

STUDENT ASSESSMENT SUBMISSION AND DECLARATION**ASSIGNMENT FINAL REPORT**

When submitting evidence for assessment, each student must sign a declaration confirming that the work is their own.

Student name: Hoang Anh Quy		Student ID: BS00311	
Class: DA06201		Assessor name: Nguyen Van Vu	
Issue date: 14/08/2024	Submission date: 14/08/2024	Submitted on: CMS	
	Re-submission date:	Re-submitted on:	
Programme: BTEC Level 5 HND Diploma in Computing			
Unit number, title and code: Unit 24: Advanced Programming for Data Analysis (H/618/5723)			
Assignment number and title: Final Report			

Grading Grid

P1	P2	P3	P4	P5	P6	P7	M1	M2	M3	M4	D1	D2

Plagiarism

Plagiarism is a particular form of cheating. Plagiarism must be avoided at all costs and students who break the rules, however innocently, may be penalised. It is your responsibility to ensure that you

understand correct referencing practices. As a university level student, you are expected to use appropriate references throughout and keep carefully detailed notes of all your sources of materials for material you have used in your work, including any material downloaded from the Internet. Please consult the relevant unit lecturer or your course tutor if you need any further advice.

Student Declaration

Student declaration

I certify that the assignment submission is entirely my own work and I fully understand the consequences of plagiarism. I declare that the work submitted for assessment has been carried out without assistance other than that which is acceptable according to the rules of the specification. I certify I have clearly referenced any sources and any artificial intelligence (AI) tools used in the work. I understand that making a false declaration is a form of malpractice.

Student signature: Hoang Anh Quy

Date: 14/08/2024

Higher Nationals – Summative Assignment Feedback Form

Student Name/ID	Hoang Anh Quy BS00311		
Unit Title	Unit 24: Advanced Programming for Data Analysis (H/618/5723)		
Assignment Number	Final report	Assessor	Nguyen Van Vu
Submission Date	14/08/2024	Date Received 1st submission	14/08/2024
Re-submission Date		Date Received 2nd submission	
Assessor Feedback: *Please note that constructive and useful feedback should allow students to understand: a) Strengths of performance b) Limitations of performance c) Any improvements needed in future assessments Feedback should be against the learning outcomes and assessment criteria to help students understand how these inform the process of judging the overall grade. Feedback should give full guidance to the students on how they have met the learning outcomes and assessment criteria.			
<i>I certify that to the best of my knowledge the evidence submitted for this assignment is the student's own. The student has clearly referenced any sources and any artificial intelligence (AI) tools used in the work. I have not solely used AI to grade the student's work.</i>			
Grade:	Assessor Signature:		Date:
Resubmission Feedback: *Please note resubmission feedback is focussed only on the resubmitted work			
Grade:	Assessor Signature:		Date:
Internal Verifier's Comments:			

Signature & Date:

* Please note that grade decisions are provisional. They are only confirmed once internal and external moderation has taken place and grades decisions have been agreed at the assessment board.

Contents

Introduction. 11

LO1 Explore the tools a programmer can use to manipulate large data sets for data analysis. 11

1 Investigate the functions of a data analysis language. P1 11

 1.1 Study Python language and choose some libraries for this assignment. 11

 1.2 Install Python through Anaconda. 12

 1.3 Understand the basic syntax of Python. 16

 1.3.1 Open Jupyter Notebook. 16

 1.3.2 Learn basic Python syntax. 17

 1.4 Choose a popular library for data analysis. 22

 1.4.1 Install libraries. 22

 1.4.2 Introduction to Libraries. 23

2 Prepare a proposal for analyzing a large dataset, including the method of analysis and the outcomes to be achieved. P2 25

 2.1 Identify the data source. 25

 2.2 Use Python to read data from CSV formats. 25

 2.3 Data processing and cleaning. 27

3 Design functions and choose charts to visualize data and results. P3 32

 3.1 Identify functions needed for data analysis. 32

 3.2 Write functions in Python to perform analytical tasks. 32

 3.3 Use data visualization libraries such as Matplotlib or Seaborn to create charts to visualize data and results. 34

4 Create a detailed test plan and identify the expected outcomes of the analysis. P4 40

 4.1 Definition of test case. 40

 4.2 Mission of Test Case. 40

 4.3 The Necessity of Test cases. 40

4.4	Create a detailed test plan and identify the expected outcomes of the analysis.	40
LO3	Develop a software tool to analyze a large data set for a given scenario.	41
5	Build a software tool for analyzing a large dataset according to a developed design. P5	41
5.1	Introducing Streamlit.	42
5.2	Write a web application in Python with the Streamlit library.	42
5.2.1	Identify the data source.	42
5.2.2	Read and process data.	43
5.2.3	Analyze and visualize data.	50
6	Implement a detailed test plan on a data analysis software tool. P6	59
6.1	Prepare a test plan (Test Plan)	59
6.1.1	Introduction to Test Plan.	59
6.1.2	Test Plan for Car Price Data Analysis Project	60
6.2	Write detailed test cases.....	62
7	Present the results of the analysis on the chosen data set. P7	64
7.1	TC01.....	65
7.2	TC02.....	66
7.3	TC03.....	67
7.4	TC04.....	68
7.5	TC05.....	69
7.6	TC06.....	70
7.7	TC07.....	71
7.8	TC08.....	72
7.9	Conduct the final round of testing for the entire application.	75
	Conclusion.....	75
	References.....	77

List of Figures:

Figure 1 Python language.....	11
Figure 2 Install Python through Anaconda.	12
Figure 3 The steps to install Python through Anaconda.....	12
Figure 4 Download Anaconda.....	13
Figure 5 Install Anaconda 1.....	13
Figure 6 Install Anaconda 2.....	13
Figure 7 Install Anaconda 3.....	14
Figure 8 Install Anaconda 4.....	14
Figure 9 Install Anaconda 5.....	14
Figure 10 Install Anaconda 6.....	15
Figure 11 Install Anaconda 7.....	15
Figure 12 Install Anaconda 8.....	15
Figure 13 Check the Anaconda version and Python version.	16
Figure 14 Start Anaconda Navigator.....	16
Figure 15 Anaconda Navigator.....	16
Figure 16 Open Jupyter Notebook 1.....	17
Figure 17 Open Jupyter Notebook 2.....	17
Figure 18 Example of variable declaration syntax.	18
Figure 19 Example of "for" loop 1.....	19
Figure 20 Example of "for" loop 2.....	19
Figure 21Example of "while" loop.....	20
Figure 22 The syntax for defining a function.	20
Figure 23 Example of Function.....	21
Figure 24 The basic syntax of conditional functions.....	21
Figure 25 Example of Conditional function.....	22
Figure 26 Use Anaconda Navigator to install the necessary libraries.....	23
Figure 27 Install Pandas, NumPy, Matplotlib, and Seaborn libraries via command line.....	23
Figure 28 Declare required libraries.	25
Figure 29 The total number of rows, and columns.....	26
Figure 30 Summary of dataframe information.	26

Figure 31 Display the top 5 rows of the dataframe.	27
Figure 32 Display the last 5 rows of the dataframe.	27
Figure 33 The count and show duplicate rows.	28
Figure 34 Remove duplicate data.	28
Figure 35 Check and count the number of missing values.	28
Figure 36 Delete the vin column.	29
Figure 37 Filling missing values for the transmission column.	29
Figure 38 Remove the missing values.	30
Figure 39 Convert the year column and the saledate column to datetime.	30
Figure 40 Creating a function that draws a scatter plot.	31
Figure 41 Scatter Plot of sellingprice over year.	31
Figure 42 Scatter Plot of condition over the year.	32
Figure 43 Delete outliers.	32
Figure 44 The function calculates basic statistics.	33
Figure 45 Create the function "plot_pie_chart".	33
Figure 46 Create the function " plot_bar_chart".	33
Figure 47 Create the function "group_and_count".	34
Figure 48 Percentage of car sales of automakers in the world.	34
Figure 49 The ratio of car models sold in the Ford brand.	35
Figure 50 The correlation between the MMR and Sellingprice.	36
Figure 51 The correlation between selling price and condition.	37
Figure 52 The trend in selling prices over the years.	38
Figure 53 A new dataframe with year and salequantity columns.	39
Figure 54 The number of cars sold over the years.	39
Figure 55 Streamlit.	42
Figure 56 Declare required libraries.	43
Figure 57 Read and show data.	43
Figure 58 Command to display the data on Streamlit.	43
Figure 59 Display the dataframe.	44
Figure 60 The count and show duplicate rows.	44
Figure 61 Handling missing values.	45
Figure 62 Check and count the number of missing values.	45

Figure 63 Fill missing values in the 'transmission' column with the mode value and drop the remaining rows with missing values.	46
Figure 64 Remove the missing values.	46
Figure 65 Convert the year column and the saledate column to datetime.	47
Figure 66 Visualize the data as a scatter plot.	47
Figure 67 Scatter Plot of sellingprice over year.	47
Figure 68 Scatter Plot of condition over the year.	48
Figure 69 Apply the outlier filters.	48
Figure 70 Data after filtering outliers.	49
Figure 71 Function to calculate basic statistics.	50
Figure 72 Function to create a pie chart.	50
Figure 73 Create a function to group columns and calculate counts.	51
Figure 74 The car sales rate of car manufacturers in the world.	51
Figure 75 Percentage of car sales of automakers in the world.	52
Figure 76 Filter DataFrame by car brand and Ratio of car models within the car brand	53
Figure 77 The best-selling car type in the Ford car brand.	53
Figure 78 The correlation between the MMR and Sellingprice code.	54
Figure 79 The correlation between the MMR and Sellingprice.	55
Figure 80 the correlation between selling price and condition.	56
Figure 81 The trend in selling prices over the years	57
Figure 82 a salequantity dataframe with year and salequantity columns	58
Figure 83 The number of cars sold over the years.	58
Figure 84 Run test TC01.	65
Figure 85 The test result TC01.	65
Figure 86 Run test TC02.	66
Figure 87 The test result TC02.	66
Figure 88 Run test TC03.	67
Figure 89 The test result TC03.	67
Figure 90 Run test TC04.	68
Figure 91 The test result TC04.	68
Figure 92 Run test TC05.	69
Figure 93 The test result TC05	69

Figure 94 Run test TC06. 70

Figure 95 The test result TC06. 70

Figure 96 Run test TC07. 71

Figure 97 The test result TC07 71

Figure 98 Run test TC08. 72

Figure 99 The test result TC08. 73

Figure 100 The test result TC08. 73

Figure 101The test result TC08. 74

Figure 102 Run a test for the entire application..... 75

Figure 103 Run a test for the entire application..... 75

List of Tables:

Table 1 Check out the scatter plot function. 40

Table 2 Check out the calculate_statistics function. 41

Table 3Test Case..... 62

Introduction.

Data analytics is a broad and dynamic field that plays a crucial role in modern decision-making processes. It involves using various tools and techniques to extract, process, and analyze large datasets to uncover patterns, trends, and insights that can significantly inform strategic decisions across different domains. In today's data-driven world, the ability to analyze and interpret data is essential for organizations across industries, including marketing, finance, healthcare, manufacturing, and human resources. This assignment explores the application of data analytics in real-life scenarios, highlighting how companies can leverage these insights to drive strategic decisions, optimize operations, and enhance overall performance.

LO1 Explore the tools a programmer can use to manipulate large data sets for data analysis.

1 Investigate the functions of a data analysis language. P1

1.1 Study Python language and choose some libraries for this assignment.

Python is a high-level computer programming language created by Guido van Rossum in 1989. It has many applications, from building websites and software to automating tasks and analysing data. Python is designed with the strong advantage of being easy to read, easy to learn, and easy to remember. This has made it one of the most popular programming languages today. (Campbell, 2024) (Glints, 2022) (McKinney, 2022)

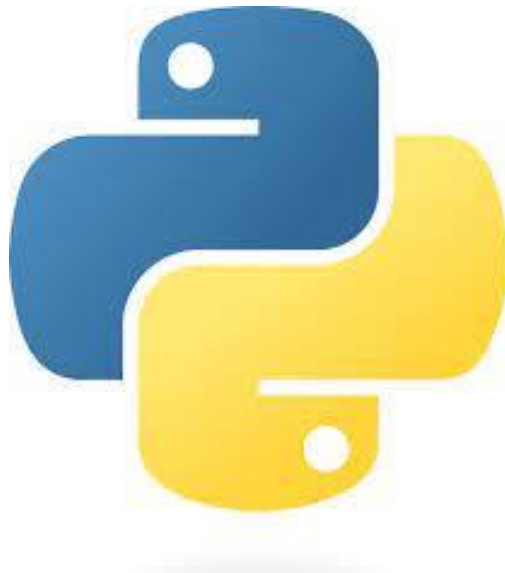


Figure 1 Python language.

1.2 Install Python through Anaconda.



Figure 2 Install Python through Anaconda.

Anaconda is an open-source distribution of Python and R, designed to simplify the management and deployment of packages and environments. Widely used in the data science and data analytics communities. Anaconda uses the Conda package management tool, allowing users to easily install, update, and manage software packages as well as create private virtual environments separately for different projects. This helps avoid conflicts between versions of software packages. (Galarnyk, 2020)

Here are the steps to install Python through Anaconda:



Figure 3 The steps to install Python through Anaconda.

1. Download Anaconda

Go to the Anaconda website (<https://www.anaconda.com/products/distribution>) select the Python 3 graphical installer and download the version of Anaconda appropriate for your operating system (Windows, macOS, or Linux).

Click the "Download" button to download the installation file.

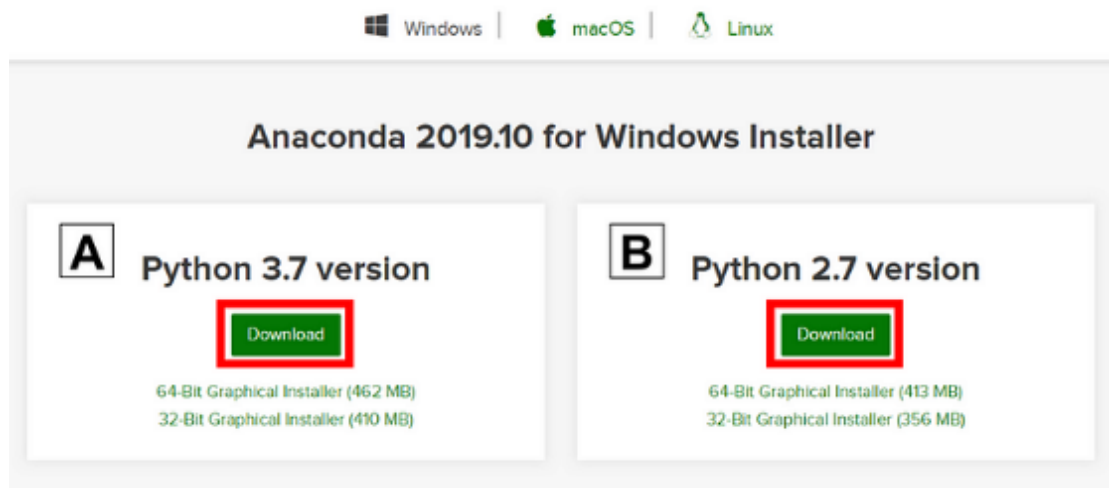


Figure 4 Download Anaconda.

2. Install Anaconda

Run the installation file you just downloaded.

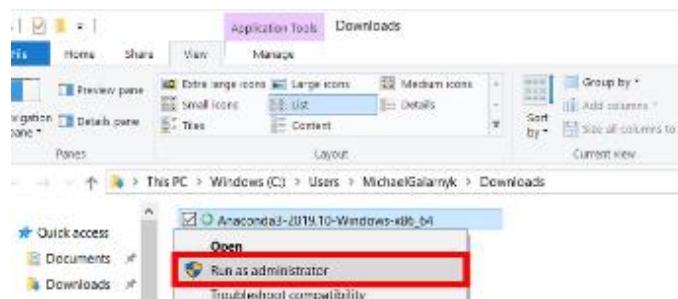


Figure 5 Install Anaconda 1.

When the installation dialog box appears, click "Next" to begin the installation process.

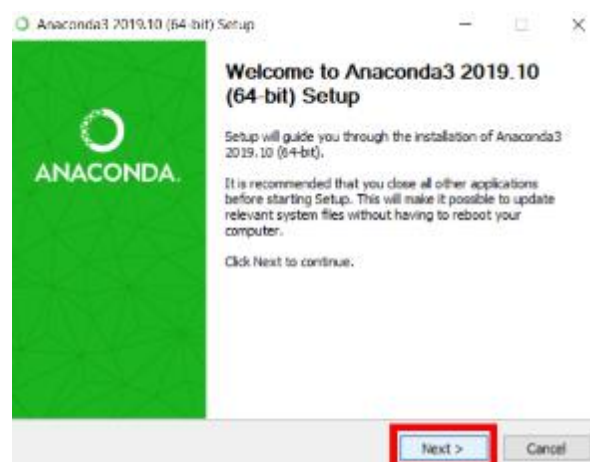


Figure 6 Install Anaconda 2.

Select the "I Agree" option to accept the terms of use.

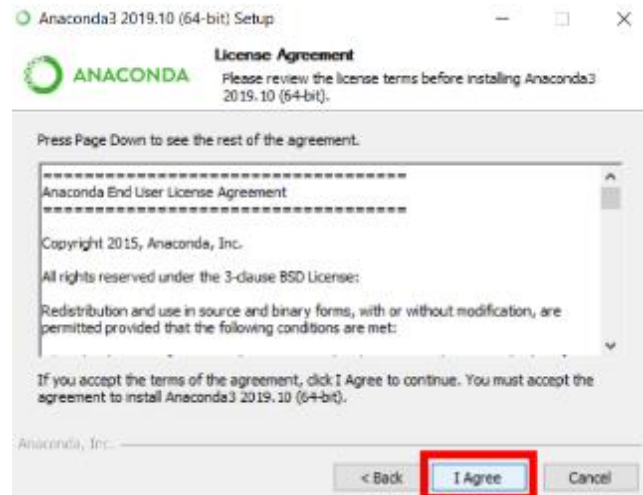


Figure 7 Install Anaconda 3.

Select the settings for "Just Me" (for the current user only) or "All Users" (for all users on the computer). Click "Next".

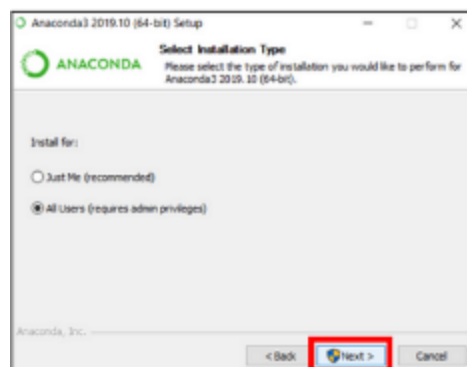


Figure 8 Install Anaconda 4.

Select the destination directory to install Anaconda. Click "Next".

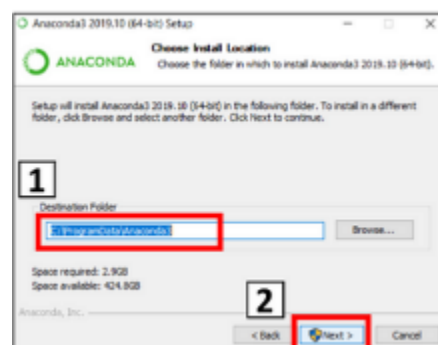


Figure 9 Install Anaconda 5.

Check the "Add Anaconda to my PATH environment variable" checkbox if you want to use Anaconda from the command line without having to specify the full path. Click "Install" to start the installation process.

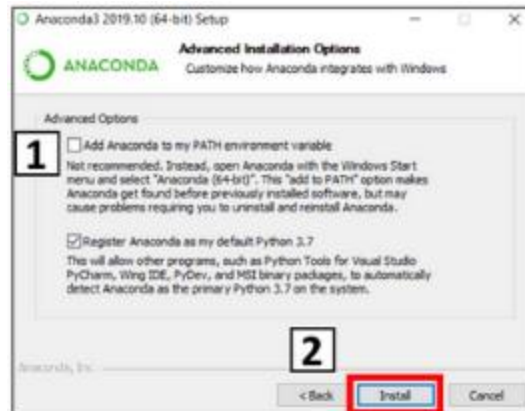


Figure 10 Install Anaconda 6.

Once the installation is complete, click "Next" and then click "Finish".

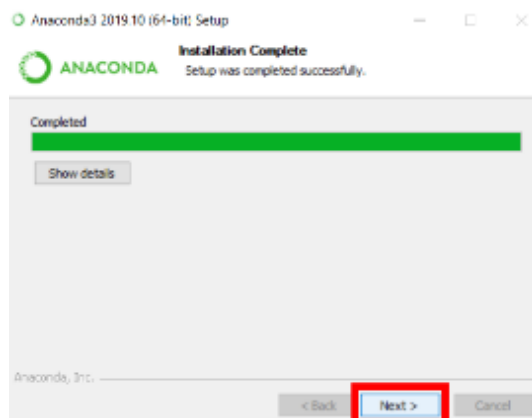


Figure 11 Install Anaconda 7.

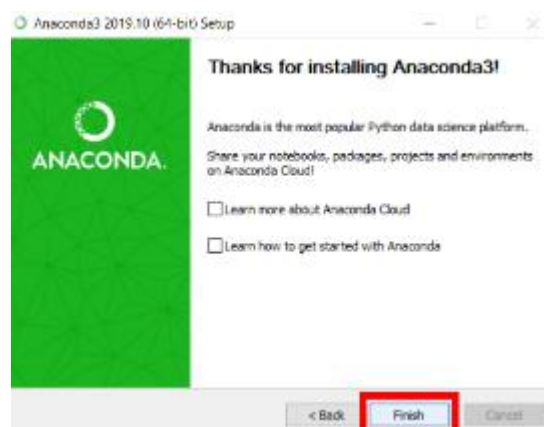


Figure 12 Install Anaconda 8.

3. Check settings.

Open Terminal (or Command Prompt on Windows) and type the command "conda --version" to check the Anaconda version and confirm that it was installed successfully.

To check the version of Python included with Anaconda, you can use the command "python --version".

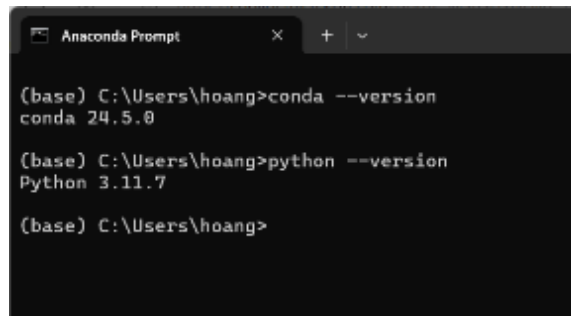


Figure 13 Check the Anaconda version and Python version.

Start Anaconda Navigator by Searching for Anaconda Navigator and Clicking on Anaconda Navigator.

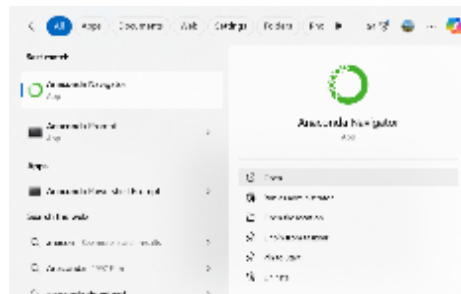


Figure 14 Start Anaconda Navigator.

From Anaconda Navigator, you can easily create new environments, install software packages, and open Jupyter Notebook or other tools.

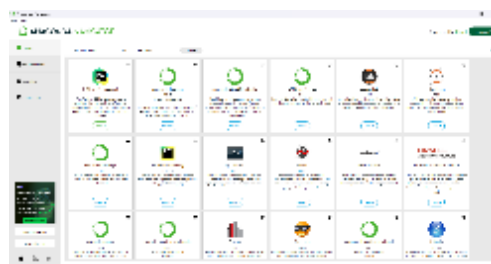


Figure 15 Anaconda Navigator.

1.3 Understand the basic syntax of Python.

1.3.1 Open Jupyter Notebook.

Anaconda integrates Jupyter Notebook, a powerful tool for writing and executing Python code in an interactive interface. Steps to open Jupyter Notebook from Anaconda Navigator:

- Create or select a specific environment for the project.

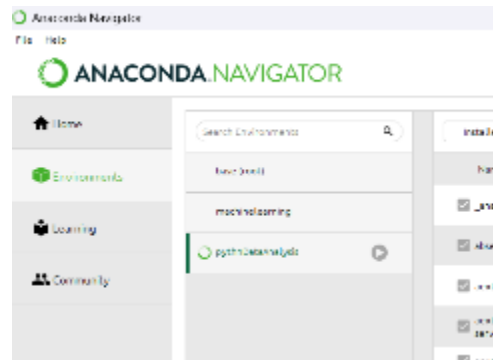


Figure 16 Open Jupyter Notebook 1.

- Return to the Home Tab select Install (if not installed) and click “Launch” under Jupyter Notebook.

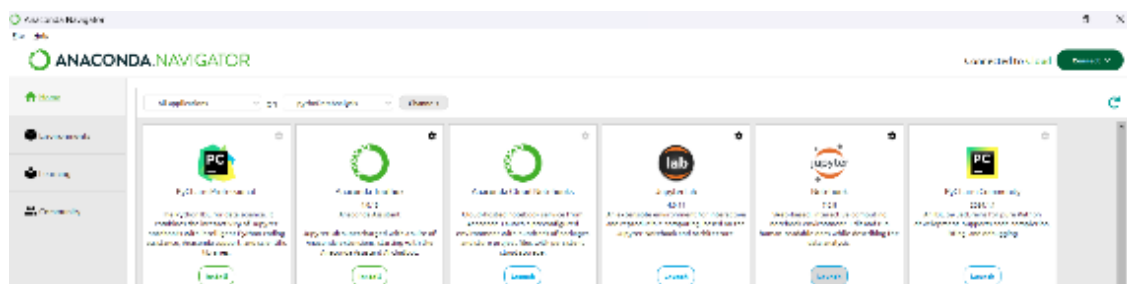


Figure 17 Open Jupyter Notebook 2.

1.3.2 Learn basic Python syntax.

1.3.2.1 Variable.

A variable in Python is a name assigned to a value to store data that a program can use and change during execution. Python is a dynamic language, so there is no need to declare a variable's data type before use, and a variable can change its data type during the program's runtime.

Declaring and using variables in Python is very simple, it just requires assigning a value to the variable using the "=" operator.

Variable naming rules:

- Variable names must begin with a letter (a-z, A-Z) or an underscore (_).
- Subsequent characters can be letters, numbers (0-9), or underscores.
- Variable names are case-sensitive (e.g. age and Age are two different variables).
- Do not use Python keywords as variable names (e.g. if, while, return, etc.).

```
File Edit View Run Kernel Settings Help
[2]: #Declare an integer variable:
      age = 25

      #Declare a float variable:
      A = 2.65

      #Declare string variable:
      name = "Quy"

      #Declare boolean variable:
      is_student = True

      #Declaring a list:
      fruits = ["apple", "banana", "cherry"]
```

Figure 18 Example of variable declaration syntax.

1.3.2.2 A loop

A loop is a control structure that allows a piece of code to be executed repeatedly. Loops are useful when you need to perform a task repeatedly with different values. Python supports two main types of loops: for and while.

- The for loop in Python is used to iterate through a sequence of values (such as a list, string, or range of numbers). The syntax of the for loop is as follows:

for variable in iterable:

- The block of code will be executed for each value in the iterable.
- Variable will in turn get the value of the elements in the iterable.

```
[4]: #Loop through a list:  
fruits = ["apple", "banana", "cherry"]  
for fruit in fruits:  
    print(fruit)
```

```
apple  
banana  
cherry
```

Figure 19 Example of "for" loop 1.

```
[5]: #Loop through a range of numbers with range():  
for i in range(5):  
    print(i)
```

```
0  
1  
2  
3  
4
```

Figure 20 Example of "for" loop 2.

- The while loop in Python continues executing the inner block of statements until the condition is selected as False. The syntax of the while loop is as follows:

while condition:

- The block command will be executed regardless of whether the condition is true (True)

```
: #Use while loop to count:  
count = 0  
while count < 5:  
    print(count)  
    count += 1
```

0
1
2
3
4

Figure 21 Example of "while" loop.

1.3.2.3 Function.

A function in Python is a block of code reorganized to perform a specific task. Functions can accept parameters and return results. Using functions helps reuse code, organize code better, and make code easier to understand.

The syntax for defining a function in Python includes the `def` keyword, the function name, a parameter list (if any), and a block of executable code. This block of code can typically return a value using the "return" keyword.

```
def function_name(parameters):  
    # Function command block  
    # Perform mission  
    # Return value if needed  
    return value
```

Figure 22 The syntax for defining a function.

- `def`: Keyword to define a function.
- `function_name`: The name of the function following variable naming rules.
- `parameters`: List of parameters (optional), there can be multiple parameters separated by commas.
- `return value`: (Optional) Returns a value from the function.

```
def greet(name):  
    print(f"Hello, {name}!")  
  
# Call function  
greet("Quy")  
  
#Function has parameters and returns value:  
def add(a, b):  
    return a + b  
  
# Call the function and save the result in a variable  
result = add(5, 3)  
print(result)
```

```
Hello, Quy!  
8
```

Figure 23 Example of Function.

1.3.2.4 Conditional function

Conditional functions in Python allow you to test conditions and perform different actions based on the results of those conditions. Conditional structures in Python primarily use if, elif (else if), and else statements.

The basic syntax of conditional functions

```
if condition1:  
    # The block will be executed if condition1 is True  
elif condition2:  
    # The block will be executed if condition2 is True  
else:  
    # The command block will be executed if all of the above conditions are False
```

Figure 24 The basic syntax of conditional functions.

```
def grade(score):  
    if score >= 90:  
        return "A"  
    elif score >= 80:  
        return "B"  
    elif score >= 70:  
        return "C"  
    elif score >= 60:  
        return "D"  
    else:  
        return "F"  
  
# Call the function and print the results  
print(grade(77))
```

C

Figure 25 Example of Conditional function.

1.4 Choose a popular library for data analysis.

1.4.1 Install libraries.

- Use Anaconda Navigator to install the necessary libraries for data analysis including Pandas, NumPy, Matplotlib, and Seaborn through the following steps:
Select the Environments tab > select the environment > select not installed > enter the library you want to download in the search bar > check the box in front of the library you want to download > click apply.

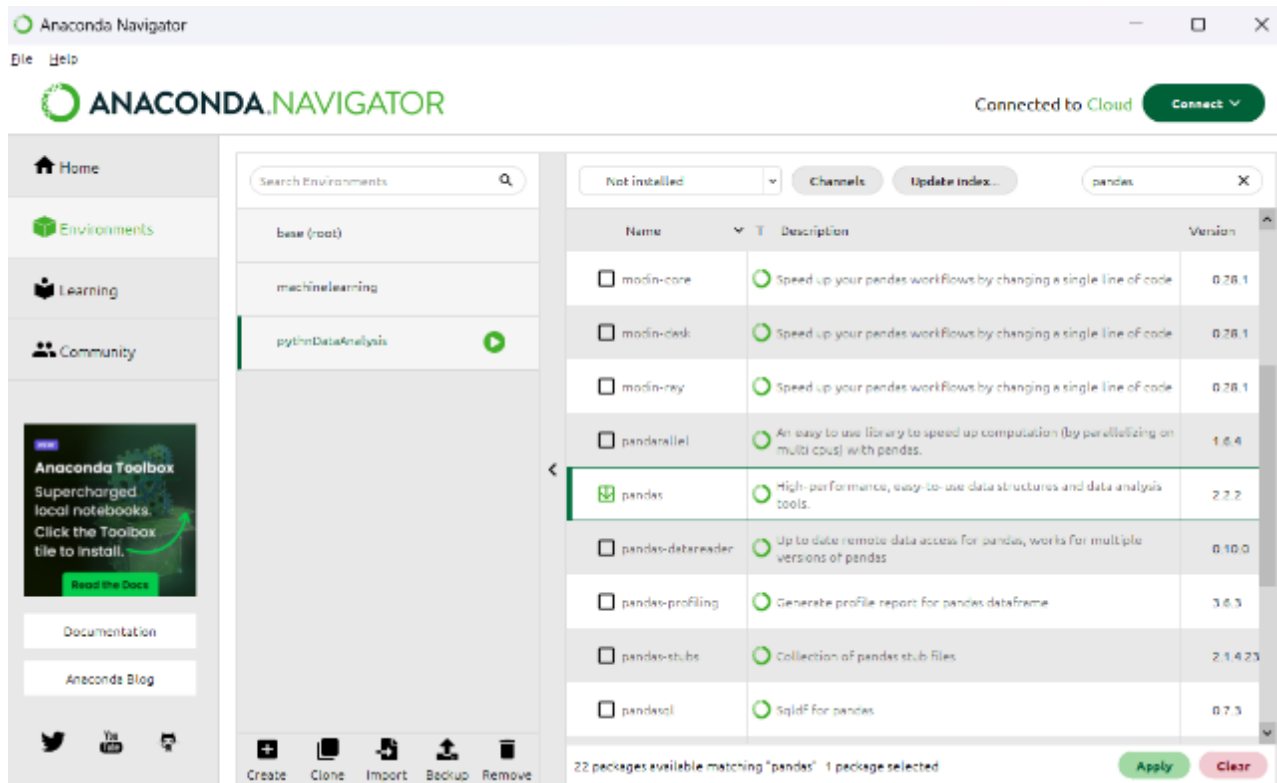


Figure 26 Use Anaconda Navigator to install the necessary libraries.

- Install Pandas, NumPy, Matplotlib, and Seaborn libraries via command line.

```
# Install Pandas
!conda install pandas

# Install NumPy
!conda install numpy

# Install Matplotlib
!conda install matplotlib

# Install Seaborn
!conda install seaborn
```

Figure 27 Install Pandas, NumPy, Matplotlib, and Seaborn libraries via command line.

1.4.2 Introduction to Libraries.

1.4.2.1 Pandas

Pandas is a powerful open-source library for the Python programming language, designed for easy data manipulation and analysis. This library provides efficient data structures and data manipulation tools, especially suitable for tabular data.

Pandas have key features such as:

- Main data structures.
- Reading and writing data.
- Handling missing data.
- Filter and select data.
- Data manipulation and transformation.

1.4.2.2 NumPy

NumPy (Numerical Python) is a basic library for scientific computing in Python. It provides a multidimensional array object (ndarray) and tools for working with these arrays.

Main function:

- Multidimensional array (ndarray): NumPy provides a powerful array object, making manipulation and calculation on numeric arrays easy and efficient.
- Vectorization operators: NumPy supports vectorization operations, which helps speed up arithmetic operations.
- Mathematical functions: NumPy provides many efficient mathematical functions for linear algebra, statistics, and Fourier transforms.
- Integrates well with other libraries: NumPy is the foundation of many other analytical and computational libraries in Python such as Pandas, Matplotlib, and Scipy.

1.4.2.3 Matplotlib

Matplotlib is a data visualization library in Python that allows the creation of high-quality graphs and charts.

Key features of Matplotlib:

- Matplotlib supports many chart types such as line plots, bar plots, scatter plots, box plots, and many other chart types.
- Matplotlib allows customization of most aspects of the plot, from color and size, to line style and labels.
- Integration with Pandas: Matplotlib integrates well with Pandas, allowing graphs to be created directly from DataFrame and Series objects.
- Ability to save graphs: Graphs can be saved in many formats such as PNG, JPG, SVG, PDF, etc.

1.4.2.4 Seaborn

Seaborn is a data visualization library based on Matplotlib that provides a high-level and easy-to-use interface for creating statistical plots.

Main function:

- Statistical charts: Seaborn provides many powerful statistical charts such as distribution plots, correlation plots, box plots, violin plots, and many other charts.
- Integration with Pandas: Seaborn integrates well with Pandas and can work directly with DataFrames.
- Theme customization: Seaborn provides beautiful and easy-to-use themes and color palettes, helping to improve the aesthetics of graphs.
- Complex data management capabilities: Seaborn supports powerful features for complex data management and visualization, helping to create multidimensional and multivariate charts.

2 Prepare a proposal for analyzing a large dataset, including the method of analysis and the outcomes to be achieved. P2

2.1 Identify the data source.

The dataset I use is a public dataset on Kaggle called Vehicle Sales Data. Vehicle Sales Data is a dataset on Kaggle that provides detailed information on vehicle sales by automakers worldwide. This dataset includes details such as year, make, model, trim, body type, transmission type, VIN (Vehicle Identification Number), registration status, condition rating, reading Odometer, exterior and interior colors, seller information, Manheim Market Report (MMR), sale price and sale date. (Anwar, 2024)

Link source: <https://www.kaggle.com/datasets/syedanwarafриди/vehicle-sales-data>.

2.2 Use Python to read data from CSV formats.

First, I will declare the required libraries.

```
#Declare required libraries  
import pandas as pd  
import numpy as np  
import seaborn as sns  
import matplotlib.pyplot as plt
```

Figure 28 Declare required libraries.

Next, I will read the data, explore and check the data such as the total number of rows, columns, and summary of dataframe information, and display the top 5 rows and last 5 rows of the dataframe.

```
df = pd.read_csv('car_prices.csv')
df.shape
```

```
(558838, 16)
```

Figure 29 The total number of rows, and columns.

The above data has 558838 rows and 16 columns.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 558838 entries, 0 to 558837
Data columns (total 16 columns):
 #   Column                Non-Null Count  Dtype  
---  -
 0   year                  558838 non-null  int64  
 1   make                  548537 non-null  object  
 2   model                 548439 non-null  object  
 3   trim                  548187 non-null  object  
 4   body                  545643 non-null  object  
 5   transmission          493486 non-null  object  
 6   vin                   558834 non-null  object  
 7   state                 558838 non-null  object  
 8   condition             547018 non-null  float64  
 9   odometer              558744 non-null  float64  
10   color                 558089 non-null  object  
11   interior              558089 non-null  object  
12   seller                558838 non-null  object  
13   mmr                   558800 non-null  float64  
14   sellingprice          558826 non-null  float64  
15   saledate              558826 non-null  object  
dtypes: float64(4), int64(1), object(11)
memory usage: 68.2+ MB
```

Figure 30 Summary of dataframe information.

df.head()

	year	make	model	trim	body	transmission	vin	state	condition	odometer	color	interior	seller	mmr	sellingprice	saledate
0	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg566472	ca	5.0	16639.0	white	black	kia motors america inc	20500.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
1	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg561319	ca	5.0	9393.0	white	beige	kia motors america inc	20900.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)
2	2014	BMW	3 Series	328i SULEV	Sedan	automatic	wba3c1c51ek116351	ca	45.0	1331.0	gray	black	financial services remarketing (lease)	31900.0	30000.0	Thu Jan 15 2015 04:30:00 GMT-0800 (PST)
3	2015	Volvo	S60	T5	Sedan	automatic	yv1612tb4f1310967	ca	41.0	14282.0	white	black	volvo na rep/world omni	27500.0	27750.0	Thu Jan 29 2015 04:30:00 GMT-0800 (PST)
4	2014	BMW	6 Series Gran Coupe	650i	Sedan	automatic	wba6b2c57ed129731	ca	43.0	2641.0	gray	black	financial services remarketing (lease)	66000.0	67000.0	Thu Dec 18 2014 12:30:00 GMT-0800 (PST)

Figure 31 Display the top 5 rows of the dataframe.

df.tail()

	year	make	model	trim	body	transmission	vin	state	condition	odometer	color	interior	seller	mmr	sellingprice	sa
558833	2012	Ram	2500	Power Wagon	Crew Cab	automatic	3c6fd5et6cg112407	wa	5.0	54393.0	white	black	i-5 uhlmann rv	30200.0	30800.0	0
558834	2012	BMW	X5	xDrive35d	SUV	automatic	5uxzw0c50c668465	ca	48.0	50561.0	black	black	financial services remarketing (lease)	29800.0	34000.0	0
558835	2015	Nissan	Altima	2.5 S	sedan	automatic	1n4a13ap0lc216050	ga	38.0	16650.0	white	black	enterprise vehicle exchange / tra / rental / t...	15100.0	11100.0	0
558836	2014	Ford	F-150	XLT	SuperCrew	automatic	1ftfw1et2eke87277	ca	34.0	15009.0	gray	gray	ford motor credit company llc pd	29600.0	26700.0	0
558837	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg566472	ca	5.0	16639.0	white	black	kia motors america inc	20500.0	21500.0	1

Figure 32 Display the last 5 rows of the dataframe.

2.3 Data processing and cleaning.

1. Handling duplicate data.

First, I will check if the data is duplicate or not, using pandas I show the count and show duplicate rows.

