> </span><a href="/wiki/Lebanon" title="Lebanon">Lebanon</a></td>
 <td>5,296,814</td>
 <td>10,400</td>
 <td>509
 </td></tr>
 <tr>
 <td>5</td>
 <td align="left"><span class="flagicon"><img alt="" class="thumbborder" data-file-height="600" data-file-width="900" decoding="async" height
="15" src="//upload.wikimedia.org/wikipedia/commons/thumb/7/72/Flag_of_the_Republic_of_China.svg/23px-Flag_of_the_Republic_of_China.svg.png"
srcset="//upload.wikimedia.org/wikipedia/commons/thumb/7/72/Flag_of_the_Republic_of_China.svg/35px-Flag_of_the_Republic_of_China.svg.png 1.5
x, //upload.wikimedia.org/wikipedia/commons/thumb/7/72/Flag_of_the_Republic_of_China.svg/45px-Flag_of_the_Republic_of_China.svg.png 2x" width
="23"/> </span><a href="/wiki/Taiwan" title="Taiwan">Taiwan</a></td>
 <td>23,580,712</td>
```

In [85]:
```
# Finding the index of the table I want to scrape. The table doesn't have a caption connected to it
# so I search for a column name. Could also search for dependency as it seems like it's the only table with that column name
for index, table in enumerate(tables):
    if ("Percentage <br/> of the world" in str(table)):
        table_index = index
print(table_index)
```

```
3
```

In [86]:
```
print(tables[table_index].prettify())
```

```
 Country
 </a>
 /
 <a href="/wiki/Dependent_territory" title="Dependent territory">
  Dependency
 </a>
</th>
<th scope="col">
 Population
</th>
<th scope="col">
 Percentage
 <br/>
 of the world
</th>
<th scope="col">
 Date
</th>
<th scope="col">
 <span class="nowrap">
```

In [87]:
```
population_data = pd.DataFrame(columns=[ "Country / Dependency", "Population", "Percentage of the world", "Date", "Source"])

for row in tables[table_index].tbody.find_all("tr"):
    col = row.find_all("td")
    if (col != []):
        country = col[0].text
        population = col[1].text.strip()
        percentage = col[2].text.strip()
        date = col[3].text.strip()
        source = col[4].text.strip()
        population_data = population_data.append({"Country / Dependency":country, "Population":population, "Percentage of the world":percentage, "

population_data['Rank'] = population_data.index +1
population_data  = population_data[['Rank','Country / Dependency', 'Population', 'Percentage of the world', 'Date',
       'Source']]
population_data
```

```
C:\Users\bjark\AppData\Local\Temp\ipykernel_1892\1808548746.py:11: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

C:\Users\bjark\AppData\Local\Temp\ipykernel_1892\1808548746.py:11: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

C:\Users\bjark\AppData\Local\Temp\ipykernel_1892\1808548746.py:11: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

C:\Users\bjark\AppData\Local\Temp\ipykernel_1892\1808548746.py:11: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

C:\Users\bjark\AppData\Local\Temp\ipykernel_1892\1808548746.py:11: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

C:\Users\bjark\AppData\Local\Temp\ipykernel_1892\1808548746.py:11: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

C:\Users\bjark\AppData\Local\Temp\ipykernel_1892\1808548746.py:11: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

C:\Users\bjark\AppData\Local\Temp\ipykernel_1892\1808548746.py:11: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

C:\Users\bjark\AppData\Local\Temp\ipykernel_1892\1808548746.py:11: FutureWarning:
```

```
The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.

C:\Users\bjark\AppData\Local\Temp\ipykernel_1892\1808548746.py:11: FutureWarning:

The frame.append method is deprecated and will be removed from pandas in a future version. Use pandas.concat instead.
```

Out[87]:

	Rank	Country / Dependency	Population	Percentage of the world	Date	Source
0	1	China	1,412,600,000	17.7%	31 Dec 2021	National annual estimate[93]
1	2	India	1,373,761,000	17.2%	1 Mar 2022	Annual national estimate[94]
2	3	United States	333,408,962	4.17%	6 Dec 2022	National population clock[95]
3	4	Indonesia	275,773,800	3.45%	1 Jul 2022	National annual estimate[96]
4	5	Pakistan	229,488,994	2.87%	1 Jul 2022	UN projection[97]
5	6	Nigeria	216,746,934	2.71%	1 Jul 2022	UN projection[97]
6	7	Brazil	215,482,336	2.69%	6 Dec 2022	National population clock[98]
7	8	Bangladesh	168,220,000	2.10%	1 Jul 2020	Annual Population Estimate[99]
8	9	Russia	147,190,000	1.84%	1 Oct 2021	2021 preliminary census results[100]
9	10	Mexico	128,271,248	1.60%	31 Mar 2022	National quarterly estimate[101]

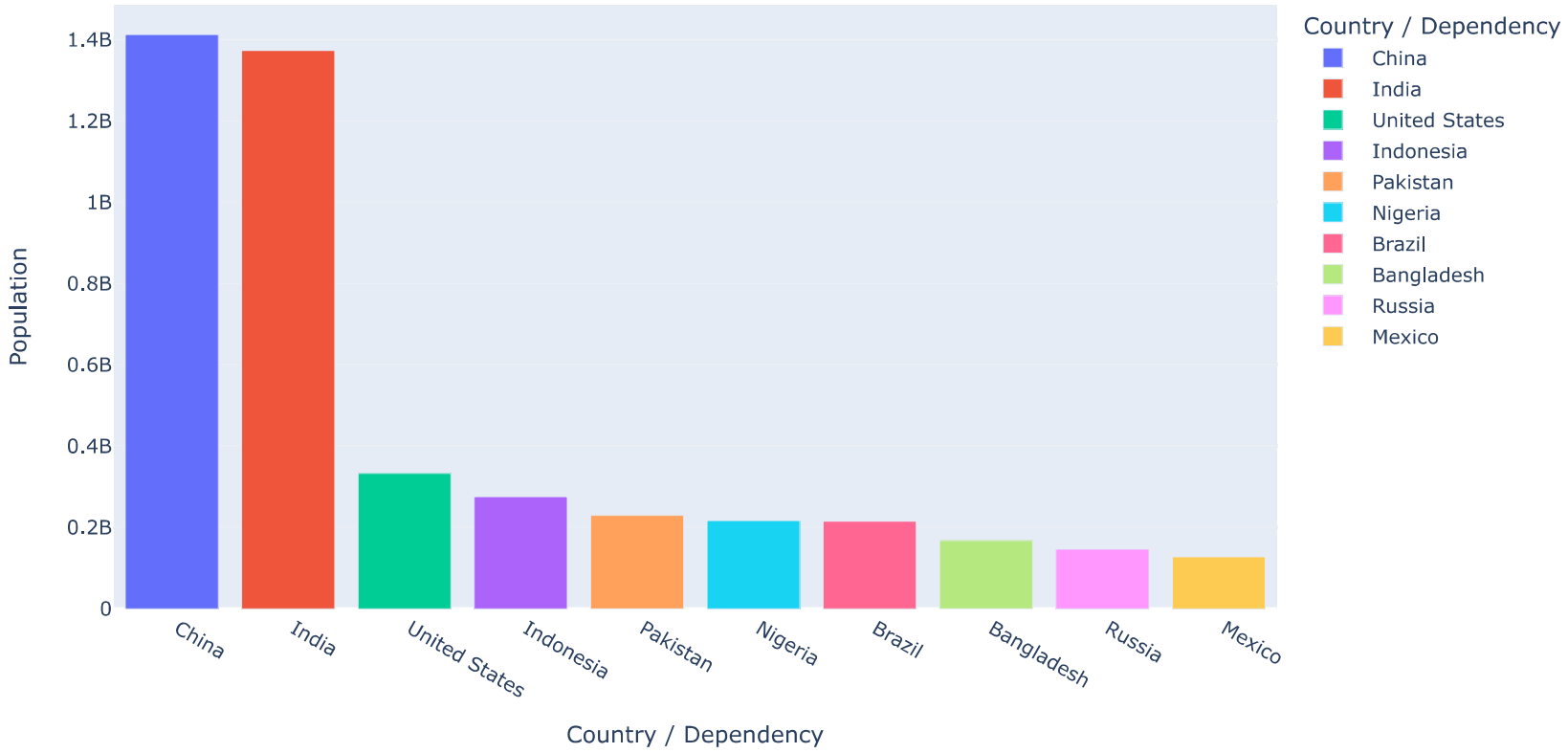
2. Create a Bar Chart for the DF_Pop, depicting the country name on X-axis and population on the y-axis. (5 points)

In [88]:
```
# Need to fix the population datatype
population_data.dtypes
```

Out[88]:
```
Rank                        int64
Country / Dependency       object
Population                 object
Percentage of the world    object
Date                       object
Source                     object
dtype: object
```

In [89]:
```
# Using str.replace to remove ',' and converting the column to int64
population_data['Population'] = population_data['Population'].str.replace(',', '').astype('int64')
```

In [90]:
```
fig = px.bar(population_data, x = 'Country / Dependency', y = 'Population', color='Country / Dependency')
fig.show()
```



References

Correlation. tutor2u. (2021, March 22). Retrieved December 5, 2022, from https://www.tutor2u.net/business/reference/correlation (https://www.tutor2u.net/business/reference/correlation)

Dobbins, T., & Burke, J. (2021). Essential Statistics for Data Science: A case study using python, part I. Retrieved December 5, 2022, from https://www.learndatasci.com/tutorials/data-science-statistics-using-python/ (https://www.learndatasci.com/tutorials/data-science-statistics-using-python/)

Geron. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow. O'Reilly Media, Incorporated.

Gupta, S. (n.d.). Exploratory Data Analysis (univariate, bivariate, and Multivariate Analysis). enjoyalgorithms. Retrieved December 5, 2022, from https://www.enjoyalgorithms.com/blog/univariate-bivariate-multivariate-analysis (https://www.enjoyalgorithms.com/blog/univariate-bivariate-multivariate-analysis)

Box Plots in Python. Box plots in Python. (n.d.). Retrieved December 4, 2022, from https://plotly.com/python/box-plots/ (https://plotly.com/python/box-plots/)

Scipy.stats.linregress. SciPy v1.9.3 Manual. (n.d.). Retrieved December 5, 2022, from https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html (https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.linregress.html)

Keprate, A.(2022). WebScraping_Lecture_11 slides from class.

Beautiful Soup documentation. Beautiful Soup 4.9.0 documentation. (n.d.). Retrieved December 6, 2022, from https://www.crummy.com/software/BeautifulSoup/bs4/doc/ (https://www.crummy.com/software/BeautifulSoup/bs4/doc/)

McKinney. (2018). Python for data analysis : data wrangling with pandas, NumPy, and IPython (Second edition.). O'Reilly.

Mandatory assignment I and II. (2022). Answers from my own mandatory assignment.

In [ ]: