Coursework 3 – Barnaby’s Brewhouse

Code:

""" python script to provide sales predictions for Barnaby's Brewhouse by

analysis of previous sales data, and facilitate inventory management and beer

production planning """

# import libraries

from tkinter import filedialog

from tkinter import\*

import matplotlib.pyplot as plt

from datetime import datetime, date

import tkinter.messagebox

from functools import partial

import logging

def get\_data() -> dict:

""" retrieve the sales data from the CSV file """

# open the file and read the data

with open("files/sales.csv", "r") as data\_file:

read\_file = data\_file.readlines()

data = {}

# split the data from each line by commas, and append it to the data

# list

fields = read\_file[0].rstrip("\n").split(",")

for n in range(6):

data[fields[n]] = []

for line in read\_file:

if line != read\_file[0]:

try:

data\_in\_line = line.rstrip("\n").split(",")

data\_in\_line[4] = str(int(data\_in\_line[4]))

for n in range(6):

data[fields[n]].append(data\_in\_line[n])

except:

continue

return data

def plot\_data() -> None:

""" create a graph of the sales data """

month\_data = get\_month\_data()

data = []

xlabel = []

# create the data array for the x axis

for key in month\_data:

data.append(month\_data[key])

xlabel.append(key)

# plot the graph and label the axis

plt.bar(xlabel, data, color="#b3ecff")

plt.ylabel('Sales')

plt.xlabel('Month')

plt.show()

def get\_month\_data() -> dict:

""" find sales data per month """

# get the relevant data from the file

data = get\_data()

sales\_data = data["Quantity ordered"]

date\_data = data["Date Required"]

# add up the quantities sold for each month

month\_data = {}

for q\_index in range(len(sales\_data)):

sales\_no = sales\_data[q\_index]

date = date\_data[q\_index]

month = date[3:6]

try:

month\_data[month] += int(sales\_no)

except:

month\_data[month] = int(sales\_no)

return month\_data

def capitalise\_first(string: str) -> str:

""" capitalise the first letter of a string """

# split the string, capitalise the first letter and rejoin it

string = list(string)

string[0] = string[0].upper()

string = ''.join(string)

return string

def predict\_new() -> int:

""" predict a new value for the sales demand next month """

month\_data = get\_month\_data()

# create a dictionary of each month and the ratio to the next month

ratios\_dict = {}

for month in MONTHS:

month\_index = MONTHS.index(month)

month = capitalise\_first(month)

next\_month = MONTHS[(month\_index+1) % 12]

next\_month = capitalise\_first(next\_month)

ratio = month\_data[next\_month]/month\_data[month]

ratios\_dict[month] = ratio

# get the names of the current month and the next month

calculate\_date = datetime.now()

calculate\_date = str(calculate\_date)[5:7]

previous\_date = MONTHS[(int(calculate\_date)-2) % 12]

current\_date = MONTHS[(int(calculate\_date)-1) % 12]

calculate\_date = MONTHS[int(calculate\_date) % 12]

previous\_date = capitalise\_first(previous\_date)

current\_date = capitalise\_first(current\_date)

calculate\_date = capitalise\_first(calculate\_date)

# get the ratio for both months, and predict the next demand

calculate\_ratio = ratios\_dict[calculate\_date]

current\_ratio = ratios\_dict[current\_date]

# the prediction is last month's data multiplied by the ratio for

# each month

input\_sales = month\_data[previous\_date]

this\_prediction = int(input\_sales)\*current\_ratio

next\_prediction = this\_prediction\*calculate\_ratio

# display these results on the GUI

this\_text.set(str(int(int(this\_prediction)/2))+" litres")

next\_text.set(str(int(int(next\_prediction)/2))+" litres")

def get\_data\_list() -> list:

""" return the same information as get\_data, but in a list form """

# get the data dictionary

data\_dict = get\_data()

# put each item of the dictionary into a list

data\_list\_row = []

for key in data\_dict:

data\_list\_row.append(data\_dict[key])

data\_list = []

# create blank lists for each list to go into the data list

for l in range(len(data\_list\_row[0])):

data\_list.append([])

# put each item from the rows in data\_list\_row into the list. We now have

# the data in the format [[invoice no. 1, customer 1, date 1, etc...],...]

for i in range(len(data\_list\_row)):

for n in range(len(data\_list\_row[i])):

data\_list[n].append(data\_list\_row[i][n])

# check whether the data is in the correct format

for value in data\_list:

try:

value[2] = datetime.strptime(value[2], "%d-%b-%y")

except:

debug = "Date for {} stored in the wrong format".format(value[0])

logger.debug(debug)

# sort the list into chronological order

sorted(data\_list, key=lambda x: data\_list[2])

return data\_list

def get\_beer\_ratios(type: int) -> list:

""" returns the ratios of the beers, and the total number of each beer """

# type is a variable used to represent whether the calculation is to be

# done for this month (0), or the next month (1)

# work out the current month as a nyumber, e.g. January = 1

if type == 0:

comparison\_text = this\_text.get().rstrip(" litres")

comparison = int(comparison\_text)

offset = 0

current\_date = date.today().strftime("%B")

# work out the next month as a number

else:

comparison\_text = next\_text.get().rstrip(" litres")

comparison = int(comparison\_text)

offset = 1

current\_date = (date.today().month+1) % 12

current\_date = datetime(year=1, month=current\_date, day=1)

current\_date = current\_date.strftime("%B")

# get the data as a list

data = get\_data\_list()

beers = {}

# for each sale, if the date matches the date that we are counting for,

# add one to a counter which is stored with that month

for entry in data:

try:

match\_date = MONTHS[(entry[2].month)-1]

except:

day = int(entry[2][0:2])

month = entry[2][3:6]

year = int(entry[2][7:])

month = MONTHS.index(month.lower())

entry[2] = datetime(year=year, month=month, day=day)

match\_date = MONTHS[(entry[2].month)-1]

if current\_date.lower()[0:3] == match\_date:

beer\_name = entry[3]

try:

beers[beer\_name] += 1

except:

beers[beer\_name] = 1

# add all the beers to the beer\_list, and all the corresponding totals to

# total\_list

beer\_list = []

total\_list = []

for beer in beers:

beer\_list.append(beer)

total\_list.append(beers[beer])

# find the total number of sales

total = 0

for value in total\_list:

total += value

# calculate each ratio from month to month

beer\_ratios = []

for value in total\_list:

ratio = comparison \* (value/total)

beer\_ratios.append(int(ratio))

return beer\_ratios, beer\_list

def get\_info(type: int) -> None:

""" creates a window which displays the number of each beers to be made """

# get the beers and ratios

beer\_ratios, beer\_list = get\_beer\_ratios(type)

# create the window

info\_window = Toplevel(bg="#515151")

info\_window.geometry("210x75")

# create labels for each beer

n = 0

for beer in beer\_list:

beer\_1 = Label(info\_window,

text=beer+": "+str(beer\_ratios[n])+" litres")

beer\_1.grid(row=n, column=0)

beer\_1.config(bg="#515151", fg="white", font=("Arial Black", "10"))

n += 1

def verify\_date(date: str) -> datetime:

""" checks whether a date is in the correct format. If it is, return the

date, otherwise, return False"""

try:

# check to see if the date can be parsed correctly

date\_str = date

date\_datetime = datetime.strptime(date\_str, "%d-%b-%Y")

return date

except:

return False

def submit\_sale(window\_widgets: list, sale\_window: tkinter.Toplevel) -> None:

""" the user will enter a sale and this functio will validate it, and update

the system with the effects of the sale """

# get all the values from the input boxes

invoice = window\_widgets[0].get()

customer = window\_widgets[1].get()

date = window\_widgets[2].get()

beer = window\_widgets[3]

gyle = window\_widgets[4].get()

quantity = window\_widgets[5].get()

logger.info("Sale submitted")

# put the values in a list

info\_list = [invoice, customer, date, beer, gyle, quantity]

# check whether all the number fields are numbers

try:

invoice = str(int(invoice))

gyle = str(int(gyle))

quantity = str(int(quantity))

except:

# return an error message

logger.info("Sale submit incorrectly - invalid string")

tkinter.messagebox.showinfo("Error", "Please input a valid entry")

return None

# check the format of the date

date\_verification = verify\_date(date)

if date\_verification is False:

# return an error message

logger.info("Sale submit incorrectly - Date format incorrect")

tkinter.messagebox.showinfo("Error",

("Please enter date in the format \n",

"DD-Month-YYYY"))

return None

# check whether all the boxes are filled in

for item in info\_list:

if item == "":

logger.info("Sale submit incorrectly - Missing value")

tkinter.messagebox.showinfo("Error",

"Please enter a value in all fields")

return None

beer = beer.get()

# add to CSV file

try:

# update the stocks

stock\_updated = update\_stock(beer, -int(quantity))

if stock\_updated:

sale\_window.destroy()

logger.info("Sale logged successfully")

else:

raise ValueError

csv\_file = open("files/sales.csv", "a")

write\_string = ""

for n in range(len(info\_list)-1):

write\_string += str(info\_list[n])+","

write\_string += info\_list[len(info\_list)-1]

write\_string += "\n"

csv\_file.write(write\_string)

csv\_file.close()

except:

logger.error("Failed to log sale - file is being edited")

tkinter.messagebox.showinfo("Error", "Please close the sales file")

def add\_sale() -> None:

""" creates a window where the user can register a sale """

# create the fields and submit button

sale\_window = Toplevel(bg="#515151")

sale\_window.geometry("800x100")

invoice = Label(sale\_window, text="Invoice No.")

invoice.grid(row=1, column=0)

invoice.config(bg="#515151", fg="white", font=("Arial Black", "10"))

invoice\_field = Entry(sale\_window)

invoice\_field.grid(row=2, column=0, sticky="NSEW")

customer = Label(sale\_window, text="Customer")

customer.grid(row=1, column=1, sticky="NSEW")

customer.config(bg="#515151", fg="white", font=("Arial Black", "10"))

customer\_field = Entry(sale\_window)

customer\_field.grid(row=2, column=1, sticky="NSEW")

date = Label(sale\_window, text="Date - DD-Month-YYYY")

date.grid(row=1, column=2, sticky="NSEW")

date.config(bg="#515151", fg="white", font=("Arial Black", "10"))

date\_field = Entry(sale\_window)

date\_field.grid(row=2, column=2, sticky="NSEW")

beer = Label(sale\_window, text="Recipe")

beer.grid(row=1, column=3, sticky="NSEW")

beer.config(bg="#515151", fg="white", font=("Arial Black", "10"))

beer\_select = StringVar()

beer\_field = OptionMenu(sale\_window, beer\_select, "Organic Red Helles",

"Organic Pilsner", "Organic Dunkel")

beer\_field.grid(row=2, column=3, sticky="NSEW")

gyle = Label(sale\_window, text="Gyle No.")

gyle.grid(row=1, column=4, sticky="NSEW")

gyle.config(bg="#515151", fg="white", font=("Arial Black", "10"))

gyle\_field = Entry(sale\_window)

gyle\_field.grid(row=2, column=4, sticky="NSEW")

quantity = Label(sale\_window, text="Quantity")

quantity.grid(row=1, column=5, sticky="NSEW")

quantity.config(bg="#515151", fg="white", font=("Arial Black", "10"))

quantity\_field = Entry(sale\_window)

quantity\_field.grid(row=2, column=5, sticky="NSEW")

window\_widgets = [invoice\_field, customer\_field, date\_field, beer\_select,

gyle\_field, quantity\_field]

submit = Button(sale\_window, text="Submit",

command=lambda: submit\_sale(window\_widgets, sale\_window))

submit.grid(row=3, column=0, sticky="NSEW")

submit.config(fg="#515151", bg="#b3ecff", font=("Arial Black", "10"))

def start\_machine(arguments: list) -> None:

""" changes the status of the machine to be running """

machine = arguments[0]

machine\_label = arguments[1]

# check if the machine is empty, in which case it cannot run.

if machines[machine][1] != "":

machines[machine][5] = "Running"

machine\_label.config(fg="green")

else:

tkinter.messagebox.showinfo("Error", "Machine is empty")

def create\_widgets(view\_window: tkinter.Toplevel) -> None:

""" refresh the widgets in the view window """

# clear the existing widgets

widget\_list = view\_window.winfo\_children()

for item in widget\_list:

if item.winfo\_children():

widget\_list.extend(item.winfo\_children())

for item in widget\_list:

item.grid\_forget()

# make the machine labels and start buttons

n = 2

machine\_labels = []

for machine in machines:

message = machine+": "+machines[machine][0]+": "+machines[machine][1]

machine\_label = Label(view\_window, text=message)

machine\_label.grid(row=n, column=0)

machine\_label.config(bg="#515151", fg="white",

font=("Arial Black", "10"))

machine\_button = Button(view\_window, text="Start")

machine\_button.grid(row=n, column=1)

machine\_button.config(command=partial(start\_machine,

[machine, machine\_label]),

fg="#515151", bg="#b3ecff",

font=("Arial Black", "10"))

machine\_labels.append(machine\_label)

n += 1

# make the batch labels and buttons

batch\_title = Label(view\_window, text="Batch Status:")

batch\_title.grid(row=n, column=0, sticky="W")

batch\_title.config(bg="#515151", fg="white", font=("Arial Black", "10"))

n += 1

for batch in status:

if status[batch] != []:

message = batch+": "+status[batch][0]

batch\_label = Label(view\_window, text=message)

batch\_label.grid(row=n, column=0)

batch\_label.config(bg="#515151", fg="white",

font=("Arial Black", "10"))

batch\_button = Button(view\_window, text="Next Stage")

batch\_button.config(command=partial(next\_stage,

[batch, batch\_label,

batch\_button, view\_window,

machine\_labels]),

fg="#515151", bg="#b3ecff",

font=("Arial Black", "10"))

batch\_button.grid(row=n, column=1, sticky="NSEW")

n += 1

add\_button = Button(view\_window, text="Edit Brewing Process",

command=lambda: add\_process(machine\_labels,

view\_window))

add\_button.grid(row=0, column=0, sticky="NSEW")

add\_button.config(fg="#515151", bg="#b3ecff", font=("Arial Black", "10"))

brewers\_title = Label(view\_window, text="Brewing Tank Status:")

brewers\_title.config(bg="#515151", fg="white", font=("Arial Black", "10"))

brewers\_title.grid(row=1, column=0, sticky="W")

def submit\_entry(view\_window: tkinter.Toplevel, beer: str, gyle: str,

quantity: str) -> None:

""" validate the entry of the beer into a machine and change the status

of the batch to be the first stage of the brewing process """

# raise an error if there is an empty box

if beer == "" or gyle == "" or quantity == "":

tkinter.messagebox.showinfo("Error, please fill out all fields")

# raise an error if the gyle number is not unique

elif gyle in status:

tkinter.messagebox.showinfo("Error, gyle number must be unique")

else:

# is the gyle number a number

try:

gyle = str(int(gyle))

# if no error, then set the batch to hot brew, and refresh

# the window

create\_widgets(view\_window)

save()

except:

# raise an error

logger.info("Batch load unsuccesful - Gyle is a number")

tkinter.messagebox.showinfo("Error, gyle number must be a number")

# repeat for the quantity

try:

quantity = str(int(quantity))

status[gyle] = ["Hot Brew", beer, quantity]

logger.info("Batch {} loaded successfully".format(gyle))

create\_widgets(view\_window)

save()

except:

logger.info("Batch load unsuccesful - Quantity is a number")

tkinter.messagebox.showinfo("Error, quantity must be a number")

def machine\_clear(batch: str, view\_window: tkinter.Toplevel,

machine\_labels: list) -> None:

""" clear the machine and set its status to running """

# clear the machine and set its status to running

for machine in machines:

if batch in machines[machine][1]:

machine\_list = [machine for machine in machines]

machines[machine][1] = ""

machines[machine][2] = ""

machines[machine][4] = 0

machines[machine][5] = "Waiting"

index = machine\_list.index(machine)

machine\_labels[index].config(fg="white")

logger.info("Machine {} was cleared of {}".format(machine, batch))

# save the files and refresh the window

save()

create\_widgets(view\_window)

def find\_machine(batch: str, view\_window: tkinter.Toplevel, type: str) -> str:

""" find an appropraite machine for the batch """

# find the volumne of the batch

batch\_info = status[batch]

volume = float(batch\_info[2])\*0.5

# find machines that aren't waiting, and can perform the correct job

possible\_machines = []

for machine in machines:

if type in machines[machine][0]:

if machines[machine][5] == "Waiting":

possible\_machines.append(machine)

machines\_list = [machine for machine in possible\_machines]

n = 0

# for every machine in possible machines, check whether it can take the job

while n < len(machines\_list):

info = machines[machines\_list[n]]

# if there is no space in the machine, move on

if float(info[4])+volume > float(info[3]):

n += 1

continue

keys = [machine for machine in machines]

machine\_index = keys.index(machine)

# if the machine is empty, add it

if info[2] == "":

return machines\_list[n]

# if the machine has the same recipe in it, add it

elif batch\_info[0] in info[2]:

return machines\_list[n]

n += 1

logger.info("All machines full or in use")

tkinter.messagebox.showinfo("Error", "No Machines Available")

return "None"

def machine\_load(batch: str, view\_window: tkinter.Toplevel, type: str) -> None:

""" load a batch into a machine """

# get an available

machine = find\_machine(batch, view\_window, type)

if machine != "None":

# change the attributes of the machine

new\_info = [batch, status[batch][1], status[batch][2]]

machines[machine][1] += " "+batch

machines[machine][2] += status[batch][1]

status\_2 = int(status[batch][2])

machines\_4 = float(machines[machine][4])

machines[machine][4] = machines\_4+status\_2

# save the changes and update the window

logger.info("{} loaded into {}".format(batch, machine))

save()

create\_widgets(view\_window)

def next\_stage(arguments: list) -> None:

""" move the batch onto the next stage of the brewing process """

# unpack the arguments

batch = arguments[0]

label = arguments[1]

button = arguments[2]

view\_window = arguments[3]

machine\_labels = arguments[4]

# find the current stage of the batch

current\_stage = status[batch][0]

if current\_stage == "Bottling":

# remove the batch from the window

update\_stock(status[batch][1], status[batch][2])

status.pop(batch)

button.grid\_forget()

label.grid\_forget()

save()

return None

# find the stage

stage\_index = PROCESS.index(current\_stage)

next\_process = PROCESS[stage\_index+1]

if next\_process == "Bottling":

for machine in machines:

current\_machine = machines[machine]

if current\_machine[5] == "Waiting" and batch in current\_machine[1]:

tkinter.messagebox.showinfo("Error. Machine is not running")

logger.info("{} finished brewing".format(batch))

return None

# remove it from the machine

machine\_clear(batch, view\_window, machine\_labels)

status[batch][0] = next\_process

new\_text = batch+": "+next\_process

label.config(text=new\_text)

logger.info("{} is now at stage {}".format(batch, next\_process))

elif next\_process == "Fermentation":

# load the batch

machine\_load(batch, view\_window, "Fermenter")

# update the batch's status

status[batch][0] = next\_process

new\_text = batch+": "+next\_process

label.config(text=new\_text)

logger.info("{} is now at stage {}".format(batch, next\_process))

elif next\_process == "Conditioning":

# check if the machine is running

machine\_list = [machine for machine in machines]

for machine in machines:

current\_machine = machines[machine]

if current\_machine[5] == "Waiting" and batch in current\_machine[1]:

tkinter.messagebox.showinfo("Error. Machine is not running")

return None

# load it into a fermenter

machine\_clear(batch, view\_window, machine\_labels)

machine\_load(batch, view\_window, "Conditioner")

# update the batch's status

status[batch][0] = next\_process

new\_text = batch+": "+next\_process

label.config(text=new\_text)

logger.info("{} is now at stage {}".format(batch, next\_process))

create\_widgets(view\_window)

save()

refresh()

def add\_process(machine\_labels: list, view\_window: tkinter.Toplevel) -> None:

""" creates a window to allow the user to add a process """

# make the window

add\_window = Toplevel(bg="#515151")

# make the fields

beer = Label(add\_window, text="Recipe")

beer.grid(row=1, column=0, sticky="NSEW")

beer.config(bg="#515151", fg="white", font=("Arial Black", "10"))

beer\_select = StringVar()

beer\_field = OptionMenu(add\_window, beer\_select, "Organic Red Helles",

"Organic Pilsner", "Organic Dunkel")

beer\_field.grid(row=2, column=0, sticky="NSEW")

gyle = Label(add\_window, text="Gyle No.")

gyle.grid(row=1, column=1, sticky="NSEW")

gyle.config(bg="#515151", fg="white", font=("Arial Black", "10"))

gyle\_field = Entry(add\_window)

gyle\_field.grid(row=2, column=1, sticky="NSEW")

quantity = Label(add\_window, text="Quantity")

quantity.grid(row=1, column=2, sticky="NSEW")

quantity.config(bg="#515151", fg="white", font=("Arial Black", "10"))

quantity\_field = Entry(add\_window)

quantity\_field.grid(row=2, column=2, sticky="NSEW")

info\_widget\_list = [beer\_field, gyle\_field, quantity\_field]

submit = Button(add\_window, text="Submit",

command=lambda: submit\_entry(view\_window,

beer\_select.get(),

gyle\_field.get(),

quantity\_field.get()))

submit.config(fg="#515151", bg="#b3ecff", font=("Arial Black", "10"))

submit.grid(row=3, column=0, sticky="NSEW")

def view\_process() -> None:

""" creates a window to view the machines and batches in """

view\_window = Toplevel(bg="#515151")

create\_widgets(view\_window)

def save() -> None:

''' save the contents of the machines and batches so that they can be

restored later '''

# clear both files

machines\_write = open("files/machines.txt", "w")

machines\_write.close()

batches\_write = open("files/batches.txt", "w")

batches\_write.close()

# update the files

machines\_file = open("files/machines.txt", "a")

for machine in machines:

machine\_string = machine

for m in machines[machine]:

machine\_string += ","+str(m)

machines\_file.write(machine\_string+"\n")

machines\_file.close()

batches\_file = open("files/batches.txt", "a")

for batch in status:

batch\_string = batch

for b in status[batch]:

batch\_string += ","+str(b)

batches\_file.write(batch\_string+"\n")

batches\_file.close()

logger.info("Saved")

def update\_stock(recipe: str, value: str) -> bool:

""" change the value stored in the stock array """

# find the index of the beer

stock\_index = RECIPES.index(recipe)

# update the value

new\_stock = stocks[stock\_index] - int(value)

if new\_stock < 0:

logger.error("Stock cannot be negative")

return False

stocks[stock\_index] += int(value)

get\_stocks()

save\_stock()

logger.info("Stock updated {} {} {}".format("Red Helles: "+str(stocks[0]),

"Pilsner: "+str(stocks[1]),

"Dunkel: "+str(stocks[2])))

return True

def recommend\_beer() -> None:

""" recommends the beer to be brewed next """

# get information

beer = ""

ratio\_1, beer\_1 = get\_beer\_ratios(0)

ratio\_2, beer\_2 = get\_beer\_ratios(1)

considered = [False, False, False]

for n in range(3):

# if the stock level is below 100, recommend that it is brewed

if stocks[n] < 100:

beer = RECIPES[n]

recommended.config(text=beer)

# if the stock is less than required, consider it to be recommended

if stocks[n] <= ratio\_1[n]+ratio\_2[n]:

considered[n] = ratio\_1[n]+ratio\_2[n]-stocks[n]

# if the beer hasn't been selected yet

if beer == "":

considered\_length = [num for num in considered if num is not False]

# if no beer is understocked, recommend the beer with the lowest stock

if len(considered\_length) == 0:

beer\_index = stocks.index(min(stocks))

else:

# find the biggest difference between the required amount and the

# current stock, and recommned that beer

maximum = max(considered)

beer\_index = considered.index(maximum)

beer = RECIPES[beer\_index]

recommended.config(text=beer)

logger.info("{} recommended to brew".format(beer))

window.update()

def get\_stocks() -> None:

""" set the text and colour of the stock attributes """

helles\_text.set(str(stocks[0]))

pilsner\_text.set(str(stocks[1]))

dunkel\_text.set(str(stocks[2]))

if stocks[0] < 100:

helles\_stock.config(fg="red")

else:

helles\_stock.config(fg="white")

if stocks[1] < 100:

pilsner\_stock.config(fg="red")

else:

pilsner\_stock.config(fg="white")

if stocks[2] < 100:

dunkel\_stock.config(fg="red")

else:

dunkel\_stock.config(fg="white")

def refresh() -> None:

""" run the functions to refresh the main page """

get\_stocks()

predict\_new()

recommend\_beer()

def save\_stock() -> None:

""" rewrite the stocks file """

stock\_file = open("files/stocks.txt", "w")

stock\_file.write(str(stocks[0])+","+str(stocks[1])+","+str(stocks[2])+"\n")

stock\_file.close()

logger.info("Stock saved")

PROCESS = ["Hot Brew", "Fermentation", "Conditioning", "Bottling"]

# unpack from the stocks file

status = {}

status\_file = open("files/batches.txt", "r")

status\_read = status\_file.readlines()

for line in status\_read:

line\_split = line.rstrip("\n").split(",")

status\_key = line\_split[0]

if status\_key != "":

line\_split.remove(line\_split[0])

status[status\_key] = [line\_split[0], line\_split[1], line\_split[2]]

status\_file.close()

# machines dictionary stores name, type, batches in the machine, capacity and

# current volume

machines = {}

machine\_file = open("files/machines.txt", "r")

machine\_read = machine\_file.readlines()

for line in machine\_read:

line\_split = line.rstrip("\n").split(",")

machine\_key = line\_split[0]

line\_split.remove(line\_split[0])

machines[machine\_key] = line\_split

machine\_file.close()

stock\_file = open("files/stocks.txt", "r")

stock\_read = stock\_file.readlines()

stocks = stock\_read[0].rstrip("\n").split(",")

stocks = [int(stock) for stock in stocks]

stock\_file.close()

MONTHS = ["jan", "feb", "mar", "apr", "may", "jun", "jul", "aug", "sep",

"oct", "nov", "dec"]

FULL\_MONTHS = ["January", "February", "March", "April", "May", "June", "July",

"August", "September", "October", "November", "December"]

info\_widgets = {}

RECIPES = ["Organic Red Helles", "Organic Pilsner", "Organic Dunkel"]

logger = logging.getLogger(\_\_name\_\_)

logger.setLevel(logging.DEBUG)

file\_handler = logging.FileHandler('files/logfile.log')

logging\_format = '%(asctime)s : %(levelname)s : %(name)s : %(message)s'

formatter = logging.Formatter(logging\_format)

file\_handler.setFormatter(formatter)

logger.addHandler(file\_handler)

# GUI setup

window = Tk()

window.geometry("575x325")

window.config(bg="#515151")

window.title("Barnaby's Brewhouse")

title = Label(window, text="BARNABY'S BREWHOUSE")

title.grid(row=0, column=0, sticky="NSEW", columnspan=3)

title.config(bg="#515151", fg="white", font=("Arial", 28))

calculate\_date = datetime.now()

calculate\_date = str(calculate\_date)[5:7]

current\_date = FULL\_MONTHS[(int(calculate\_date)-1) % 12]

calculate\_date = FULL\_MONTHS[int(calculate\_date) % 12]

this\_month = Label(window, text="Demand for "+current\_date+":")

this\_month.grid(row=2, column=0, sticky="NSEW")

this\_month.config(bg="#515151", fg="white", font=("Arial Black", "10"))

this\_info = Button(window, text="More Info", command=lambda: get\_info(0))

this\_info.grid(row=4, column=0, sticky="NSEW")

this\_info.config(fg="#515151", bg="#b3ecff", font=("Arial Black", "10"))

next\_info = Button(window, text="More Info", command=lambda: get\_info(1))

next\_info.grid(row=4, column=1, sticky="NSEW")

next\_info.config(fg="#515151", bg="#b3ecff", font=("Arial Black", "10"))

next\_month = Label(window, text="Demand for "+calculate\_date+":")

next\_month.grid(row=2, column=1, sticky="NSEW")

next\_month.config(bg="#515151", fg="white", font=("Arial Black", "10"))

this\_text = StringVar()

this\_value = Label(window, textvariable=this\_text)

this\_value.grid(row=3, column=0, sticky="NSEW")

this\_value.config(bg="#515151", fg="white", font=("Arial Black", "10"))

next\_text = StringVar()

next\_value = Label(window, textvariable=next\_text)

next\_value.grid(row=3, column=1, sticky="NSEW")

next\_value.config(bg="#515151", fg="white", font=("Arial Black", "10"))

graph\_button = Button(window, text="Plot Monthly Data", command=plot\_data)

graph\_button.grid(row=5, column=0, sticky="NSEW")

graph\_button.config(fg="#515151", bg="#b3ecff", font=("Arial Black", "10"))

manage\_button = Button(window, text="Add Data", command=add\_sale)

manage\_button.grid(row=6, column=0, sticky="NSEW")

manage\_button.config(fg="#515151", bg="#b3ecff", font=("Arial Black", "10"))

view\_button = Button(window, text="View Brewing Process", command=view\_process)

view\_button.grid(row=7, column=0, sticky="NSEW")

view\_button.config(fg="#515151", bg="#b3ecff", font=("Arial Black", "10"))

pilsner\_label = Label(window, text="Organic Pilsner Stock:")

pilsner\_label.grid(row=8, column=0, sticky="NSEW")

pilsner\_label.config(bg="#515151", fg="white", font=("Arial Black", "10"))

pilsner\_text = StringVar()

pilsner\_stock = Label(window, textvariable=pilsner\_text)

pilsner\_stock.grid(row=9, column=0, sticky="NSEW")

pilsner\_stock.config(bg="#515151", fg="white", font=("Arial Black", "10"))

helles\_label = Label(window, text="Organic Red Helles Stock:")

helles\_label.grid(row=8, column=1, sticky="NSEW")

helles\_label.config(bg="#515151", fg="white", font=("Arial Black", "10"))

helles\_text = StringVar()

helles\_stock = Label(window, textvariable=helles\_text)

helles\_stock.grid(row=9, column=1, sticky="NSEW")

helles\_stock.config(bg="#515151", fg="white", font=("Arial Black", "10"))

dunkel\_label = Label(window, text="Organic Dunkel Stock:")

dunkel\_label.grid(row=8, column=2, sticky="NSEW")

dunkel\_label.config(bg="#515151", fg="white", font=("Arial Black", "10"))

dunkel\_text = StringVar()

dunkel\_stock = Label(window, textvariable=dunkel\_text)

dunkel\_stock.grid(row=9, column=2, sticky="NSEW")

dunkel\_stock.config(bg="#515151", fg="white", font=("Arial Black", "10"))

get\_stocks()

recommend\_label = Label(window, text="Recommended beer to brew:")

recommend\_label.grid(row=10, column=0, sticky="NSEW")

recommend\_label.config(bg="#515151", fg="white", font=("Arial Black", "10"))

recommended = Label(window, text="")

recommended.grid(row=10, column=1, sticky="NSEW")

recommended.config(bg="#515151", fg="white", font=("Arial Black", "10"))

refresh\_button = Button(window, text="Refresh", command=refresh)

refresh\_button.grid(row=11, column=0, sticky="NSEW")

refresh\_button.config(fg="#515151", bg="#b3ecff", font=("Arial Black", "10"))

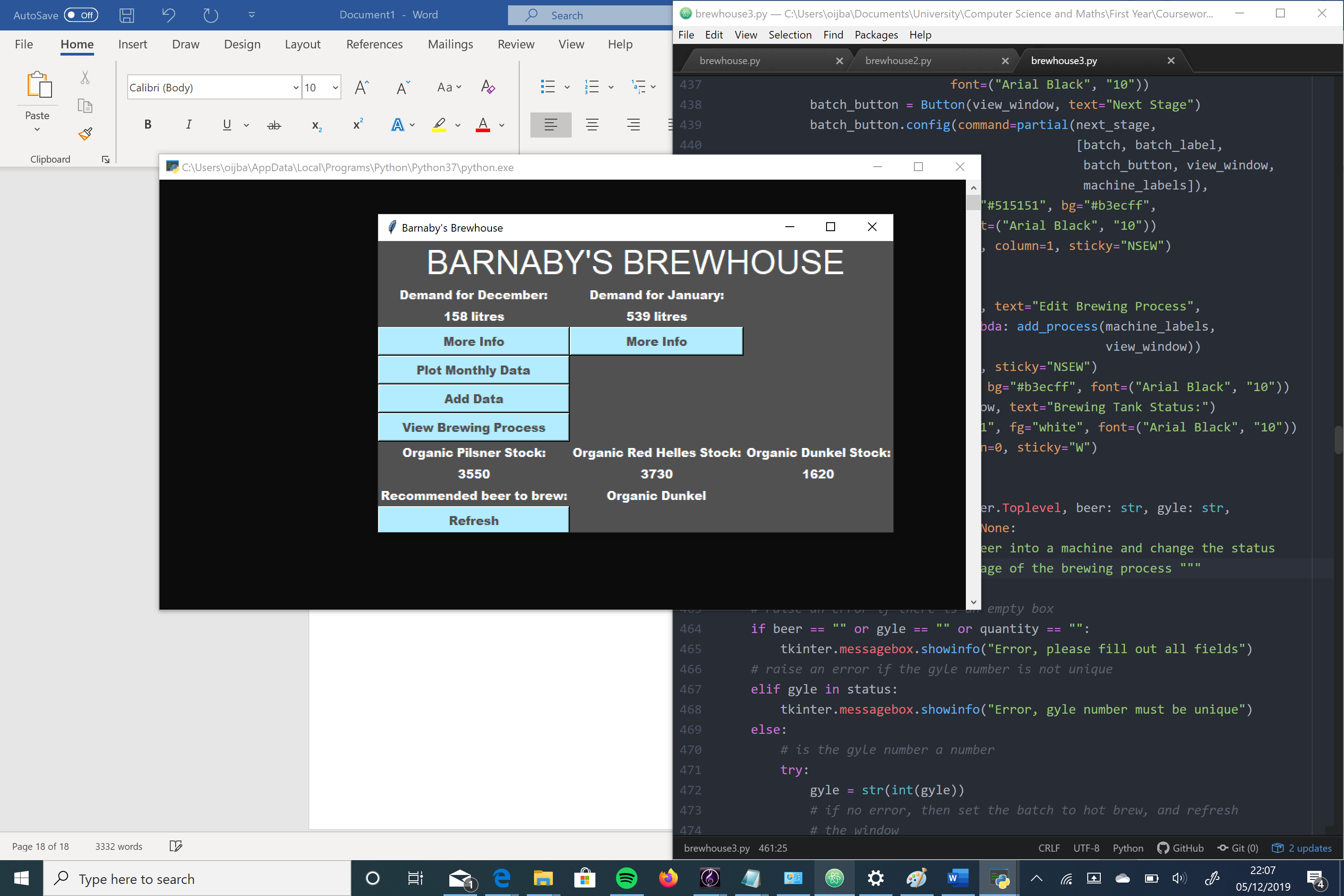
predict\_new()

recommend\_beer()

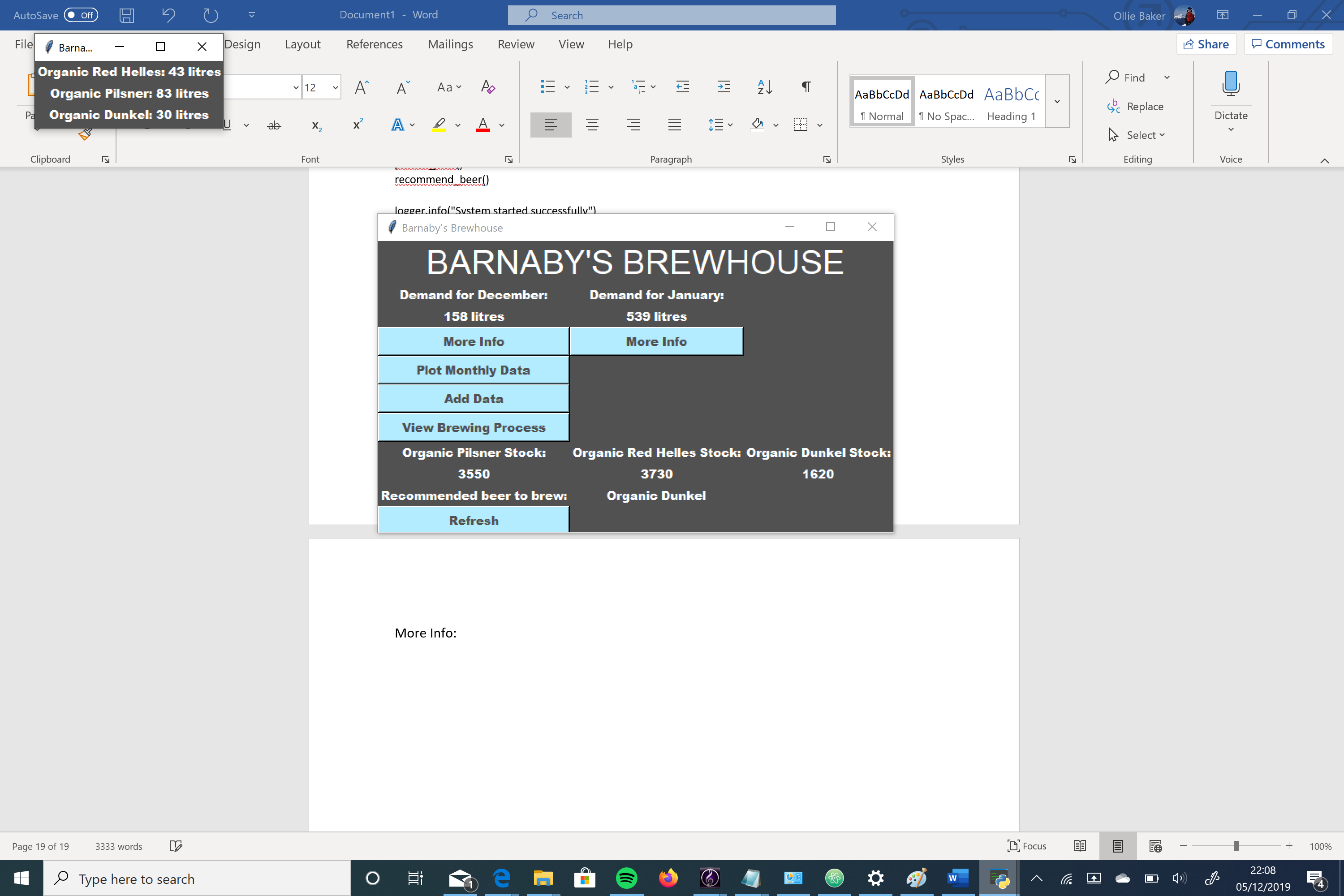
logger.info("System started successfully")

window.mainloop()

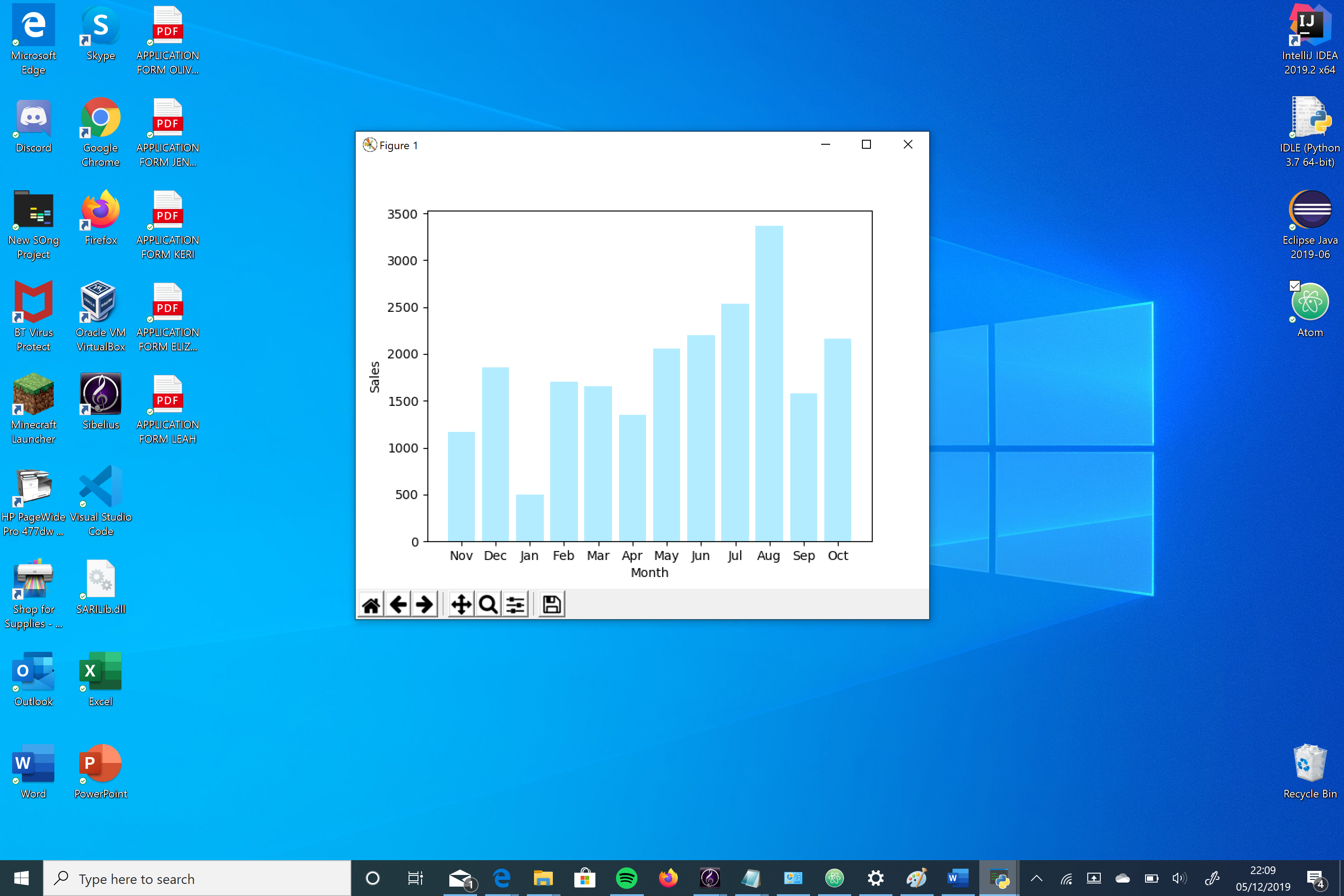
Main Interface:



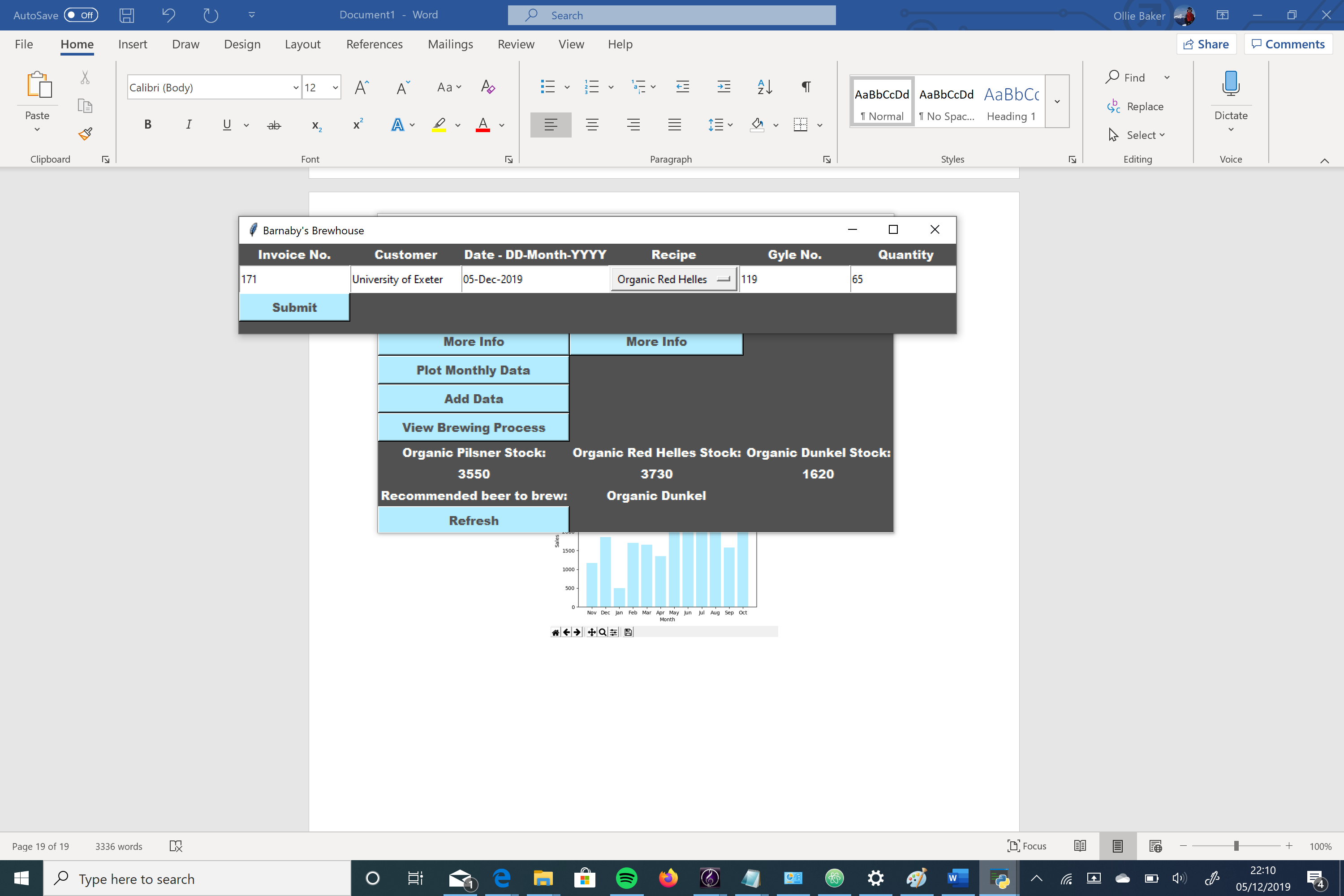
More Info:



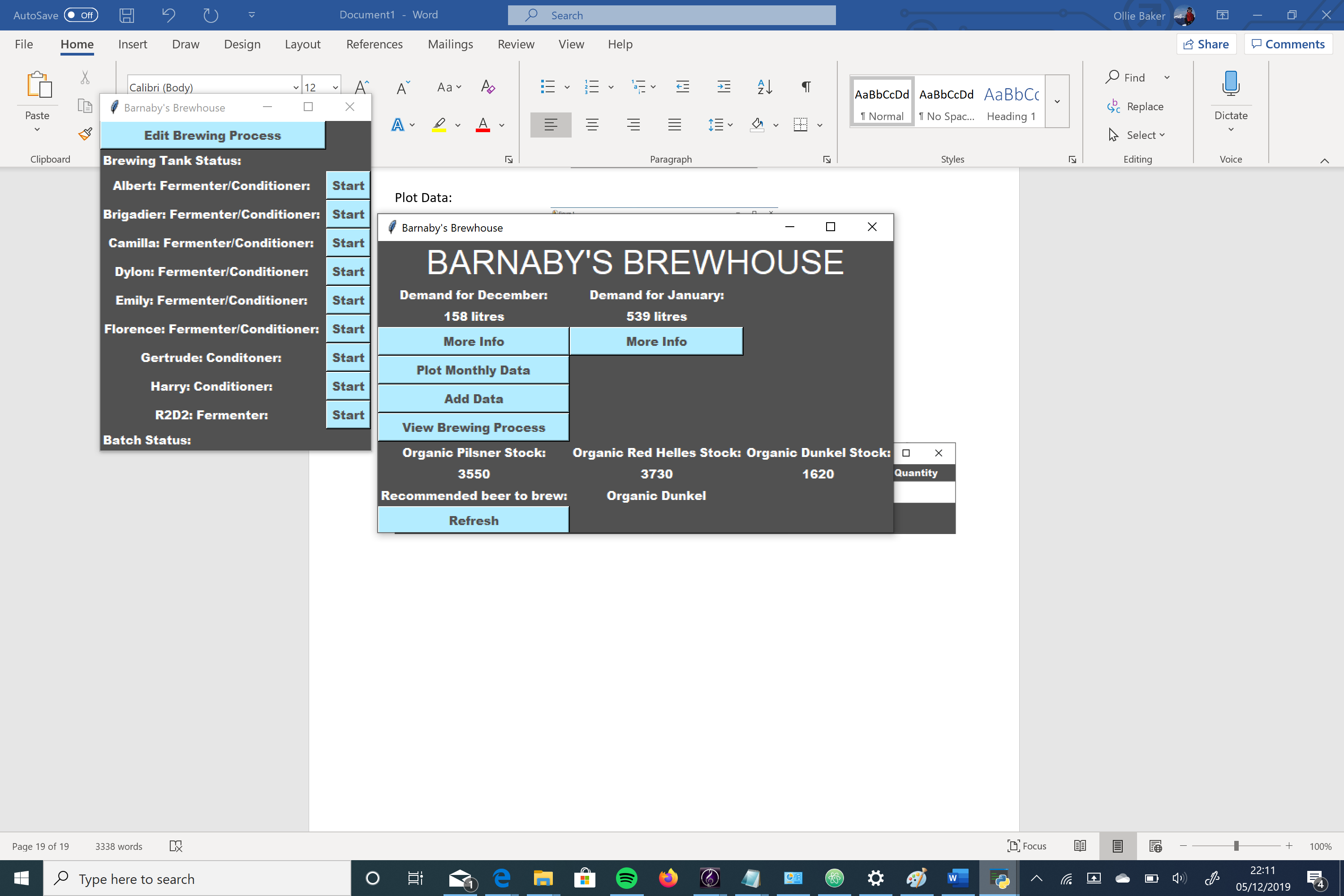
Plot Data:



Add Data:



View Brewing Process:



Edit Brewing Process:

