Lab 04: Data App

Author: Nigel Nelson

Introduction:

- This lab builds off of *Lab 1: Data Cleaning* by using the resulting cleaned data set as the primary data set for this lab. The data set, *CleanedSacramentorealestatetransactions*, has 919 entries and 14 variables describing real estate transactions in California. This includes the **street** address, the **city**, the **zip**, the **state**, the numbers of **beds**, the number of **baths**, the square footage of residential space(**sq_ft**), the **type** of real estate, the **sale date**, the sale **price**, the **latitude**, the **longitude**, whether it is an **empty_lot**, and the **street_type**. Specifically, this lab focuses on the creation of data apps. These data apps allow intuitive interactions with underlying data sets that enable non-technical users to analyze data on their own. The tool used in this lab to create data a data app is Bokeh, an open-source visualization library. In order to explore the capabilities of this visualization library, students are tasked with creating a method that makes a Bokeh figure with subtle interactivity. Next, a method that allows the filtering of real estate properties is created so that users can fine tune the properties displayed to them. Lastly, These methods are combined with supplied code to create an interactive figure with range sliders that allow user to filter the properties displayed to them in real time.

Imports:

In [1]:
```
import pandas as pd
import numpy as np
from bokeh.io import show, output_notebook, push_notebook, output_file
from bokeh.plotting import figure
from bokeh.models import ColumnDataSource, HoverTool, Column
from bokeh.palettes import all_palettes
from bokeh.models.widgets import CheckboxGroup, RangeSlider, DataTable, DateForma
from bokeh.layouts import column, row, WidgetBox
from bokeh.application.handlers import FunctionHandler
from bokeh.application import Application
output_notebook()
```

(https://bokeh.org)BokehJS 2.3.2 successfully loaded.

Importing the Real Estate Transactions Data Set:

In [2]:
```
re_transactions = pd.read_csv('CleanedSacramentorealestatetransactions.csv',
                              dtype={'city': 'category', 'zip': 'category',
                                     'state': 'category', 'type': 'category',
                                     'street_type': 'category'})
re_transactions.head()
```

Out[2]:

	street	city	zip	state	beds	baths	sq__ft	type	sale_date	price
0	3526 HIGH ST	SACRAMENTO	95838	CA	2	1	836	Residential	Wed May 21 00:00:00 EDT 2008	59222
1	51 OMAHA CT	SACRAMENTO	95823	CA	3	1	1167	Residential	Wed May 21 00:00:00 EDT 2008	68212
2	2796 BRANCH ST	SACRAMENTO	95815	CA	2	1	796	Residential	Wed May 21 00:00:00 EDT 2008	68880
3	2805 JANETTE WAY	SACRAMENTO	95815	CA	2	1	852	Residential	Wed May 21 00:00:00 EDT 2008	69307
4	6001 MCMAHON DR	SACRAMENTO	95824	CA	2	1	797	Residential	Wed May 21 00:00:00 EDT 2008	81900

Adding Color Attribute Correlated with Filtered Type Variable:

In [3]:
```
type_to_color = {'Residential': '#FF0000', 'Condo': '#0000FF', 'Multi-Family': '#
re_transactions['color'] = re_transactions['type'].map(type_to_color)
re_transactions['color'] = re_transactions['color'].astype('category')
re_transactions.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 919 entries, 0 to 918
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   street       919 non-null    object
 1   city         919 non-null    category
 2   zip          919 non-null    category
 3   state        919 non-null    category
 4   beds         919 non-null    int64
 5   baths        919 non-null    int64
 6   sq__ft       919 non-null    int64
 7   type         919 non-null    category
 8   sale_date    919 non-null    object
 9   price        919 non-null    int64
 10  latitude     919 non-null    float64
 11  longitude    919 non-null    float64
 12  empty_lot    919 non-null    bool
 13  street_type  919 non-null    category
 14  color        918 non-null    category
dtypes: bool(1), category(6), float64(2), int64(4), object(2)
memory usage: 68.8+ KB
```

Part 1: Display Real Estate on a Scatter Plot:

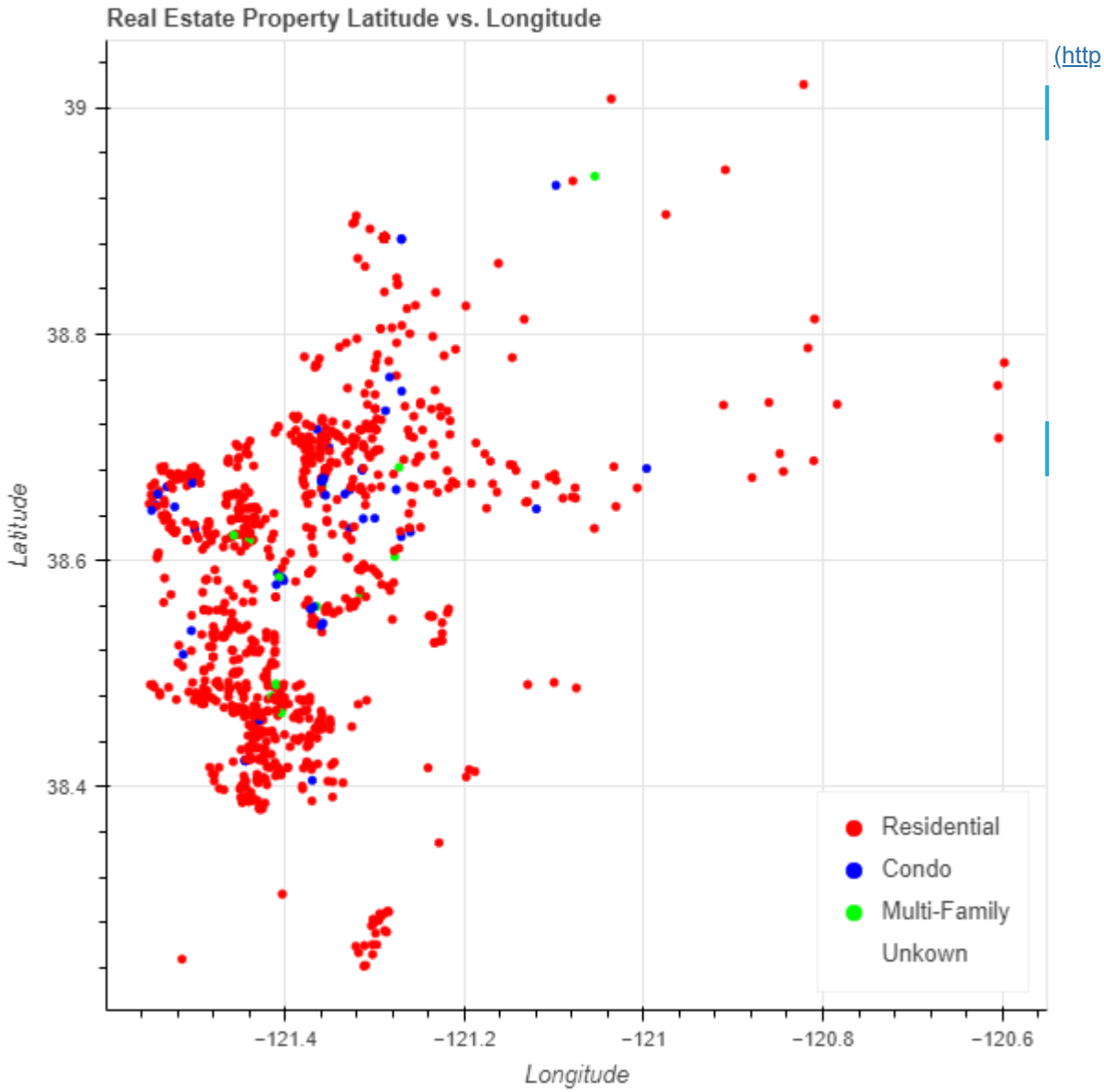
Creating Make_Plot method

- The make_plot method is responsible for taking a ColumnDataSource object, and returning a Bokeh figure object. This figure consists of a scatter plot of latitudes and longitudes that are colored according to the residence type. In addition, when the cursor hovers over a point in this figure, other attributes such as address, price, square footage, beds, and baths will be displayed to the user.

In [4]:
```
def make_plot(cds):
    tool_tips = [
        ("Adsress", "@street"),
        ("Price", "@price"),
        ("Square Feet", "@sq__ft"),
        ("Beds", "@beds"),
        ("Baths", "@baths"),
    ]
    fig = figure(title="Real Estate Property Latitude vs. Longitude", x_axis_labe
               y_axis_label='Latitude', tooltips=tool_tips)
    fig.circle(x = 'longitude', y = 'latitude', source = cds, color = 'color',
                size = 4, legend_field = 'type')
    fig.legend.location = 'bottom_right'
    return fig
```

Displaying Real Estate Transactions Figure:

