Final_Code

April 7, 2024

1 Real Estate Pricing Analysis

[1]:
```
import pandas as pd
df = pd.read_csv("realtor-data.csv")
```

Data Dictionary

[2]:
```
# Add descriptions
description = {
    'status': 'Current standing of the home (for sale or ready to build)',
    'bed': 'Number of beds in the home',
    'bath': 'Number of baths in the home',
    'acre_lot': 'Size of the lot',
    'city': 'City where the home is located',
    'state': 'State where the home is located',
    'zip_code': 'Zip code of the home',
    'house_size': 'Square fottage of the home',
    'prev_sold_date': 'Date when the home was previously sold',
    'price': 'Current sale price or previously sold price if the house is not␣
 ↪for sale'
             }
# Initialize an empty dictionary to store data types
dtype_dict = {}

# Iterate through each column and store its data type in the dictionary
for col in df.columns:
    dtype_dict[col] = str(df[col].dtype)

series1 = pd.Series(description, name='description')
series1 = series1.rename_axis('column')
series2 = pd.Series(dtype_dict, name='data_type')
series2 = series2.rename_axis('column')



# Combining the Series into a DataFrame using pd.merge()
data_dictionary = pd.merge(series1, series2, left_index=True, right_index=True)
```

```
print('Data Dictionary\n')
print(data_dictionary.to_markdown())
```

```
Data Dictionary

| column         | description
| data_type   |
|:---------------|:-----------------------------------------------------------------
------------|:------------|
| status         | Current standing of the home (for sale or ready to build)
| object      |
| bed            | Number of beds in the home
| float64     |
| bath           | Number of baths in the home
| float64     |
| acre_lot       | Size of the lot
| float64     |
| city           | City where the home is located
| object      |
| state          | State where the home is located
| object      |
| zip_code       | Zip code of the home
| float64     |
| house_size     | Square fottage of the home
| float64     |
| prev_sold_date | Date when the home was previously sold
| object      |
| price          | Current sale price or previously sold price if the house is
not for sale | float64     |
```

Unique States

[3]:
```
unique_states_count = df['state'].nunique()

print("Number of unique states:", unique_states_count)

unique_states = df['state'].unique()

for state in unique_states:
    print(state)
```

```
Number of unique states: 25
Puerto Rico
Virgin Islands
Massachusetts
Connecticut
New Hampshire
Vermont
```

```
New Jersey
New York
South Carolina
Tennessee
Rhode Island
Virginia
Wyoming
Maine
Georgia
Pennsylvania
West Virginia
Delaware
Louisiana
Ohio
California
Colorado
Maryland
Missouri
District of Columbia
```

Data Cleansing

[4]:
```
df_cleaned = df.dropna()
```

[5]:
```
df_filtered = df[~df['state'].isin(['Puerto Rico', 'Virgin Islands'])]
```

[6]:
```
df_filtered = df_filtered.dropna()
```

[7]:
```
df_filtered['zip_code'] = df_filtered['zip_code'].astype(int).astype(str)
```

[8]:
```
df_filtered.tail()
```

[8]:

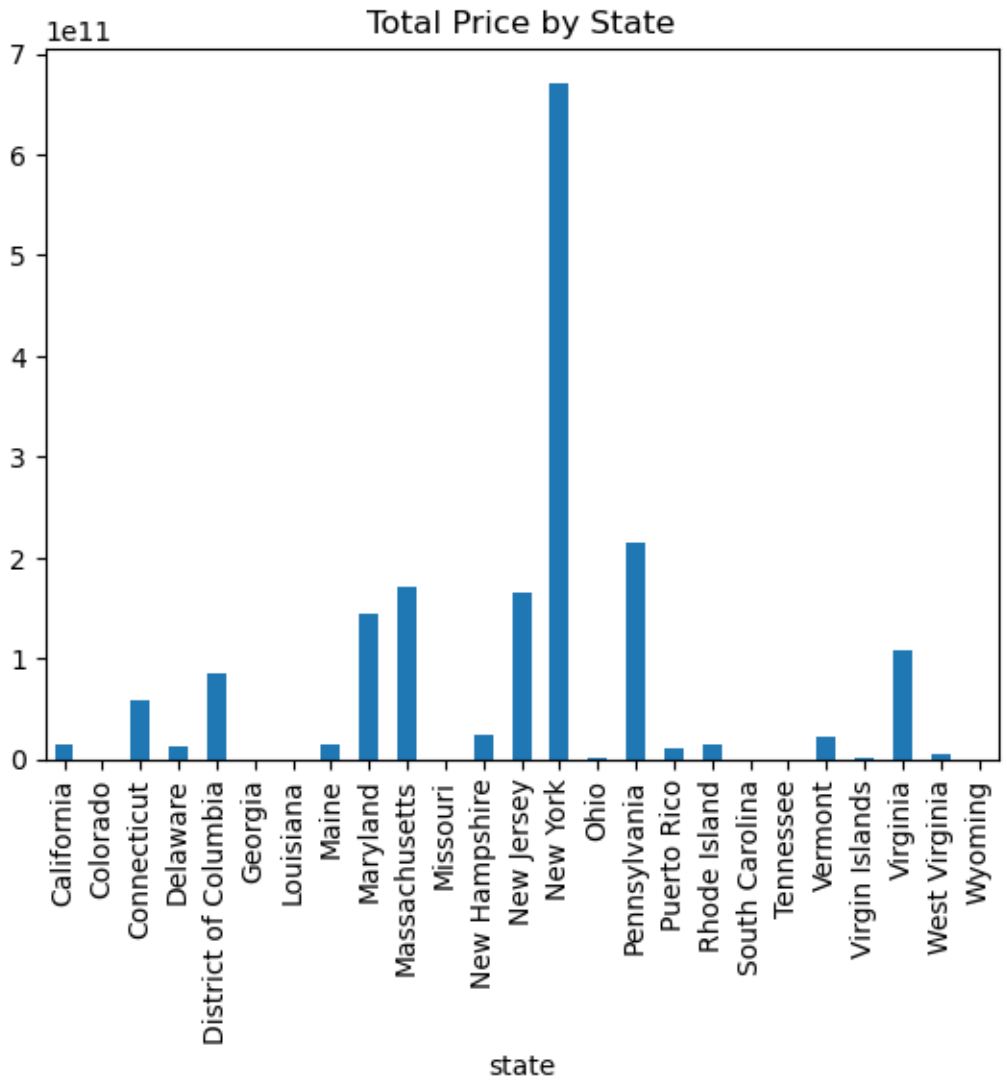
	status	bed	bath	acre_lot	city	state	zip_code	house_size	prev_sold_date	price
2701660	for_sale	3.0	3.0	0.40	Fredericksburg	Virginia	22405	2316.0	1999-06-01	439900.0
2701661	for_sale	4.0	3.0	0.06	Fredericksburg	Virginia	22407	2080.0	2004-04-22	319900.0
2701663	for_sale	3.0	3.0	0.50	Stafford	Virginia	22556	1747.0	2002-04-17	430000.0
2701664	for_sale	5.0	5.0	0.16	Stafford	Virginia	22554	4549.0	2017-01-30	744900.0
2701665	for_sale	4.0	3.0	0.27	Fredericksburg	Virginia	22407	2192.0	2020-04-28	425000.0

[9]:
```
import matplotlib.pyplot as plt
```

[10]:
```
state_set = df[['state', 'price']]
```

[11]:
```
state_agg = state_set.groupby('state')['price'].sum().rename('total price')
```

[12]:
```
state_agg.plot(kind='bar', x = 'state', y = 'total price')
plt.title('Total Price by State')
plt.show()
```



[13]:
```
subset_ny = df_filtered[df_filtered['state'] == 'New York']
```

[14]:
```
ny_summed_prices = df_filtered.groupby('zip_code')['price'].sum().reset_index()

ny_summed_prices.head()
```

[14]:

	zip_code	price
0	10001	213750000.0
1	10002	689000.0
2	10003	275915000.0
3	10004	100000000.0
4	10005	14795000.0

Create Geospatial Map of New York with Prices

[15]:
```
import pgeocode
# Store lat and lon values
latitude_values = []
longitude_values = []
# nomi that contains values for US
nomi = pgeocode.Nominatim('us')
# Iterate through zip codes in the data to join lat and lon info
for zip_code in ny_summed_prices['zip_code'].tolist():
    try:
        location = nomi.query_postal_code(zip_code)
        latitude = location.latitude
        longitude = location.longitude
        latitude_values.append(latitude)
        longitude_values.append(longitude)
    except:
        latitude_values.append(None)
        longitude_values.append(None)

# Appended values to ny subset
ny_summed_prices['Latitude'] = latitude_values
ny_summed_prices['Longitude'] = longitude_values
```

[16]:
```
# Drop any nullss
ny_summed_prices = ny_summed_prices.dropna()
from folium.plugins import HeatMap
import folium
# Create a map centered on New York
ny_map = folium.Map(location=[40.7128, -74.006], zoom_start=11)

# Create a HeatMap layer using house prices and coordinates
heat_data = [[row['Latitude'], row['Longitude'], row['price']] for idx, row in␣
 ↪ny_summed_prices.iterrows()]
HeatMap(heat_data).add_to(ny_map)

# Display the heat map
ny_map
```

[16]: <folium.folium.Map at 0x7f9ff3fcc580>

[17]:
```
import seaborn as sns
```

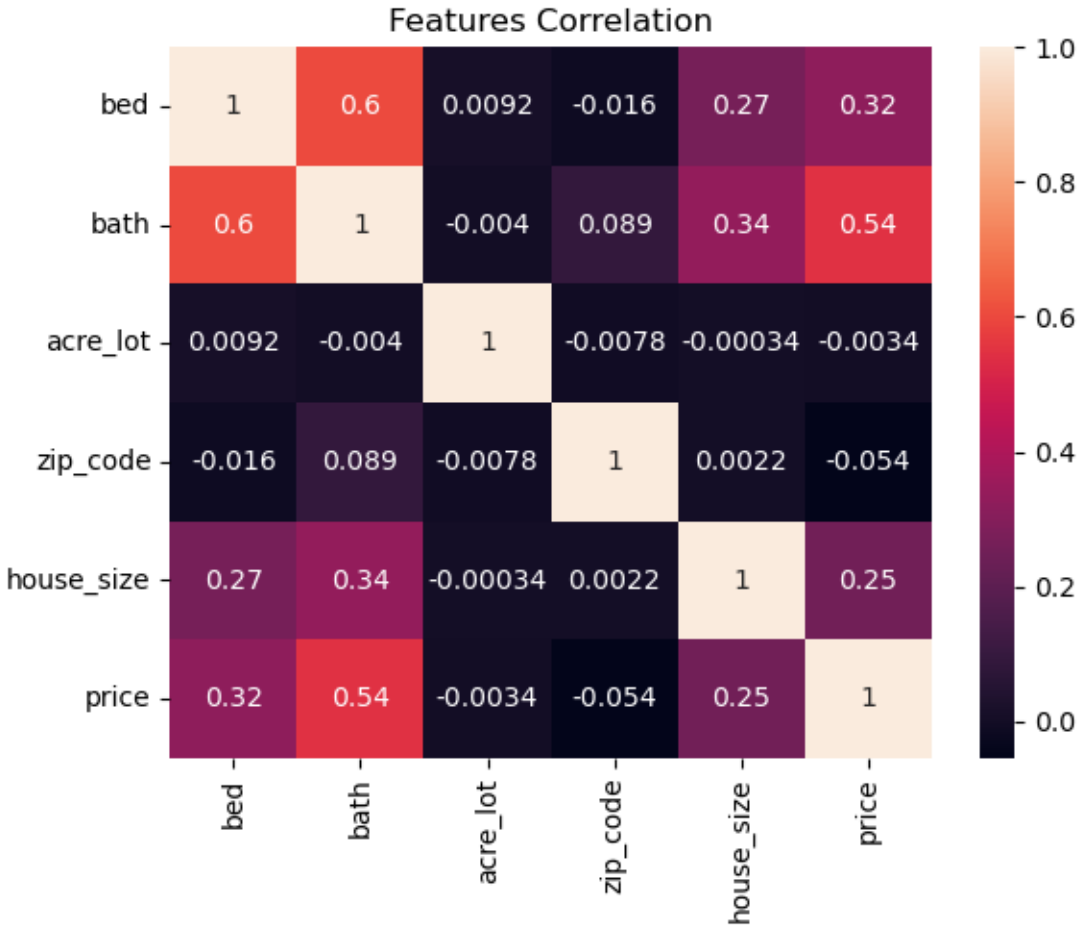
Correlation Matrix

[18]:
```
# Drop string values
df_num = df_filtered.drop(columns=['status', 'city', 'state', 'prev_sold_date'])
```

[19]:
```
# Create correlation matrix
corr =  df_num.corr()
```

[20]:
```
# Display correlation matrix
sns.heatmap(corr, annot=True)
plt.title('Features Correlation')
```

[20]:
```
Text(0.5, 1.0, 'Features Correlation')
```



Model #1 Linear Regression with zipcode

[33]:
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
```

[22]:
```
# Create ny subset without lat and lon
ny_summed_prices = df_filtered.groupby('zip_code')['price'].sum().reset_index()

ny_summed_prices.head()
```

[22]:

	zip_code	price
0	10001	213750000.0
1	10002	689000.0
2	10003	275915000.0
3	10004	100000000.0
4	10005	14795000.0

[23]:
```
# Separate the target from the features
feature = ny_summed_prices.drop('price', axis=1)
target = ny_summed_prices['price']

#Split the data into training and test
X_train, X_test, y_train, y_test = train_test_split(feature, target,␣
 ↪test_size=0.2, random_state=42)
```

[34]:
```
# Create and fit the linear regression model using the training data
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test data
predictions = model.predict(X_test)


print("Model 1 Results")
# print("Predictions on test data:", predictions)
# Calculate Mean Squared Error
mse = mean_squared_error(y_test, predictions)
print("Mean Squared Error:", mse)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
print("Mean Absolute Error:", mae)

# Calculate R-squared score
r2 = r2_score(y_test, predictions)
print("R-squared score:", r2)
```

```
Model 1 Results
```

```
Mean Squared Error: 1341708654258.3293
Mean Absolute Error: 436656.6665262822
R-squared score: 0.3201483824250393

/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/sklearn/utils/validation.py:623: FutureWarning: is_sparse is deprecated
and will be removed in a future version. Check `isinstance(dtype,
pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/sklearn/utils/validation.py:623: FutureWarning: is_sparse is deprecated
and will be removed in a future version. Check `isinstance(dtype,
pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
```

Model #2 Using Numerical Fields in the dataset

[25]:
```
# Create subset without string values
test_ny = subset_ny.drop(columns=['status', 'city', 'state', 'prev_sold_date'])
```

[26]:
```
test_ny.head()
```

[26]:

	bed	bath	acre_lot	zip_code	house_size	price
54248	3.0	2.0	2.02	12521	1600.0	425000.0
54258	4.0	2.0	0.24	12521	1239.0	225000.0
54267	4.0	1.0	4.20	12516	1500.0	299999.0
54268	3.0	2.0	2.90	12529	1404.0	374900.0
54278	3.0	2.0	1.20	12546	1350.0	375000.0

[27]:
```
# Separate the target from the features
x = test_ny.drop('price', axis=1)
y = test_ny['price']

#Split the data into training and test
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
 ↪random_state=42)
```

[35]:
```
# Create and fit the linear regression model using the training data
model = LinearRegression()
model.fit(X_train, y_train)

# Make predictions on the test data
predictions = model.predict(X_test)

# print("Predictions on test data:", predictions)
print("Model 2 Results")
# Calculate Mean Squared Error
mse = mean_squared_error(y_test, predictions)
```

```
print("Mean Squared Error:", mse)

# Calculate Mean Absolute Error
mae = mean_absolute_error(y_test, predictions)
print("Mean Absolute Error:", mae)

# Calculate R-squared score
r2 = r2_score(y_test, predictions)
print("R-squared score:", r2)
```

```
Model 2 Results
Mean Squared Error: 1341708654258.3293
Mean Absolute Error: 436656.6665262822
R-squared score: 0.3201483824250393
```

```
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/sklearn/utils/validation.py:623: FutureWarning: is_sparse is deprecated
and will be removed in a future version. Check `isinstance(dtype,
pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/sklearn/utils/validation.py:623: FutureWarning: is_sparse is deprecated
and will be removed in a future version. Check `isinstance(dtype,
pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
```

Model #3 Using Random Forest Regression

[30]:
```
from sklearn.ensemble import RandomForestRegressor

# Create and fit the Random Forest Regressor model using the training data
rf_model = RandomForestRegressor()
rf_model.fit(X_train, y_train)

# Make predictions on the test data
rf_predictions = rf_model.predict(X_test)

# Calculate Mean Squared Error
rf_mse = mean_squared_error(y_test, rf_predictions)

# Calculate Mean Absolute Error
rf_mae = mean_absolute_error(y_test, rf_predictions)

# Calculate R-squared score
rf_r2 = r2_score(y_test, rf_predictions)

print("Model 3 Results")
print("Random Forest Regressor - Mean Squared Error:", rf_mse)
```

```
print("Random Forest Regressor - Mean Absolute Error:", rf_mae)
print("Random Forest Regressor - R-squared score:", rf_r2)
```

```
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/sklearn/utils/validation.py:623: FutureWarning: is_sparse is deprecated
and will be removed in a future version. Check `isinstance(dtype,
pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
/Users/feliperodriguez/opt/anaconda3/lib/python3.9/site-
packages/sklearn/utils/validation.py:623: FutureWarning: is_sparse is deprecated
and will be removed in a future version. Check `isinstance(dtype,
pd.SparseDtype)` instead.
  if not hasattr(array, "sparse") and array.dtypes.apply(is_sparse).any():
```

```
Model 3 Results
Random Forest Regressor - Mean Squared Error: 24709493631.44506
Random Forest Regressor - Mean Absolute Error: 8271.197990803117
Random Forest Regressor - R-squared score: 0.9874795551467302
```

[ ]: