Analysis of a Property Dataset

The assignment will focus on pre-processing data using the Pandas library, followed by the creation of plots using matplotlib and seaborn libraries.

KATE expects your code to define variables with specific names that correspond to certain things we are interested in.

KATE will run your notebook from top to bottom and check the latest value of those variables, so make sure you don't overwrite them.

- Remember to uncomment the line assigning the variable to your answer and don't change the variable or function names.
- Use copies of the original or previous DataFrames to make sure you do not overwrite them by mistake.

You will find instructions below about how to define each variable.

Once you're happy with your code, upload your notebook to KATE to check your feedback.

Importing Libraries

Run the following cell to import packages and set plotting styling.

The plotting styling should not be changed; doing so may result in KATE incorrectly evaluating your plots.

Note: `matplotlib` does a lot of work in the background to "guess" what figure to plot on. This can have the effect of modifying figures you have created before in the notebook, which will cause your plots to be wrong on KATE. To ensure your plots are always created properly, call `plt.figure()` before each command that creates a new plot, this will ensure you plot on a new figure everytime.

In [1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
sns.set()

from matplotlib.axes._axes import _log as matplotlib_axes_logger
matplotlib_axes_logger.setLevel('ERROR') #prevents unnecessary matplotlib warnings about
```

About the Dataset

Please refer to the `data/data-dictionary.pdf` file outlining details about the dataset each field (properties and their characteristics)

Importing the Dataset

Use `.read_csv()` to get our dataset `data/assessments.csv` and assign to DataFrame `df` :

In [2]:
```
df = pd.read_csv('data/assessments.csv')
```

Running `df.head()` and `df.info()` will show us how the DataFrame is structured:

In [3]:
```
df.head()
```

Out[3]:

	PROPERTYADDRESS	PROPERTYCITY	PROPERTYZIP	MUNIDESC	SCHOOLDESC	NEIGHDESC	TAXDES
0	GRANT ST	PITTSBURGH	15219	1st Ward - PITTSBURGH	City Of Pittsburgh	PITTSBURGH URBAN	10 Exemp
1	FORT DUQUESNE BLVD	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	PENNHOUSE & GATEWAY TOWER	20 Taxabl
2	FORT DUQUESNE BLVD	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	PENNHOUSE & GATEWAY TOWER	20 Taxabl
3	SMALLMAN ST	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	STRIP LOFT	20 Taxabl
4	SMALLMAN ST	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	STRIP LOFT	20 Taxabl

5 rows × 47 columns

In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1650 entries, 0 to 1649
Data columns (total 47 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   PROPERTYADDRESS     1650 non-null   object 
 1   PROPERTYCITY        1650 non-null   object 
 2   PROPERTYZIP         1650 non-null   int64  
 3   MUNIDESC            1650 non-null   object 
 4   SCHOOLDESC          1650 non-null   object 
 5   NEIGHDESC           1648 non-null   object 
 6   TAXDESC             1650 non-null   object 
 7   OWNERDESC           1650 non-null   object 
 8   CLASSDESC           1650 non-null   object 
 9   USEDESC             1650 non-null   object 
 10  LOTAREA             1650 non-null   int64  
 11  HOMESTEADFLAG       906 non-null    object 
 12  ABATEMENTFLAG       0 non-null      float64
 13  RECORDDATE          869 non-null    object 
 14  SALEDATE            1634 non-null   object 
 15  SALEPRICE           1625 non-null   float64
 16  SALEDESC            1628 non-null   object 
 17  PREVSALEDATE        969 non-null    object 
 18  PREVSALEPRICE       964 non-null    float64
 19  COUNTYBUILDING      1650 non-null   int64  
 20  COUNTYLAND          1650 non-null   int64  
 21  COUNTYTOTAL         1650 non-null   int64  
 22  COUNTYEXEMPTBLDG    1650 non-null   int64  
 23  LOCALBUILDING       1650 non-null   int64  
 24  LOCALLAND           1650 non-null   int64  
 25  LOCALTOTAL          1650 non-null   int64  
 26  FAIRMARKETBUILDING  1650 non-null   int64  
 27  FAIRMARKETLAND      1650 non-null   int64  
 28  FAIRMARKETTOTAL     1650 non-null   int64  
 29  STYLEDESC           1330 non-null   object 
 30  STORIES             1330 non-null   float64
 31  YEARBLT             1330 non-null   float64
 32  EXTFINISH_DESC      1329 non-null   object 
 33  ROOFDESC            1325 non-null   object 
 34  BASEMENTDESC        1329 non-null   object 
 35  GRADE               1330 non-null   object 
 36  GRADEDESC           1330 non-null   object 
 37  CONDITION           1329 non-null   float64
 38  CONDITIONDESC       1329 non-null   object 
 39  CDUDESC             1329 non-null   object 
 40  TOTALROOMS          1329 non-null   float64
 41  BEDROOMS            1330 non-null   float64
 42  FULLBATHS           1330 non-null   float64
 43  HALFBATHS           1321 non-null   float64
 44  HEATINGCOOLINGDESC  1329 non-null   object 
 45  FIREPLACES          1227 non-null   float64
 46  BSMTGARAGE          1263 non-null   float64
dtypes: float64(12), int64(12), object(23)
memory usage: 606.0+ KB
```

Charting Residential Properties with Pandas

Q1. Refer to the `df` DataFrame. Create a new DataFrame called `res` containing only entries from `df` with a `CLASSDESC` of `'RESIDENTIAL'`.

- Use the `.copy()` method to ensure you have a distinct DataFrame in memory
- Call the new dataframe `res`

See below code syntax for some guidance:

```
res['CLASSDESC']=='RESIDENTIAL'
```

In [5]:
```
#add your code below
res = df.copy()
mask = res['CLASSDESC'] == 'RESIDENTIAL'
res = res[mask]
res
```

Out[5]:

	PROPERTYADDRESS	PROPERTYCITY	PROPERTYZIP	MUNIDESC	SCHOOLDESC	NEIGHDESC	TAXD
1	FORT DUQUESNE BLVD	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	PENNHOUSE & GATEWAY TOWER	Tax
2	FORT DUQUESNE BLVD	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	PENNHOUSE & GATEWAY TOWER	Tax
3	SMALLMAN ST	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	STRIP LOFT	Tax
4	SMALLMAN ST	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	STRIP LOFT	Tax
5	SMALLMAN ST	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	STRIP LOFT	Tax
...	...	...	...	...	...	...	
1644	PHILLIPS LN	MC KEES ROCKS	15136	Robinson	Montour	93903	Tax
1645	REITER RD	PITTSBURGH	15235	Penn Hills	Penn Hills Twp	PENN HILLS TOWNSHIP	Tax
1646	SILVER PINES DR	GIBSONIA	15044	Pine	Pine-Richland	93501	Tax
1647	HIGHVIEW DR	PITTSBURGH	15241	Upper St. Clair	Upper St Clair	UPPER ST. CLAIR TOWNSHIP	Tax
1649	TRAILSIDE CT	CORAOPOLIS	15108	North Fayette	West Allegheny	92902	Tax

1504 rows × 47 columns

Q2. Create a new DataFrame called `res_16` containing only properties from `res` with `BEDROOMS` greater than 0 and less than 7.

- Use the `.copy()` method so that you have a distinct DataFrame in memory
- Call the new dataframe `res_16`
- Use the `.notnull()` method to filter out the rows in `BEDROOMS` which are null
- Use the `.astype()` method to change the data type of the `BEDROOMS` column to `int` : `.astype(int)`
- Filter the new DataFrame to only contain rows where `BEDROOMS` is greater than `0` and less than `7` : `(res_16['BEDROOMS'] > 0) & (res_16['BEDROOMS'] < 7)`

In [6]:
```
#add your code below
res_16 = res[res['BEDROOMS'].notnull()].copy()
res_16['BEDROOMS'] = res_16['BEDROOMS'].astype(int)
res_16 = res_16[(res_16['BEDROOMS'] > 0) & (res_16['BEDROOMS'] < 7)]
res_16
```

Out[6]:

	PROPERTYADDRESS	PROPERTYCITY	PROPERTYZIP	MUNIDESC	SCHOOLDESC	NEIGHDESC	TAXD
1	FORT DUQUESNE BLVD	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	PENNHOUSE & GATEWAY TOWER	Tax
2	FORT DUQUESNE BLVD	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	PENNHOUSE & GATEWAY TOWER	Tax
3	SMALLMAN ST	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	STRIP LOFT	Tax
4	SMALLMAN ST	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	STRIP LOFT	Tax
5	SMALLMAN ST	PITTSBURGH	15222	2nd Ward - PITTSBURGH	City Of Pittsburgh	STRIP LOFT	Tax
...	...	...	...	...	...	...	
1644	PHILLIPS LN	MC KEES ROCKS	15136	Robinson	Montour	93903	Tax
1645	REITER RD	PITTSBURGH	15235	Penn Hills	Penn Hills Twp	PENN HILLS TOWNSHIP	Tax
1646	SILVER PINES DR	GIBSONIA	15044	Pine	Pine-Richland	93501	Tax
1647	HIGHVIEW DR	PITTSBURGH	15241	Upper St. Clair	Upper St Clair	UPPER ST. CLAIR TOWNSHIP	Tax
1649	TRAILSIDE CT	CORAOPOLIS	15108	North Fayette	West Allegheny	92902	Tax

1304 rows × 47 columns

Q3. Use `.groupby()` on `res_16` DataFrame to create a Series with an index of `BEDROOMS` and values of the `.mean()` of `FULLBATHS` for each number of `BEDROOMS`. Assign this series to a new variable called `bed_bath`:

See below code syntax for some guidance:

```
bed_bath = DataFrame_Name.groupby(by=...)[column].mean()
```

Below snippet showcases how the resulting series should look like:

```
BEDROOMS
1    1.030303
2    1.173469
3    1.354132
4    2.236301
...
...
```

In [7]:
```
#add your code below
bed_bath = res_16.groupby(by='BEDROOMS')['FULLBATHS'].mean()
bed_bath
```

Out[7]:
```
BEDROOMS
1    1.030303
2    1.173469
3    1.354132
4    2.236301
5    2.925926
6    3.552632
Name: FULLBATHS, dtype: float64
```

Q4. Refer to the `bed_bath` variable from above question, also note `bed_bath` is a pandas `series` data object.

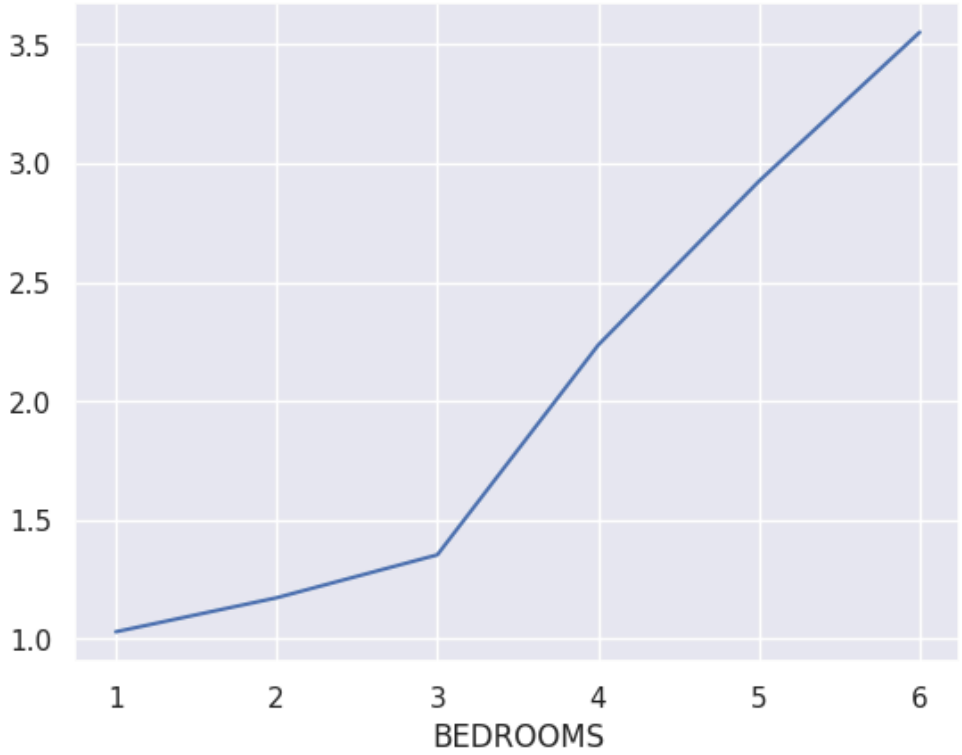
Use the `.plot()` method on `bed_bath` to create a line plot with `kind` parameter set to `line` .

- This should result in a line plot of `BEDROOMS` on the **x-axis** with the average number of `FULLBATHS` on the **y-axis**
- Save your plot into a new variable `bb_line`

See below code syntax for some guidance:

```
plt.figure()
bb_line = Series_Name.plot(kind='line')
```

In [8]:
```
#add your code below
#We create a new figure to make sure other figures in the notebook don't get modified
plt.figure()
bb_line = bed_bath.plot(kind='line')
```



Q5. Refer to the `res_16` DataFrame.

- Using the `res_16['BEDROOMS']` Series calculate `.value_counts()` for each value in the series

- Then use `.sort_index()` to order it by the index
- Save the results to a new variable called `beds`

See below code syntax for some guidance:

```
beds = Series_Name.value_counts().sort_index()
```

Below snippet showcases how the resulting series should look like:

```
1     33
2    294
3    593
4    292
...
...
```

In [9]:
```
#add your code below
beds = res_16['BEDROOMS'].value_counts().sort_index()
beds
```

Out[9]:
```
1     33
2    294
3    593
4    292
5     54
6     38
Name: BEDROOMS, dtype: int64
```

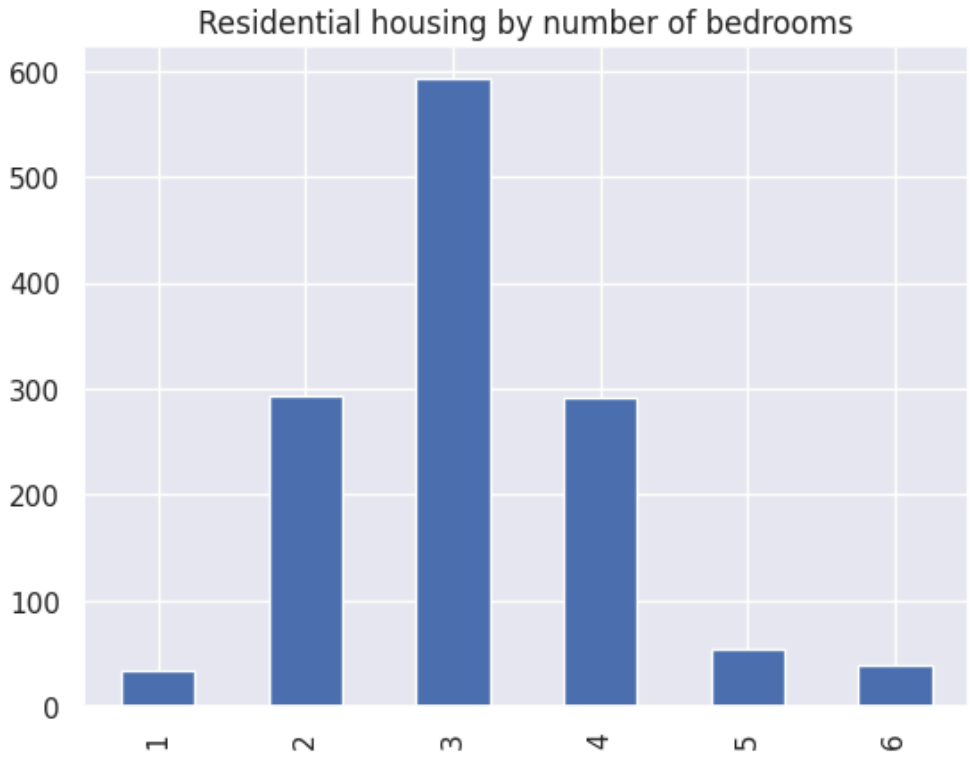
Q6. Refer to the `beds` variable from above question, also note `beds` is a pandas `series` data object.

- Use the `.plot()` method on `beds` to create a bar plot with `kind` parameter set to `bar` and `title` parameter set to `Residential housing by number of bedrooms`
- Save your line plot into a variable called `beds_bar`

See below code syntax for some guidance:

```
plt.figure()
beds_bar = Series_Name.plot(kind='bar', title=...)
```

In [10]:
```
#add your code below
#We create a new figure to make sure other figures in the notebook don't get modified
plt.figure()
beds_bar = beds.plot(kind='bar', title='Residential housing by number of bedrooms')
```



Q7. Create a function called `zip_land` which takes two arguments: a DataFrame (with the same columns as `df`) and an integer (which it can be assumed will always be present in the `PROPERTYZIP` column of the DataFrame).

This function will need to filter down the `df` argument to the rows where the `PROPERTYZIP` column is equal to the `zip_code` argument, before returning a `scatter` plot with the following properties:

- `x` ='LOTAREA'
- `y` ='FAIRMARKETLAND'
- `xlim` and `ylim` both from `0` to double the `.mean()` of the respective column values
- `alpha` =0.4
- `figsize` =(12,10)

```
def zip_land(df, zip_code):
    my_plot = [command to plot]
    return my_plot
```

