Housing Rental Analysis for San Francisco

In this challenge, you will find data visualization skills, including aggregation, interactive visualizations, and geospatial analysis, to find properties in the San Francisco market that are viable investment opportunities.

Instructions

Use the `san_francisco_housing.ipynb` notebook to visualize and analyze the real-estate data.

Note that this assignment requires you to create a visualization by using hvPlot and GeoViews. Additionally, you need to read the `sfo_neighborhoods_census_data.csv` file from the `Resources` folder into the notebook and create the DataFrame that you'll use in the analysis.

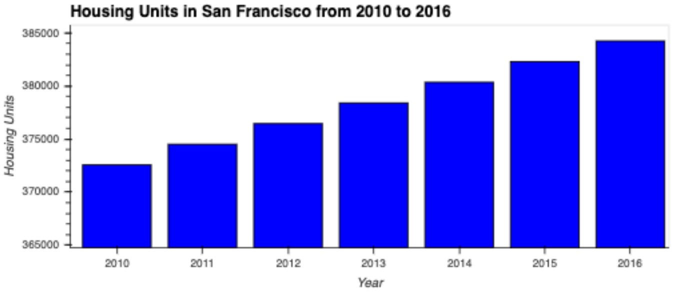
The main task in this Challenge is to visualize and analyze the real-estate data in your Jupyter notebook. Use the `san_francisco_housing.ipynb` notebook to complete the following tasks:

- Calculate and plot the housing units per year.
- Calculate and plot the average prices per square foot.
- Compare the average prices by neighborhood.
- Build an interactive neighborhood map.
- Compose your data story.

Calculate and Plot the Housing Units per Year

For this part of the assignment, use numerical and visual aggregation to calculate the number of housing units per year, and then visualize the results as a bar chart. To do so, complete the following steps:

1. Use the `groupby` function to group the data by year. Aggregate the results by the `mean` of the groups.
2. Use the `hvplot` function to plot the `housing_units_by_year` DataFrame as a bar chart. Make the x-axis represent the `year` and the y-axis represent the `housing_units` .
3. Style and format the line plot to ensure a professionally styled visualization.
4. Note that your resulting plot should appear similar to the following image:



5. Answer the following question:

   - What's the overall trend in housing units over the period that you're analyzing?

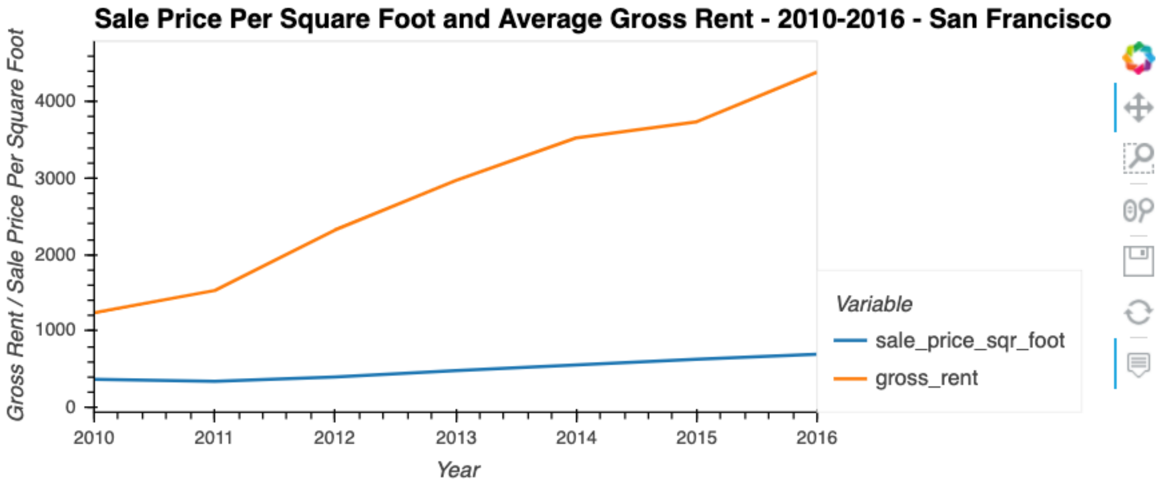
Calculate and Plot the Average Sale Prices per Square Foot

For this part of the assignment, use numerical and visual aggregation to calculate the average prices per square foot, and then visualize the results as a bar chart. To do so, complete the following steps:

1. Group the data by year, and then average the results. What's the lowest gross rent that's reported for the years that the DataFrame includes?
2. Create a new DataFrame named `prices_square_foot_by_year` by filtering out the "housing_units" column. The new DataFrame should include the averages per year for only the sale price per square foot and the gross rent.
3. Use hvPlot to plot the `prices_square_foot_by_year` DataFrame as a line plot.

> **Hint** This single plot will include lines for both `sale_price_sqr_foot` and `gross_rent` .

4. Style and format the line plot to ensure a professionally styled visualization.
5. Note that your resulting plot should appear similar to the following image:

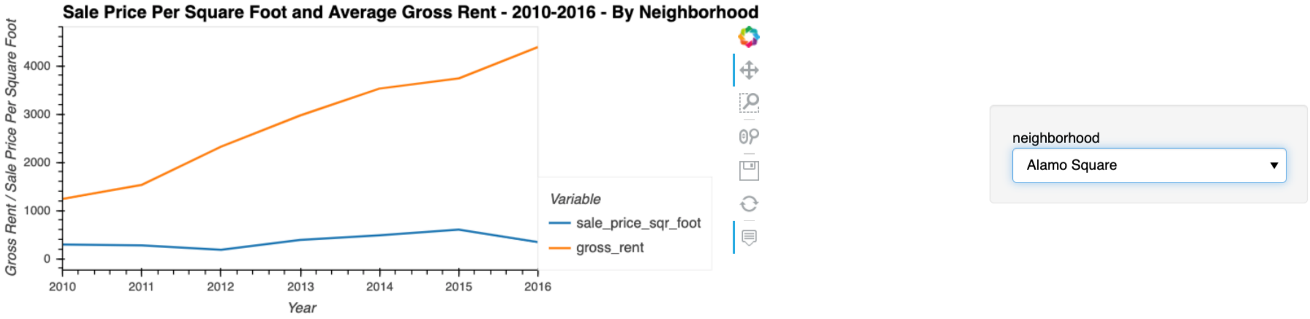


6. Use both the `prices_square_foot_by_year` DataFrame and interactive plots to answer the following questions:
   - Did any year experience a drop in the average sale price per square foot compared to the previous year?
   - If so, did the gross rent increase or decrease during that year?

Compare the Average Sale Prices by Neighborhood

For this part of the assignment, use interactive visualizations and widgets to explore the average sale price per square foot by neighborhood. To do so, complete the following steps:

1. Create a new DataFrame that groups the original DataFrame by year and neighborhood. Aggregate the results by the `mean` of the groups.
2. Filter out the "housing_units" column to create a DataFrame that includes only the `sale_price_sqr_foot` and `gross_rent` averages per year.
3. Create an interactive line plot with hvPlot that visualizes both `sale_price_sqr_foot` and `gross_rent`. Set the x-axis parameter to the year (`x="year"`). Use the `groupby` parameter to create an interactive widget for `neighborhood`.
4. Style and format the line plot to ensure a professionally styled visualization.
5. Note that your resulting plot should appear similar to the following image:


Sale Price Per Square Foot and Average Gross Rent - 2010-2016 - By Neighborhood
Gross Rent / Sale Price Per Square Foot
0
1000
2000
3000
4000
2010
2011
2012
2013
2014
2015
2016
Year
Variable
sale_price_sqr_foot
gross_rent
neighborhood
Alamo Square

6. Use the interactive visualization to answer the following question:
   - For the Anza Vista neighborhood, is the average sale price per square foot for 2016 more or less than the price that's listed for 2012?

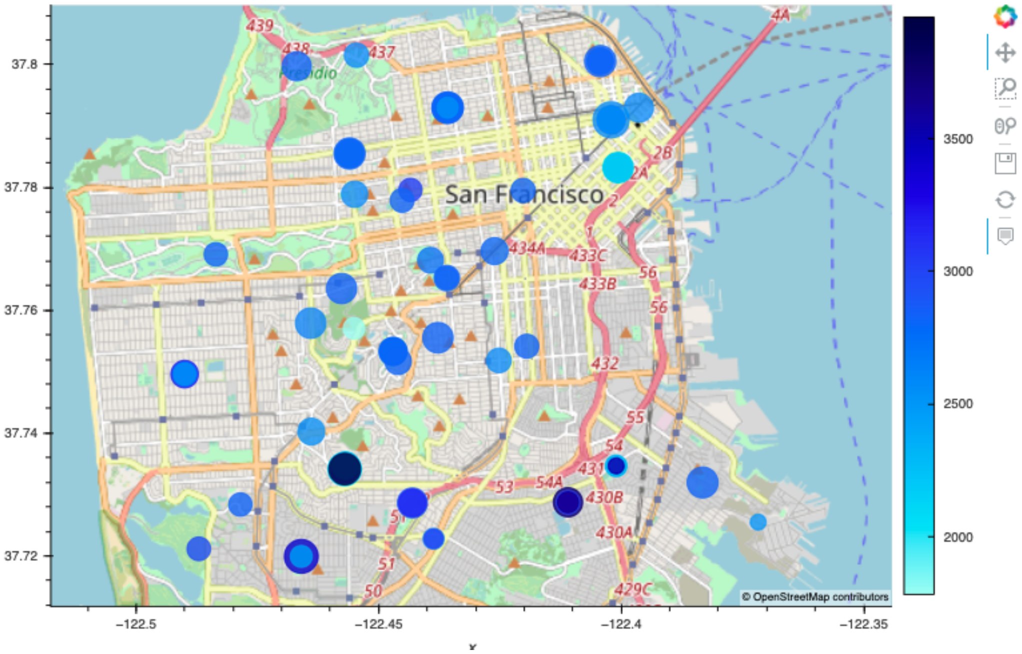
Build an Interactive Neighborhood Map

For this part of the assignment, explore the geospatial relationships in the data by using interactive visualizations with hvPlot and GeoViews. To build your map, use the `sfo_data_df` DataFrame (created during the initial import), which includes the neighborhood location data with the average prices. To do all this, complete the following steps:

1. Read the `neighborhood_coordinates.csv` file from the `Resources` folder into the notebook, and create a DataFrame named `neighborhood_locations_df` . Be sure to set the `index_col` of the DataFrame as "Neighborhood".
2. Using the original `sfo_data_df` Dataframe, create a DataFrame named `all_neighborhood_info_df` that groups the data by neighborhood. Aggregate the results by the `mean` of the group.
3. Review the two code cells that concatenate the `neighborhood_locations_df` DataFrame with the `all_neighborhood_info_df` DataFrame. Note that the first cell uses the [Pandas concat function (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html) to create a DataFrame named `all_neighborhoods_df` . The second cell cleans the data and sets the "Neighborhood" column. Be sure to run these cells to create the `all_neighborhoods_df` DataFrame, which you'll need to create the geospatial visualization.
4. Using hvPlot with GeoViews enabled, create a `points` plot for the `all_neighborhoods_df` DataFrame. Be sure to do the following:
   - Set the `geo` parameter to True.
   - Set the `size` parameter to "sale_price_sqr_foot".
   - Set the `color` parameter to "gross_rent".
   - Set the `frame_width` parameter to 700.

- Set the `frame_height` parameter to 500.
- Include a descriptive title.

Note that your resulting plot should appear similar to the following image:



5. Use the interactive map to answer the following question:

   - Which neighborhood has the highest gross rent, and which has the highest sale price per square foot?

Compose Your Data Story

Based on the visualizations that you created, answer the following questions:

- How does the trend in rental income growth compare to the trend in sales prices? Does this same trend hold true for all the neighborhoods across San Francisco?
- What insights can you share with your company about the potential one-click, buy-and-rent strategy that they're pursuing? Do neighborhoods exist that you would suggest for investment, and why?

In [1]:
```
# Import the required libraries and dependencies
import pandas as pd
import hvplot.pandas
from pathlib import Path
import matplotlib.pyplot as plt
```

Note: selenium and conda-forge firefox geckodriver were downloaded for experimenting with regards to downloading plot images using Matplotlib.

In [2]:
```
pip install selenium
```

```
Requirement already satisfied: selenium in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (4.7.2)
Requirement already satisfied: trio-websocket~=0.9 in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (fr
om selenium) (0.9.2)
Requirement already satisfied: urllib3[socks]~=1.26 in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (f
rom selenium) (1.26.9)
Requirement already satisfied: trio~=0.17 in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (from seleni
um) (0.22.0)
Requirement already satisfied: certifi>=2021.10.8 in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (fro
m selenium) (2022.12.7)
Requirement already satisfied: async-generator>=1.9 in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (f
rom trio~=0.17->selenium) (1.10)
Requirement already satisfied: sortedcontainers in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (from
trio~=0.17->selenium) (2.4.0)
Requirement already satisfied: sniffio in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (from trio~=0.1
7->selenium) (1.2.0)
Requirement already satisfied: exceptiongroup>=1.0.0rc9 in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-package
s (from trio~=0.17->selenium) (1.1.0)
Requirement already satisfied: outcome in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (from trio~=0.1
7->selenium) (1.2.0)
Requirement already satisfied: attrs>=19.2.0 in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (from tri
o~=0.17->selenium) (21.4.0)
Requirement already satisfied: idna in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (from trio~=0.17->
selenium) (3.3)
Requirement already satisfied: wsproto>=0.14 in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (from tri
o-websocket~=0.9->selenium) (1.2.0)
Requirement already satisfied: PySocks!=1.5.7,<2.0,>=1.5.6 in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-pack
ages (from urllib3[socks]~=1.26->selenium) (1.7.1)
Requirement already satisfied: h11<1,>=0.9.0 in /Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages (from wsp
roto>=0.14->trio-websocket~=0.9->selenium) (0.14.0)
Note: you may need to restart the kernel to use updated packages.
```

In [3]:
```
conda install -c conda-forge firefox geckodriver
```

```
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.


Note: you may need to restart the kernel to use updated packages.
```

Import the data

In [2]:
```
ng the read_csv function and Path module, create a DataFrame
importing the sfo_neighborhoods_census_data.csv file from the Resources folder
ata_df = pd.read_csv('/Users/mohjaiswal/Desktop/Unit-6-Homework-Asn/Starter_Code/Resources/sfo_neighborhoods_census_data
iew the first and last five rows of the DataFrame
ay(sfo_data_df.head(),sfo_data_df.tail())
```

	year	neighborhood	sale_price_sqr_foot	housing_units	gross_rent
0	2010	Alamo Square	291.182945	372560	1239
1	2010	Anza Vista	267.932583	372560	1239
2	2010	Bayview	170.098665	372560	1239
3	2010	Buena Vista Park	347.394919	372560	1239
4	2010	Central Richmond	319.027623	372560	1239

	year	neighborhood	sale_price_sqr_foot	housing_units	gross_rent
392	2016	Telegraph Hill	903.049771	384242	4390
393	2016	Twin Peaks	970.085470	384242	4390
394	2016	Van Ness/ Civic Center	552.602567	384242	4390
395	2016	Visitacion Valley	328.319007	384242	4390
396	2016	Westwood Park	631.195426	384242	4390

In [3]:
```
#Optional Case (OMIT) - I tried creating a multi-axis plot instead the plot in section 2, step 3 (Hvplot of gross rent

#The following is the code for a dataframe for the same.
df_optional = (sfo_data_df.groupby('year').mean())
df_optional["YEAR"] = [2010,2011,2012,2013,2014,2015,2016]
df_optional
```

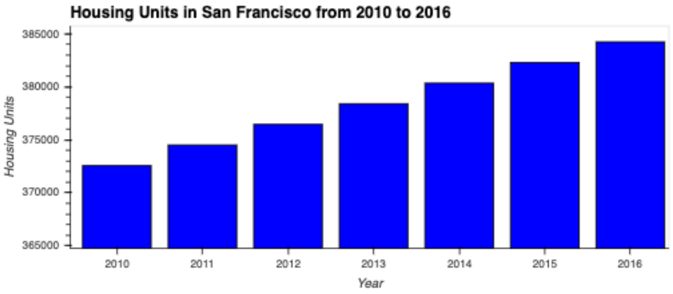
Out[3]:

	sale_price_sqr_foot	housing_units	gross_rent	YEAR
year				
2010	369.344353	372560.0	1239.0	2010
2011	341.903429	374507.0	1530.0	2011
2012	399.389968	376454.0	2324.0	2012
2013	483.600304	378401.0	2971.0	2013
2014	556.277273	380348.0	3528.0	2014
2015	632.540352	382295.0	3739.0	2015
2016	697.643709	384242.0	4390.0	2016

Calculate and Plot the Housing Units per Year

For this part of the assignment, use numerical and visual aggregation to calculate the number of housing units per year, and then visualize the results as a bar chart. To do so, complete the following steps:

1. Use the `groupby` function to group the data by year. Aggregate the results by the `mean` of the groups.
2. Use the `hvplot` function to plot the `housing_units_by_year` DataFrame as a bar chart. Make the x-axis represent the `year` and the y-axis represent the `housing_units` .
3. Style and format the line plot to ensure a professionally styled visualization.
4. Note that your resulting plot should appear similar to the following image:



5. Answer the following question:

- What's the overall trend in housing units over the period that you're analyzing?

Step 1: Use the `groupby` function to group the data by year. Aggregate the results by the `mean` of the groups.

In [4]:

```
# Create a numerical aggregation that groups the data by the year and then averages the results.
housing_units_by_year = (sfo_data_df.groupby('year').mean().sort_values('housing_units'))
# Review the DataFrame
housing_units_by_year = housing_units_by_year.drop(columns = ["sale_price_sqr_foot","gross_rent"]).copy()
# Describe DataFrame
housing_units_by_year.describe()
```

Out[4]:

	housing_units
count	7.000000
mean	378401.000000
std	4206.000713
min	372560.000000
25%	375480.500000
50%	378401.000000
75%	381321.500000
max	384242.000000

Step 2: Use the `hvplot` function to plot the `housing_units_by_year` DataFrame as a bar chart. Make the x-axis represent the `year` and the y-axis represent the `housing_units`.

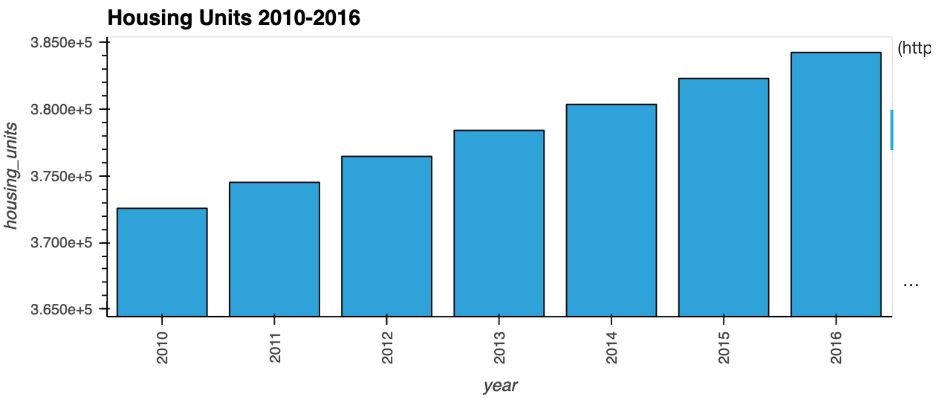
Step 3: Style and format the line plot to ensure a professionally styled visualization.

Step 4: Compare with read.md output to ensure coherency.

In [5]:

```
# Create a visual aggregation explore the housing units by year
barplot = housing_units_by_year.hvplot.bar(rot=90, label="Housing Units 2010-2016")
barplot
```

Out[5]:



In [6]:

```
#Save this box plot
hvplot.save(barplot, 'barplot.png')
```

In [7]:

```
CAGR = (384242.000000/372560.000000)**(1/7.0)-1
CAGR
CAGR_pct = CAGR*100
CAGR_pct
```

Out[7]:

```
0.44203788571399727
```

Step 5: Answer the following question:

Question: What is the overall trend in housing_units over the period being analyzed?

In [8]:

```
print ('The number of Housing Units in San Francisco showed an upward trend; growing at a compounded annual growth rate
```

```
The number of Housing Units in San Francisco showed an upward trend; growing at a compounded annual growth rate of 0.
44 %.
```

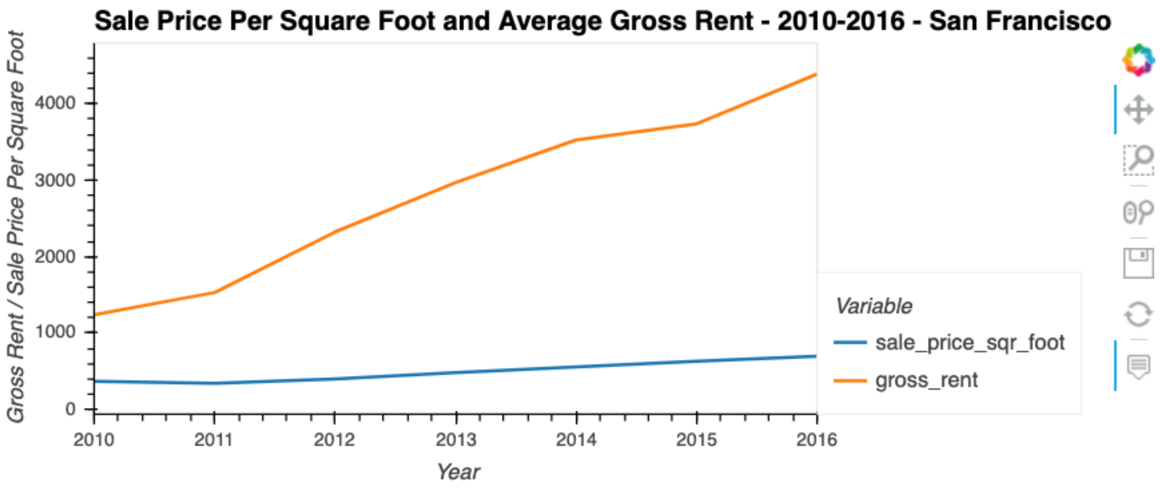
Calculate and Plot the Average Sale Prices per Square Foot

For this part of the assignment, use numerical and visual aggregation to calculate the average prices per square foot, and then visualize the results as a bar chart. To do so, complete the following steps:

1. Group the data by year, and then average the results. What's the lowest gross rent that's reported for the years that the DataFrame includes?
2. Create a new DataFrame named `prices_square_foot_by_year` by filtering out the "housing_units" column. The new DataFrame should include the averages per year for only the sale price per square foot and the gross rent.
3. Use hvPlot to plot the `prices_square_foot_by_year` DataFrame as a line plot.

> **Hint** This single plot will include lines for both `sale_price_sqr_foot` and `gross_rent` .

4. Style and format the line plot to ensure a professionally styled visualization.
5. Note that your resulting plot should appear similar to the following image:



6. Use both the `prices_square_foot_by_year` DataFrame and interactive plots to answer the following questions:

- Did any year experience a drop in the average sale price per square foot compared to the previous year?
- If so, did the gross rent increase or decrease during that year?

Step 1: Group the data by year, and then average the results.

In [9]:
```
# Create a numerical aggregation by grouping the data by year and averaging the results
prices_square_foot_by_year = (sfo_data_df.groupby('year').mean())

# Review the resulting DataFrame
display(prices_square_foot_by_year.head(2),prices_square_foot_by_year.tail(2))
```

	sale_price_sqr_foot	housing_units	gross_rent
year			
2010	369.344353	372560.0	1239.0
2011	341.903429	374507.0	1530.0

	sale_price_sqr_foot	housing_units	gross_rent
year			
2015	632.540352	382295.0	3739.0
2016	697.643709	384242.0	4390.0

In [10]:
```
min_rent = prices_square_foot_by_year['gross_rent'].min()
min_rent
```

Out[10]:
```
1239.0
```

Question: What is the lowest gross rent reported for the years included in the DataFrame?

In [11]:
```
print ('The lowest gross rent reported for the years included in the DataFrame is $', round(min_rent))
```

```
The lowest gross rent reported for the years included in the DataFrame is $ 1239
```

Step 2: Create a new DataFrame named `prices_square_foot_by_year` by filtering out the “housing_units” column. The new DataFrame should include the averages per year for only the sale price per square foot and the gross rent.

In [12]:
```
# Filter out the housing_units column, creating a new DataFrame
# Keep only sale_price_sqr_foot and gross_rent averages per year
prices_square_foot_by_year = prices_square_foot_by_year.drop(columns = ["housing_units"]).copy()

# Review the DataFrame
prices_square_foot_by_year
```

Out[12]:

	sale_price_sqr_foot	gross_rent
year		
2010	369.344353	1239.0
2011	341.903429	1530.0
2012	399.389968	2324.0
2013	483.600304	2971.0
2014	556.277273	3528.0
2015	632.540352	3739.0
2016	697.643709	4390.0

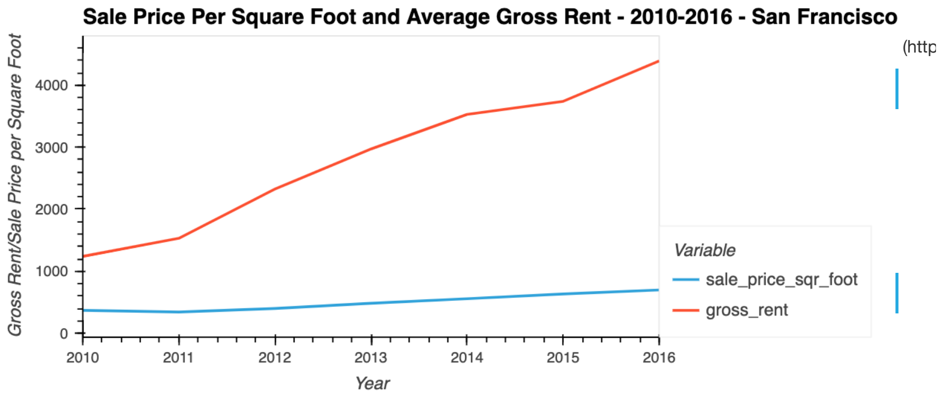
Step 3: Use hvPlot to plot the `prices_square_foot_by_year` DataFrame as a line plot.

> **Hint** This single plot will include lines for both `sale_price_sqr_foot` and `gross_rent`

Step 4: Style and format the line plot to ensure a professionally styled visualization.

In [13]:
```
# Plot prices_square_foot_by_year.
# Inclued labels for the x- and y-axes, and a title.
lineplot = prices_square_foot_by_year.hvplot.line(
    xlabel ='Year',
    ylabel = 'Gross Rent/Sale Price per Square Foot',
    label = 'Sale Price Per Square Foot and Average Gross Rent - 2010-2016 - San Francisco'
).opts(yformatter ='%.0f')
lineplot
```

Out[13]:

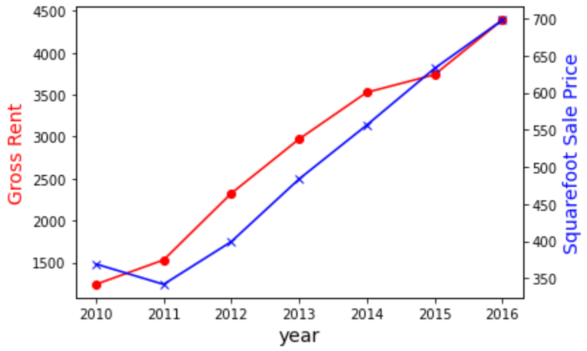


In [14]:
```
hvplot.save(lineplot, 'Lineplot.png')
```

PS Note: To better show the change in sale_price we should use a multi-axis line plot with two different Y axis. We can do this with the MatPlotLib twinx() function to create the second axis object.

In [15]:

```
#OMIT from marking (Optional)
# create figure and axis objects with subplots()
fig,ax = plt.subplots()
# make a plot
ax.plot(df_optional.YEAR,
        df_optional.gross_rent,
        color="red",
        marker="o")
# set x-axis label
ax.set_xlabel("year", fontsize = 14)
# set y-axis label
ax.set_ylabel("Gross Rent",
              color="red",
              fontsize=14)
# twin object for two different y-axis on the sample plot
ax2=ax.twinx()
# make a plot with different y-axis using second axis object
ax2.plot(df_optional.YEAR,
         df_optional.sale_price_sqr_foot,
         color="blue",
         marker="x")
# set y-axis label the second time
ax2.set_ylabel("Squarefoot Sale Price",color="blue",fontsize=14)
plt.show()
# save the plot as a file
fig.savefig('two_different_y_axis_for_single_python_plot_with_twinx.jpg',
            format='jpeg',
            dpi=100,
            bbox_inches='tight')
plt.show()
```



In [16]:

```
prices_square_foot_by_year['sale_price_sqr_foot'].pct_change()
print("Square Foot Price:",

      prices_square_foot_by_year['sale_price_sqr_foot'].pct_change()
     )

print(
      "Gross Rental Prices:",

      prices_square_foot_by_year['gross_rent'].pct_change(),
     )
```

```
Square Foot Price: year
2010          NaN
2011   -0.074296
2012    0.168137
2013    0.210847
2014    0.150283
2015    0.137095
2016    0.102924
Name: sale_price_sqr_foot, dtype: float64
Gross Rental Prices: year
2010          NaN
2011    0.234867
2012    0.518954
2013    0.278399
2014    0.187479
2015    0.059807
2016    0.174111
Name: gross_rent, dtype: float64
```

Step 6: Use both the `prices_square_foot_by_year` DataFrame and interactive plots to answer the following questions:

Question: Did any year experience a drop in the average sale price per square foot compared to the previous year?

```
In [17]: print('There is a drop of 7.43% in avg sale price in 2011 compared to 2010.')
```

```
There is a drop of 7.43% in avg sale price in 2011 compared to 2010.
```

Question: If so, did the gross rent increase or decrease during that year?

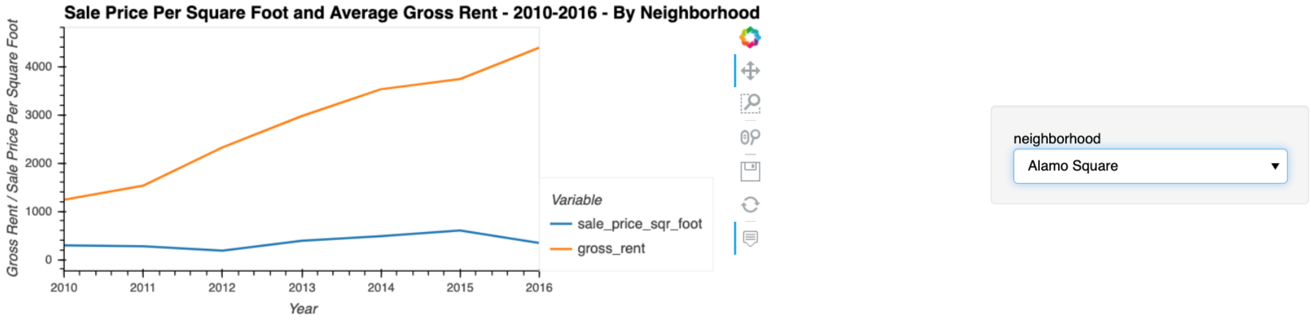
```
In [18]: print('There is an increase of 23.49% in avg sale price in 2011 compared to 2010.')
```

```
There is an increase of 23.49% in avg sale price in 2011 compared to 2010.
```

Compare the Average Sale Prices by Neighborhood

For this part of the assignment, use interactive visualizations and widgets to explore the average sale price per square foot by neighborhood. To do so, complete the following steps:

1. Create a new DataFrame that groups the original DataFrame by year and neighborhood. Aggregate the results by the `mean` of the groups.
2. Filter out the “housing_units” column to create a DataFrame that includes only the `sale_price_sqr_foot` and `gross_rent` averages per year.
3. Create an interactive line plot with hvPlot that visualizes both `sale_price_sqr_foot` and `gross_rent` . Set the x-axis parameter to the year (`x="year"`). Use the `groupby` parameter to create an interactive widget for `neighborhood` .
4. Style and format the line plot to ensure a professionally styled visualization.
5. Note that your resulting plot should appear similar to the following image:


Sale Price Per Square Foot and Average Gross Rent - 2010-2016 - By Neighborhood
Gross Rent / Sale Price Per Square Foot
4000
3000
2000
1000
0
2010
2011
2012
2013
2014
2015
2016
Year
Variable
sale_price_sqr_foot
gross_rent

neighborhood

Alamo Square ▾

6. Use the interactive visualization to answer the following question:

   - For the Anza Vista neighborhood, is the average sale price per square foot for 2016 more or less than the price that’s listed for 2012?

Step 1: Create a new DataFrame that groups the original DataFrame by year and neighborhood. Aggregate the results by the `mean` of the groups.

In [19]:

```
# Group by year and neighborhood and then create a new dataframe of the mean values
prices_by_year_by_neighborhood = (sfo_data_df.groupby(['year','neighborhood']).mean())
# Review the DataFrame
prices_by_year_by_neighborhood
```

Out[19]:

		sale_price_sqr_foot	housing_units	gross_rent
year	**neighborhood**			
2010	**Alamo Square**	291.182945	372560.0	1239.0
	Anza Vista	267.932583	372560.0	1239.0
	Bayview	170.098665	372560.0	1239.0
	Buena Vista Park	347.394919	372560.0	1239.0
	Central Richmond	319.027623	372560.0	1239.0
...	**...**	...	...	...
2016	**Telegraph Hill**	903.049771	384242.0	4390.0
	Twin Peaks	970.085470	384242.0	4390.0
	Van Ness/ Civic Center	552.602567	384242.0	4390.0
	Visitacion Valley	328.319007	384242.0	4390.0
	Westwood Park	631.195426	384242.0	4390.0

397 rows × 3 columns

Step 2: Filter out the “housing_units” column to create a DataFrame that includes only the `sale_price_sqr_foot` and `gross_rent` averages per year.

In [20]:

```
# Filter out the housing_units
prices_by_year_by_neighborhood =  prices_by_year_by_neighborhood.drop(columns = ["housing_units"]).copy()

# Review the first and last five rows of the DataFrame
prices_by_year_by_neighborhood
```

Out[20]:

		sale_price_sqr_foot	gross_rent
year	**neighborhood**		
2010	**Alamo Square**	291.182945	1239.0
	Anza Vista	267.932583	1239.0
	Bayview	170.098665	1239.0
	Buena Vista Park	347.394919	1239.0
	Central Richmond	319.027623	1239.0
...	**...**	...	...
2016	**Telegraph Hill**	903.049771	4390.0
	Twin Peaks	970.085470	4390.0
	Van Ness/ Civic Center	552.602567	4390.0
	Visitacion Valley	328.319007	4390.0
	Westwood Park	631.195426	4390.0

397 rows × 2 columns

In [21]:

```
prices_by_year_by_neighborhood
```

Out[21]:

		sale_price_sqr_foot	gross_rent
year	**neighborhood**		
2010	**Alamo Square**	291.182945	1239.0
	Anza Vista	267.932583	1239.0
	Bayview	170.098665	1239.0
	Buena Vista Park	347.394919	1239.0
	Central Richmond	319.027623	1239.0
...	**...**	...	...
2016	**Telegraph Hill**	903.049771	4390.0
	Twin Peaks	970.085470	4390.0
	Van Ness/ Civic Center	552.602567	4390.0
	Visitacion Valley	328.319007	4390.0
	Westwood Park	631.195426	4390.0

397 rows × 2 columns

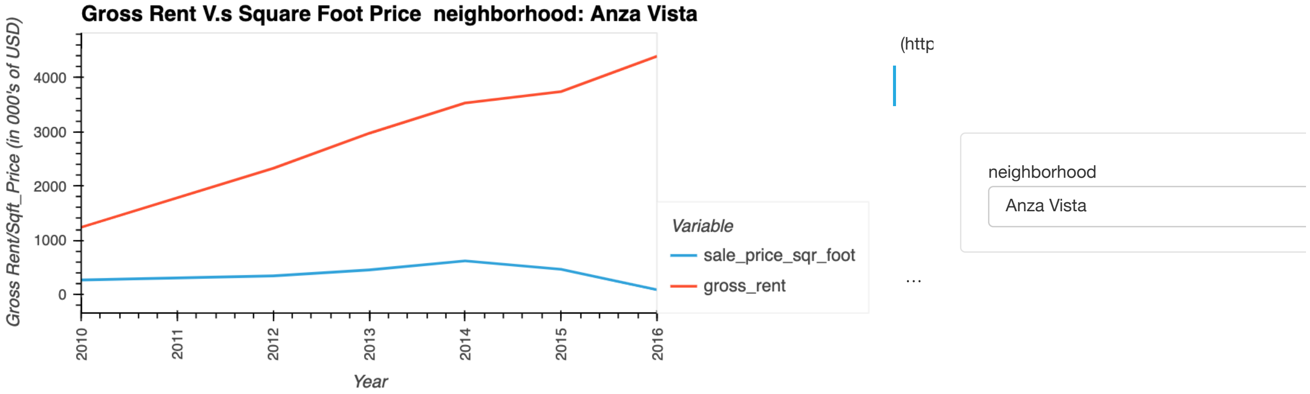
Step 3: Create an interactive line plot with hvPlot that visualizes both `sale_price_sqr_foot` and `gross_rent`. Set the x-axis parameter to the year (`x="year"`). Use the `groupby` parameter to create an interactive widget for `neighborhood`.

Step 4: Style and format the line plot to ensure a professionally styled visualization.

In [22]:

```
# Use hvplot to create an interactive line plot of the average price per square foot
# The plot should have a dropdown selector for the neighborhood
lineplot2 = prices_by_year_by_neighborhood.hvplot.line(
    xlabel="Year",
    ylabel="Gross Rent/Sqft_Price (in 000's of USD)",
    label="Gross Rent V.s Square Foot Price",
    groupby = "neighborhood",
    rot=90
).opts(
    yformatter='%.0f'
)
lineplot2
```

Out[22]:



Step 6: Use the interactive visualization to answer the following question:

Question: For the Anza Vista neighborhood, is the average sale price per square foot for 2016 more or less than the price that's listed for 2012?

In [23]:

```
print('For the Anza Vista neighborhood, the average sale price per square foot for 2016 is 74.34% less than the price t
```

```
For the Anza Vista neighborhood, the average sale price per square foot for 2016 is 74.34% less than the price that’s
listed for 2012.
```

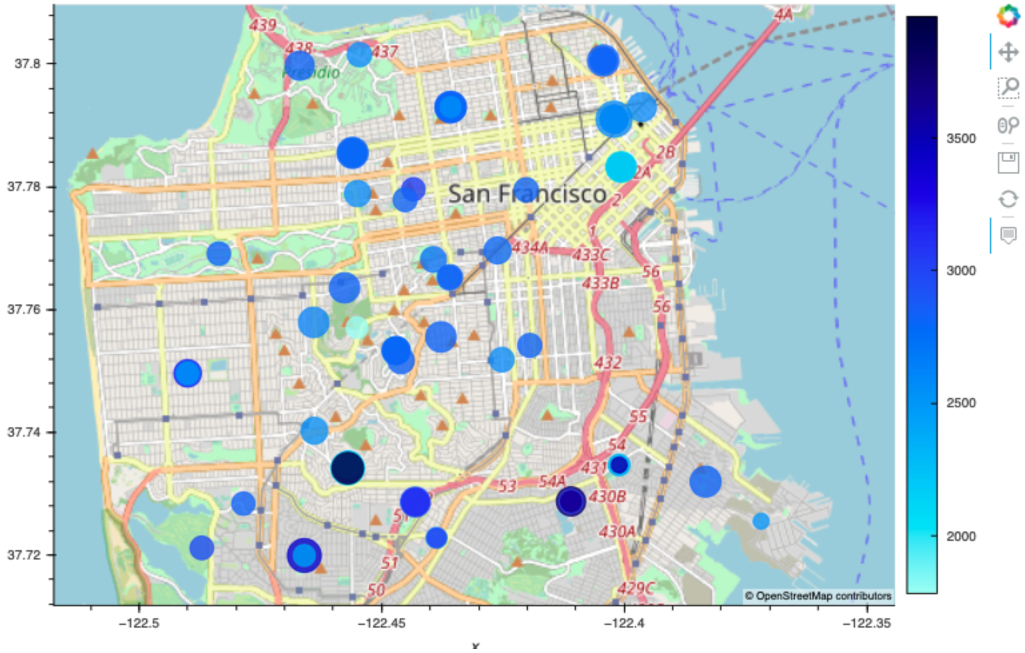
Build an Interactive Neighborhood Map

For this part of the assignment, explore the geospatial relationships in the data by using interactive visualizations with hvPlot and GeoViews. To build your map, use the `sfo_data_df` DataFrame (created during the initial import), which includes the neighborhood location data with the average prices. To do all this, complete the following steps:

1. Read the `neighborhood_coordinates.csv` file from the `Resources` folder into the notebook, and create a DataFrame named `neighborhood_locations_df`. Be sure to set the `index_col` of the DataFrame as "Neighborhood".
2. Using the original `sfo_data_df` Dataframe, create a DataFrame named `all_neighborhood_info_df` that groups the data by neighborhood. Aggregate the results by the `mean` of the group.
3. Review the two code cells that concatenate the `neighborhood_locations_df` DataFrame with the `all_neighborhood_info_df` DataFrame. Note that the first cell uses the [Pandas concat function (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html) to create a DataFrame named `all_neighborhoods_df`. The second cell cleans the data and sets the "Neighborhood" column. Be sure to run these cells to create the `all_neighborhoods_df` DataFrame, which you'll need to create the geospatial visualization.
4. Using hvPlot with GeoViews enabled, create a `points` plot for the `all_neighborhoods_df` DataFrame. Be sure to do the following:

   - Set the `size` parameter to "sale_price_sqr_foot".
   - Set the `color` parameter to "gross_rent".
   - Set the `size_max` parameter to "25".
   - Set the `zoom` parameter to "11".

Note that your resulting plot should appear similar to the following image:



5. Use the interactive map to answer the following question:

   - Which neighborhood has the highest gross rent, and which has the highest sale price per square foot?

Step 1: Read the `neighborhood_coordinates.csv` file from the `Resources` folder into the notebook, and create a DataFrame named `neighborhood_locations_df`. Be sure to set the `index_col` of the DataFrame as “Neighborhood”.

In [24]:

```
/mohjaiswal/Desktop/Unit-6-Homework-Asn/Starter_Code/Resources/neighborhoods_coordinates.csv',index_col ="Neighborhood")
```

Out[24]:

	Lat	Lon
Neighborhood		
Alamo Square	37.791012	-122.402100
Anza Vista	37.779598	-122.443451
Bayview	37.734670	-122.401060
Bayview Heights	37.728740	-122.410980
Bernal Heights	37.728630	-122.443050
...	...	...
West Portal	37.740260	-122.463880
Western Addition	37.792980	-122.435790
Westwood Highlands	37.734700	-122.456854
Westwood Park	37.734150	-122.457000
Yerba Buena	37.792980	-122.396360

73 rows × 2 columns

Step 2: Using the original `sfo_data_df` Dataframe, create a DataFrame named `all_neighborhood_info_df` that groups the data by neighborhood. Aggregate the results by the `mean` of the group.

In [25]:

```
# Calculate the mean values for each neighborhood
all_neighborhood_info_df = sfo_data_df.groupby('neighborhood').mean()

# Review the resulting DataFrame
all_neighborhood_info_df
```

Out[25]:

	year	sale_price_sqr_foot	housing_units	gross_rent
neighborhood				
Alamo Square	2013.000000	366.020712	378401.00	2817.285714
Anza Vista	2013.333333	373.382198	379050.00	3031.833333
Bayview	2012.000000	204.588623	376454.00	2318.400000
Bayview Heights	2015.000000	590.792839	382295.00	3739.000000
Bernal Heights	2013.500000	576.746488	379374.50	3080.333333
...	...	...	...	...
West Portal	2012.250000	498.488485	376940.75	2515.500000
Western Addition	2012.500000	307.562201	377427.50	2555.166667
Westwood Highlands	2012.000000	533.703935	376454.00	2250.500000
Westwood Park	2015.000000	687.087575	382295.00	3959.000000

In [51]:

```
#Dropping 'year'
all_neighborhood_info_df = all_neighborhood_info_df.drop(columns='year')
# Review the resulting DataFrame
all_neighborhood_info_df
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [51], in <cell line: 2>()
      1 #Dropping 'year'
----> 2 all_neighborhood_info_df = all_neighborhood_info_df.drop(columns='year')
      3 # Review the resulting DataFrame
      4 all_neighborhood_info_df

File ~/opt/anaconda3/lib/python3.9/site-packages/pandas/util/_decorators.py:311, in deprecate_nonkeyword_arguments.<l
ocals>.decorate.<locals>.wrapper(*args, **kwargs)
    305 if len(args) > num_allow_args:
    306     warnings.warn(
    307         msg.format(arguments=arguments),
    308         FutureWarning,
    309         stacklevel=stacklevel,
    310     )
--> 311 return func(*args, **kwargs)

File ~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/frame.py:4954, in DataFrame.drop(self, labels, axis, ind
ex, columns, level, inplace, errors)
   4806 @deprecate_nonkeyword_arguments(version=None, allowed_args=["self", "labels"])
   4807 def drop(
   4808     self,
   (...)
   4815     errors: str = "raise",
   4816 ):
   4817     """
   4818     Drop specified labels from rows or columns.
   4819
   (...)
   4952             weight  1.0     0.8
   4953     """
-> 4954     return super().drop(
   4955         labels=labels,
   4956         axis=axis,
   4957         index=index,
   4958         columns=columns,
   4959         level=level,
   4960         inplace=inplace,
   4961         errors=errors,
   4962     )

File ~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/generic.py:4267, in NDFrame.drop(self, labels, axis, ind
ex, columns, level, inplace, errors)
   4265 for axis, labels in axes.items():
   4266     if labels is not None:
-> 4267         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4269 if inplace:
   4270     self._update_inplace(obj)

File ~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/generic.py:4311, in NDFrame._drop_axis(self, labels, axi
s, level, errors, consolidate, only_slice)
   4309         new_axis = axis.drop(labels, level=level, errors=errors)
   4310     else:
-> 4311         new_axis = axis.drop(labels, errors=errors)
   4312     indexer = axis.get_indexer(new_axis)
   4314 # Case for non-unique axis
   4315 else:

File ~/opt/anaconda3/lib/python3.9/site-packages/pandas/core/indexes/base.py:6644, in Index.drop(self, labels, error
s)
   6642 if mask.any():
   6643     if errors != "ignore":
-> 6644         raise KeyError(f"{list(labels[mask])} not found in axis")
   6645     indexer = indexer[~mask]
   6646 return self.delete(indexer)

KeyError: "['year'] not found in axis"
```

In [52]:

```
all_neighborhood_info_df.to_csv('hw6_data.csv', index=False)
```

Step 3: Review the two code cells that concatenate the `neighborhood_locations_df` DataFrame with the `all_neighborhood_info_df` DataFrame.

Note that the first cell uses the [Pandas concat function (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.concat.html) to create a DataFrame named `all_neighborhoods_df` .

The second cell cleans the data and sets the “Neighborhood” column.

Be sure to run these cells to create the `all_neighborhoods_df` DataFrame, which you’ll need to create the geospatial visualization.

In [27]:

```
# Using the Pandas `concat` function, join the
# neighborhood_locations_df and the all_neighborhood_info_df DataFrame
# The axis of the concatenation is "columns".
# The concat function will automatically combine columns with
# identical information, while keeping the additional columns.
all_neighborhoods_df = pd.concat(
    [neighborhood_locations_df, all_neighborhood_info_df],
    axis="columns",
    sort=False
)

# Review the resulting DataFrame
display(all_neighborhoods_df.head())
display(all_neighborhoods_df.tail())
```

	Lat	Lon	sale_price_sqr_foot	housing_units	gross_rent
Alamo Square	37.791012	-122.402100	366.020712	378401.0	2817.285714
Anza Vista	37.779598	-122.443451	373.382198	379050.0	3031.833333
Bayview	37.734670	-122.401060	204.588623	376454.0	2318.400000
Bayview Heights	37.728740	-122.410980	590.792839	382295.0	3739.000000
Bernal Heights	37.728630	-122.443050	NaN	NaN	NaN

	Lat	Lon	sale_price_sqr_foot	housing_units	gross_rent
Yerba Buena	37.79298	-122.39636	576.709848	377427.5	2555.166667
Bernal Heights	NaN	NaN	576.746488	379374.5	3080.333333
Downtown	NaN	NaN	391.434378	378401.0	2817.285714
Ingleside	NaN	NaN	367.895144	377427.5	2509.000000
Outer Richmond	NaN	NaN	473.900773	378401.0	2817.285714

In [55]:

```
all_neighborhoods_df.to_csv('hw6_data_final.csv', index=False)
```

In [56]:

```
# Call the dropna function to remove any neighborhoods that do not have data
all_neighborhoods_df = all_neighborhoods_df.reset_index().dropna()

# Rename the "index" column as "Neighborhood" for use in the Visualization
all_neighborhoods_df = all_neighborhoods_df.rename(columns={"index": "Neighborhood"})

# Review the resulting DataFrame
display(all_neighborhoods_df.head())
display(all_neighborhoods_df.tail())
```

	Neighborhood	Neighborhood	Lat	Lon	sale_price_sqr_foot	housing_units	gross_rent
0	0	Alamo Square	37.791012	-122.402100	366.020712	378401.0	2817.285714
1	1	Anza Vista	37.779598	-122.443451	373.382198	379050.0	3031.833333
2	2	Bayview	37.734670	-122.401060	204.588623	376454.0	2318.400000
3	3	Bayview Heights	37.728740	-122.410980	590.792839	382295.0	3739.000000
4	5	Buena Vista Park	37.768160	-122.439330	452.680591	378076.5	2698.833333

	Neighborhood	Neighborhood	Lat	Lon	sale_price_sqr_foot	housing_units	gross_rent
64	68	West Portal	37.74026	-122.463880	498.488485	376940.75	2515.500000
65	69	Western Addition	37.79298	-122.435790	307.562201	377427.50	2555.166667
66	70	Westwood Highlands	37.73470	-122.456854	533.703935	376454.00	2250.500000
67	71	Westwood Park	37.73415	-122.457000	687.087575	382295.00	3959.000000

Step 4: Using hvPlot with GeoViews enabled, create a `points` plot for the `all_neighborhoods_df` DataFrame. Be sure to do the following:

- Set the `geo` parameter to True.
- Set the `size` parameter to “sale_price_sqr_foot”.
- Set the `color` parameter to “gross_rent”.
- Set the `frame_width` parameter to 700.
- Set the `frame_height` parameter to 500.
- Include a descriptive title.

In [43]:

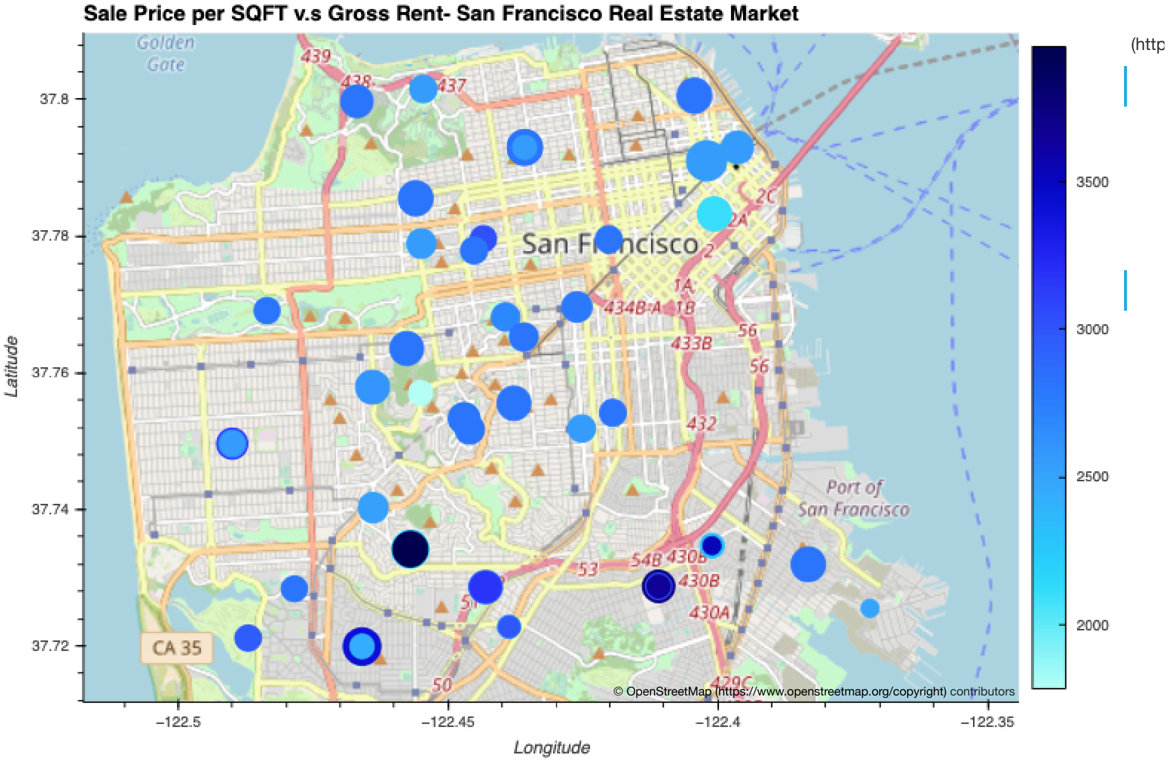
```
import cartopy
import geoviews as gv
import pyproj
```

In [44]:

```
# Create a plot to analyze neighborhood info
all_neighborhoods_df.hvplot.points(
    'Lon',
    'Lat',
    xlabel = 'Longitude',
    ylabel = 'Latitude',
    geo = True,
    tiles = 'OSM',
    size = 'sale_price_sqr_foot',
    color = 'gross_rent',
    frame_width = 700,
    frame_height = 500,
    title = "Sale Price per SQFT v.s Gross Rent- San Francisco Real Estate Market"
)
```

```
/Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages/cartopy/crs.py:245: ShapelyDeprecationWarning: __len__ fo
r multi-part geometries is deprecated and will be removed in Shapely 2.0. Check the length of the `geoms` property in
stead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages/cartopy/crs.py:297: ShapelyDeprecationWarning: Iteration
over multi-part geometries is deprecated and will be removed in Shapely 2.0. Use the `geoms` property to access the c
onstituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages/cartopy/crs.py:364: ShapelyDeprecationWarning: __len__ fo
r multi-part geometries is deprecated and will be removed in Shapely 2.0. Check the length of the `geoms` property in
stead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
/Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages/cartopy/crs.py:245: ShapelyDeprecationWarning: __len__ fo
r multi-part geometries is deprecated and will be removed in Shapely 2.0. Check the length of the `geoms` property in
stead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages/cartopy/crs.py:297: ShapelyDeprecationWarning: Iteration
over multi-part geometries is deprecated and will be removed in Shapely 2.0. Use the `geoms` property to access the c
onstituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages/cartopy/crs.py:364: ShapelyDeprecationWarning: __len__ fo
r multi-part geometries is deprecated and will be removed in Shapely 2.0. Check the length of the `geoms` property in
stead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
/Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages/cartopy/crs.py:245: ShapelyDeprecationWarning: __len__ fo
r multi-part geometries is deprecated and will be removed in Shapely 2.0. Check the length of the `geoms` property in
stead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages/cartopy/crs.py:297: ShapelyDeprecationWarning: Iteration
over multi-part geometries is deprecated and will be removed in Shapely 2.0. Use the `geoms` property to access the c
onstituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/mohjaiswal/opt/anaconda3/lib/python3.9/site-packages/cartopy/crs.py:364: ShapelyDeprecationWarning: __len__ fo
r multi-part geometries is deprecated and will be removed in Shapely 2.0. Check the length of the `geoms` property in
stead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
```

Out[44]:


Sale Price per SQFT v.s Gross Rent- San Francisco Real Estate Market
Latitude
37.8
37.78
37.76
37.74
37.72
Longitude
−122.5
−122.45
−122.4
−122.35
3500
3000
2500
2000
Golden Gate
San Francisco
Port of San Francisco
CA 35
© OpenStreetMap (https://www.openstreetmap.org/copyright) contributors
(http

Step 5: Use the interactive map to answer the following question:

Question: Which neighborhood has the highest gross rent, and which has the highest sale price per square foot?

Answer: Westwood Park has the highest gross rent which is 3539 USD. Union Square District has the highest sales price per square foot at 904 USD.

In [60]:

```
all_neighborhoods_df.max()
```

Out[60]:

```
Neighborhood                    72
Neighborhood           Yerba Buena
Lat                       37.80152
Lon                     -122.37178
sale_price_sqr_foot     903.993258
housing_units             382295.0
gross_rent                  3959.0
dtype: object
```

Compose Your Data Story

Based on the visualizations that you have created, compose a data story that synthesizes your analysis by answering the following questions:

Question: How does the trend in rental income growth compare to the trend in sales prices? Does this same trend hold true for all the neighborhoods across San Francisco?

Answer: Growth rate for rental income is very high, where there is a significant upward trend. Price per square foot on the other hand is relatively flat in comparison (Same axis). This might seem flatter due to they both having the same axes. While it is true that Rental income growth is higher compared to sqare footage price, (430% vs 122%) both have a high percent change. We can see the individual change better in the optional diagram in line 15. Note: Gross rent is inclusive of all livable area not just 1 square foot.

Question: What insights can you share with your company about the potential one-click, buy-and-rent strategy that they're pursuing? Do neighborhoods exist that you would suggest for investment, and why?

Answer: It would depend solely on the person carrying out the click. Their goals, budget, tenure of and ability to carry out on a buy or rent decision. If they are purely trying to create value, they should only focus on buying a house as we can see the gross rent outpaces housing prices. Over time it will be cheaper to own their own home rather than renting. They will not only pay more money. But if they want to earn income from real estate. They should own a home in high rent neighbourhood, but live in a location with moderate rents. (Look at high rentals but lowest sq/ft prices.

```
In [59]: all_neighborhoods_df.describe()
```

Out[59]:

	Neighborhood	Lat	Lon	sale_price_sqr_foot	housing_units	gross_rent
count	69.000000	69.000000	69.000000	69.000000	69.000000	69.000000
mean	36.768116	37.760641	-122.438264	484.561198	378203.007971	2764.561111
std	21.188333	0.026469	0.028855	146.952722	1320.744135	364.019847
min	0.000000	37.719930	-122.489990	170.292549	374507.000000	1781.500000
25%	19.000000	37.734670	-122.456854	388.765927	377427.500000	2555.166667
50%	37.000000	37.755540	-122.439330	478.228553	378401.000000	2817.285714
75%	55.000000	37.785530	-122.410980	583.749269	378401.000000	2817.285714
max	72.000000	37.801520	-122.371780	903.993258	382295.000000	3959.000000

```
In [64]: !export PATH=/Library/TeX/texbin:$/Library/TeX/texbin/xelatex
```

```
In [65]:  jupyter nbconvert MJ_Working.ipynb --to pdf
```

```
  Input In [65]
    jupyter nbconvert MJ_Working.ipynb --to pdf
            ^
SyntaxError: invalid syntax
```

```
In [ ]:
```