In [5]:
```
cds = ColumnDataSource(re_transactions)
fig = make_plot(cds)
show(fig)
```



Part 2: Refine ColumnDataSource Object based on Search Criteria

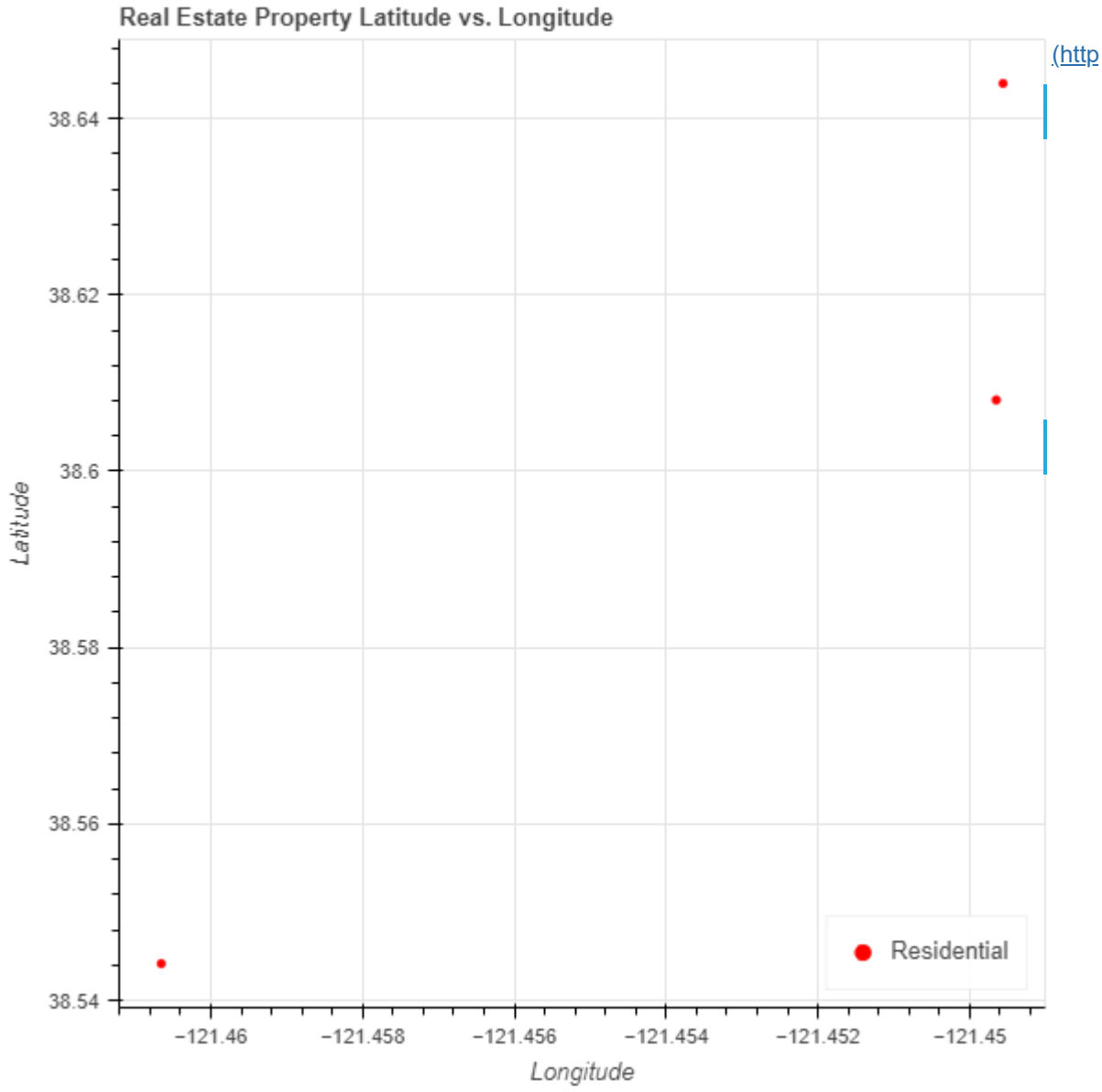
Creating make_dataset method:

- This methods accepts a DataFrame and a list with search criteria for real estate transaction, and return a new ColumnDataSource object that only contains properties that match the search criteria.`

In [6]:
```
def make_dataset(df, selected_type, price_range, baths_range, sq_ft_range, beds_r
    temp_df = df.copy()
    temp_df = temp_df[temp_df['type'].isin(selected_type)]
    temp_df = temp_df[temp_df['price'].between(price_range[0], price_range[1])]
    temp_df = temp_df[temp_df['sq__ft'].between(sq_ft_range[0], sq_ft_range[1])]
    temp_df = temp_df[temp_df['baths'].isin(list(range(baths_range[0], baths_rang
    temp_df = temp_df[temp_df['beds'].isin(list(range(beds_range[0], beds_range[1
    return ColumnDataSource(temp_df)
```

Using make_dataset to Display Filtered Properties:

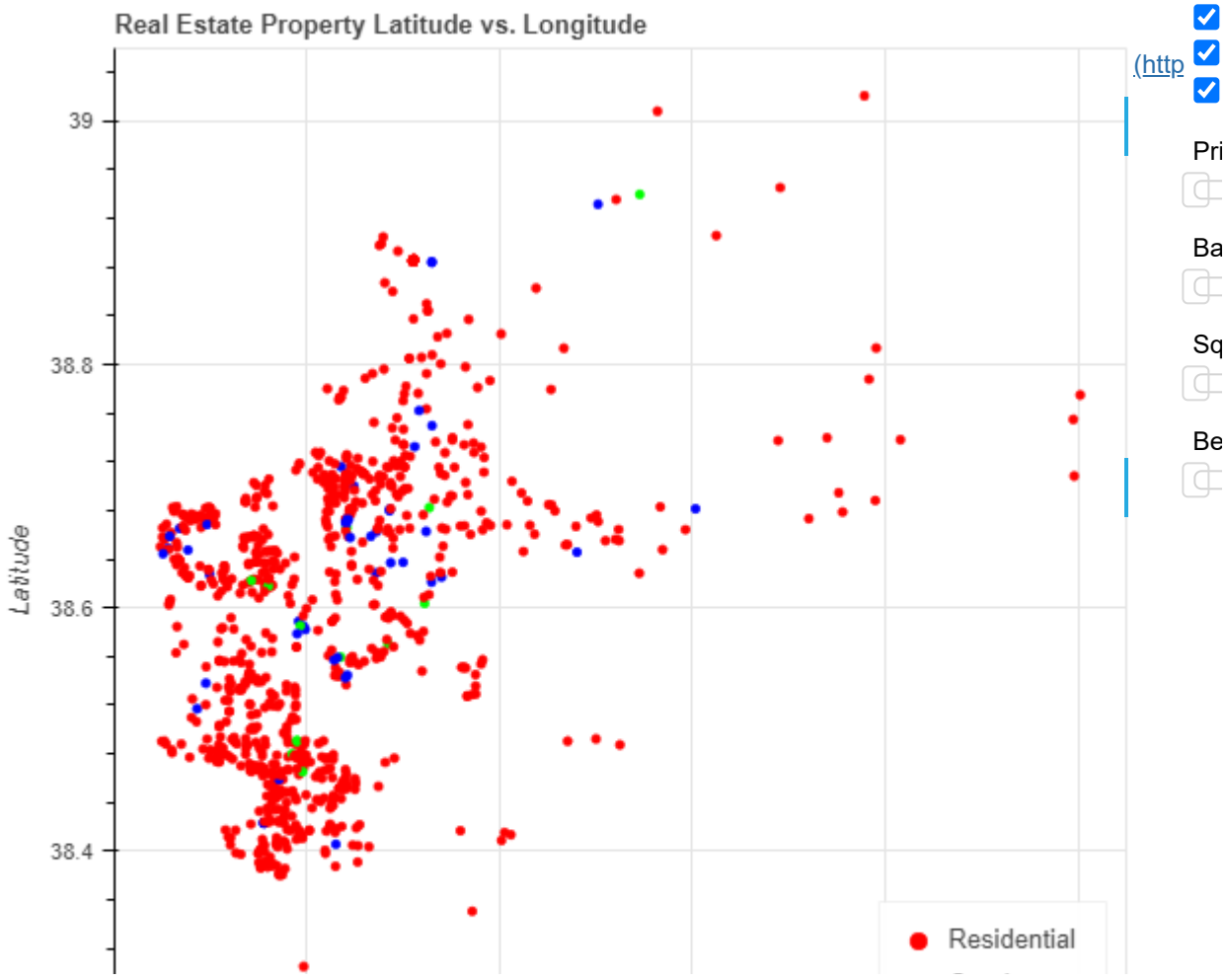
In [7]:
```
selected_type = ['Residential']
price_range = [50000, 75000]
baths_range = [1,2]
beds_range = [1,2]
sq_ft_range = [1000, 2000]

results = make_dataset(re_transactions, selected_type, price_range, baths_range,
fig = make_plot(results)
show(fig)
```

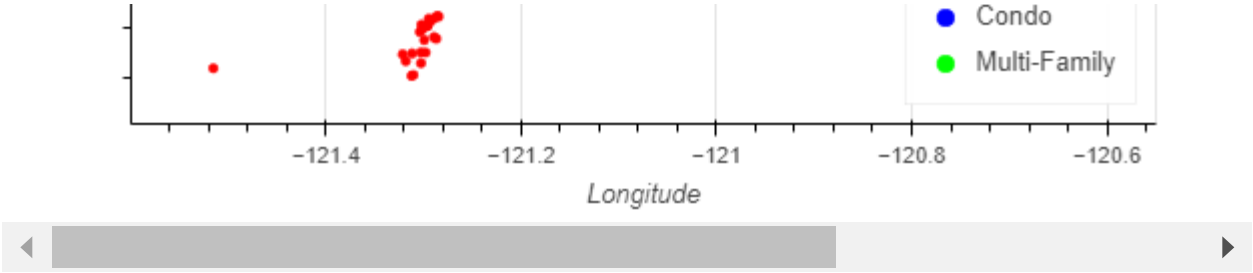


Part 3: Add Widgets and Create and Interactive Visualization

In [8]:
```
housing_selection = CheckboxGroup(labels=['Residential', 'Multi-Family', 'Condo']
range_slider_price = RangeSlider(start = re_transactions['price'].min(), end = re
                                 value = (re_transactions['price'].min(), re_tran
                                 step = 1000, title = 'Price Range')
range_slider_baths = RangeSlider(start = re_transactions['baths'].min(), end = re
                                 value = (re_transactions['baths'].min(), re_tran
                                 step = 1, title = 'Baths Range')
range_slider_sq_ft = RangeSlider(start = re_transactions['sq__ft'].min(), end = r
                                 value = (re_transactions['sq__ft'].min(), re_tra
                                 step = 100, title = 'Square Feet Range')
range_slider_beds = RangeSlider(start = re_transactions['beds'].min(), end = re_t
                                value = (re_transactions['beds'].min(), re_transa
                                step = 1, title = 'Beds Range')

controls = Column(housing_selection, range_slider_price, range_slider_baths, rang
source = make_dataset(re_transactions, selected_type, price_range, baths_range, b
figure_object = make_plot(source)
```

In [9]:
```
# Update function takes three default parameters
def update(attr, old, new):
    # Get the list of carriers for the graph
    selected_type = [housing_selection.labels[i] for i in housing_selection.activ
    price_range = [range_slider_price.value[0], range_slider_price.value[1]]
    baths_range = [range_slider_baths.value[0], range_slider_baths.value[1]]
    sq_ft_range = [range_slider_sq_ft.value[0], range_slider_sq_ft.value[1]]
    beds_range = [range_slider_beds.value[0], range_slider_beds.value[1]]
    # Make a new dataset based on the selected carriers and the
    # make_dataset function defined earlier
    new_src = make_dataset(re_transactions, selected_type, price_range, baths_ran
    # Update the source used in the quad glpyhs
    source.data.update(new_src.data)

def modify_doc(doc):

    housing_selection.on_change('active', update)
    range_slider_price.on_change('value', update)
    range_slider_baths.on_change('value', update)
    range_slider_beds.on_change('value', update)
    range_slider_sq_ft.on_change('value', update)

    doc.add_root(row(figure_object,column(controls)))
    #If you want to add A table to the visualization
    #doc.add_root(row(figure_object,column(controls)))

show(modify_doc)
```





Conclusion:

- This lab focused on the exploration of data apps. Specifically, the open source visualization library *Bokeh* was used in this lab to explore the creation and use of data apps. Students were tasked with defining methods that created interactive figures, as well as filtered the real estate transaction data set to filter properties by the range of values passed in by the user. Finally, these methods were combined with provided code to create a final interactive visualization that gives users the ability to filter properties using user-friendly input such as check boxes and range sliders. Through this process, the importance of reading documentation when using open source libraries was realized, as much more time was spent than needed attempting to use the Bokeh library based on analogies to other similar visualization tools. In addition, the importance of manipulating DataFrames using a copy of the original was realized as artifacts of manipulating in-place resulted in unexpected behavior that was difficult to debug. Overall, this lab explored the Bokeh library which enables the creation interactive visualizations, and also highlighted the importance of making data accessible to a wider audience of non-technical users.