```
print(df.duplicated().sum())
```

1

```
df[df.duplicated()]
```

	year	make	model	trim	body	transmission	vin	state	condition	odometer	color	interior	seller	mmr	sellingprice	saledate
558837	2015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg566472	ca	5.0	16639.0	white	black	kia motors america inc	20500.0	21500.0	Tue Dec 16 2014 12:30:00 GMT-0800 (PST)

Figure 33 The count and show duplicate rows.

I found out there is a duplicate row of data at index 558837. So, I will proceed to delete this row from the dataframe.

```
df = df.drop_duplicates()

print(df.duplicated().sum())
```

0

Figure 34 Remove duplicate data.

2. Handling missing values.

Check and count the number of missing values (NA/null) in each column of the DataFrame.

```
# Check and count the number of missing values (NA/null) in each column of the DataFrame.
df.isna().sum()
```

```
year          0
make         10301
model        10399
trim         10651
body         13195
transmission  65352
vin           4
state         0
condition    11820
odometer      94
color        749
interior     749
seller        0
mmr          38
sellingprice  12
saledate      12
dtype: int64
```

Figure 35 Check and count the number of missing values.

I will remove the vin column in the dataframe because I realize this column is not important and meaningful for my analysis, it is just the Vehicle Identifier, a unique code for each vehicle.

```
df.drop(columns=['vin'], inplace=True)
df.shape
```

```
(558837, 15)
```

Figure 36 Delete the vin column.

My data has 558838 records. Except for the transmission column, which has a lot of missing data, the other columns account for a fairly small proportion of this data, so I will fill in the missing data for the transmission column and delete the rows with missing values. again. Deleting records with missing values will not affect the results of my analysis because I have a good number of data points.

```
#filling missing values
mode_value = df['transmission'].mode()[0]
df['transmission'].fillna(mode_value, inplace=True)
df.isna().sum()
```

```
year          0
make          10301
model         10399
trim          10651
body          13195
transmission   0
state          0
....         ....
```

Figure 37 Filling missing values for the transmission column.

```
df.dropna(inplace=True)
df.isna().sum()
```

```
year          0
make          0
model         0
trim          0
body          0
transmission  0
state         0
condition     0
odometer      0
color         0
interior      0
seller        0
mmr           0
sellingprice  0
saledate      0
dtype: int64
```

Figure 38 Remove the missing values.

3. Convert data to the required format.

Convert the year column and saledate column to datetime keep only the date part and remove the time part.

```
# Convert column 'year' to datetime
df['year'] = pd.to_datetime(df['year'], format='%Y')
```

```
# Convert the saledate column to datetime and keep only the date part and remove the time part.
df['saledate'] = pd.to_datetime(df['saledate'], utc=True, format='mixed').dt.date
```

Figure 39 Convert the year column and the saledate column to datetime.

4. Handling data outliers.

First, I will visualize the data as a scatter plot so I can see the outliers. Additionally, I also create a scatter plot function to save resources and time.

```
def scatter_plot(data, x_column, y_column):  
  
    plt.figure(figsize=(10, 8))  
    plt.scatter(data[x_column], data[y_column], color="b", alpha=0.5, marker="o")  
    plt.title(f'Scatter Plot of {y_column} over {x_column}')  
    plt.xlabel(x_column)  
    plt.ylabel(y_column)  
    plt.grid(True)  
    plt.show()
```

Figure 40 Creating a function that draws a scatter plot.

```
scatter_plot(df, 'year', 'sellingprice')
```

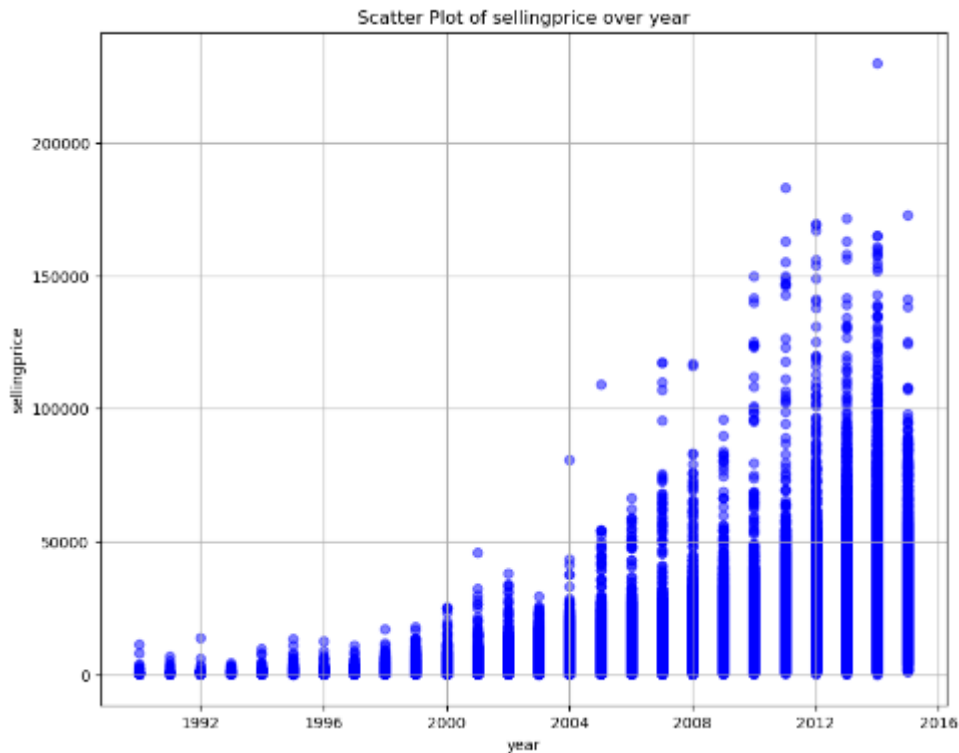


Figure 41 Scatter Plot of sellingprice over year.

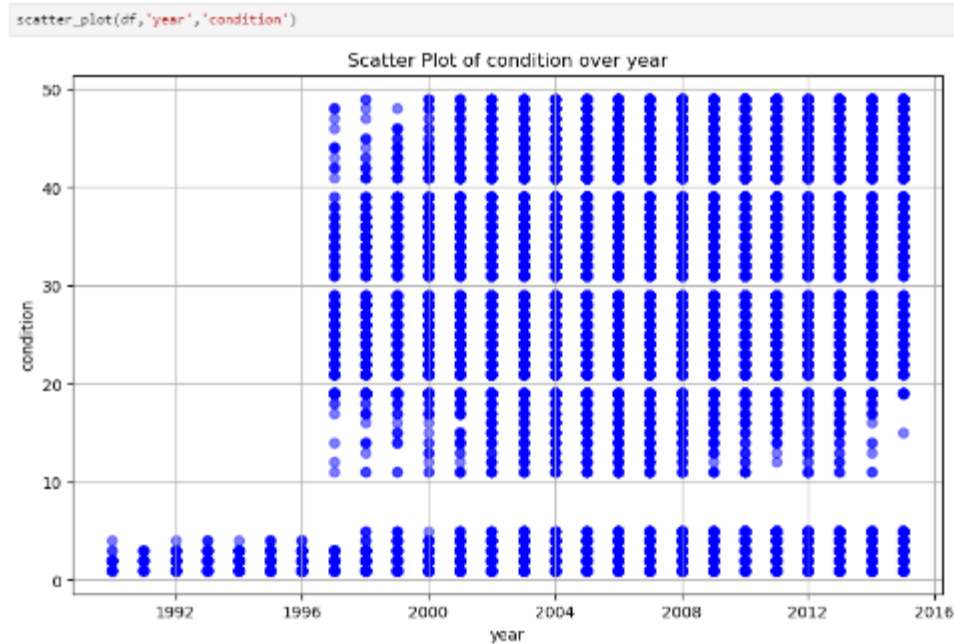


Figure 42 Scatter Plot of condition over the year.

From the chart, we see that points with a value greater than 175,000 in the sellingprice column and points with a value less than or equal to 10 in the condition column are outliers. I will proceed to delete these points.

```
df = df.loc[(df['sellingprice'] <= 175000)]
df = df.loc[(df['condition'] > 10)]
```

Figure 43 Delete outliers.

3 Design functions and choose charts to visualize data and results. P3

3.1 Identify functions needed for data analysis.

To analyze data effectively, many functions are needed to process, explore, and visualize data. In the data that I prepare to analyze and visualize, the following functions are needed: filter and select data, sort data, calculate basic statistics, functions to group data by one or more columns, and aggregate calculations, functions to draw data visualization tables.

3.2 Write functions in Python to perform analytical tasks.

The function calculates basic statistics


```
#The function calculates basic statistics  
def calculate_statistics(data, column):  
    mean = data[column].mean()  
    median = data[column].median()  
    std_dev = data[column].std()  
    return mean, median, std_dev
```

Figure 44 The function calculates basic statistics.

Create the function "plot_pie_chart" to draw a pie chart.

```
def plot_pie_chart(df, column):  
    data = df[column].value_counts()  
    plt.figure(figsize=(10, 8))  
    plt.pie(data, labels=data.index, autopct='%1.1f%%', startangle=140, colors=sns.color_palette("husl", len(data)))  
    plt.title(f'Ratio of values in the column {column}')  
    plt.axis('equal')  
    plt.show()
```

Figure 45 Create the function "plot_pie_chart".

Create the function "plot_bar_chart" to draw a bar chart.

```
def plot_bar_chart(df, x, y, hue=None):  
    plt.figure(figsize=(12, 6))  
    sns.barplot(data=df, x=x, y=y, hue=hue, palette='viridis')  
    plt.title(f'Column chart: {y} against {x}', fontsize=16)  
    plt.xlabel(x, fontsize=14)  
    plt.ylabel(y, fontsize=14)  
    plt.xticks(rotation=45)  
    plt.show()
```

Figure 46 Create the function "plot_bar_chart".

Create a function to group columns and calculate counts

```
def group_and_count(df, group_col, count_col):
    """
    The function groups data by column 'group_col' and counts by column 'count_col'.

    Parameters:
    - df: DataFrame contains data
    - group_col: Column name used for grouping (eg: 'make', 'year')
    - count_col: Column name used to count quantity (eg 'sales')

    Return:
    - New DataFrame after grouping and counting
    """
    # Group and total quantities
    grouped_df = df.groupby(group_col)[count_col].count().reset_index()

    return grouped_df
```

Figure 47 Create the function "group_and_count".

3.3 Use data visualization libraries such as Matplotlib or Seaborn to create charts to visualize data and results.

I drew a pie chart representing the car sales rate of car manufacturers in the world. The chart is shown below.

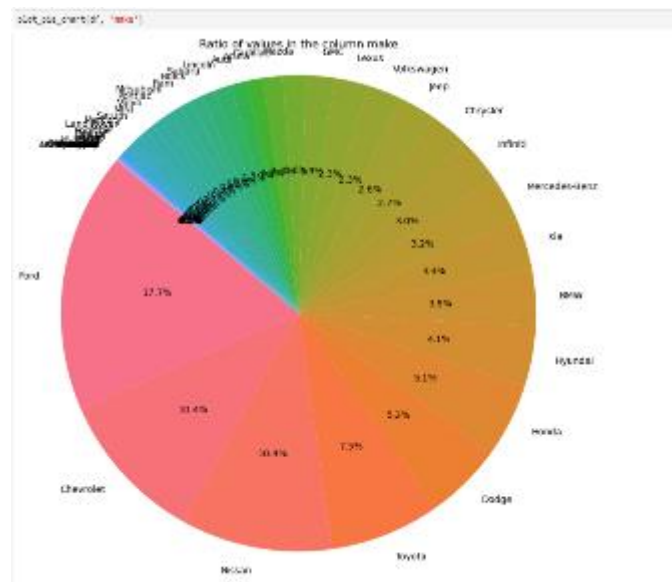


Figure 48 Percentage of car sales of automakers in the world.

Through the chart above, we can see the best-selling car brands, led by Ford with 17.7% of the car sales market, followed by Chevrolet and Nissan with 10.4%, followed by Toyota with 7.5%. Those are the 4 car brands most trusted and favored by consumers.

Going further, I will filter the dataframe by Ford car brand and visualize to see the best-selling car type in the Ford car brand through the pie chart drawn below.

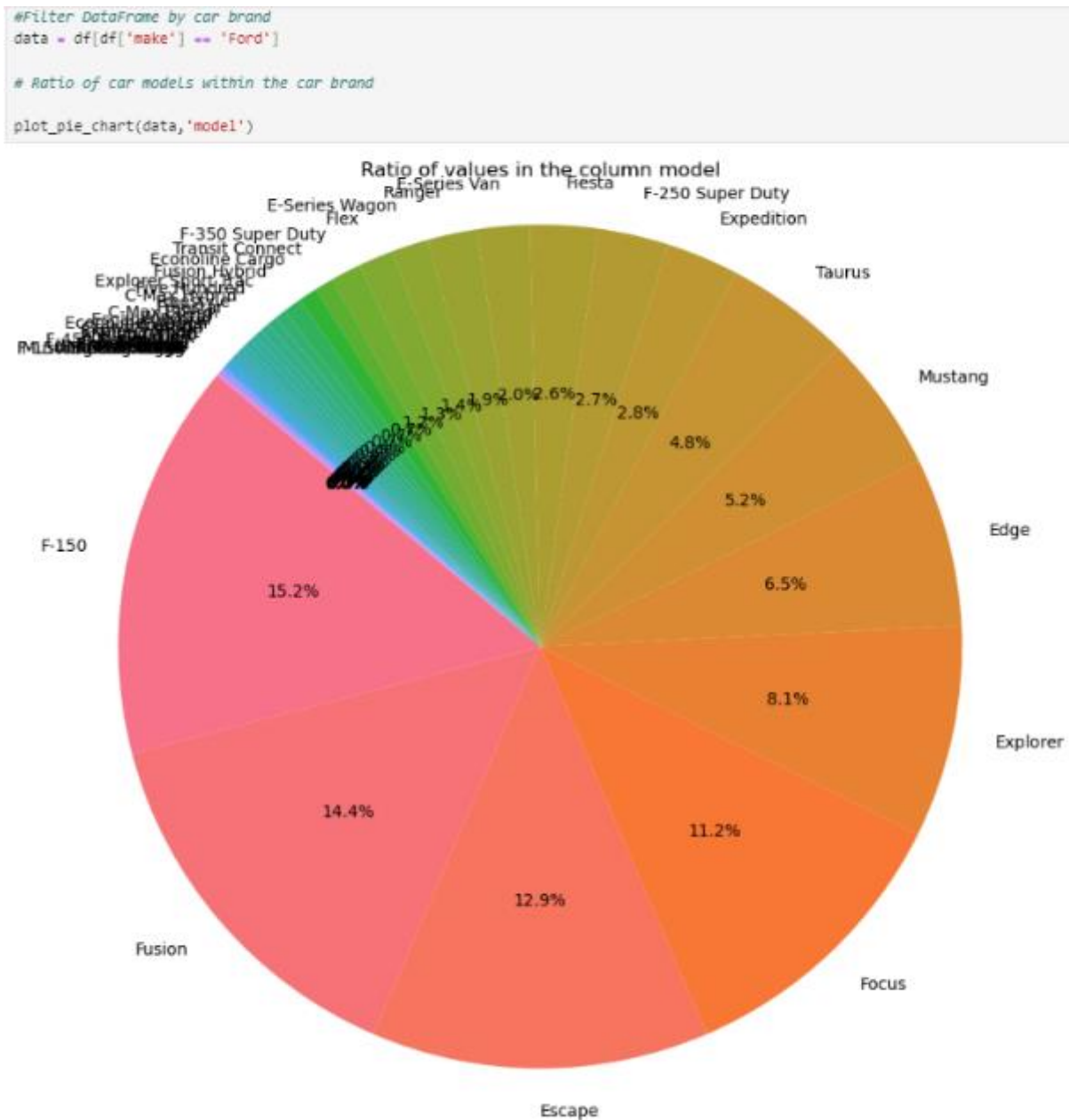


Figure 49 The ratio of car models sold in the Ford brand.

Through the chart above, I see that the best-selling Ford cars are F-150, Fusion, Escape, Focus, and Explorer with corresponding rates of 15.2%, 14.4%, 12.9%, 11.2%, and 8.1%.

The chart below shows the correlation between the Manheim Market Report (the vehicle's estimated market value) and the selling price.

```
scatter_plot(df,'mmr','sellingprice')
```

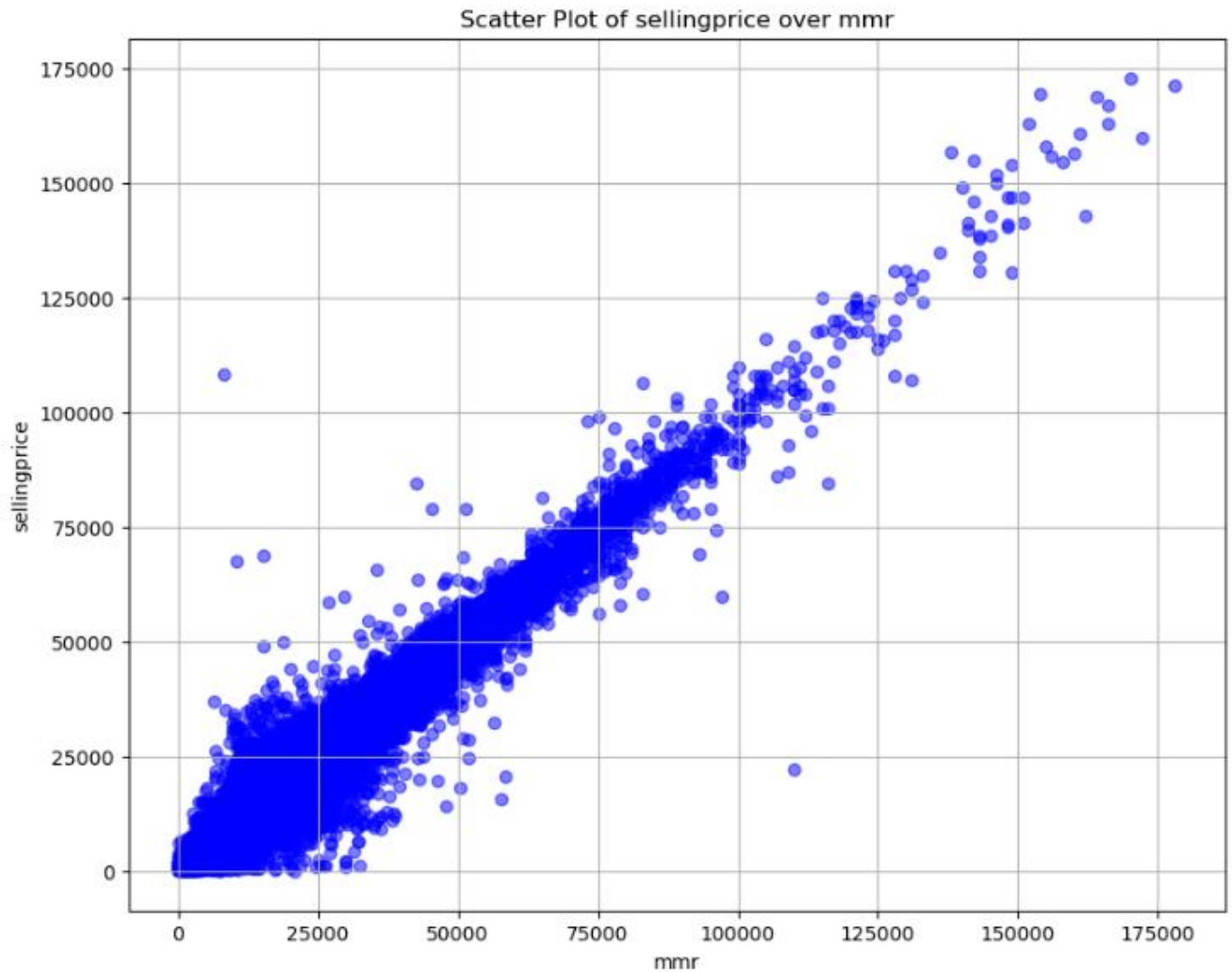


Figure 50 The correlation between the MMR and Sellingprice.

Looking at the chart, we see that the selling price is proportional to the estimated market value of the car, the higher the MMR, the higher the selling price.

The next chart shows the correlation between selling price and condition.

```
scatter_plot(df, 'condition', 'sellingprice')
```

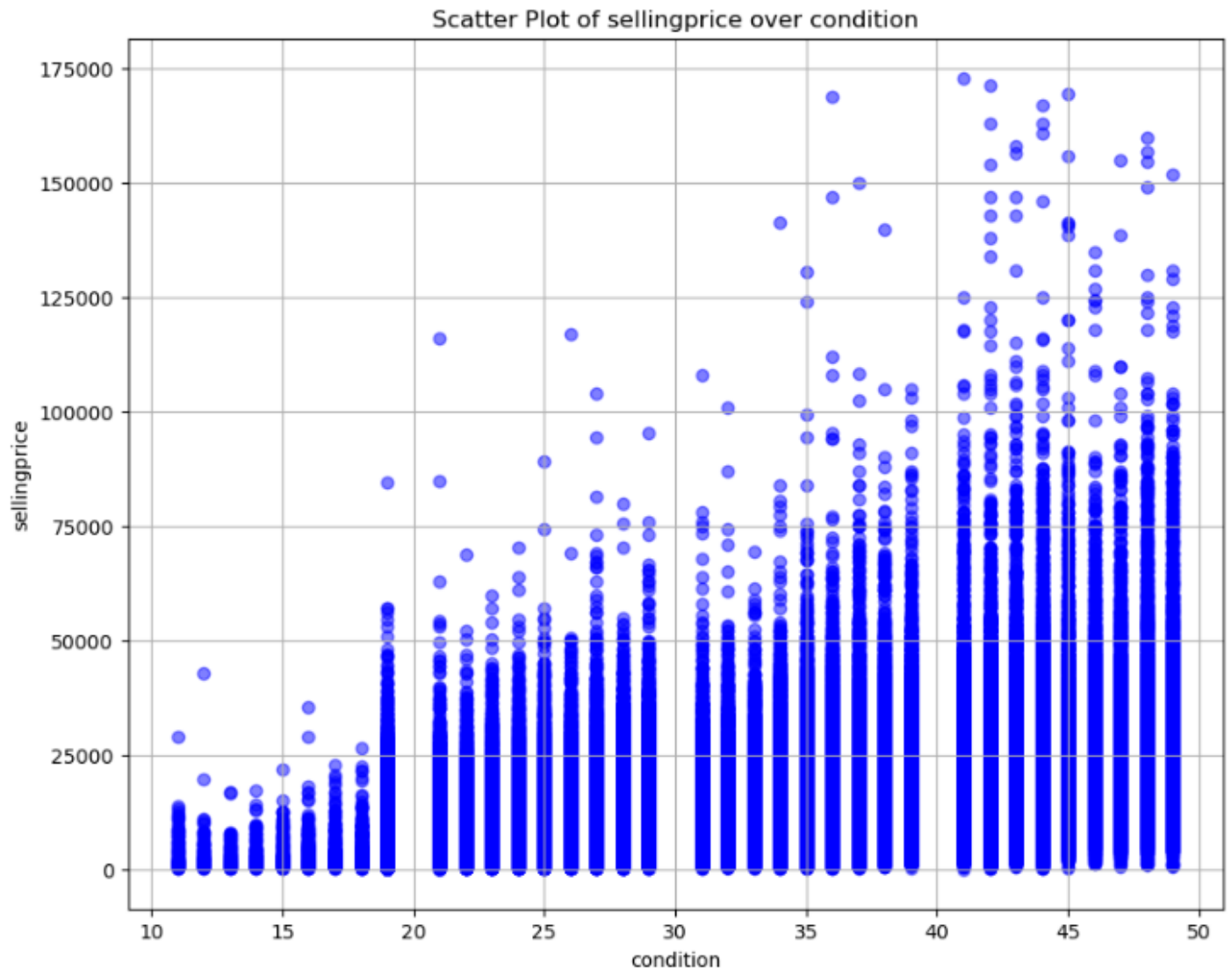


Figure 51 The correlation between selling price and condition.

Similar to MMR, the Condition is also proportional to the selling price of the car. The higher the amenities and quality of the car, the higher the selling price will be.

Next, I will look at the trend in selling prices over the years. I will visualize the data as a column chart below.

```
df_sorted = df.sort_values(by='year', ascending=True)
```

```
plot_bar_chart(df_sorted, 'year', 'sellingprice', hue=None)
```

C:\Users\hoang\AppData\Local\Temp\ipykernel_7764\264567326.py:3: FutureWarning:

Passing 'palette' without assigning 'hue' is deprecated and will be removed in v0.14.0. Assign the 'x' variable to 'hue' and set 'legend=False' for the same effect.

```
sns.barplot(data=df, x=x, y=y, hue=hue, palette='viridis')
```

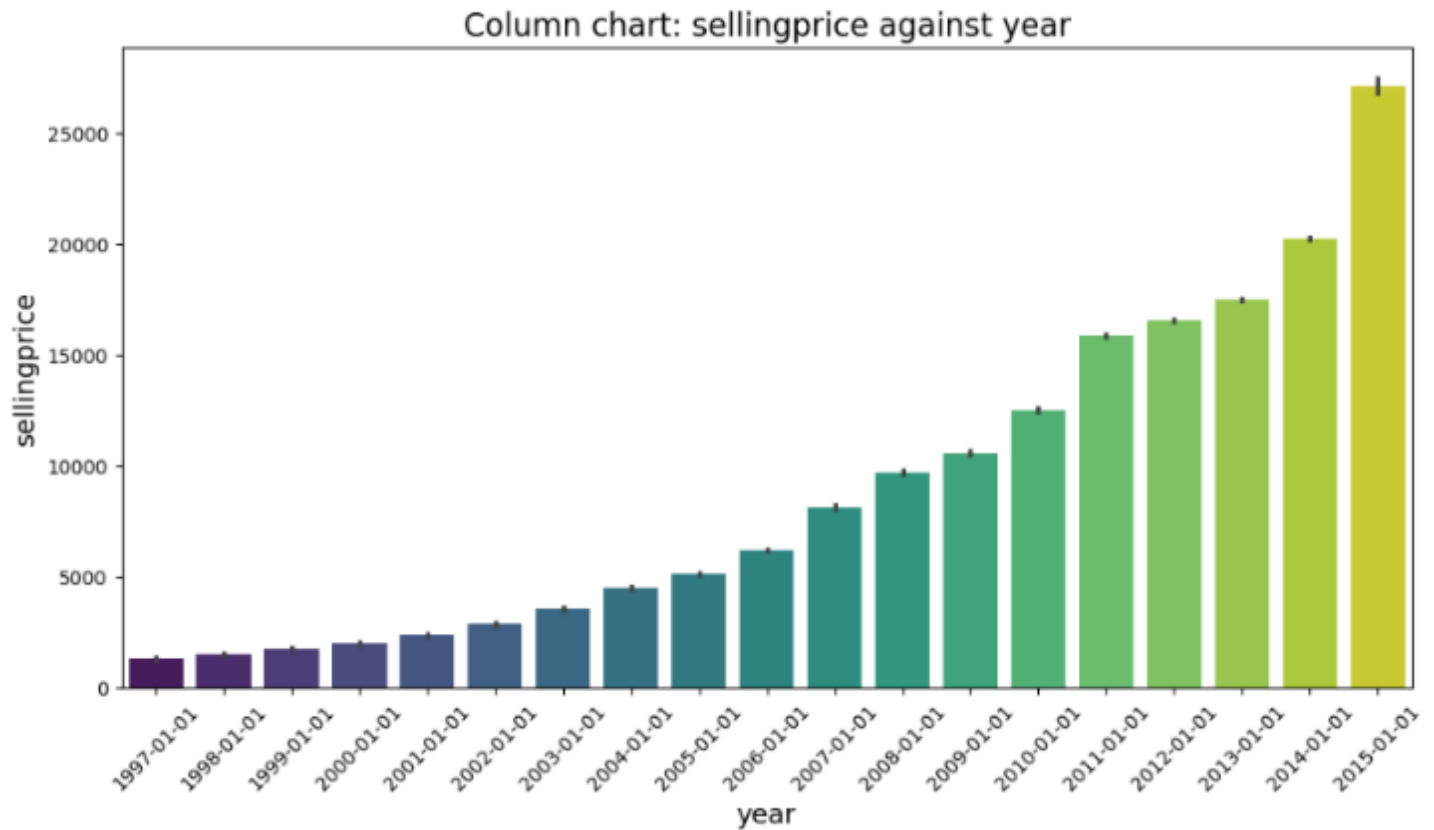


Figure 52 The trend in selling prices over the years.

From the chart, it is easy to see that car selling prices tend to increase gradually over the years.

Next, I will group the two columns to form a new dataframe with year and salequantity columns. I then visualized this data to look at car sales over the years. The chart below will show this.

```
salequantity = group_and_count(df, 'year', 'make')
salequantity = salequantity.rename(columns={'make': 'salequantity'})
print(salequantity)
```

	year	salequantity
0	1997-01-01	786
1	1998-01-01	1088
2	1999-01-01	1844
3	2000-01-01	2944
4	2001-01-01	4576
5	2002-01-01	7149
6	2003-01-01	9881
7	2004-01-01	13189
8	2005-01-01	16491
9	2006-01-01	21208
10	2007-01-01	25325
11	2008-01-01	27219
12	2009-01-01	18111
13	2010-01-01	23305
14	2011-01-01	43974
15	2012-01-01	93446
16	2013-01-01	89099
17	2014-01-01	60365
18	2015-01-01	6046

Figure 53 A new dataframe with year and salequantity columns.

```
plot_bar_chart(salequantity, 'year', 'salequantity', hue='salequantity')
```

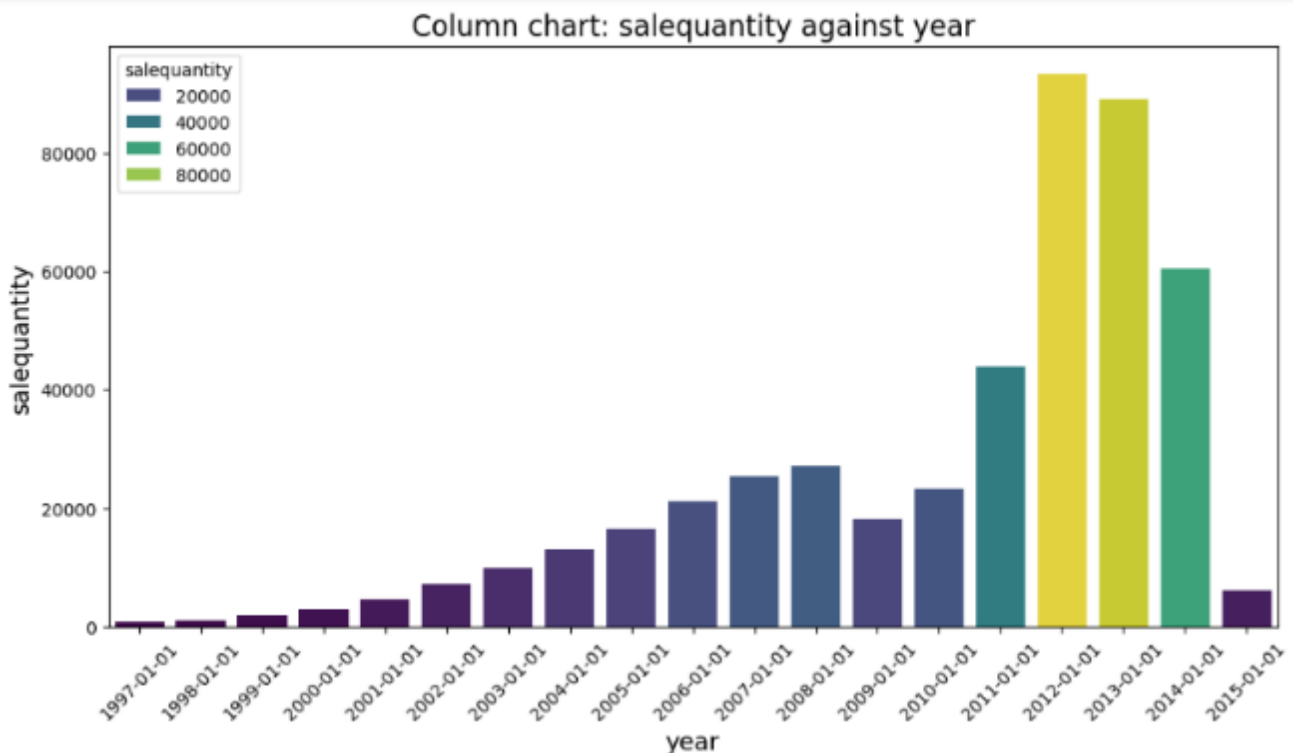


Figure 54 The number of cars sold over the years.

Overall, it can be seen that the number of cars sold has increased over the years. However, in 2015, the number of sales decreased rapidly. Combined with the previous chart, it can be seen that part of the reason is because the selling price in 2015 increased too high compared to previous years.

4 Create a detailed test plan and identify the expected outcomes of the analysis. P4

4.1 Definition of test case.

In programming and software testing, a test case is a specific case to test a part of a system's functionality, or a set of input data used to check the correctness of a program, function, or feature. The goal of creating test cases is to determine whether they work as expected or not.

4.2 Mission of Test Case

In software testing, test cases are responsible for verifying the correctness of a part of the program or the entire program. Here are some important tasks of test cases:

- **Verify correctness:** Test cases help check whether the source code works according to expectations or not. It ensures that the program executes the correct logic and returns the expected results.
- **Error detection:** Test cases help detect errors in source code. If there is an error, the test case will help determine the location and cause of the error so it can be corrected.
- **Test for special situations:** Test cases ensure that the program properly handles special situations, such as empty input data, boundary values, or maximum/minimum cases.
- **Performance and load testing:** Some test cases test the program's performance, such as execution time, loading multiple data, or loading from multiple sources simultaneously.
- **Integration testing:** Test cases check the integration between system components, ensuring they work properly when combined.

4.3 The Necessity of Test cases.

Test cases play a very important role in the software testing process. It helps ensure that all software functions operate according to established requirements and standards. Test cases provide a detailed guide for testers, helping them know the exact steps to take, input data, and expected results. This helps minimize errors and ensure consistency during testing. Furthermore, test cases also help detect potential errors and problems early, thereby saving time and costs on software repair and maintenance.

4.4 Create a detailed test plan and identify the expected outcomes of the analysis.

Table 1 Check out the scatter plot function.

Check out the scatter plot function
Test case 1: Data in the correct format

Goal: Test if the function can create a scatter plot with properly formatted data.	Implementation steps: call function: Scatter_plot with 2 columns year(datetime) and sellingprice(float) <code>scatter_plot(df,'year','sellingprice')</code>	Expected result: Scatter plot displayed with x-axis as 'year' and y-axis as 'sellingprice'.
Test case 2: Missing or invalid data: Data is missing a column.		
Goal: Test whether the function handles errors when columns x_column or y_column are missing.	Implementation steps: call function: Scatter_plot Input data is column year and column vin (removed from dataframe: df) <code>scatter_plot(df,'year','vin')</code>	Expected result: The function reports an error that column 'vin' was not found.

Table 2 Check out the calculate_statistics function.

Check out the calculate_statistics function.		
Test case 3: Data has null value		
Goal: Make sure the function works properly with different data sets and conditions. Check if the function handles null or NaN values in column column	Implementation steps: Creates a DataFrame with a single column named 'A', containing the values [1, 2, None, 4, 5]. Call function calculate_statistics with data as data and column 'A' <code>calculate_statistics(data, 'A')</code>	Expected result: The function ignores null values and performs exact calculations with the remaining values. <ul style="list-style-type: none"> • mean = 3.0 • median = 3.0 • td_dev ≈ 1.82

LO3 Develop a software tool to analyze a large data set for a given scenario.

5 Build a software tool for analyzing a large dataset according to a developed design. P5

5.1 Introducing Streamlit.

Streamlit is an open-source library for Python that makes it easy to create web applications for data science and machine learning projects. Designed specifically for data scientists and machine learning engineers, Streamlit allows you to build intuitive and efficient user interfaces without needing deep knowledge of web development. (Nhan, 2019) (MyGPT, 2023) (Huy, 2021) (Ichi.pro, n.d.)



Figure 55 Streamlit.

Some notable features of Streamlit:

- Easy to use: You can create web applications with just a few lines of Python code.
- Good integration with Python libraries: Streamlit is compatible with many popular libraries such as pandas, matplotlib, seaborn, plotly, Keras, PyTorch, etc.
- No front-end knowledge required: You don't need to know HTML, CSS, or JavaScript to use Streamlit.
- Visual Data Display: Streamlit supports displaying charts, graphs, maps, images, audio, video, and more.
- Easy Deployment: You can deploy your app to Streamlit Cloud for free if the source code is publicly hosted on GitHub.

Streamlit is a powerful and easy-to-use tool that helps you quickly create web applications to present and share your data analysis results.

5.2 Write a web application in Python with the Streamlit library.

5.2.1 Identify the data source.

In this assignment, I continue to use the dataset from assignment 1. It is a public dataset on Kaggle called Vehicle Sales Data. It provides detailed information about vehicle sales of automobile manufacturers worldwide. This dataset includes details such as year, make, model, body type, transmission type, VIN (Vehicle

Identification Number), registration status, condition rating, odometer reading, exterior and interior color, seller information, Manheim Market Report (MMR), sale price, and sale date.

5.2.2 Read and process data.

First, I will declare the required libraries.

```
car_prices_app.py > ...
1  import streamlit as st
2  import pandas as pd
3  import numpy as np
4  import seaborn as sns
5  import matplotlib.pyplot as plt
```

Figure 56 Declare required libraries.

Next, I will read the data, explore and check the data like row count, column count, and data frame summary, and display the data via Streamlit.

```
# Load data
df = pd.read_csv('car_prices.csv')

# Display the original DataFrame
st.write(df)

df.shape
```

Figure 57 Read and show data.

To display the data on Streamlit, in the Terminal, I will run the command Streamlit run car_prices_app.py. This command will start the Streamlit application and open it in a web browser.

```
PS C:\Users\hoang\OneDrive\Tài liệu\Advanced Programming for Data Analysis\streamlit> streamlit run car_prices_app.py
```

Figure 58 Command to display the data on Streamlit.

	year	make	model	trim	body	transmission	vin	state	conditi
0	2,015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg566472	ca	
1	2,015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg561319	ca	
2	2,014	BMW	3 Series	328i SULEV	Sedan	automatic	wba3c1c51ek116351	ca	
3	2,015	Volvo	S60	T5	Sedan	automatic	yv1612tb4f1310987	ca	
4	2,014	BMW	6 Series G	650i	Sedan	automatic	wba6b2c57ed129731	ca	
5	2,015	Nissan	Altima	2.5 S	Sedan	automatic	1n4al3ap1fn326013	ca	
6	2,014	BMW	M5	Base	Sedan	automatic	wbsfv9c51ed593089	ca	
7	2,014	Chevrolet	Cruze	1LT	Sedan	automatic	1g1pc5sb2e7128460	ca	
8	2,014	Audi	A4	2.0T Premiur	Sedan	automatic	wauffaf13en030343	ca	
9	2,014	Chevrolet	Camaro	LT	Convert	automatic	2g1fb3d37e9218789	ca	

(558838, 16)

Figure 59 Display the dataframe.

We see the original data includes 558838 rows and 16 columns.

Handling duplicate data: I will check if the data is duplicated by using pandas to process, calculate and display the count and duplicate rows to the web via Streamlit.

Check for duplicates

Drop duplicate

Number of duplicate rows: 1

	year	make	model	trim	body	transmission	vin	state	condition	odor
558,837	2,015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg566472	ca	5	1

Number of duplicate rows after dropping duplicates: 0

Figure 60 The count and show duplicate rows.

I found out that there is a duplicate data row at index 558837. So I proceeded to delete this row from the dataframe.

Handling missing values: Check and count the number of missing values (NA/null) in each column of the DataFrame. Also, I will remove the "vin" column in the dataframe because I realize this column is not important and has no meaning for my analysis, it is just the Vehicle Identifier, a unique code for each vehicle.

```
# Check and count the number of missing values (NA/null) in each column
st.write("Missing values in each column before cleaning")
st.write(df.isna().sum())

# Drop 'vin' column
df.drop(columns=['vin'], inplace=True)
st.write(f"DataFrame shape after dropping 'vin' column: {df.shape}")
```

Figure 61 Handling missing values.

Missing values in each column before cleaning

	0
year	0
make	10,301
model	10,399
trim	10,651
body	13,195
transmission	65,352
vin	4
state	0
condition	11,820
odometer	94

DataFrame shape after dropping 'vin' column: (558837, 15)

Figure 62 Check and count the number of missing values.

My data has 558838 records. Except for the transmission column, there is a lot of missing data, the other columns make up a fairly small percentage of this data, so I will fill in the missing data for the transmission column and delete the rows with missing values. Deleting records with missing values will not affect my analysis results because I have a large number of data points.

```
# Fill missing values in 'transmission' column with the mode value
mode_value = df['transmission'].mode()[0]
df['transmission'].fillna(mode_value, inplace=True)
st.write("Missing values in each column after filling 'transmission' column")
st.write(df.isna().sum())

# Drop remaining rows with missing values
df.dropna(inplace=True)
st.write("Missing values in each column after dropping remaining rows")
st.write(df.isna().sum())
```

Figure 63 Fill missing values in the 'transmission' column with the mode value and drop the remaining rows with missing values.

Missing values in each column after dropping remaining rows

	0
model	0
trim	0
body	0
transmission	0
state	0
condition	0
odometer	0
color	0
interior	0
seller	0

Figure 64 Remove the missing values.

Convert data to the required format:

Convert the year column and saledate column to datetime keep only the date part and remove the time part.

```
# Convert 'year' to datetime
df['year'] = pd.to_datetime(df['year'], format='%Y')

# Convert 'saledate' to datetime and keep only the date part
df['saledate'] = pd.to_datetime(df['saledate'], utc=True, format='mixed').dt.date
```

Figure 65 Convert the year column and the saledate column to datetime.

Handling data outliers:

First, I will visualize the data as a scatter plot so that outliers can be seen using the "st.scatter_chart" function provided by the Streamlit library.

```
st.write("Scatter Plot of sellingprice over year.")
st.scatter_chart(df, x="year", y="sellingprice")
st.write("Scatter Plot of condition over the year.")
st.scatter_chart(df, x="year", y="condition")
```

Figure 66 Visualize the data as a scatter plot.

Scatter Plot of sellingprice over year.

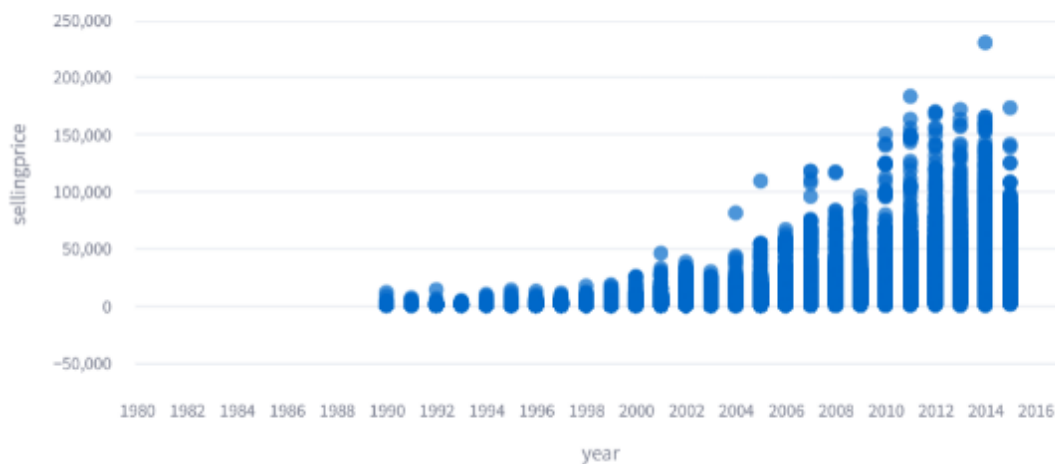


Figure 67 Scatter Plot of sellingprice over year.

Scatter Plot of condition over the year.

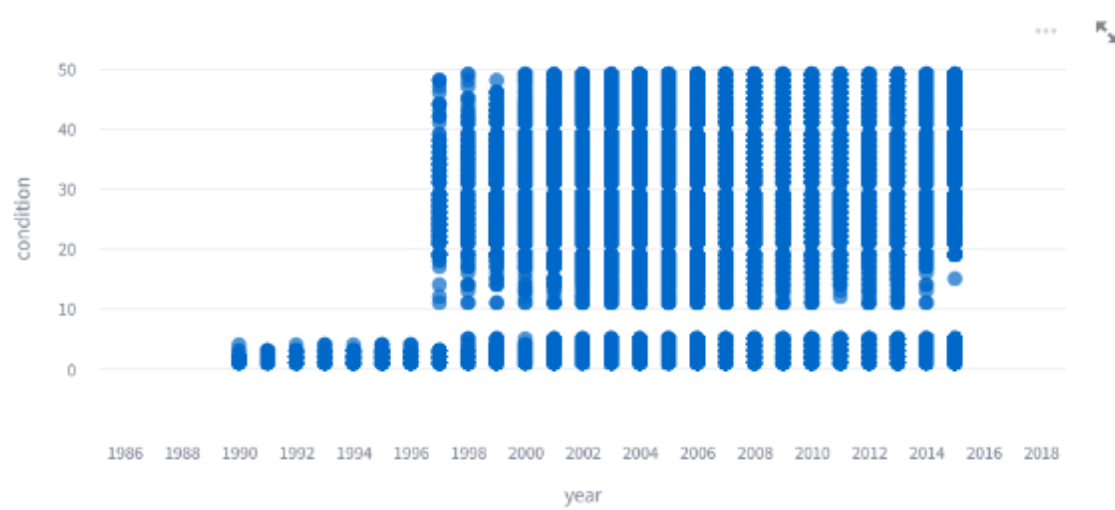


Figure 68 Scatter Plot of condition over the year.

From the chart, we see that points with a value greater than 175,000 in the sellingprice column and points with a value less than or equal to 10 in the condition column are outliers. I will proceed to delete these points.

```
# Apply the outlier filters
df = df.loc[(df['sellingprice'] <= 175000)]
df = df.loc[(df['condition'] > 10)]
```

Figure 69 Apply the outlier filters.

DataFrame shape after filtering outliers: (466046, 15)

	year	make	model	trim	body	transmission	state	col
76,407	2011-01-01 00:00:00	Toyota	RAV4	Base	SUV	automatic	oh	
76,408	2011-01-01 00:00:00	Toyota	RAV4	Base	SUV	automatic	oh	
76,409	2011-01-01 00:00:00	Toyota	Sienna	LE 7-Passenger Mobilit	Minivan	automatic	fl	
76,410	2011-01-01 00:00:00	Toyota	RAV4	Base	SUV	automatic	oh	
76,411	2011-01-01 00:00:00	Toyota	RAV4	Base	SUV	automatic	oh	
76,412	2011-01-01 00:00:00	Toyota	RAV4	Base	SUV	automatic	oh	
76,413	2011-01-01 00:00:00	Toyota	Prius	Three	Hatchback	automatic	oh	
76,414	2011-01-01 00:00:00	Toyota	RAV4	Base	SUV	automatic	oh	
76,415	2011-01-01 00:00:00	Toyota	RAV4	Sport	SUV	automatic	nj	
76,416	2011-01-01 00:00:00	Toyota	RAV4	Base	SUV	automatic	oh	
76,417	2011-01-01 00:00:00	Toyota	RAV4	Base	SUV	automatic	oh	

Figure 70 Data after filtering outliers.

It can be seen that after being processed and cleaned, the data has been reduced from 558838 rows and 16 columns to 466046 rows and 15 columns.

Cleaned data will:

- Ensure accuracy: Raw data often contains many errors such as missing values, outliers, invalid or inconsistent data.
- Remove noise, improve the quality and reliability of analysis.
- Improve model performance: For machine learning models, clean and high-quality data will help the model learn better and make more accurate predictions, avoiding overfitting.
- Data uniformity: Data from different sources may not be consistent in format, measurement units or data types. Data cleaning and normalization helps ensure that all data is consistent and easy to compare or combine.
- Detect and correct errors, adjust for outliers and ensure that data is complete and accurate.

- Increase efficiency and save time: Clean data increases efficiency in the analysis process and saves time, as there is no need to spend a lot of time dealing with errors during the analysis or model building process.

5.2.3 Analyze and visualize data.

5.2.3.1 Write functions in Python to perform analytical tasks.

To analyze data effectively, many functions are needed to process, explore and visualize data. In the data that I prepare for analysis and visualization, the following functions are needed: filtering and selecting data, sorting data, calculating basic statistics, functions to group data by one or more columns and summarizing calculations, functions to draw data visualization tables. Therefore, I will proceed to write some necessary functions for this assignment to save resources and time.

Function to calculate basic statistics.

```
def calculate_statistics(data, column):  
    mean = data[column].mean()  
    median = data[column].median()  
    std_dev = data[column].std()  
    stat_column = column  
    st.write(f"### Basic Statistics for {stat_column}")  
    st.write(f"Mean: {mean}")  
    st.write(f"Median: {median}")  
    st.write(f"Standard Deviation: {std_dev}")
```

Figure 71 Function to calculate basic statistics.

Function to create a pie chart.

```
def plot_pie_chart(data, column):  
    sizes = data[column].value_counts()  
    labels = sizes.index  
    fig, ax = plt.subplots()  
    ax.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=90)  
    ax.axis('equal') # Equal aspect ratio ensures that pie is drawn as a circle.  
    st.pyplot(fig)
```

Figure 72 Function to create a pie chart.

Create a function to group columns and calculate counts.

```
def group_and_count(df, group_col, count_col):  
    # Group and total quantities  
    grouped_df = df.groupby(group_col)[count_col].count().reset_index()  
    return grouped_df
```

Figure 73 Create a function to group columns and calculate counts.

5.2.3.2 Analyze and visualize data.

I drew a pie chart representing the car sales rate of car manufacturers in the world. The chart is shown below.

```
#A pie chart representing the car sales rate of car manufacturers in the world  
st.write("### Percentage of car sales of automakers in the world")  
plot_pie_chart(df, 'make')
```

Figure 74 The car sales rate of car manufacturers in the world.

Percentage of car sales of automakers in the world

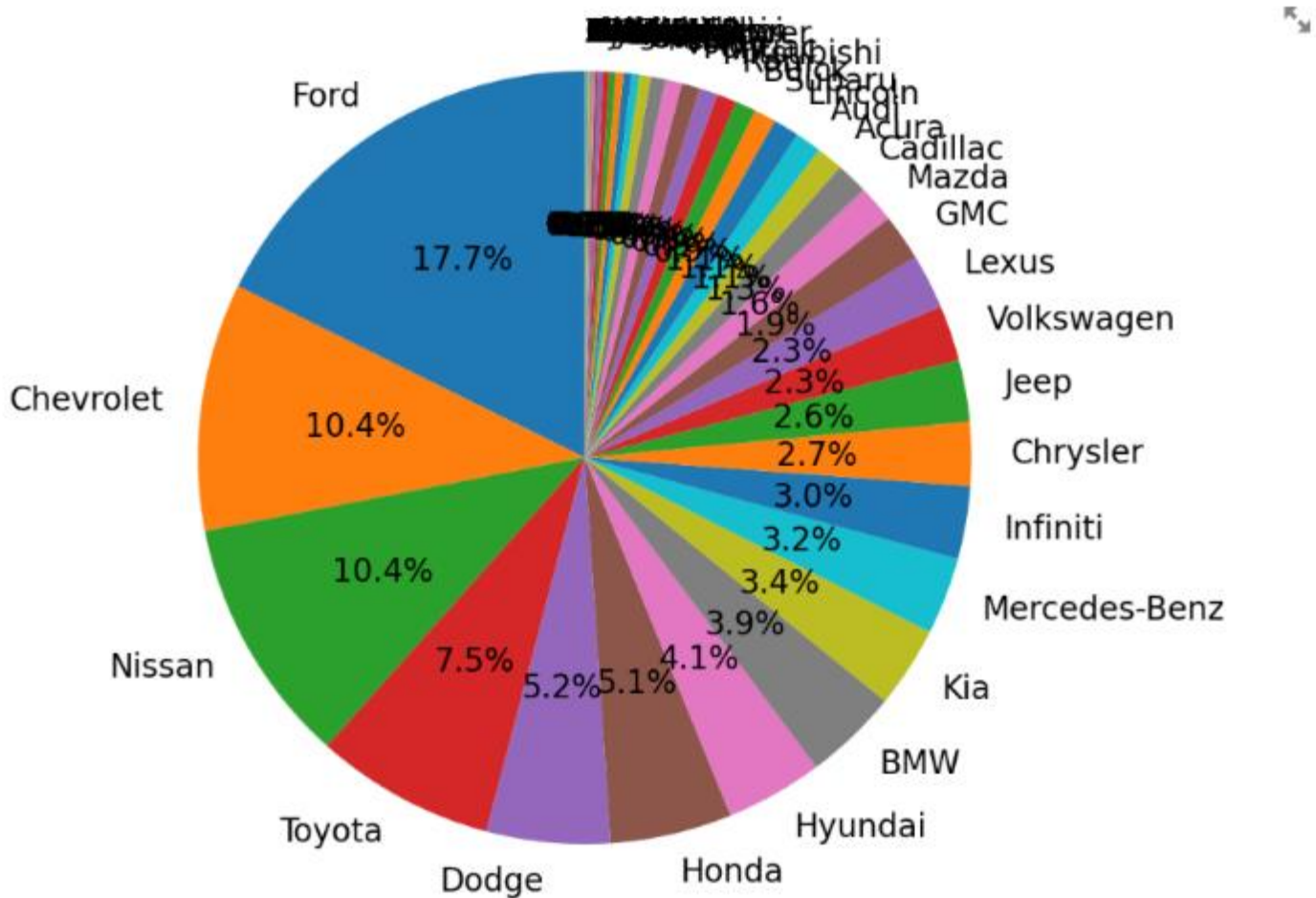


Figure 75 Percentage of car sales of automakers in the world.

Through the chart above, we can see the best-selling car brands, led by Ford with 17.7% of the car sales market share, followed by Chevrolet and Nissan with 10.4%, and then Toyota with 7.5%. These are the 4 car brands that are most trusted and preferred by consumers.

Besides, Brands such as Honda, Dodge, Hyundai, Kia, Mercedes-Benz, Infiniti, Chrysler, Jeep, Volkswagen, Lexus, GMC, Mazda, and Cadillac all have smaller market shares, ranging from 2.3% to 5.2%. This shows the fierce competition among car manufacturers.

Although Ford, Chevrolet, Nissan, and Toyota account for the majority of the market share, there are still many different brands that account for smaller parts of the market. This shows the diversity in consumer choices.

Luxury brands such as Mercedes-Benz, Lexus, and Cadillac are also present in the chart, indicating that there is a large consumer base willing to pay more for luxury vehicles. Therefore, there are separate policies for each consumer cluster based on financial level.

This chart may reflect current consumer trends, with a preference for established and reputable brands such as Ford and Chevrolet.

Going further, I will filter the dataframe by Ford car brand and visualize to see the best-selling car type in the Ford car brand through the pie chart drawn below.

```
#Filter DataFrame by car brand
data = df [df['make'] == 'Ford']

st.write("### The best-selling car type in the Ford car brand")
# Ratio of car models within the car brand
plot_pie_chart(data, 'model')
```

Figure 76 Filter DataFrame by car brand and Ratio of car models within the car brand

The best-selling car type in the Ford car brand

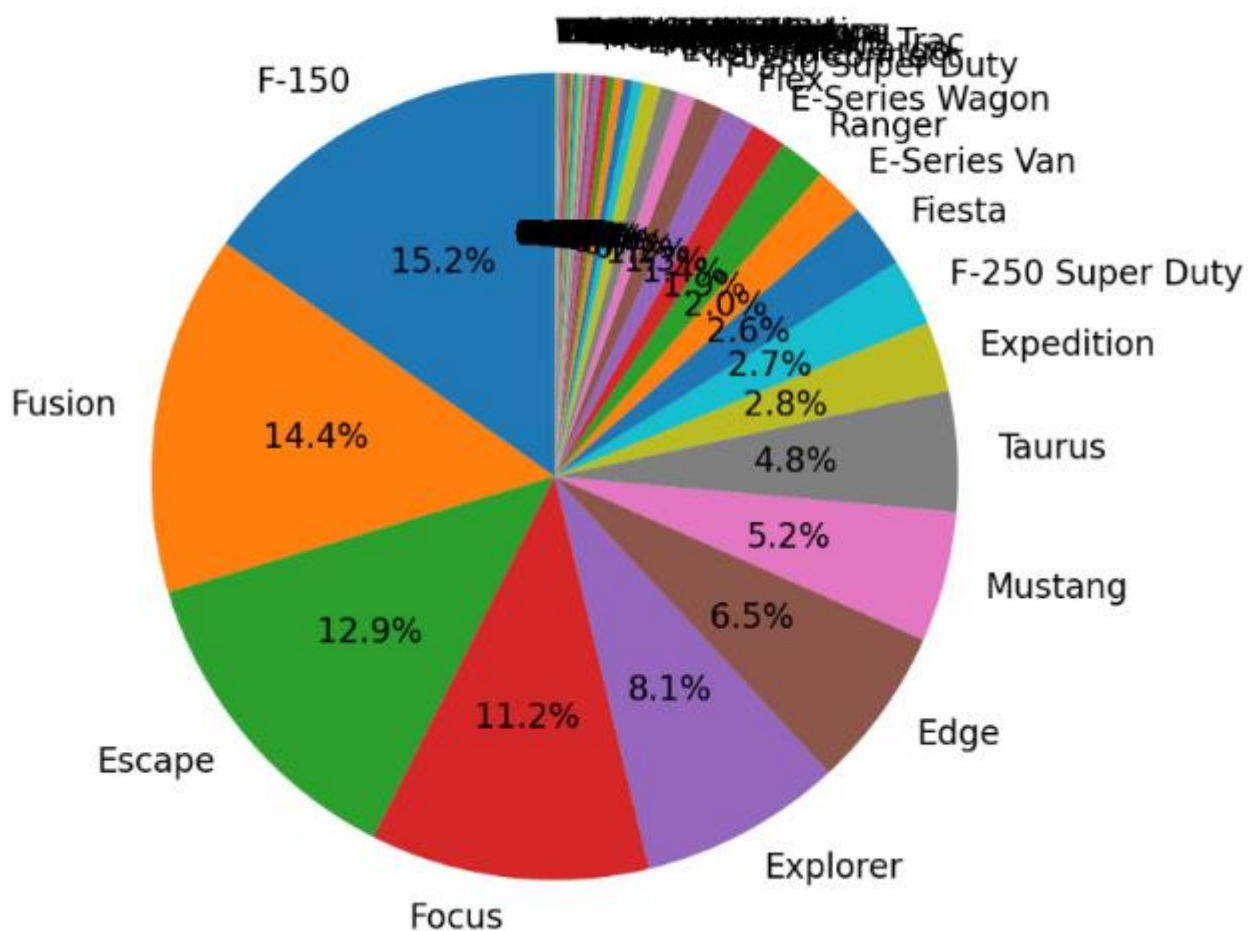


Figure 77 The best-selling car type in the Ford car brand.

Through the chart above, I see that the best-selling Ford models are the F-150, Fusion, Escape, Focus, and Explorer with respective percentages of 15.2%, 14.4%, 12.9%, 11.2%, and 8.1%.

In addition, Ford has a wide range of best-selling models, from pickup trucks (F-150) to sedans (Fusion and Focus), and SUVs (Escape and Explorer). Other models such as Mustang, Edge, Expedition, and other lines also contribute to the total sales, although the percentage is lower. This shows that Ford meets the diverse needs of consumers and Ford has a rich and diverse product portfolio.

Additionally, the Popularity of pickup trucks: F-150 accounts for the highest percentage (15.2%), showing that Ford pickup trucks are very popular, possibly due to their versatility and powerful performance.

Besides, models like Fusion, Escape, Focus, and Explorer all have quite close sales ratios, showing strong competition among Ford's vehicle lines.

The chart below shows the correlation between the Manheim Market Report (the vehicle's estimated market value) and the selling price.

```
#The correlation between the MMR and Sellingprice
st.write("###The correlation between the MMR and Sellingprice")
st.line_chart(df, x="sellingprice", y="mmr")
```

Figure 78 The correlation between the MMR and Sellingprice code.

###The correlation between the MMR and Sellingprice

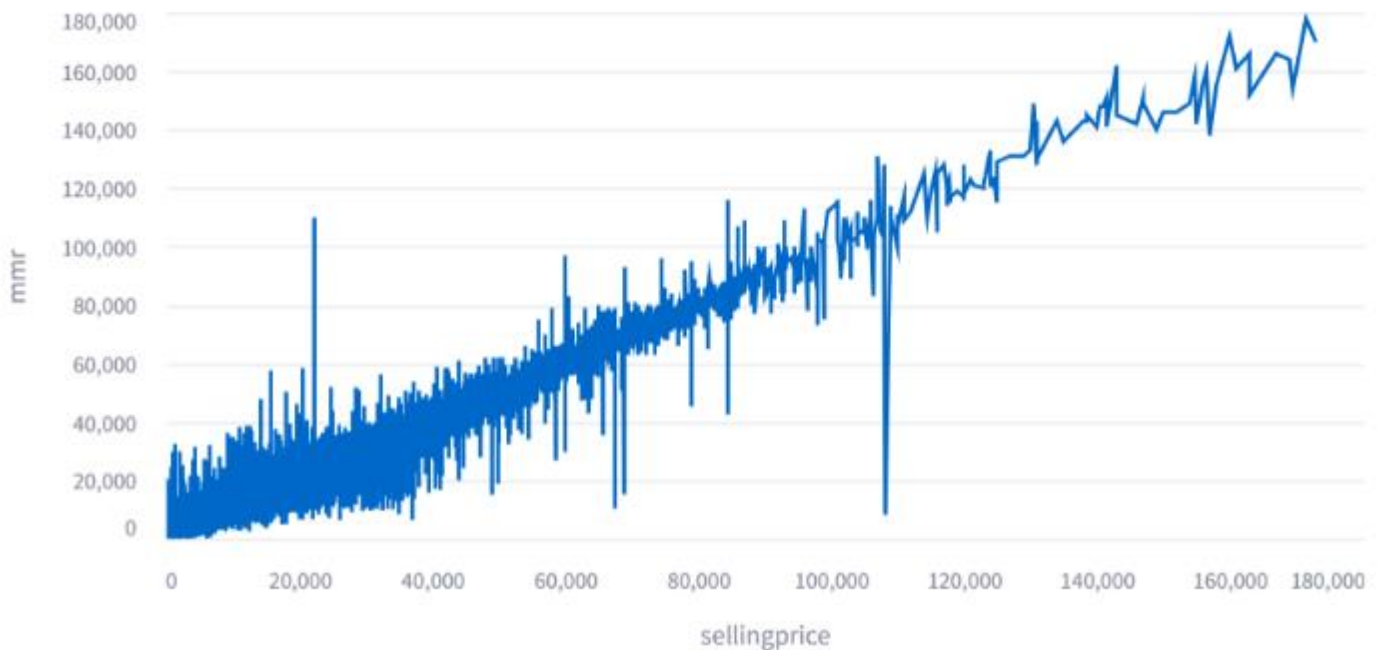


Figure 79 The correlation between the MMR and Sellingprice.

Looking at the chart, we see a clear linear relationship between the selling price and the estimated market value (MMR). This means that as the estimated market value of the vehicle increases, the selling price also increases proportionally.

Besides, the data points are concentrated at the lower end of both axes, indicating that the majority of vehicles have lower market values and selling prices. However, there are still a few vehicles with higher values, creating a data range that extends upwards.

In addition, the MMR appears to be a reliable indicator of the selling price of a vehicle. This can be useful for both buyers and sellers in pricing their vehicles appropriately.

Finally, this chart may reflect current market trends, where the estimated market value of a vehicle has a large impact on the actual selling price.

The next chart shows the correlation between selling price and condition.

```
#the correlation between selling price and condition
st.write("### The correlation between selling price and condition")
st.scatter_chart(df, x="condition",y="sellingprice")
```

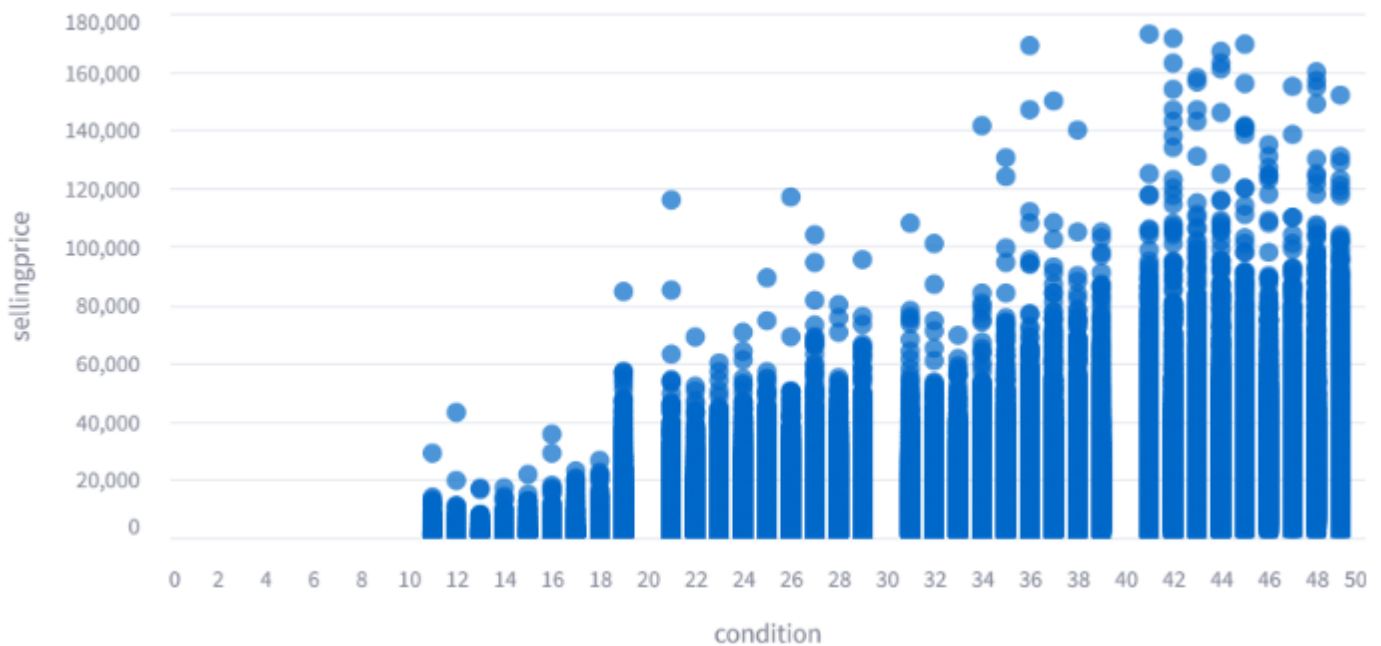



Figure 80 the correlation between selling price and condition.

The chart shows a clear linear relationship between vehicle condition and selling price. This means that as the condition of the vehicle improves, the selling price increases accordingly.

In addition, the data points are concentrated at the lower end of both axes, indicating that the majority of vehicles are in lower condition and selling price. However, there are still a few vehicles in better condition and selling price, creating a data range that extends upwards.

Additionally, the Condition Index appears to be a reliable indicator of the selling price of a vehicle. This can be useful for both buyers and sellers in pricing their vehicles appropriately.

Besides, this chart may reflect the current market trend, where the condition of the vehicle has a large impact on the actual selling price.

Next, I will look at the trend in selling prices over the years.

```
#the trend in selling prices over the years
st.write("### The trend in selling prices over the years")
df_sorted = df.sort_values(by='year', ascending=True)
st.bar_chart(df_sorted, x="year", y="sellingprice")
```


###The trend in selling prices over the years

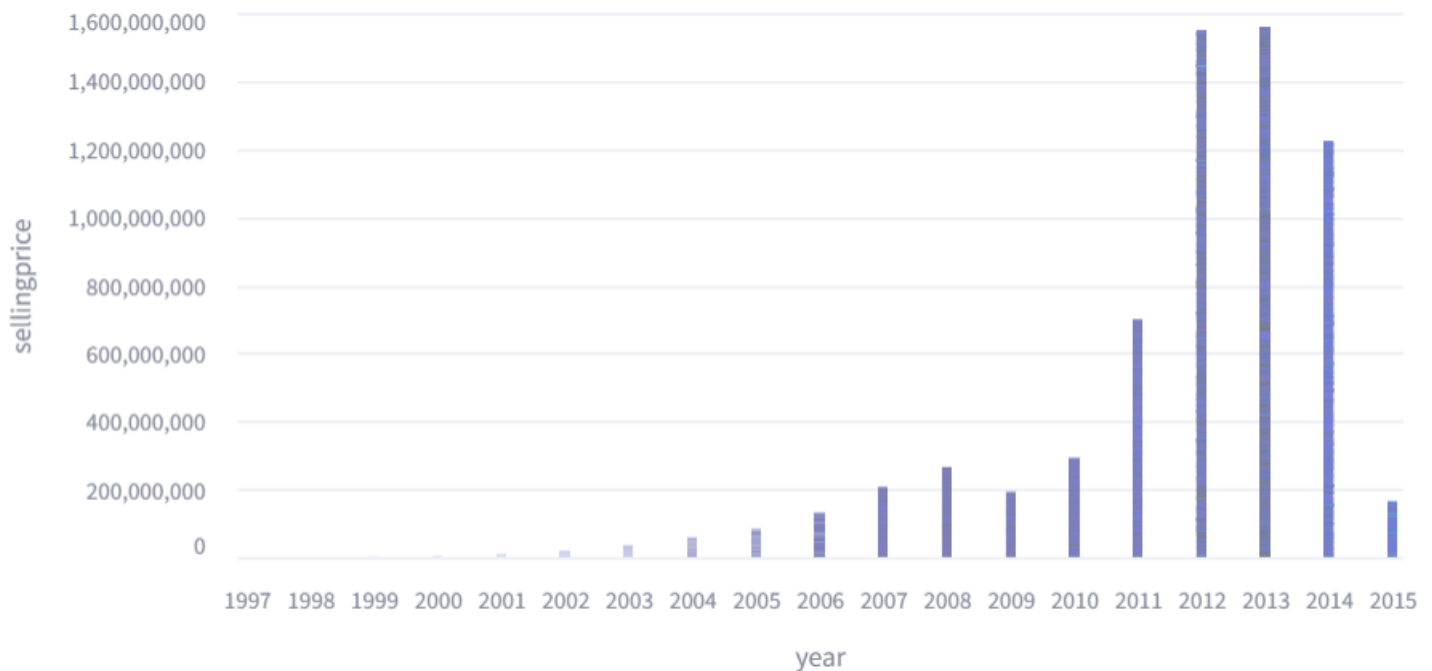


Figure 81 The trend in selling prices over the years

The chart shows that from around 2010 onwards, car prices increased sharply. This may reflect economic growth or increased demand for cars during this period. Before 2010, car prices tended to be more stable with some small fluctuations. This may indicate that the car market went through a period of stability before growing strongly. The chart also shows a long-term upward trend in car prices, which may reflect rising production costs, technological improvements, and consumer demand. Small fluctuations in prices may reflect the impact of external factors such as economic crises, changes in tax policies, or the launch of new car models.

Next, I will group the two columns to form a new dataframe with year and salequantity columns. I then visualized this data to look at car sales over the years. The chart below will show this.

```
#The number of cars sold over the years.
salequantity = group_and_count(df, 'year', 'make')

salequantity = salequantity.rename(columns={'make': 'salequantity'})

st.write(salequantity )
st.write("### The number of cars sold over the years.")
st.bar_chart(salequantity, x="year", y="salequantity")
```

	year	salequantity
0	1997-01-01 00:00:00	786
1	1998-01-01 00:00:00	1,088
2	1999-01-01 00:00:00	1,844
3	2000-01-01 00:00:00	2,944
4	2001-01-01 00:00:00	4,576
5	2002-01-01 00:00:00	7,149
6	2003-01-01 00:00:00	9,881
7	2004-01-01 00:00:00	13,189
8	2005-01-01 00:00:00	16,491
9	2006-01-01 00:00:00	21,208

Figure 82 a salequantity dataframe with year and salequantity columns

###The number of cars sold over the years.

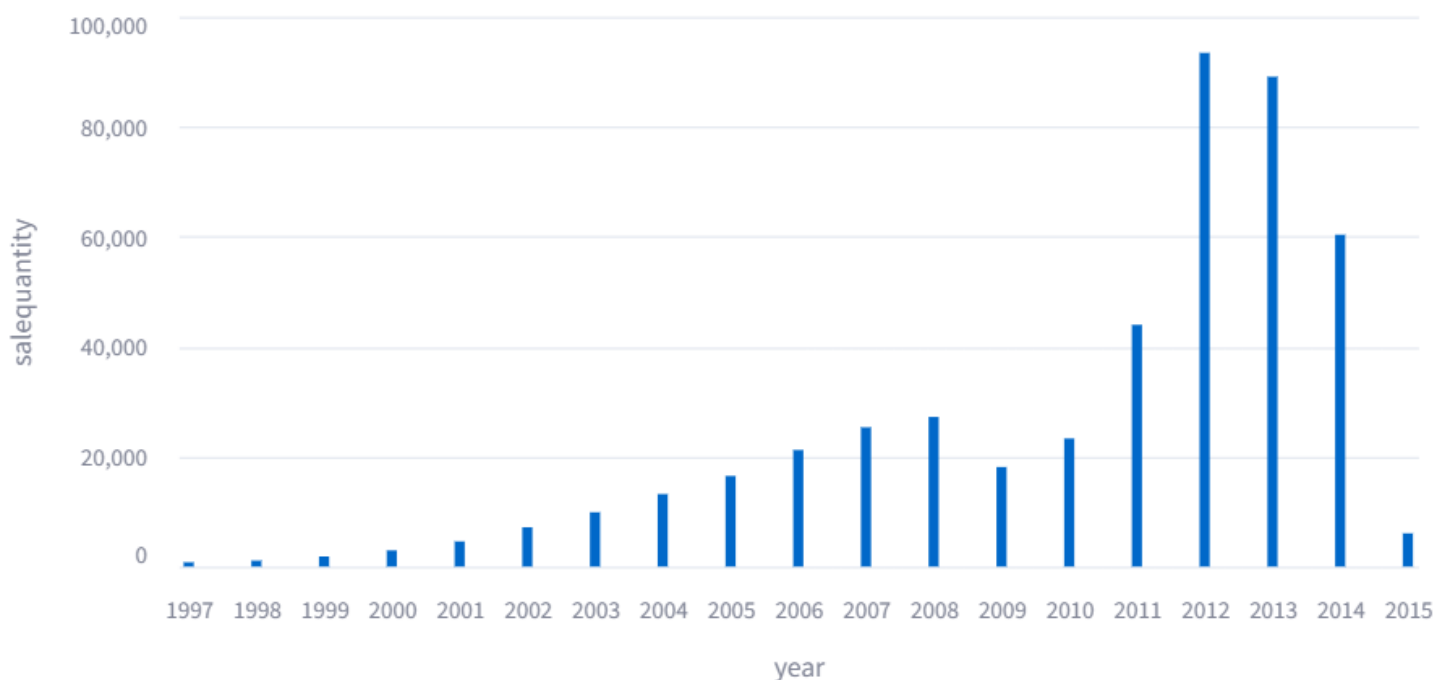


Figure 83 The number of cars sold over the years.

Although there was a sharp decline in 2015, the overall trend was still an increase in the number of cars sold over the years. This could reflect an increase in demand for cars or a growth in the car market. However, in 2015, the number of cars sold decreased rapidly. Combined with the chart seen earlier, it is possible that part of the reason was that the selling price in 2015 was too high compared to previous years. This shows the relationship between selling price and the number of cars sold, where too high a price can reduce demand.

From about 2010 to 2014, the number of cars sold increased sharply, which could reflect a period of economic growth or an increase in demand for cars during this period.

6 Implement a detailed test plan on a data analysis software tool. P6

6.1 Prepare a test plan (Test Plan)

6.1.1 Introduction to Test Plan.

Test Plan is a detailed document that describes the strategy, objectives, scope, methodology, and schedule of software testing activities. It helps ensure that all aspects of the testing process are considered and performed systematically. (Huong, 2018)

Here are the main steps to creating a Test Plan:

Define Test Objectives: The testing objectives are what you want to achieve through the testing process. These objectives may include:

- Identify defects: Detect and document errors in the software.
- Ensure correctness: Ensure that the software functions correctly according to the defined requirements.
- Compatibility testing: Ensure that new features do not affect existing functionality.
- Performance evaluation: Evaluate the performance of the software under different conditions.

Define the Scope of Testing: The scope of testing defines what will be tested and what will not be tested. This helps focus on the most important areas of the software and avoid wasting resources on unnecessary areas. Factors to consider include:

- Key features: Identify the key features of the software that need to be tested.
- Secondary Features: Identify secondary features that can be tested if sufficient time and resources are available.
- Constraints: Identify the limitations of testing, such as features or areas that are not in scope for testing.

Identify the Types of Tests to be Performed: Types of testing may include:

- Functional testing: Ensures that the software functions according to the functional requirements.
- Performance testing: Evaluates the performance of the software under different conditions, such as high load or long duration.

- Security testing: Ensures that the software is safe from security threats.
- Usability testing: Evaluate the ease of use of the software for end users.
- Integration testing: Ensures that the different components of the software work well together.

Identify the Necessary Resources:

- Human Resources: Identify the members of the testing team and their roles.
- Hardware: Identify the equipment needed to perform testing, such as computers, servers, and mobile devices.
- Software: Identify the necessary testing tools, such as test automation tools and test environments.
- Test Environments: Identify the necessary testing environments, such as development environments, test environments, and production environments.

Schedule Testing: The testing schedule defines the start and end times of testing activities. It also includes milestones and contingency time for unforeseen activities. Factors to consider include:

- Testing Time: Identify the time required to perform different types of testing.
- Milestones: Identify important milestones in the testing process, such as completion of functional testing or performance testing.
- Contingency Time: Allow time for unforeseen activities, such as finding bugs or changing requirements.

A detailed and comprehensive Test Plan helps ensure that all aspects of the testing process are considered and performed systematically. This not only helps in early detection and correction of bugs but also ensures that the software meets the user's requirements and expectations.

6.1.2 Test Plan for Car Price Data Analysis Project

Identify Test Objectives

- Objective 1: Ensure that the initial data loading and display functions work correctly.
- Objective 2: Verify the data cleaning process, including handling duplicate data, missing values, and removing unnecessary columns.
- Objective 3: Verify that data transformations (e.g., date conversions) are correct.
- Objective 4: Verify the correctness of outlier detection and removal.
- Objective 5: Ensure that statistical calculations and visualizations (scatter plots, pie charts, line charts, bar charts) are correct and displayed correctly.
- Objective 6: Verify the overall functionality and integration of the Streamlit application.

Define the Scope of Testing

- In-Scope:
 - Loading and displaying the dataset.
 - Data cleaning operations: checking for duplicates, handling missing values, and dropping columns.
 - Data transformations: date conversions.
 - Outlier detection and removal processes.
 - Statistical calculations (mean, median, standard deviation).
 - Visualizations: scatter plots, pie charts, line charts, bar charts.
 - Streamlit application's UI/UX components and interactivity.
- Out-of-Scope:
 - Testing external dependencies and data sources outside the provided dataset.
 - Performance testing under high load conditions.

Identify the Types of Tests to be Performed:

- Unit Tests: Verify the correctness of individual functions for data cleaning, transformations, and calculations.
- Integration Tests: Ensure that different modules of the application (data loading, cleaning, visualization) work together seamlessly.
- Functional Tests: Test the overall functionality of the application from data loading to visualization.
- UI/UX Tests: Validate the user interface for usability, responsiveness, and correctness of displayed information.
- Regression Tests: Ensure that new changes do not break existing functionality.

Identify the Necessary Resources

- Personnel:
 - Data Analyst/Engineer for creating and executing test cases.
 - QA Engineer for validating the application.
 - Developer for addressing issues found during testing.
- Tools and Software:
 - Streamlit for running the application.
 - Pandas, NumPy, Seaborn, Matplotlib for data handling and visualization.
 - Pytest or Unittest for writing test cases.

Schedule Testing:

- Day 1:
 - Define test cases for data loading and initial display.
 - Execute unit tests for data cleaning functions.
 - Validate date conversion functions.
- Day 2:
 - Perform integration tests for data cleaning and transformations.
 - Execute functional tests for the outlier detection and removal process.
 - Validate statistical calculation functions.
- Day 3:
 - Conduct UI/UX testing for the Streamlit application.
 - Validate all visualizations (scatter plots, pie charts, line charts, bar charts).
 - Perform regression tests to ensure existing functionality is intact.
- Day 4:
 - Address any issues found during testing.
 - Conduct the final round of testing for the entire application.
 - Prepare and submit the test report.

6.2 Write detailed test cases.

Table 3 Test Case

Test Case ID	Description	Precondition	Step Details	Input data	Expected result
TC01	Verify data loading and initial display.	The dataset file (car_prices.csv) is available in the working directory.	1. Load the dataset using <code>pd.read_csv('car_prices.csv')</code> . 2. Display the dataset using <code>st.write(df)</code> .	car_prices.csv	The dataset should be loaded and displayed correctly in Streamlit.
TC02	Check for duplicates in the dataset	The dataset is loaded into a DataFrame.	1. Count duplicate rows using <code>df.duplicated().sum()</code> . 2. Drop duplicates using <code>df.drop_duplicates()</code> .	car_prices.csv	The duplicate count before dropping should match the expected

			3. Verify no duplicates remain using <code>df.duplicated().sum()</code> .		duplicates, and the count after dropping should be zero.
TC03	Handle missing values in the 'transmission' column.	The dataset is loaded, and the 'vin' column is dropped.	1. Fill missing values in the 'transmission' column with the mode value. 2. Verify no missing values remain in the 'transmission' column.	car_prices.csv	The 'transmission' column should have no missing values after filling.
TC04	Drop rows with any remaining missing values.	The dataset is loaded, and missing values in the 'transmission' column are filled.	1. Drop rows with any remaining missing values using <code>df.dropna()</code> . 2. Verify no missing values remain in any column.	car_prices.csv	The DataFrame should have no missing values in any column.
TC05	Convert 'year' and 'saledate' columns to datetime.	The dataset is loaded, and all missing values are handled.	1. Convert 'year' to datetime using <code>pd.to_datetime(df['year'], format='%Y')</code> . 2. Convert 'saledate' to datetime and keep only the date part using <code>pd.to_datetime(df['saledate'], utc=True, format='mixed').dt.date</code> . 3. Verify the conversions are correct.	car_prices.csv	The 'year' and 'saledate' columns should be correctly converted to datetime format.

TC06	Filter outliers based on 'sellingprice' and 'condition'.	The dataset is loaded, and all columns are correctly formatted.	<ol style="list-style-type: none"> 1. Filter out rows with 'sellingprice' > 175000 and 'condition' <= 10. 2. Verify the remaining data has no outliers based on the specified criteria. 	car_prices.csv	The DataFrame should have no outliers based on the specified criteria.
TC07	Verify statistical calculations for 'sellingprice'.	The dataset is loaded and cleaned.	<ol style="list-style-type: none"> 1. Calculate the mean, median, and standard deviation of 'sellingprice'. 2. Verify the calculations are correct. 	Mean, Median and Standard Deviation values of the "sellingprice" column	The calculated mean, median, and standard deviation should match expected values.
TC08	Verify visualizations (scatter plots, pie charts, line charts, bar charts).	The dataset is loaded and cleaned.	<ol style="list-style-type: none"> 1. Create and display scatter plots, pie charts, line charts, and bar charts as specified. 2. Verify the visualizations are rendered correctly and match the expected results. 	Display plots as specified.	All visualizations should be displayed correctly in Streamlit.

7 Present the results of the analysis on the chosen data set. P7

I will use Pytest to run the code tests for the test cases listed above.

Pytest is a popular testing framework for Python used to write simple as well as scalable test cases for applications and libraries. It supports a wide range of testing needs, from simple unit tests to complex functional tests. Pytest is known for its ease of use, powerful features, and ability to handle complex test scenarios with minimal code.

Pytest will automatically discover test functions, run them, and provide reports on the results.

7.1 TC01

```
test_car_price_analysis.py > ...
1 import pandas as pd
2 import pytest
3 import streamlit as st
4 from datetime import datetime
5
6 # Define test functions
7 def test_data_loading():
8     df = pd.read_csv('car_prices.csv')
9     assert df.shape[0] > 0, "Data loading failed"
10    st.write(df)
11
12
13 # Run tests
14 if __name__ == "__main__":
15     pytest.main()
16
17 #pytest test car price analysis.py
18 #streamlit run test_car_price_analysis.py
```

test car price analysis.py . [100%]

----- warnings summary -----

test car price analysis.py: 16 warnings

C:\Users\hoang\anaconda3\envs\pythndataanalysis\lib\site-packages\pyarrow\pandas_compat.py:373: DeprecationWarning: is_sparse is deprecated and will be removed in a future version. Check 'isinstance(dtype, pd.SparseDtype)' instead.

if pandas.api.is_sparse(col):

-- Docs: <https://docs.pytest.org/en/stable/how-to/capture-warnings.html>

1 passed, 16 warnings in 0.13s

Figure 84 Run test TC01.

	year	make	model	trim	body	transmission	vin	state	conditi
0	2,015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg566472	ca	
1	2,015	Kia	Sorento	LX	SUV	automatic	5xyktca69fg561319	ca	
2	2,014	BMW	3 Series	328i SULEV	Sedan	automatic	wba3c1c51ek116351	ca	
3	2,015	Volvo	S60	T5	Sedan	automatic	yv1612tb4f1310987	ca	
4	2,014	BMW	6 Series G	650i	Sedan	automatic	wba6b2c57ed129731	ca	
5	2,015	Nissan	Altima	2.5 S	Sedan	automatic	1n4al3ap1fn326013	ca	
6	2,014	BMW	M5	Base	Sedan	automatic	wbsfv9c51ed593089	ca	
7	2,014	Chevrolet	Cruze	1LT	Sedan	automatic	1g1pc5sb2e7128460	ca	
8	2,014	Audi	A4	2.0T Premiur	Sedan	automatic	wauffaf13en030343	ca	
9	2,014	Chevrolet	Camaro	LT	Convert	automatic	2g1fb3d37e9218789	ca	

Figure 85 The test result TC01.

The test result is the same as the expected result: pass.

7.2 TC02.

```
test_car_price_analysis.py > ...
1 import pandas as pd
2 import pytest
3 import streamlit as st
4 from datetime import datetime
5
6 def test_check_duplicates():
7     df = pd.read_csv('car_prices.csv')
8     initial_duplicates = df.duplicated().sum()
9     df = df.drop_duplicates()
10    final_duplicates = df.duplicated().sum()
11    assert final_duplicates == 0, f"Duplicate data handling failed. Initial: {initial_duplicates}, After removal: {final_duplicates}"
12    st.write(f"Initial duplicates: {initial_duplicates}, After removal: {final_duplicates}")
13
14
15 # Run tests
16 if __name__ == "__main__":
17     pytest.main()
18 #pytest test_car_price_analysis.py
19 #streamlit run test_car_price_analysis.py
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python + - [] [x] [y] [z]

Stopping...

PS C:\Users\hoang\OneDrive\Tài liệu> Advanced Programming for Data Analysis\streamlit> **pytest** test_car_price_analysis.py

test session starts

platform win32 -- Python 3.9.18, pytest-7.4.4, pluggy-1.0.0
rootdir: C:\Users\hoang\OneDrive\Tài liệu> Advanced Programming for Data Analysis\streamlit
plugins: anyio-4.2.0
collected 1 item

test_car_price_analysis.py .

1 passed in 4.52s

Figure 86 Run test TC02.

Initial duplicates: 1, After removal: 0

Figure 87 The test result TC02.

The test result is the same as the expected result: pass.

7.3 TC03

```
test_car_price_analysis.py > --
1  import pandas as pd
2  import pytest
3  import streamlit as st
4  from datetime import datetime
5
6  def test_handle_missing_values():
7      df = pd.read_csv('car_prices.csv').drop(columns=['vin'])
8      mode_value = df['transmission'].mode()[0]
9      df['transmission'].fillna(mode_value, inplace=True)
10     assert df['transmission'].isna().sum() == 0, "Missing values in 'transmission' were not filled correctly"
11     st.write(df.isna().sum())
12
13 # Run tests
14 if __name__ == "__main__":
15     |   pytest.main()
16     @pytest test_car_price_analysis.py
17     @streamlit run test_car_price_analysis.py
```

PROBLEMS

OUTPUT

DEBUG CONSOLE

TERMINAL

PORTS

df['transmission'].fillna(mode_value, inplace=True)

test_car_price_analysis.py::test_handle_missing_values

C:\Users\hoang\anaconda3\envs\pythnDataAnalysis\lib\site-packages\pyarrow\pandas_compat.py:373: DeprecationWarning: is_sparse is deprecated and will be removed in a future version. Check "isinstance(dtype, pd.SparseDtype)" instead.

if _pandas_api.is_sparse(col):

-- Docs: <https://docs.pytest.org/en/stable/how-to/capture-warnings.html>

----- 1 passed, 2 warnings in 2.41s -----

Figure 88 Run test TC03.

	0
year	0
make	10,301
model	10,399
trim	10,651
body	13,195
transmission	0
state	0
condition	11,820
odometer	94
color	749

Figure 89 The test result TC03.

The test result is the same as the expected result: pass.

7.4 TC04

```
test car price analysis.py > ...
1 import pandas as pd
2 import pytest
3 import streamlit as st
4 from datetime import datetime
5
6 def test_drop_remaining_missing_values():
7     df = pd.read_csv('car_prices.csv').drop(columns=['vin'])
8     mode_value = df['transmission'].mode()[0]
9     df['transmission'].fillna(mode_value, inplace=True)
10    df.dropna(inplace=True)
11    assert df.isna().sum().sum() == 0, "Remaining missing values were not dropped"
12    st.write(df.isna().sum())
13
14
15 # Run tests
16 if __name__ == "__main__":
17     pytest.main()
18 #pytest test_car_price_analysis.py
19 #streamlit run test_car_price_analysis.py
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS Python + - [] [] ... ^ X

```
df['transmission'].fillna(mode_value, inplace=True)
test_car_price_analysis.py::test_drop_remaining_missing_values
C:\Users\hoang\anaconda3\envs\pythonDataAnalysis\lib\site-packages\pyarrow\pandas_compat.py:373: DeprecationWarning: is_sparse is deprecated and will be removed in a
future version. Check 'isinstance(dtype, pd.SparseDtype)' instead.
  if _pandas_api.is_sparse(col):
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
1 passed, 2 warnings in 2.74s
```

Figure 90 Run test TC04.

	0
year	0
make	0
model	0
trim	0
body	0
transmission	0
state	0
condition	0
odometer	0
color	0

Figure 91 The test result TC04.

The test result is the same as the expected result: pass.

7.5 TC05

```

test_car_price_analysis.py > ...
1 import pandas as pd
2 import pytest
3 import streamlit as st
4 from datetime import datetime
5
6 def test_convert_datetime():
7     df = pd.read_csv('car_prices.csv')
8     # Remove invalid year values
9     current_year = datetime.now().year
10    df = df[(df['year'] >= 1886) & (df['year'] <= current_year)]
11
12    df['year'] = pd.to_datetime(df['year'], format='%Y')
13    df['saledate'] = pd.to_datetime(df['saledate'], utc=True, errors='coerce').dt.date
14    assert pd.api.types.is_datetime64_any_dtype(df['year']), "'year' was not converted to datetime"
15    assert pd.api.types.is_object_dtype(df['saledate']), "'saledate' was not converted to date"
16    st.write(df)
17
18
19
20 # Run tests
21 if __name__ == "__main__":
22     pytest.main()
23
24 #test_car_price_analysis.py
25

```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

df['transmission'].fillna(mode_value, inplace=True)

test_car_price_analysis.py::test_drop_remaining_missing_values

C:\Users\hoang\anaconda3\envs\pythondatalanalysis\lib\site-packages\pyarrow\pandas_compat.py:373: DeprecationWarning: is_sparse is deprecated and will be removed in a future version. Check 'isinstance(dtype, pd.SparseDtype)' instead.

if _pandas_api.is_sparse(col):

-- Docs: <https://docs.pytest.org/en/stable/how-to/capture-warnings.html>

----- 1 passed, 2 warnings in 2.71s -----

Figure 92 Run test TC05.

	year	make		mmr	sellingprice	saledate
0	2015-01-01 00:00:00	Kia		20,500	21,500	2014-12-16
1	2015-01-01 00:00:00	Kia		20,800	21,500	2014-12-16
2	2014-01-01 00:00:00	BMW	ise)	31,900	30,000	2015-01-14
3	2015-01-01 00:00:00	Volvo		27,500	27,750	2015-01-28
4	2014-01-01 00:00:00	BMW	ise)	66,000	67,000	2014-12-18
5	2015-01-01 00:00:00	Nissan	rental / tulsa	15,350	10,900	2014-12-30
6	2014-01-01 00:00:00	BMW		69,000	65,000	2014-12-17
7	2014-01-01 00:00:00	Chevrolet	rental / tulsa	11,900	9,800	2014-12-16
8	2014-01-01 00:00:00	Audi		32,100	32,250	2014-12-18
9	2014-01-01 00:00:00	Chevrolet		26,300	17,500	2015-01-19

Figure 93 The test result TC05

The test result is the same as the expected result: pass.

7.6 TC06

```

test_car_price_analysis.py > ...
1  import pandas as pd
2  import pytest
3  import streamlit as st
4  from datetime import datetime
5
6  def test_filter_outliers():
7      df = pd.read_csv('car_prices.csv')
8      df = df.loc[(df['sellingprice'] <= 175000)]
9      df = df.loc[(df['condition'] > 10)]
10     assert df['sellingprice'].max() <= 175000, "Outlier values in 'sellingprice' were not filtered"
11     assert df['condition'].min() > 10, "Outlier values in 'condition' were not filtered"
12     st.write(df)
13
14
15
16
17 # Run tests
18 if __name__ == "__main__":
19     pytest.main()
20 #pytest test_car_price_analysis.py
21 #streamlit run test_car_price_analysis.py

```

test_car_price_analysis.py . [100%]

----- warnings summary -----

test_car_price_analysis.py: 16 warnings
 C:\Users\hoang\anaconda3\envs\pythonDataAnalysis\lib\site-packages\pyarrow\pandas_compat.py:373: DeprecationWarning: is_sparse is deprecated and will be removed in a future version. check 'isinstance(dtype, pd.SparseDtype)' instead.
 if _pandas_api.is_sparse(col):

-- Docs: <https://docs.pytest.org/en/stable/how-to/capture-warnings.html>

----- 1 passed, 16 warnings in 2.95s -----

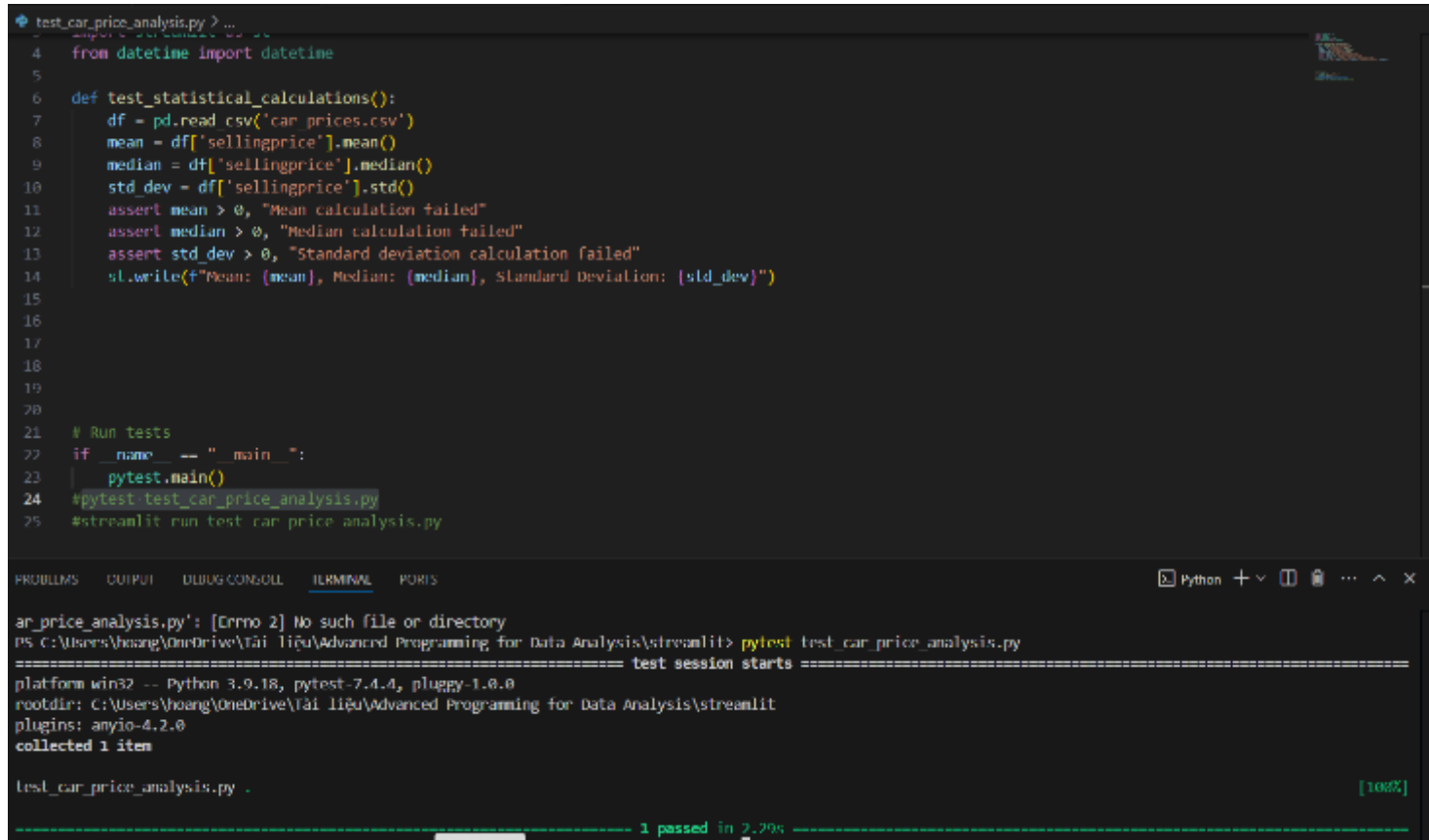
Figure 94 Run test TC06.

	condition	odometer	color	interior	seller	mmr	sellingprice
2	45	1,331	gray	black	financial services remarketing (lease)	31,900	30,000
3	41	14,282	white	black	volvo na rep/world omni	27,500	27,750
4	43	2,641	gray	black	financial services remarketing (lease)	66,000	67,000
6	34	14,943	black	black	the hertz corporation	69,000	65,000
8	42	9,557	white	black	audi mission viejo	32,100	32,250
10	48	14,414	black	black	desert auto trade	47,300	49,750
11	48	2,034	red	tan	kia motors finance	15,150	17,700
17	49	7,983	white	black	audi north scottsdale	37,100	40,000
18	17	13,441	black	black	wellis fargo dealer services	17,750	17,000
19	34	8,819	black	black	the hertz corporation	68,000	67,200

Figure 95 The test result TC06.

The test result is the same as the expected result: pass.

7.7 TC07



```

test_car_price_analysis.py > ...
4 from datetime import datetime
5
6 def test_statistical_calculations():
7     df = pd.read_csv('car_prices.csv')
8     mean = df['sellingprice'].mean()
9     median = df['sellingprice'].median()
10    std_dev = df['sellingprice'].std()
11    assert mean > 0, "Mean calculation failed"
12    assert median > 0, "Median calculation failed"
13    assert std_dev > 0, "Standard deviation calculation failed"
14    sl.write(f"Mean: {mean}, Median: {median}, Standard Deviation: {std_dev}")
15
16
17
18
19
20
21 # Run tests
22 if __name__ == "__main__":
23     pytest.main()
24 #pytest test_car_price_analysis.py
25 #streamlit run test_car_price_analysis.py

```

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + - - - - -
ar_price_analysis.py: [Errno 2] No such file or directory
PS C:\Users\hoang\OneDrive\Tài liệu\Advanced Programming for Data Analysis\streamlit> pytest test_car_price_analysis.py
===== test session starts =====
platform win32 -- Python 3.9.18, pytest-7.4.4, pluggy-1.0.0
rootdir: C:\Users\hoang\OneDrive\Tài liệu\Advanced Programming for Data Analysis\streamlit
plugins: anyio-4.2.0
collected 1 item

test_car_price_analysis.py . [100%]

===== 1 passed in 2.29s =====

```

Figure 96 Run test TC07.

Mean: 13611.372926456535, Median: 12100.0, Standard Deviation: 9749.498615651328

Figure 97 The test result TC07

The test result is the same as the expected result: pass.

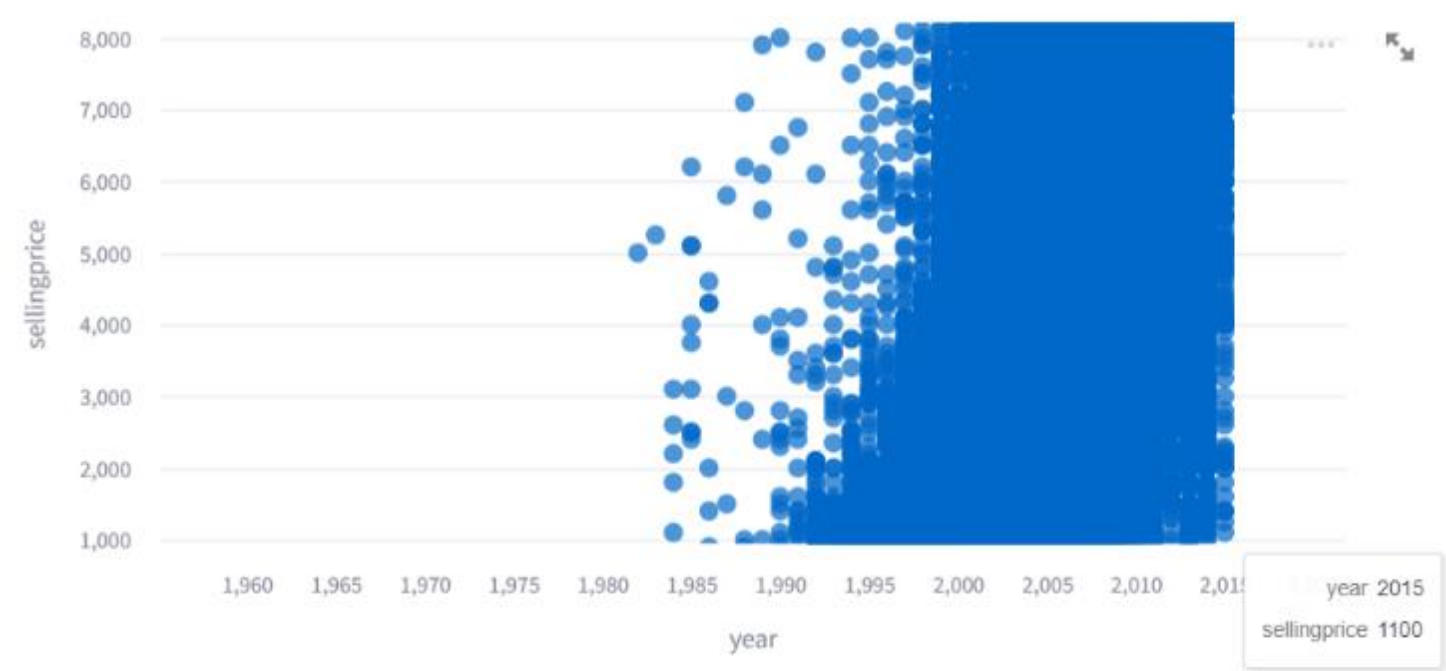
7.8 TC08

```
test_car_price_analysis.py > ...
1 import pandas as pd
2 import pytest
3 import streamlit as st
4 from datetime import datetime
5
6 def test_visualizations():
7     df = pd.read_csv('car_prices.csv')
8     st.scatter_chart(df, x="year", y="sellingprice")
9     st.write("Scatter plot created.")
10    st.bar_chart(df, x="year", y="sellingprice")
11    st.write("Bar chart created.")
12    st.line_chart(df, x="saledate", y="sellingprice")
13    st.write("Line chart created.")
14
15
16 # Run Tests
17 if __name__ == "__main__":
18     pytest.main()
19 #pytest test_car_price_analysis.py
20 #streamlit run test_car_price_analysis.py
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS Python + - [] [] ... ^ x

```
test_car_price_analysis.py::test_visualizations
test_car_price_analysis.py::test_visualizations
test_car_price_analysis.py::test_visualizations
test_car_price_analysis.py::test_visualizations
test_car_price_analysis.py::test_visualizations
C:\Users\hoang\anaconda3\envs\pythndata\Analysis\lib\site-packages\pandas\_compat.py:373: DeprecationWarning: is_sparse is deprecated and will be removed in a
future version. Check 'isinstance(dtype, pd.SparseDtype)' instead.
  if _pandas_api.is_sparse(col):
-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 1 passed, 126 warnings in 3.49s =====
```

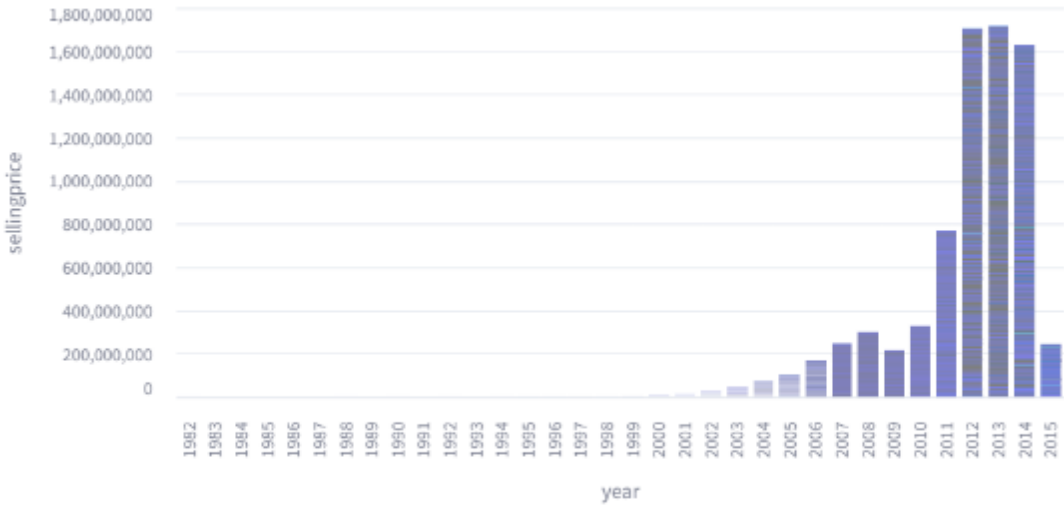
Figure 98 Run test TC08.



Scatter plot created.

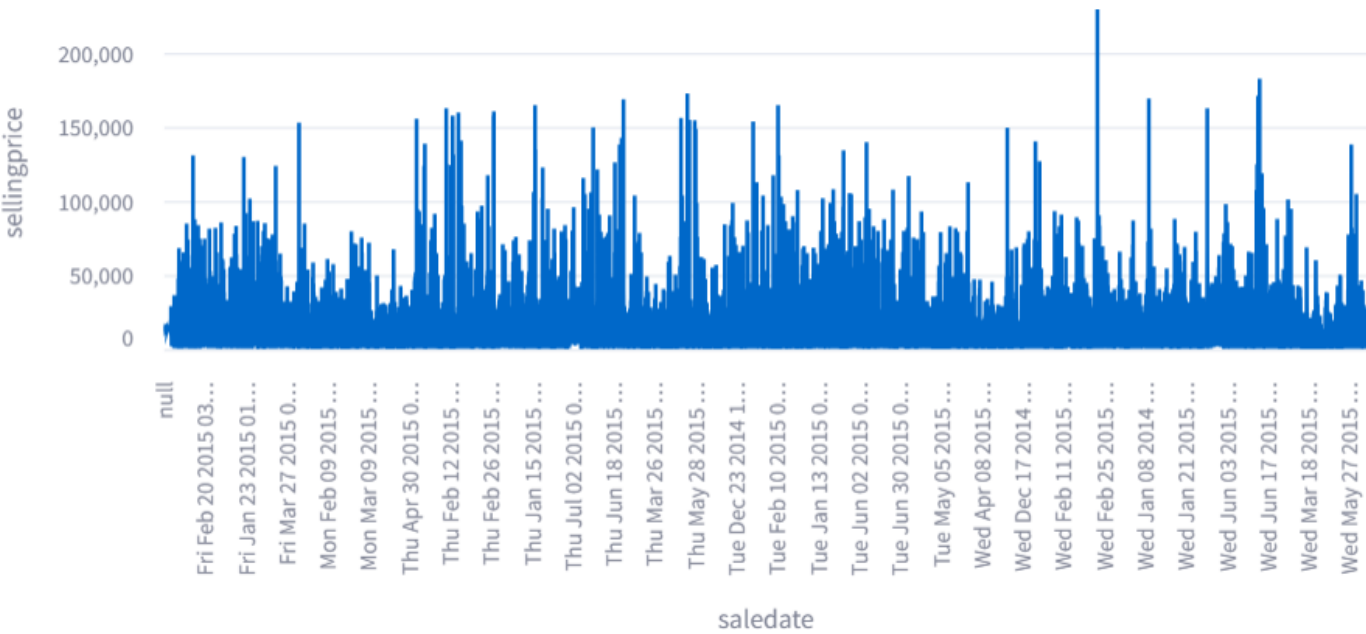
Figure 99 The test result TC08.

Scatter plot created.



Bar chart created.

Figure 100 The test result TC08.



Line chart created.

Figure 101The test result TC08.

The test result is the same as the expected result: pass.

7.9 Conduct the final round of testing for the entire application.

```
PS C:\Users\hoang\OneDrive\Tài liệu\Advanced Programming for Data Analysis\streamlit> pytest test_car_price_analysis.py
===== test session starts =====
platform win32 -- Python 3.9.18, pytest-7.4.4, pluggy-1.0.0
rootdir: C:\Users\hoang\OneDrive\Tài liệu\Advanced Programming for Data Analysis\streamlit
plugins: anyio-4.2.0
collected 8 items

test_car_price_analysis.py ..... [100%]

===== warnings summary =====
test_car_price_analysis.py: 56 warnings
  C:\Users\hoang\anaconda3\envs\pythnDataAnalysis\lib\site-packages\pyarrow\pandas_compat.py:373: DeprecationWarning: is_sparse is deprecated and will be removed in a
future version. Check `isinstance(dtype, pd.SparseDtype)` instead.
    if _pandas_api.is_sparse(col):

test_car_price_analysis.py::test_handle_missing_values
  C:\Users\hoang\OneDrive\Tài liệu\Advanced Programming for Data Analysis\streamlit\test_car_price_analysis.py:24: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
  The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

  For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

    df['transmission'].fillna(mode_value, inplace=True)

test_car_price_analysis.py::test_drop_remaining_missing_values
  C:\Users\hoang\OneDrive\Tài liệu\Advanced Programming for Data Analysis\streamlit\test_car_price_analysis.py:31: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through chained assignment using an inplace method.
  The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we are setting values always behaves as a copy.

  For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform the operation inplace on the original object.

    df['transmission'].fillna(mode_value, inplace=True)

test_car_price_analysis.py::test_convert_datetime
  C:\Users\hoang\OneDrive\Tài liệu\Advanced Programming for Data Analysis\streamlit\test_car_price_analysis.py:43: UserWarning: Could not infer format, so each element will be parsed individually, falling back to 'dateutil'. To ensure parsing is consistent and as-expected, please specify a format.
    df['saledate'] = pd.to_datetime(df['saledate'], utc=True, errors='coerce').dt.date
```

Figure 102 Run a test for the entire application.

```
test_car_price_analysis.py: 99 warnings
  C:\Users\hoang\anaconda3\envs\pythnDataAnalysis\lib\site-packages\altair\utils\schemapi.py:353: DeprecationWarning: jsonschema.RefResolver is deprecated as of v4.18.0, in favor of the https://github.com/python-jsonschema/referencing library, which provides more compliant referencing behavior as well as more flexible APIs for customization. A future release will remove RefResolver. Please file a feature request (on referencing) if you are missing an API for the kind of customization you need.
    resolver = jsonschema.RefResolver.from_schema(root or schema)

test_car_price_analysis.py: 21 warnings
  C:\Users\hoang\anaconda3\envs\pythnDataAnalysis\lib\site-packages\altair\utils\schemapi.py:118: DeprecationWarning: jsonschema.RefResolver is deprecated as of v4.18.0, in favor of the https://github.com/python-jsonschema/referencing library, which provides more compliant referencing behavior as well as more flexible APIs for customization. A future release will remove RefResolver. Please file a feature request (on referencing) if you are missing an API for the kind of customization you need.
    resolver = jsonschema.RefResolver.from_schema(root_schema)

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 8 passed, 179 warnings in 16.77s =====
```

Figure 103 Run a test for the entire application.

All tests were collected and run successfully, with no errors in the test cases.

All tests passed. This is a good sign that the code is working as expected.

Conclusion.

This assignment provides a comprehensive exploration of data analytics, underscoring its pivotal role in transforming raw data into actionable insights that significantly impact decision-making across various sectors. By investigating Python and its libraries, we gain valuable skills in data manipulation and analysis, essential for

real-world applications. The process of analyzing large datasets, from data collection and cleaning to testing and validation, highlights the critical steps necessary to ensure data quality and reliability. Ultimately, this assignment not only enhances our understanding of data analytics but also equips us with the tools to drive informed decision-making and foster innovation across diverse fields.

References

Anwar, S., 2024. *Vehicle Sales Data*. [Online]
Available at: <https://www.kaggle.com/datasets/syednwarafriadi/vehicle-sales-data>
[Accessed 21 6 2024].

Campbell, S., 2024. *Python Tutorial for Beginners: Learn Programming Basics*. [Online]
Available at: <https://www.guru99.com/python-tutorials.html>
[Accessed 21 6 2024].

Galarnyk, M., 2020. *nstall Python/Anaconda on Windows (2020)*. [Online]
Available at: https://ichi.pro/windows-ni-python-anaconda-o-insuto-ru-suru-2020-273647549392445#google_vignette
[Accessed 21 6 2024].

Glints, 2022. *Python Là Gì? Tất Tần Tật Về Ngôn Ngữ Lập Trình Python*. [Online]
Available at: <https://glints.com/vn/blog/ngon-ngu-lap-trinh-python-la-gi/>
[Accessed 21 6 2024].

Huong, V. T. T., 2018. *The concept of a test plan and questions to ask when creating a test plan*. [Online]
Available at: <https://viblo.asia/p/khai-niem-test-plan-va-nhung-cau-hoi-can-dat-khi-tao-test-plan-1Je5EE145nL>
[Accessed 7 8 2024].

Huy, T. V., 2021. *Introduction to Streamlit*. [Online]
Available at: <https://huytranvan2010.github.io/Introduction-to-streamlit/>
[Accessed 6 8 2024].

Ichi.pro, n.d. *Streamlit - Revolutionizing Data Application Creation*. [Online]
Available at: <https://ichi.pro/vi/streamlit-cach-mang-hoa-viec-tao-ung-dung-du-lieu-248940993548562>
[Accessed 6 8 2024].

McKinney, W., 2022. *Python for Data Analysis*. 3rd ed. s.l.:O'Reilly Media.

MyGPT, 2023. *How to Use Streamlit Library in Python - MyGPT*. [Online]
Available at: <https://mygpt.vn/huong-dan-su-dung-thu-vien-streamlit-trong-python/>
[Accessed 6 8 2024].

Nhan, N., 2019. *Streamlit: A Dedicated Tool for Data Analytics Applications with Python*. [Online] Available at: <https://fullstackstation.com/gioi-thieu-streamlit-la-gi/> [Accessed 6 8 2024].