In [1]:
```
"""The point of this code block is to import libraries and load data.

The data files are four CSV files located in the Data Source subfolder
of the root drive.
"""
# Import libraries. Commented out the pip install commands since they
# only need to be run once.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
# %pip install seaborn
# %pip install scipy
# %pip install folium
# %pip install geopy
import folium
import seaborn as sns
from datetime import datetime


# Read CSV files into DataFrames.
#../Data Source/airbnb_vancouver_2023_listings/
abnb_q4 = pd.read_csv('listings_Dec2023.csv')
abnb_q3 = pd.read_csv('listings_Sep2023.csv')
abnb_q2 = pd.read_csv('listings_Jun2023.csv')
abnb_q1 = pd.read_csv('listings_Mar2023.csv')

# Stack the four DataFrames into one and print out columns.
abnb_orig = pd.concat(
    [abnb_q4,
     abnb_q3,
     abnb_q2,
     abnb_q1
     ],
    ignore_index=True
)
```

In [2]:
```
"""Data cleaning code."""
# Keep just the code we want.
abnb = abnb_orig[
    ['last_scraped',
     'id',
     'host_id',
     'host_since',
     'host_is_superhost',
     'host_total_listings_count',
     'neighbourhood_cleansed',
     'latitude',
     'longitude',
     'property_type',
     'room_type',
     'accommodates',
     'beds',
     'price',
     'number_of_reviews',
```

```
    'number_of_reviews_ltm',
    'review_scores_rating',
    'reviews_per_month'
    ]
]

# Rename the neighbourhood column to make it more concise.
abnb = abnb.rename(columns={"neighbourhood_cleansed": "neighbourhood"})

# Convert prices from string to float.
# Replace $ symbol with nothing.
abnb.loc[:, 'price'] = abnb.loc[:, "price"].str.replace("$", "")
# Replace commas with nothing.
abnb.loc[:, 'price'] = abnb.loc[:, "price"].str.replace(",", "")
# Change column type from string to numeric.
abnb["price"] = pd.to_numeric(abnb["price"])

# Get rid of rows that do not have price or bed data.
abnb = abnb[-pd.isna(abnb["price"])]
abnb = abnb[-pd.isna(abnb["beds"])]

# The original DataFrame had a problem with outliers.
# Remove rows with a price above $1000.00.
abnb = abnb[abnb['price'] <= 1000]

# Make a month column out of the last_scraped column.
abnb['last_scraped'] = pd.to_datetime(abnb['last_scraped'])
abnb['month'] = abnb['last_scraped'].dt.month

# Inspect values.
print(abnb['month'].value_counts())

# Make a quarter column.
month_to_quarter = {
    3: 1,
    6: 2,
    9: 3,
    12: 4
}
abnb['quarter'] = abnb['month'].map(month_to_quarter)

# Define the q4 DataFrame as anything in the 12th month.
abnb_q4 = abnb[abnb['month'] == 12]

# Show some information.
print(abnb.info())
print(abnb_q4.info())
print(abnb.head(5))
```

```
month
9     6591
6     6236
3     5873
12    5769
Name: count, dtype: int64
<class 'pandas.core.frame.DataFrame'>
Index: 24469 entries, 0 to 25715
Data columns (total 20 columns):
 #   Column                     Non-Null Count  Dtype         
---  ------                     --------------  -----         
 0   last_scraped               24469 non-null  datetime64[ns]
 1   id                         24469 non-null  int64         
 2   host_id                    24469 non-null  int64         
 3   host_since                 24469 non-null  object        
 4   host_is_superhost          23107 non-null  object        
 5   host_total_listings_count  24469 non-null  int64         
 6   neighbourhood              24469 non-null  object        
 7   latitude                   24469 non-null  float64       
 8   longitude                  24469 non-null  float64       
 9   property_type              24469 non-null  object        
 10  room_type                  24469 non-null  object        
 11  accommodates               24469 non-null  int64         
 12  beds                       24469 non-null  float64       
 13  price                      24469 non-null  float64       
 14  number_of_reviews          24469 non-null  int64         
 15  number_of_reviews_ltm      24469 non-null  int64         
 16  review_scores_rating       20502 non-null  float64       
 17  reviews_per_month          20495 non-null  float64       
 18  month                      24469 non-null  int32         
 19  quarter                    24469 non-null  int64         
dtypes: datetime64[ns](1), float64(6), int32(1), int64(7), object(5)
memory usage: 3.8+ MB
None
<class 'pandas.core.frame.DataFrame'>
Index: 5769 entries, 0 to 6688
Data columns (total 20 columns):
 #   Column                     Non-Null Count  Dtype         
---  ------                     --------------  -----         
 0   last_scraped               5769 non-null   datetime64[ns]
 1   id                         5769 non-null   int64         
 2   host_id                    5769 non-null   int64         
 3   host_since                 5769 non-null   object        
 4   host_is_superhost          5724 non-null   object        
 5   host_total_listings_count  5769 non-null   int64         
 6   neighbourhood              5769 non-null   object        
 7   latitude                   5769 non-null   float64       
 8   longitude                  5769 non-null   float64       
 9   property_type              5769 non-null   object        
 10  room_type                  5769 non-null   object        
 11  accommodates               5769 non-null   int64         
 12  beds                       5769 non-null   float64       
 13  price                      5769 non-null   float64       
 14  number_of_reviews          5769 non-null   int64         
 15  number_of_reviews_ltm      5769 non-null   int64         
 16  review_scores_rating       4864 non-null   float64       
```

```
 17  reviews_per_month          4857 non-null   float64
 18  month                      5769 non-null   int32
 19  quarter                    5769 non-null   int64
dtypes: datetime64[ns](1), float64(6), int32(1), int64(7), object(5)
memory usage: 923.9+ KB
None
  last_scraped     id  host_id  host_since host_is_superhost  \
0   2023-12-14  13188    51466  2009-11-04                f
1   2023-12-14  13221    51634  2009-11-05                f
2   2023-12-14  13358    52116  2009-11-07                f
3   2023-12-13  13490    52467  2009-11-08                t
4   2023-12-13  14267    56030  2009-11-20                f

   host_total_listings_count             neighbourhood   latitude   longitude  \
0                          3               Riley Park  49.247730 -123.105090
1                          4               Riley Park  49.254890 -123.097080
2                          1                 Downtown  49.281174 -123.125931
3                          4  Kensington-Cedar Cottage  49.256220 -123.066070
4                          1  Kensington-Cedar Cottage  49.249220 -123.081390

        property_type        room_type  accommodates  beds  price  \
0  Entire rental unit  Entire home/apt             4   2.0  150.0
1  Entire rental unit  Entire home/apt             4   2.0  120.0
2        Entire condo  Entire home/apt             2   1.0  165.0
3  Entire rental unit  Entire home/apt             2   1.0  150.0
4         Entire home  Entire home/apt             4   2.0  150.0

   number_of_reviews  number_of_reviews_ltm  review_scores_rating  \
0                283                     30                  4.84
1                 15                      0                  4.73
2                493                     55                  4.68
3                101                      5                  4.93
4                 33                      0                  4.76

   reviews_per_month  month  quarter
0               1.68     12        4
1               0.15     12        4
2               3.00     12        4
3               0.66     12        4
4               0.21     12        4
```

In [3]:
```
print(abnb['quarter'].value_counts())
```

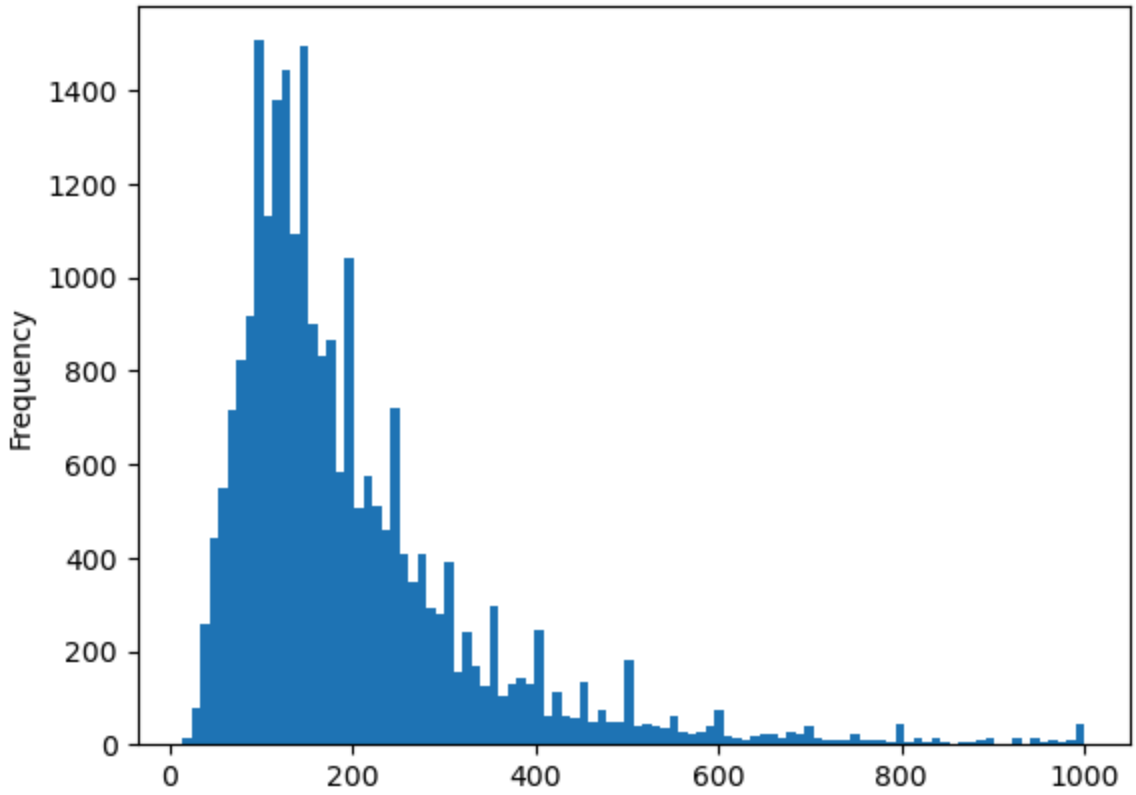
```
quarter
3    6591
2    6236
1    5873
4    5769
Name: count, dtype: int64
```

In [4]:
```
"""Histogram visualization of price in the 15 neighbourhoods
with the most listings.
"""

# Show the top 15 neighbourhoods with the most listings.
top_area = (
```

```
    abnb['neighbourhood']
    .value_counts()
    .sort_values(ascending=False)
    .head(15)
    .index
)
print(top_area)

# Plot the price of the top 15 neighbourhoods by number of listings.
abnb['price'].plot.hist(bins=100)
```

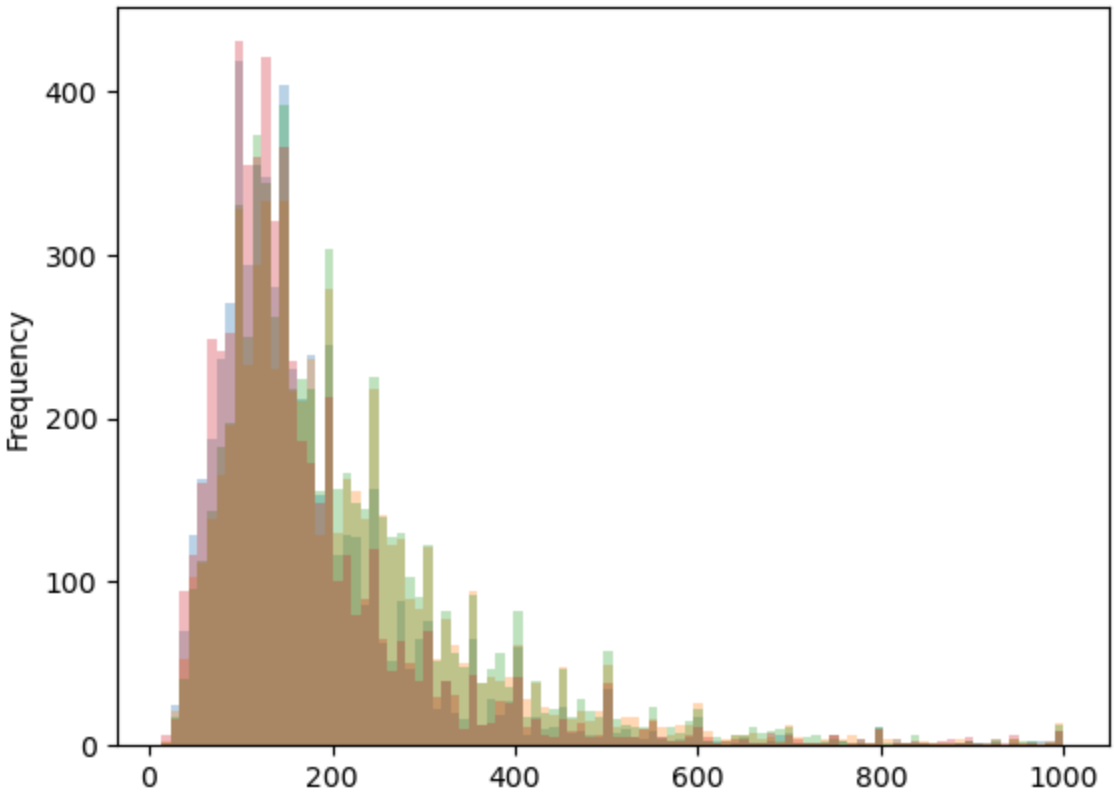
```
Index(['Downtown', 'West End', 'Kitsilano', 'Mount Pleasant',
       'Kensington-Cedar Cottage', 'Downtown Eastside', 'Riley Park',
       'Grandview-Woodland', 'Hastings-Sunrise', 'Renfrew-Collingwood',
       'Dunbar Southlands', 'Marpole', 'Fairview', 'Sunset',
       'Victoria-Fraserview'],
      dtype='object', name='neighbourhood')
```

Out[4]: <Axes: ylabel='Frequency'>



In [5]:
```
# Plot the price of the top 15 neighbourhoods by number of quarter.
abnb.groupby(["quarter"])["price"].plot(kind="hist", bins=100, alpha=0.3)
```

Out[5]:
```
quarter
1    Axes(0.125,0.11;0.775x0.77)
2    Axes(0.125,0.11;0.775x0.77)
3    Axes(0.125,0.11;0.775x0.77)
4    Axes(0.125,0.11;0.775x0.77)
Name: price, dtype: object
```



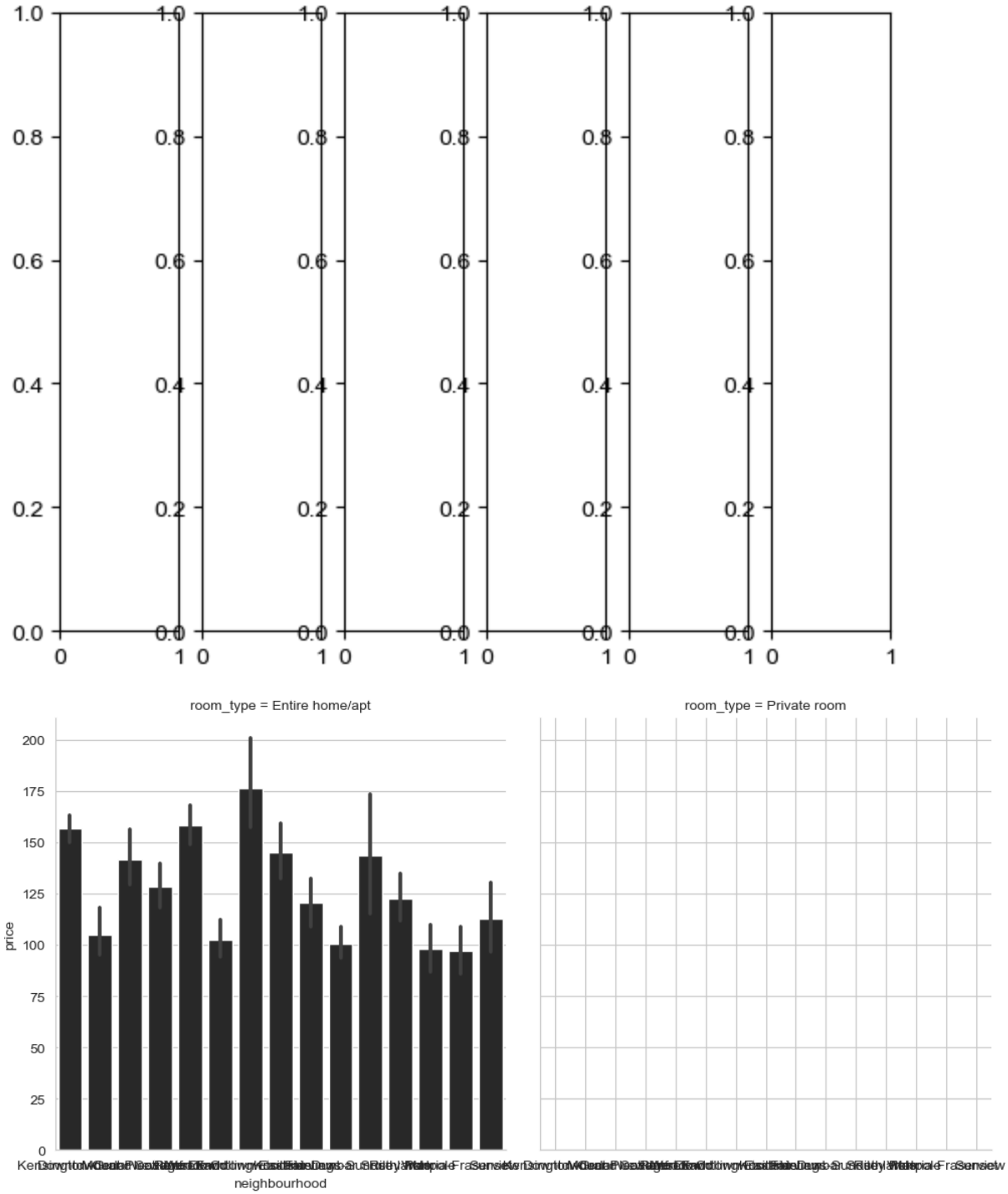
In [6]:
```
"""Subplot code."""

# Set subplot options.
fig, axes = plt.subplots(1, 6)
sns.set_style('whitegrid')
sns.set_palette('Greys_r')

# First subplot is a bar chart showing price by neighbourhood.
axes[0] = sns.catplot(
    x='neighbourhood',
    y='price',
    col='room_type',
    hue='accommodates',
    kind='bar',
    data=(abnb[abnb['accommodates'].isin([2])
               & abnb['room_type'].isin(['Entire home/apt', 'Private room'])
               & abnb['neighbourhood'].isin(top_area)
               & abnb['month'].isin([12])
               ]
          )
)
axes[0].set_xticklabels(rotation=90)

# Create pivot table for second axis plot.
stat_by_neigh = pd.pivot_table(abnb_q4,
                               values=['price',
                                       'review_scores_rating',
                                       'id',
                                       'number_of_reviews_ltm',
                                       'latitude',
                                       'longitude'
```

```
                                        ],
                                index=['neighbourhood'],
                                aggfunc={'price': 'median',
                                         'review_scores_rating': 'median',
                                         'id': 'count',
                                         'number_of_reviews_ltm': 'sum',
                                         'latitude': 'median',
                                         'longitude': 'median'
                                         }
                                )

# Sort pivot table by descending review scores.
stat_by_neigh = (
    stat_by_neigh.sort_values(
        by='review_scores_rating',
        ascending=False)
    .reset_index()
)

# Second subplot shows review scores by neighborhood.
axes[1] = sns.catplot(
    x='neighbourhood',
    y='review_scores_rating',
    kind='bar',
    data=stat_by_neigh
)
axes[1].set_xticklabels(rotation=90)
axes[1].set(ylim=(4.7, 5))

# Third subplot is a bar chart of most-expensive to least-expensive
# median prices by neighbourhood.
axes[2] = sns.catplot(
    x='neighbourhood',
    y='price',
    kind='bar',
    data=stat_by_neigh.sort_values(by='price', ascending=False)
)
axes[2].set_xticklabels(rotation=90)
axes[2].set(ylim=(70, None))

# Show the pivot table.
stat_by_neigh

# Fourth subplot is a box plot of price in the Fairview and Victoria-
# Fraserview neighborhoods.
axes[3] = sns.catplot(
    x='neighbourhood',
    y='price',
    kind='box',
    palette='Greys_r',
    data=(
        abnb[
            abnb['room_type'].isin(['Private room'])
            & abnb['neighbourhood'].isin(['Fairview', 'Victoria-Fraserview'])
        ]
    )
```

```
)

# Fifth subplot is a box plot of price in all neighbourhoods by quarter in Private
axes[4] = sns.catplot(
    x='quarter',
    y='price',
    kind='box',
    palette='Greys_r',
    data=(abnb[abnb['room_type'].isin(['Private room'])])
)

# Sixth subplot is a box plot of price by quarter in entire homes.
axes[4] = sns.catplot(
    x='quarter',
    y='price',
    kind='box',
    palette='Greys_r',
    data=(abnb[abnb['room_type'].isin(['Entire home/apt'])])
)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[6], line 9
      6 sns.set_palette('Greys_r')
      8 # First subplot is a bar chart showing price by neighbourhood.
----> 9 axes[0] = sns.catplot(
     10     x='neighbourhood',
     11     y='price',
     12     col='room_type',
     13     hue='accommodates',
     14     kind='bar',
     15     data=(abnb[abnb['accommodates'].isin([2])
     16                & abnb['room_type'].isin(['Entire home/apt', 'Private room'])
     17                & abnb['neighbourhood'].isin(top_area)
     18                & abnb['month'].isin([12])
     19                ]
     20           )
     21 )
     22 axes[0].set_xticklabels(rotation=90)
     24 # Create pivot table for second axis plot.

File c:\Users\jverc\anaconda3\Lib\site-packages\seaborn\categorical.py:3244, in catplot(data, x, y, hue, row, col, col_wrap, estimator, errorbar, n_boot, units, seed, order, hue_order, row_order, col_order, height, aspect, kind, native_scale, formatter, orient, color, palette, hue_norm, legend, legend_out, sharex, sharey, margin_titles, facet_kws, ci, **kwargs)
   3241 g = FacetGrid(**facet_kws)
   3243 # Draw the plot onto the facets
-> 3244 g.map_dataframe(plot_func, x=x, y=y, hue=hue, **plot_kws)
   3246 if p.orient == "h":
   3247     g.set_axis_labels(p.value_label, p.group_label)

File c:\Users\jverc\anaconda3\Lib\site-packages\seaborn\axisgrid.py:819, in FacetGrid.map_dataframe(self, func, *args, **kwargs)
    816     kwargs["data"] = data_ijk
    818     # Draw the plot
--> 819     self._facet_plot(func, ax, args, kwargs)
    821 # For axis labels, prefer to use positional args for backcompat
    822 # but also extract the x/y kwargs and use if no corresponding arg
    823 axis_labels = [kwargs.get("x", None), kwargs.get("y", None)]

File c:\Users\jverc\anaconda3\Lib\site-packages\seaborn\axisgrid.py:848, in FacetGrid._facet_plot(self, func, ax, plot_args, plot_kwargs)
    846     plot_args = []
    847     plot_kwargs["ax"] = ax
--> 848 func(*plot_args, **plot_kwargs)
    850 # Sort out the supporting information
    851 self._update_legend_data(ax)

File c:\Users\jverc\anaconda3\Lib\site-packages\seaborn\categorical.py:2763, in barplot(data, x, y, hue, order, hue_order, estimator, errorbar, n_boot, units, seed, orient, color, palette, saturation, width, errcolor, errwidth, capsize, dodge, ci, ax, **kwargs)
   2760 if ax is None:
   2761     ax = plt.gca()
-> 2763 plotter.plot(ax, kwargs)
```

```
   2764 return ax

File c:\Users\jverc\anaconda3\Lib\site-packages\seaborn\categorical.py:1587, in _Bar
Plotter.plot(self, ax, bar_kws)
   1585 """Make the plot."""
   1586 self.draw_bars(ax, bar_kws)
-> 1587 self.annotate_axes(ax)
   1588 if self.orient == "h":
   1589     ax.invert_yaxis()

File c:\Users\jverc\anaconda3\Lib\site-packages\seaborn\categorical.py:767, in _Cate
goricalPlotter.annotate_axes(self, ax)
    764     ax.set_ylim(-.5, len(self.plot_data) - .5, auto=None)
    766 if self.hue_names is not None:
--> 767     ax.legend(loc="best", title=self.hue_title)

File c:\Users\jverc\anaconda3\Lib\site-packages\matplotlib\axes\_axes.py:322, in Axe
s.legend(self, *args, **kwargs)
    204 @_docstring.dedent_interpd
    205 def legend(self, *args, **kwargs):
    206     """
    207     Place a legend on the Axes.
    208
   (...)
    320     .. plot:: gallery/text_labels_and_annotations/legend.py
    321     """
--> 322     handles, labels, kwargs = mlegend._parse_legend_args([self], *args, **kw
args)
    323     self.legend_ = mlegend.Legend(self, handles, labels, **kwargs)
    324     self.legend_._remove_method = self._remove_legend

File c:\Users\jverc\anaconda3\Lib\site-packages\matplotlib\legend.py:1361, in _parse
_legend_args(axs, handles, labels, *args, **kwargs)
   1357     handles = [handle for handle, label
   1358                in zip(_get_legend_handles(axs, handlers), labels)]
   1360 elif len(args) == 0:  # 0 args: automatically detect labels and handles.
-> 1361     handles, labels = _get_legend_handles_labels(axs, handlers)
   1362     if not handles:
   1363         log.warning(
   1364             "No artists with labels found to put in legend.  Note that "
   1365             "artists whose label start with an underscore are ignored "
   1366             "when legend() is called with no argument.")

File c:\Users\jverc\anaconda3\Lib\site-packages\matplotlib\legend.py:1291, in _get_l
egend_handles_labels(axs, legend_handler_map)
   1289 for handle in _get_legend_handles(axs, legend_handler_map):
   1290     label = handle.get_label()
-> 1291     if label and not label.startswith('_'):
   1292         handles.append(handle)
   1293         labels.append(label)

AttributeError: 'numpy.int64' object has no attribute 'startswith'
```



In [8]:
```
"""This code block creates a map showing median price and rating among
Vancouver neighbourhoods.
"""

# Create map object.
map = folium.Map(location=[49.24445179910618, -123.11257056533111],
                 zoom_start=12,
                 tiles="cartodb positron"
                 )

stat_by_neigh_all = pd.pivot_table(abnb,
                                   values=['price',
```

```
                                        'review_scores_rating',
                                        'id',
                                        'number_of_reviews_ltm',
                                        'latitude',
                                        'longitude'
                                        ],
                                index=['neighbourhood'],
                                aggfunc={'price': 'median',
                                         'review_scores_rating': 'median',
                                         'id': 'count',
                                         'number_of_reviews_ltm': 'sum',
                                         'latitude': 'median',
                                         'longitude': 'median'
                                         }
                                )

# Sort pivot table by descending review scores.
stat_by_neigh_all = (
    stat_by_neigh_all.sort_values(
        by='review_scores_rating',
        ascending=False)
    .reset_index()
)

# Add popup icons to map.
for index, row in stat_by_neigh_all.iterrows():
    folium.Marker(
        [row['latitude'],
         row['longitude']
         ],
        popup=(
                row['neighbourhood'] + "\n"
                + "Rating: " + str(row['review_scores_rating'])
                + "\n" + "Price: $" + str(row['price'])
        )
    ).add_to(map)

# Show map.
map
```

Out[8]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [9]:
```
"""This code block creates a map showing median price and rating among
Vancouver neighbourhoods in Q1.
"""

# Create map object.
map = folium.Map(location=[49.24445179910618, -123.11257056533111],
                 zoom_start=12,
                 tiles="cartodb positron"
                 )

abnb_q1 = abnb[abnb['quarter'] == 1]

stat_by_neigh_q1 = pd.pivot_table(abnb_q1,
                                  values=['price',
                                          'review_scores_rating',
                                          'id',
                                          'number_of_reviews_ltm',
                                          'latitude',
                                          'longitude'
                                          ],
                                  index=['neighbourhood'],
                                  aggfunc={'price': 'median',
                                           'review_scores_rating': 'median',
                                           'id': 'count',
                                           'number_of_reviews_ltm': 'sum',
                                           'latitude': 'median',
                                           'longitude': 'median'
                                           }
                                  )

# Sort pivot table by descending review scores.
stat_by_neigh_q1 = (
    stat_by_neigh_q1.sort_values(
        by='review_scores_rating',
```

```
        ascending=False)
    .reset_index()
)

# Add popup icons to map.
for index, row in stat_by_neigh_q1.iterrows():
    folium.Marker(
        [row['latitude'],
         row['longitude']
         ],
        popup=(
                row['neighbourhood'] + "\n"
                + "Rating: " + str(row['review_scores_rating'])
                + "\n" + "Price: $" + str(row['price'])
        )
    ).add_to(map)

# Show map.
map
```

Out[9]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [10]:
```
"""This code block creates a map showing median price and rating among
Vancouver neighbourhoods in Q2.
"""

# Create map object.
map = folium.Map(location=[49.24445179910618, -123.11257056533111],
                 zoom_start=12,
                 tiles="cartodb positron"
                 )

abnb_q2 = abnb[abnb['quarter'] == 2]

stat_by_neigh_q2 = pd.pivot_table(abnb_q2,
                                  values=['price',
                                          'review_scores_rating',
```

```
                                        'id',
                                        'number_of_reviews_ltm',
                                        'latitude',
                                        'longitude'
                                        ],
                                index=['neighbourhood'],
                                aggfunc={'price': 'median',
                                         'review_scores_rating': 'median',
                                         'id': 'count',
                                         'number_of_reviews_ltm': 'sum',
                                         'latitude': 'median',
                                         'longitude': 'median'
                                         }
                                )

# Sort pivot table by descending review scores.
stat_by_neigh_q2 = (
    stat_by_neigh_q2.sort_values(
        by='review_scores_rating',
        ascending=False)
    .reset_index()
)

# Add popup icons to map.
for index, row in stat_by_neigh_q2.iterrows():
    folium.Marker(
        [row['latitude'],
         row['longitude']
         ],
        popup=(
                row['neighbourhood'] + "\n"
                + "Rating: " + str(row['review_scores_rating'])
                + "\n" + "Price: $" + str(row['price'])
        )
    ).add_to(map)

# Show map.
map
```

Out[10]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [ ]:
```
"""This code block creates a map showing median price and rating among
Vancouver neighbourhoods in Q3.
"""

# Create map object.
map = folium.Map(location=[49.24445179910618, -123.11257056533111],
                 zoom_start=12,
                 tiles="cartodb positron"
                 )

abnb_q3 = abnb[abnb['quarter'] == 3]

stat_by_neigh_q3 = pd.pivot_table(abnb_q3,
                                  values=['price',
                                          'review_scores_rating',
                                          'id',
                                          'number_of_reviews_ltm',
                                          'latitude',
                                          'longitude'
                                          ],
                                  index=['neighbourhood'],
                                  aggfunc={'price': 'median',
                                           'review_scores_rating': 'median',
                                           'id': 'count',
                                           'number_of_reviews_ltm': 'sum',
                                           'latitude': 'median',
                                           'longitude': 'median'
                                           }
                                  )

# Sort pivot table by descending review scores.
stat_by_neigh_q3 = (
    stat_by_neigh_q3.sort_values(
        by='review_scores_rating',
```

```
        ascending=False)
    .reset_index()
)

# Add popup icons to map.
for index, row in stat_by_neigh_q3.iterrows():
    folium.Marker(
        [row['latitude'],
         row['longitude']
         ],
        popup=(
                row['neighbourhood'] + "\n"
                + "Rating: " + str(row['review_scores_rating'])
                + "\n" + "Price: $" + str(row['price'])
        )
    ).add_to(map)

# Show map.
map
```

Out[ ]: Make this Notebook Trusted to load map: File -> Trust Notebook

In [11]:
```
"""This code block creates a map showing median price and rating among
Vancouver neighbourhoods in Q4 (using the already-created stat_by_neigh pivot
table since that was Q4).
"""

# Create map object.
map = folium.Map(location=[49.24445179910618, -123.11257056533111],
                 zoom_start=12,
                 tiles="cartodb positron"
                 )

# Add popup icons to map.
for index, row in stat_by_neigh.iterrows():
    folium.Marker(
        [row['latitude'],
```

```
         row['longitude']
         ],
        popup=(
                row['neighbourhood'] + "\n"
                + "Rating: " + str(row['review_scores_rating'])
                + "\n" + "Price: $" + str(row['price'])
        )
    ).add_to(map)

# Show map.
map
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[11], line 13
      7 map = folium.Map(location=[49.24445179910618, -123.11257056533111],
      8                  zoom_start=12,
      9                  tiles="cartodb positron"
     10                  )
     12 # Add popup icons to map.
---> 13 for index, row in stat_by_neigh.iterrows():
     14     folium.Marker(
     15         [row['latitude'],
     16          row['longitude']
   (...)
     22         )
     23     ).add_to(map)
     25 # Show map.

NameError: name 'stat_by_neigh' is not defined
```

In [ ]:
```
#drop any rows where there are no listed prices or ratings
abnb = abnb.rename(columns={"review_scores_rating":"rating"})
abnb = abnb.dropna()

display(abnb)
```

In [ ]:
```
#boxplot by neighborhood for all listings with rating
abnb.boxplot(by ='neighbourhood', column =['rating'], grid = False,rot=90)
```

In [ ]:
```
#to remove outliers, find interquartile range (IQR) for ratings
Q1 = abnb['rating'].quantile(0.25)
Q3 = abnb['rating'].quantile(0.75)
IQR = Q3 - Q1

# Filtering Values in interquartile range (between Q1-1.5IQR and Q3+1.5IQR)
rating_iqr = abnb.query('(@Q1 - 1.5 * @IQR) <= rating <= (@Q3 + 1.5 * @IQR)')


display(rating_iqr)
```

In [ ]:
```
#boxplot by neighborhood for interquartile range
rating_iqr.boxplot(by ='neighbourhood', column =['rating'], grid = False,rot=90)
```

In [ ]:
```
#Using IQR range of data, create scatterplots for quick visualization of any correl

#Removing columns not needed for scatterplot
rating_iqr_scat = rating_iqr[['neighbourhood', 'rating', 'price']]

#scatterplot rating vs price
sns.jointplot(data=rating_iqr_scat, x="rating", y="price", hue="neighbourhood")
plt.legend(bbox_to_anchor=(1, 1.2), loc='upper left', borderaxespad=6.5)


#scatterplot matrix rating vs price
sns.pairplot(rating_iqr_scat, hue='neighbourhood')



#correlation between price and rating
print(rating_iqr_scat[['rating', 'price']].corr())
```

In [ ]:
```
#calculating median price and median rating, grouped by neighbourhood

rating_iqr_neigh = pd.pivot_table(rating_iqr,values=['price','rating','id','number_
                                  index=['neighbourhood'],aggfunc={'price':'median','r
                                                                   'id':'count','numbe
                                                                   'latitude':'median'

rating_iqr_neigh = rating_iqr_neigh.sort_values(by='rating',ascending=False).reset_
rating_iqr_neigh = rating_iqr_neigh.sort_values(by='price', ascending=False)

rating_iqr_neigh = rating_iqr_neigh.rename(columns={"price":"median_price", "rating

display(rating_iqr_neigh)
```

In [ ]:
```
#bar chart mean rating by neighborhood

axes[1]=sns.catplot(x='neighbourhood',y='median_rating',kind='bar',data=rating_iqr_
axes[1].set_axis_labels("Neighbourhood", "Median Rating")
axes[1].set_xticklabels(rotation=90)
axes[1].set(ylim=(4.7,5))
```

In [ ]:
```
#scatterplot of median rating vs median price using Seaborn

ax = sns.lmplot(x='median_rating', # Horizontal axis
                y='median_price', # Vertical axis
                data=rating_iqr_neigh, # Data source
                fit_reg=False, # Don't fix a regression line
                aspect=2) # size and dimension

plt.title('Median Rating vs Median Price by Neighborhood')
# Set x-axis label
plt.xlabel('Median Rating')
# Set y-axis label
plt.ylabel('Median Price')
```

```
def label_point(x, y, val, ax):
    a = pd.concat({'x': x, 'y': y, 'val': val}, axis=1)
    for i, point in a.iterrows():
        ax.text(point['x']+0.001, point['y'], str(point['val']))

label_point(rating_iqr_neigh.median_rating, rating_iqr_neigh.median_price, rating_i
```

In [ ]:
```
#histogram of number of reviews


rating_iqr['number_of_reviews'].hist(grid=False, bins=range(0,500,50))
plt.xlabel('Number of reviews for a listing')
plt.ylabel('Number of listings')
plt.title('Summary of listings based on total reviews')
```

In [ ]:
```
#histogram of reviews of listings with 50 reviews or less

rating_iqr['number_of_reviews'].hist(grid=False, bins=range(0,50,1))
plt.xlabel('Number of reviews for a given listing')
plt.ylabel('Number of listings')
plt.title('Number of listings with 50 reviews or less')
```

In [12]:
```
"This Block of Code adds the Vancouver Parks/Green Spaces Dataset to be used for an

parks_df = pd.read_csv('parks.csv',sep=';')

parks_df[['Latitude', 'Longitude']] = parks_df['GoogleMapDest'].str.split(',', expa

parks_df
```

Out[12]:

	ParkID	Name	Official	Advisories	SpecialFeatures	Facilities	Washrooms	Street
0	1	Arbutus Village Park	1	N	N	Y	N	
1	4	Park Site on Puget Drive	0	N	N	N	N	
2	10	Andy Livingstone Park	1	N	N	Y	Y	
3	14	Coopers' Park	1	N	Y	Y	N	
4	18	Devonian Harbour Park	1	N	Y	Y	N	
...	...	...	...	...	...	...	...	
211	231	Camosun	0	N	N	N	N	
212	233	Arbutus Greenway Park	1	N	N	N	N	
213	237	Yaletown Park	1	N	N	N	N	
214	244	Fraser River Trail	1	N	N	N	N	
215	245	Trillium Park	0	N	N	Y	Y	

216 rows × 17 columns

In [13]:
```
"This is to verify if there's any differences in the neighbourhood attribute"
"THERE IS, We will need to create a map to replace the values in the park dataset t

print("Air BNB Neighbourhoods: ", abnb['neighbourhood'].unique(),'\n')
print("Parks Neighbourhoods: ", parks_df['NeighbourhoodName'].unique(),'\n')
```

```
Air BNB Neighbourhoods:  ['Riley Park' 'Downtown' 'Kensington-Cedar Cottage' 'Hastin
gs-Sunrise'
 'Grandview-Woodland' 'Mount Pleasant' 'West End' 'Renfrew-Collingwood'
 'Kitsilano' 'Downtown Eastside' 'Arbutus Ridge' 'Killarney'
 'South Cambie' 'Fairview' 'Dunbar Southlands' 'Shaughnessy'
 'West Point Grey' 'Kerrisdale' 'Sunset' 'Victoria-Fraserview' 'Marpole'
 'Strathcona' 'Oakridge']

Parks Neighbourhoods:  ['Arbutus-Ridge' 'Downtown' 'Dunbar-Southlands' 'Fairview'
 'Grandview-Woodland' 'Hastings-Sunrise' 'Kensington-Cedar Cottage'
 'Kerrisdale' 'Killarney' 'Kitsilano' 'Marpole' 'Mount Pleasant'
 'Renfrew-Collingwood' 'Riley Park' 'Shaughnessy' 'Strathcona' 'Sunset'
 'Victoria-Fraserview' 'West End' 'West Point Grey' 'South Cambie'
 'Oakridge']
```

In [14]:
```
"This will create the mapping for the two different values, This dataset is also mi

neighbourhood_mapping = {
    'Arbutus-Ridge': 'Arbutus Ridge',
    'Dunbar-Southlands': 'Dunbar Southlands',

}

parks_df['NeighbourhoodName'] = parks_df['NeighbourhoodName'].replace(neighbourhood
parks_df['NeighbourhoodName']
```

Out[14]:
```
0          Arbutus Ridge
1          Arbutus Ridge
2               Downtown
3               Downtown
4               Downtown
              ...
211    Dunbar Southlands
212            Kitsilano
213             Downtown
214              Marpole
215           Strathcona
Name: NeighbourhoodName, Length: 216, dtype: object
```

In [15]:
```
"This requires GEOPY addin"
"If you haven't already kindly uncomment and run the following line"

# %pip install geopy

"WILL TAKE A LONG TIME TO RUN 1-2 MINS Depnding on your CPU as it will use great ci
"calculate the distance of each individual row of our dataset"

#Calculating the distance to the nearest park using great_circle from geopy

from geopy.distance import great_circle

# Define the function to find the nearest park's distance
def find_nearest_park_distance(airbnb_location, parks_df):
    min_distance = float('inf')  # Initialize with a very large number
```

```
    # Iterate over each park in the DataFrame
    for _, park in parks_df.iterrows():
        park_location = (park['Latitude'], park['Longitude'])
        distance = great_circle(airbnb_location, park_location).meters  # Calculate

        # Update min_distance if the current park is closer
        if distance < min_distance:
            min_distance = distance

    return min_distance  # Return the minimum distance

# Apply the function to each Airbnb listing to calculate the closest park distance
abnb['distance_to_closest_park'] = abnb.apply(
    lambda x: find_nearest_park_distance((x['latitude'], x['longitude']), parks_df),
    axis=1
)
```

In [16]:
```
abnb = abnb.round({'distance_to_closest_park': 1})
abnb
```

Out[16]:

	last_scraped	id	host_id	host_since	host_is_superhost	host_to
0	2023-12-14	13188	51466	2009-11-04	f	
1	2023-12-14	13221	51634	2009-11-05	f	
2	2023-12-14	13358	52116	2009-11-07	f	
3	2023-12-13	13490	52467	2009-11-08	t	
4	2023-12-13	14267	56030	2009-11-20	f	
...	...	...	...	...	...	
25711	2023-03-14	845284101352923786	504936838	2023-03-11	f	
25712	2023-03-14	845854561820158781	49461922	2015-11-20	f	
25713	2023-03-14	845891831251005257	97054642	2016-09-28	f	
25714	2023-03-14	845936827653470100	3664868	2012-09-24	f	
25715	2023-03-14	845960950280088622	430820918	2021-11-07	t	

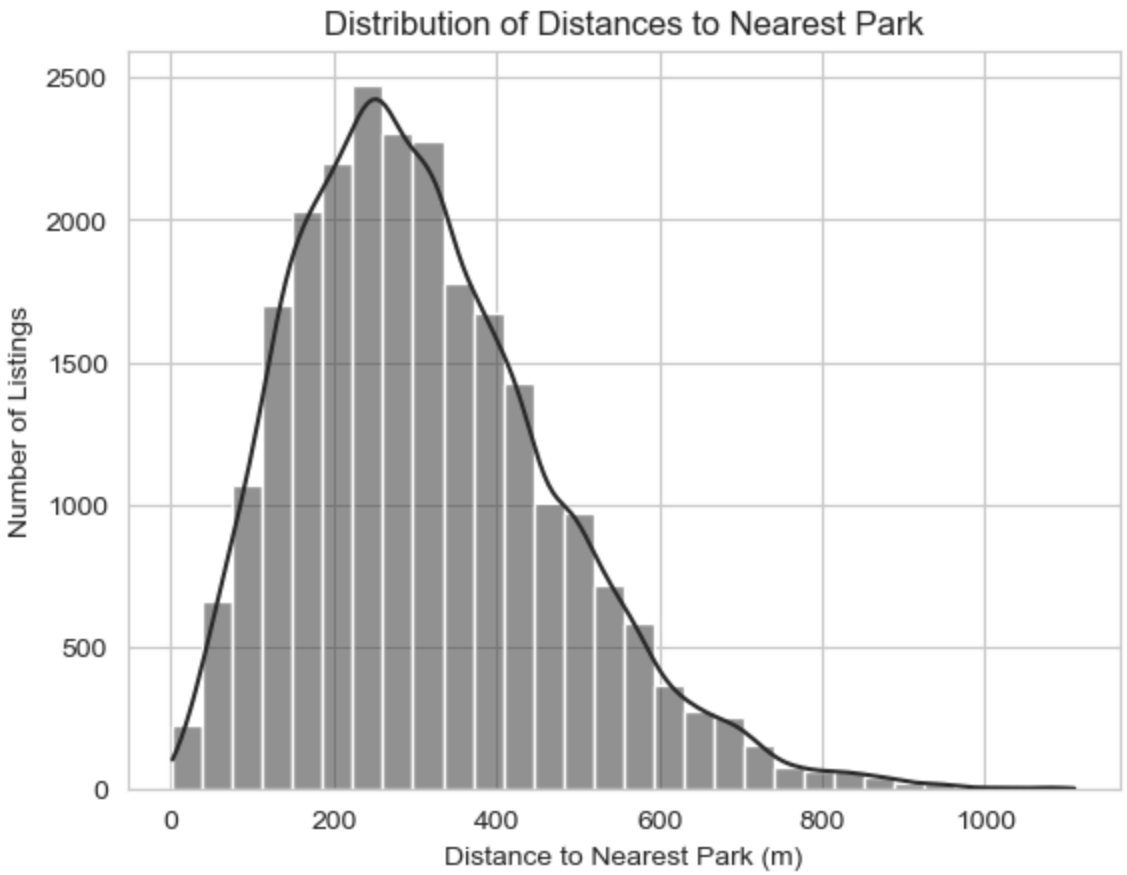
24469 rows × 21 columns

In [17]:
```
abnb.info()
```

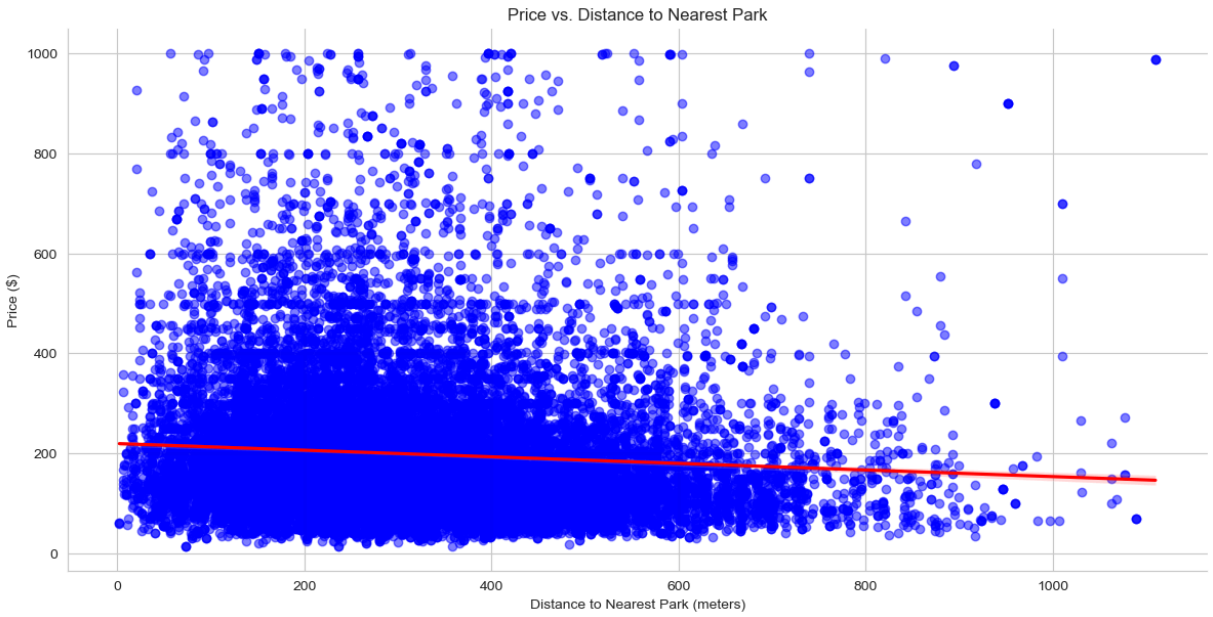
```
<class 'pandas.core.frame.DataFrame'>
Index: 24469 entries, 0 to 25715
Data columns (total 21 columns):
 #   Column                      Non-Null Count  Dtype         
---  ------                      --------------  -----         
 0   last_scraped                24469 non-null  datetime64[ns]
 1   id                          24469 non-null  int64         
 2   host_id                     24469 non-null  int64         
 3   host_since                  24469 non-null  object        
 4   host_is_superhost           23107 non-null  object        
 5   host_total_listings_count   24469 non-null  int64         
 6   neighbourhood               24469 non-null  object        
 7   latitude                    24469 non-null  float64       
 8   longitude                   24469 non-null  float64       
 9   property_type               24469 non-null  object        
 10  room_type                   24469 non-null  object        
 11  accommodates                24469 non-null  int64         
 12  beds                        24469 non-null  float64       
 13  price                       24469 non-null  float64       
 14  number_of_reviews           24469 non-null  int64         
 15  number_of_reviews_ltm       24469 non-null  int64         
 16  review_scores_rating        20502 non-null  float64       
 17  reviews_per_month           20495 non-null  float64       
 18  month                       24469 non-null  int32         
 19  quarter                     24469 non-null  int64         
 20  distance_to_closest_park    24469 non-null  float64       
dtypes: datetime64[ns](1), float64(7), int32(1), int64(7), object(5)
memory usage: 4.0+ MB
```

In [18]:
```
sns.histplot(abnb['distance_to_closest_park'], bins=30, kde=True)
plt.title('Distribution of Distances to Nearest Park')
plt.xlabel('Distance to Nearest Park (m)')
plt.ylabel('Number of Listings')
plt.show()
```

```
c:\Users\jverc\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is deprecated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [19]:
```
sns.lmplot(x='distance_to_closest_park', y='price', data=abnb,
           aspect=2, height=6, line_kws={'color': 'red'}, scatter_kws={'alpha':0.5,

plt.title('Price vs. Distance to Nearest Park')
plt.xlabel('Distance to Nearest Park (meters)')
plt.ylabel('Price ($)')

plt.show()
```



In [20]:
```
#Adding distance to closest park to our nieghborhood pivot to find the average dist

stat_by_neigh_dist = pd.pivot_table(abnb,
                                values=['price',
                                        'review_scores_rating',
                                        'id',
                                        'number_of_reviews_ltm',
                                        'latitude',
                                        'longitude','distance_to_closest_park'
                                        ],
                                index=['neighbourhood'],
                                aggfunc={'price': 'median',
                                         'review_scores_rating': 'median',
                                         'id': 'count',
                                         'number_of_reviews_ltm': 'sum',
                                         'latitude': 'median',
                                         'longitude': 'median', 'distance_to_closest
                                         }
                                )

# Sort pivot table by descending distance to closest park.
stat_by_neigh_dist = (
    stat_by_neigh_dist.sort_values(
        by='distance_to_closest_park',
        ascending=False)
    .reset_index()
)

