In [28]:

```
import pandas as pd
df = pd.read_csv('NYC_Property_Sales_Data_Geocoded.csv', low_memory=False)

missing_percentages = (df.isnull().sum() / len(df)) * 100
columns_to_drop = missing_percentages[missing_percentages > 70].index
df.drop(columns=columns_to_drop, inplace=True)
print(columns_to_drop)
df.shape

threshold = 0.5 * len(df.columns)
missing_values_per_row = df.isnull().sum(axis=1)
rows_to_drop = df[missing_values_per_row > threshold]
print("Number of rows with more than 50% missing values:", len(rows_to_drc
df.drop(index=rows_to_drop.index, inplace=True)
print("Shape of the DataFrame after dropping rows:", df.shape)

df['YEAR BUILT'].replace('0', pd.NaT, inplace=True)
df['YEAR BUILT'] = pd.to_datetime(df['YEAR BUILT'], format='%Y', errors='c
df.dropna(subset=['YEAR BUILT'], inplace=True)

def clean_square_feet(value):
    try:
        # Check if the value is not null and is a string
        if pd.notnull(value) and isinstance(value, str):
            # Remove commas and spaces
            cleaned_value = value.replace(',', '').replace(' ', '')
            # Convert to float
            return float(cleaned_value)
        else:
            # Return None for non-string or NaN values
            return None
    except ValueError:
        # Handle exception for strings like '- 0'
        return None  # or any other appropriate action

# Apply the cleaning function to 'LAND SQUARE FEET' column
df['LAND SQUARE FEET'] = df['LAND SQUARE FEET'].apply(clean_square_feet)
df['LAND SQUARE FEET'] = df['LAND SQUARE FEET'].astype(float)

def clean_square_feet(value):
    try:
        # Check if the value is not null and is a string
        if pd.notnull(value) and isinstance(value, str):
            # Remove commas and spaces
            cleaned_value = value.replace(',', '').replace(' ', '')
            # Convert to float
            return float(cleaned_value)
        else:
            # Return None for non-string or NaN values
            return None
    except ValueError:
        # Handle exception for strings like '- 0'
        return None  # or any other appropriate action

# Apply the cleaning function to 'LAND SQUARE FEET' column
df['GROSS SQUARE FEET'] = df['GROSS SQUARE FEET'].apply(clean_square_feet)
df['GROSS SQUARE FEET'] = df['GROSS SQUARE FEET'].astype(float)
```

```

df['LAND SQUARE FEET'] = df['LAND SQUARE FEET'].replace(-0.0, 0.0)
df['GROSS SQUARE FEET'] = df['GROSS SQUARE FEET'].replace(-0.0, 0.0)
df['LAND SQUARE FEET'].fillna(0.0, inplace=True)
df['GROSS SQUARE FEET'].fillna(0.0, inplace=True)

df['COMMERCIAL UNITS'] = df['COMMERCIAL UNITS'].fillna(0)
df['RESIDENTIAL UNITS'] = df['RESIDENTIAL UNITS'].fillna(0)

df['NTA'].fillna('Unknown', inplace=True)
df.drop(columns=['BIN', 'BBL'], inplace=True)
df.drop(columns=['Census Tract', 'BOROUGH', 'Council District', 'Community

df['TAX CLASS AS OF FINAL ROLL'] = df.groupby('YEAR BUILT')['TAX CLASS AS 
df['BUILDING CLASS AS OF FINAL ROLL'] = df.groupby('YEAR BUILT')['BUILDING

# Ensure 'Latitude' and 'Longitude' are in numeric form (if they're not al
df['Latitude'] = pd.to_numeric(df['Latitude'], errors='coerce')
df['Longitude'] = pd.to_numeric(df['Longitude'], errors='coerce')

# Group by 'BLOCK' (or another geographical marker) and interpolate withi
df['Latitude'] = df.groupby('BLOCK')['Latitude'].transform(lambda x: x.int
df['Longitude'] = df.groupby('BLOCK')['Longitude'].transform(lambda x: x.i
df['Latitude'] = df.groupby('NEIGHBORHOOD')['Latitude'].transform(lambda x
df['Longitude'] = df.groupby('NEIGHBORHOOD')['Longitude'].transform(lambda

df['TOTAL UNITS'] = df['RESIDENTIAL UNITS'] + df['COMMERCIAL UNITS']
df['ZIP CODE'] = df.groupby('NEIGHBORHOOD')['ZIP CODE'].transform(lambda x
df['SALE DATE'] = pd.to_datetime(df['SALE DATE'])
df['BUILDING CLASS CATEGORY'] = df['BUILDING CLASS CATEGORY'].str.replace(
df['BUILDING CLASS CATEGORY'] = df['BUILDING CLASS CATEGORY'].str.title()

df.isnull().sum()
```

```
Index(['EASE-MENT', 'APARTMENT NUMBER', 'Census Tract 2020', 'NTA Code'], dty
pe='object')
Number of rows with more than 50% missing values: 18
Shape of the DataFrame after dropping rows: (606242, 27)
```

Out[28]:

```
NEIGHBORHOOD                         0
BUILDING CLASS CATEGORY              0
TAX CLASS AS OF FINAL ROLL           0
BLOCK                                0
LOT                                  0
BUILDING CLASS AS OF FINAL ROLL      0
ADDRESS                              0
ZIP CODE                             0
RESIDENTIAL UNITS                    0
COMMERCIAL UNITS                     0
TOTAL UNITS                          0
LAND SQUARE FEET                     0
GROSS SQUARE FEET                    0
YEAR BUILT                           0
TAX CLASS AT TIME OF SALE            0
BUILDING CLASS AT TIME OF SALE       0
SALE PRICE                           0
SALE DATE                            0
Latitude                             0
Longitude                            0
NTA                                  0
dtype: int64
```

In [29]:

```
#1
import streamlit as st

# Set the title of your Streamlit app
st.title("NYC Property Sales Data Exploration")

# Create a sidebar for the date range filter
st.sidebar.title("Filters")

# Get the minimum and maximum sale dates from your dataset for the date in
min_date = df['SALE DATE'].min()
max_date = df['SALE DATE'].max()

# Use Streamlit's date_input widget to get a date range from the user
date_range = st.sidebar.date_input("Sale Date Range", value=(min_date, max

# Filter the DataFrame based on the selected date range
filtered_df = df[(df['SALE DATE'] >= pd.to_datetime(date_range[0])) & (df[

# Select only the specified columns for display
columns_to_display = ['NEIGHBORHOOD', 'BLOCK', 'ADDRESS', 'ZIP CODE', 'YEA
filtered_df = filtered_df[columns_to_display]

# Display the filtered DataFrame
st.dataframe(filtered_df)

# Show the number of sales in the selected period
st.write(f"Total sales in the selected period: {len(filtered_df)}")
```

In [30]:

```
#2
import streamlit as st
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming your DataFrame is named 'df'


# Filter out rows where 'SALE PRICE' is 0 or NaN, as these do not contribu
df = df[(df['SALE PRICE'] > 0) & (df['SALE PRICE'].notnull())]

# Set the title of your Streamlit app
st.title("Sales Distribution by Neighborhood")

# Create a selectbox for choosing a Building Class Category
selected_building_class = st.selectbox("Select Building Class Category:",

# Filter the DataFrame based on the selected building class
filtered_df = df[df['BUILDING CLASS CATEGORY'] == selected_building_class]

# Create the box plot
plt.figure(figsize=(12, 8))  # Adjust the figure size as needed
sns.boxplot(
    x='SALE PRICE',
    y='NEIGHBORHOOD',
    data=filtered_df,
    palette='coolwarm'  # Use a colorful palette. Other options: 'vibrant'
)
plt.xticks(rotation=45)  # Rotate x-axis labels for better readability
plt.title("Sales Distribution by Neighborhood for " + selected_building_cl
plt.xlabel("Sale Price")
plt.ylabel("Neighborhood")

# Optional: Add thousands separator for x-axis labels for better readabili
from matplotlib.ticker import StrMethodFormatter
plt.gca().xaxis.set_major_formatter(StrMethodFormatter('{x:,.0f}'))

# Display the plot in Streamlit
st.pyplot(plt)
```

Out[30]:

```
DeltaGenerator()
```

In [31]:

```
#3
import streamlit as st
import pandas as pd
import numpy as np
import altair as alt


# Extract year from 'SALE DATE' column
df['YEAR'] = df['SALE DATE'].dt.year

# Set title
st.title('NYC Property Sales Time Series Analysis')

# Sidebar filters
st.sidebar.header('Filters for Sales Time Series Analysis')
min_date = min(df['SALE DATE']).date()
max_date = max(df['SALE DATE']).date()
start_date = st.sidebar.date_input('Start Date', min_value=min_date, max_v
end_date = st.sidebar.date_input('End Date', min_value=min_date, max_value

# Convert start_date and end_date to datetime
start_date = pd.to_datetime(start_date)
end_date = pd.to_datetime(end_date)

# Filter data based on selected dates
filtered_data = df[(df['SALE DATE'] >= start_date) & (df['SALE DATE'] <= e

# Group data by YEAR and calculate total sales count
time_series_data = filtered_data.groupby('YEAR').size().reset_index(name='

# Plot time series
chart = alt.Chart(time_series_data).mark_line().encode(
    x='YEAR:O',  # Using ordinal scale for discrete years
    y='Total Sales'
).properties(
    width=800,
    height=500
).interactive()

st.altair_chart(chart, use_container_width=True)

# Summary statistics
total_sales = filtered_data.shape[0]
average_price = filtered_data['SALE PRICE'].mean()
median_price = filtered_data['SALE PRICE'].median()

st.subheader('Summary Statistics')
st.write(f'Total Sales: {total_sales}')
st.write(f'Average Sale Price: ${average_price:,.2f}')
st.write(f'Median Sale Price: ${median_price:,.2f}')
```

In [32]:

```
#4
import streamlit as st
import pandas as pd
import altair as alt

st.sidebar.header('Filter for Year Built')


# Filter by Year Built (Building Age)
year_built_range = st.sidebar.slider("Select Year Built Range:", 1798, 202


# Convert 'YEAR BUILT' column to integer type
df['YEAR BUILT'] = pd.to_numeric(df['YEAR BUILT'], errors='coerce')

# Apply filters to the DataFrame
filtered_df = df[(df['YEAR BUILT'] >= year_built_range[0]) &
                 (df['YEAR BUILT'] <= year_built_range[1])]

# Display histogram for Building Age
histogram_chart = alt.Chart(filtered_df).mark_bar().encode(
    x=alt.X('YEAR BUILT', title='Year Built'),
    y=alt.Y('count()', title='Number of Sales'),
    tooltip=['YEAR BUILT', 'count()']
).properties(
    width=600,
    height=400
).interactive()

st.subheader("Distribution of Property Sales by Year Built")
st.altair_chart(histogram_chart)
```

Out[32]:

```
DeltaGenerator()
```

In [33]:

```
#5
import streamlit as st
import pandas as pd
import altair as alt


# Set the title of your Streamlit app
st.subheader("Property Characteristics vs. Sale Price Analysis")

# Create filter widgets
selected_property_characteristic = st.selectbox("Select Property Character
                                                ['LAND SQUARE FEET', 'TOTA
selected_neighborhood = st.selectbox("Select Neighborhood:", df['NEIGHBORH
selected_tax_class = st.selectbox("Select Tax Class:", df['TAX CLASS AS OF

# Apply filters to the DataFrame
filtered_df = df[(df['NEIGHBORHOOD'] == selected_neighborhood) &
                 (df['TAX CLASS AS OF FINAL ROLL'] == selected_tax_class)]

# Create scatter plot
scatter_plot = alt.Chart(filtered_df).mark_circle().encode(
    x=selected_property_characteristic,
    y='SALE PRICE',
    tooltip=['ADDRESS', 'SALE PRICE', 'BUILDING CLASS CATEGORY']
).properties(
    width=800,
    height=500
).interactive()

# Display scatter plot
st.subheader("Property Characteristics vs. Sale Price")
st.altair_chart(scatter_plot)

# Summary statistics
average_price = filtered_df['SALE PRICE'].mean()
median_price = filtered_df['SALE PRICE'].median()
total_sales = len(filtered_df)

st.subheader("Summary Statistics")
st.write(f"Average Sale Price: ${average_price:,.2f}")
st.write(f"Median Sale Price: ${median_price:,.2f}")
st.write(f"Total Sales: {total_sales}")
```

In [34]:

```
#6

import streamlit as st
import pandas as pd
import geopandas as gpd
import plotly.express as px
import altair as alt
from shapely.geometry import Point

# Create Streamlit app
st.title('NYC Property Sales by Neighborhood')

# Filter for Date Range
start_date = df['SALE DATE'].min()
end_date = df['SALE DATE'].max()

# Filter for Property Type
property_type = st.selectbox('Property Type', df['BUILDING CLASS CATEGORY'

# Filter data based on selected filters
filtered_df = df[(df['SALE DATE'] >= start_date) & (df['SALE DATE'] <= enc
                 (df['BUILDING CLASS CATEGORY'] == property_type)]

# Aggregate data by neighborhood
neighborhood_sales = filtered_df.groupby('NEIGHBORHOOD').size().reset_inde

# Create choropleth map using Plotly
fig = px.choropleth_mapbox(neighborhood_sales,
                           locations='NEIGHBORHOOD',
                           geojson="https://raw.githubusercontent.com/dwil
                           color='Total Sales',
                           color_continuous_scale="Viridis",
                           range_color=(0, neighborhood_sales['Total Sales
                           mapbox_style="carto-positron",
                           zoom=10, center={"lat": 40.7128, "lon": -74.006
                           opacity=0.5,
                           labels={'Total Sales': 'Total Sales'}
                          )

fig.update_layout(margin={"r":0,"t":0,"l":0,"b":0})

# Display choropleth map
st.plotly_chart(fig)
```

Out[34]:

```
DeltaGenerator()
```

In [35]:

```
#7
import streamlit as st
import pandas as pd
import plotly.express as px

# Create Streamlit app
st.title('Property Characteristics vs. Sale Price Analysis')

# Filter widgets
property_type = st.selectbox('Select Property Type', df['BUILDING CLASS CA
neighborhood = st.selectbox('Select Neighborhood', df['NEIGHBORHOOD'].unic
start_date = st.date_input('Start Date', min_value=pd.to_datetime(df['SALE
end_date = st.date_input('End Date', min_value=pd.to_datetime(df['SALE DAT

# Convert start_date and end_date to datetime objects
start_date = pd.to_datetime(start_date)
end_date = pd.to_datetime(end_date)

# Filter data based on selected filters
filtered_df = df[(df['BUILDING CLASS CATEGORY'] == property_type) &
                 (df['NEIGHBORHOOD'] == neighborhood) &
                 (pd.to_datetime(df['SALE DATE']) >= start_date) &
                 (pd.to_datetime(df['SALE DATE']) <= end_date)]

# Plotly scatter plot
fig = px.scatter(filtered_df, x='TOTAL UNITS', y='SALE PRICE',
                 hover_data=['ADDRESS', 'LAND SQUARE FEET', 'GROSS SQUARE
                 trendline='ols', title='Total Units vs. Sale Price',
                 labels={'TOTAL UNITS': 'Total Units', 'SALE PRICE': 'Sale

# Customize layout
fig.update_layout(showlegend=True)

# Display plot
st.plotly_chart(fig)

# Summary statistics
average_price = filtered_df['SALE PRICE'].mean()
median_price = filtered_df['SALE PRICE'].median()
total_sales = len(filtered_df)

st.subheader("Summary Statistics")
st.write(f"Average Sale Price: ${average_price:,.2f}")
st.write(f"Median Sale Price: ${median_price:,.2f}")
st.write(f"Total Sales: {total_sales}")
```

In [36]:

```python
#8
import streamlit as st
import pandas as pd
import plotly.express as px

# Calculate the total number of residential and commercial units
total_residential_units = df['RESIDENTIAL UNITS'].sum()
total_commercial_units = df['COMMERCIAL UNITS'].sum()

# Create a DataFrame for the pie chart
data = pd.DataFrame({
    'Unit Type': ['Residential Units', 'Commercial Units'],
    'Total Units': [total_residential_units, total_commercial_units]
})

# Create an interactive pie chart using Plotly Express
fig = px.pie(data, values='Total Units', names='Unit Type',
             title='Unit Type Distribution',
             hover_name='Unit Type',
             labels={'Unit Type': 'Unit Type'},
             hole=0.3)

# Add labels to the pie chart sectors
fig.update_traces(textinfo='percent+label')

# Display the pie chart
st.plotly_chart(fig)
```

Out[36]:

```
DeltaGenerator()
```

In [37]:

```
#9
import streamlit as st
import pandas as pd
import altair as alt

# Filter for Date Range
start_date = df['SALE DATE'].min()
end_date = df['SALE DATE'].max()

# Filter data based on selected filters
filtered_df = df[(df['SALE DATE'] >= start_date) & (df['SALE DATE'] <= enc

# Create a bar chart for distribution of sales by tax class
tax_class_counts = filtered_df['TAX CLASS AS OF FINAL ROLL'].value_counts(
tax_class_counts.columns = ['Tax Class', 'Number of Sales']

# Plotting with Altair
bar_chart = alt.Chart(tax_class_counts).mark_bar().encode(
    x=alt.X('Tax Class:O', title='Tax Class'),
    y=alt.Y('Number of Sales:Q', title='Number of Sales'),
    tooltip=['Tax Class', 'Number of Sales']
).properties(
    width=600,
    height=400
).interactive()

# Add chart title and labels
bar_chart = bar_chart.properties(
    title="Distribution of Sales by Tax Class"
).configure_axis(
    labelFontSize=12,
    titleFontSize=14
).configure_title(
    fontSize=16,
    anchor='middle'
)

# Display the chart
st.altair_chart(bar_chart, use_container_width=True)

```

Out[37]:

```
DeltaGenerator()
```

In [38]:

```
#10
# Group data by neighborhood and sum the residential and commercial units
units_by_neighborhood = df.groupby('NEIGHBORHOOD')[['RESIDENTIAL UNITS', '

# Melt the DataFrame to long format for easier plotting
units_by_neighborhood_melted = units_by_neighborhood.melt(id_vars='NEIGHBO

# Plot stacked bar chart
bar_chart = alt.Chart(units_by_neighborhood_melted).mark_bar().encode(
    x='NEIGHBORHOOD:N',
    y='Total Units:Q',
    color='Unit Type:N',
    tooltip=['NEIGHBORHOOD', 'Total Units', 'Unit Type']
).properties(
    width=800,
    height=500
).interactive()

# Add chart title and labels
bar_chart = bar_chart.properties(
    title="Number of Residential and Commercial Units by Neighborhood"
).configure_axis(
    labelFontSize=12,
    titleFontSize=14
).configure_title(
    fontSize=16,
    anchor='middle'
)

# Display the chart
st.write(bar_chart)
```

In [ ]:

```
 
```