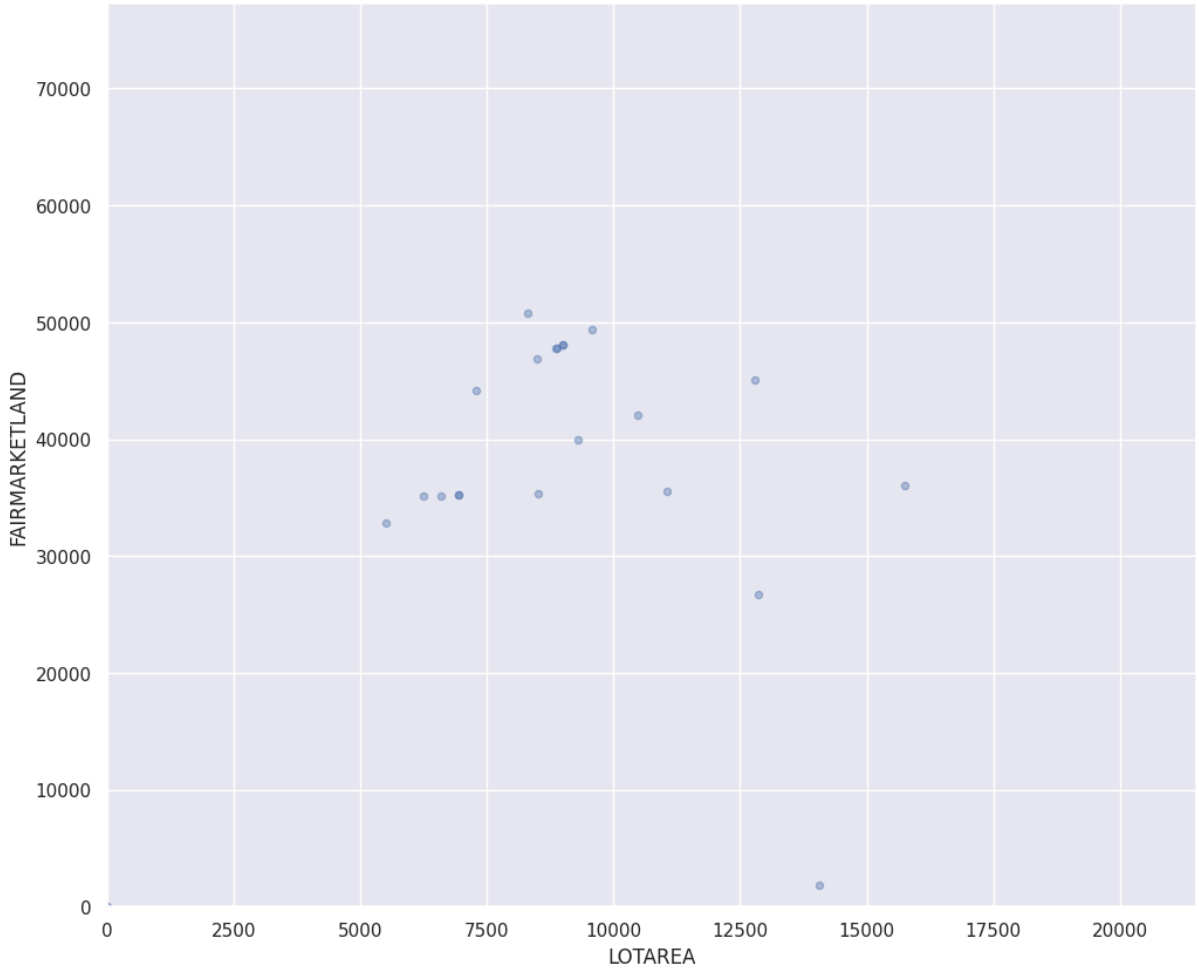
Please note you have been provided with the code for this question to carry out the necessary analysis. Simply uncomment the lines of code and run the code cell to produce the desired results.

In [11]:
```
#add your code below

def zip_land(df, zip_code):
    sub = df[(df['PROPERTYZIP'] == zip_code)]
    zip_chart = sub.plot(x='LOTAREA',
                         y='FAIRMARKETLAND',
                         kind='scatter',
                         xlim=(0, sub['LOTAREA'].mean() * 2),
                         ylim=(0, sub['FAIRMARKETLAND'].mean() * 2),
                         alpha=0.4,
                         figsize=(12, 10));

    return zip_chart
```

Run the following code cell to check that your function returns a chart as expected:

In [12]:
```
plt.figure()
zip_chart = zip_land(df, 15236)
```

```
<Figure size 640x480 with 0 Axes>
```



Charting Property Values with Seaborn

Q8. Refer to the `df` DataFrame. Create a new DataFrame called `sales` which contains only entries from `df` with a `SALEDESC` of `'VALID SALE'`.

- Use the `.copy()` method to ensure you have a distinct DataFrame in memory

- Call the new dataframe `sales`

See below code syntax for some guidance:

```
sales['SALEDESC']=='VALID SALE'
```

In [13]:
```
#add your code below
sales = df.copy()
mask_1 = sales['SALEDESC'] == 'VALID SALE'
sales = sales[mask_1]
sales
```

Out[13]:

	PROPERTYADDRESS	PROPERTYCITY	PROPERTYZIP	MUNIDESC	SCHOOLDESC	NEIGHDESC	TAXD
21	BIGELOW BLVD	PITTSBURGH	15213	4th Ward - PITTSBURGH	City Of Pittsburgh	10401	Ta
22	BIGELOW BLVD	PITTSBURGH	15213	4th Ward - PITTSBURGH	City Of Pittsburgh	10401	Ta
25	FORBES AVE	PITTSBURGH	15213	4th Ward - PITTSBURGH	City Of Pittsburgh	PITTSBURGH URBAN	Ta
28	BOUNDARY ST	PITTSBURGH	15213	4th Ward - PITTSBURGH	City Of Pittsburgh	10403	Ta
35	S CRAIG ST	PITTSBURGH	15213	4th Ward - PITTSBURGH	City Of Pittsburgh	PITTSBURGH URBAN	Ta
...	...	...	...	...	...	...	
1642	PARK AVE	PITTSBURGH	15221	Braddock Hills	Woodland Hills	BRADDOCK HILLS	Ta
1643	BIG ROCK RD	ALLISON PARK	15101	Hampton	Hampton Township	91402	Ta
1644	PHILLIPS LN	MC KEES ROCKS	15136	Robinson	Montour	93903	Ta
1645	REITER RD	PITTSBURGH	15235	Penn Hills	Penn Hills Twp	PENN HILLS TOWNSHIP	Ta
1647	HIGHVIEW DR	PITTSBURGH	15241	Upper St. Clair	Upper St Clair	UPPER ST. CLAIR TOWNSHIP	Ta

428 rows × 47 columns

Q9. Add a column to `sales` called `PITTSBURGH` , containing boolean values of `True` where `PROPERTYCITY` equals `PITTSBURGH` and `False` if not.

See below code syntax for some guidance:

```
sales[new_column_name] = sales['PROPERTYCITY'] == 'PITTSBURGH'
```

In [14]:
```
#add your code below
sales['PITTSBURGH'] = sales['PROPERTYCITY'] == 'PITTSBURGH'
sales
```

Out[14]:

	PROPERTYADDRESS	PROPERTYCITY	PROPERTYZIP	MUNIDESC	SCHOOLDESC	NEIGHDESC	TAXD
21	BIGELOW BLVD	PITTSBURGH	15213	4th Ward - PITTSBURGH	City Of Pittsburgh	10401	Ta:
22	BIGELOW BLVD	PITTSBURGH	15213	4th Ward - PITTSBURGH	City Of Pittsburgh	10401	Ta:
25	FORBES AVE	PITTSBURGH	15213	4th Ward - PITTSBURGH	City Of Pittsburgh	PITTSBURGH URBAN	Ta:
28	BOUNDARY ST	PITTSBURGH	15213	4th Ward - PITTSBURGH	City Of Pittsburgh	10403	Ta:
35	S CRAIG ST	PITTSBURGH	15213	4th Ward - PITTSBURGH	City Of Pittsburgh	PITTSBURGH URBAN	Ta:
...	...	...	...	...	...	...	
1642	PARK AVE	PITTSBURGH	15221	Braddock Hills	Woodland Hills	BRADDOCK HILLS	Ta:
1643	BIG ROCK RD	ALLISON PARK	15101	Hampton	Hampton Township	91402	Ta:
1644	PHILLIPS LN	MC KEES ROCKS	15136	Robinson	Montour	93903	Ta:
1645	REITER RD	PITTSBURGH	15235	Penn Hills	Penn Hills Twp	PENN HILLS TOWNSHIP	Ta:
1647	HIGHVIEW DR	PITTSBURGH	15241	Upper St. Clair	Upper St Clair	UPPER ST. CLAIR TOWNSHIP	Ta:

428 rows × 48 columns

Q10. Create a seaborn `.violinplot()` with the following properties:

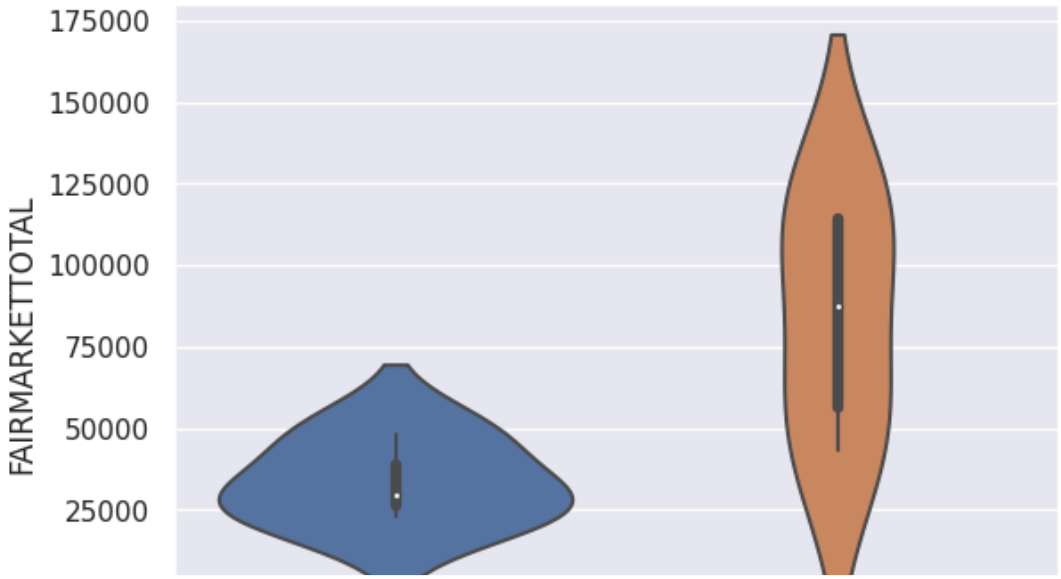
`x = 'PITTSBURGH'`
`y = 'FAIRMARKETTOTAL'`
`data` = only entries from `sales` where `sales['BEDROOMS'] == 1]`

Call the new variable `pitts_violin`

See below code syntax for some guidance:

```
plt.figure()
pitts_violin = sns.violinplot(x=..., y=..., data=...);
```

In [15]:
```
#add your code below
#We create a new figure to make sure other figures in the notebook don't get modified
plt.figure()
pitts_violin = sns.violinplot(x='PITTSBURGH', y='FAIRMARKETTOTAL', data = sales[sales['BE
plt.show()
```



Q11. Create a seaborn `.regplot()` with the following properties:

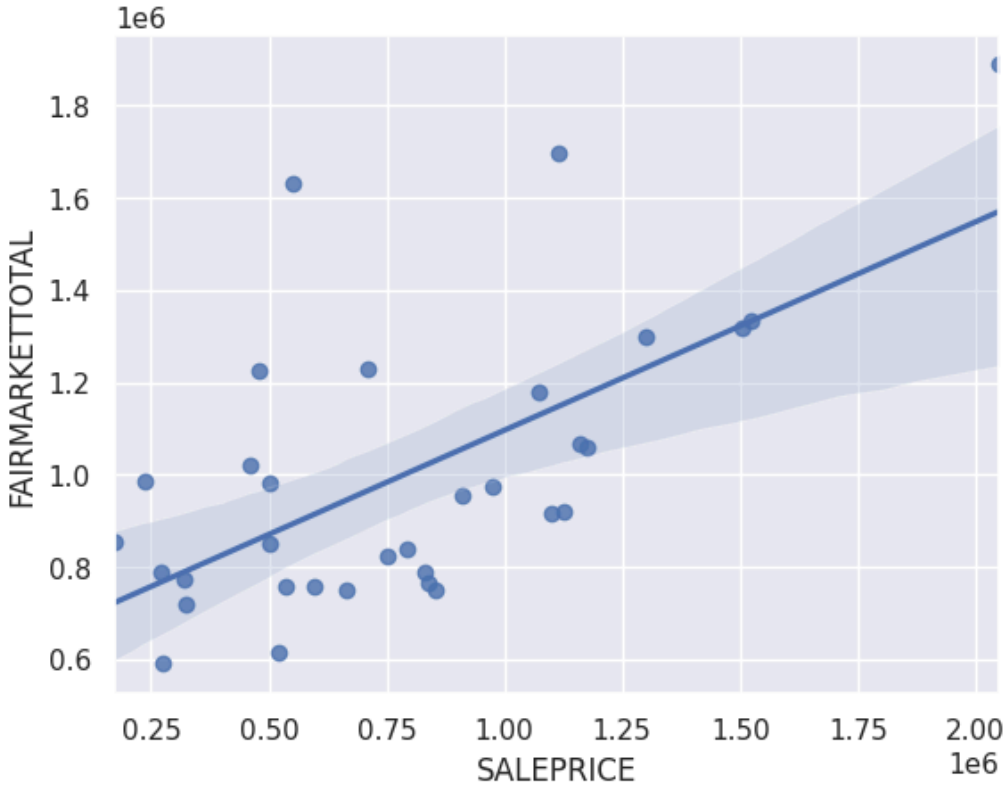
`x = 'SALEPRICE'`
`y = 'FAIRMARKETTOTAL'`
`data` = only entries from `sales` where `sales['GRADEDESC'] == 'EXCELLENT'`

Call the new variable `exc_reg`

See below code syntax for some guidance:

```
plt.figure()
exc_reg = sns.regplot(x=..., y=..., data=...)
```

In [16]:
```
#add your code below
#We create a new figure to make sure other figures in the notebook don't get modified
plt.figure()
exc_reg = sns.regplot(x='SALEPRICE', y='FAIRMARKETTOTAL', data=sales[sales['GRADEDESC'] =
```



Q12. Create a DataFrame called `bus` which contains only entries from `sales` where `CLASSDESC .isin(['COMMERCIAL', 'INDUSTRIAL', 'AGRICULTURAL'])` .

Please note you have been provided with the code for this question to carry out the necessary data manipulation work. Simply uncomment the lines of code and run the code cell to produce the desired results.

In [17]:
```
#add your code below
bus = sales[sales['CLASSDESC'].isin(['COMMERCIAL', 'INDUSTRIAL', 'AGRICULTURAL'])]
bus.head()
```

Out[17]:

	PROPERTYADDRESS	PROPERTYCITY	PROPERTYZIP	MUNIDESC	SCHOOLDESC	NEIGHDESC	TAXDE
25	FORBES AVE	PITTSBURGH	15213	4th Ward - PITTSBURGH	City Of Pittsburgh	PITTSBURGH URBAN	Taxa
35	S CRAIG ST	PITTSBURGH	15213	4th Ward - PITTSBURGH	City Of Pittsburgh	PITTSBURGH URBAN	Taxa
51	PENN AVE	PITTSBURGH	15201	6th Ward - PITTSBURGH	City Of Pittsburgh	PITTSBURGH URBAN	Taxa
162	HASTINGS ST	PITTSBURGH	15206	14th Ward - PITTSBURGH	City Of Pittsburgh	PITTSBURGH URBAN	Taxa
197	E CARSON ST	PITTSBURGH	15203	17th Ward - PITTSBURGH	City Of Pittsburgh	PITTSBURGH URBAN	Taxa

5 rows × 48 columns

Q13. Create a DataFrame using the `.groupby()` method on the `bus` DataFrame with the following properties:

- Data grouped by `['CLASSDESC', 'PITTSBURGH']` where the values are of the `.mean()` of `'FAIRMARKETTOTAL'`
- Use `.reset_index()` so that a DataFrame is created
- Use `.sort_values(by='FAIRMARKETTOTAL')` to order it

Call the new dataframe `bus_value`

Please note you have been provided with the code for this question to carry out the necessary data manipulation work. Simply uncomment the lines of code and run the code cell to produce the desired results.

In [18]:
```
#add your code below
bus_value = bus.groupby(['CLASSDESC', 'PITTSBURGH'])['FAIRMARKETTOTAL'].mean().reset_inde
bus_value
```

Out[18]:

	CLASSDESC	PITTSBURGH	FAIRMARKETTOTAL
0	COMMERCIAL	False	191475.0
2	INDUSTRIAL	False	362800.0
1	COMMERCIAL	True	674725.0

Q14. Create a seaborn `.barplot()` with the following properties:

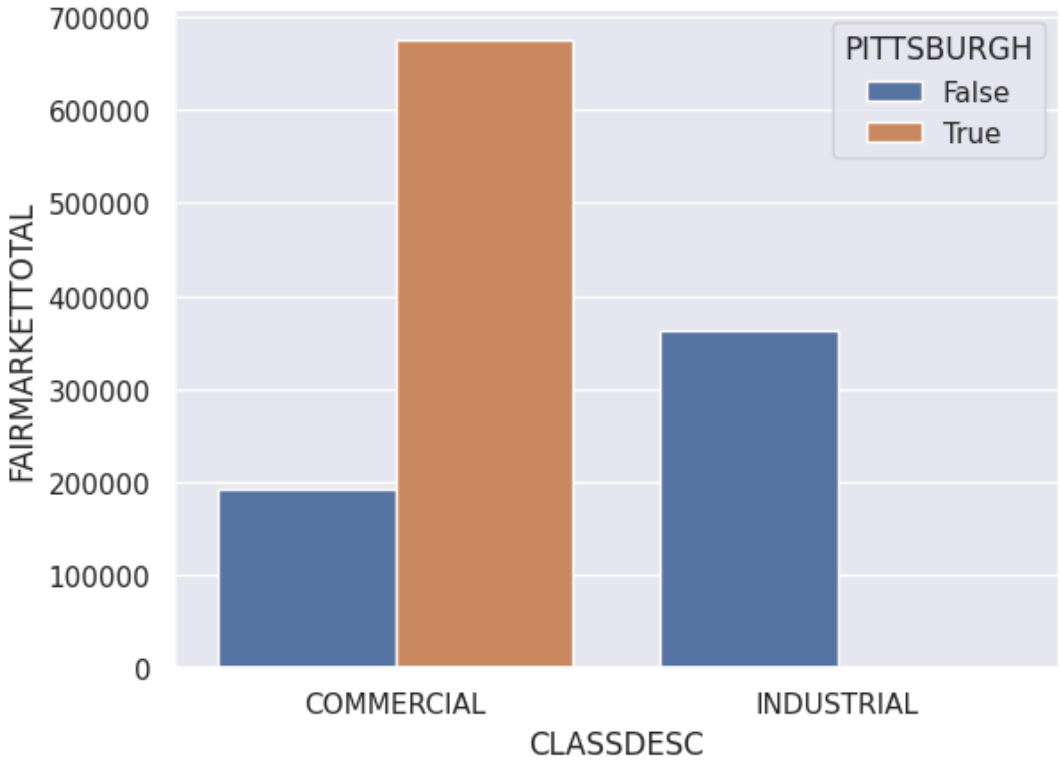
- `x = 'CLASSDESC'`
- `y = 'FAIRMARKETTOTAL'`
- `hue = 'PITTSBURGH'`
- `data = bus_value`

Call the new variable `bus_bar`

See below code syntax for some guidance:

```
plt.figure()
bus_bar = sns.barplot(x=..., y=..., hue=..., data=...);
```

In [19]:
```
#add your code below
#We create a new figure to make sure other figures in the notebook don't get modified
plt.figure()
bus_bar = sns.barplot(x='CLASSDESC', y='FAIRMARKETTOTAL', hue='PITTSBURGH', data=bus_valu
```



In [ ]: