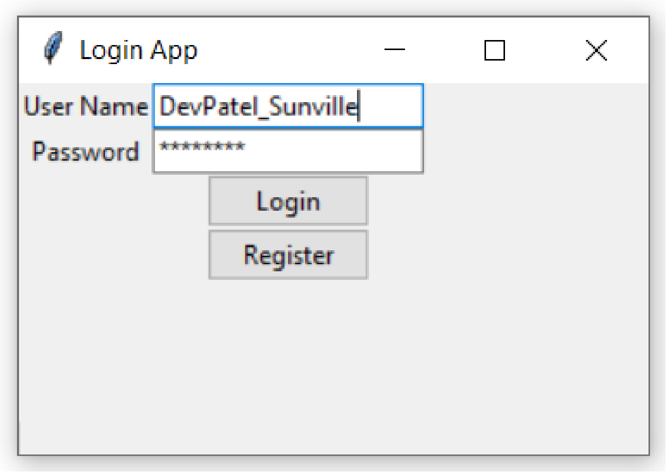
Sunville Properties

Submitted by: Dev Bankimbhai Patel

Date of Submission: 18/07/2022



User Name: DevPatel_Sunville
Password: *********

College Name & Logo:

Dwarkadas J. Sanghvi College of Engineering

Shri Vile Parle Kelavani Mandal's
DWARKADAS J. SANGHVI COLLEGE OF ENGINEERING
(Autonomous College Affiliated to the University of Mumbai)
NAAC Accredited with "A" Grade (CGPA : 3.18)

Certificate Of Completion

This is to certify that, <u>Mr. Dev Bankimbhai Patel</u> has successfully implemented an application designed to study the data and generate insights for Sunville Properties.

The Application has been accepted as a completed project as it meets all the requirements specified.

Acknowledgements:

I would like to express my sincere gratitude to my teacher and mentor Mr. Junaid Khateeb for providing their invaluable guidance, comments and suggestions throughout the course of the training and the project.

Table of Contents

SECTION 1

System Requirement Specification

Sunville Properties is a Colorado based property consultants who have appointed their agents across Major Cities across the globe. They have sub-Companies which take care the business in different countries and are placed in the countries from where they operate from. The Company currently has been using multiple forms of data storage and want to streamline their working using an application, which can help them seamlessly navigate via different forms of storage. Also, the company seeks some insights into the current data and also going further in future. So, it has requested specific modules to be introduced in the system.

PART-A:

The Expected modules are

A1) A Visual Interface to add the data into each of their tables. A login authentication is mandatory for anyone to be able to modify the data.

A2) The company needs an order look up (i.e., search) based on the following criteria,

a) Order number
b) Order Date
c) Customer code

Kindly note: the company might use either one or all of them together at a time.

A3) Generate a report that highlights the balance amounts for all orders in descending order. Do mention the name and code of the agent handling the order. This information needs to be updated in the database.

A4) Which is the country with maximum number of registered customer and what is the collective payment amount and outstanding amount for all these customers collectively.

PART-B:

The company needs the following insights

On selection of the year, system should help them get the following

B1) The total property area sold vs total property are leased in Sq.-M only.

B2) Of the years 2017,2018,2019- which year got maximum leased area in CA and WS countries.

B3) What are the Agent codes of all the agents who have got deals in 'OWNED' categories across the years?

B4) For the city of Chilliwack, which agent has got the maximum deals in leased form.

B5) Compare the performance of all agents based on the area leased and owned for the years 2017,2018 and 2019. Who has been the best performer?

B6) What is the amount of property area sold for the month of July for all the years?

B7) The Company seeks a time series analysis report of the orders received.

Company also seeks any interesting insights or patterns that can help make better business decision. Analysts' viewpoint is welcomed by Sunville properties.

SECTION 2

Technology Used

Python was used in the application for creating the GUI as well as for coding the backend.
Python is a computer programming language often used to build websites and software, automate tasks, and conduct data analysis. Python is a general-purpose language, meaning it can be used to create a variety of different programs and isn't specialized for any specific problems.

Python offers multiple options for developing GUI (Graphical User Interface). Out of all the GUI methods, tkinter is the most commonly used method. It is a standard Python interface to the Tk GUI toolkit shipped with Python. Python with tkinter is the fastest and easiest way to create the GUI applications.

Data analysis and visualisation, too, becomes easy with Python.

The IDE used was Jupyter Notebook.
A Jupyter Notebook provides an easy-to-use, interactive data science environment that doesn't only work as an integrated development environment (IDE), but also as a presentation or educational tool. Jupyter is a way of working with Python inside a virtual "notebook" and is popular with data scientists in large part due to its flexibility.
Jupyter provides a framework and interface that encourages good knowledge management practices by allowing for clear documentation and chronological explanation of the implemented code.

The WampServer was used for database connection.
WampServer is available for Windows working framework. WAMP full form: Windows, Apache, MySQL, and PHP. One noteworthy impediment of WAMP server is that it works just with the Windows framework. It additionally works in the Apache webserver that is good with Windows. To save the data of your site, you depend on a MySQL database. All of these are associated by means of PHP, the programming language.

A database connection is a facility that allows the client software to talk to database server software, whether on the same machine or not. A connection is required to send commands and receive answers, usually in the form of a result set. WampServer allows us to make this connection. The PHP language allows connection to the MySQL database to other databases with three extensions.

It is easy to Use. WAMP makes it easy to code PHP and Creating Databases (in MySQL) in Windows platform.

SECTION 3

Data Provided by the Client

Data Provided for the Database:

The data has been provided in the above format in form of SQL query. There is data for 4 tables namely, agents, customer, company, orders.

Agents:

Agent code, name, phone number, commission, working area and country have been provided. Agent Code cannot be null and are of 4 characters and all attributes are of varchar type except commission which is in decimal format up to 2 decimal places. Country can be a null value.

```
CREATE TABLE IF NOT EXISTS `agents` (
  `AGENT_CODE` varchar(6) NOT NULL DEFAULT '',
  `AGENT_NAME` varchar(40) DEFAULT NULL,
  `WORKING_AREA` varchar(35) DEFAULT NULL,
  `COMMISSION` decimal(10,2) DEFAULT NULL,
  `PHONE_NO` varchar(15) DEFAULT NULL,
  `COUNTRY` varchar(25) DEFAULT NULL,
  PRIMARY KEY (`AGENT_CODE`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `agents`
--

INSERT INTO `agents` (`AGENT_CODE`, `AGENT_NAME`, `WORKING_AREA`, `COMMISSION`, `PHONE_NO`, `COUNTRY`) VALUES
('A007  ', 'Ramasundar                              ', 'Bangalore                          ', '0.15', '077-25814763   ', '\r'),
('A003  ', 'Alex                                    ', 'London                             ', '0.13', '075-12458969   ', '\r'),
('A008  ', 'Alford                                  ', 'New York                           ', '0.12', '044-25874365   ', '\r'),
('A011  ', 'Ravi Kumar                              ', 'Bangalore                          ', '0.15', '077-45625874   ', '\r'),
('A010  ', 'Santakumar                              ', 'Chennai                            ', '0.14', '007-22388644   ', '\r'),
('A012  ', 'Lucida                                  ', 'San Jose                           ', '0.12', '044-52981425   ', '\r'),
('A005  ', 'Anderson                                ', 'Brisban                            ', '0.13', '045-21447739   ', '\r'),
('A001  ', 'Subbarao                                ', 'Bangalore                          ', '0.14', '077-12346674   ', '\r'),
('A002  ', 'Mukesh                                  ', 'Mumbai                             ', '0.11', '029-12358964   ', '\r'),
('A006  ', 'McDen                                   ', 'London                             ', '0.15', '078-22255588   ', '\r'),
('A004  ', 'Ivan                                    ', 'Torento                            ', '0.15', '008-22544166   ', '\r'),
('A009  ', 'Benjamin                                ', 'Hampshair                          ', '0.11', '008-22536178   ', '\r');

-- --------------------------------------------------------
```

Customer:

We have been provided with the Customer Code, Name, City, Country, Working Area, Grade, Phone Number and the code of the Agent they deal with and the finances have been stored in form of Opening Amount, Receive Amount, Payment Amount and Outstanding Amount.

Customer Code is alphanumeric with length 6. Grade is a single digit integer value and all of the four amounts have been stored in decimal format up to two decimal places. Agent code can be used for a relationship between agents and customer tables.

```
CREATE TABLE IF NOT EXISTS `customer` (
  `CUST_CODE` varchar(6) NOT NULL,
  `CUST_NAME` varchar(40) NOT NULL,
  `CUST_CITY` varchar(35) DEFAULT NULL,
  `WORKING_AREA` varchar(35) NOT NULL,
  `CUST_COUNTRY` varchar(20) NOT NULL,
  `GRADE` decimal(10,0) DEFAULT NULL,
  `OPENING_AMT` decimal(12,2) NOT NULL,
  `RECEIVE_AMT` decimal(12,2) NOT NULL,
  `PAYMENT_AMT` decimal(12,2) NOT NULL,
  `OUTSTANDING_AMT` decimal(12,2) NOT NULL,
  `PHONE_NO` varchar(17) NOT NULL,
  `AGENT_CODE` varchar(6) DEFAULT NULL,
  KEY `CUSTCITY` (`CUST_CITY`),
  KEY `CUSTCITY_COUNTRY` (`CUST_CITY`,`CUST_COUNTRY`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `customer` (`CUST_CODE`, `CUST_NAME`, `CUST_CITY`, `WORKING_AREA`, `CUST_COUNTRY`, `GRADE`, `OPENING_AMT`,
`RECEIVE_AMT`, `PAYMENT_AMT`, `OUTSTANDING_AMT`, `PHONE_NO`, `AGENT_CODE`) VALUES
('C00013', 'Holmes', 'London', 'London', 'UK', '2', '6000.00', '5000.00', '7000.00', '4000.00', 'BBBBBBB', 'A003  '),
('C00001', 'Micheal', 'New York', 'New York', 'USA', '2', '3000.00', '5000.00', '2000.00', '6000.00', 'CCCCCCC', 'A008  '),
('C00020', 'Albert', 'New York', 'New York', 'USA', '3', '5000.00', '7000.00', '6000.00', '6000.00', 'BBBBSBB', 'A008  '),
('C00025', 'Ravindran', 'Bangalore', 'Bangalore', 'India', '2', '5000.00', '7000.00', '4000.00', '8000.00', 'AVAVAVA', 'A011  '),
('C00024', 'Cook', 'London  ', 'London', 'UK', '2', '4000.00', '9000.00', '7000.00', '6000.00', 'FSDDSDF', 'A006  '),
('C00015', 'Stuart', 'London  ', 'London', 'UK', '1', '6000.00', '8000.00', '3000.00', '11000.00', 'GFSGERS', 'A003  '),
('C00002', 'Bolt', 'New York', 'New York', 'USA', '3', '5000.00', '7000.00', '9000.00', '3000.00', 'DDNRDRH', 'A008  '),
('C00018', 'Fleming', 'Brisban ', 'Brisban', 'Australia', '2', '7000.00', '7000.00', '9000.00', '5000.00', 'NHBGVFC', 'A005  '),
('C00021', 'Jacks', 'Brisban ', 'Brisban', 'Australia', '1', '7000.00', '7000.00', '7000.00', '7000.00', 'WERTGDF', 'A005  '),
('C00019', 'Yearannaidu', 'Chennai ', 'Chennai', 'India', '1', '8000.00', '7000.00', '7000.00', '8000.00', 'ZZZZBFV', 'A010  '),
('C00005', 'Sasikant', 'Mumbai  ', 'Mumbai', 'India', '1', '7000.00', '11000.00', '7000.00', '11000.00', '147-25896312', 'A002  '),
('C00007', 'Ramanathan', 'Chennai ', 'Chennai', 'India', '1', '7000.00', '11000.00', '9000.00', '9000.00', 'GHRDWSD', 'A010  '),
('C00022', 'Avinash', 'Mumbai  ', 'Mumbai', 'India', '2', '7000.00', '11000.00', '9000.00', '9000.00', '113-12345678', 'A002  '),
('C00004', 'Winston', 'Brisban ', 'Brisban', 'Australia', '1', '5000.00', '8000.00', '7000.00', '6000.00', 'AAAAAAA', 'A005  '),
('C00023', 'Karl', 'London  ', 'London', 'UK', '0', '4000.00', '6000.00', '7000.00', '3000.00', 'AAAABAA', 'A006  '),
('C00006', 'Shilton', 'Torento ', 'Torento', 'Canada', '1', '10000.00', '7000.00', '6000.00', '11000.00', 'DDDDDDD', 'A004  '),
('C00010', 'Charles', 'Hampshair  ', 'Hampshair', 'UK', '3', '6000.00', '4000.00', '5000.00', '5000.00', 'MMMMMMM', 'A009  '),
('C00017', 'Srinivas', 'Bangalore  ', 'Bangalore', 'India', '2', '8000.00', '4000.00', '3000.00', '9000.00', 'AAAAAAB', 'A007  '),
('C00012', 'Steven', 'San Jose', 'San Jose', 'USA', '1', '5000.00', '7000.00', '9000.00', '3000.00', 'KRFYGJK', 'A012  '),
('C00008', 'Karolina', 'Torento ', 'Torento', 'Canada', '1', '7000.00', '7000.00', '9000.00', '5000.00', 'HJKORED', 'A004  '),
('C00003', 'Martin', 'Torento ', 'Torento', 'Canada', '2', '8000.00', '7000.00', '7000.00', '8000.00', 'MJYURFD', 'A004  '),
('C00009', 'Ramesh', 'Mumbai  ', 'Mumbai', 'India', '3', '8000.00', '7000.00', '3000.00', '12000.00', 'Phone No', 'A002  '),
('C00014', 'Rangarappa', 'Bangalore ', 'Bangalore', 'India', '2', '8000.00', '11000.00', '7000.00', '12000.00', 'AAAATGF', 'A001  '),
('C00016', 'Venkatpati', 'Bangalore', 'Bangalore', 'India', '2', '8000.00', '11000.00', '7000.00', '12000.00', 'JRTVFDD', 'A007  '),
('C00011', 'Sundariya', 'Chennai ', 'Chennai', 'India', '3', '7000.00', '11000.00', '7000.00', '11000.00', 'PPHGRTS', 'A010  ');
```

Company:

We have been given Company ID, Name and the city of the entity. Company ID is unique as it is the primary key.

```
CREATE TABLE IF NOT EXISTS `company` (
  `COMPANY_ID` varchar(6) NOT NULL DEFAULT '',
  `COMPANY_NAME` varchar(25) DEFAULT NULL,
  `COMPANY_CITY` varchar(25) DEFAULT NULL,
  PRIMARY KEY (`COMPANY_ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

--
-- Dumping data for table `company`
--

INSERT INTO `company` (`COMPANY_ID`, `COMPANY_NAME`, `COMPANY_CITY`) VALUES
('18', 'Order All', 'Boston\r'),
('15', 'Jack Hill Ltd', 'London\r'),
('16', 'Akas House', 'Delhi\r'),
('17', 'Foilos housing.', 'London\r'),
('19', 'stop-and-buy', 'New York\r')
```

Orders:

This table includes Order number, date, customer code of the customer who bought the product, Agent code of the agent who sold it, the Order description and the total order amount and the advance amount. Relationships between the agents, customers and orders table can be derived based on agent code and customer code respectively.

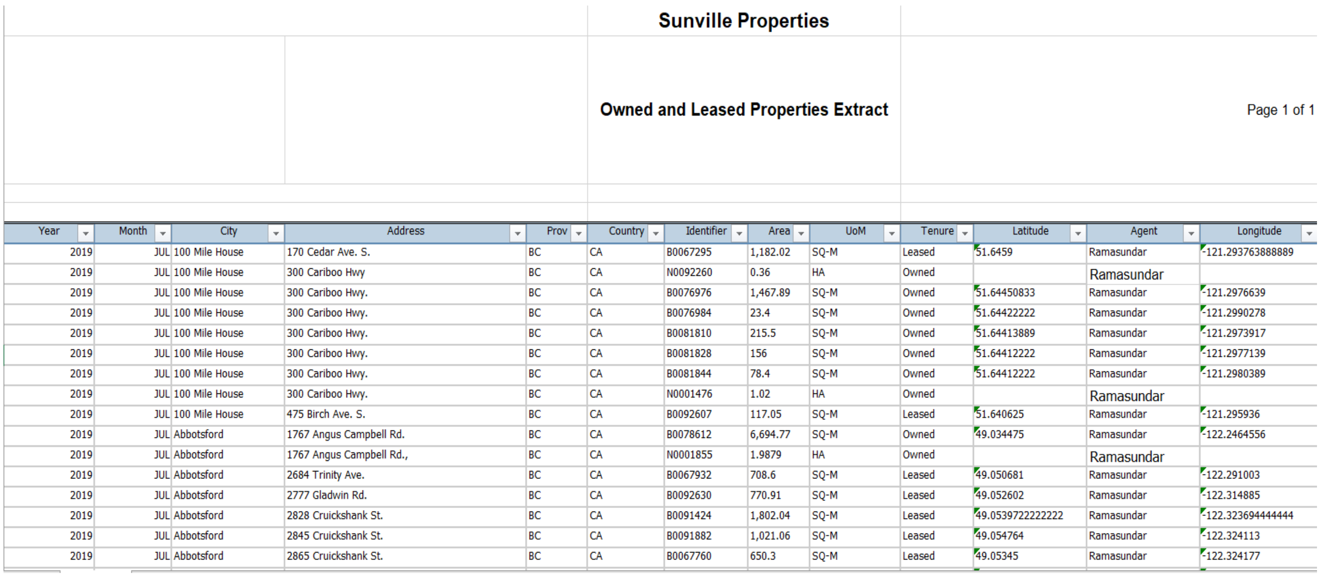
```
CREATE TABLE IF NOT EXISTS `orders` (
  `ORD_NUM` decimal(6,0) NOT NULL,
  `ORD_AMOUNT` decimal(12,2) NOT NULL,
  `ADVANCE_AMOUNT` decimal(12,2) NOT NULL,
  `ORD_DATE` date NOT NULL,
  `CUST_CODE` varchar(6) NOT NULL,
  `AGENT_CODE` varchar(6) NOT NULL,
  `ORD_DESCRIPTION` varchar(60) NOT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `orders` (`ORD_NUM`, `ORD_AMOUNT`, `ADVANCE_AMOUNT`, `ORD_DATE`, `CUST_CODE`, `AGENT_CODE`, `ORD_DESCRIPTION`) VALUES
('200100', '1000.00', '600.00', '2008-01-08', 'C00015', 'A003  ', 'SOD\r'),
('200110', '3000.00', '500.00', '2008-04-15', 'C00019', 'A010  ', 'SOD\r'),
('200107', '4500.00', '900.00', '2008-08-30', 'C00007', 'A010  ', 'SOD\r'),
('200112', '2000.00', '400.00', '2008-05-30', 'C00016', 'A007  ', 'SOD\r'),
('200113', '4000.00', '600.00', '2008-06-10', 'C00022', 'A002  ', 'SOD\r'),
('200102', '2000.00', '300.00', '2008-05-25', 'C00012', 'A012  ', 'SOD\r'),
('200114', '3500.00', '2000.00', '2008-08-15', 'C00002', 'A008  ', 'SOD\r'),
('200122', '2500.00', '400.00', '2008-09-16', 'C00003', 'A004  ', 'SOD\r'),
('200118', '500.00', '100.00', '2008-07-20', 'C00023', 'A006  ', 'SOD\r'),
('200119', '4000.00', '700.00', '2008-09-16', 'C00007', 'A010  ', 'SOD\r'),
('200121', '1500.00', '600.00', '2008-09-23', 'C00008', 'A004  ', 'SOD\r'),
('200130', '2500.00', '400.00', '2008-07-30', 'C00025', 'A011  ', 'SOD\r'),
('200134', '4200.00', '1800.00', '2008-09-25', 'C00004', 'A005  ', 'SOD\r'),
('200115', '2000.00', '1200.00', '2008-02-08', 'C00013', 'A013  ', 'SOD\r'),
('200108', '4000.00', '600.00', '2008-02-15', 'C00008', 'A004  ', 'SOD\r'),
('200103', '1500.00', '700.00', '2008-05-15', 'C00021', 'A005  ', 'SOD\r'),
('200105', '2500.00', '500.00', '2008-07-18', 'C00025', 'A011  ', 'SOD\r'),
('200109', '3500.00', '800.00', '2008-07-30', 'C00011', 'A010  ', 'SOD\r'),
('200101', '3000.00', '1000.00', '2008-07-15', 'C00001', 'A008  ', 'SOD\r'),
('200111', '1000.00', '300.00', '2008-07-10', 'C00020', 'A008  ', 'SOD\r'),
('200104', '1500.00', '500.00', '2008-03-13', 'C00006', 'A004  ', 'SOD\r'),
('200106', '2500.00', '700.00', '2008-04-20', 'C00005', 'A002  ', 'SOD\r'),
('200125', '2000.00', '600.00', '2008-10-10', 'C00018', 'A005  ', 'SOD\r'),
('200117', '800.00', '200.00', '2008-10-20', 'C00014', 'A001  ', 'SOD\r'),
('200123', '500.00', '100.00', '2008-09-16', 'C00022', 'A002  ', 'SOD\r'),
('200120', '500.00', '100.00', '2008-07-20', 'C00009', 'A002  ', 'SOD\r'),
('200116', '500.00', '100.00', '2008-07-13', 'C00010', 'A009  ', 'SOD\r'),
('200124', '500.00', '100.00', '2008-06-20', 'C00017', 'A007  ', 'SOD\r'),
('200126', '500.00', '100.00', '2008-06-24', 'C00022', 'A002  ', 'SOD\r'),
('200129', '2500.00', '500.00', '2008-07-20', 'C00024', 'A006  ', 'SOD\r'),
('200127', '2500.00', '400.00', '2008-07-20', 'C00015', 'A003  ', 'SOD\r'),
('200128', '3500.00', '1500.00', '2008-07-20', 'C00009', 'A002  ', 'SOD\r'),
('200135', '2000.00', '800.00', '2008-09-16', 'C00007', 'A010  ', 'SOD\r'),
('200131', '900.00', '150.00', '2008-08-26', 'C00012', 'A012  ', 'SOD\r'),
('200133', '1200.00', '400.00', '2008-06-29', 'C00009', 'A002  ', 'SOD\r'),
('200132', '4000.00', '2000.00', '2008-08-15', 'C00013', 'A013  ', 'SOD\r');
```

This data has been fed in the WampServer to create a database ‘sunville’ which includes these 4 tables and a new table called ‘balance’ which has the data of amount yet to be paid by the customer on a particular order, which will be explained further in detail.

Data for Insights in Part-B:

The data has been provided in the following format. The first 6 lines are introduction to the dataset. The dataset includes following attributes- Year, Month, City, Address, Prov (Province), Country, Identifier, Area, UoM (Units of Measurement), Tenure, Latitude, Longitude and Agent.

This gives information about the year and month of the sale, the city, province, country, address, latitude and longitude of the property, the area and unit of measurement of the given area of property, an identifier and tenure which indicates whether the property has been Leased or Owned.



Sunville Properties

Owned and Leased Properties Extract

Page 1 of 1

Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
2019	JUL	100 Mile House	170 Cedar Ave. S.	BC	CA	B0067295	1,182.02	SQ-M	Leased	51.6459	Ramasundar	-121.293763888889
2019	JUL	100 Mile House	300 Cariboo Hwy	BC	CA	N0092260	0.36	HA	Owned		Ramasundar	
2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076976	1,467.89	SQ-M	Owned	51.64450833	Ramasundar	-121.2976639
2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076984	23.4	SQ-M	Owned	51.64422222	Ramasundar	-121.2990278
2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0081810	215.5	SQ-M	Owned	51.64413889	Ramasundar	-121.2973917
2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0081828	156	SQ-M	Owned	51.64412222	Ramasundar	-121.2977139
2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0081844	78.4	SQ-M	Owned	51.64412222	Ramasundar	-121.2980389
2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	N0001476	1.02	HA	Owned		Ramasundar	
2019	JUL	100 Mile House	475 Birch Ave. S.	BC	CA	B0092607	117.05	SQ-M	Leased	51.640625	Ramasundar	-121.295936
2019	JUL	Abbotsford	1767 Angus Campbell Rd.	BC	CA	B0078612	6,694.77	SQ-M	Owned	49.034475	Ramasundar	-122.2464556
2019	JUL	Abbotsford	1767 Angus Campbell Rd.,	BC	CA	N0001855	1.9879	HA	Owned		Ramasundar	
2019	JUL	Abbotsford	2684 Trinity Ave.	BC	CA	B0067932	708.6	SQ-M	Leased	49.050681	Ramasundar	-122.291003
2019	JUL	Abbotsford	2777 Gladwin Rd.	BC	CA	B0092630	770.91	SQ-M	Leased	49.052602	Ramasundar	-122.314885
2019	JUL	Abbotsford	2828 Cruickshank St.	BC	CA	B0091424	1,802.04	SQ-M	Leased	49.0539722222222	Ramasundar	-122.323694444444
2019	JUL	Abbotsford	2845 Cruickshank St.	BC	CA	B0091882	1,021.06	SQ-M	Leased	49.054764	Ramasundar	-122.324113
2019	JUL	Abbotsford	2865 Cruickshank St.	BC	CA	B0067760	650.3	SQ-M	Leased	49.05345	Ramasundar	-122.324177

The Excel Sheet has 1215 lines of information and 1206 data instances.

This Excel Sheet can be accessed using the following link:

https://docs.google.com/spreadsheets/d/1nnZxJoVklbibDGO3ZEeKub0BtnmqaGSa/edit?usp=sharing&ouid=114429766290466708338&rtpof=true&sd=true

The WampServer has been used for storage of database.

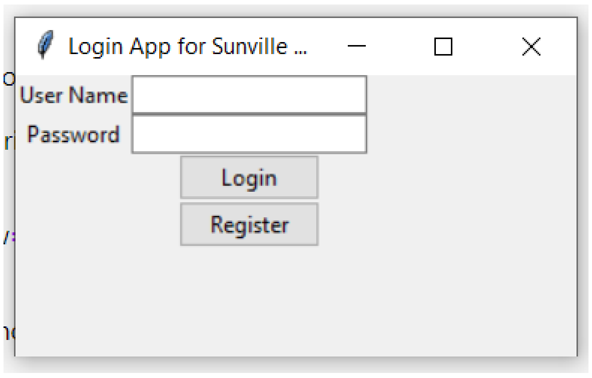
SECTION 4

Screenshots

Part-A:

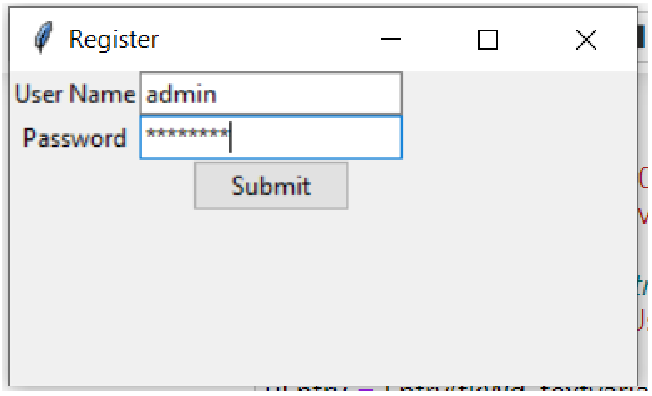
A1) Login Authentication and Add to Tables

1. Login Pop Up:



2. Register Screen:

   Using the username: admin and password: admin123

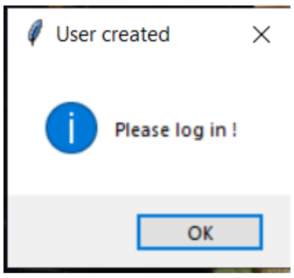


Register

User Name admin

Password ********

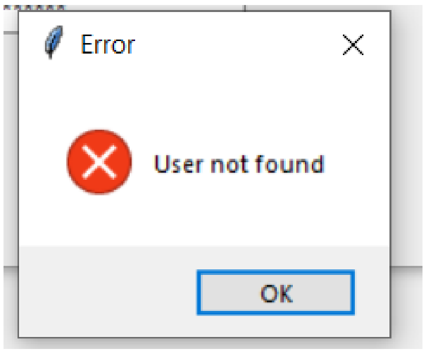
Successful Register



User created

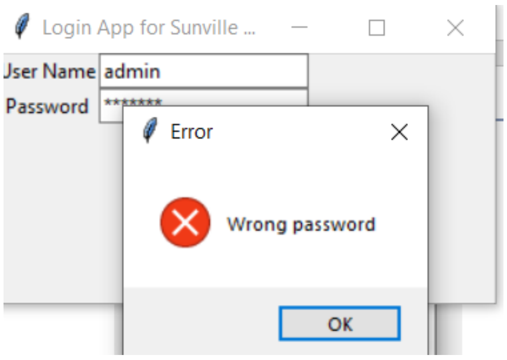
Please log in !

Login with Incorrect User Name:


Error

User not found

Login with Incorrect Password:



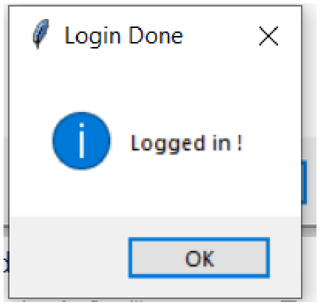
Jser Name admin

Password *******

Error

Wrong password

Successful Login:


Login Done

Logged in !

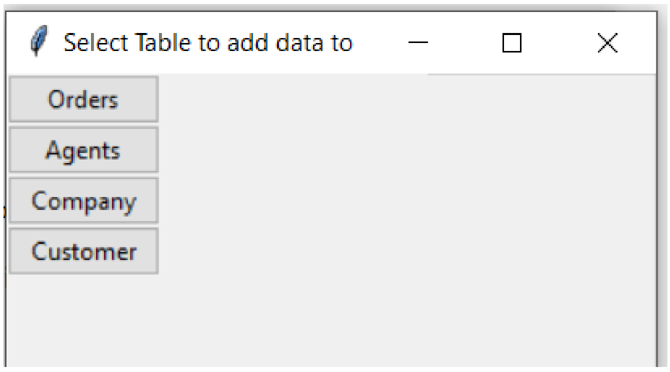
Code for Login Authentication:

```
#Importing Required Libraries
import tkinter
from tkinter import *
from tkinter import messagebox
from functools import partial
import sqlite3
import pymysql
from venv import create
from tkinter.ttk import *
from tkcalendar import Calendar, DateEntry
import re
from datetime import datetime
import pandas as pd
from pandastable import Table, TableModel

def Login(username, password):
    c.execute(search_by_name, (username.get(),))
    print()
    user = c.fetchall()
    if len(user) == 0:
```

```python
         messagebox.showerror("Error", "User not found")
    else:
        c.execute(validate , (username.get() , password.get()))
        validated = c.fetchall()
        if len(validated) == 0:
            messagebox.showerror("Error", "Wrong password")
        else:
            messagebox.showinfo("Login Done", "Logged in !")

            options()

def Register():

    Window = Toplevel(tkWd)

    def Submit(username, password):
        c.execute(create_user, (username.get(), password.get(),))
        conn.commit()
        if messagebox.showinfo("User created" , "Please log in !"):
            Window.withdraw()

    Window.title("Register")

    # sets the geometry of toplevel
    Window.geometry("300x150")
    # username label and text entry box
    uLabel = Label(Window, text="User Name").grid(row=0, column=0)
    username = StringVar()
    uEntry = Entry(Window, textvariable=username).grid(row=0, column=1)

    # password label and password entry box
    pLabel = Label(Window,text="Password").grid(row=1, column=0)
    password = StringVar()
    pEntry = Entry(Window, textvariable=password, show='*').grid(row=1, column=1)
    Submit = partial(Submit, username, password)
    Button(Window, text="Submit", command=Submit).grid(row=4, column=1)

conn = sqlite3.connect('cred.db')
c = conn.cursor()
create_table = """CREATE TABLE IF NOT EXISTS user(username TEXT NOT NULL, password TEXT NOT
NULL)"""
create_user = """INSERT INTO user(username,password) VALUES(?,?)"""
search_by_name = """SELECT * FROM user where username=?"""
validate = """SELECT * FROM user where username=? and password=?"""
c.execute(create_table)

# window
tkWd = Tk()
tkWd.geometry('300x150+600+300')
tkWd.title('Login App for Sunville Properties')

# username label and text entry box
uLabel = Label(tkWd, text="User Name").grid(row=0, column=0)
username = StringVar()
uEntry = Entry(tkWd, textvariable=username).grid(row=0, column=1)

# password label and password entry box
pLabel = Label(tkWd,text="Password").grid(row=1, column=0)
```

```
password = StringVar()

pEntry = Entry(tkWd, textvariable=password, show='*').grid(row=1, column=1)

Login = partial(Login, username, password)

# login , register buttons
loginButton = Button(tkWd, text="Login", command=Login).grid(row=4, column=1)
regButton = Button(tkWd, text="Register", command=Register).grid(row=5, column=1)

tkWd.mainloop()
```

3. Menu After Successful Login:

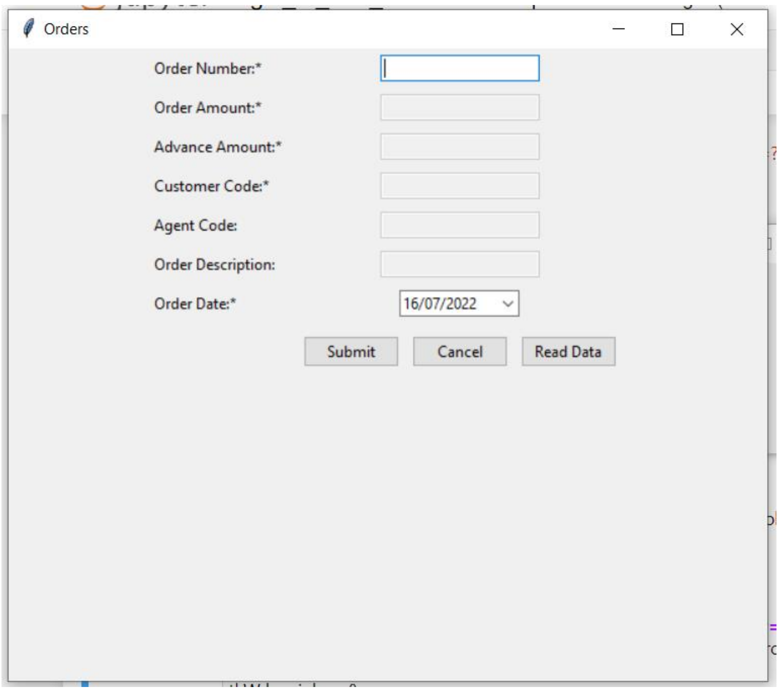


```
def options():
    tkWd = Tk()
    tkWd.geometry('200x150+600+300')
    tkWd.title('Select Table to add data to')

    ordersButton = Button(tkWd, text="Orders", command=orders).grid(row=2, column=1)
    agentsButton = Button(tkWd, text="Agents", command=agents).grid(row=3, column=1)
    companyButton = Button(tkWd, text="Company", command=company).grid(row=4, column=1)
    customerButton = Button(tkWd, text="Customer", command=customer).grid(row=5, column=1)

    tkWd.mainloop()

def orders():
    if(root):
        root.mainloop()
    %run addorders
def agents():
    if(root):
        root.mainloop()
    %run addagents
def company():
    if(root):
        root.mainloop()
    %run addcompany
def customer():
    if(root):
        root.mainloop()
    %run addcustomer
```

4. Add to tables:

a) Add to orders table

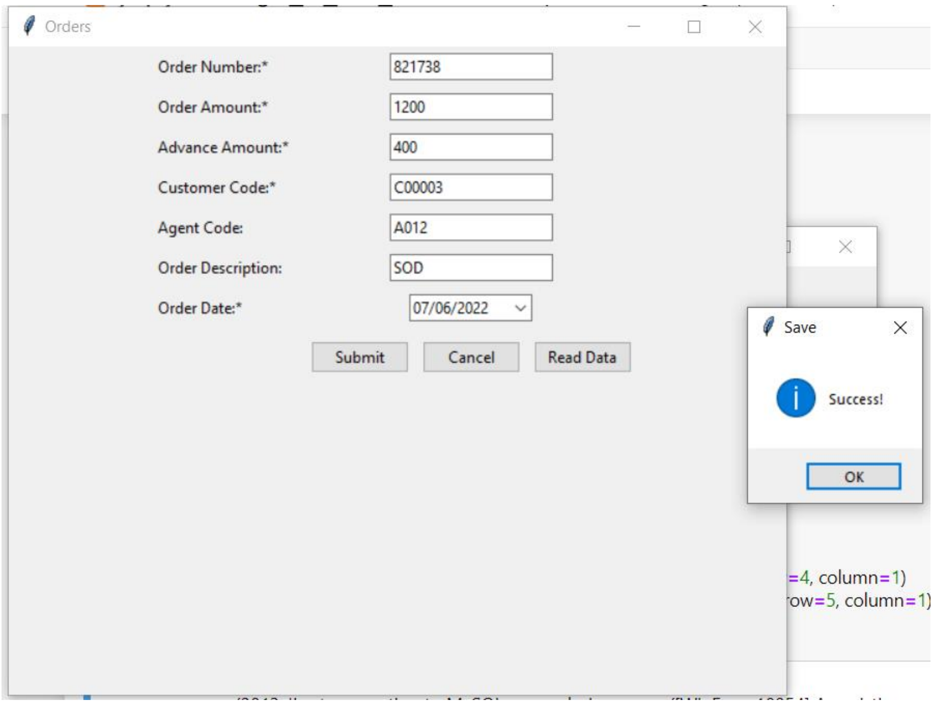
On Selecting Orders in the Menu:



Orders

Order Number:*	
Order Amount:*	
Advance Amount:*	
Customer Code:*	
Agent Code:	
Order Description:	
Order Date:*	16/07/2022

On appropriate filling of all cells and selecting submit:



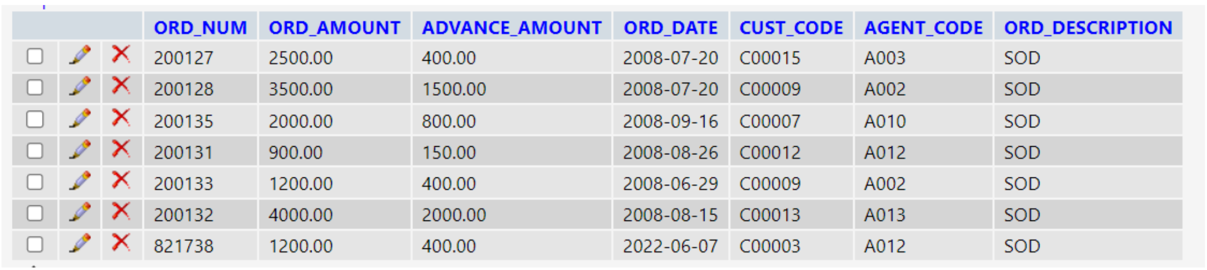
Orders

Order Number:*	821738
Order Amount:*	1200
Advance Amount:*	400
Customer Code:*	C00003
Agent Code:	A012
Order Description:	SOD
Order Date:*	07/06/2022

Save

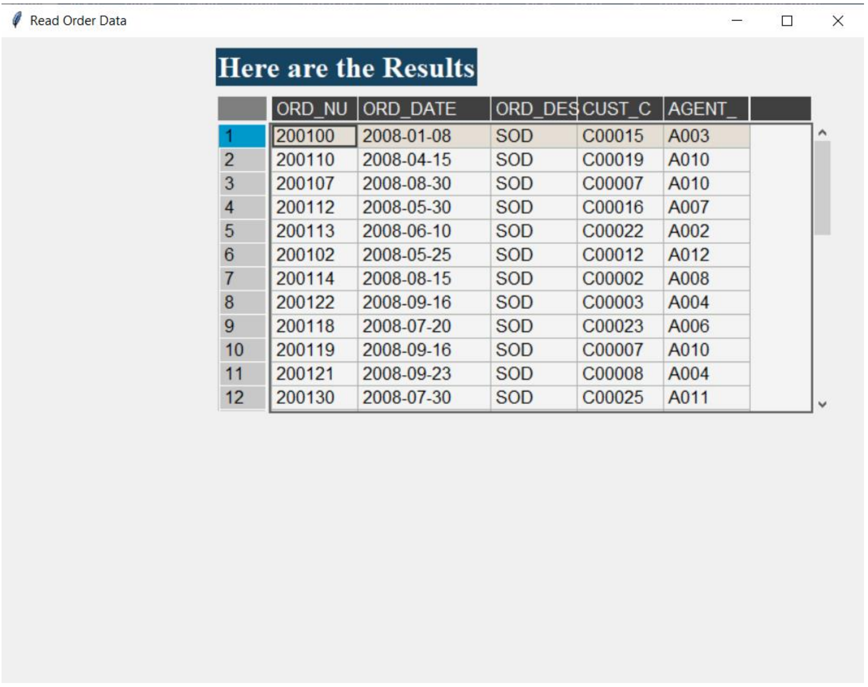
Success!

Database: (new addition in last row)



			ORD_NUM	ORD_AMOUNT	ADVANCE_AMOUNT	ORD_DATE	CUST_CODE	AGENT_CODE	ORD_DESCRIPTION
☐	✎	✗	200127	2500.00	400.00	2008-07-20	C00015	A003	SOD
☐	✎	✗	200128	3500.00	1500.00	2008-07-20	C00009	A002	SOD
☐	✎	✗	200135	2000.00	800.00	2008-09-16	C00007	A010	SOD
☐	✎	✗	200131	900.00	150.00	2008-08-26	C00012	A012	SOD
☐	✎	✗	200133	1200.00	400.00	2008-06-29	C00009	A002	SOD
☐	✎	✗	200132	4000.00	2000.00	2008-08-15	C00013	A013	SOD
☐	✎	✗	821738	1200.00	400.00	2022-06-07	C00003	A012	SOD

On selecting Read Data in Orders Window:



Here are the Results

	ORD_NU	ORD_DATE	ORD_DES	CUST_C	AGENT_
1	200100	2008-01-08	SOD	C00015	A003
2	200110	2008-04-15	SOD	C00019	A010
3	200107	2008-08-30	SOD	C00007	A010
4	200112	2008-05-30	SOD	C00016	A007
5	200113	2008-06-10	SOD	C00022	A002
6	200102	2008-05-25	SOD	C00012	A012
7	200114	2008-08-15	SOD	C00002	A008
8	200122	2008-09-16	SOD	C00003	A004
9	200118	2008-07-20	SOD	C00023	A006
10	200119	2008-09-16	SOD	C00007	A010
11	200121	2008-09-23	SOD	C00008	A004
12	200130	2008-07-30	SOD	C00025	A011

The screenshots for testing of program with incorrect format of data has been presented in Section 5 of the Report.

Code for addorders.py:

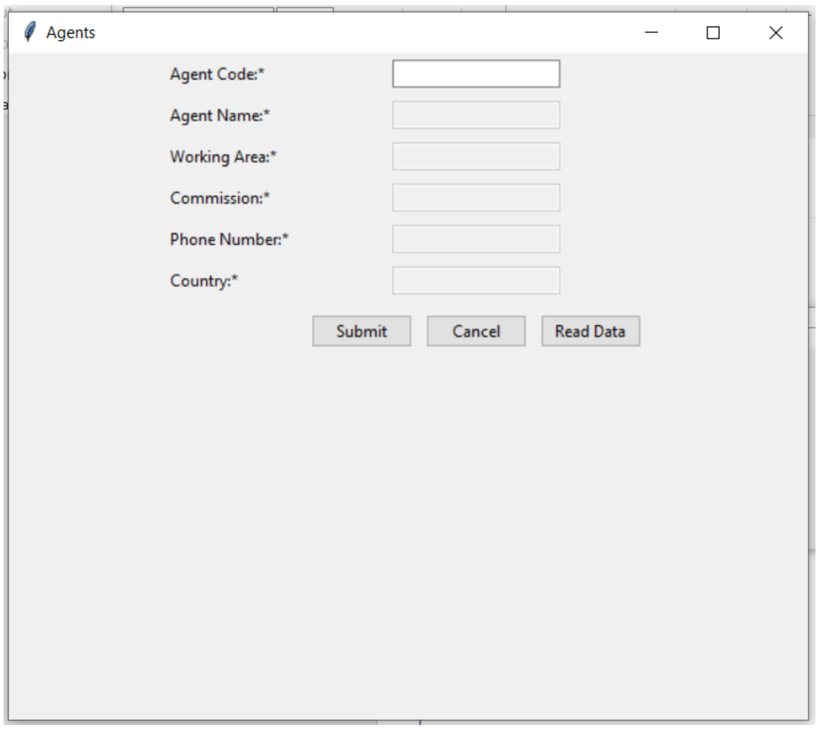
```python
import tkinter
from tkinter import *
from tkinter import messagebox
from functools import partial
import sqlite3
import pymysql
from venv import create
from tkinter.ttk import *
from tkcalendar import Calendar, DateEntry
import re
from datetime import datetime
import pandas as pd
from pandastable import Table, TableModel

# Saving student form data
def addOrders():
    # Checking last input for validating, if not validated, shows error message
    if (len(ord_desc.get()) != 3 or ord_desc.get().isalpha()) == False:
        messagebox.showinfo("Save" , "Not Validated!")
        return
    try:
        # Processing for three date variables: Birth Date, Start Date, End Date
        ordDateObj = datetime.strptime(ord_date.get(), '%d/%m/%Y')

        # Connection for MySQL database
        conn = pymysql.connect(user="root", password="", host="localhost",
database="sunville")
        cur = conn.cursor()
        # Executing insert query
        cur.execute("""insert into orders(ORD_NUM ,ORD_AMOUNT,ADVANCE_AMOUNT,
ORD_DATE,CUST_CODE,AGENT_CODE,ORD_DESCRIPTION) values(%s,%s,%s,%s,%s,%s,%s)""" ,
                    (ord_num.get(),ord_amt.get(),adv_amt.get(),ordDateObj.strftime("%Y-%m-
%d"),cust_code.get(),agent_code.get(),ord_desc.get()))
        conn.close()
        # Show message for success
        messagebox.showinfo("Save" , "Success!")
        b1['state']= 'enabled'
        # Initializing for each input.
        ord_amtEntry.config(state='disabled')
        adv_amtEntry.config(state='disabled')
        cust_codeEntry.config(state='disabled')
        agent_codeEntry.config(state='disabled')
        ord_descEntry.config(state='disabled')
    except Exception as e:
        print(e)
        # If error on saving, shows error message.
        messagebox.showerror("Save" ,"Failed to save!")
# When clicking cancel button, application will be closed.
def cancel():
    ord_num.set("")
    ord_amt.set("")
    adv_amt.set("")
    cust_code.set("")
    agent_code.set("")
    ord_desc.set("")
    root.destroy()

# Validating for each input
def validate(event , input):
    if( input == "Order Number"):
        ord_number = ord_num.get()
        if (len(ord_number) != 6 or ord_number.isdigit()==False):
            messagebox.showerror("Invalid!" ,"Order number has to be a 6 digit number.")
            ord_numEntry.focus_set()
        else:
            ord_amtEntry.focus_set()
            ord_amtEntry.config(state='normal')
    elif(input == "Order Amount"):
        if (ord_amt.get().isdigit()==True and float(ord_amt.get()) > 0 ):
            adv_amtEntry.focus_set()
            adv_amtEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Order amount has to be a number greater than
0.")
            ord_amtEntry.focus_set()
    elif( input == "Advance Amount"):
        if (adv_amt.get().isdigit()==True and float(adv_amt.get()) > 0 and
float(adv_amt.get()) <=float(ord_amt.get()) ):
            cust_codeEntry.focus_set()
            cust_codeEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Advance amount has to be a number greater
than 0 and not greater than Order Amount.")
            adv_amtEntry.focus_set()
    elif( input == "Customer Code"):
        if (len(cust_code.get()) == 6 and cust_code.get().isalnum()==True):
            agent_codeEntry.focus_set()
            agent_codeEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Customer code has to be alphanumberic with
length 6.")
            cust_codeEntry.focus_set()
    elif( input == "Agent Code"):
        if (len(agent_code.get()) == 4 and agent_code.get().isalnum()==True):
            ord_descEntry.focus_set()
            ord_descEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Agent code has to be alphanumberic with
length 4.")
            agent_codeEntry.focus_set()
    elif( input == "Order Description"):
        if len(ord_desc.get()) == 3 and ord_desc.get().isalpha():
            pass
        else:
            messagebox.showerror("Invalid!" ,"Order description has to be only three
letters long.")
            ord_descEntry.focus_set()

def readdata():
    readwindow = Tk()
    readwindow.title("Read Order Data")
    readwindow.geometry('{}x{}'.format(800, 600))

    mainframe1 = Frame(readwindow)
    l = Label(readwindow, text='Here are the Results',font=('times', 20, 'bold'),background
= '#154360',foreground = '#FDFEFE')
    l.place(x = 200, y = 10)

    df = pd.DataFrame()
    df = TableModel.getSampleData()


    conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
    cur = conn.cursor()
        # Executing insert query
    query = "select ORD_NUM,ORD_DATE,ORD_DESCRIPTION,CUST_CODE,AGENT_CODE from orders"
    cur.execute(query)
    df = pd.DataFrame(list(cur.fetchall()),columns
=['ORD_NUM','ORD_DATE','ORD_DESCRIPTION','CUST_CODE','AGENT_CODE'])
    #print (df)

    table =Table(mainframe1, dataframe=df,showtoolbar=True, showstatusbar=True )
    table.currwidth = 700
    table.currheight = 500
```

```python
    mainframe1.place(x = 200,y =200,anchor = "w")
    try:
        table.show()
    except:
        pass
    conn.close()

# Creating main window and setting with width and height
root = Tk()
root.title("Orders")
root.geometry('{}x{}'.format(600, 500))
mainframe = Frame(root)
mainframe.pack()

# Setting string variable for 6 input
ord_num = StringVar()
ord_amt = StringVar()
adv_amt = StringVar()
cust_code = StringVar()
agent_code = StringVar()
ord_desc = StringVar()
# Input for Order Number
ord_numEntry = Entry(mainframe, width=20, textvariable=ord_num)
ord_numEntry.grid(row=0, column=1 ,padx=5, pady=5)
ord_numEntry.bind("<Return>", lambda event: validate(event, "Order Number"))
ord_numEntry.bind("<Tab>", lambda event: validate(event, "Order Number"))
# Input for Order Amount
ord_amtEntry = Entry(mainframe, width=20, textvariable=ord_amt)
ord_amtEntry.grid(row=1, column=1 ,padx=5, pady=5)
ord_amtEntry.bind("<Return>", lambda event: validate(event, "Order Amount"))
ord_amtEntry.bind("<Tab>", lambda event: validate(event, "Order Amount"))
# Input for Advance Amount
adv_amtEntry = Entry(mainframe, width=20, textvariable=adv_amt)
adv_amtEntry.grid(row=2, column=1 ,padx=5, pady=5)
adv_amtEntry.bind("<Return>", lambda event: validate(event, "Advance Amount"))
adv_amtEntry.bind("<Tab>", lambda event: validate(event, "Advance Amount"))
# Input for Customer Code
cust_codeEntry = Entry(mainframe, width=20, textvariable=cust_code)
cust_codeEntry.grid(row=3, column=1 ,padx=5, pady=5)
cust_codeEntry.bind("<Return>", lambda event: validate(event, "Customer Code"))
cust_codeEntry.bind("<Tab>", lambda event: validate(event, "Customer Code"))
# Input for Agent Code
agent_codeEntry = Entry(mainframe, width=20, textvariable=agent_code)
agent_codeEntry.grid(row=4, column=1 ,padx=5, pady=5)
agent_codeEntry.bind("<Return>", lambda event: validate(event, "Agent Code"))
agent_codeEntry.bind("<Tab>", lambda event: validate(event, "Agent Code"))
# Input for Order Description
ord_descEntry = Entry(mainframe, width=20, textvariable=ord_desc)
ord_descEntry.grid(row=5, column=1 ,padx=5, pady=5)
ord_descEntry.bind("<Return>", lambda event: validate(event, "Order Description"))
ord_descEntry.bind("<Tab>", lambda event: validate(event, "Order Description"))

# First rest 5 inputs will be disabled for checking validation

ord_amtEntry.config(state='disabled')
adv_amtEntry.config(state='disabled')
cust_codeEntry.config(state='disabled')
agent_codeEntry.config(state='disabled')
```

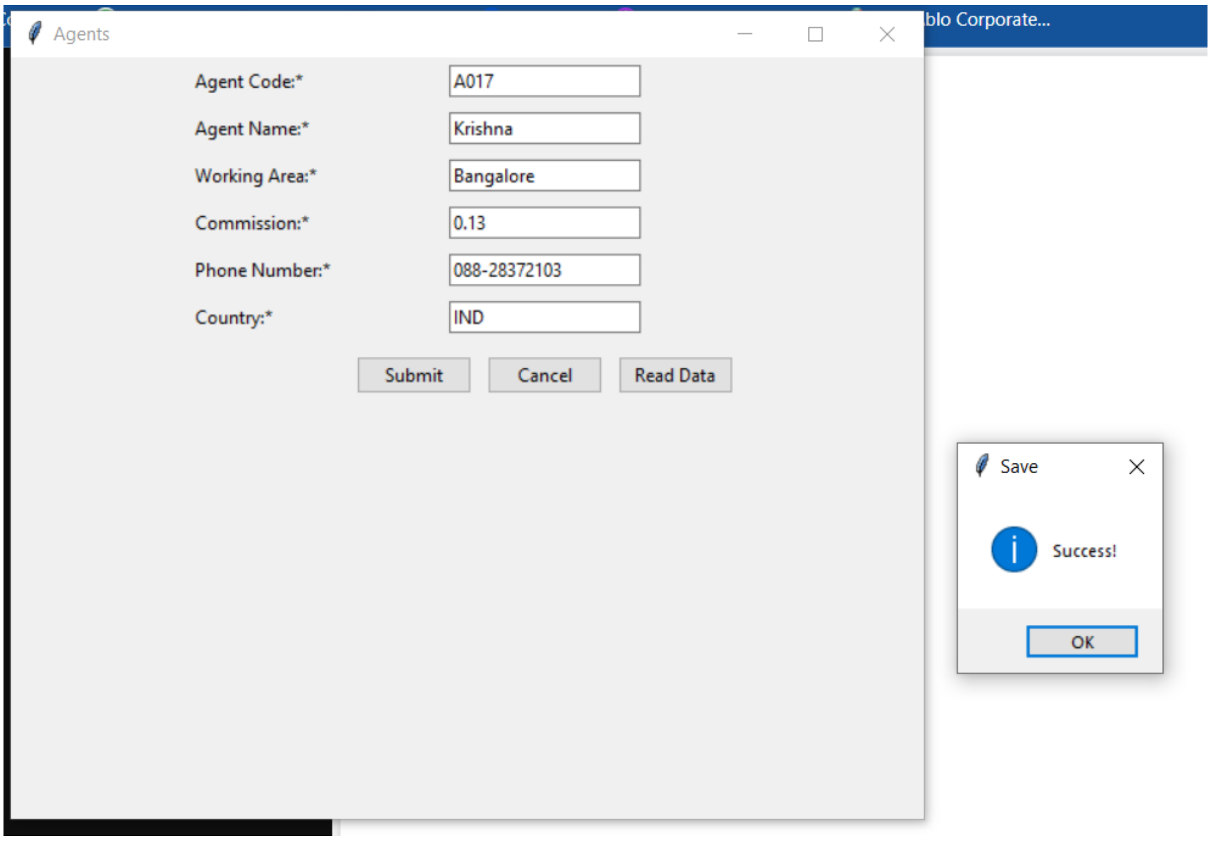
```
ord_descEntry.config(state='disabled')

# Date picker for start date,  end date
ord_date = StringVar()
DateEntry(mainframe , textvariable = ord_date , date_pattern='dd/mm/y' ).grid(row=7,
column=1, padx=5, pady=5)

# Setting labels for each input
Label(mainframe, text='Order Number:*', anchor='w').grid(row=0, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Order Amount:*', anchor='w').grid(row=1, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Advance Amount:*', anchor='w').grid(row=2, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Customer Code:*', anchor='w').grid(row=3, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Agent Code:', anchor='w').grid(row=4, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Order Description:', anchor='w').grid(row=5, column=0 ,padx=5,
pady=5, sticky="w")

Label(mainframe, text='Order Date:*', anchor='w').grid(row=7, column=0 ,padx=5, pady=5,
sticky="w")

# Buttons for submit and cancel
btnFrame = Frame(mainframe)
Button(btnFrame, text="Submit", command=addOrders).grid(row=0, column=1, padx=5, pady=5)
Button(btnFrame, text="Cancel", command=cancel).grid(row=0, column=2, padx=5, pady=5)
b1 = Button(btnFrame, text="Read Data",command= readdata )

b1.grid(row=0, column=3, padx=5, pady=5 )
btnFrame.grid(row=10, column=1, padx=5, pady=5)

root.mainloop()
```

b) Add to agents table

On Selecting Agents in the Menu:



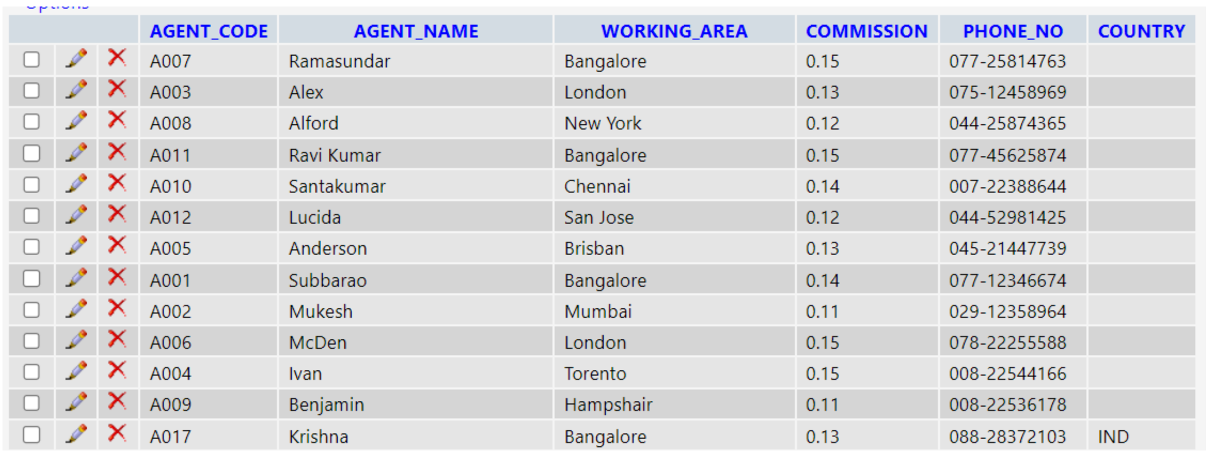
On appropriate filling of each cell and selecting submit:



Agent Code:*	A017
Agent Name:*	Krishna
Working Area:*	Bangalore
Commission:*	0.13
Phone Number:*	088-28372103
Country:*	IND

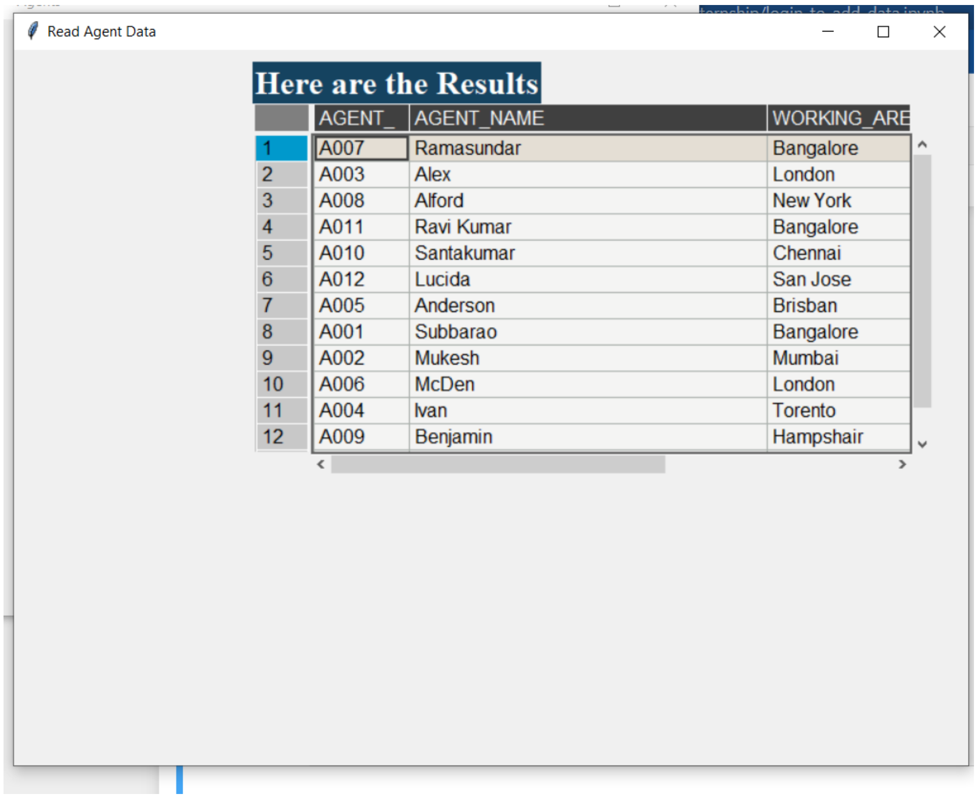
Success!

Database after successful submit: (new addition in last row)



			AGENT_CODE	AGENT_NAME	WORKING_AREA	COMMISSION	PHONE_NO	COUNTRY
☐	✎	✗	A007	Ramasundar	Bangalore	0.15	077-25814763	
☐	✎	✗	A003	Alex	London	0.13	075-12458969	
☐	✎	✗	A008	Alford	New York	0.12	044-25874365	
☐	✎	✗	A011	Ravi Kumar	Bangalore	0.15	077-45625874	
☐	✎	✗	A010	Santakumar	Chennai	0.14	007-22388644	
☐	✎	✗	A012	Lucida	San Jose	0.12	044-52981425	
☐	✎	✗	A005	Anderson	Brisban	0.13	045-21447739	
☐	✎	✗	A001	Subbarao	Bangalore	0.14	077-12346674	
☐	✎	✗	A002	Mukesh	Mumbai	0.11	029-12358964	
☐	✎	✗	A006	McDen	London	0.15	078-22255588	
☐	✎	✗	A004	Ivan	Torento	0.15	008-22544166	
☐	✎	✗	A009	Benjamin	Hampshair	0.11	008-22536178	
☐	✎	✗	A017	Krishna	Bangalore	0.13	088-28372103	IND

On selecting Read Data in Agents Window:



Here are the Results

	AGENT_	AGENT_NAME	WORKING_ARE
1	A007	Ramasundar	Bangalore
2	A003	Alex	London
3	A008	Alford	New York
4	A011	Ravi Kumar	Bangalore
5	A010	Santakumar	Chennai
6	A012	Lucida	San Jose
7	A005	Anderson	Brisban
8	A001	Subbarao	Bangalore
9	A002	Mukesh	Mumbai
10	A006	McDen	London
11	A004	Ivan	Torento
12	A009	Benjamin	Hampshair

The screenshots for testing of program with incorrect format of data has been presented in Section 5 of the Report.

Code for addagents.py

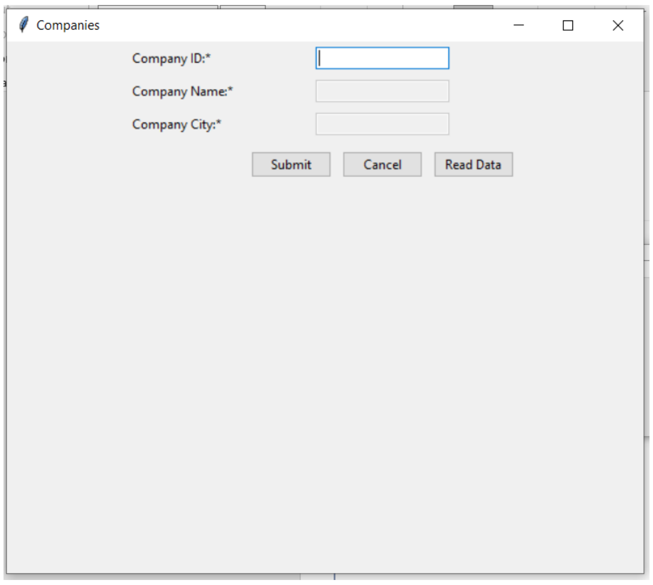
```
import tkinter
from tkinter import *
from tkinter import messagebox
from functools import partial
import sqlite3
import pymysql
from venv import create
from tkinter.ttk import *
from tkcalendar import Calendar, DateEntry
import re
```

```
from datetime import datetime
import pandas as pd
from pandastable import Table, TableModel

# Saving student form data
def addAgents():
    # Checking last input for validating, if not validated, shows error message

    try:
        # Connection for MySQL database
        conn = pymysql.connect(user="root", password="", host="localhost",
database="sunville")
        cur = conn.cursor()
        # Executing insert query
        cur.execute("""insert into agents(AGENT_CODE ,AGENT_NAME,WORKING_AREA,
COMMISSION,PHONE_NO,COUNTRY) values(%s,%s,%s,%s,%s,%s)""" ,
                    (agent_code.get(),agent_name.get(),working_area.get(),commission.get(),
phone_no.get(),country.get()))
        conn.close()
        # Show message for success
        messagebox.showinfo("Save" , "Success!")
        b1['state']= 'enabled'
        # Initializing for each input.
        agent_nameEntry.config(state='disabled')
        working_areaEntry.config(state='disabled')
        commissionEntry.config(state='disabled')
        phone_noEntry.config(state='disabled')
        countryEntry.config(state='disabled')
    except Exception as e:
        print(e)
        # If error on saving, shows error message.
        messagebox.showerror("Save" ,"Failed to save!")
# When clicking cancel button, application will be closed.
def cancel():
    agent_code.set("")
    agent_name.set("")
    working_area.set("")
    commission.set("")
    phone_no.set("")
    country.set("")
    root.destroy()

# Validating for each input
def validate(event , input):
    if( input == "Agent Code"):
        agentcode = agent_code.get()
        if (len(agentcode) == 4 and agentcode.isalnum()==True):
            agent_nameEntry.focus_set()
            agent_nameEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Agent code has to be alphanumeric with length
4.")
            agent_codeEntry.focus_set()
    elif(input == "Agent Name"):
        if (agent_name.get().isalpha()==True and len(agent_name.get())>1):
            working_areaEntry.focus_set()
            working_areaEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Agent Name has to be longer than 1 letter.")
            agent_nameEntry.focus_set()
    elif(input == "Working Area"):
        if (working_area.get().isalpha()==True and len(working_area.get())>1 ):
            commissionEntry.focus_set()
            commissionEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Working Area has to be longer than 1
letter.")
            working_areaEntry.focus_set()
    elif( input == "Commission"):
        if (commission.get().replace('.', '', 1).isdigit()==True):
            phone_noEntry.focus_set()
            phone_noEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Commission has to be in decimal format (e.g.,
0.11 for 11%).")
            commissionEntry.focus_set()
    elif( input == "Phone Number"):
        phn = phone_no.get().replace('-', '', 1)
        if (len(phn) == 11 and phn.isdigit()==True):
            countryEntry.focus_set()
            countryEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Phone Number has to be 11 digits long.")
            phone_noEntry.focus_set()
    elif( input == "Country"):
        if len(country.get()) > -1 and country.get().isalpha()==True:
            pass
        else:
            messagebox.showerror("Invalid!" ,"Country has to be written using alphabets
only.")
            countryEntry.focus_set()

def readagents():
    readwindow = Tk()
    readwindow.title("Read Agent Data")
    readwindow.geometry('{}x{}'.format(800, 600))

    mainframe1 = Frame(readwindow)
    l = Label(readwindow, text='Here are the Results',font=('times', 20, 'bold'),background
= '#154360',foreground = '#FDFEFE')
    l.place(x = 200, y = 10)

    df = pd.DataFrame()
    df = TableModel.getSampleData()

    conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
    cur = conn.cursor()
        # Executing insert query
    query = "select AGENT_CODE,AGENT_NAME,WORKING_AREA,PHONE_NO from agents"
    cur.execute(query)
    df = pd.DataFrame(list(cur.fetchall()),columns
=['AGENT_CODE','AGENT_NAME','WORKING_AREA','PHONE_NO'])
    #print (df)

    table =Table(mainframe1, dataframe=df,showtoolbar=True, showstatusbar=True )
    table.currwidth = 700
    table.currheight = 500
    mainframe1.place(x = 200,y =200,anchor = "w")
    try:
        table.show()
    except:
        pass
    conn.close()

# Creating main window and setting with width and height
root = Tk()
root.title("Agents")
root.geometry('{}x{}'.format(600, 500))
mainframe = Frame(root)
mainframe.pack()

# Setting string variable for input
agent_code = StringVar()
agent_name = StringVar()
working_area = StringVar()
commission = StringVar()
phone_no = StringVar()
country = StringVar()

agent_codeEntry = Entry(mainframe, width=20, textvariable=agent_code)
agent_codeEntry.grid(row=0, column=1 ,padx=5, pady=5)
agent_codeEntry.bind("<Return>", lambda event: validate(event, "Agent Code"))
agent_codeEntry.bind("<Tab>", lambda event: validate(event, "Agent Code"))

agent_nameEntry = Entry(mainframe, width=20, textvariable=agent_name)
agent_nameEntry.grid(row=1, column=1 ,padx=5, pady=5)
agent_nameEntry.bind("<Return>", lambda event: validate(event, "Agent Name"))
agent_nameEntry.bind("<Tab>", lambda event: validate(event, "Agent Name"))

working_areaEntry = Entry(mainframe, width=20, textvariable=working_area)
working_areaEntry.grid(row=2, column=1 ,padx=5, pady=5)
working_areaEntry.bind("<Return>", lambda event: validate(event, "Working Area"))
working_areaEntry.bind("<Tab>", lambda event: validate(event, "Working Area"))

commissionEntry = Entry(mainframe, width=20, textvariable=commission)
commissionEntry.grid(row=3, column=1 ,padx=5, pady=5)
commissionEntry.bind("<Return>", lambda event: validate(event, "Commission"))
commissionEntry.bind("<Tab>", lambda event: validate(event, "Commission"))

phone_noEntry = Entry(mainframe, width=20, textvariable=phone_no)
phone_noEntry.grid(row=4, column=1 ,padx=5, pady=5)
phone_noEntry.bind("<Return>", lambda event: validate(event, "Phone Number"))
phone_noEntry.bind("<Tab>", lambda event: validate(event, "Phone Number"))

countryEntry = Entry(mainframe, width=20, textvariable=country)
countryEntry.grid(row=5, column=1 ,padx=5, pady=5)
countryEntry.bind("<Return>", lambda event: validate(event, "Country"))
countryEntry.bind("<Tab>", lambda event: validate(event, "Country"))



agent_nameEntry.config(state='disabled')
working_areaEntry.config(state='disabled')
commissionEntry.config(state='disabled')
```

```
phone_noEntry.config(state='disabled')
countryEntry.config(state='disabled')

# Setting labels for each input
Label(mainframe, text='Agent Code:*', anchor='w').grid(row=0, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Agent Name:*', anchor='w').grid(row=1, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Working Area:*', anchor='w').grid(row=2, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Commission:*', anchor='w').grid(row=3, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Phone Number:*', anchor='w').grid(row=4, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Country:*', anchor='w').grid(row=5, column=0 ,padx=5, pady=5,
sticky="w")

# Buttons for submit and cancel
btnFrame = Frame(mainframe)
Button(btnFrame, text="Submit", command=addAgents).grid(row=0, column=1, padx=5, pady=5)
Button(btnFrame, text="Cancel", command=cancel).grid(row=0, column=2, padx=5, pady=5)
b1 = Button(btnFrame, text="Read Data",command=readagents)

b1.grid(row=0, column=3, padx=5, pady=5 )
btnFrame.grid(row=10, column=1, padx=5, pady=5)

root.mainloop()
```

c) Add to company table

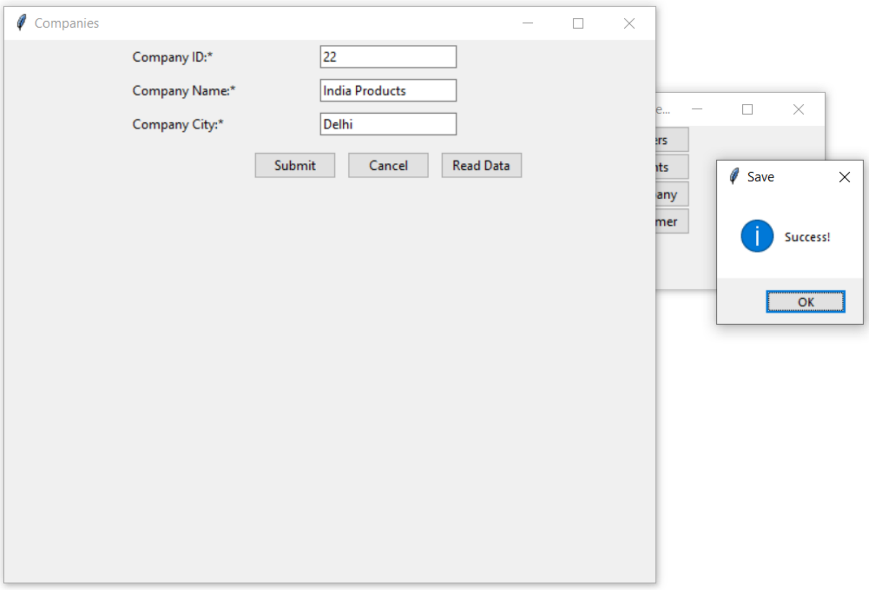
On Selecting Company in the Menu:



Companies

Company ID:*	
Company Name:*	
Company City:*	

On appropriate filling of each cell and selecting submit:

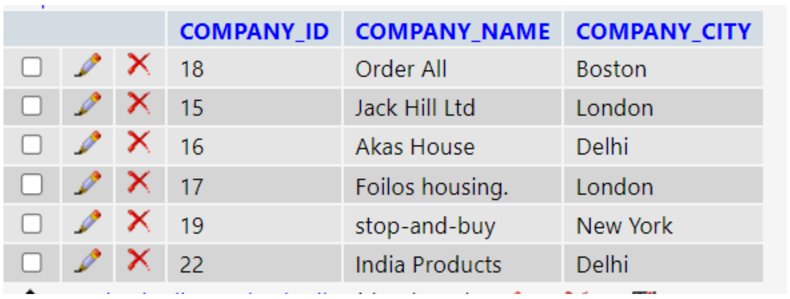


Companies

Company ID:*	22
Company Name:*	India Products
Company City:*	Delhi

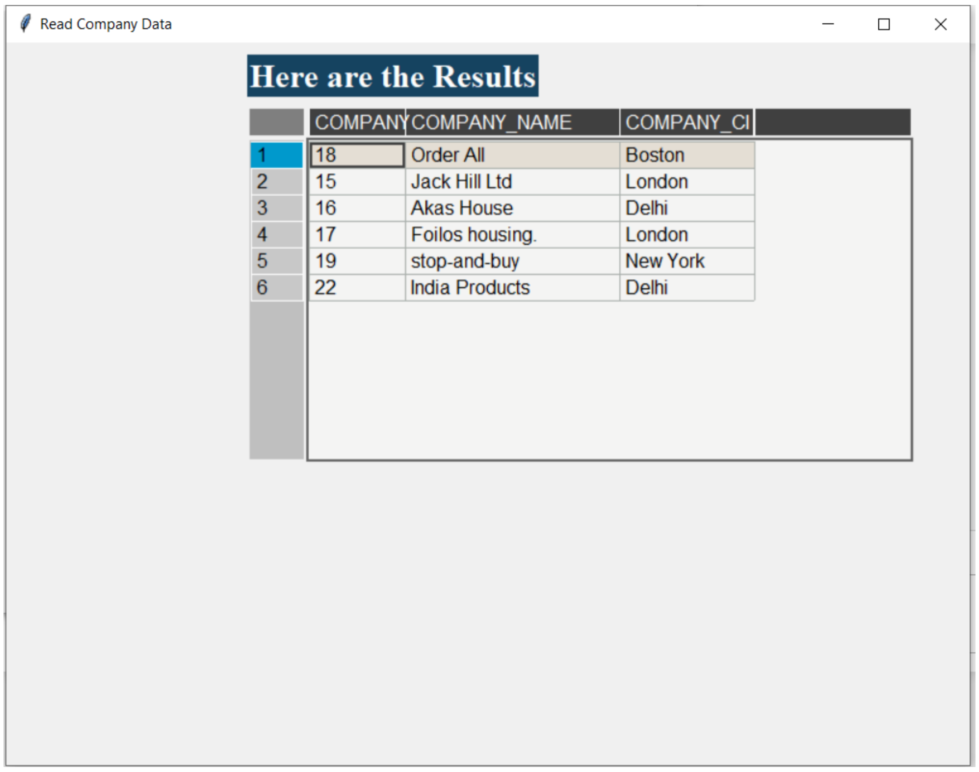
Success!

Database after successful submit: (new addition in last row)



COMPANY_ID	COMPANY_NAME	COMPANY_CITY
18	Order All	Boston
15	Jack Hill Ltd	London
16	Akas House	Delhi
17	Foilos housing.	London
19	stop-and-buy	New York
22	India Products	Delhi

On selecting Read Data in Company Window:


Read Company Data

Here are the Results

	COMPANY	COMPANY_NAME	COMPANY_CI
1	18	Order All	Boston
2	15	Jack Hill Ltd	London
3	16	Akas House	Delhi
4	17	Foilos housing.	London
5	19	stop-and-buy	New York
6	22	India Products	Delhi

The screenshots for testing of program with incorrect format of data has been presented in Section 5 of the Report.

Code for addcompany.py:

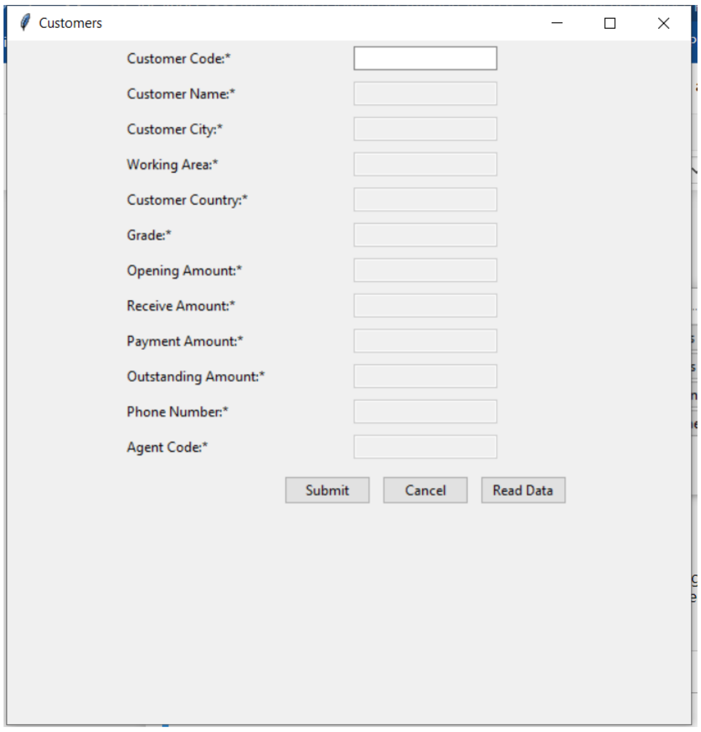
```
import tkinter
from tkinter import *
from tkinter import messagebox
from functools import partial
import sqlite3
import pymysql
from venv import create
from tkinter.ttk import *
from tkcalendar import Calendar, DateEntry
import re
from datetime import datetime
import pandas as pd
from pandastable import Table, TableModel

# Saving student form data
def addCompanies():
    # Checking last input for validating, if not validated , shows error message

    if len(comp_city.get()) < 0 or comp_city.get().isalpha()==False:
        messagebox.showinfo("Save" , "Not Validated!")
        return
    try:
        # Connection for mysql database
        conn = pymysql.connect(user="root", password="", host="localhost",
database="sunville")
        cur = conn.cursor()
        # Executing insert query
        cur.execute("""insert into company(COMPANY_ID, COMPANY_NAME,COMPANY_CITY)
values(%s,%s,%s)""" ,
                    (comp_id.get(),comp_name.get(),comp_city.get()))
        conn.close()
        # Show message for success
        messagebox.showinfo("Save" , "Success!")
        b1['state']= 'enabled'
        # Initializing for each input.
        comp_nameEntry.config(state='disabled')
        comp_cityEntry.config(state='disabled')

    except Exception as e:
        print(e)
        # If error on saving, shows error message.
        messagebox.showerror("Save" ,"Failed to save!")
# When clicking cancel button, application will be closed.
def cancel():
    comp_id.set("")
    comp_name.set("")
    comp_city.set("")
    root.destroy()

# Validating for each input
def validate(event , input):

    if( input == "Company ID"):
        compid = comp_id.get()
        if (len(compid) != 2 or compid.isdigit()==False):
            messagebox.showerror("Invalid!" ,"Company ID has to be numeric with length 2.")
            comp_idEntry.focus_set()
        else:
            comp_nameEntry.focus_set()
            comp_nameEntry.config(state='normal')
    elif(input == "Company Name"):
        if (comp_name.get().isdigit()==False and len(comp_name.get())>1):
            comp_cityEntry.focus_set()
            comp_cityEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Company Name has to be longer that 1
character.")
            agent_nameEntry.focus_set()
    elif(input == "Company City"):
        if (comp_city.get().isalpha()==True and len(comp_city.get())>1 ):
            pass
        else:
            messagebox.showerror("Invalid!" ,"Company City has to be longer than 1
letter.")
            comp_cityEntry.focus_set()

def readcompanies():
    readwindow = Tk()
    readwindow.title("Read Company Data")
    readwindow.geometry('{}x{}'.format(800, 600))

    mainframe1 = Frame(readwindow)
    l = Label(readwindow, text='Here are the Results',font=('times', 20, 'bold'),background
= '#154360',foreground = '#FDFEFE')
```

```
    l.place(x = 200, y = 10)

    df = pd.DataFrame()
    df = TableModel.getSampleData()

    conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
    cur = conn.cursor()
        # Executing insert query
    query = "select COMPANY_ID,COMPANY_NAME,COMPANY_CITY from company"
    cur.execute(query)
    df = pd.DataFrame(list(cur.fetchall()),columns
=['COMPANY_ID','COMPANY_NAME','COMPANY_CITY'])
    #print (df)

    table =Table(mainframe1, dataframe=df,showtoolbar=True, showstatusbar=True )
    table.currwidth = 700
    table.currheight = 500
    mainframe1.place(x = 200,y =200,anchor = "w")
    try:
        table.show()
    except:
        pass
    conn.close()

# Creating main window and setting with width and height
root = Tk()
root.title("Companies")
root.geometry('{}x{}'.format(600, 500))
mainframe = Frame(root)
mainframe.pack()

# Setting string variable for input
comp_id = StringVar()
comp_name = StringVar()
comp_city = StringVar()

comp_idEntry = Entry(mainframe, width=20, textvariable=comp_id)
comp_idEntry.grid(row=0, column=1 ,padx=5, pady=5)
comp_idEntry.bind("<Return>", lambda event: validate(event, "Company ID"))
comp_idEntry.bind("<Tab>", lambda event: validate(event, "Company ID"))

comp_nameEntry = Entry(mainframe, width=20, textvariable=comp_name)
comp_nameEntry.grid(row=1, column=1 ,padx=5, pady=5)
comp_nameEntry.bind("<Return>", lambda event: validate(event, "Company Name"))
comp_nameEntry.bind("<Tab>", lambda event: validate(event, "Company Name"))

comp_cityEntry = Entry(mainframe, width=20, textvariable=comp_city)
comp_cityEntry.grid(row=2, column=1 ,padx=5, pady=5)
comp_cityEntry.bind("<Return>", lambda event: validate(event, "Company City"))
comp_cityEntry.bind("<Tab>", lambda event: validate(event, "Company City"))

comp_nameEntry.config(state='disabled')
comp_cityEntry.config(state='disabled')

# Setting labels for each input
Label(mainframe, text='Company ID:*', anchor='w').grid(row=0, column=0 ,padx=5, pady=5,
sticky="w")
```

```
Label(mainframe, text='Company Name:*', anchor='w').grid(row=1, column=0 ,padx=5, pady=5, 
sticky="w")
Label(mainframe, text='Company City:*', anchor='w').grid(row=2, column=0 ,padx=5, pady=5, 
sticky="w")

# Buttons for submit and cancel
btnFrame = Frame(mainframe)
Button(btnFrame, text="Submit", command=addCompanies).grid(row=0, column=1, padx=5, pady=5)
Button(btnFrame, text="Cancel", command=cancel).grid(row=0, column=2, padx=5, pady=5)
b1 = Button(btnFrame, text="Read Data",command=readcompanies)

b1.grid(row=0, column=3, padx=5, pady=5 )
btnFrame.grid(row=10, column=1, padx=5, pady=5)

root.mainloop()
```

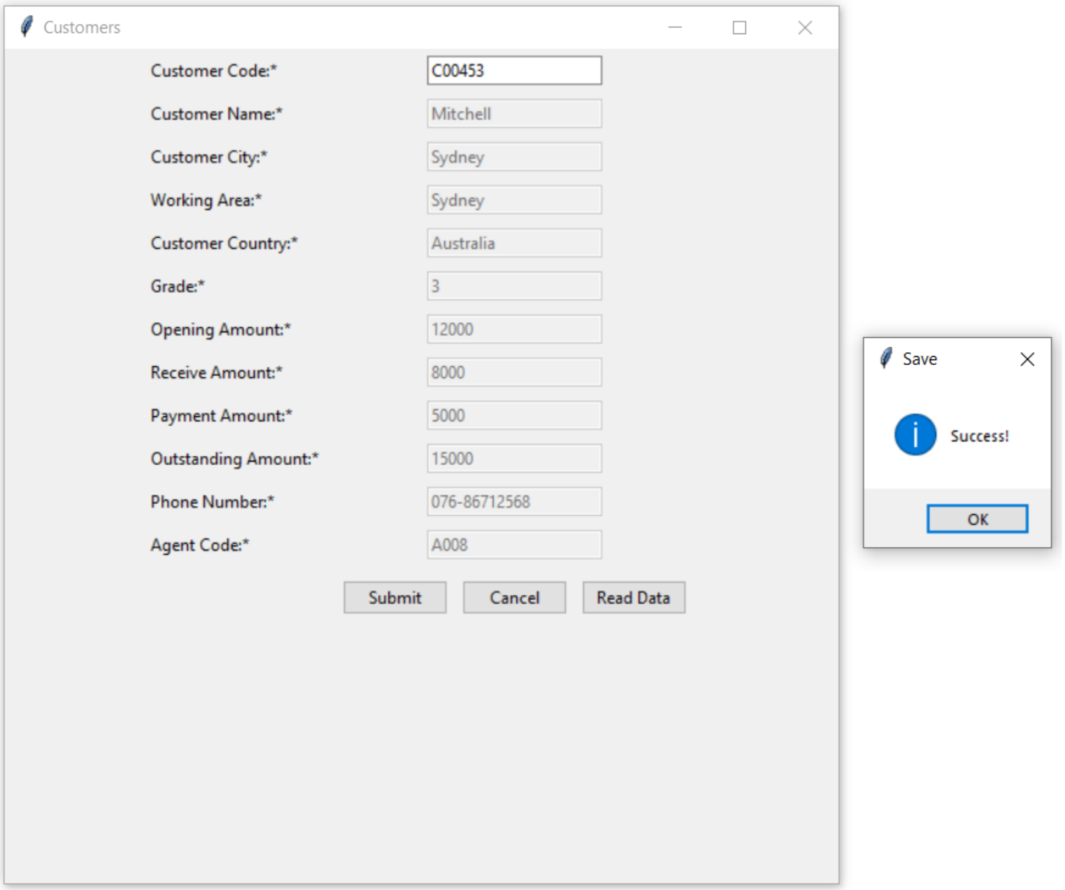
d) Add to customer table

On Selecting Customer in the Menu:



Customer Code:*	
Customer Name:*	
Customer City:*	
Working Area:*	
Customer Country:*	
Grade:*	
Opening Amount:*	
Receive Amount:*	
Payment Amount:*	
Outstanding Amount:*	
Phone Number:*	
Agent Code:*	

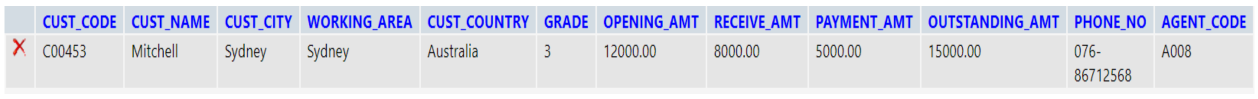
On appropriate filling of each cell and selecting submit:



Customer Code:*	C00453
Customer Name:*	Mitchell
Customer City:*	Sydney
Working Area:*	Sydney
Customer Country:*	Australia
Grade:*	3
Opening Amount:*	12000
Receive Amount:*	8000
Payment Amount:*	5000
Outstanding Amount:*	15000
Phone Number:*	076-86712568
Agent Code:*	A008

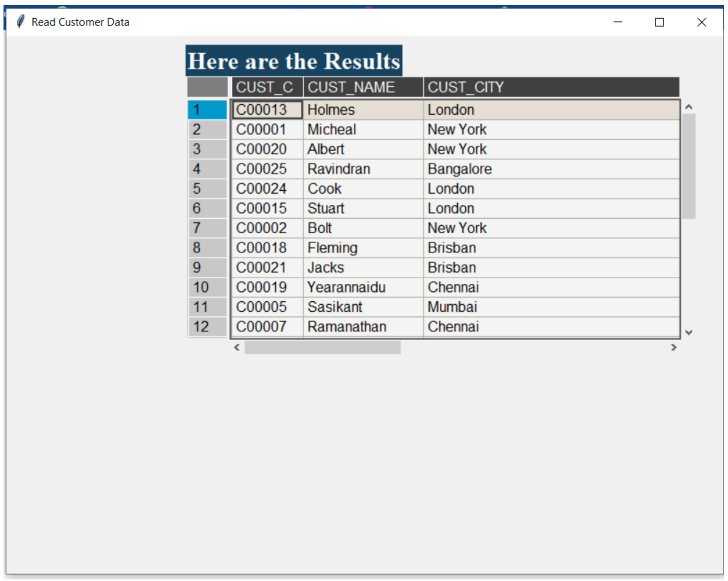
Success!

Database after successful submit: (new addition in last row)



	CUST_CODE	CUST_NAME	CUST_CITY	WORKING_AREA	CUST_COUNTRY	GRADE	OPENING_AMT	RECEIVE_AMT	PAYMENT_AMT	OUTSTANDING_AMT	PHONE_NO	AGENT_CODE
✗	C00453	Mitchell	Sydney	Sydney	Australia	3	12000.00	8000.00	5000.00	15000.00	076-86712568	A008

On selecting Read Data in Customer Window:



Here are the Results

	CUST_C	CUST_NAME	CUST_CITY
1	C00013	Holmes	London
2	C00001	Micheal	New York
3	C00020	Albert	New York
4	C00025	Ravindran	Bangalore
5	C00024	Cook	London
6	C00015	Stuart	London
7	C00002	Bolt	New York
8	C00018	Fleming	Brisban
9	C00021	Jacks	Brisban
10	C00019	Yearannaidu	Chennai
11	C00005	Sasikant	Mumbai
12	C00007	Ramanathan	Chennai

The screenshots for testing of program with incorrect format of data has been presented in Section 5 of the Report.

Code for addcustomer.py:

```python
import tkinter
from tkinter import *
from tkinter import messagebox
from functools import partial
import sqlite3
import pymysql
from venv import create
from tkinter.ttk import *
from tkcalendar import Calendar, DateEntry
import re
from datetime import datetime
import pandas as pd
from pandastable import Table, TableModel

# Saving student form data
def addCustomers():
    # Checking last input for validating, if not validated, shows error message

    if len(agent_code.get()) != 4 or agent_code.get().isalnum()==False:
        messagebox.showinfo("Save" , "Not Validated!")
        return
    try:
        # Connection for mysql database
        conn = pymysql.connect(user="root", password="", host="localhost",
database="sunville")
        cur = conn.cursor()
        # Executing insert query
        cur.execute("""insert into customer(CUST_CODE ,CUST_NAME,CUST_CITY,
WORKING_AREA,CUST_COUNTRY,GRADE,OPENING_AMT,RECEIVE_AMT,PAYMENT_AMT,OUTSTANDING_AMT,PHONE_N
O,AGENT_CODE)
                    values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)""" ,
                    (cust_code.get(),cust_name.get(),cust_city.get(),working_area.get(),cus
t_country.get(),grade.get(),opening_amt.get(),receive_amt.get(),payment_amt.get(),out_amt.g
et(),phone_no.get(),agent_code.get()))
        conn.close()
        # Show message for success
        messagebox.showinfo("Save" , "Success!")
        b1['state']= 'enabled'
        # Initializing for each input.
        cust_nameEntry.config(state='disabled')
        cust_cityEntry.config(state='disabled')
        working_areaEntry.config(state='disabled')
        cust_countryEntry.config(state='disabled')
        gradeEntry.config(state='disabled')
        opening_amtEntry.config(state='disabled')
        receive_amtEntry.config(state='disabled')
        payment_amtEntry.config(state='disabled')
        out_amtEntry.config(state='disabled')
        phone_noEntry.config(state='disabled')
        agent_codeEntry.config(state='disabled')

    except Exception as e:
        print(e)
        # If error on saving, shows error message.
        messagebox.showerror("Save" ,"Failed to save!")
# When clicking cancel button, application will be closed.
def cancel():
    cust_code.set("")
    cust_name.set("")
    cust_city.set("")
    working_area.set("")
    cust_country.set("")
    grade.set("")
    opening_amt.set("")
    receive_amt.set("")
    payment_amt.set("")
    out_amt.set("")
    phone_no.set("")
    agent_code.set("")
    root.destroy()

# Validating for each input
def validate(event , input):
    if( input == "Customer Code"):
        custcode = cust_code.get()
        if (len(custcode) == 6 and custcode.isalnum()==True):
            cust_nameEntry.focus_set()
            cust_nameEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Customer code has to be alphanumeric with
length 6.")
```

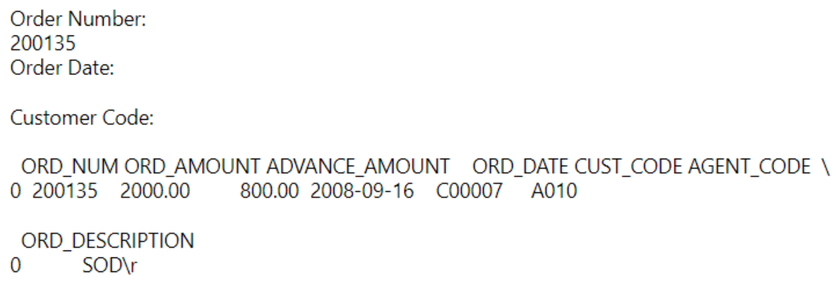
```
            cust_codeEntry.focus_set()
    elif(input == "Customer Name"):
        if (cust_name.get().isalpha()==True and len(cust_name.get())>0):
            cust_cityEntry.focus_set()
            cust_cityEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Customer Name has to be atleast 1 letter.")
            cust_nameEntry.focus_set()
    elif(input == "Customer City"):
        if (cust_city.get().isalpha()==True and len(cust_city.get())>1 ):
            working_areaEntry.focus_set()
            working_areaEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Customer city has to be longer than 1
letter.")
            cust_cityEntry.focus_set()
    elif( input == "Working Area"):
        if (len(working_area.get()) >0 and working_area.get().isalpha()==True):
            cust_countryEntry.focus_set()
            cust_countryEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Working area has to be longer than 1
letter.")
            working_areaEntry.focus_set()
    elif( input == "Customer Country"):
        if (len(cust_country.get()) >0 and cust_country.get().isalpha()==True):
            gradeEntry.focus_set()
            gradeEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Customer Country has to be longer than 1
letter.")
            cust_countryEntry.focus_set()
    elif( input == "Grade"):
        if grade.get().isdigit()==True:
            if (int(grade.get()) >=0 and int(grade.get())<=10):
                opening_amtEntry.focus_set()
                opening_amtEntry.config(state='normal')
            else:
                messagebox.showerror("Invalid!" ,"Grade should lie between 0-10.")
                gradeEntry.focus_set()
        else:
            messagebox.showerror("Invalid!" ,"Grade should be numeric.")
            gradeEntry.focus_set()
    elif( input == "Opening Amount"):
        if ((len(opening_amt.get()) >=2 and len(opening_amt.get())<=12) and
opening_amt.get().isdigit()==True):
            receive_amtEntry.focus_set()
            receive_amtEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Opening amount has to be numeric with length
2-12.")
            opening_amtEntry.focus_set()
    elif(input == "Receive Amount"):
        if ((len(receive_amt.get()) >=2 and len(receive_amt.get())<=12) and
receive_amt.get().isdigit()==True):
            payment_amtEntry.focus_set()
            payment_amtEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Receive amount has to be numeric with length
2-12.")
            receive_amtEntry.focus_set()
    elif(input == "Payment Amount"):
        if ((len(payment_amt.get()) >=2 and len(payment_amt.get())<=12) and
payment_amt.get().isdigit()==True):
            out_amtEntry.focus_set()
            out_amtEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Payment amount has to be numeric with length
2-12.")
            payment_amtEntry.focus_set()
    elif( input == "Outstanding Amount"):
        if ((len(out_amt.get()) >=2 and len(out_amt.get())<=12) and
out_amt.get().isdigit()==True):
            phone_noEntry.focus_set()
            phone_noEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Outstanding amount has to be numeric with
length 2-12.")
            out_amtEntry.focus_set()
    elif( input == "Phone No"):
        phn = phone_no.get().replace('-', '', 1)
        if (len(phn) == 11 and phn.isdigit()==True):

            agent_codeEntry.focus_set()
            agent_codeEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Phone Number must be 11 digits.")
            phone_noEntry.focus_set()
    elif( input == "Agent Code"):
        if (len(agent_code.get())==4 and agent_code.get().isalnum()==True):
            pass
        else:
            messagebox.showerror("Invalid!" ,"Agent Code must be alphanumeric with length
4.")
            agent_codeEntry.focus_set()

def readcustomers():
    readwindow = Tk()
    readwindow.title("Read Customer Data")
    readwindow.geometry('{}x{}'.format(800, 600))

    mainframe1 = Frame(readwindow)
    l = Label(readwindow, text='Here are the Results',font=('times', 20, 'bold'),background
= '#154360',foreground = '#FDFEFE')
    l.place(x = 200, y = 10)

    df = pd.DataFrame()
    df = TableModel.getSampleData()

    conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
    cur = conn.cursor()
        # Executing insert query
    query = "select CUST_CODE ,CUST_NAME,CUST_CITY,
WORKING_AREA,CUST_COUNTRY,GRADE,OPENING_AMT,RECEIVE_AMT,PAYMENT_AMT,OUTSTANDING_AMT,PHONE_N
O,AGENT_CODE from customer"
    cur.execute(query)
    df = pd.DataFrame(list(cur.fetchall()),columns =['CUST_CODE' ,'CUST_NAME','CUST_CITY',
'WORKING_AREA','CUST_COUNTRY','GRADE','OPENING_AMT','RECEIVE_AMT','PAYMENT_AMT','OUTSTANDIN
G_AMT','PHONE_NO','AGENT_CODE'])
    #print (df)

    table =Table(mainframe1, dataframe=df,showtoolbar=True, showstatusbar=True )
    table.currwidth = 700
    table.currheight = 500
    mainframe1.place(x = 200,y =200,anchor = "w")
    try:
        table.show()
    except:
        pass
    conn.close()

# Creating main window and setting with width and height
root = Tk()
root.title("Customers")
root.geometry('{}x{}'.format(600, 600))
mainframe = Frame(root)
mainframe.pack()

# Setting string variable for input
cust_name = StringVar()
cust_city = StringVar()
working_area = StringVar()
cust_country = StringVar()
grade = StringVar()
opening_amt = StringVar()
receive_amt = StringVar()
payment_amt = StringVar()
out_amt = StringVar()
phone_no = StringVar()
agent_code = StringVar()
cust_code = StringVar()

cust_codeEntry = Entry(mainframe, width=20, textvariable=cust_code)
cust_codeEntry.grid(row=0, column=1 ,padx=5, pady=5)
cust_codeEntry.bind("<Return>", lambda event: validate(event, "Customer Code"))
cust_codeEntry.bind("<Tab>", lambda event: validate(event, "Customer Code"))

cust_nameEntry = Entry(mainframe, width=20, textvariable=cust_name)
cust_nameEntry.grid(row=1, column=1 ,padx=5, pady=5)
cust_nameEntry.bind("<Return>", lambda event: validate(event, "Customer Name"))
cust_nameEntry.bind("<Tab>", lambda event: validate(event, "Customer Name"))

cust_cityEntry = Entry(mainframe, width=20, textvariable=cust_city)
cust_cityEntry.grid(row=2, column=1 ,padx=5, pady=5)
cust_cityEntry.bind("<Return>", lambda event: validate(event, "Customer City"))
cust_cityEntry.bind("<Tab>", lambda event: validate(event, "Customer City"))

working_areaEntry = Entry(mainframe, width=20, textvariable=working_area)
working_areaEntry.grid(row=3, column=1 ,padx=5, pady=5)
working_areaEntry.bind("<Return>", lambda event: validate(event, "Working Area"))
working_areaEntry.bind("<Tab>", lambda event: validate(event, "Working Area"))

cust_countryEntry = Entry(mainframe, width=20, textvariable=cust_country)
cust_countryEntry.grid(row=4, column=1 ,padx=5, pady=5)
cust_countryEntry.bind("<Return>", lambda event: validate(event, "Customer Country"))
cust_countryEntry.bind("<Tab>", lambda event: validate(event, "Customer Country"))

gradeEntry = Entry(mainframe, width=20, textvariable=grade)
gradeEntry.grid(row=5, column=1 ,padx=5, pady=5)
gradeEntry.bind("<Return>", lambda event: validate(event, "Grade"))
gradeEntry.bind("<Tab>", lambda event: validate(event, "Grade"))

opening_amtEntry = Entry(mainframe, width=20, textvariable=opening_amt)
opening_amtEntry.grid(row=6, column=1 ,padx=5, pady=5)
opening_amtEntry.bind("<Return>", lambda event: validate(event, "Opening Amount"))
opening_amtEntry.bind("<Tab>", lambda event: validate(event, "Opening Amount"))

receive_amtEntry = Entry(mainframe, width=20, textvariable=receive_amt)
receive_amtEntry.grid(row=7, column=1 ,padx=5, pady=5)
receive_amtEntry.bind("<Return>", lambda event: validate(event, "Receive Amount"))
receive_amtEntry.bind("<Tab>", lambda event: validate(event, "Receive Amount"))

payment_amtEntry = Entry(mainframe, width=20, textvariable=payment_amt)
payment_amtEntry.grid(row=8, column=1 ,padx=5, pady=5)
payment_amtEntry.bind("<Return>", lambda event: validate(event, "Payment Amount"))
payment_amtEntry.bind("<Tab>", lambda event: validate(event, "Payment Amount"))

out_amtEntry = Entry(mainframe, width=20, textvariable=out_amt)
out_amtEntry.grid(row=9, column=1 ,padx=5, pady=5)
out_amtEntry.bind("<Return>", lambda event: validate(event, "Outstanding Amount"))
out_amtEntry.bind("<Tab>", lambda event: validate(event, "Outstanding Amount"))
phone_noEntry = Entry(mainframe, width=20, textvariable=phone_no)
phone_noEntry.grid(row=10, column=1 ,padx=5, pady=5)
phone_noEntry.bind("<Return>", lambda event: validate(event, "Phone No"))
phone_noEntry.bind("<Tab>", lambda event: validate(event, "Phone No"))

agent_codeEntry = Entry(mainframe, width=20, textvariable=agent_code)
agent_codeEntry.grid(row=11, column=1 ,padx=5, pady=5)
agent_codeEntry.bind("<Return>", lambda event: validate(event, "Agent Code"))
agent_codeEntry.bind("<Tab>", lambda event: validate(event, "Agent Code"))

cust_nameEntry.config(state='disabled')
cust_cityEntry.config(state='disabled')
working_areaEntry.config(state='disabled')
cust_countryEntry.config(state='disabled')
gradeEntry.config(state='disabled')
opening_amtEntry.config(state='disabled')
receive_amtEntry.config(state='disabled')
payment_amtEntry.config(state='disabled')
out_amtEntry.config(state='disabled')
phone_noEntry.config(state='disabled')
agent_codeEntry.config(state='disabled')

# Setting labels for each input
Label(mainframe, text='Customer Code:*', anchor='w').grid(row=0, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Customer Name:*', anchor='w').grid(row=1, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Customer City:*', anchor='w').grid(row=2, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Working Area:*', anchor='w').grid(row=3, column=0 ,padx=5, pady=5,
sticky="w")
```

```
Label(mainframe, text='Customer Country:*', anchor='w').grid(row=4, column=0 ,padx=5,
pady=5, sticky="w")
Label(mainframe, text='Grade:*', anchor='w').grid(row=5, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Opening Amount:*', anchor='w').grid(row=6, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Receive Amount:*', anchor='w').grid(row=7, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Payment Amount:*', anchor='w').grid(row=8, column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Outstanding Amount:*', anchor='w').grid(row=9, column=0 ,padx=5,
pady=5, sticky="w")
Label(mainframe, text='Phone Number:*', anchor='w').grid(row=10,column=0 ,padx=5, pady=5,
sticky="w")
Label(mainframe, text='Agent Code:*', anchor='w').grid(row=11,column=0 ,padx=5, pady=5,
sticky="w")

# Buttons for submit and cancel
btnFrame = Frame(mainframe)
Button(btnFrame, text="Submit", command=addCustomers).grid(row=15, column=1, padx=5,
pady=5)
Button(btnFrame, text="Cancel", command=cancel).grid(row=15, column=2, padx=5, pady=5)
b1 = Button(btnFrame, text="Read Data",command=readcustomers)

b1.grid(row=15, column=3, padx=5, pady=5 )
btnFrame.grid(row=15, column=1, padx=5, pady=5)

root.mainloop()
```

A2)

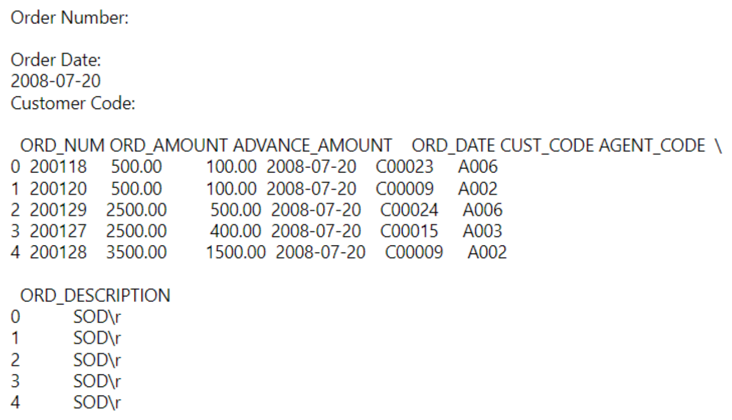
Order Lookup:

a) Based on only Order Number:



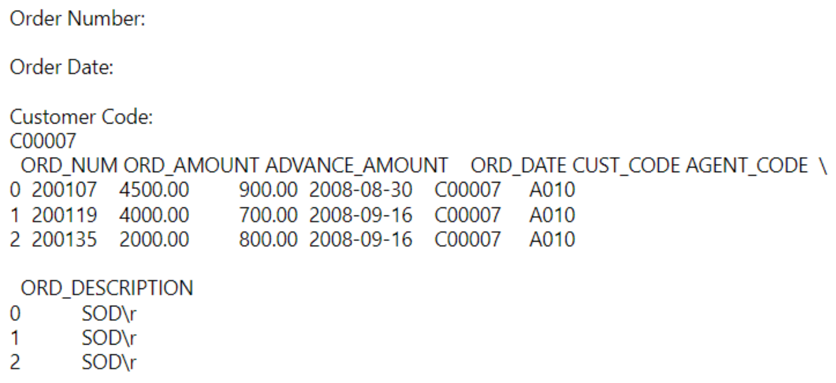
```
Order Number:
200135
Order Date:

Customer Code:

  ORD_NUM ORD_AMOUNT ADVANCE_AMOUNT    ORD_DATE CUST_CODE AGENT_CODE  \
0  200135    2000.00         800.00  2008-09-16    C00007     A010

  ORD_DESCRIPTION
0          SOD\r
```

b) Based on only Order Date:



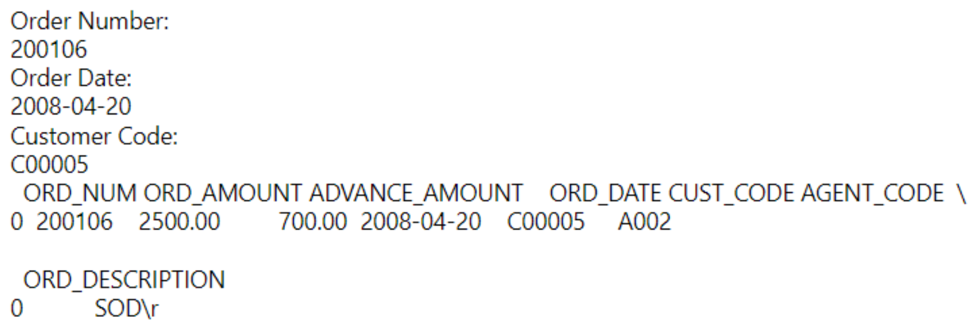
```
Order Number:

Order Date:
2008-07-20
Customer Code:

  ORD_NUM ORD_AMOUNT ADVANCE_AMOUNT    ORD_DATE CUST_CODE AGENT_CODE  \
0  200118     500.00         100.00  2008-07-20    C00023     A006
1  200120     500.00         100.00  2008-07-20    C00009     A002
2  200129    2500.00         500.00  2008-07-20    C00024     A006
3  200127    2500.00         400.00  2008-07-20    C00015     A003
4  200128    3500.00        1500.00  2008-07-20    C00009     A002

  ORD_DESCRIPTION
0          SOD\r
1          SOD\r
2          SOD\r
3          SOD\r
4          SOD\r
```

c) Based on only Customer Code:



```
Order Number:

Order Date:

Customer Code:
C00007
  ORD_NUM ORD_AMOUNT ADVANCE_AMOUNT    ORD_DATE CUST_CODE AGENT_CODE  \
0  200107    4500.00         900.00  2008-08-30    C00007     A010
1  200119    4000.00         700.00  2008-09-16    C00007     A010
2  200135    2000.00         800.00  2008-09-16    C00007     A010

  ORD_DESCRIPTION
0          SOD\r
1          SOD\r
2          SOD\r
```

d) Based on all three combined:



```
Order Number:
200106
Order Date:
2008-04-20
Customer Code:
C00005
  ORD_NUM ORD_AMOUNT ADVANCE_AMOUNT    ORD_DATE CUST_CODE AGENT_CODE  \
0  200106    2500.00         700.00  2008-04-20    C00005     A002

  ORD_DESCRIPTION
0          SOD\r
```

Similarly, any two of the three can also be used to lookup orders.

Code for Order Lookup:

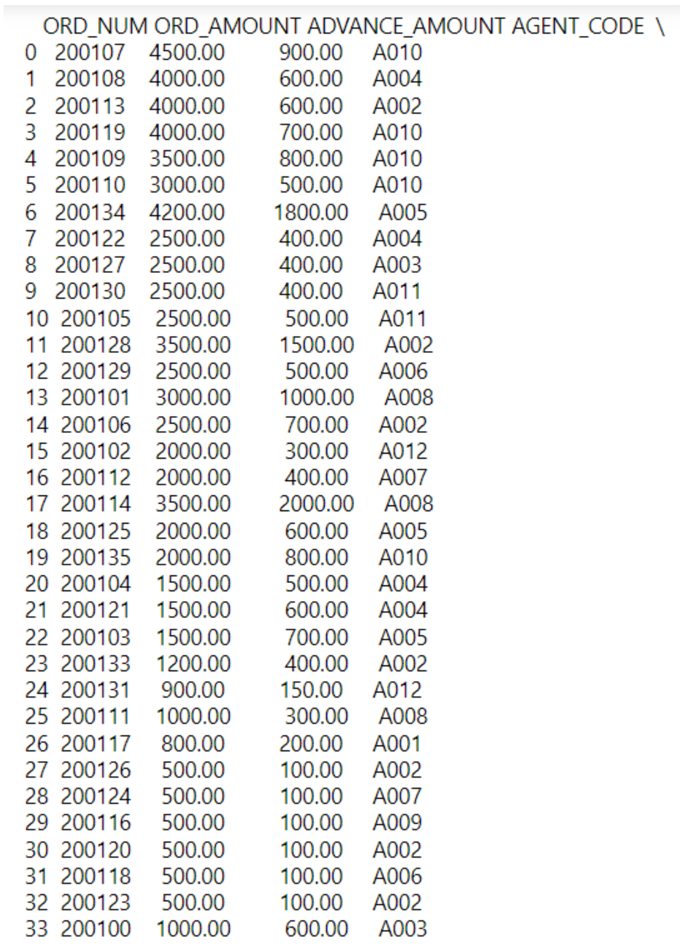
```
import pymysql
import tkinter
from tkinter import *
from tkinter import messagebox
import pandas as pd
from pandastable import Table, TableModel
```

```
# Creating main window and setting with width and height

print("Order Number:")
ordnum=input()

print("Order Date:")
orddate=input()

print("Customer Code:")
custcode=input()

df = TableModel.getSampleData()


conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
cur = conn.cursor()

if ordnum=="":
    if orddate=="":
        if custcode=="":
            query = ("select * from orders")
            cur.execute(query)
        else:
            query = ("select * from orders WHERE CUST_CODE=%s")
            cur.execute(query,(custcode))
    else:
        if custcode=="":
            query = ("select * from orders WHERE ORD_DATE=%s")
            cur.execute(query,(orddate))
        else:
            query = ("select * from orders WHERE ORD_DATE=%s and CUST_CODE=%s")
            cur.execute(query,(orddate,custcode))
else:
    if orddate=="":
        if custcode=="":
            query = ("select * from orders WHERE ORD_NUM = %s")
            cur.execute(query,(ordnum))
        else:
            query = ("select * from orders WHERE ORD_NUM = %s and CUST_CODE=%s")
            cur.execute(query,(ordnum,custcode))
    else:
        if custcode=="":
            query = ("select * from orders WHERE ORD_NUM = %s and ORD_DATE=%s")
            cur.execute(query,(ordnum, orddate))
        else:
            query = ("select * from orders WHERE ORD_NUM = %s and ORD_DATE=%s and
CUST_CODE=%s")
```

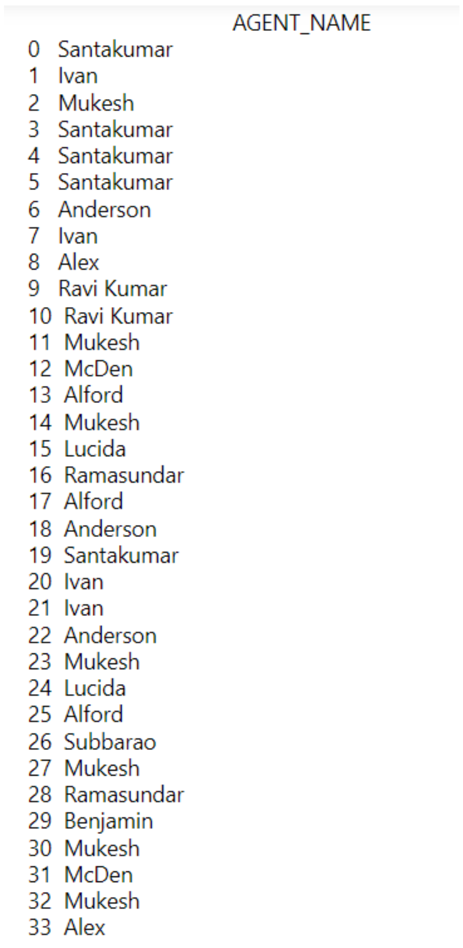
```
            cur.execute(query,(ordnum, orddate,custcode))

df = pd.DataFrame(list(cur.fetchall()),columns
=['ORD_NUM','ORD_AMOUNT','ADVANCE_AMOUNT','ORD_DATE','CUST_CODE','AGENT_CODE','ORD_DESCRIPT
ION'])
print (df)

conn.close()
```

A3) Balance Amounts

Balance amounts in descending order:



```
   ORD_NUM ORD_AMOUNT ADVANCE_AMOUNT AGENT_CODE  \
0   200107    4500.00         900.00       A010
1   200108    4000.00         600.00       A004
2   200113    4000.00         600.00       A002
3   200119    4000.00         700.00       A010
4   200109    3500.00         800.00       A010
5   200110    3000.00         500.00       A010
6   200134    4200.00        1800.00       A005
7   200122    2500.00         400.00       A004
8   200127    2500.00         400.00       A003
9   200130    2500.00         400.00       A011
10  200105    2500.00         500.00       A011
11  200128    3500.00        1500.00       A002
12  200129    2500.00         500.00       A006
13  200101    3000.00        1000.00       A008
14  200106    2500.00         700.00       A002
15  200102    2000.00         300.00       A012
16  200112    2000.00         400.00       A007
17  200114    3500.00        2000.00       A008
18  200125    2000.00         600.00       A005
19  200135    2000.00         800.00       A010
20  200104    1500.00         500.00       A004
21  200121    1500.00         600.00       A004
22  200103    1500.00         700.00       A005
23  200133    1200.00         400.00       A002
24  200131     900.00         150.00       A012
25  200111    1000.00         300.00       A008
26  200117     800.00         200.00       A001
27  200126     500.00         100.00       A002
28  200124     500.00         100.00       A007
29  200116     500.00         100.00       A009
30  200120     500.00         100.00       A002
31  200118     500.00         100.00       A006
32  200123     500.00         100.00       A002
33  200100    1000.00         600.00       A003
```

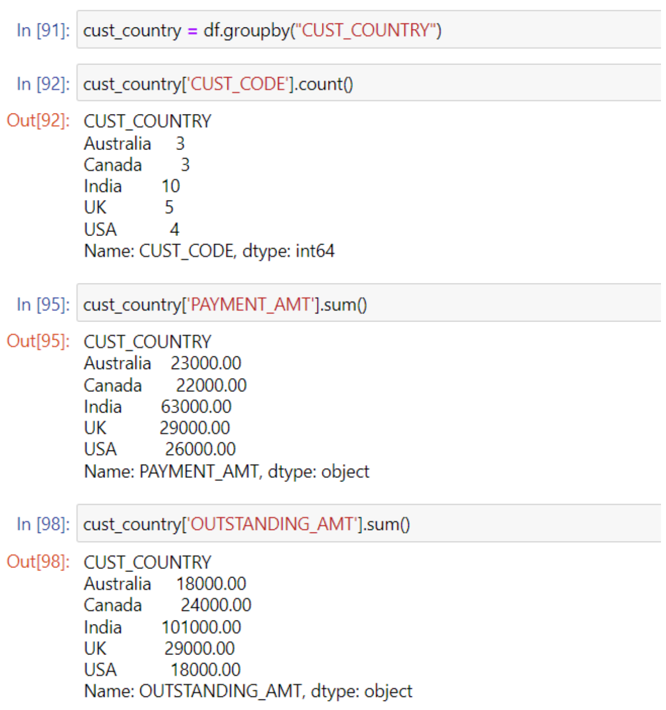


```
                                AGENT_NAME
0   Santakumar
1   Ivan
2   Mukesh
3   Santakumar
4   Santakumar
5   Santakumar
6   Anderson
7   Ivan
8   Alex
9   Ravi Kumar
10  Ravi Kumar
11  Mukesh
12  McDen
13  Alford
14  Mukesh
15  Lucida
16  Ramasundar
17  Alford
18  Anderson
19  Santakumar
20  Ivan
21  Ivan
22  Anderson
23  Mukesh
24  Lucida
25  Alford
26  Subbarao
27  Mukesh
28  Ramasundar
29  Benjamin
30  Mukesh
31  McDen
32  Mukesh
33  Alex
```

Code:

```
# Creating main window and setting with width and height

df = TableModel.getSampleData()

conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
cur = conn.cursor()

# query = ("""CREATE TABLE balance
#        AS (SELECT orders.ORD_NUM, orders.ORD_AMOUNT, orders.ADVANCE_AMOUNT,
orders.AGENT_CODE, agents.AGENT_NAME
#       FROM orders, agents
#       WHERE orders.AGENT_CODE = agents.AGENT_CODE)""")
# cur.execute(query)

query="select * from balance ORDER BY (ORD_AMOUNT - ADVANCE_AMOUNT) DESC"
cur.execute(query)
df = pd.DataFrame(list(cur.fetchall()),columns
=['ORD_NUM','ORD_AMOUNT','ADVANCE_AMOUNT','AGENT_CODE','AGENT_NAME'])
print (df)

conn.close()
```

A4) Customer Countries



```
In [91]: cust_country = df.groupby("CUST_COUNTRY")

In [92]: cust_country['CUST_CODE'].count()

Out[92]: CUST_COUNTRY
         Australia    3
         Canada       3
         India       10
         UK           5
         USA          4
         Name: CUST_CODE, dtype: int64

In [95]: cust_country['PAYMENT_AMT'].sum()

Out[95]: CUST_COUNTRY
         Australia   23000.00
         Canada      22000.00
         India       63000.00
         UK          29000.00
         USA         26000.00
         Name: PAYMENT_AMT, dtype: object

In [98]: cust_country['OUTSTANDING_AMT'].sum()

Out[98]: CUST_COUNTRY
         Australia    18000.00
         Canada       24000.00
         India       101000.00
         UK           29000.00
         USA          18000.00
         Name: OUTSTANDING_AMT, dtype: object
```

Code:

```
# Creating main window and setting with width and height

df = TableModel.getSampleData()

conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
cur = conn.cursor()

query="select CUST_CODE, CUST_COUNTRY,PAYMENT_AMT,OUTSTANDING_AMT from customer"
cur.execute(query)
df = pd.DataFrame(list(cur.fetchall()),columns =['CUST_CODE',
'CUST_COUNTRY','PAYMENT_AMT','OUTSTANDING_AMT'])
print (df)

conn.close()
```

```
cust_country = df.groupby("CUST_COUNTRY")
cust_country['CUST_CODE'].count()
cust_country['PAYMENT_AMT'].sum()
cust_country['OUTSTANDING_AMT'].sum()
```

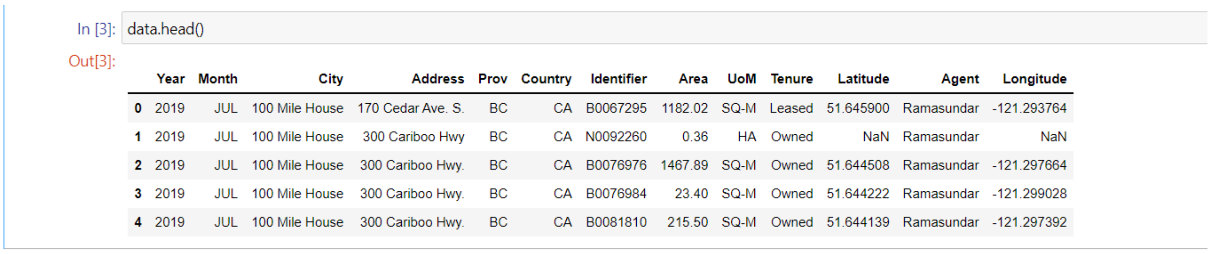
Part-B: Code with Screenshots

Importing Libraries and dataset:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
data = pd.read_excel("property.xlsx", skiprows=8,skipfooter =1)
```

```
data.head()
```

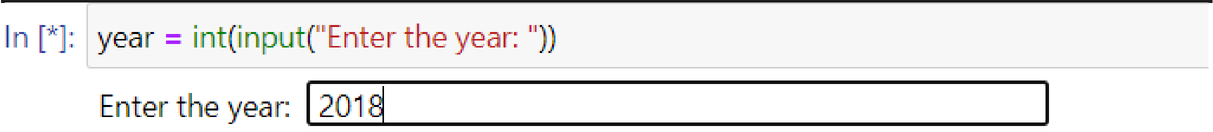


```
In [3]: data.head()
Out[3]:
```

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
0	2019	JUL	100 Mile House	170 Cedar Ave. S.	BC	CA	B0067295	1182.02	SQ-M	Leased	51.645900	Ramasundar	-121.293764
1	2019	JUL	100 Mile House	300 Cariboo Hwy	BC	CA	N0092260	0.36	HA	Owned	NaN	Ramasundar	NaN
2	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076976	1467.89	SQ-M	Owned	51.644508	Ramasundar	-121.297664
3	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076984	23.40	SQ-M	Owned	51.644222	Ramasundar	-121.299028
4	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0081810	215.50	SQ-M	Owned	51.644139	Ramasundar	-121.297392

B1)

```
year = int(input("Enter the year: "))
```



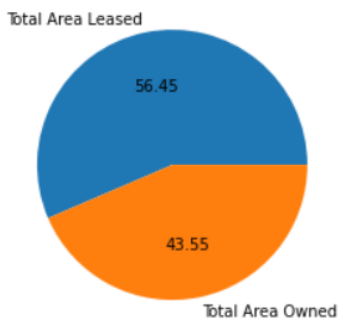
```
In [*]: year = int(input("Enter the year: "))

        Enter the year: 2018
```

```
d1 = data[(data["UoM"]=="SQ-M") & (data['Year']==year)]
d1.head()
d1_leased = d1[d1["Tenure"] == "Leased"]
d1_owned  = d1[d1["Tenure"] == "Owned"]
totLeased = d1_leased["Area"].sum()
totOwned = d1_owned["Area"].sum()
print("In SQ-M\nTotal area Leased =",totLeased,"\nTotal area owned =",totOwned)
```

```
In SQ-M
Total area Leased = 73766.96
Total area owned = 56914.37000000001
```

```
plt.pie([totLeased,totOwned],labels= ["Total Area Leased","Total Area
Owned"],autopct='%.2f')
```

```
Out[8]: ([<matplotlib.patches.Wedge at 0x22b0000b6d0>,
          <matplotlib.patches.Wedge at 0x22b0000bdc0>],
         [Text(-0.2213050872731415, 1.077508263702431, 'Total Area Leased'),
          Text(0.22130508727314088, -1.0775082637024311, 'Total Area Owned')],
         [Text(-0.1207118657853499, 0.5877317802013259, '56.45'),
          Text(0.12071186578534956, -0.587731780201326, '43.55')])
```



B2)

```
data_CA = data[(data["Country"] == "CA") & (data["Tenure"] == "Leased")]
data_WS = data[(data["Country"] == "WS") & (data["Tenure"] == "Leased")]
data_WS.head()
data_CA_2017 = data_CA[data_CA["Year"] == 2017]
data_CA_2018 = data_CA[data_CA["Year"] == 2018]
data_CA_2019 = data_CA[data_CA["Year"] == 2019]
CA_2017 = data_CA_2017["Area"].sum()
CA_2018 = data_CA_2018["Area"].sum()
CA_2019 = data_CA_2019["Area"].sum()
print("Leased area in CA per year:")
print(f"2017: {CA_2017},\n2018: {CA_2018},\n2019: {CA_2019}")
```

```
Leased area in CA per year:
2017: 70660.0792,
2018: 66458.39110000001,
2019: 213945.70029999997
```

The maximum leased area in CA was in 2019.

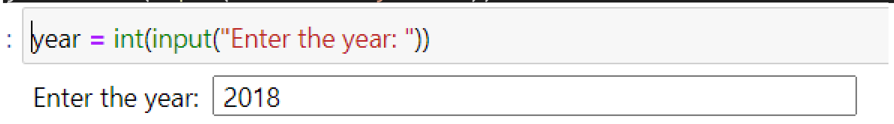
```
data_WS_2017 = data_WS[data_WS["Year"] == 2017]
data_WS_2018 = data_WS[data_WS["Year"] == 2018]
data_WS_2019 = data_WS[data_WS["Year"] == 2019]
WS_2017 = data_WS_2017["Area"].sum()
WS_2018 = data_WS_2018["Area"].sum()
WS_2019 = data_WS_2019["Area"].sum()
print("Leased area in WS per year:")
print(f"2017: {WS_2017},\n2018: {WS_2018},\n2019: {WS_2019}")
```

```
Leased area in WS per year:
2017: 69052.36,
2018: 7328.419999999999,
2019: 62758.43609999999
```

The maximum leased area in WS was in 2017.

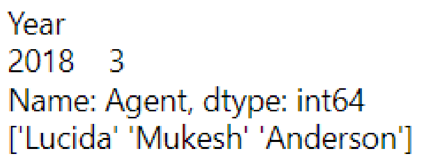
B3)

```
year = int(input("Enter the year: "))
```



```
: year = int(input("Enter the year: "))

  Enter the year: 2018
```

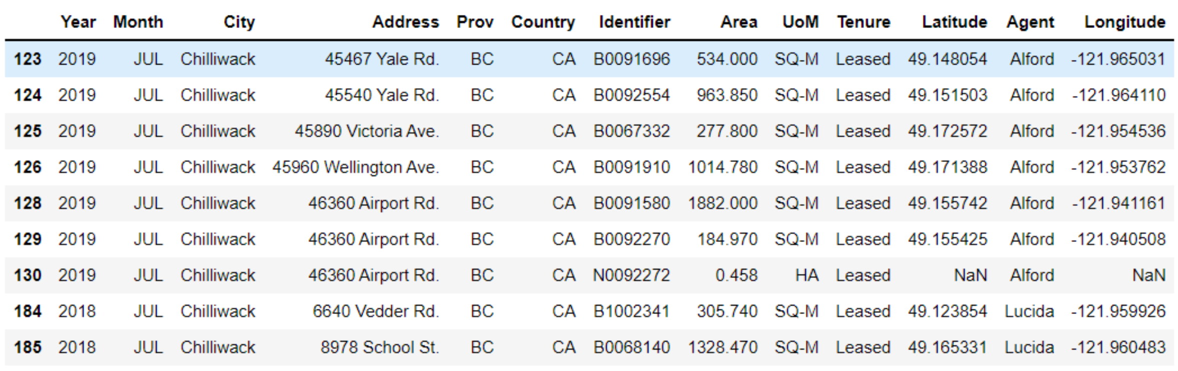
```
data_owned = data[(data["Tenure"]== "Owned")&(data["Year"]==year)]
data_owned.head()
data_owned = data_owned.groupby("Year")
data_owned.head()
print(data_owned["Agent"].nunique())
d = data_owned["Agent"].unique()
for i in d:
    print(i)
```



```
Year
2018    3
Name: Agent, dtype: int64
['Lucida' 'Mukesh' 'Anderson']
```

B4)

```
data_chilliwack = data[data["City"] == "Chilliwack"]
data_chilliwack_leased = data_chilliwack[data_chilliwack["Tenure"] == "Leased"]
data_chilliwack_leased.groupby("Agent")
data_chilliwack_leased
```



	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
123	2019	JUL	Chilliwack	45467 Yale Rd.	BC	CA	B0091696	534.000	SQ-M	Leased	49.148054	Alford	-121.965031
124	2019	JUL	Chilliwack	45540 Yale Rd.	BC	CA	B0092554	963.850	SQ-M	Leased	49.151503	Alford	-121.964110
125	2019	JUL	Chilliwack	45890 Victoria Ave.	BC	CA	B0067332	277.800	SQ-M	Leased	49.172572	Alford	-121.954536
126	2019	JUL	Chilliwack	45960 Wellington Ave.	BC	CA	B0091910	1014.780	SQ-M	Leased	49.171388	Alford	-121.953762
128	2019	JUL	Chilliwack	46360 Airport Rd.	BC	CA	B0091580	1882.000	SQ-M	Leased	49.155742	Alford	-121.941161
129	2019	JUL	Chilliwack	46360 Airport Rd.	BC	CA	B0092270	184.970	SQ-M	Leased	49.155425	Alford	-121.940508
130	2019	JUL	Chilliwack	46360 Airport Rd.	BC	CA	N0092272	0.458	HA	Leased	NaN	Alford	NaN
184	2018	JUL	Chilliwack	6640 Vedder Rd.	BC	CA	B1002341	305.740	SQ-M	Leased	49.123854	Lucida	-121.959926
185	2018	JUL	Chilliwack	8978 School St.	BC	CA	B0068140	1328.470	SQ-M	Leased	49.165331	Lucida	-121.960483

B5)

```
data_year = data.groupby(["Year","Agent"])
data_year.head()
data_year["Area"].sum()
```

```
Year  Agent
2017  Alford          29593.8090
      Lucida          13914.3562
      Subbarao       197999.8562
2018  Anderson        81852.1996
      Lucida           3916.4332
      Mukesh          44975.5461
2019  Alford          60342.3226
      Anderson        33344.7545
      Benjamin        94510.6454
      Lucida         153362.9382
      McDen           77545.5536
      Mukesh          93555.1494
      Ramasundar      22763.6769
      Ramasundar          0.3600
      Ravi Kumar      67688.3404
      Subbarao        88292.7933
2020  Lucida         200186.4284
      Subbarao        37067.9420
Name: Area, dtype: float64
```

B6)

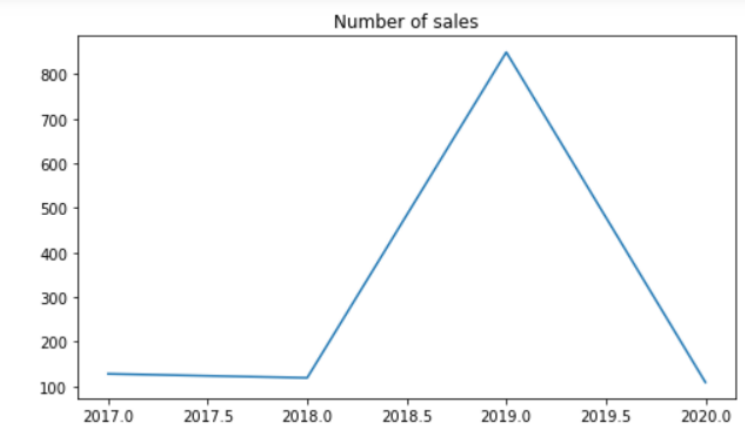
```
data_jul = data[data["Month"]=='JUL']
data_jul.head()
data_jul=data_jul.groupby("Year")
data_jul.head()
data_jul=data_jul.groupby("Year")
data_jul.head()
```

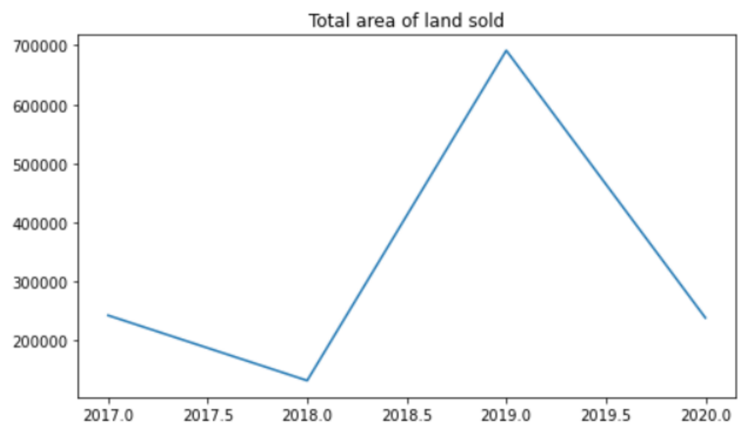
```
Year
2017    241508.0214
2018    130744.1789
2019    691406.5343
2020    237254.3704
Name: Area, dtype: float64
```

B7)

```
data_year = data.groupby("Year")
data_year.head()
list(data_year["Area"].sum())
```

```
X = [2017,2018,2019,2020]
figure, axis = plt.subplots(nrows = 2, ncols=1,figsize =(8,10))
axis[0].plot(X, list(data_year["Identifier"].nunique()))
axis[0].set_title("Number of sales")
axis[1].plot(X, list(data_year["Area"].sum()))
axis[1].set_title("Total area of land sold")
```


Number of sales
800
700
600
500
400
300
200
100
2017.0
2017.5
2018.0
2018.5
2019.0
2019.5
2020.0


Total area of land sold
700000
600000
500000
400000
300000
200000
2017.0
2017.5
2018.0
2018.5
2019.0
2019.5
2020.0

SECTION 5

Testing

1) The login authentication was tested by trying to login using unregistered username and trying to login with registered username and incorrect password. Result- The program did not start with an appropriate registered pair of username and password.
2) Validations:
   a. Add to 'order' table

Attribute	**Validation**
Order Number	6-digit number
Order Amount	A positive number
Advance Amount	A positive number not greater than Order Amount
Customer Code	Alphanumeric with 6 characters long
Agent Code	Alphanumeric with 4 characters long
Order Description	3-character long string

   b. Add to 'agents' table

Attribute	**Validation**
Agent Code	Alphanumeric with 4 characters long
Agent Name	A string with at least 1 character, no digits allowed
Working Area	A string with at least 1 character, no digits allowed
Commission	Float
Phone Number	11-digit number may or may not be separated by a hyphen '- '
Country	A string, can be left empty (NULL)

   c. Add to 'company' table

Attribute	**Validation**
Company ID	2-digit number
Company Name	A string with at least 2 characters, no digits allowed
Company City	A string with at least 2 characters, no digits allowed

d. Add to ‘customer’ table

Attribute	**Validation**
Customer Code	Alphanumeric with 6 characters long
Customer Name	A string with at least 1 character, no digits allowed
Customer City	A string with at least 2 characters, no digits allowed
Working Area	A string with at least 1 character, no digits allowed
Customer Country	A string with at least 1 character, no digits allowed
Grade	Integer between 0 and 10, both inclusive
Opening Amount	Numeric with length 2-12
Receive Amount	Numeric with length 2-12
Payment Amount	Numeric with length 2-12
Outstanding Amount	Numeric with length 2-12
Phone Number	11-digit number may or may not be separated by a hyphen ‘- ‘
Agent Code	Alphanumeric with 4 characters long

3) Test for adding an entry with duplicate primary key was also performed. Result- The program does not let the entry to be saved.

4) Multiple devices were used for running the program to ensure the compatibility and performance of the apps across devices.

5) All the codes were run multiple times with different inputs during the development of the program and the final test run.

TO NOTE:

While using the login app to add data to the tables, the window for login should be closed before clicking on any of the four options of the tables you want to select.

Login data of registered usernames and password is stored in cred.db

SECTION 6

Final Code

The final code is displayed in the following order:

1. Part-A1-Login_to_add_data.ipynb
    a. addorders.py
    b. addagents.py
    c. addcompany.py
    d. addcustomer.py
2. Part-A2,A3.A4.ipynb
3. Part-B-Insights.ipynb

Part-A1-Login_to_add_data.ipynb

In [7]:

```
import tkinter
from tkinter import *
from tkinter import messagebox
from functools import partial
import sqlite3
import pymysql
from venv import create
from tkinter.ttk import *
from tkcalendar import Calendar, DateEntry
import re
from datetime import datetime
import pandas as pd
from pandastable import Table, TableModel
root=None
```

In [8]:

```
def orders():
    if(root):
        root.mainloop()
    %run addorders
def agents():
    if(root):
        root.mainloop()
    %run addagents
def company():
    if(root):
        root.mainloop()
    %run addcompany
def customer():
    if(root):
        root.mainloop()
    %run addcustomer
```

In [9]:

```
def options():
    tkWd = Tk()
    tkWd.geometry('200x150+600+300')
    tkWd.title('Select Table to add data to')

    ordersButton = Button(tkWd, text="Orders", command=orders).grid(row=2, column=1)
    agentsButton = Button(tkWd, text="Agents", command=agents).grid(row=3, column=1)
    companyButton = Button(tkWd, text="Company", command=company).grid(row=4, column=1)
    customerButton = Button(tkWd, text="Customer", command=customer).grid(row=5, column=
1)

    tkWd.mainloop()
```

In [10]:

```
def Login(username, password):
    c.execute(search_by_name, (username.get(),))
    print()
    user = c.fetchall()
    if len(user) == 0:
         messagebox.showerror("Error", "User not found")
    else:
        c.execute(validate , (username.get() , password.get()))
        validated = c.fetchall()
        if len(validated) == 0:
            messagebox.showerror("Error", "Wrong password")
        else:
            messagebox.showinfo("Login Done", "Logged in !")

            options()
```

In [11]:

```
def Register():

    Window = Toplevel(tkWd)

    def Submit(username, password):
        c.execute(create_user, (username.get(), password.get(),))
        conn.commit()
        if messagebox.showinfo("User created" , "Please log in !"):
            Window.withdraw()

    Window.title("Register")

    # sets the geometry of toplevel
    Window.geometry("300x150")
    # username label and text entry box
    uLabel = Label(Window, text="User Name").grid(row=0, column=0)
    username = StringVar()
    uEntry = Entry(Window, textvariable=username).grid(row=0, column=1)

    # password label and password entry box
    pLabel = Label(Window,text="Password").grid(row=1, column=0)
    password = StringVar()
    pEntry = Entry(Window, textvariable=password, show='*').grid(row=1, column=1)
    Submit = partial(Submit, username, password)
    Button(Window, text="Submit", command=Submit).grid(row=4, column=1)
```

In [12]:

```
conn = sqlite3.connect('cred.db')
c = conn.cursor()
create_table = """CREATE TABLE IF NOT EXISTS user(username TEXT NOT NULL, password TEXT N
OT NULL)"""
create_user = """INSERT INTO user(username,password) VALUES(?,?)"""
search_by_name = """SELECT * FROM user where username=?"""
validate = """SELECT * FROM user where username=? and password=?"""
c.execute(create_table)

# window
tkWd = Tk()
tkWd.geometry('300x150+600+300')
tkWd.title('Login App')

# username label and text entry box
uLabel = Label(tkWd, text="User Name").grid(row=0, column=0)
username = StringVar()
uEntry = Entry(tkWd, textvariable=username).grid(row=0, column=1)

# password label and password entry box
pLabel = Label(tkWd,text="Password").grid(row=1, column=0)
password = StringVar()

pEntry = Entry(tkWd, textvariable=password, show='*').grid(row=1, column=1)

Login = partial(Login, username, password)

# login , register buttons
loginButton = Button(tkWd, text="Login", command=Login).grid(row=4, column=1)
regButton = Button(tkWd, text="Register", command=Register).grid(row=5, column=1)

tkWd.mainloop()
```

```
#addorders.py

import tkinter
from tkinter import *
from tkinter import messagebox
from functools import partial
import sqlite3
import pymysql
from venv import create
from tkinter.ttk import *
from tkcalendar import Calendar, DateEntry
import re
from datetime import datetime
import pandas as pd
from pandastable import Table, TableModel

# Saving student form data
def addOrders():
    # Checking last input for validating , if not validated , shows error message
    if (len(ord_desc.get()) != 3 or ord_desc.get().isalpha()) == False:
        messagebox.showinfo("Save" , "Not Validated!")
        return

    try:
        # processing for three date variables : Birth Date , Start Date , End Date
        ordDateObj = datetime.strptime(ord_date.get(), '%d/%m/%Y')

        # Connecetion for mysql database
        conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
        cur = conn.cursor()
        # Excuting insert query
        cur.execute("""insert into orders(ORD_NUM ,ORD_AMOUNT,ADVANCE_AMOUNT,
ORD_DATE,CUST_CODE,AGENT_CODE,ORD_DESCRIPTION) values(%s,%s,%s,%s,%s,%s,%s)""" ,
                    (ord_num.get(),ord_amt.get(),adv_amt.get(),ordDateObj.strftime("%Y-
%m-%d"),cust_code.get(),agent_code.get(),ord_desc.get()))
        conn.close()
        # Show message for successing
        messagebox.showinfo("Save" , "Success!")
        b1['state']= 'enabled'
        # Initializing for each input.
        ord_amtEntry.config(state='disabled')
        adv_amtEntry.config(state='disabled')
        cust_codeEntry.config(state='disabled')
        agent_codeEntry.config(state='disabled')
        ord_descEntry.config(state='disabled')
    except Exception as e:
        print(e)
        # If error on saving , shows error message.
        messagebox.showerror("Save" ,"Failed to save!")
# When clicking cancel button , application will be closed.
def cancel():
    ord_num.set("")
    ord_amt.set("")
    adv_amt.set("")
    cust_code.set("")
    agent_code.set("")
    ord_desc.set("")
    root.destroy()

# Validating for each input
def validate(event , input):
    if( input == "Order Number"):
        ord_number = ord_num.get()
        if (len(ord_number) != 6 or ord_number.isdigit()==False):
            messagebox.showerror("Invalid!" ,"Order number has to be a 6 digit number.")
            ord_numEntry.focus_set()
        else:
            ord_amtEntry.focus_set()
            ord_amtEntry.config(state='normal')
    elif(input == "Order Amount"):
        if (ord_amt.get().isdigit()==True and float(ord_amt.get()) > 0 ):
            adv_amtEntry.focus_set()
            adv_amtEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Order amount has to be a number greater than 0.")
            ord_amtEntry.focus_set()
    elif( input == "Advance Amount"):
        if (adv_amt.get().isdigit()==True and float(adv_amt.get()) >= 0 and float(adv_amt.get()) <=float(ord_amt.get()) ):
            cust_codeEntry.focus_set()
            cust_codeEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Advance amount has to be a number greater than 0 and not greater than Order
Amount.")
            adv_amtEntry.focus_set()
    elif( input == "Customer Code"):
        if (len(cust_code.get()) == 6 and cust_code.get().isalnum()==True):
            agent_codeEntry.focus_set()
            agent_codeEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Customer code has to be alphanumberic with length 6.")
            cust_codeEntry.focus_set()
    elif( input == "Agent Code"):
        if (len(agent_code.get()) == 4 and agent_code.get().isalnum()==True):
            ord_descEntry.focus_set()
            ord_descEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Agent code has to be alphanumberic with length 4.")
            agent_codeEntry.focus_set()
    elif( input == "Order Description"):
        if len(ord_desc.get()) == 3 and ord_desc.get().isalpha():
            pass
        else:
            messagebox.showerror("Invalid!" ,"Order description has to be only three letters long.")
            ord_descEntry.focus_set()

def readdata():
    readwindow = Tk()
    readwindow.title("Read Order Data")
    readwindow.geometry('{}x{}'.format(800, 600))

    mainframe1 = Frame(readwindow)
    l = Label(readwindow, text='Here are the Results',font=('times', 20, 'bold'),background = '#154360',foreground = '#FDFEFE')
    l.place(x = 200, y = 10)

    df = pd.DataFrame()
    df = TableModel.getSampleData()


    conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
    cur = conn.cursor()
        # Excuting insert query
    query = "select ORD_NUM,ORD_DATE,ORD_DESCRIPTION,CUST_CODE,AGENT_CODE from orders"
    cur.execute(query)
    df = pd.DataFrame(list(cur.fetchall()),columns
=['ORD_NUM','ORD_DATE','ORD_DESCRIPTION','CUST_CODE','AGENT_CODE'])
    #print (df)

    table =Table(mainframe1, dataframe=df,showtoolbar=True, showstatusbar=True )
    table.currwidth = 700
    table.currheight = 500
    mainframe1.place(x = 200,y =200,anchor = "w")
    try:
        table.show()
    except:
        pass
    conn.close()


# Creating main window and setting with width and height
root = Tk()
root.title("Orders")
root.geometry('{}x{}'.format(600, 500))
mainframe = Frame(root)
mainframe.pack()

# Setting string variable for 6 input
ord_num = StringVar()
ord_amt = StringVar()
adv_amt = StringVar()
cust_code = StringVar()
agent_code = StringVar()
ord_desc = StringVar()
# Input for Order Number
ord_numEntry = Entry(mainframe, width=20, textvariable=ord_num)
ord_numEntry.grid(row=0, column=1 ,padx=5, pady=5)
ord_numEntry.bind("<Return>", lambda event: validate(event, "Order Number"))
ord_numEntry.bind("<Tab>", lambda event: validate(event, "Order Number"))
# Input for Order Amount
ord_amtEntry = Entry(mainframe, width=20, textvariable=ord_amt)
ord_amtEntry.grid(row=1, column=1 ,padx=5, pady=5)
ord_amtEntry.bind("<Return>", lambda event: validate(event, "Order Amount"))
ord_amtEntry.bind("<Tab>", lambda event: validate(event, "Order Amount"))
# Input for Advance Amount
adv_amtEntry = Entry(mainframe, width=20, textvariable=adv_amt)
adv_amtEntry.grid(row=2, column=1 ,padx=5, pady=5)
adv_amtEntry.bind("<Return>", lambda event: validate(event, "Advance Amount"))
adv_amtEntry.bind("<Tab>", lambda event: validate(event, "Advance Amount"))
# Input for Customer Code
cust_codeEntry = Entry(mainframe, width=20, textvariable=cust_code)
cust_codeEntry.grid(row=3, column=1 ,padx=5, pady=5)
cust_codeEntry.bind("<Return>", lambda event: validate(event, "Customer Code"))
cust_codeEntry.bind("<Tab>", lambda event: validate(event, "Customer Code"))
# Input for Agent Code
agent_codeEntry = Entry(mainframe, width=20, textvariable=agent_code)
agent_codeEntry.grid(row=4, column=1 ,padx=5, pady=5)
agent_codeEntry.bind("<Return>", lambda event: validate(event, "Agent Code"))
agent_codeEntry.bind("<Tab>", lambda event: validate(event, "Agent Code"))
# Input for Order Description
ord_descEntry = Entry(mainframe, width=20, textvariable=ord_desc)
ord_descEntry.grid(row=5, column=1 ,padx=5, pady=5)
ord_descEntry.bind("<Return>", lambda event: validate(event, "Order Description"))
ord_descEntry.bind("<Tab>", lambda event: validate(event, "Order Description"))

# First rest 5 inputs will be disabled for checking validation

ord_amtEntry.config(state='disabled')
adv_amtEntry.config(state='disabled')
cust_codeEntry.config(state='disabled')
agent_codeEntry.config(state='disabled')
ord_descEntry.config(state='disabled')

# Date picker for start date ,  end date
ord_date = StringVar()
DateEntry(mainframe , textvariable = ord_date , date_pattern='dd/mm/y' ).grid(row=7, column=1, padx=5, pady=5)

# Setting labels for each input
Label(mainframe, text='Order Number:*', anchor='w').grid(row=0, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Order Amount:*', anchor='w').grid(row=1, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Advance Amount:*', anchor='w').grid(row=2, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Customer Code:*', anchor='w').grid(row=3, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Agent Code:', anchor='w').grid(row=4, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Order Description:', anchor='w').grid(row=5, column=0 ,padx=5, pady=5, sticky="w")

Label(mainframe, text='Order Date:*', anchor='w').grid(row=7, column=0 ,padx=5, pady=5, sticky="w")

# Buttons for submit and cancel
btnFrame = Frame(mainframe)
Button(btnFrame, text="Submit", command=addOrders).grid(row=0, column=1, padx=5, pady=5)
Button(btnFrame, text="Cancel", command=cancel).grid(row=0, column=2, padx=5, pady=5)
b1 = Button(btnFrame, text="Read Data",command= readdata )

b1.grid(row=0, column=3, padx=5, pady=5 )
btnFrame.grid(row=10, column=1, padx=5, pady=5)



root.mainloop()
```

```python
#addagents.py

import tkinter
from tkinter import *
from tkinter import messagebox
from functools import partial
import sqlite3
import pymysql
from venv import create
from tkinter.ttk import *
from tkcalendar import Calendar, DateEntry
import re
from datetime import datetime
import pandas as pd
from pandastable import Table, TableModel

def addAgents():

   try:

       conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
       cur = conn.cursor()

       cur.execute("""insert into agents(AGENT_CODE ,AGENT_NAME,WORKING_AREA,
COMMISSION,PHONE_NO,COUNTRY) values(%s,%s,%s,%s,%s,%s)""" ,
                  (agent_code.get(),agent_name.get(),working_area.get(),commission.get(),phone_no.get(),country.get()))
       conn.close()

       messagebox.showinfo("Save" , "Success!")
       b1['state']= 'enabled'

       agent_nameEntry.config(state='disabled')
       working_areaEntry.config(state='disabled')
       commissionEntry.config(state='disabled')
       phone_noEntry.config(state='disabled')
       countryEntry.config(state='disabled')
   except Exception as e:
       print(e)

       messagebox.showerror("Save" ,"Failed to save!")

def cancel():
   agent_code.set("")
   agent_name.set("")
   working_area.set("")
   commission.set("")
   phone_no.set("")
   country.set("")
   root.destroy()


def validate(event , input):
   if( input == "Agent Code"):
       agentcode = agent_code.get()
       if (len(agentcode) == 4 and agentcode.isalnum()==True):
          agent_nameEntry.focus_set()
          agent_nameEntry.config(state='normal')
       else:
          messagebox.showerror("Invalid!" ,"Agent code has to be alphanumeric with length 4.")
          agent_codeEntry.focus_set()
   elif(input == "Agent Name"):
       if (agent_name.get().isalpha()==True and len(agent_name.get())>1):
          working_areaEntry.focus_set()
          working_areaEntry.config(state='normal')
       else:
            messagebox.showerror("Invalid!" ,"Agent Name has to be longer than 1 letter.")
            agent_nameEntry.focus_set()
    elif(input == "Working Area"):
        if (working_area.get().isalpha()==True and len(working_area.get())>1 ):
            commissionEntry.focus_set()
            commissionEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Working Area has to be longer than 1 letter.")
            working_areaEntry.focus_set()
    elif( input == "Commission"):
        if (commission.get().replace('.', '', 1).isdigit()==True):
            phone_noEntry.focus_set()
            phone_noEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Commission has to be in decimal format (e.g., 0.11 for 11%).")
            commissionEntry.focus_set()
    elif( input == "Phone Number"):
        phn = phone_no.get().replace('-', '', 1)
        if (len(phn) == 11 and phn.isdigit()==True):
            countryEntry.focus_set()
            countryEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Phone Number has to be 11 digits long.")
            phone_noEntry.focus_set()
    elif( input == "Country"):
        if len(country.get()) > -1 and country.get().isalpha()==True:
            pass
        else:
            messagebox.showerror("Invalid!" ,"Country has to be written using alphabets only.")
            countryEntry.focus_set()

def readagents():
    readwindow = Tk()
    readwindow.title("Read Agent Data")
    readwindow.geometry('{}x{}'.format(800, 600))

    mainframe1 = Frame(readwindow)
    l = Label(readwindow, text='Here are the Results',font=('times', 20, 'bold'),background = '#154360',foreground = '#FDFEFE')
    l.place(x = 200, y = 10)

    df = pd.DataFrame()
    df = TableModel.getSampleData()


    conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
    cur = conn.cursor()

    query = "select AGENT_CODE,AGENT_NAME,WORKING_AREA,PHONE_NO from agents"
    cur.execute(query)
    df = pd.DataFrame(list(cur.fetchall()),columns =['AGENT_CODE','AGENT_NAME','WORKING_AREA','PHONE_NO'])


    table =Table(mainframe1, dataframe=df,showtoolbar=True, showstatusbar=True )
    table.currwidth = 700
    table.currheight = 500
    mainframe1.place(x = 200,y =200,anchor = "w")
    try:
        table.show()
    except:
        pass
    conn.close()



root = Tk()
root.title("Agents")
root.geometry('{}x{}'.format(600, 500))
```

```
mainframe = Frame(root)
mainframe.pack()

agent_code = StringVar()
agent_name = StringVar()
working_area = StringVar()
commission = StringVar()
phone_no = StringVar()
country = StringVar()

agent_codeEntry = Entry(mainframe, width=20, textvariable=agent_code)
agent_codeEntry.grid(row=0, column=1 ,padx=5, pady=5)
agent_codeEntry.bind("<Return>", lambda event: validate(event, "Agent Code"))
agent_codeEntry.bind("<Tab>", lambda event: validate(event, "Agent Code"))

agent_nameEntry = Entry(mainframe, width=20, textvariable=agent_name)
agent_nameEntry.grid(row=1, column=1 ,padx=5, pady=5)
agent_nameEntry.bind("<Return>", lambda event: validate(event, "Agent Name"))
agent_nameEntry.bind("<Tab>", lambda event: validate(event, "Agent Name"))

working_areaEntry = Entry(mainframe, width=20, textvariable=working_area)
working_areaEntry.grid(row=2, column=1 ,padx=5, pady=5)
working_areaEntry.bind("<Return>", lambda event: validate(event, "Working Area"))
working_areaEntry.bind("<Tab>", lambda event: validate(event, "Working Area"))

commissionEntry = Entry(mainframe, width=20, textvariable=commission)
commissionEntry.grid(row=3, column=1 ,padx=5, pady=5)
commissionEntry.bind("<Return>", lambda event: validate(event, "Commission"))
commissionEntry.bind("<Tab>", lambda event: validate(event, "Commission"))

phone_noEntry = Entry(mainframe, width=20, textvariable=phone_no)
phone_noEntry.grid(row=4, column=1 ,padx=5, pady=5)
phone_noEntry.bind("<Return>", lambda event: validate(event, "Phone Number"))
phone_noEntry.bind("<Tab>", lambda event: validate(event, "Phone Number"))

countryEntry = Entry(mainframe, width=20, textvariable=country)
countryEntry.grid(row=5, column=1 ,padx=5, pady=5)
countryEntry.bind("<Return>", lambda event: validate(event, "Country"))
countryEntry.bind("<Tab>", lambda event: validate(event, "Country"))

agent_nameEntry.config(state='disabled')
working_areaEntry.config(state='disabled')
commissionEntry.config(state='disabled')
phone_noEntry.config(state='disabled')
countryEntry.config(state='disabled')

Label(mainframe, text='Agent Code:*', anchor='w').grid(row=0, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Agent Name:*', anchor='w').grid(row=1, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Working Area:*', anchor='w').grid(row=2, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Commission:*', anchor='w').grid(row=3, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Phone Number:*', anchor='w').grid(row=4, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Country:*', anchor='w').grid(row=5, column=0 ,padx=5, pady=5, sticky="w")

btnFrame = Frame(mainframe)
Button(btnFrame, text="Submit", command=addAgents).grid(row=0, column=1, padx=5, pady=5)
Button(btnFrame, text="Cancel", command=cancel).grid(row=0, column=2, padx=5, pady=5)
b1 = Button(btnFrame, text="Read Data",command=readagents)

b1.grid(row=0, column=3, padx=5, pady=5 )
btnFrame.grid(row=10, column=1, padx=5, pady=5)



root.mainloop()
```

```python
#addcompany.py

import tkinter
from tkinter import *
from tkinter import messagebox
from functools import partial
import sqlite3
import pymysql
from venv import create
from tkinter.ttk import *
from tkcalendar import Calendar, DateEntry
import re
from datetime import datetime
import pandas as pd
from pandastable import Table, TableModel

def addCompanies():

    if len(comp_city.get()) < 0 or comp_city.get().isalpha()==False:
        messagebox.showinfo("Save" , "Not Validated!")
        return
    try:
        conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
        cur = conn.cursor()

        cur.execute("""insert into company(COMPANY_ID, COMPANY_NAME,COMPANY_CITY) values(%s,%s,%s)""" ,
                (comp_id.get(),comp_name.get(),comp_city.get()))
        conn.close()

        messagebox.showinfo("Save" , "Success!")
        b1['state']= 'enabled'

        comp_nameEntry.config(state='disabled')
        comp_cityEntry.config(state='disabled')

    except Exception as e:
        print(e)

        messagebox.showerror("Save" ,"Failed to save!")

def cancel():
    comp_id.set("")
    comp_name.set("")
    comp_city.set("")
    root.destroy()

def validate(event , input):

    if( input == "Company ID"):
        compid = comp_id.get()
        if (len(compid) != 2 or compid.isdigit()==False):
            messagebox.showerror("Invalid!" ,"Company ID has to be numeric with length 2.")
            comp_idEntry.focus_set()
        else:
            comp_nameEntry.focus_set()
            comp_nameEntry.config(state='normal')
    elif(input == "Company Name"):
        if (comp_name.get().isdigit()==False and len(comp_name.get())>1):
            comp_cityEntry.focus_set()
            comp_cityEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Company Name has to be longer that 1 character.")
            agent_nameEntry.focus_set()
    elif(input == "Company City"):
        if (comp_city.get().isalpha()==True and len(comp_city.get())>1 ):
            pass
        else:
            messagebox.showerror("Invalid!" ,"Company City has to be longer than 1 letter.")
            comp_cityEntry.focus_set()


def readcompanies():
    readwindow = Tk()
    readwindow.title("Read Company Data")
    readwindow.geometry('{}x{}'.format(800, 600))

    mainframe1 = Frame(readwindow)
    l = Label(readwindow, text='Here are the Results',font=('times', 20, 'bold'),background = '#154360',foreground = '#FDFEFE')
    l.place(x = 200, y = 10)

    df = pd.DataFrame()
    df = TableModel.getSampleData()


    conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
    cur = conn.cursor()

    query = "select COMPANY_ID,COMPANY_NAME,COMPANY_CITY from company"
    cur.execute(query)
    df = pd.DataFrame(list(cur.fetchall()),columns =['COMPANY_ID','COMPANY_NAME','COMPANY_CITY'])


    table =Table(mainframe1, dataframe=df,showtoolbar=True, showstatusbar=True )
    table.currwidth = 700
    table.currheight = 500
    mainframe1.place(x = 200,y =200,anchor = "w")
    try:
        table.show()
    except:
        pass
    conn.close()


root = Tk()
root.title("Companies")
root.geometry('{}x{}'.format(600, 500))
mainframe = Frame(root)
mainframe.pack()

comp_id = StringVar()
comp_name = StringVar()
comp_city = StringVar()

comp_idEntry = Entry(mainframe, width=20, textvariable=comp_id)
comp_idEntry.grid(row=0, column=1 ,padx=5, pady=5)
comp_idEntry.bind("<Return>", lambda event: validate(event, "Company ID"))
comp_idEntry.bind("<Tab>", lambda event: validate(event, "Company ID"))

comp_nameEntry = Entry(mainframe, width=20, textvariable=comp_name)
comp_nameEntry.grid(row=1, column=1 ,padx=5, pady=5)
comp_nameEntry.bind("<Return>", lambda event: validate(event, "Company Name"))
comp_nameEntry.bind("<Tab>", lambda event: validate(event, "Company Name"))

comp_cityEntry = Entry(mainframe, width=20, textvariable=comp_city)
comp_cityEntry.grid(row=2, column=1 ,padx=5, pady=5)
comp_cityEntry.bind("<Return>", lambda event: validate(event, "Company City"))
comp_cityEntry.bind("<Tab>", lambda event: validate(event, "Company City"))

comp_nameEntry.config(state='disabled')
comp_cityEntry.config(state='disabled')
```

```
Label(mainframe, text='Company ID:*', anchor='w').grid(row=0, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Company Name:*', anchor='w').grid(row=1, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Company City:*', anchor='w').grid(row=2, column=0 ,padx=5, pady=5, sticky="w")



btnFrame = Frame(mainframe)
Button(btnFrame, text="Submit", command=addCompanies).grid(row=0, column=1, padx=5, pady=5)
Button(btnFrame, text="Cancel", command=cancel).grid(row=0, column=2, padx=5, pady=5)
b1 = Button(btnFrame, text="Read Data",command=readcompanies)

b1.grid(row=0, column=3, padx=5, pady=5 )
btnFrame.grid(row=10, column=1, padx=5, pady=5)



root.mainloop()
```

```python
#addcustomer.py

import tkinter
from tkinter import *
from tkinter import messagebox
from functools import partial
import sqlite3
import pymysql
from venv import create
from tkinter.ttk import *
from tkcalendar import Calendar, DateEntry
import re
from datetime import datetime
import pandas as pd
from pandastable import Table, TableModel

def addCustomers():

    if len(agent_code.get()) != 4 or agent_code.get().isalnum()==False:
        messagebox.showinfo("Save" , "Not Validated!")
        return
    try:
        conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
        cur = conn.cursor()
        cur.execute("""insert into customer(CUST_CODE ,CUST_NAME,CUST_CITY,
WORKING_AREA,CUST_COUNTRY,GRADE,OPENING_AMT,RECEIVE_AMT,PAYMENT_AMT,OUTSTANDING_AMT,PHONE_NO,AGENT_CODE)
                    values(%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)""" ,

(cust_code.get(),cust_name.get(),cust_city.get(),working_area.get(),cust_country.get(),grade.get(),opening_amt.get(),receive_amt.get(),payment_amt.get(),out_amt.get(),phone_no.get(),agent_code.get()))

        conn.close()
        messagebox.showinfo("Save" , "Success!")
        b1['state']= 'enabled'
        cust_nameEntry.config(state='disabled')
        cust_cityEntry.config(state='disabled')
        working_areaEntry.config(state='disabled')
        cust_countryEntry.config(state='disabled')
        gradeEntry.config(state='disabled')
        opening_amtEntry.config(state='disabled')
        receive_amtEntry.config(state='disabled')
        payment_amtEntry.config(state='disabled')
        out_amtEntry.config(state='disabled')
        phone_noEntry.config(state='disabled')
        agent_codeEntry.config(state='disabled')

    except Exception as e:
        print(e)

        messagebox.showerror("Save" ,"Failed to save!")

def cancel():
    cust_code.set("")
    cust_name.set("")
    cust_city.set("")
    working_area.set("")
    cust_country.set("")
    grade.set("")
    opening_amt.set("")
    receive_amt.set("")
    payment_amt.set("")
    out_amt.set("")
    phone_no.set("")
    agent_code.set("")
    root.destroy()

def validate(event , input):
    if( input == "Customer Code"):
        custcode = cust_code.get()
        if (len(custcode) == 6 and custcode.isalnum()==True):
            cust_nameEntry.focus_set()
            cust_nameEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Customer code has to be alphanumeric with length 6.")
            cust_codeEntry.focus_set()
    elif(input == "Customer Name"):
        if (cust_name.get().isalpha()==True and len(cust_name.get())>0):
            cust_cityEntry.focus_set()
            cust_cityEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Customer Name has to be atleast 1 letter.")
            cust_nameEntry.focus_set()
    elif(input == "Customer City"):
        if (cust_city.get().isalpha()==True and len(cust_city.get())>1 ):
            working_areaEntry.focus_set()
            working_areaEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Customer city has to be longer than 1 letter.")
            cust_cityEntry.focus_set()
    elif( input == "Working Area"):
        if (len(working_area.get()) >0 and working_area.get().isalpha()==True):
            cust_countryEntry.focus_set()
            cust_countryEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Working area has to be at least 1 letter.")
            working_areaEntry.focus_set()
    elif( input == "Customer Country"):
        if (len(cust_country.get()) >0 and cust_country.get().isalpha()==True):
            gradeEntry.focus_set()
            gradeEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Customer Country has to be at least than 1 letter.")
            cust_countryEntry.focus_set()
    elif( input == "Grade"):
        if grade.get().isdigit()==True:
            if (int(grade.get()) >=0 and int(grade.get())<=10):
                opening_amtEntry.focus_set()
                opening_amtEntry.config(state='normal')
            else:
                messagebox.showerror("Invalid!" ,"Grade should lie between 0-10.")
                gradeEntry.focus_set()
        else:
            messagebox.showerror("Invalid!" ,"Grade should be numeric.")
            gradeEntry.focus_set()
    elif( input == "Opening Amount"):
        if ((len(opening_amt.get()) >=2 and len(opening_amt.get())<=12) and opening_amt.get().isdigit()==True):
            receive_amtEntry.focus_set()
            receive_amtEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Opening amount has to be numeric with length 2-12.")
            opening_amtEntry.focus_set()
    elif(input == "Receive Amount"):
        if ((len(receive_amt.get()) >=2 and len(receive_amt.get())<=12) and receive_amt.get().isdigit()==True):
            payment_amtEntry.focus_set()
            payment_amtEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Receive amount has to be numeric with length 2-12.")
            receive_amtEntry.focus_set()
    elif(input == "Payment Amount"):
        if ((len(payment_amt.get()) >=2 and len(payment_amt.get())<=12) and payment_amt.get().isdigit()==True):
            out_amtEntry.focus_set()
            out_amtEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Payment amount has to be numeric with length 2-12.")
            payment_amtEntry.focus_set()
    elif( input == "Outstanding Amount"):
        if ((len(out_amt.get()) >=2 and len(out_amt.get())<=12) and out_amt.get().isdigit()==True):
            phone_noEntry.focus_set()
            phone_noEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Outstanding amount has to be numeric with length 2-12.")
            out_amtEntry.focus_set()
    elif( input == "Phone No"):
        phn = phone_no.get().replace('-', '', 1)
        if (len(phn) == 11 and phn.isdigit()==True):

            agent_codeEntry.focus_set()
            agent_codeEntry.config(state='normal')
        else:
            messagebox.showerror("Invalid!" ,"Phone Number must be 11 digits.")
            phone_noEntry.focus_set()
    elif( input == "Agent Code"):
        if (len(agent_code.get())==4 and agent_code.get().isalnum()==True):
            pass
        else:
            messagebox.showerror("Invalid!" ,"Agent Code must be alphanumeric with length 4.")
            agent_codeEntry.focus_set()

def readcustomers():
    readwindow = Tk()
    readwindow.title("Read Customer Data")
    readwindow.geometry('{}x{}'.format(800, 600))

    mainframe1 = Frame(readwindow)
    l = Label(readwindow, text='Here are the Results',font=('times', 20, 'bold'),background = '#154360',foreground = '#FDFEFE')
    l.place(x = 200, y = 10)

    df = pd.DataFrame()
    df = TableModel.getSampleData()


    conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
    cur = conn.cursor()

    query = "select CUST_CODE ,CUST_NAME,CUST_CITY,
WORKING_AREA,CUST_COUNTRY,GRADE,OPENING_AMT,RECEIVE_AMT,PAYMENT_AMT,OUTSTANDING_AMT,PHONE_NO,AGENT_CODE from customer"
    cur.execute(query)
    df = pd.DataFrame(list(cur.fetchall()),columns =['CUST_CODE' ,'CUST_NAME','CUST_CITY',
'WORKING_AREA','CUST_COUNTRY','GRADE','OPENING_AMT','RECEIVE_AMT','PAYMENT_AMT','OUTSTANDING_AMT','PHONE_NO','AGENT_CODE'])


    table =Table(mainframe1, dataframe=df,showtoolbar=True, showstatusbar=True )
    table.currwidth = 700
    table.currheight = 500
    mainframe1.place(x = 200,y =200,anchor = "w")
    try:
        table.show()
    except:
        pass
    conn.close()


root = Tk()
root.title("Customers")
root.geometry('{}x{}'.format(600, 600))
mainframe = Frame(root)
mainframe.pack()

cust_name = StringVar()
cust_city = StringVar()
working_area = StringVar()
cust_country = StringVar()
grade = StringVar()
opening_amt = StringVar()
receive_amt = StringVar()
payment_amt = StringVar()
```

```
out_amt = StringVar()
phone_no = StringVar()
agent_code = StringVar()
cust_code = StringVar()

cust_codeEntry = Entry(mainframe, width=20, textvariable=cust_code)
cust_codeEntry.grid(row=0, column=1 ,padx=5, pady=5)
cust_codeEntry.bind("<Return>", lambda event: validate(event, "Customer Code"))
cust_codeEntry.bind("<Tab>", lambda event: validate(event, "Customer Code"))

cust_nameEntry = Entry(mainframe, width=20, textvariable=cust_name)
cust_nameEntry.grid(row=1, column=1 ,padx=5, pady=5)
cust_nameEntry.bind("<Return>", lambda event: validate(event, "Customer Name"))
cust_nameEntry.bind("<Tab>", lambda event: validate(event, "Customer Name"))

cust_cityEntry = Entry(mainframe, width=20, textvariable=cust_city)
cust_cityEntry.grid(row=2, column=1 ,padx=5, pady=5)
cust_cityEntry.bind("<Return>", lambda event: validate(event, "Customer City"))
cust_cityEntry.bind("<Tab>", lambda event: validate(event, "Customer City"))

working_areaEntry = Entry(mainframe, width=20, textvariable=working_area)
working_areaEntry.grid(row=3, column=1 ,padx=5, pady=5)
working_areaEntry.bind("<Return>", lambda event: validate(event, "Working Area"))
working_areaEntry.bind("<Tab>", lambda event: validate(event, "Working Area"))

cust_countryEntry = Entry(mainframe, width=20, textvariable=cust_country)
cust_countryEntry.grid(row=4, column=1 ,padx=5, pady=5)
cust_countryEntry.bind("<Return>", lambda event: validate(event, "Customer Country"))
cust_countryEntry.bind("<Tab>", lambda event: validate(event, "Customer Country"))

gradeEntry = Entry(mainframe, width=20, textvariable=grade)
gradeEntry.grid(row=5, column=1 ,padx=5, pady=5)
gradeEntry.bind("<Return>", lambda event: validate(event, "Grade"))
gradeEntry.bind("<Tab>", lambda event: validate(event, "Grade"))

opening_amtEntry = Entry(mainframe, width=20, textvariable=opening_amt)
opening_amtEntry.grid(row=6, column=1 ,padx=5, pady=5)
opening_amtEntry.bind("<Return>", lambda event: validate(event, "Opening Amount"))
opening_amtEntry.bind("<Tab>", lambda event: validate(event, "Opening Amount"))

receive_amtEntry = Entry(mainframe, width=20, textvariable=receive_amt)
receive_amtEntry.grid(row=7, column=1 ,padx=5, pady=5)
receive_amtEntry.bind("<Return>", lambda event: validate(event, "Receive Amount"))
receive_amtEntry.bind("<Tab>", lambda event: validate(event, "Receive Amount"))

payment_amtEntry = Entry(mainframe, width=20, textvariable=payment_amt)
payment_amtEntry.grid(row=8, column=1 ,padx=5, pady=5)
payment_amtEntry.bind("<Return>", lambda event: validate(event, "Payment Amount"))
payment_amtEntry.bind("<Tab>", lambda event: validate(event, "Payment Amount"))

out_amtEntry = Entry(mainframe, width=20, textvariable=out_amt)
out_amtEntry.grid(row=9, column=1 ,padx=5, pady=5)
out_amtEntry.bind("<Return>", lambda event: validate(event, "Outstanding Amount"))
out_amtEntry.bind("<Tab>", lambda event: validate(event, "Outstanding Amount"))

phone_noEntry = Entry(mainframe, width=20, textvariable=phone_no)
phone_noEntry.grid(row=10, column=1 ,padx=5, pady=5)
phone_noEntry.bind("<Return>", lambda event: validate(event, "Phone No"))
phone_noEntry.bind("<Tab>", lambda event: validate(event, "Phone No"))

agent_codeEntry = Entry(mainframe, width=20, textvariable=agent_code)
agent_codeEntry.grid(row=11, column=1 ,padx=5, pady=5)
agent_codeEntry.bind("<Return>", lambda event: validate(event, "Agent Code"))
agent_codeEntry.bind("<Tab>", lambda event: validate(event, "Agent Code"))


cust_nameEntry.config(state='disabled')
cust_cityEntry.config(state='disabled')
working_areaEntry.config(state='disabled')
cust_countryEntry.config(state='disabled')
gradeEntry.config(state='disabled')
opening_amtEntry.config(state='disabled')
receive_amtEntry.config(state='disabled')
payment_amtEntry.config(state='disabled')
out_amtEntry.config(state='disabled')
phone_noEntry.config(state='disabled')
agent_codeEntry.config(state='disabled')

Label(mainframe, text='Customer Code:*', anchor='w').grid(row=0, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Customer Name:*', anchor='w').grid(row=1, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Customer City:*', anchor='w').grid(row=2, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Working Area:*', anchor='w').grid(row=3, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Customer Country:*', anchor='w').grid(row=4, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Grade:*', anchor='w').grid(row=5, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Opening Amount:*', anchor='w').grid(row=6, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Receive Amount:*', anchor='w').grid(row=7, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Payment Amount:*', anchor='w').grid(row=8, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Outstanding Amount:*', anchor='w').grid(row=9, column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Phone Number:*', anchor='w').grid(row=10,column=0 ,padx=5, pady=5, sticky="w")
Label(mainframe, text='Agent Code:*', anchor='w').grid(row=11,column=0 ,padx=5, pady=5, sticky="w")

btnFrame = Frame(mainframe)
Button(btnFrame, text="Submit", command=addCustomers).grid(row=15, column=1, padx=5, pady=5)
Button(btnFrame, text="Cancel", command=cancel).grid(row=15, column=2, padx=5, pady=5)
b1 = Button(btnFrame, text="Read Data",command=readcustomers)

b1.grid(row=15, column=3, padx=5, pady=5 )
btnFrame.grid(row=15, column=1, padx=5, pady=5)


root.mainloop()
```

Part-A2,A3.A4.ipynb

In [1]:

```
import pymysql
import tkinter
from tkinter import *
from tkinter import messagebox
import pandas as pd
from pandastable import Table, TableModel
```

A2) The company needs an order look up (i.e. search) based on the following criteria,

a) Order number b) Order Date c) Customer code

In [2]:

```
# Creating main window and setting with width and height

print("Order Number:")
ordnum=input()

print("Order Date:")
orddate=input()

print("Customer Code:")
custcode=input()

df = TableModel.getSampleData()


conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
cur = conn.cursor()

if ordnum=="":
    if orddate=="":
        if custcode=="":
            query = ("select * from orders")
            cur.execute(query)
        else:
            query = ("select * from orders WHERE CUST_CODE=%s")
            cur.execute(query,(custcode))
    else:
        if custcode=="":
            query = ("select * from orders WHERE ORD_DATE=%s")
            cur.execute(query,(orddate))
        else:
            query = ("select * from orders WHERE ORD_DATE=%s and CUST_CODE=%s")
            cur.execute(query,(orddate,custcode))
else:
    if orddate=="":
        if custcode=="":
            query = ("select * from orders WHERE ORD_NUM = %s")
            cur.execute(query,(ordnum))
        else:
            query = ("select * from orders WHERE ORD_NUM = %s and CUST_CODE=%s")
            cur.execute(query,(ordnum,custcode))
    else:
        if custcode=="":
            query = ("select * from orders WHERE ORD_NUM = %s and ORD_DATE=%s")
            cur.execute(query,(ordnum, orddate))
        else:
            query = ("select * from orders WHERE ORD_NUM = %s and ORD_DATE=%s and CUST_C
ODE=%s")
```

```
            cur.execute(query,(ordnum, orddate,custcode))

df = pd.DataFrame(list(cur.fetchall()),columns =['ORD_NUM','ORD_AMOUNT','ADVANCE_AMOUNT'
,'ORD_DATE','CUST_CODE','AGENT_CODE','ORD_DESCRIPTION'])
print (df)

conn.close()
```

```
Order Number:
200135
Order Date:

Customer Code:

  ORD_NUM ORD_AMOUNT ADVANCE_AMOUNT    ORD_DATE CUST_CODE AGENT_CODE  \
0  200135    2000.00         800.00  2008-09-16    C00007      A010

  ORD_DESCRIPTION
0            SOD\r
```

A3) Generate a report that highlights the balance amounts for all orders in descending order. Do mention the name and code of the agent handling the order.

In [3]:

```
# Creating main window and setting with width and height

df = TableModel.getSampleData()

conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
cur = conn.cursor()

# query = ("""CREATE TABLE balance
#        AS (SELECT orders.ORD_NUM, orders.ORD_AMOUNT, orders.ADVANCE_AMOUNT, orders.AGENT
_CODE, agents.AGENT_NAME
#        FROM orders, agents
#        WHERE orders.AGENT_CODE = agents.AGENT_CODE)""")
# cur.execute(query)

query="select * from balance ORDER BY (ORD_AMOUNT - ADVANCE_AMOUNT) DESC"
cur.execute(query)
df = pd.DataFrame(list(cur.fetchall()),columns =['ORD_NUM','ORD_AMOUNT','ADVANCE_AMOUNT'
,'AGENT_CODE','AGENT_NAME'])
print (df)

conn.close()
```

```
   ORD_NUM ORD_AMOUNT ADVANCE_AMOUNT AGENT_CODE  \
0   200107    4500.00         900.00     A010
1   200108    4000.00         600.00     A004
2   200113    4000.00         600.00     A002
3   200119    4000.00         700.00     A010
4   200109    3500.00         800.00     A010
5   200110    3000.00         500.00     A010
6   200134    4200.00        1800.00     A005
7   200122    2500.00         400.00     A004
8   200127    2500.00         400.00     A003
9   200130    2500.00         400.00     A011
10  200105    2500.00         500.00     A011
11  200128    3500.00        1500.00     A002
12  200129    2500.00         500.00     A006
13  200101    3000.00        1000.00     A008
14  200106    2500.00         700.00     A002
15  200102    2000.00         300.00     A012
16  200112    2000.00         400.00     A007
17  200114    3500.00        2000.00     A008
18  200125    2000.00         600.00     A005
19  200135    2000.00         800.00     A010
20  200104    1500.00         500.00     A004
21  200121    1500.00         600.00     A004
22  200103    1500.00         700.00     A005
```

```
23  200133    1200.00          400.00     A002
24  200131     900.00          150.00     A012
25  200111    1000.00          300.00     A008
26  200117     800.00          200.00     A001
27  200126     500.00          100.00     A002
28  200124     500.00          100.00     A007
29  200116     500.00          100.00     A009
30  200120     500.00          100.00     A002
31  200118     500.00          100.00     A006
32  200123     500.00          100.00     A002
33  200100    1000.00          600.00     A003

                                   AGENT_NAME
0   Santakumar
1   Ivan
2   Mukesh
3   Santakumar
4   Santakumar
5   Santakumar
6   Anderson
7   Ivan
8   Alex
9   Ravi Kumar
10  Ravi Kumar
11  Mukesh
12  McDen
13  Alford
14  Mukesh
15  Lucida
16  Ramasundar
17  Alford
18  Anderson
19  Santakumar
20  Ivan
21  Ivan
22  Anderson
23  Mukesh
24  Lucida
25  Alford
26  Subbarao
27  Mukesh
28  Ramasundar
29  Benjamin
30  Mukesh
31  McDen
32  Mukesh
33  Alex
```

A4) Which is the country with maximum number of registered customer and what is the collective payment amount and outstanding amount for all these customers collectively.

In [4]:

```
# Creating main window and setting with width and height

df = TableModel.getSampleData()

conn = pymysql.connect(user="root", password="", host="localhost", database="sunville")
cur = conn.cursor()

query="select CUST_CODE, CUST_COUNTRY,PAYMENT_AMT,OUTSTANDING_AMT from customer"
cur.execute(query)
df = pd.DataFrame(list(cur.fetchall()),columns =['CUST_CODE', 'CUST_COUNTRY','PAYMENT_AM
T','OUTSTANDING_AMT'])
print (df)

conn.close()
```

```
   CUST_CODE CUST_COUNTRY PAYMENT_AMT OUTSTANDING_AMT
```

```
0     C00013            UK    7000.00          4000.00
1     C00001           USA    2000.00          6000.00
2     C00020           USA    6000.00          6000.00
3     C00025         India    4000.00          8000.00
4     C00024            UK    7000.00          6000.00
5     C00015            UK    3000.00         11000.00
6     C00002           USA    9000.00          3000.00
7     C00018     Australia    9000.00          5000.00
8     C00021     Australia    7000.00          7000.00
9     C00019         India    7000.00          8000.00
10    C00005         India    7000.00         11000.00
11    C00007         India    9000.00          9000.00
12    C00022         India    9000.00          9000.00
13    C00004     Australia    7000.00          6000.00
14    C00023            UK    7000.00          3000.00
15    C00006        Canada    6000.00         11000.00
16    C00010            UK    5000.00          5000.00
17    C00017         India    3000.00          9000.00
18    C00012           USA    9000.00          3000.00
19    C00008        Canada    9000.00          5000.00
20    C00003        Canada    7000.00          8000.00
21    C00009         India    3000.00         12000.00
22    C00014         India    7000.00         12000.00
23    C00016         India    7000.00         12000.00
24    C00011         India    7000.00         11000.00
```

In [5]:

```
cust_country = df.groupby("CUST_COUNTRY")
```

In [6]:

```
cust_country['CUST_CODE'].count()
```

Out[6]:

```
CUST_COUNTRY
Australia     3
Canada        3
India        10
UK            5
USA           4
Name: CUST_CODE, dtype: int64
```

In [7]:

```
cust_country['PAYMENT_AMT'].sum()
```

Out[7]:

```
CUST_COUNTRY
Australia    23000.00
Canada       22000.00
India        63000.00
UK           29000.00
USA          26000.00
Name: PAYMENT_AMT, dtype: object
```

In [8]:

```
cust_country['OUTSTANDING_AMT'].sum()
```

Out[8]:

```
CUST_COUNTRY
Australia     18000.00
Canada        24000.00
India        101000.00
UK            29000.00
USA           18000.00
Name: OUTSTANDING_AMT, dtype: object
```

Part-B-Insights.ipynb

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:

```
data = pd.read_excel("property.xlsx", skiprows=8,skipfooter =1)
```

In [3]:

```
data.head()
```

Out[3]:

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
0	2019	JUL	100 Mile House	170 Cedar Ave. S.	BC	CA	B0067295	1182.02	SQ-M	Leased	51.645900	Ramasundar	-121.293764
1	2019	JUL	100 Mile House	300 Cariboo Hwy	BC	CA	N0092260	0.36	HA	Owned	NaN	Ramasundar	NaN
2	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076976	1467.89	SQ-M	Owned	51.644508	Ramasundar	-121.297664
3	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076984	23.40	SQ-M	Owned	51.644222	Ramasundar	-121.299028
4	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0081810	215.50	SQ-M	Owned	51.644139	Ramasundar	-121.297392

B1) The total property area sold vs total property are leased in Sq-M only.

In [27]:

```
year = int(input("Enter the year: "))
```

```
Enter the year: 2018
```

In [5]:

```
d1 = data[(data["UoM"]=="SQ-M") & (data['Year']==year)]
```

In [6]:

```
d1.head()
```

Out[6]:

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
172	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002658	11.15	SQ-M	Owned	49.105522	Lucida	-121.363500
173	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002659	11.61	SQ-M	Owned	49.104260	Lucida	-121.634800

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
174	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002660	20.62	SQ-M	Owned	49.103180	Lucida	-121.634500
175	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002661	5.88	SQ-M	Owned	49.104572	Lucida	-121.635587
176	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002662	30.09	SQ-M	Owned	49.105490	Lucida	-121.634100

In [7]:

```
d1_leased = d1[d1["Tenure"] == "Leased"]
d1_owned  = d1[d1["Tenure"] == "Owned"]
totLeased = d1_leased["Area"].sum()
totOwned = d1_owned["Area"].sum()
print("In SQ-M\nTotal area Leased =",totLeased,"\nTotal area owned =",totOwned)
```

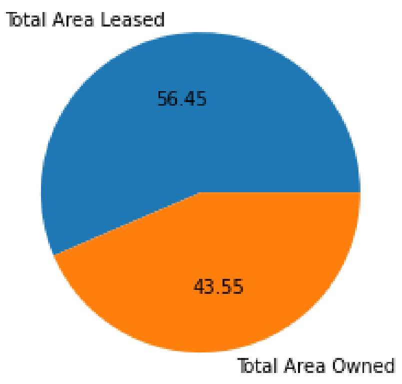
```
In SQ-M
Total area Leased = 73766.96
Total area owned = 56914.37000000001
```

In [8]:

```
plt.pie([totLeased,totOwned],labels= ["Total Area Leased","Total Area Owned"],autopct='%
.2f')
```

Out[8]:

```
([<matplotlib.patches.Wedge at 0x22b0000b6d0>,
  <matplotlib.patches.Wedge at 0x22b0000bdc0>],
 [Text(-0.2213050872731415, 1.077508263702431, 'Total Area Leased'),
  Text(0.22130508727314088, -1.0775082637024311, 'Total Area Owned')],
 [Text(-0.1207118657853499, 0.5877317802013259, '56.45'),
  Text(0.12071186578534956, -0.587731780201326, '43.55')])
```



B2) Of the years 2017,2018,2019- which year got maximum leased area in CA and WS countries.

In [9]:

```
data_CA = data[(data["Country"] == "CA") & (data["Tenure"] == "Leased")]
data_WS = data[(data["Country"] == "WS") & (data["Tenure"] == "Leased")]
data_WS.head()
```

Out[9]:

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
883	2018	JUL	Salmo Creston Summit	Summit Of Stagleap	BC	WS	B0092344	131.60	SQ-M	Leased	49.059076	Anderson	-117.039199
884	2018	JUL	Salmo Creston Summit	Summit Of Stagleap	BC	WS	N0001589	5.00	HA	Leased	NaN	Anderson	NaN

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
886	2018	JUL	Salmon Arm	351 Hudson Ave. N.E.	BC	WS	B0090378	191.94	SQ-M	Leased	50.702291	Anderson	-119.280780
887	2018	JUL	Salmon Arm	550 2nd. Ave N.E.	BC	WS	B0092308	1858.65	SQ-M	Leased	50.700583	Anderson	-119.279056
888	2018	JUL	Salmon Arm	550 Lakeshore Rd. N.E.	BC	WS	B0091854	824.98	SQ-M	Leased	50.704920	Anderson	-119.278596

In [10]:

```
data_CA_2017 = data_CA[data_CA["Year"] == 2017]
data_CA_2018 = data_CA[data_CA["Year"] == 2018]
data_CA_2019 = data_CA[data_CA["Year"] == 2019]
CA_2017 = data_CA_2017["Area"].sum()
CA_2018 = data_CA_2018["Area"].sum()
CA_2019 = data_CA_2019["Area"].sum()
print("Leased area in CA per year:")
print(f"2017: {CA_2017},\n2018: {CA_2018},\n2019: {CA_2019}")
```

```
Leased area in CA per year:
2017: 70660.0792,
2018: 66458.39110000001,
2019: 213945.70029999997
```

The maximum leased area in CA was in 2019.

In [11]:

```
data_WS_2017 = data_WS[data_WS["Year"] == 2017]
data_WS_2018 = data_WS[data_WS["Year"] == 2018]
data_WS_2019 = data_WS[data_WS["Year"] == 2019]
WS_2017 = data_WS_2017["Area"].sum()
WS_2018 = data_WS_2018["Area"].sum()
WS_2019 = data_WS_2019["Area"].sum()
print("Leased area in WS per year:")
print(f"2017: {WS_2017},\n2018: {WS_2018},\n2019: {WS_2019}")
```

```
Leased area in WS per year:
2017: 69052.36,
2018: 7328.419999999999,
2019: 62758.43609999999
```

The maximum leased area in WS was in 2017.

B3) What are the Agent codes of all the agents who have got deals in 'OWNED' categories across the years.

In [28]:

```
year = int(input("Enter the year: "))
```

```
Enter the year: 2018
```

In [13]:

```
data_owned = data[(data["Tenure"]== "Owned")&(data["Year"]==year)]
data_owned.head()
```

Out[13]:

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
172	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002658	11.15	SQ-M	Owned	49.105522	Lucida	-121.363500
				57657									

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
173	2018	JUL	Chilliwack	Chilliwack Lake Rd.	BC	WS	B1002659	11.61	SQ-M	Owned	49.104260	Lucida	-121.634800
174	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002660	20.62	SQ-M	Owned	49.103180	Lucida	-121.634500
175	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002661	5.88	SQ-M	Owned	49.104572	Lucida	-121.635587
176	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002662	30.09	SQ-M	Owned	49.105490	Lucida	-121.634100

In [14]:

```
data_owned = data_owned.groupby("Year")
data_owned.head()
```

Out[14]:

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
172	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002658	11.15	SQ-M	Owned	49.105522	Lucida	-121.363500
173	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002659	11.61	SQ-M	Owned	49.104260	Lucida	-121.634800
174	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002660	20.62	SQ-M	Owned	49.103180	Lucida	-121.634500
175	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002661	5.88	SQ-M	Owned	49.104572	Lucida	-121.635587
176	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002662	30.09	SQ-M	Owned	49.105490	Lucida	-121.634100

In [15]:

```
print(data_owned["Agent"].nunique())
d = data_owned["Agent"].unique()
for i in d:
    print(i)
```

```
Year
2018    3
Name: Agent, dtype: int64
['Lucida' 'Mukesh' 'Anderson']
```

B4) For the city of Chilliwack, which agent has got the maximum deals in leased form

In [16]:

```
data_chilliwack = data[data["City"] == "Chilliwack"]
data_chilliwack_leased = data_chilliwack[data_chilliwack["Tenure"] == "Leased"]
```

In [17]:

```
data_chilliwack_leased.groupby("Agent")
data_chilliwack_leased
```

Out[17]:

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
123	2019	JUL	Chilliwack	45467 Yale	BC	CA	B0091696	534.000	SQ-	Leased	49.148054	Alford	-

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
124	2019	JUL	Chilliwack	45540 Yale Rd.	BC	CA	B0092554	963.850	SQ-M	Leased	49.151503	Alford	-121.964110
125	2019	JUL	Chilliwack	45890 Victoria Ave.	BC	CA	B0067332	277.800	SQ-M	Leased	49.172572	Alford	-121.954536
126	2019	JUL	Chilliwack	45960 Wellington Ave.	BC	CA	B0091910	1014.780	SQ-M	Leased	49.171388	Alford	-121.953762
128	2019	JUL	Chilliwack	46360 Airport Rd.	BC	CA	B0091580	1882.000	SQ-M	Leased	49.155742	Alford	-121.941161
129	2019	JUL	Chilliwack	46360 Airport Rd.	BC	CA	B0092270	184.970	SQ-M	Leased	49.155425	Alford	-121.940508
130	2019	JUL	Chilliwack	46360 Airport Rd.	BC	CA	N0092272	0.458	HA	Leased	NaN	Alford	NaN
184	2018	JUL	Chilliwack	6640 Vedder Rd.	BC	CA	B1002341	305.740	SQ-M	Leased	49.123854	Lucida	-121.959926
185	2018	JUL	Chilliwack	8978 School St.	BC	CA	B0068140	1328.470	SQ-M	Leased	49.165331	Lucida	-121.960483

We can see that Alford has 7 and Lucida has 2 deals in leased form in the city of Chilliwack. Hence, Alford has the maximum deals of such kind.

B5) Compare the performance of all agents based on the area leased and owned for the years 2017,2018 and 2019. Who has been the best performer?

In [18]:

```
data_year = data.groupby(["Year","Agent"])
```

In [19]:

```
data_year.head()
```

Out[19]:

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitu
0	2019	JUL	100 Mile House	170 Cedar Ave. S.	BC	CA	B0067295	1182.0200	SQ-M	Leased	51.645900	Ramasundar	121.293
1	2019	JUL	100 Mile House	300 Cariboo Hwy	BC	CA	N0092260	0.3600	HA	Owned	NaN	Ramasundar	N
2	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076976	1467.8900	SQ-M	Owned	51.644508	Ramasundar	121.297
3	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076984	23.4000	SQ-M	Owned	51.644222	Ramasundar	121.299
4	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0081810	215.5000	SQ-M	Owned	51.644139	Ramasundar	121.297
...	...	...	...	...	...	...	...	...	...	...	...	...	
1110	2017	JUL	Victoria	1515 Blanshard St.	BC	CA	N0001031	0.7608	HA	Owned	NaN	Subbarao	N
1111	2017	JUL	Victoria	1520 Blanshard St.	BC	CA	B0063404	3396.1000	SQ-M	Leased	48.428250	Subbarao	123.362
				1675									

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitu
1112	2017	JUL	Victoria	1675 Douglas St.	BC	CA	B0067667	3390.8800	SQ-M	Leased	48.429104	Subbarao	123.3640
1113	2017	JUL	Victoria	1802 Douglas St.	BC	CA	B0067183	6376.3700	SQ-M	Leased	48.430528	Subbarao	123.3649
1114	2017	JUL	Victoria	1803 Douglas St.	BC	CA	B1002347	1233.9000	SQ-M	Leased	48.430393	Subbarao	123.3642

86 rows × 13 columns

In [20]:

```
data_year["Area"].sum()
```

Out[20]:

```
Year  Agent
2017  Alford                29593.8090
      Lucida                13914.3562
      Subbarao             197999.8562
2018  Anderson              81852.1996
      Lucida                 3916.4332
      Mukesh                44975.5461
2019  Alford                60342.3226
      Anderson              33344.7545
      Benjamin              94510.6454
      Lucida               153362.9382
      McDen                 77545.5536
      Mukesh                93555.1494
      Ramasundar            22763.6769
      Ramasundar                0.3600
      Ravi Kumar            67688.3404
      Subbarao              88292.7933
2020  Lucida               200186.4284
      Subbarao              37067.9420
Name: Area, dtype: float64
```

In 2017, Subbarao, in 2018, Anderson and in 2019 as well as in 2020, Lucida performed the best in terms of area leased or owned.

B6) What is the amount of property area sold for the month of july for all the years?

In [21]:

```
data_jul = data[data["Month"]=='JUL']
data_jul.head()
```

Out[21]:

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude
0	2019	JUL	100 Mile House	170 Cedar Ave. S.	BC	CA	B0067295	1182.02	SQ-M	Leased	51.645900	Ramasundar	-121.293764
1	2019	JUL	100 Mile House	300 Cariboo Hwy	BC	CA	N0092260	0.36	HA	Owned	NaN	Ramasundar	NaN
2	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076976	1467.89	SQ-M	Owned	51.644508	Ramasundar	-121.297664
3	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076984	23.40	SQ-M	Owned	51.644222	Ramasundar	-121.299028
4	2019	JUL	100 Mile House	300 Cariboo	BC	CA	B0081810	215.50	SQ-M	Owned	51.644139	Ramasundar	-121.297392

Year	Month	City	Hwy. Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longitude

In [22]:

```
data_jul=data_jul.groupby("Year")
data_jul.head()
```

Out[22]:

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longi
0	2019	JUL	100 Mile House	170 Cedar Ave. S.	BC	CA	B0067295	1182.0200	SQ-M	Leased	51.645900	Ramasundar	121.29
1	2019	JUL	100 Mile House	300 Cariboo Hwy	BC	CA	N0092260	0.3600	HA	Owned	NaN	Ramasundar	
2	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076976	1467.8900	SQ-M	Owned	51.644508	Ramasundar	121.29
3	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076984	23.4000	SQ-M	Owned	51.644222	Ramasundar	121.29
4	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0081810	215.5000	SQ-M	Owned	51.644139	Ramasundar	121.29
54	2017	JUL	Bella Coola	636 Cliff St.	BC	CA	N0001484	0.1561	HA	Owned	NaN	Alford	
55	2017	JUL	Bob Quinn Lake	Hwy. 37	BC	CA	N2000483	0.0279	HA	Leased	NaN	Alford	
56	2017	JUL	Bob Quinn Lake	Stewart Cassiar Hwy. 37	BC	CA	B1002402	55.1800	SQ-M	Owned	56.977742	Alford	130.25
57	2017	JUL	Bob Quinn Lake	Stewart Cassiar Hwy. 37	BC	CA	B1002403	55.1800	SQ-M	Owned	56.981039	Alford	130.24
58	2017	JUL	Burnaby	3133 Sumner Ave.	BC	CA	B0067103	552.1300	SQ-M	Leased	49.255998	Alford	123.00
172	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002658	11.1500	SQ-M	Owned	49.105522	Lucida	121.36
173	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002659	11.6100	SQ-M	Owned	49.104260	Lucida	121.63
174	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002660	20.6200	SQ-M	Owned	49.103180	Lucida	121.63
175	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002661	5.8800	SQ-M	Owned	49.104572	Lucida	121.63
176	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002662	30.0900	SQ-M	Owned	49.105490	Lucida	121.63
994	2020	JUL	Trail	1051 Farwell St.	BC	WS	B0068035	504.8000	SQ-M	Leased	49.097189	Lucida	117.70
995	2020	JUL	Trail	1520 Bay Ave.	BC	WS	B0067368	545.3000	SQ-M	Leased	49.094904	Lucida	117.70
996	2020	JUL	Trout Lake	Lardeau St.	BC	WS	B0086660	53.5000	SQ-M	Owned	50.646944	Lucida	117.54
997	2020	JUL	Trout Lake	Lardeau St.	BC	WS	B0086686	13.4000	SQ-M	Owned	50.646806	Lucida	117.54

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longi
998	2020	JUL	Trout Lake	Lardeau St.	BC	WS	B0086694	8.9000	SQ-M	Owned	50.646972	Lucida	117.54

In [23]:

```
data_jul["Area"].sum()
```

Out[23]:

```
Year
2017    241508.0214
2018    130744.1789
2019    691406.5343
2020    237254.3704
Name: Area, dtype: float64
```

B7) The Company seeks a time series analysis report of the orders received.

In [24]:

```
data_year = data.groupby("Year")
data_year.head()
```

Out[24]:

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longi
0	2019	JUL	100 Mile House	170 Cedar Ave. S.	BC	CA	B0067295	1182.0200	SQ-M	Leased	51.645900	Ramasundar	121.29
1	2019	JUL	100 Mile House	300 Cariboo Hwy	BC	CA	N0092260	0.3600	HA	Owned	NaN	Ramasundar	
2	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076976	1467.8900	SQ-M	Owned	51.644508	Ramasundar	121.29
3	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0076984	23.4000	SQ-M	Owned	51.644222	Ramasundar	121.29
4	2019	JUL	100 Mile House	300 Cariboo Hwy.	BC	CA	B0081810	215.5000	SQ-M	Owned	51.644139	Ramasundar	121.29
54	2017	JUL	Bella Coola	636 Cliff St.	BC	CA	N0001484	0.1561	HA	Owned	NaN	Alford	
55	2017	JUL	Bob Quinn Lake	Hwy. 37	BC	CA	N2000483	0.0279	HA	Leased	NaN	Alford	
56	2017	JUL	Bob Quinn Lake	Stewart Cassiar Hwy. 37	BC	CA	B1002402	55.1800	SQ-M	Owned	56.977742	Alford	130.25
57	2017	JUL	Bob Quinn Lake	Stewart Cassiar Hwy. 37	BC	CA	B1002403	55.1800	SQ-M	Owned	56.981039	Alford	130.24
58	2017	JUL	Burnaby	3133 Sumner Ave.	BC	CA	B0067103	552.1300	SQ-M	Leased	49.255998	Alford	123.00
172	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002658	11.1500	SQ-M	Owned	49.105522	Lucida	121.36
173	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002659	11.6100	SQ-M	Owned	49.104260	Lucida	121.63
174	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002660	20.6200	SQ-M	Owned	49.103180	Lucida	121.63

	Year	Month	City	Address	Prov	Country	Identifier	Area	UoM	Tenure	Latitude	Agent	Longi
175	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002661	5.8800	SQ-M	Owned	49.104572	Lucida	121.63
176	2018	JUL	Chilliwack	57657 Chilliwack Lake Rd.	BC	WS	B1002662	30.0900	SQ-M	Owned	49.105490	Lucida	121.63
994	2020	JUL	Trail	1051 Farwell St.	BC	WS	B0068035	504.8000	SQ-M	Leased	49.097189	Lucida	117.70
995	2020	JUL	Trail	1520 Bay Ave.	BC	WS	B0067368	545.3000	SQ-M	Leased	49.094904	Lucida	117.70
996	2020	JUL	Trout Lake	Lardeau St.	BC	WS	B0086660	53.5000	SQ-M	Owned	50.646944	Lucida	117.54
997	2020	JUL	Trout Lake	Lardeau St.	BC	WS	B0086686	13.4000	SQ-M	Owned	50.646806	Lucida	117.54
998	2020	JUL	Trout Lake	Lardeau St.	BC	WS	B0086694	8.9000	SQ-M	Owned	50.646972	Lucida	117.54

In [25]:

```
list(data_year["Area"].sum())
```

Out[25]:

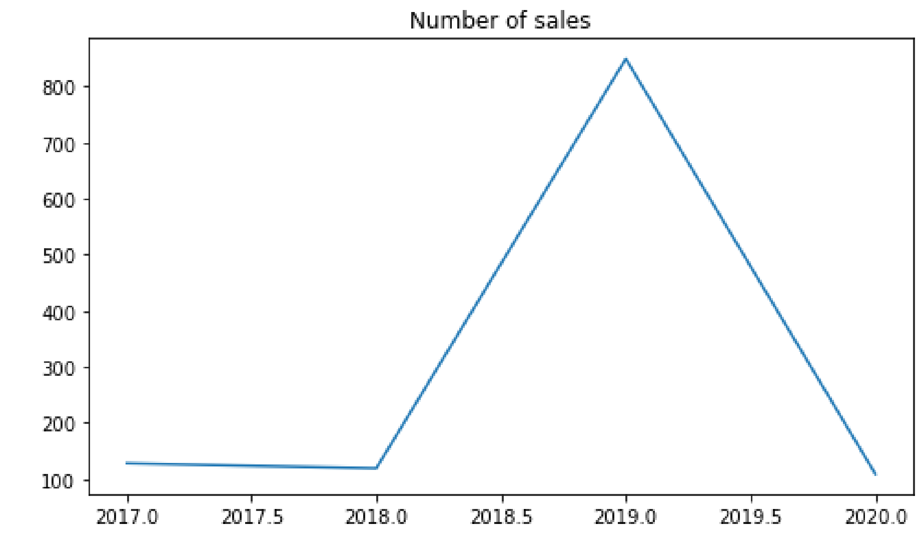
```
[241508.02140000006, 130744.17889999998, 691406.5343000002, 237254.37039999984]
```

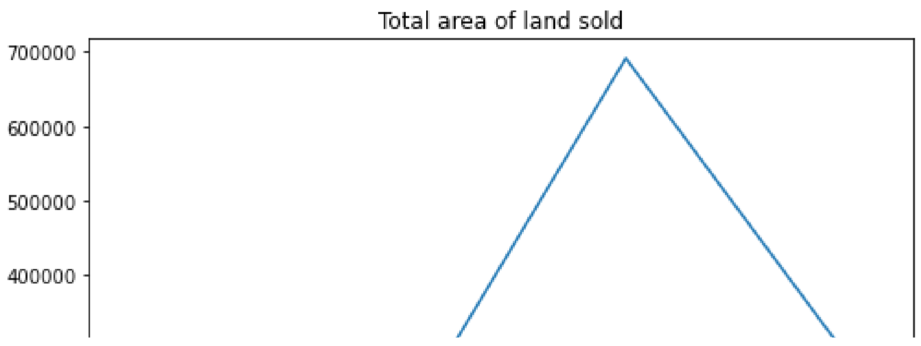
In [26]:

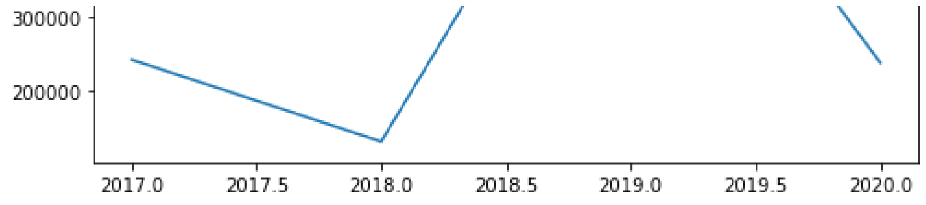
```
X = [2017,2018,2019,2020]
figure, axis = plt.subplots(nrows = 2, ncols=1,figsize =(8,10))
axis[0].plot(X, list(data_year["Identifier"].nunique()))
axis[0].set_title("Number of sales")
axis[1].plot(X, list(data_year["Area"].sum()))
axis[1].set_title("Total area of land sold")
```

Out[26]:

```
Text(0.5, 1.0, 'Total area of land sold')
```







We can see that the company had huge success in 2019, so the strategies used in the year were effective. The company can try to follow the tactics used in 2019 and try to emulate that performance.