Task4b Review

In this evaluation I will outline how the solution produced meets the system and user requirements as well as suggesting how the solution could be further developed.

Main Menu

The main menu provides the user with a list of options and will run the corresponding functions based on their input.

```
def mainmenu():
    print("\t\t****Welcome to the Dashboard****")
    print('1) Return all current data')
    print('2) Return data for a specific region')
    print('3) Return data for different property types within a specific
region')
    print('4) Return data for different property sizes within a specific
region')
    print('5) Compare overall increase in property value by region')
    print('To quit, enter any other value')
```

This menu displays the choices to the user in a meaningful way and is easy to use as the user only needs to enter a number for each option. The code can handle user errors such as entering a number out of range or not entering a number at all.

The main loop will keep running the menu until the user chooses to quit by entering any value that isn't one of the menu options.

Get_region function

This function will present the user with a list of all the regions present in the data and like the main menu, ask them to enter a number corresponding to one of the regions. If the user enters an incorrect value, the code will handle the error and ask them again for the input.

```
print("Regions:")
for i, region in enumerate(df["Region"].unique()):
    print(str(i+1) + ")", region)
```

The above code will print all the unique regions present in the data frame and put a number in the front to show which number the user needs to enter to choose that region

```
        #allows the user to enter the region as a code from 1 to 8
        region_code = input("Please enter the name of the region you
would like to check: ")
```

The functionality of this code is easy to use because it only requires the user to enter a single digit and because it displays the options in a clear and meaningful way.

Get_dates function

This function will ask the user for a start and end date, validate them and return the values. This code is easy to use as it clearly displays what the user needs to enter by giving them an example of what they need to enter.

```
        startdate = input("PLEASE ENTER A START DATE AS MONTH-YEAR e.g.
JAN-20: ")
```

The code will check the data frame to see if the user's input is one of the column names. If the input was not present in the data frame, the user will be prompted to enter the date again. The same process is repeated for getting the "enddate" and the values are returned once they are both valid.

Return data for a specific region

This section of the code is designed to meet the first of the system requirements which is to allow users to select a region and show the trends for increases in property over time.

```
    #asks the user for a region, startdate and enddate, makes sure they
are valid and calls the region_check function using them as the
parameters
    elif x == "2":
        region = get_region()
        startdate, enddate = get_dates()
        region_check(region, startdate, enddate)
```

This code will run when the user chooses the second option on the main menu and it will first call the "get_region" function and retrieve the "region" variable.

It will then get the "startdate" and "enddate" variables from calling the "get_dates" function.

Finally, the code will call the "region_check" function with these values as parameters.

Region_check function

This function will take the "region", "startdate" and "enddate" variables and produce a line graph showing average percentage increase of properties in that region between the two time periods.

```
    df1 = df.loc[:, startdate:enddate]
    df2 = df.loc[:, 'Region Code':'Rooms']

    result = pd.concat([df2, df1], axis=1,
join='inner').where(df2["Region"] == region)
    result = pd.DataFrame(result)
    result.dropna(inplace=True)
```

The above code creates an extract of the data frame which only has the data within the specified time period. And will then filter it to only show records relating to the selected region.

```
ave = df1.mean()
ave.plot()

    plt.title("Average percentage increase in " + region + " from " +
startdate + " to " + enddate)
    plt.show()
    return result
```

The above code will plot a graph showing the mean percentage increase from each month within the specified time period, give the graph a title that is meaningful to the user and display the graph.

The information being displayed in a graph makes use of appropriate graphical output in way that is relevant to the end user, the information is clear to understand and is displayed in a meaningful way.

Return data for different property types within a specific region

This section of the code is designed to meet part of the system requirement for the code to identify trends and patterns over time for specific property types of different sizes within a selected region. I

decided to break this requirement down into two functions, one for property types (this one) and another for property sizes.

```
    #asks the user for a region, startdate and enddate, makes sure they
are valid and calls the property_types function using them as the
parameters
    elif x == "3":
        region = get_region()
        startdate, enddate = get_dates()
        property_types(region, startdate, enddate)
```

This code will run when the user chooses the third option on the main menu and it will first call the "get_region" function and retrieve the "region" variable.

It will then get the "startdate" and "enddate" variables from calling the "get_dates" function.

Finally, the code will call the "property_types" function with these values as parameters.

Property_types function

This function will display a line graph showing the average percentage increase for each property type in a specified region over a specified time period.

```
        extract.mean().plot()
    plt.legend([property_type for property_type in df1["Property
Type"].unique()], title="Property Types:")
    #titles the graph appropriatley and includes the values inputted by
the user
    plt.title("Different property types in " + region + " from " +
startdate + " to " + enddate)
    #displays the graph
    plt.show()
```

This code plots and displays the graph

The graph will show a different line for each different property type, this displays the information to the user in a clear and meaningful way and allows them to easily compare the differences between the property types. The graph has a key showing which line is which making it easy to read and a title that tells the user in a meaningful way what the graph shows.

Return data for different property sizes within a specific region

This section of the code is designed to meet part of the system requirement for the code to identify trends and patterns over time for specific property types of different sizes within a selected region. I decided to break this requirement down into two functions, one for property types and another for property sizes (this one).

```
    #asks the user for a region, startdate and enddate, makes sure they
are valid and calls the property_sizes function using them as the
parameters
    elif x == "4":
        region = get_region()
        startdate, enddate = get_dates()
        property_sizes(region, startdate, enddate)
```

This code will run when the user chooses the third option on the main menu and it will first call the "get_region" function and retrieve the "region" variable.

It will then get the "startdate" and "enddate" variables from calling the "get_dates" function.

Finally, the code will call the "property_sizes" function with these values as parameters.

Property_sizes function

This function will display a line graph showing the average percentage increase for each property size in a specified region over a specified time period.

```
        extract.mean().plot()
    plt.legend([room_count for room_count in df1["Rooms"].unique()],
title="Rooms:")
    #titles the graph appropriatley and includes the values inputted by
the user

    plt.title("Different sized properties in " + region + " from " +
startdate + " to " + enddate)
    #displays the graph
    plt.show()
```

This code plots and displays the graph

The graph will show a different line for each different property size, this displays the information to the user in a clear and meaningful way and allows them to easily compare the differences between the property sizes. The graph has a key showing which line is which making it easy to read and a title that tells the user in a meaningful way what the graph shows.

Compare overall increase in property value by region

This section of code is designed to meet the system requirement to identify trends and patterns over time for the region with the highest overall increase in property value. I decided to display this information using a bar graph in order to show the user in a clear and meaningful way, the comparison between all the regions within the specified time period, with the region with the most overall percentage increase being obvious because of it having the biggest bar.

```
    #asks the user for startdate and enddate, makes sure they are valid
and calls the compare_overall_increase function using them as the
parameters
    elif x == "5":
        startdate, enddate = get_dates()
        compare_overall_increase(startdate, enddate)
```

First the code will get the “startdate” and “enddate” variables from calling the “get_dates” function.

Then it will call the “compare_overall_increase” function with these dates as parameters.

Compare_overall_increase function

This function takes a “startdate” and “endate” value and creates an extract of the data frame using only the dates within this time frame. It will then work out the overall average percentage increase for every region and display that information in a bar graph.

```
    for region in df["Region"].unique():
        #creates an extract of the dataframe only including rows where
 the region matches the "region" value and only includes the columns that
 are between the start and end date
        df1 = df.loc[(df["Region"] == region)].loc[:, startdate:enddate]
        #gets the sum of all percentages in the extract, round it to 2
 decimal places and adds it to the totals array
        total_increase = round(df1.sum("columns").sum(), 2)
        totals.append(total_increase)
```

This code creates the extract of the data frame only including correct dates and works out the total percentage increase for each region.

```
    #plots a bar graph with the region names on the x axis and the
corrosponding totals as the y axis
    #this will show in a visually meaningful way, the difference between
the overall increase in property value from different regions
    plt.bar([region for region in df["Region"].unique()], totals)
    #titles the graph appropriatley and includes the values inputted by
the user
    plt.title("Overall increase in property value from " + startdate + "
to " + enddate)
    #displays the graph
    plt.show()
```

The above code will plot and display the bar graph

What could be improved

Even though the code I wrote uses accepted conventions for variable and function names consistently, the pre-existing code contains some names that don't use accepted naming conventions such as "mainmenu()", "alldata()", "startdate" and "enddate". The code could be corrected to change these names to make the code slightly more readable/maintainable.

The code could also be improved by having a more sophisticated graphical user interface throughout the program including the main menu. This would provide the user with an even better experience and make the solution easier to use as well as display the information in a way that is clear to understand and more visually meaningful to the end user.

Conclusion

Overall, the solution I have developed meets all of the system requirements as well as carefully considering the experience of the end user. I have chosen to display the information using line and bar graphs in order to make the information easily understood and visually meaningful to the user. My solution is also easy to use by giving the user clear instructions of how to use the program and what inputs they need to enter.