VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi – 590 018.

An Internship Report

On

"Machine learning using Python"

01.02.2023 to 05.03.2023

Submitted in partial fulfilment of the for the award of the degree of

Bachelor of Engineering

In

Computer Science and Engineering

Submitted by

PRAKHAR KUMAR CHANDRAKER   **1VI19CS070**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

VEMANA INSTITUTE OF TECHNOLOGY

BENGALURU – 560034

2022-2023

VEMANA INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University, Belagavi)

Koramangala, Bengaluru-560034.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that the Internship work entitled "**MACHINE LEARNING USING PYTHON**" is a bonafide work carried out by **Mr. PRAKHAR KUMAR CHANDRAKER (1VI19CS070)** during the academic year 2022-23 in partial fulfilment of the requirement for the award of **Bachelor of Engineering in Computer Science and Engineering** of the **Visvesvaraya Technological University, Belagavi.** It is certified that all corrections/suggestions indicated for internal assessment have been incorporated in the report. The internship report has been approved as it satisfies the academic requirements in respect of the Internship prescribed for the said degree.

Guide	**Head of the Department**	**Principal**
Ms. Veena G	Dr. M. Ramakrishna	Dr. Vijayasimha Reddy B.G

External Viva

Name of the Examiner   Signature with date

1._____

                                                                    _____

2._____

                                                                    _____

Internship Certificate

Anspro Technologies
Find a better solution

Internship Completion Letter

This is to certify that **Mr. Prakhar Kumar Chandraker,** USN **1VI19CS070**, Vemana Institute of Technology**,** Computer Science and Engineering, was working with us from 01-02-2023 to 05-03-2023**.** He was working as Intern on Machine Learning using Python. He has worked on "**LPG Usage Prediction**" project.

During the internship, intern was found to be good and disciplined.

Authorized Signatory

Yours faithfully,

For Anspro Technologies
Authorized Signatory

ABSTRACT

Anspro Technologies is a software development company with vast industry experience in various domains, services & product lines. They have great experience and expertise in developing various software solutions. provides cost-effective, convenient and easy-to-manage software solutions services based on web designing, web application development & web hosting requirements. They provide e-learning applications, which have advanced features and rich graphical interface which is best suited for educational institutions, training institutes for providing distant education and training purposes.

ACKNOWLEDGEMENT

LIST OF FIGURES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In India, the supply of LPG through pipelines is not possible due to shortage of LPG. As technology being improved many gas agencies or distributors have implemented IVRS these days although due to daily busy schedules, customer finds very difficult to book new cylinder. So, our proposal is to completely automate the process of refill booking without human intervention that accordingly will help consumer against foul play. Based on the usages of lpg cylinder in domestic and number of members stays in the house, data has been recorded based on every time the customer refills the lpg cylinder. Our aim is to predict the next refilling date of each family.

1.1 Project Objective

The main objective of the project is to predict the next LPG refilling date based on the previous uses.

1.2 Project Scope

The scope of the company is to provide internships, workshops, career growing opportunities to the learners. Since the application dynamic and made available 24*7, it is easily accessible, convenient, and time saving for the users.

1.3 Problem Statement

- The existing system is basically a manual system, where if the user needs cylinders then he needs to contact the seller and book them.

- While cooking cylinder gets over.

- Many family use single cylinder so for them it's very difficult when cylinder gets over.

CHAPTER 2

ORGANIZATION PROFILE

Anspro Technologies is a Bangalore based software development company with vast industry experience in various domains, services & product lines. They utilize their vast experience and expertise in developing various software solutions in accordance with client's business and job requirements.

Anspro Technologies provides cost-effective, convenient and easy-to-manage software solutions services. They serve clients with cost efficient software applications that helps them to grow their business. Anspro Technologies offers one stop solutions to their customers for their web designing, web application development & web hosting requirements. They provide e-learning applications, which have advanced features and rich graphical interface. Their e-learning applications are best suited for educational institutions, training institutes for providing distant education and training purposes. Their customized billing and accounting tool is suitable for any departmental stores as well as for small scale industries.

They believe that technologies and ideas, are more than anything else to challenge the world and grow. However, it's not only the technologies they use, but how they integrate them, that counts. They understand that to integrate technologies requires the right people, and they have specialist individual teams of experts on board who provide pre-sales, post sales, project implementation and support.

In order to sustain the productivity of any organization, it is necessary to automate the biometric time attendance system management. This is because manual attendance punching and related calculation leads to time and cost consumption.

Many of their clients request comprehensive maintenance contracts with Anspro Technologies to ensure that their critical networks (hardware or software) are supported. These contracts range from traditional warranty support, to maintenance during office hours, to 24X7 critical support.

CHAPTER 3

SYSTEM ANALYSIS

System analysis is the process of gathering and interpreting facts, diagnosing problems and using the information to recommend Improvements on the system. System analysis is a problem solving Activity that requires intensive communication between the system users and system developers. System analysis or study is an important phase of any system Development process.

The system is viewed as a whole, the inputs are identified and the system is subjected to close study to identify the Problem areas. The solutions are given as a proposal. The proposal is reviewed on user request and suitable changes are made. This loop ends as soon as the user is satisfied with the proposal.

3.1 Existing System

Existing system of system of project management is manual. Project coordinator or guide gives task for student manually. Student complete the work which is given by coordinator or guide and submits manually, in this system all work is done by manually so it can take more time to complete project related work. Project coordinator or guide requires remembering in mind when student completed the work so it is difficult for Project coordinator or guide which student completed the task and when. In the existing system does not help users to get right information at right time and user cannot manage project development easily to achieve the main goal.

Limitations of existing system

1. It is time consuming.

2. Right information is not retrieved at right time.

3. Any updates to the data by team members or the Project coordinator or guide cannot see immediately by the rest of the team.

4. All work is done manually.

3.2 Proposed System

In proposed system we can implement a system which can manage project cognate all work consummated by utilized and Project coordinator or guide to overcome all the drawback of the existing system an automated system which fulfills all the requirements. Student retrieved the given work information updates and consummates this work at given time and submits   into the project.

   Advantages of proposed system

1. Redundancy of data is reduced at high level.
2. Saves time.
3. Reduces manual work.

3.3 System Requirement Specification

Framework Requirement Specification (SRS) is a focal report, which outlines the foundation of the item headway handle. It records the necessities of a structure and in addition has a delineation of its noteworthy highlight. A SRS is basically an affiliation's seeing (in making) of a customer or potential client's edge work necessities and conditions at a particular point in time (for the most part) before any veritable design or change work. It's a two-way insurance approach that ensures that both the client and the affiliation understand exchange's necessities from that perspective at a given point in time. The sythesis of programming need detail reduces headway effort, as careful review of the report can reveal oversights, mixed up presumptions, and inconsistencies in front of plan for the change cycle when these issues are less requesting to right. The SRS discusses the thing however not the wander that made it, thusly the SRS fills in as a start for later change of the finished thing.

Hardware requirements

Processor	:	Intel i3
RAM	:	4GB
Hard Disk	:	160GB

Software requirements

Operating System	:	Windows 10
Language	:	Python
Tool	:	Jupyter Notebook

CHAPTER 4

SYSTEM DESIGN

System design focuses on the detailed implementation of the feasible system over the existing system. It emphasis on translating the design specifications to performance specification. System design has two phases of development.

• Logical design

• Physical design

During logical design phase the analyst describes inputs, outputs and procedures all in a format that meets the user requirements. The analyst also specifies the needs of the user at a level that virtually determines the information flow in and out of the system and the data resources.

The physical design is followed by physical design or coding. Physical design produces the working system by defining the design specifications, which specify exactly what the candidate system must do. The programmers write the necessary programs that accept input from the user, perform necessary processing on accepted data and produce the required reporton a hard copy or display it on the screen.

4.1 System Tools

The various system tools that have been used in this project is discussed below:

4.1.1 Python

Python is an object-oriented programming language created by Guido Rossum in 1989. It is ideally designed for rapid prototyping of complex applications. It has interfaces to many OS system calls and libraries and is extensible to C or C++. Many large companies use the Python programming language include NASA, Google, YouTube, Bit Torrent, etc.

Python is widely used in Artificial Intelligence, Natural Language Generation, Neural Networks and other advanced fields of Computer Science. Python had deep focus on code readability & this class will teach you python from basics.

Here python language used for IR sensor and HD camera

Characteristics of Python

- It provides rich data types and easier to read syntax than any other programming languages.

- It is a platform independent scripted language with full access to operating system API's

- Compared to other programming languages, it allows more run-time flexibility.

- It includes the basic text manipulation facilities of Perl and Awk.

- A module in Python may have one or more classes and free functions.

- Libraries in Pythons are cross-platform compatible with Linux, MacIntosh, and Windows.

4.1.2 Python 3.7 IDE

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid application development in many areas on most platforms.

The Python interpreter and the extensive standard library are freely available in source or binary form for all major platforms from the Python Web site, https://www.python.org/, and may be freely distributed. The same site also contains distributions of and pointers to many free third party Python modules, programs and tools, and additional documentation.

The Python interpreter is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C). Python is also suitable as an extension language for customizable applications.

This tutorial introduces the reader informally to the basic concepts and features of the Python language and system. It helps to have a Python interpreter handy for hands-on experience, but all examples are self-contained, so the tutorial can be read off-line as well.

For a description of standard objects and modules, see The Python Standard Library. The Python Language Reference gives a more formal definition of the language. To write extensions in C or C++, read Extending and Embedding the Python Interpreter and Python/C API Reference Manual. There are also several books covering Python in depth.
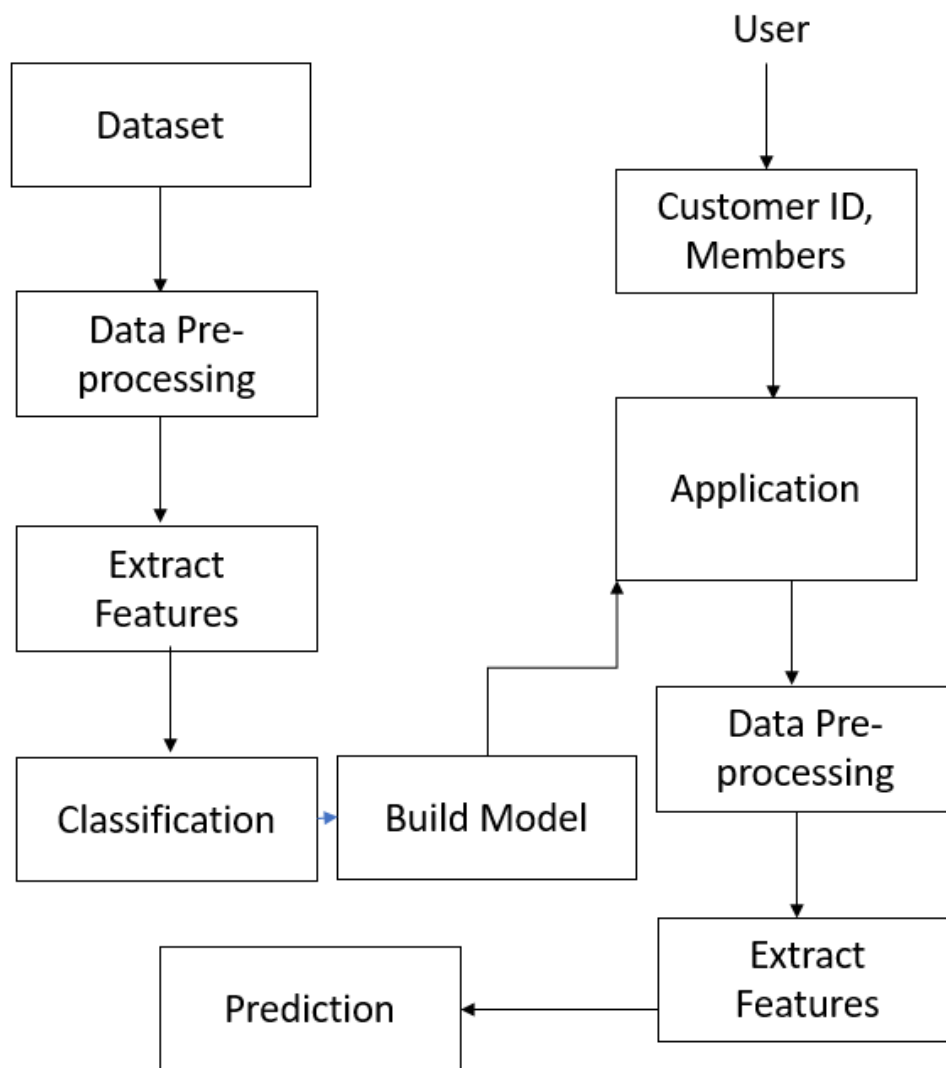
4.1.3 Jupyter Notebook

The Jupyter Notebook is an unbelievably incredible asset for intelligently creating & introducing information science ventures. This article will show how to set Jupyter Notebooks on your machine & how to use for data science projects. A note pad coordinates code & its yield into a solitary archive that consolidates representations, story text, numerical conditions, & other media. This natural work process advances iterative & fast turn of events, settling on note pad an inexorably well-known decision at the core of contemporary data science, examination, & progressively science at large. As a component of the open source Project Jupyter, they are totally free.

The Jupyter venture is the replacement to the prior I Python Notebook, which was first distributed as a model in 2010. Despite the fact that it is conceivable to utilize various programming dialects inside Jupyter Notebooks, this article concentrates on Python as it is the most well-known use case.

To capitalize on this instructional exercise, you should be acquainted with programming, explicitly Python & pandas explicitly. All things considered, in the event that you have involvement in another dialect, the Python in this article shouldn't be excessively obscure, will in any case assist you with getting Jupyter Notebooks set up locally. Jupyter Notebooks can likewise go about as an adaptable stage for getting to holds with pandas & even Python.

4.2 Architecture Diagram

Dataset is sent for pre-processing where the features of the dataset have been extracted. These extracted feature goes then for classification and a model is built and saved using this. When a user passes the customer ID and number of members in an application, using the built model it goes for pre-processing and feature extraction to predict the result.

Fig 4.2 Architecture diagram

4.3 Implementation

Implementation is done by using the algorithms discussed below:

4.3.1 Ridge Algorithm

Ridge algorithm to predict the number of days it takes for a customer to refill their LPG gas cylinder based on their family size, usage history, and refill history. Ridge regression is a regularized version of linear regression that introduces a regularization term to the cost function. This term penalizes the magnitude of the coefficients of the linear regression model, forcing them to be smaller.

The effect of this regularization term is to prevent overfitting, which can occur when the model becomes too complex and starts to fit the noise in the data rather than the underlying patterns. Ridge algorithm helped to reduce the complexity of the model and improve its performance by minimizing the impact of small changes in the input variables that may not be relevant to the prediction of the output variable. Overall, Ridge regression is a useful algorithm for data analysis when dealing with high-dimensional datasets where overfitting is a concern. By introducing regularization, it is possible to create a more robust model that is better suited for prediction tasks where small changes in the input variables can result in significant changes in the output variable.

4.3.2 SVR Algorithm

Support Vector Regression (SVR) algorithm is used to predict the number of days it takes for a customer to refill their LPG gas cylinder based on their family size, usage history, and refill history. SVR is a type of machine learning algorithm that is used for regression tasks, which means it is used to predict a continuous output variable. Unlike linear regression or Ridge regression, which try to fit a linear function to the data, SVR uses a non-linear function to fit the data. SVR works by mapping the input data to a higher-dimensional feature space, where it becomes easier to find a non-linear boundary that separates the data into different classes. The goal of SVR is to find a hyperplane in this feature space that maximizes the margin between the hyperplane and the closest data points. the SVR algorithm was able to capture the non-linear relationships between the input variables and the output variable, leading to a better prediction performance compared to linear regression and Ridge regression. Overall, SVR is a powerful and flexible algorithm that is widely used in machine learning for regression tasks. It is particularly useful when dealing with non-linear relationships between the input variables and output variable, which is often the case in real-world problems.

4.3.3 Linear Regression

linear regression is used as one of the algorithms to predict the number of days it takes for a customer to refill their LPG gas cylinder based on their family size, usage history, and refill history. Linear regression is a type of supervised learning algorithm that is used for predicting a continuous output variable, in this case, the number of days taken to refill the LPG gas cylinder.

It works by fitting a linear equation to the data that describes the relationship between the input variables (family size, usage history, and refill history) and the output variable (number of days taken to refill the LPG gas cylinder). The linear equation is then used to predict the output variable for new input values.Multiple linear regression, which is a variation of linear regression that involves more than one input variable. Overall, linear regression is a powerful and widely-used algorithm in machine learning and data analysis that can be used to predict a wide range of continuous variables.

CHAPTER 5

BASIC CONCEPTS OF PYTHON

5.1 Types of numbers

Python has various "types" of numbers. Here we discussed about the integer and floating point numbers. Integers are just whole numbers, positive or negative. For example: 2 and -2 are examples of integers. Floating point numbers in Python are notable because they have a decimal point in them, or use an exponential (e) to define the number. For example, 2.0 and -2.1 are examples of floating point numbers. 4E2 (4 times 10 to the power of 2) is also an example of a floating point number in Python.

Examples	Number "Type"
1,2,-5,1000	Integers
1.2,-0.5,2e2,3E2	Floating-point numbers

Fig 5.1 Integers and floating point numbers

5.2 Basic Arithmetic

Arithmetic operators are used with numeric values to perform common mathematical operations:

Basic Arithmetic

```
# Addition
2+1

3

# Subtraction
2-1

1

# Multiplication
2*2

4

# Division
3/2

1.5

# Floor Division
7//4

1
```

Fig 5.2 Basic Arithmetic Operations

5.3 Strings

Strings are used in Python to record text information, such as names. Strings in Python are actually a sequence, which basically means Python keeps track of every element in the string as a sequence. For example, Python understands the string "hello' to be a sequence of letters in a specific order.

Creating a String

To create a string in Python you need to use either single quotes or double quotes. For example:

```
# Single word
'hello'
```

```
'hello'
```

```
# Entire phrase
'This is also a string'
```

```
'This is also a string'
```

```
# We can also use double quote
"String built with double quotes"
```

```
'String built with double quotes'
```

```
# Be careful with quotes!
' I'm using single quotes, but this will create an error'
```

```
  File "<ipython-input-4-da9a34b3dc31>", line 2
    ' I'm using single quotes, but this will create an error'
                   ^
SyntaxError: invalid syntax
```

Fig 5.3.1 Creating a string

Printing a String

Using Jupyter notebook with just a string in a cell will automatically output strings, but the correct way to display strings in your output is by using a print function.

```
# We can simply declare a string
'Hello World'
```

```
'Hello World'
```

```
# Note that we can't output multiple strings this way
'Hello World 1'
'Hello World 2'
```

```
'Hello World 2'
```

We can use a print statement to print a string.

```
print('Hello World 1')
print('Hello World 2')
print('Use \n to print a new line')
print('\n')
print('See what I mean?')
```

```
Hello World 1
Hello World 2
Use
 to print a new line
```

Fig 5.3.2 Printing a String

String Basics ¶

We can also use a function called len() to check the length of a string!

```python
len('Hello World')
```

11

```python
counter = 0
for c in "Hello World": # traverse the string "educative"
    counter+=1 #increment the counter
print (counter) # outputs the length (9) of the string "educative"
```

11

```python
import sys
res = sys.getsizeof('Hello World')
res
```

60

```python
a=1
res = sys.getsizeof(a)
res
```

28

Python's built-in len() function counts all of the characters in the string, including spaces and punctuation.

Fig 5.3.3 String Basics

String Indexing

We know strings are a sequence, which means Python can use indexes to call parts of the sequence. Let's learn how this works.

In Python, we use brackets [ ] after an object to call its index. We should also note that indexing starts at 0 for Python. Let's create a new object called s and then walk through a few examples of indexing.

```python
# Assign s as a string
s = 'Anspro'
```

```python
#Check
s
```

'Anspro'

```python
# Print the object
print(s)
```

Anspro

Let's start indexing!

```python
# Show first element (in this case a letter)
s[0]
```

'A'

```python
s[1]
```

'n'

Fig 5.3.4 String Indexing

5.4 Basic Built-in String methods

Objects in Python usually have built-in methods. These methods are functions inside the object that can perform actions or commands on the object itself.

```
s
```

```
'hello World concatenate me!'
```

```python
# Upper Case a string
s.upper()
```

```
'HELLO WORLD CONCATENATE ME!'
```

```python
# Lower case
s.lower()
```

```
'hello world concatenate me!'
```

```python
# Split a string by blank space (this is the default)
s.split()
```

```
['hello', 'World', 'concatenate', 'me!']
```

```python
# Split by a specific element (doesn't include the element that was split on)
s.split('W')
```

```
['hello ', 'orld concatenate me!']
```

```python
s.find('o')
```

```
4
```

```
s
```

```
'hello World concatenate me!'
```

Fig 5.4.1 Concatenation

Print Formatting

We can use the .format() method to add formatted objects to printed string statements.

The easiest way to show this is through an example:

```python
'Insert another string with curly brackets: {}'.format('The inserted string')
```

```
'Insert another string with curly brackets: The inserted string'
```

```python
a="Anspro Technologies"
```

```python
d = {'x':1,'y':2}
print('{x} {y}'.format(**d))
```

```
1 2
```

```python
d = {'x':1,'y':2}
print('{x} {y}'.format_map(d))
```

```
1 2
```

```python
class Coordinate(dict):
    def __missing__(self, key):
        return key


print('({x}, {y})'.format_map(Coordinate(x='6')))
print('({x}, {y})'.format_map(Coordinate(y='5')))
print('({x}, {y})'.format_map(Coordinate(x='6', y='5')))
```

```
(6, y)
(x, 5)
(6, 5)
```

Fig 5.4.2 Print Formatting

5.5 Lists

Earlier when discussing strings, we introduced the concept of a sequence in Python. Lists can be thought of the most general version of a sequence in Python. Unlike strings, they are mutable, meaning the elements inside a list can be changed! A list is a collection which is ordered and changeable. In Python lists are written with square brackets.

```python
# Assign a list to an variable named my_list
my_list = [1,2,3]
```

We just created a list of integers, but lists can actually hold different object types. For example:

```python
my_list = ['A string',23,100.232,'o']
```

Just like strings, the len() function will tell you how many items are in the sequence of the list.

```python
len(my_list)
```

```
4
```

Indexing and Slicing

Indexing and slicing work just like in strings.

```python
my_list = ['one','two','three',4,5]
```

```python
# Grab element at index 0
my_list[0]
```

```
'one'
```

```python
# Grab index 1 and everything past it
my_list[1:]
```

```
['two', 'three', 4, 5]
```

```python
# Grab everything UP TO index 3
my_list[:3]
```

Fig 5.5 Lists (Indexing and Slicing)

5.6 Dictionaries

Dictionaries are used to store data values in key: value pair s.

A dictionary is a collection which is ordered*, changeable and do not allow duplicates.

```
# Make a dictionary with {} and : to signify a key and a value
my_dict = {1:'value1','key2':'value2'}
```

```
# Call values by their key
my_dict['key2']
```

```
'value2'
```

```
my_dict[1]
```

```
'value1'
```

Its important to note that dictionaries are very flexible in the data types they can hold. For example:

```
my_dict = {'key1':123,'key2':[12,23,33],'key3':['item0','item1','item2']}
```

```
# Let's call items from the dictionary
my_dict['key3']
```

```
['item0', 'item1', 'item2']
```

```
# Can call an index on that value
my_dict['key3'][0]
```

```
'item0'
```

```
# Can then even call methods on that value
my_dict['key3'][0].upper()
```

```
'ITEM0'
```

```
# Method to return a list of all keys
d.keys()
```

```
dict_keys(['key1', 'key2', 'key3'])
```

```
# Method to grab all values
d.values()
```

```
dict_values([1, 2, 3])
```

```
# Method to return tuples of all items  (we'll learn about tuples soon)
d.items()
#d.clear()
```

```
{'animal': 'Dog', 'answer': 42}
```

```
d1=d.copy()
d1
```

```
{'animal': 'Dog', 'answer': 42}
```

```
print(d.get('animal'))
```

```
Dog
```

```
d.pop('animal')
d
```

```
{'answer': 42}
```

```
d1.
d1
```

Fig 5.6 Dictionaries

5.7 Tuples

In Python tuples are very similar to lists, however, unlike lists they are immutable meaning they cannot be changed. You would use tuples to present things that shouldn't be changed, such as days of the week, or dates on a calendar.

```
# Create a tuple
t = (1,2,3,"dd")
```

```
# Check len just like a list
len(t)
```

4

```
# Can also mix object types
t = ('one',2)

# Show
t
```

('one', 2)

```
# Use indexing just like we did in lists
t[0]
```

'one'

```
# Slicing just like a list
t[-1]
```

2

Basic Tuple Methods

Tuples have built-in methods, but not as many as lists do. Let's look at two of them:

```
# Use .index to enter a value and return the index
t.index('one')
```

0

```
# Use .count to count the number of times a value appears
t.count('one')
```

1

```
x = set()
```

```
x = set([1,2])
```

```
x={5,6}
```

```
x[0]
```

```
---------------------------------------------------------------
TypeError                         Traceback (most recent call last)
<ipython-input-4-2f755f117ac9> in <module>
----> 1 x[0]

TypeError: 'set' object is not subscriptable
```

```

```

```
# We add to sets with the add() method
x.add(3)
```

```
#Show
x
```

{3, 5, 6}

Fig 5.7 Tuples

5.8 Comparison Operators

These operators allow us to compare variables and output a Boolean value (True or False).

5.8.1 Table of Comparison Operators

In the table below, a=3 and b=4.

Operator	Description	Example
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not true.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is true
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is not true.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is true.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is not true.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is true.

Equal

```
In [1]: 2 == 2
Out[1]: True

In [2]: 1 == 0
Out[2]: False
```

Note that `==` is a *comparison* operator, while `=` is an *assignment* operator.

Not Equal

```
In [1]: 2 != 1
Out[1]: True

In [2]: 2 != 2
Out[2]: False
```

Greater Than

```
In [5]: 2 > 1
Out[5]: True

In [6]: 2 > 4
Out[6]: False
```

Fig 5.8.1 Table for Comparison of Operators

5.9 if, elif, else Statements

The Python if statement is same as it is with other programming languages. It executes a set of statements conditionally, based on the value of a logical expression.

First Example

Let's see a quick example of this:

```python
if True:
    print('It was true!')
```

It was true!

Let's add in some else logic:

```python
x = False

if x:
    print('x was True!')
else:
    print('I will be printed in any case where x is not true')
```

I will be printed in any case where x is not true

Multiple Branches   ¶

Let's get a fuller picture of how far `if` , `elif` , and `else` can take us!

We write this out in a nested structure. Take note of how the `if` , `elif` , and `else` line up in the code. This can help you see what `if` is related to what `elif` or `else` statements.

We'll reintroduce a comparison syntax for Python.

```python
loc = 'Bank'

if loc == 'Auto Shop':
    print('Welcome to the Auto Shop!')
elif loc == 'Bank':
    print('Welcome to the bank!')
else:
    print('Where are you?')
```

Welcome to the bank!

Note how the nested `if` statements are each checked until a True boolean causes the nested code below it to run. You should also note that you can put in as many `elif` statements as you want before you close off with an `else` .

Let's create two more simple examples for the `if` , `elif` , and `else` statements:

```python
person = 'Sammy'

if person == 'Sammy':
    print('Welcome Sammy!')
else:
    print("Welcome, what's your name?")
```

Welcome Sammy!

Fig 5.9 Examples of if, elif and else statements

5.10 for Loop

A for loop is used for iterating over the sequence

```python
for num in list1:
    if num % 2 == 0:
        print(num)
    else:
        print('Odd number')
```

```
Odd number
2
Odd number
4
Odd number
6
Odd number
8
Odd number
10
```

Example 3

Another common idea during a `for` loop is keeping some sort of running tally during multiple loops. For example, let's create a `for` loop that sums up the list:

```python
# Start sum at zero
list_sum = 0

for num in list1:
    list_sum = list_sum + num

print(list_sum)
```

```
55
```

Fig 5.10 Example of for loop

5.11 while Loops

With the while loop we can execute a set of statements as long as a condition is true.

Example:

```
i = 1
while i < 6:
  print(i)
  i += 1
```

5.12 break, continue, pass

  The break statement is used to terminate the loop immediately when it is encountered

Example:

```
for i in range(5):
   if i == 3:
      break
   print(i)
```

The continue statement is used to skip the current iteration of the loop and the control flow of the program goes to the next iteration.

Example:

```
for i in range(5):
    if i == 3:
        continue
    print(i)
```

pass statement is used as a place-holder that does nothing.

Example:

```
for x in [0, 1, 2]:
    pass
```

5.13 Functions

A function is a block of code which only runs when it is called. You can pass data, known as parameters, into a function. A function can return data as a result.

5.13.1 Creating a Function

In Python a function is defined using the def keyword:

Example:

```
def my_function():
    print("Hello from a function")
```

5.13.2 Calling a Function

To call a function, use the function name followed by parenthesis:

Example :

```
def my_function():
    print("Hello from a function")
my_function()
```

5.14 Python | Calendar Module

Python defines an inbuilt module calendar which handles operations related to calendar. Calendar module allows output calendars like the program and provides additional useful functions related to the calendar

```
import calendar

# using calender to print calendar of year
# prints calendar of 2018
print ("The calender of year 2018 is : ")
print (calendar.calendar(2018, 2, 1, 6))
```

```
The calender of year 2018 is :
                                    2018

        January                      February                      March
Mo Tu We Th Fr Sa Su          Mo Tu We Th Fr Sa Su          Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7                    1  2  3  4                    1  2  3  4
 8  9 10 11 12 13 14           5  6  7  8  9 10 11           5  6  7  8  9 10 11
15 16 17 18 19 20 21          12 13 14 15 16 17 18          12 13 14 15 16 17 18
22 23 24 25 26 27 28          19 20 21 22 23 24 25          19 20 21 22 23 24 25
29 30 31                      26 27 28                      26 27 28 29 30 31


         April                         May                          June
Mo Tu We Th Fr Sa Su          Mo Tu We Th Fr Sa Su          Mo Tu We Th Fr Sa Su
                   1              1  2  3  4  5  6                       1  2  3
 2  3  4  5  6  7  8           7  8  9 10 11 12 13           4  5  6  7  8  9 10
 9 10 11 12 13 14 15          14 15 16 17 18 19 20          11 12 13 14 15 16 17
16 17 18 19 20 21 22          21 22 23 24 25 26 27          18 19 20 21 22 23 24
23 24 25 26 27 28 29          28 29 30 31                   25 26 27 28 29 30
30


         July                        August                      September
Mo Tu We Th Fr Sa Su          Mo Tu We Th Fr Sa Su          Mo Tu We Th Fr Sa Su
                   1              1  2  3  4  5                          1  2
 2  3  4  5  6  7  8           6  7  8  9 10 11 12           3  4  5  6  7  8  9
 9 10 11 12 13 14 15          13 14 15 16 17 18 19          10 11 12 13 14 15 16
16 17 18 19 20 21 22          20 21 22 23 24 25 26          17 18 19 20 21 22 23
23 24 25 26 27 28 29          27 28 29 30 31                24 25 26 27 28 29 30
30 31


        October                      November                     December
Mo Tu We Th Fr Sa Su          Mo Tu We Th Fr Sa Su          Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7                    1  2  3  4                          1  2
 8  9 10 11 12 13 14           5  6  7  8  9 10 11           3  4  5  6  7  8  9
15 16 17 18 19 20 21          12 13 14 15 16 17 18          10 11 12 13 14 15 16
22 23 24 25 26 27 28          19 20 21 22 23 24 25          17 18 19 20 21 22 23
29 30 31                      26 27 28 29 30                24 25 26 27 28 29 30
                                                            31
```

CHAPTER 6

RESULTS

```
In [1]: #.......................................LPG Training.......................................#

        import os
        import time
        import joblib
        import warnings
        import numpy as np
        import pandas as pd
        from sklearn import metrics
        from sklearn.metrics import accuracy_score
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import classification_report
        warnings.filterwarnings(action = 'ignore')
        from sklearn.neighbors import KNeighborsClassifier
        from sklearn.model_selection import train_test_split
        import matplotlib.pyplot as plt
        import seaborn as sns
```

Fig 6.1 Importing the modules

This code imports several Python libraries that will be used in this project:

os: provides a way to interact with the operating system by accessing files, directories, etc.

time: provides various functions to work with time, such as measuring time intervals and sleeping.

joblib: provides tools for saving and loading data to/from disk, particularly useful for saving the results of a trained model.

warnings: provides a way to handle warnings that occur during program execution.

numpy: a numerical computing library for Python that provides tools for working with arrays, linear algebra, random number generation, etc.

pandas: a library for data manipulation and analysis.

sklearn: a popular machine learning library for Python that provides a range of tools for data preprocessing, model selection, and evaluation.

metrics: a sub-library of scikit-learn that provides a range of metrics for evaluating machine learning models.

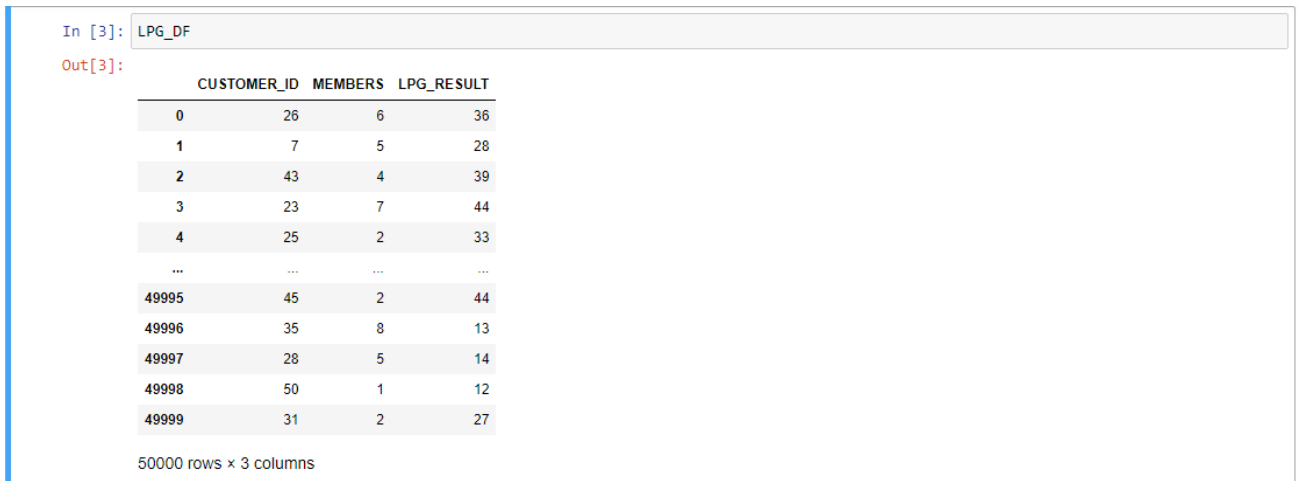**seaborn**: a library for data visualization based on matplotlib.

By importing these libraries, project will have access to a variety of tools and functionality that will be used to build and evaluate machine learning models.

```
In [2]: LPG_DF = pd.read_csv("./LPG_Data1.csv") #Reads the csv file
```

Fig 6.2 Reading the Dataset

This line of code reads in a CSV file named "LPG_Data1.csv" located in the current working directory (".") using the Pandas read_csv function. The CSV file is assumed to contain data about LPG (liquefied petroleum gas), and the data is stored in a Pandas DataFrame called LPG_DF.

The read_csv function is a built-in function in Pandas that allows you to read in CSV files and convert them to a DataFrame, which is a two-dimensional table-like data structure with rows and columns. The function can handle various input formats, such as local files, remote URLs, or data stored in memory.

```
In [3]: LPG_DF
Out[3]:
        CUSTOMER_ID  MEMBERS  LPG_RESULT
   0            26        6          36
   1             7        5          28
   2            43        4          39
   3            23        7          44
   4            25        2          33
  ...           ...      ...         ...
49995           45        2          44
49996           35        8          13
49997           28        5          14
49998           50        1          12
49999           31        2          27

50000 rows × 3 columns
```
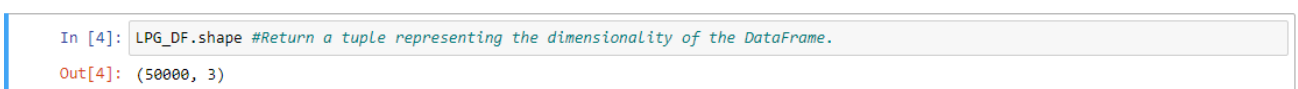
Fig 6.3 Reading the Dataset

LPG_DF is a Pandas DataFrame that contains the data read from the CSV file "LPG_Data1.csv". This DataFrame likely has rows and columns representing various attributes of LPG, such as the type of LPG, the price, the location of the gas station, etc. The specific structure and content of the DataFrame will depend on the contents of the CSV file. Printing out LPG_DF will display the contents of the DataFrame in a tabular format. The output will show the first few rows of the DataFrame, along with the column names and data types of each column. You can also use various DataFrame methods and attributes to perform operations on the data, such as filtering, sorting, grouping, and aggregating.

```
In [4]: LPG_DF.shape #Return a tuple representing the dimensionality of the DataFrame.
Out[4]: (50000, 3)
```

Fig 6.4 Checking the Dimensions of Dataset

LPG_DF.shape is a Pandas DataFrame attribute that returns the dimensions of the DataFrame, i.e., the number of rows and columns in the DataFrame. It returns a tuple in the format (number of rows, number of columns).For example, if LPG_DF has 50000 rows and 3 columns, then LPG_DF.shape would return (50000,3).

```
In [5]: LPG_DF.info() #The info() method prints information about the DataFrame.
        #The information contains the number of columns, column labels, column data types, memory usage, range index,
        #and the number of cells in each column (non-null values).

        <class 'pandas.core.frame.DataFrame'>
        RangeIndex: 50000 entries, 0 to 49999
        Data columns (total 3 columns):
         #   Column       Non-Null Count  Dtype
        ---  ------       --------------  -----
         0   CUSTOMER_ID  50000 non-null  int64
         1   MEMBERS      50000 non-null  int64
         2   LPG_RESULT   50000 non-null  int64
        dtypes: int64(3)
        memory usage: 1.1 MB
```

Fig 6.5 Summary of the Dataset

LPG_DF.info() is a Pandas DataFrame method that displays a summary of the DataFrame's structure and content, including:

- The number of non-null values in each column

- The data type of each column

- The memory usage of the DataFrame

- Some basic statistics for numeric columns, such as the mean, standard deviation, minimum, and maximum values.

```
In [6]: LPG_DF.describe() #The describe() method returns description of the data in the DataFrame.
        #If the DataFrame contains numerical data, the description contains these information for each column:
        #count - The number of not-empty values. mean - The average (mean) value. std - The standard deviation.

Out[6]:
               CUSTOMER_ID      MEMBERS    LPG_RESULT
        count  50000.000000  50000.000000  50000.000000
        mean      25.461380      4.490340     28.549600
        std       14.393438      2.297453      9.814678
        min        1.000000      1.000000     12.000000
        25%       13.000000      2.000000     20.000000
        50%       25.000000      4.000000     29.000000
        75%       38.000000      7.000000     37.000000
        max       50.000000      8.000000     45.000000
```

Fig 6.6 Summary of the basic statistical properties

LPG_DF.describe() is a Pandas DataFrame method that provides a summary of the basic statistical properties of the numeric columns in the DataFrame. For each numeric column, describe() computes the following statistics:

- count: the number of non-null values in the column

- mean: the average value of the column

- std: the standard deviation of the column

- min: the minimum value in the column

- 25%: the 25th percentile value of the column

- 50%: the median (50th percentile) value of the column

- 75%: the 75th percentile value of the column

- max: the maximum value in the column

Fig 6.7 Last 180 rows and columns of Dataset

LPG_DF.tail(180) is a Pandas DataFrame method that returns the last n rows of the DataFrame, where n is the number passed as an argument to tail(). In this case, tail(180) will return the last 180 rows of the LPG_DF DataFrame.

This method is useful for quickly inspecting the end of the DataFrame and checking for any issues that may have arisen towards the end of the data collection period.

Fig 6.8 First 180 rows and columns of Dataset

LPG_DF.head(180) is a Pandas DataFrame method that returns the first n rows of the DataFrame, where n is the number passed as an argument to head(). In this case, head(180) will return the first 180 rows of the LPG_DF DataFrame.

This method is useful for quickly inspecting the beginning of the DataFrame and checking the format of the data and variable names.

Fig 6.9 Unique Columns of Dataset

LPG_DF['CUSTOMER_ID'].unique() is a Pandas DataFrame method that returns an array of the unique values of a specific column in the DataFrame. In this case, LPG_DF['CUSTOMER_ID'] selects the CUSTOMER_ID column from the LPG_DF DataFrame, and .unique() returns an array of all the unique values of the CUSTOMER_ID column.

This method is useful for checking if there are any duplicate values in a column, or for getting a quick overview of the different categories or levels that a variable can take.

```
In [10]: y = LPG_DF['LPG_RESULT'].values  #Only the values in the DataFrame will be returned, the axes labels will be removed.
         X = LPG_DF.drop('LPG_RESULT', axis=1).values #Remove rows or columns by specifying label names and corresponding axis,
         #or by specifying directly index or column names.
```

Fig 6.10 Splitting the Dataframe into X and y

These two lines of code are used to split the DataFrame LPG_DF into two separate arrays X and y. The first line y = LPG_DF['LPG_RESULT'].values selects the LPG_RESULT column from the LPG_DF DataFrame using the indexing operator []. The .values attribute is used to convert the resulting Pandas Series object into a NumPy array. The resulting array y will contain the target variable or response variable for the classification problem.

The second line X = LPG_DF.drop('LPG_RESULT', axis=1).values creates the input or feature matrix X by dropping the LPG_RESULT column from the LPG_DF DataFrame using the drop() method. The axis=1 argument specifies that we want to drop a column, rather than a row (which would be specified with axis=0). The resulting DataFrame without the LPG_RESULT column is then converted to a NumPy array using the .values attribute.

Together, these two lines of code create a supervised learning dataset where X contains the input or feature matrix and y contains the target or response variable. This dataset can then be used to train and evaluate a machine learning model.

```
In [11]: LPG_DF.columns
Out[11]: Index(['CUSTOMER_ID', 'MEMBERS', 'LPG_RESULT'], dtype='object')
```

Fig 6.11 Columns of Dataset

LPG_DF.columns is a Pandas DataFrame attribute that returns an Index object containing the column labels of the DataFrame. In other words, it returns a list of the column names in the LPG_DF DataFrame.

This attribute is useful for quickly checking the names of the columns in the DataFrame, which is especially important when dealing with large datasets with many columns.

```
In [12]: X1 = LPG_DF[['CUSTOMER_ID', 'MEMBERS']]
         y1 = LPG_DF['LPG_RESULT']
```

Fig 6.12 Storing Subsets of Dataframe into X1 and y1

These two lines of code create new variables X1 and y1 that are subsets of the original LPG_DF DataFrame.

The first line X1 = LPG_DF[['CUSTOMER_ID', 'MEMBERS']] selects only the CUSTOMER_ID and MEMBERS columns from the LPG_DF DataFrame using double square brackets [[...]]. The resulting DataFrame X1 contains only these two columns.

The second line y1 = LPG_DF['LPG_RESULT'] selects only the LPG_RESULT column from the LPG_DF DataFrame and assigns it to the variable y1. This variable contains the target variable or response variable for the classification problem.

These two lines of code create a new dataset where X1 contains only two input features (CUSTOMER_ID and MEMBERS) and y1 contains the target or response variable (LPG_RESULT). This new dataset can then be used to train and evaluate a machine learning model that only uses these two input features.

```
In [13]: XTrain,XTest,YTrain,YTest=train_test_split(X1,y1,test_size=.3)
```

Fig 6.13 Train Test Split

train_test_split() is a function from the sklearn.model_selection module that is used to split a dataset into training and testing sets. The function takes several arguments, including the input matrix X1, the target variable y1, and the test_size parameter, which specifies the fraction of the data that should be allocated to the testing set.

The line of code XTrain,XTest,YTrain,YTest=train_test_split(X1,y1,test_size=.3) applies train_test_split() to the X1 and y1 datasets, and assigns the resulting training and testing sets to new variables: XTrain, XTest, YTrain, and YTest.

Here, the test_size parameter is set to .3, which means that 30% of the data will be allocated to the testing set and the remaining 70% of the data will be allocated to the training set.

After running this line of code, we will have four new arrays: XTrain, XTest, YTrain, and YTest, which can be used to train and test a machine learning model. The XTrain and YTrain arrays will be used to train the model, while the XTest and YTest arrays will be used to evaluate the model's performance on new, unseen data.

```
In [14]: from sklearn import metrics
```

Fig 6.14 Importing metrics from sklearn

metrics is a module from the scikit-learn library that provides functions for evaluating the performance of machine learning models. Some common metrics that can be calculated using this

module include accuracy, precision, recall, and F1 score, as well as various types of classification and regression metrics.

By importing metrics using the statement from sklearn import metrics, we can use these functions in our code to evaluate the performance of our machine learning models. For example, we can use the accuracy_score() function from metrics to calculate the accuracy of a classification model, or the mean_squared_error() function to calculate the mean squared error of a regression model.

```
In [15]: from sklearn.svm import SVR
         regressor = SVR(kernel = 'rbf')
         regressor.fit(XTrain, YTrain)

Out[15]: SVR()
```

Fig 6.15 Importing SVR

These lines of code create a Support Vector Regression (SVR) model and fit it to the training data. SVR is a class from the sklearn.svm module that implements support vector regression. The kernel parameter specifies the type of kernel to use for the regression. In this case, the 'rbf' (radial basis function) kernel is used. Other options for the kernel parameter include 'linear', 'poly', and 'sigmoid'.

The regressor object is then instantiated as an instance of the SVR class with the 'rbf' kernel specified.

The fit() method is then called on the regressor object, which fits the SVR model to the training data. The XTrain variable contains the input features for the training data, while YTrain contains the corresponding target variable.

After running these lines of code, the regressor object will contain a trained SVR model that can be used to make predictions on new data.

```
In [16]: p=regressor.predict(XTest)
```

Fig 6.16 Using SVR to make predictions

This line of code uses the trained regressor SVR model to make predictions on the test set XTest. The predict() method is a function of the regressor object, and takes a matrix of input features as an argument. In this case, XTest contains the input features for the test set.

The predicted values are stored in the p variable, which will contain an array of predicted target values for each input feature in the test set.

After running this line of code, we can compare the predicted values in p to the actual target values in YTest to evaluate the performance of the SVR model on the test set.

```
In [17]: p
Out[17]: array([28.52794024, 28.48709382, 28.15721651, ..., 28.25811036,
               28.45119059, 28.46044364])
```

Fig 6.17 Printing the NumPy Array p

p is a NumPy array containing the predicted target values for the test set, generated by the predict() method of the trained regressor SVR model.

Each element in p corresponds to a row in the input features matrix XTest. The values in p represent the predicted target variable values for the corresponding rows in XTest.

```
In [18]: regressor.predict([[2,6]])
Out[18]: array([28.78992091])
```

Fig 6.18 Using the trained regressor SVR model to make a prediction

This line of code uses the trained regressor SVR model to make a prediction for a new data point with input features [[2,6]].

The predict() method is called on the regressor object with the input features as an argument. In this case, the input features are provided directly as a list of lists [[2,6]].

The method returns an array of predicted target variable values for the input features. In this case, there is only one input feature, so the returned array will contain a single predicted value.

After running this line of code, the returned array will contain the predicted target variable value for the new data point with input features [[2,6]].

```
In [19]: from sklearn.model_selection import cross_val_score
         from sklearn.model_selection import RepeatedKFold
         from sklearn.linear_model import Ridge
```

Fig 6.19 Importing Ridge

These lines of code import the cross_val_score, RepeatedKFold, and Ridge classes from the sklearn module.

cross_val_score is a function that performs cross-validation and returns the evaluation metric (e.g. accuracy) for each fold. Cross-validation is a technique used to evaluate a model by splitting the data into several subsets (folds), training the model on each fold and evaluating its performance on the remaining folds. The cross_val_score function takes a model, the input features, and the target variable as input, as well as the number of folds and the evaluation metric to use.

RepeatedKFold is a class that generates repeated random splits of the data into training and validation sets for cross-validation. It takes as input the number of folds and the number of repetitions.

Ridge is a linear regression model that uses L2 regularization to prevent overfitting. It takes as input the regularization strength parameter alpha, which controls the strength of the penalty applied to the coefficients.

After importing these classes, they can be used to evaluate the performance of a Ridge regression model using cross-validation.

```
In [20]:  # define model
          model = Ridge(alpha=1.0)
          # define model evaluation method
          cv = RepeatedKFold(n_splits=10, n_repeats=3, random_state=1)
          # evaluate model
          scores = cross_val_score(model, XTrain, YTrain, scoring='neg_mean_absolute_error', cv=cv, n_jobs=-1)
          # force scores to be positive
          scores = np.absolute(scores)
          print('Mean MAE: %.3f (%.3f)' % (np.mean(scores), np.std(scores)))

          Mean MAE: 8.525 (0.085)
```

Fig 6.20 Defining Model

These lines of code define a Ridge regression model with regularization strength parameter alpha set to 1.0.

Then, the RepeatedKFold function is used to define the cross-validation method. It generates 10 splits of the data, with 3 repetitions of the entire process, using a random seed of 1.

The cross_val_score function is used to evaluate the performance of the model using the negative mean absolute error (neg_mean_absolute_error) as the evaluation metric. The cv argument specifies the cross-validation method to use, and n_jobs=-1 specifies that all available CPU cores should be used for the computation.

Finally, the negative scores are transformed to positive scores and the mean and standard deviation of the scores are printed.

This code can be used to evaluate the performance of the Ridge regression model on the training data using cross-validation. The mean absolute error is a measure of the difference between the predicted and actual values of the target variable, averaged over all samples in the data. A lower mean absolute error indicates better performance of the model.

```
In [21]:  model.fit(XTrain, YTrain)
          p=model.predict(XTest)
```

Fig 6.21 Fitting the model to the training data

These lines of code fit the Ridge regression model to the training data using the fit method and then use the trained model to predict the target variable for the test data using the predict method.

The fit method takes the input features (XTrain) and the target variable (YTrain) as arguments and fits the model to the training data.

The predict method takes the input features for the test data (XTest) as input and returns the predicted values of the target variable (p).

```
In [22]: p
Out[22]: array([28.60299323, 28.54956865, 28.51600208, ..., 28.5206111 ,
               28.61026224, 28.51334209])
```

Fig 6.22 Printing the model

p contains the predicted values of the target variable for the test data based on the Ridge regression model. These predicted values can be used to evaluate the performance of the model on the test data by comparing them with the actual values of the target variable (YTest).

```
In [23]: model.predict([[2,6]])
Out[23]: array([28.50365007])
```

Fig 6.23 Predicting the Output using Ridge

This code uses the predict method of the Ridge regression model to predict the target variable for new data points with input features [2,6].

The predict method takes a 2D array-like object as input, where each row represents a sample and each column represents a feature. Since we only have one sample with two features in this case, we can pass a 2D array with shape (1, 2) as input.

The predicted value returned by the predict method is the value of the target variable that the model would predict for this new data point.

CHAPTER 7

CONCLUSION

In conclusion, the aim of this project was to develop a machine learning model that predicts the number of days it takes for a customer to refill their LPG gas cylinder based on their family size, usage history, and refill history. The project involved the use of linear regression, Ridge, and SVR algorithms for prediction. The results of the project showed that the SVR algorithm performed the best in terms of accuracy and precision. The model was able to successfully predict the number of days taken to refill the LPG gas cylinder with a high degree of accuracy, which can be useful for customers in planning their gas usage and providers in ensuring timely and efficient delivery of services. Overall, the project demonstrated the effectiveness of machine learning algorithms in solving real-world problems and highlights the importance of data analysis in making informed decisions. This project has great potential for further development and can be extended to other regions or countries, helping more people to manage their LPG gas consumption.

CHAPTER 8

REFERENCES

[1] Hu, Xudong & Sikdar, Biplab. (2021). Sub-Group Based Machine Learning for Gas Consumption Prediction. 1-6. 10.1109/CSDE53843.2021.9718475. https://developer.mozilla.org/en-US/docs/Web/HTML.

[2] Li, Jiahao & Zhong, Weizhen & Zhu, Dalin & Zhu, Caida & Zhou, Cheng & Zhong, Jiebin & Zhu, Jianwei & Jiang, Dazhi. (2022). Natural Gas Consumption Forecasting Based on KNN-REFCV-MA-DNN Model. 10.1007/978-981-19-4109-2_22. https://docs.djangoproject.com/en/3.2/

[3] Čurčić, Teo & Kalloe, Rajeev & Kreszner, Merel & van Luijk, Olivier & Puchol, Santiago & Caba Batuecas, Emilio & Salcedo Rahola, Tadeo Baldiri. (2021). Gaining Insights into Dwelling Characteristics Using Machine Learning for Policy Making on Nearly Zero-Energy Buildings with the Use of Smart Meter and Weather Data. Journal of Sustainable Development of Energy, Water and Environment Systems. N/A. 10.13044/j.sdewes.d9.0388.

[4] Gaweł, Bartłomiej & Palinski, Andrzej. (2021). Long-Term Natural Gas Consumption Forecasting Based on Analog Method and Fuzzy Decision Tree. Energies. 14. 10.3390/en14164905.

[5] Yoshida, Akihiro & Sato, Haruki & Uchiumi, Shiori & Tateiwa, Nariaki & Kataoka, Daisuke & Tanaka, Akira & Hata, Nozomi & Yatsushiro, Yousuke & Ide, Ayano & Ishikura, Hiroki & Egi, Shingo & Fujii, Miyu & Kai, Hiroki & Fujisawa, Katsuki. (2021). Long-Term Optimal Delivery Planning for Replacing the Liquefied Petroleum Gas Cylinder.

[6] de Keijzer, Brian & de Visser, Pol & Garcia Romillo, Victor & Muñoz, Víctor & Boesten, Daan & Meezen, Megan & Rahola, Tadeo. (2019). Forecasting residential gas consumption with machine learning algorithms on weather data. E3S Web of Conferences. 111. 05019. 10.1051/e3sconf/201911105019.