**Programs of Questions**

Q1 :

# import the sys module

import sys

# function to calculate fuel level after driving a certain number of miles

def drive(miles, fuel\_efficiency, fuel\_level):

# maximum distance that can be driven with the current fuel level

max\_distance = fuel\_efficiency \* fuel\_level

# if the number of miles requested to drive is more than the maximum distance, the car cannot drive that far

if miles > max\_distance:

# print a message indicating how far the car can drive with the current fuel level

print(f"You don't have enough fuel to drive {miles} miles. You can drive another {max\_distance} miles on this gas.")

# return the current fuel level as it is

return fuel\_level

# otherwise, the car can drive the requested distance

else:

# calculate the fuel used to drive the requested distance

fuel\_used = miles / fuel\_efficiency

# update the fuel level accordingly

fuel\_level -= fuel\_used

# print a message indicating the distance driven and the remaining fuel level

print(f"You drove {miles} miles. You have {fuel\_level:.2f} gallons of gas left.")

# return the updated fuel level

return fuel\_level

# function to add gas to the car's tank

def add\_gas(gallons, tank\_size, fuel\_level):

# if the number of gallons to add is not positive, it is an invalid input

if gallons <= 0:

print("You must enter a positive number of gallons.")

# otherwise, add the requested number of gallons to the fuel level

else:

fuel\_level += gallons

# if the fuel level exceeds the tank capacity, fill the tank and print a message

if fuel\_level > tank\_size:

fuel\_level = tank\_size

print(f"You added {gallons:.2f} gallons, but your tank is full. You have {fuel\_level:.2f} gallons in your tank.")

# otherwise, print a message indicating the number of gallons added and the current fuel level

else:

print(f"You added {gallons:.2f} gallons. You have {fuel\_level:.2f} gallons in your tank.")

# return the updated fuel level

return fuel\_level

# function to show the current fuel level

def show\_fuel\_level(fuel\_level):

# print a message indicating the current fuel level

print(f"You have {fuel\_level:.2f} gallons of gas left.")

# function to log the action performed and its result in a file

def log\_action(action, result):

# open a file named "LogFuel.txt" in append mode and assign it to the variable "logfile"

with open("Desktop\LogFuel.txt", "a") as logfile:

# write a line in the file indicating the action and its result

logfile.write(f"{action}: {result}\n")

# main function that runs the fuel management program

def main():

# ask the user to enter the fuel efficiency of the car in miles per gallon

fuel\_efficiency = float(input("Please enter the car's fuel efficiency (miles/gallon):"))

# ask the user to enter the tank size of the car in gallons

tank\_size = float(input("Please enter the size of the fuel tank (in gallons): "))

# initialize the current fuel level to zero

fuel\_level = 0.0

# start a loop that runs until the user chooses to exit

while True:

# display the available

# Print the menu options

print("What would you like to do:")

print("1. See current fuel level")

print("2. Drive")

print("3. Add gas")

print("4. Exit")

# Get the user's choice

choice = input()

# If the user chooses to see the current fuel level, call the show\_fuel\_level function and log the action

if choice == "1":

show\_fuel\_level(fuel\_level)

log\_action("Current Fuel Level", fuel\_level)

# If the user chooses to drive, the drive function is called and the user is prompted to enter

# the number of miles they want to drive. The fuel level is updated based on the miles driven,

# and the log\_action function is called to record this action in a log file.

elif choice == "2":

miles\_to\_drive = float(input("How many miles do you want to drive? "))

fuel\_level = drive(miles\_to\_drive, fuel\_efficiency, fuel\_level)

log\_action(f"Drive {miles\_to\_drive} miles", fuel\_level)

# If the user chooses to add gas, the add\_gas function is called and the user is prompted to enter

# the number of gallons they want to add. The fuel level is updated based on the amount added,

# and the log\_action function is called to record this action in a log file.

elif choice == "3":

gallons\_to\_add = float(input("How much gas do you want to add? "))

fuel\_level = add\_gas(gallons\_to\_add, tank\_size, fuel\_level)

log\_action(f"Add {gallons\_to\_add} gallons", fuel\_level)

# If the user chooses to exit, break function is called to terminate the program.

elif choice == "4":

break

# If the user enters an invalid choice, a message is displayed and they are prompted to try again.

else:

print("Invalid choice. Please enter a number between 1 and 4.")

print("GoodBye")

# The main function is called to start the program.

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Output :**

Please enter the car's fuel efficiency (miles/gallon):50

Please enter the size of the fuel tank (in gallons): 30

What would you like to do:

1. See current fuel level

2. Drive

3. Add gas

4. Exit

1

You have 0.00 gallons of gas left.

What would you like to do:

1. See current fuel level

2. Drive

3. Add gas

4. Exit

2

How many miles do you want to drive? 3

You don't have enough fuel to drive 3.0 miles. You can drive another 0.0 miles on this gas.

What would you like to do:

1. See current fuel level

2. Drive

3. Add gas

4. Exit

28

Invalid choice. Please enter a number between 1 and 4.

What would you like to do:

1. See current fuel level

2. Drive

3. Add gas

4. Exit

2

How many miles do you want to drive? 30

You don't have enough fuel to drive 30.0 miles. You can drive another 0.0 miles on this gas.

What would you like to do:

1. See current fuel level

2. Drive

3. Add gas

4. Exit

5

Invalid choice. Please enter a number between 1 and 4.

What would you like to do:

1. See current fuel level

2. Drive

3. Add gas

4. Exit

4

GoodBye

**Q2 :**

Q1 : There may be duplicate records in the data. Remove them. How many records do you have now?

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import LogisticRegression

# Load the data

data = pd.read\_csv('Desktop\AluminiumSheetPricesData.csv')

#Length of records before removing duplicate records

ActualLength = len(data)

# Remove duplicate records

data = data.drop\_duplicates()

# Print the number of records

print("Number of records before removing duplicates:", ActualLength)

print("Number of records after removing duplicates:", len(data))

**Output : -**

A black text on a white background

Description automatically generated with low confidence

Q2 : Draw a Histogram of the Price variable. Is it a bell curve? If not, what is it?

import matplotlib.pyplot as plt

# Draw a histogram of the Price variable

plt.hist(data['price'], bins=30)

plt.xlabel('Price')

plt.ylabel('Frequency')

plt.title('Histogram of Price')

plt.show()

**Output :**

A picture containing text, screenshot, diagram, plot

Description automatically generated

A histogram of the Price variable reveals that the distribution is positively right-skewed with a long tail to the right. This means that there are some very high-priced aluminum sheets in the dataset.

Q3 : Do some basic data exploration (e.g. using commands as head( ), info( ), describe( ), nunique( ), etc). Which variables will you NOT select?

# Printing the first few records

print(data.head())

# Printing the column information

print(data.info())

# Printing the statistical summary of the data

print(data.describe())

# Printing the number of unique values in each column

print(data.nunique())

**Output :**

A screenshot of a computer

Description automatically generated with medium confidence**A picture containing text, screenshot, font

Description automatically generated**

**Answer :**

Based on the data exploration, the variables that could potentially be excluded from the model are:

From the output, we can see that the grade and thickness variables are strings(objects) and have a large number of unique values compared to the other variables.

Thickness: it has only 8 unique values, and it's not clear how it relates to the other variables. Grade: it has only 7 unique values, and it's not clear how it relates to the other variables. We will choose not to select these variables for our analysis.

Q4 : Are there any outliers in the data? What about missing values? If any of either, treat them.

**A picture containing screenshot, diagram, rectangle, line

Description automatically generated**

**A picture containing text, screenshot, font

Description automatically generated**

**A screenshot of a computer

Description automatically generated with medium confidence**

Q5 : Partition the data into a training set (with 70% of the observations), and testing set (with 30% of the observations) using the random state of 12345 for cross validation.

**Output :**

X\_train shape: (37542, 25)

y\_train shape: (37542,)

X\_test shape: (16090, 25)

y\_test shape: (16090,)

Q6 : On the partitioned data, build the best KNN model. Show the accuracy numbers. (Hint: What is the best value of k? How do you decide the ‘best k’?)

**Output :**

Best k: 4

Accuracy: 0.8428772886647922

**A picture containing line, plot, diagram, text

Description automatically generated**

Train set accuracy: 0.9952053699856162

Test set accuracy: 0.14195152268489744

Q7 : On the partitioned data, build the best logistic regression model. Show the accuracy numbers

Output :

Train set accuracy: 0.09762399445953865

Test set accuracy: 0.09683032939714108

Q8 : Based on the results of k-nearest neighbor, and logistic regression, what is the best model to classify the data? Provide explanation to support your argument.

Answer :

Based on the results, KNN model performs significantly better than logistic regression. The accuracy of the KNN model on the training set is very high, at 0.995, indicating that it is overfitting the data. However, on the test set, it still performs better than logistic regression with an accuracy of 0.142.

On the other hand, the logistic regression model performs very poorly on both the training and test sets, with an accuracy of only 0.098.

Therefore, based on these results, the KNN model is the better choice for classifying the data. However, it is important to note that further analysis and fine-tuning of the models may be necessary to achieve even better results.