stat_by_neigh_dist
```

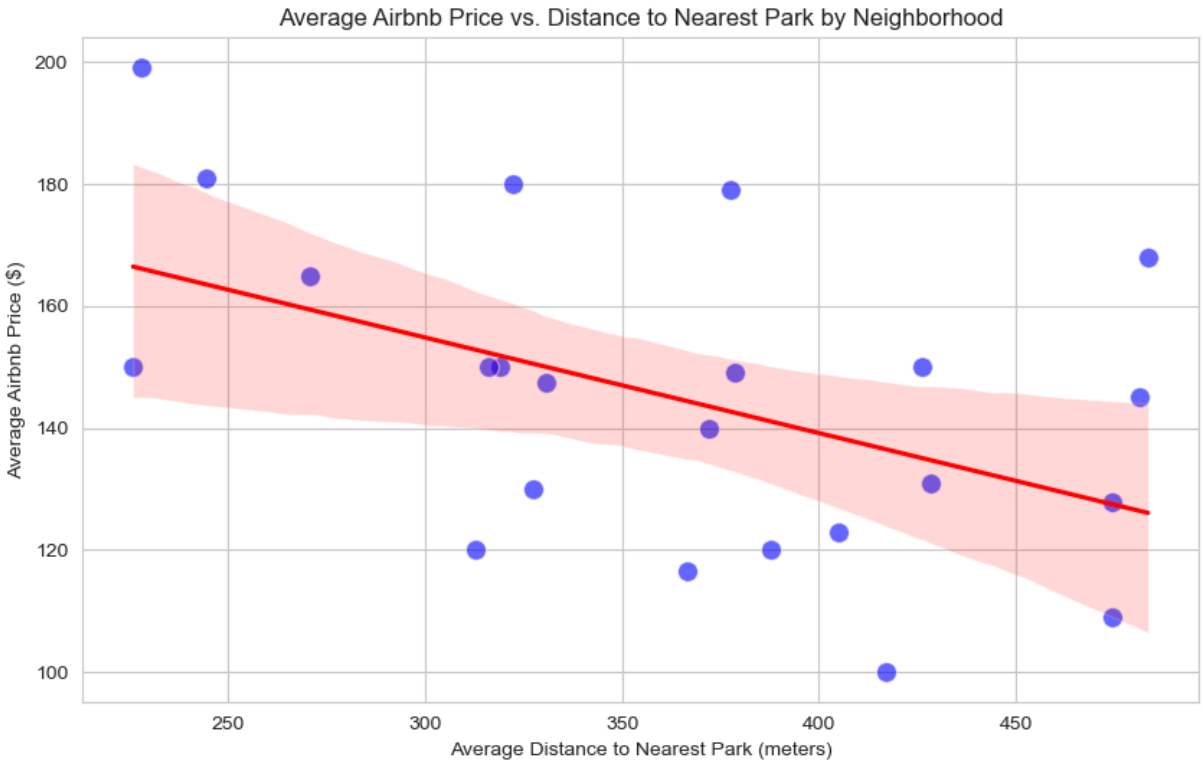
Out[20]:

	neighbourhood	distance_to_closest_park	id	latitude	longitude	number_of_revi
0	West Point Grey	483.553659	410	49.262738	-123.203940	
1	Dunbar Southlands	481.574282	766	49.244788	-123.184290	
2	Killarney	474.433333	333	49.225689	-123.035865	
3	Kerrisdale	474.359882	339	49.226650	-123.154324	
4	Sunset	428.406503	569	49.223111	-123.094410	
5	Shaughnessy	426.038047	297	49.245120	-123.137850	
6	Oakridge	416.995466	397	49.226350	-123.122280	
7	Victoria-Fraserview	404.967568	518	49.220119	-123.066560	
8	Hastings-Sunrise	387.859590	975	49.278040	-123.042930	
9	Riley Park	378.791018	1169	49.248240	-123.101450	
10	Strathcona	377.584286	140	49.269670	-123.098900	
11	Fairview	372.017447	619	49.261960	-123.129440	
12	Renfrew-Collingwood	366.389093	926	49.246115	-123.043579	
13	South Cambie	330.737931	290	49.251465	-123.119484	
14	Kensington-Cedar Cottage	327.557568	1480	49.248760	-123.074640	
15	Kitsilano	322.464606	1828	49.266900	-123.161059	
16	Arbutus Ridge	318.828321	399	49.246340	-123.163950	
17	Mount Pleasant	316.049130	1610	49.263174	-123.100809	
18	Marpole	312.941294	649	49.212850	-123.129020	
19	West End	270.560163	1963	49.285250	-123.132040	
20	Downtown Eastside	244.335720	1313	49.281122	-123.100790	
21	Downtown	227.773900	6479	49.278650	-123.121720	
22	Grandview-Woodland	225.795700	1000	49.272607	-123.065406	

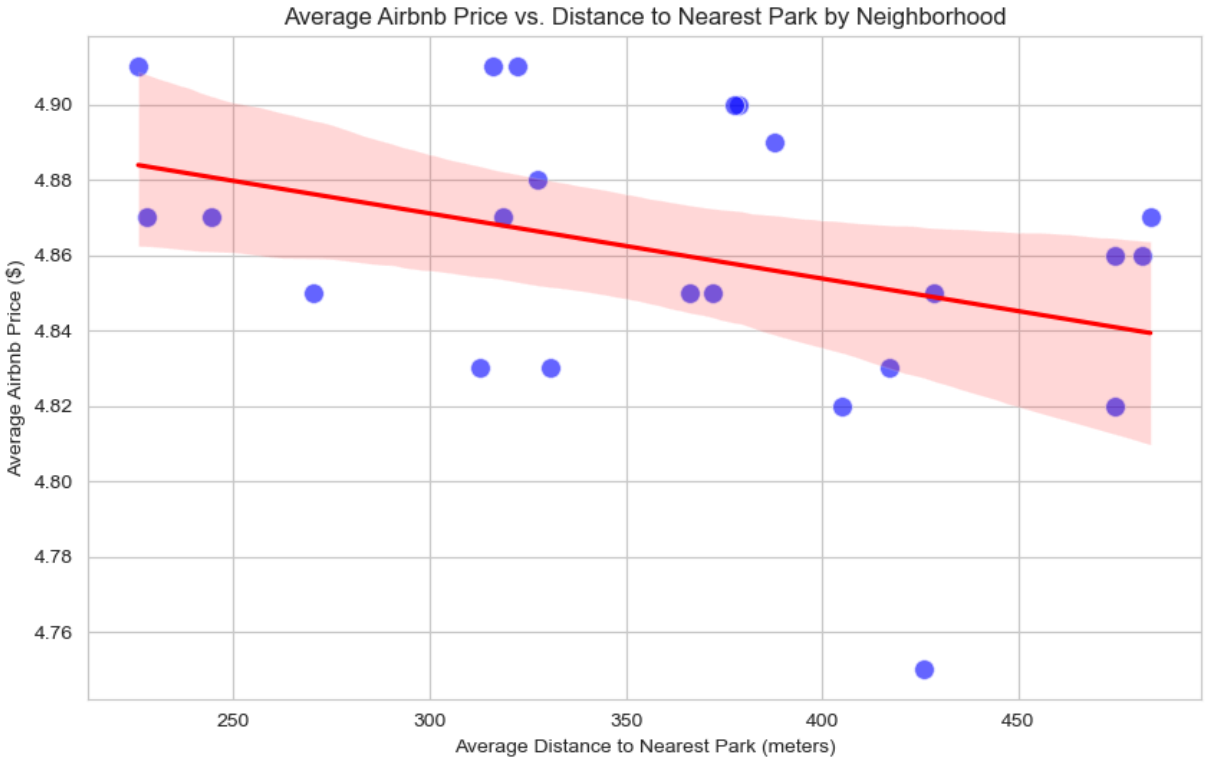
In [21]:

```
stat_by_neigh_dist['distance_to_closest_park'] = pd.to_numeric(stat_by_neigh_dist['
stat_by_neigh_dist['price'] = pd.to_numeric(stat_by_neigh_dist['price'], errors='co

# Create the scatter plot
```

```
plt.figure(figsize=(10, 6))
sns.scatterplot(x='distance_to_closest_park', y='price', data=stat_by_neigh_dist, s

#Adding a regression Line to visualize the correlation
sns.regplot(x='distance_to_closest_park', y='price', data=stat_by_neigh_dist, scatt

plt.title('Average Airbnb Price vs. Distance to Nearest Park by Neighborhood')
plt.xlabel('Average Distance to Nearest Park (meters)')
plt.ylabel('Average Airbnb Price ($)')
plt.grid(True)
plt.show()
```



In [23]:
```
stat_by_neigh_dist['distance_to_closest_park'] = pd.to_numeric(stat_by_neigh_dist['
stat_by_neigh_dist['review_scores_rating'] = pd.to_numeric(stat_by_neigh_dist['revi

# Create the scatter plot
plt.figure(figsize=(10, 6))
sns.scatterplot(x='distance_to_closest_park', y='review_scores_rating', data=stat_b

#Adding a regression Line to visualize the correlation
sns.regplot(x='distance_to_closest_park', y='review_scores_rating', data=stat_by_ne

plt.title('Average Airbnb Price vs. Distance to Nearest Park by Neighborhood')
plt.xlabel('Average Distance to Nearest Park (meters)')
plt.ylabel('Average Airbnb Price ($)')
plt.grid(True)
plt.show()
```



In [27]:
```
corr_price= stat_by_neigh_dist['distance_to_closest_park'].corr(stat_by_neigh_dist[
corr_rating= stat_by_neigh_dist['distance_to_closest_park'].corr(stat_by_neigh_dist
print(corr_price)
print(corr_rating)
```

```
-0.4866759593786387
-0.3657685751060855
```

In [24]:
```
"""This code block creates a map showing median price rating and distance to parks
Vancouver neighbourhoods.
"""

# Create map object.
map = folium.Map(location=[49.24445179910618, -123.11257056533111],
                 zoom_start=12,
                 tiles="cartodb positron"
                 )

# Add popup icons to map.
for index, row in stat_by_neigh_dist.iterrows():
    folium.Marker(
        [row['latitude'],
         row['longitude']
         ],
        popup=(
                row['neighbourhood'] + "\n"
                + "Rating: " + str(row['review_scores_rating'])
                + "\n" + "Price: $" + str(row['price']) + '\n' +"Park Distance :" +
        )
    ).add_to(map)
```

```
# Show map.
map
```

Out[24]: Make this Notebook Trusted to load map: File -> Trust Notebook