# GROUP PROJECT-4

## Project Report: Development of Comprehensive Dashboards Using Web Frameworks

## Course: IE6600 Computation and Visualization

## Spring Semester 2024

Group Number - 10

Sri Sai Prabhath Reddy Gudipalli (002207631)

Deepthi Umesha (002661626)

Ashwini Mahadevaswamy (002661627)

Yashwant Dontam (002844794)

# Introduction

This project report explores and analyzes New York City property sales data from January 2016 to December 2022. Our goal is to reveal trends, distributions, and anomalies in real estate transactions across different neighborhoods to aid stakeholders in making informed decisions. We utilize advanced data manipulation and visualization techniques, including geocoding, data cleaning, and interactive visualizations through tools like Pandas and Streamlit. The insights derived from this analysis provide a comprehensive overview of the NYC real estate market, highlighting active areas and identifying sales patterns related to property characteristics.

# Task: Data Acquisition and Inspection

```python
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)

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_drop))
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='coerce')
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
```

```python
# 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 Board'], inplace=True)
df['TAX CLASS AS OF FINAL ROLL'] = df.groupby('YEAR BUILT')['TAX CLASS AS OF FINAL ROLL'].transform(lambda x: x.fillna(x.mode()[0] if not x.mode().empty else "Unknown"))
df['BUILDING CLASS AS OF FINAL ROLL'] = df.groupby('YEAR BUILT')['BUILDING CLASS AS OF FINAL ROLL'].transform(lambda x: x.fillna(x.mode()[0] if not x.mode().empty else "Unknown"))

# Ensure 'Latitude' and 'Longitude' are in numeric form (if they're not already)
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 within each group
df['Latitude'] = df.groupby('BLOCK')['Latitude'].transform(lambda x: x.interpolate(method='linear', limit_direction='both'))
df['Longitude'] = df.groupby('BLOCK')['Longitude'].transform(lambda x: x.interpolate(method='linear', limit_direction='both'))
df['Latitude'] = df.groupby('NEIGHBORHOOD')['Latitude'].transform(lambda x: x.fillna(x.mean()))
df['Longitude'] = df.groupby('NEIGHBORHOOD')['Longitude'].transform(lambda x: x.fillna(x.mean()))
df['TOTAL UNITS'] = df['RESIDENTIAL UNITS'] + df['COMMERCIAL UNITS']
df['ZIP CODE'] = df.groupby('NEIGHBORHOOD')['ZIP CODE'].transform(lambda x: x.fillna(x.mode()[0] if not x.mode().empty else np.nan))
df['SALE DATE'] = pd.to_datetime(df['SALE DATE'])
df['BUILDING CLASS CATEGORY'] = df['BUILDING CLASS CATEGORY'].str.replace(r'^\d+\s+', '', regex=True)
df['BUILDING CLASS CATEGORY'] = df['BUILDING CLASS CATEGORY'].str.title()
df['ZIP CODE'] = df['ZIP CODE'].astype(str).str.replace('\.0', '', regex=True)
```

In the preprocessing stage of our analysis of the New York City Property Sales dataset, we executed several essential steps to ensure the data's integrity and usability. These steps, implemented using the Python library Pandas. Initially, we loaded the dataset, optimizing for memory efficiency due to the data's extensive size. We then identified columns with excessive missing values—specifically, those with over 70% absence—and removed these columns as they could compromise the validity of any subsequent analysis. Similarly, rows with more than half of their values missing were also excluded.

Next, we transformed the 'YEAR BUILT' column from a string representation to a datetime format, handling any non-standard entries, such as '0', by converting them to a missing value indicator. We also cleaned numeric fields, particularly square footage measurements, by stripping non-numeric characters

and converting the strings to floats, ensuring that these values were correctly interpreted for quantitative analysis.

To refine the dataset further, we addressed missing geographical coordinates by interpolating 'Latitude' and 'Longitude' within groups of the same 'BLOCK' and 'NEIGHBORHOOD', which preserved spatial relationships. Additionally, we engineered a new 'TOTAL UNITS' feature by summing the counts of residential and commercial units, providing a more comprehensive metric for property analysis. Through these preprocessing measures, we established a robust foundation for detailed analysis, enabling us to draw more accurate and meaningful insights from the property sales data.

## Task: Dashboard Development and Web Deployment

```
#1
import streamlit as st

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

# Create a sidebar for the date range filter
st.sidebar.header("Filter for Data Exploration")


# Get the minimum and maximum sale dates from your dataset for the date input range
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_date), min_value=min_date, max_value=max_date)

# Filter the DataFrame based on the selected date range
filtered_df = df[(df['SALE DATE'] >= pd.to_datetime(date_range[0])) & (df['SALE DATE'] <= pd.to_datetime(date_range[1]))]

# Select only the specified columns for display
columns_to_display = ['NEIGHBORHOOD', 'BLOCK', 'ADDRESS', 'ZIP CODE', 'YEAR BUILT', 'SALE PRICE', 'SALE DATE']
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)}")


```

The code here utilizes Streamlit, an open-source app framework, for creating a simple web application for exploring property sales data in New York City.

The script starts by importing Streamlit and sets up a header for the web application titled "NYC Property Sales Data Exploration". This establishes the main purpose of the application for end-users. It introduces a sidebar with a date range filter, prompting users to select a specific period for analysis. Utilizing Streamlit's interactive date_input widget, the script fetches the minimum and maximum sale dates from

the dataset to set the boundaries for the date selection. Users can refine the data displayed by selecting a date range within these limits.

Once a date range is chosen, the script filters the DataFrame accordingly. It ensures that only the data entries with sale dates within the selected range are included in the subsequent analysis. The script then focuses on displaying the filtered data. Specific columns are selected for this purpose, such as 'NEIGHBORHOOD', 'BLOCK', 'ADDRESS', 'ZIP CODE', 'YEAR BUILT', 'SALE PRICE', and 'SALE DATE', providing a concise yet informative view. The filtered DataFrame is displayed in the main panel of the web application.

| | NEIGHBORHOOD | BLOCK | ADDRESS | ZIP CODE | YEAR BUILT | SALE PRICE |
|---|---|---|---|---|---|---|
| 0 | CHELSEA | 697 | 555 WEST 25TH STREET | 10001 | 1926-01-01 00:00:00 | 43,300,000 |
| 1 | CHELSEA | 697 | 511 WEST 25TH STREET | 10001 | 1917-01-01 00:00:00 | 148,254,147 |
| 2 | CHELSEA | 700 | 538 WEST 29TH STREET | 10001 | 1910-01-01 00:00:00 | 11,000,000 |
| 3 | CHELSEA | 712 | 450 WEST 15TH | 10011 | 1936-01-01 00:00:00 | 591,800,000 |
| 4 | CHELSEA | 746 | 340 WEST 23RD STREET | 10011 | 1900-01-01 00:00:00 | 0 |
| 5 | CHELSEA | 802 | 158 WEST 27 STREET | 10001 | 1913-01-01 00:00:00 | 99,350,000 |
| 6 | CHELSEA | 803 | 307 7 AVENUE | 10001 | 1926-01-01 00:00:00 | 115,000,000 |
| 7 | CHELSEA | 697 | 521 WEST 25TH STREET | 10001 | 1910-01-01 00:00:00 | 148,254,147 |
| 8 | CHELSEA | 772 | 250 WEST 23RD STREET | 10011 | 1948-01-01 00:00:00 | 14,500,000 |
| 9 | CHELSEA | 772 | 254 WEST 23RD STREET | 10011 | 1920-01-01 00:00:00 | 4,750,000 |

Total sales in the selected period: 558785

The image displayed from a web application, showcasing a table of property sales data filtered by a specific period. The table contains columns for 'NEIGHBORHOOD', 'BLOCK', 'ADDRESS', 'ZIP CODE', 'YEAR BUILT', and 'SALE PRICE'. Each row corresponds to a property sale in the Chelsea neighborhood, providing details such as the exact address, the block number, the postal code, the construction date of the property, and the sale price in USD. The sale prices vary significantly, highlighting the diversity in property values within the area. For instance, the property at '555 WEST 25TH STREET' sold for $43,300,000, while the one at '340 WEST 23RD STREET' shows a sale price of $0, which may indicate a non-monetary transaction or missing data.

Below the table, a summarizing statement reveals the total number of sales transactions in the selected period, amounting to 558,785. This reflects the application's capability to not only present detailed records but also provide aggregate information that can be vital for understanding overall market activity during a given timeframe.

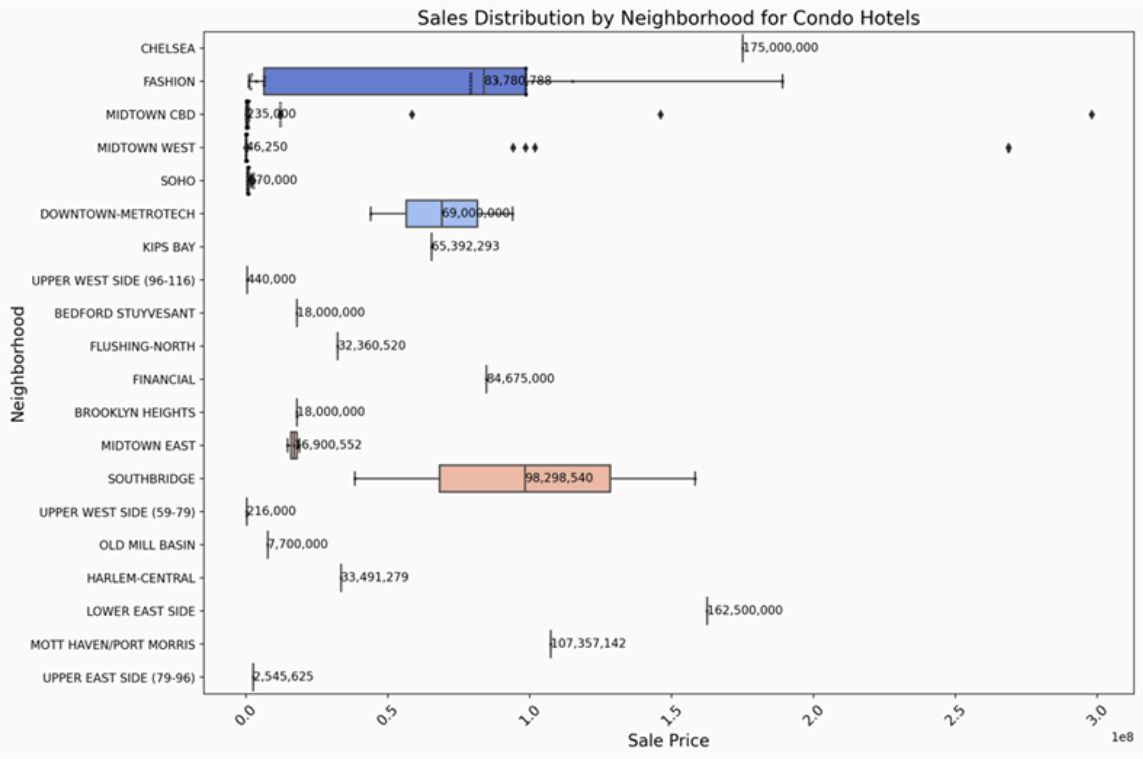
```python
#2
import streamlit as st
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.ticker import StrMethodFormatter

# Assuming your DataFrame is named 'df'
# Convert 'SALE PRICE' to numeric and filter
df = df[(df['SALE PRICE'] > 0) & (df['SALE PRICE'].notnull())]

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

# Create a selectbox for Building Class Category
selected_building_class = st.selectbox("Select Building Class Category:", df['BUILDING CLASS CATEGORY'].unique())

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

# Start the plot
plt.figure(figsize=(14, 10))  # Increased figure size for clarity
sns.boxplot(
    x='SALE PRICE',
    y='NEIGHBORHOOD',
    data=filtered_df,
    palette='coolwarm',  # A lighter palette for a fresh look
)

# Improve readability
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=10)
plt.xlabel("Sale Price", fontsize=14)
plt.ylabel("Neighborhood", fontsize=14)
plt.title(f"Sales Distribution by Neighborhood for {selected_building_class}", fontsize=16)

# Overlay with swarmplot to show individual data points, commented out by default due to performance concerns
sns.swarmplot(x='SALE PRICE', y='NEIGHBORHOOD', data=filtered_df, color='black', alpha=0.5, size=3)

# Annotate median prices
medians = filtered_df.groupby(['NEIGHBORHOOD'])['SALE PRICE'].median().sort_values(ascending=False)
for i, neighborhood in enumerate(filtered_df['NEIGHBORHOOD'].unique()):
    median_val = medians[neighborhood]
    plt.text(median_val, i, f'{median_val:,.0f}', va='center', fontsize=10, color='black')

# Display the plot
st.pyplot(plt)

```

The provided script is a Python-based tool that uses Streamlit for interactivity, along with Pandas for data handling, and Seaborn and Matplotlib for visualization, enabling users to interactively explore property sale prices by neighborhood. It includes a Streamlit select box for filtering data by building class and generates a boxplot to visualize sale price distributions, which are enhanced with median price annotations for immediate insights. This setup offers an intuitive interface for users to analyze the real estate market trends within various neighborhoods without delving into the underlying code.

## Sales Distribution by Neighborhood

Select Building Class Category:

Condo Hotels



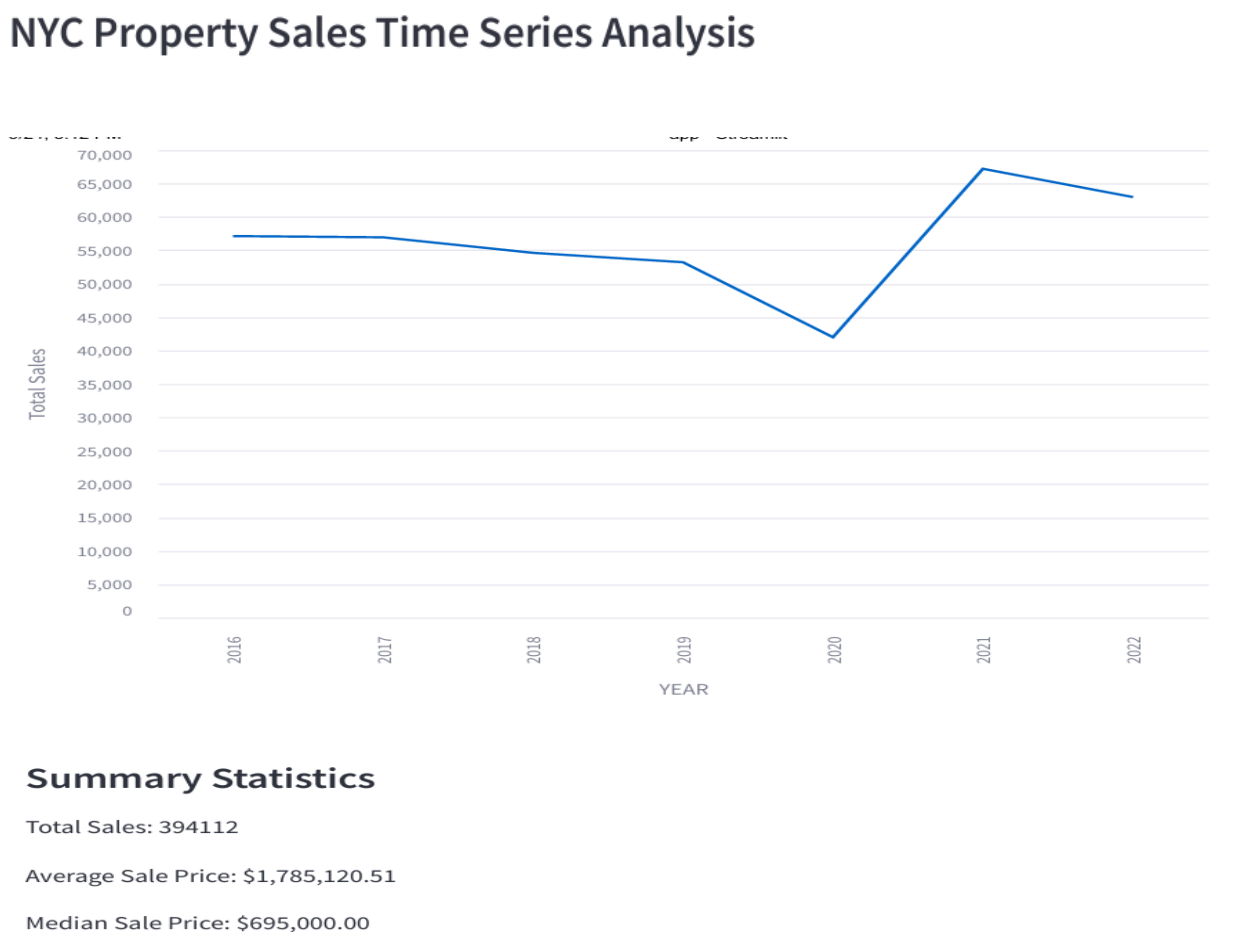

The above depict an interactive visualization of New York City's Condo Hotel sales, grouped by neighborhood. A boxplot graphically represents the range and outliers of sale prices, allowing for quick comparison across areas. Users can select different building classes from a dropdown to dynamically update the chart, making it a flexible tool for analyzing real estate market trends.

```python
#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

# 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_value=max_date, value=min_date)
end_date = st.sidebar.date_input('End Date', min_value=min_date, max_value=max_date, value=max_date)

# 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'] <= end_date)]

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

st.subheader("NYC Property Sales Time Series Analysis")

# 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}')


```

The script is designed to analyze property sales over time within a Streamlit web application. It begins by extracting the year from a 'SALE DATE' column in the data and then uses Streamlit to create interactive sidebar filters that allow users to select a date range for analysis. With the specified date range, the script filters the data and groups it by year to calculate the total number of sales per year.

An Altair line chart is then created to display this time series data, showing how the number of sales has changed over time. The chart is configured to be interactive, providing users with an engaging way to explore the sales trends.

In addition, the script computes summary statistics, including the total number of sales, the average sale price, and the median sale price for the filtered data. These statistics are displayed under a 'Summary Statistics' subheader in the Streamlit app, offering users quick numerical insights into the property sales data within their selected time frame.

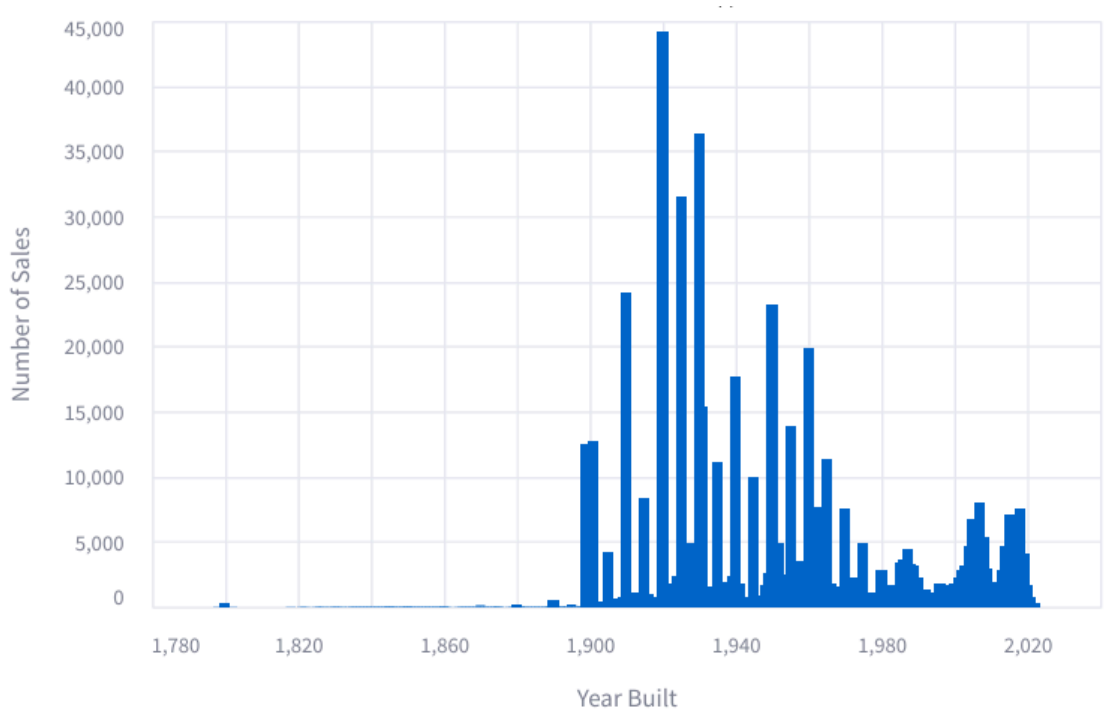



The above visualization is a line chart from a time series analysis of property sales in New York City, spanning from 2016 to 2022. The chart indicates the number of property sales each year, revealing trends such as a decline leading up to 2020, a sharp dip in 2020, followed by a significant recovery in 2021, and a slight drop in 2022. This visual suggests fluctuating real estate activity, with a notable impact around 2020, possibly due to external factors such as economic conditions or events impacting the housing market.

```python
#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, 2022, (1798, 2022))

# Load your cleaned dataset
df = pd.read_csv("NYC_Property_Sales_Data.csv")

# 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)
```

The script creates an interactive feature within a Streamlit application that allows users to filter property sales data based on the year a property was built. A slider in the application's sidebar enables users to define a range of years for which they want to analyze sales. The data is then loaded from a CSV file, and the 'YEAR BUILT' column is converted to a numeric type to ensure proper filtering.

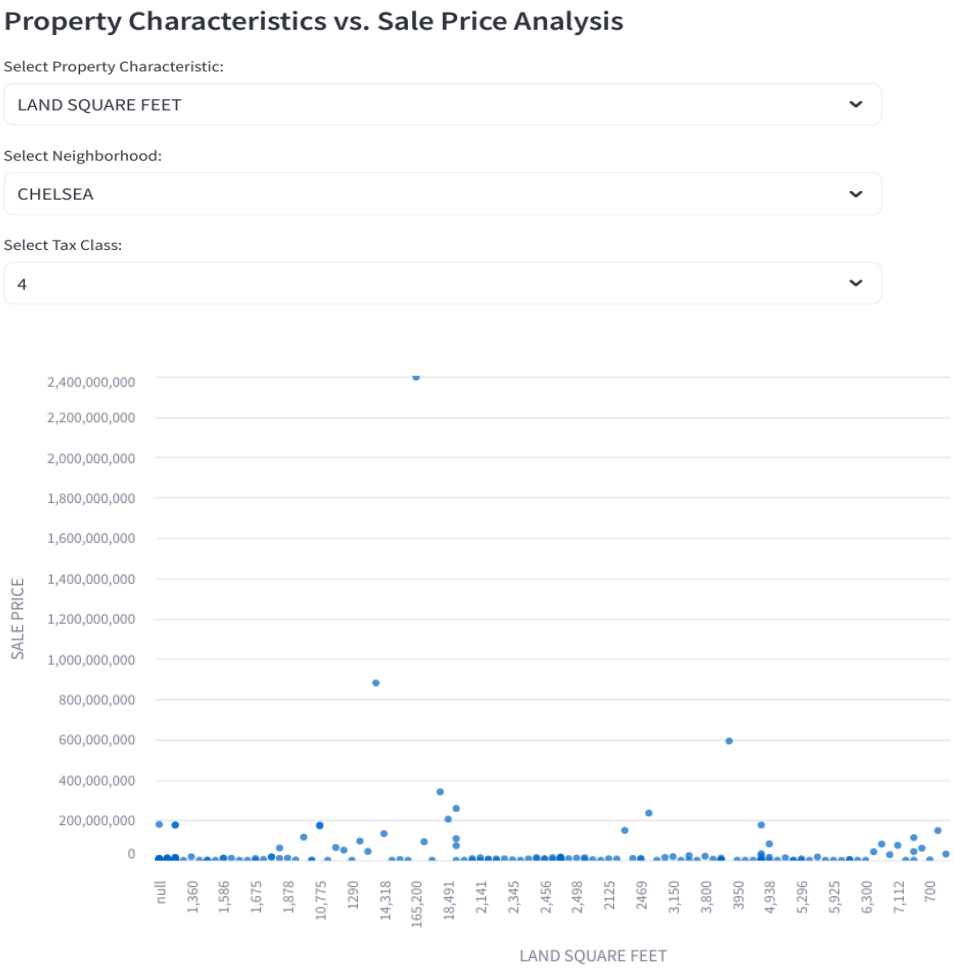
Applying the selected year range as a filter to the dataset, the script generates a histogram using Altair that displays the distribution of the number of sales per year built within the chosen range. The histogram is designed to be interactive, allowing users to explore different facets of the data in a more engaging way. The visualization aids in identifying trends or patterns based on the age of properties sold within the specified timeframe.




The above visualization displays a histogram illustrating the distribution of property sales in New York City by year built. The x-axis represents the construction year of properties, while the y-axis indicates the number of sales. There is a notable concentration of sales for properties built in the early 20th century, with peaks occurring around the 1920s. The frequency of sales declines for properties built in the mid-to-late 20th century, with a resurgence in sales for more recent constructions. This visualization aids in understanding popular times for property development in relation to sales activity.

```
#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 Characteristic:",
                                                ['LAND SQUARE FEET', 'TOTAL UNITS', 'GROSS SQUARE FEET'])
selected_neighborhood = st.selectbox("Select Neighborhood:", df['NEIGHBORHOOD'].unique())
selected_tax_class = st.selectbox("Select Tax Class:", df['TAX CLASS AS OF FINAL ROLL'].unique())

# 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}")

```

The above code sets up an interactive web application using Streamlit for visualizing the relationship between property characteristics and sale prices. It allows users to select a property characteristic, neighborhood, and tax class from dropdown menus. Applying these filters to the dataset, it then generates an Altair scatter plot to visually explore how the chosen characteristic impacts sale prices in the selected neighborhood and tax class. Additionally, it calculates and displays summary statistics, including the average and median sale prices, as well as the total number of sales, offering a quick numerical insight into the filtered subset of property sales data.




## Summary Statistics

Average Sale Price: $32,092,956.80

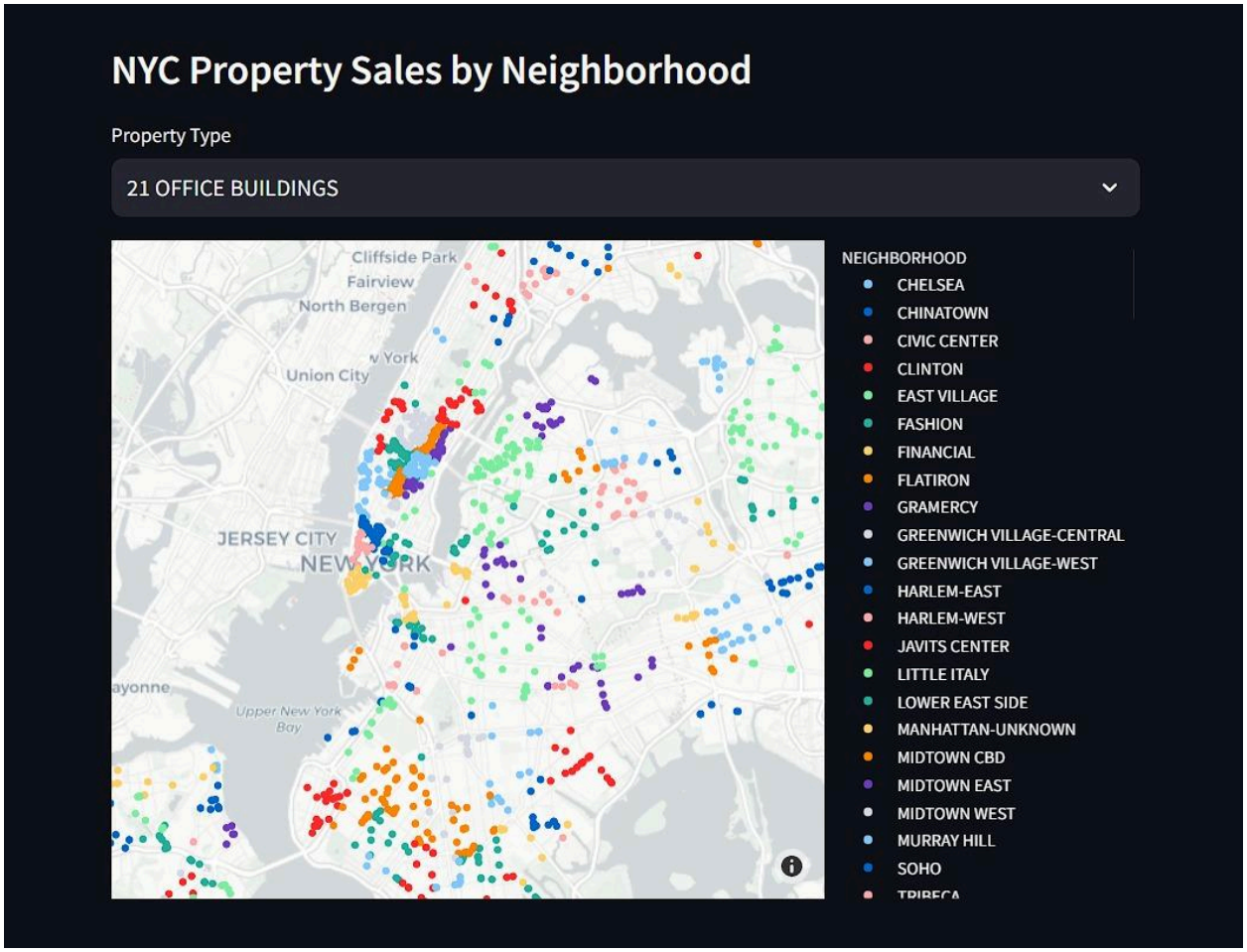
Median Sale Price: $2,290,000.00

Total Sales: 288

The above visualization illustrates a scatter plot depicting the relationship between land square footage and property sale prices. Each point represents a property, with the x-axis indicating the land size in square feet and the y-axis showing the sale price in USD. The plot reveals a distribution of sale prices across various land sizes, with a cluster of data points at the lower end of both axes, suggesting that smaller land sizes are more commonly sold and generally at lower prices. There are also outliers indicating that some larger properties command significantly higher prices, although such sales are less frequent. This visualization is useful for understanding how land size influences property value in the real estate market.

```
#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.subheader('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'].unique())

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

# Plot map using Plotly
fig = px.scatter_mapbox(filtered_df, lat="Latitude", lon="Longitude", color="NEIGHBORHOOD",
                        hover_data=["ADDRESS", "SALE PRICE"],
                        mapbox_style="carto-positron", zoom=10)

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

# Display map with points
st.plotly_chart(fig)


```

The script configures a Streamlit web application to visualize New York City property sales on an interactive map. Users can filter sales by date and property type via Streamlit's sidebar widgets. The data is then plotted on a map using Plotly Express, with points representing sales, color-coded by neighborhood, and including details like address and sale price on hover. This visualization allows for an intuitive understanding of the geographical distribution of property sales within the selected parameters.




The above visualization showcases a map from a web application depicting the distribution of property sales in various neighborhoods across New York City, with a focus on office buildings. Color-coded dots on the map represent individual property sales, with the colors corresponding to different neighborhoods as indicated in the legend. This visual format provides a clear geographic distribution of office building sales, allowing users to quickly discern patterns and concentrations of sales activity across the city. The interactive dropdown menu enables users to select and filter the property type, in this case, '21 OFFICE BUILDINGS', for a tailored analytical view.

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

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

# Filter widgets
property_type = st.selectbox('Select Property Type', df['BUILDING CLASS CATEGORY'].unique(), key='property_type_selectbox')
neighborhood = st.selectbox('Select Neighborhood', df['NEIGHBORHOOD'].unique(), key='neighborhood_selectbox')
start_date = st.date_input('Start Date', min_value=pd.to_datetime(df['SALE DATE']).min(), max_value=pd.to_datetime(df['SALE DATE']).max(),
value=pd.to_datetime(df['SALE DATE']).min(), key='start_date_input')
end_date = st.date_input('End Date', min_value=pd.to_datetime(df['SALE DATE']).min(), max_value=pd.to_datetime(df['SALE DATE']).max(),
value=pd.to_datetime(df['SALE DATE']).max(), key='end_date_input')

# 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 FEET'],
                 trendline='ols', title='Total Units vs. Sale Price',
                 labels={'TOTAL UNITS': 'Total Units', 'SALE PRICE': 'Sale Price'})

# 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}")

```

The above code generates an interactive plot for a real estate web application, correlating property characteristics with sale prices. Utilizing Streamlit, users can filter properties by type and neighborhood and select a sale date range. After data filtering based on these selections, a Plotly scatter plot is created, highlighting the relationship between 'TOTAL UNITS' and 'SALE PRICE', enriched with a trend line. Additional hover details like 'ADDRESS' and 'GROSS SQUARE FEET' provide deeper insights. The plot is customizable and designed for clarity in presentation. Summary statistics such as average and median sale prices, as well as the total number of sales, are computed and displayed, giving users immediate access to key market indicators.

## Property Characteristics vs. Sale Price Analysis

Select Property Type

21 OFFICE BUILDINGS

Select Neighborhood

CHELSEA

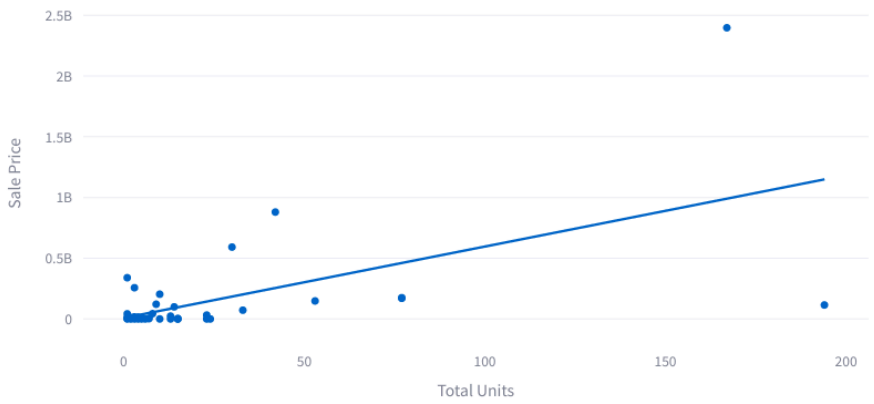
Start Date

2016/01/01

End Date

2022/12/31

**Total Units vs. Sale Price**




## Summary Statistics

Average Sale Price: $138,743,985.24

Median Sale Price: $11,666,666.50

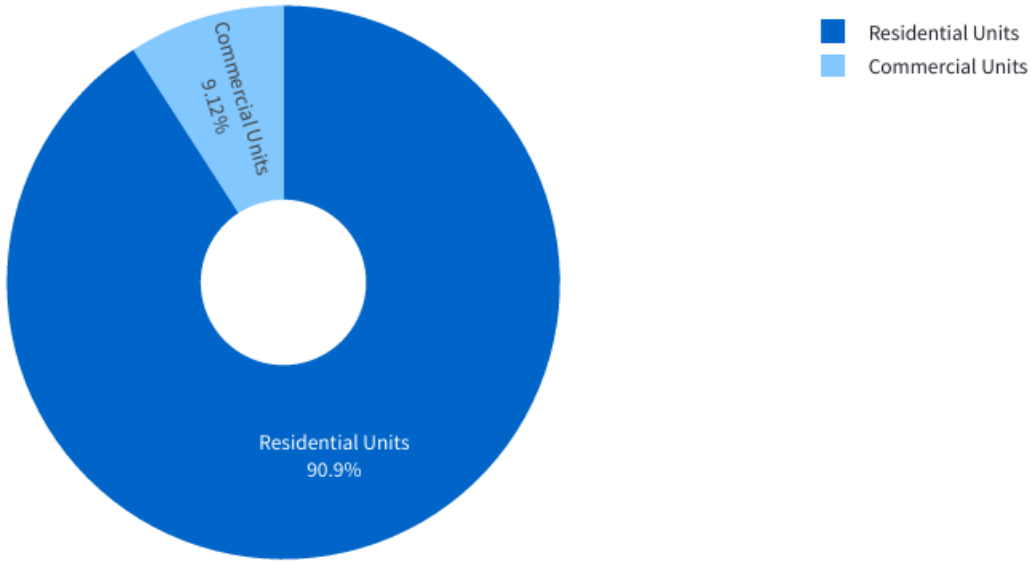
Total Sales: 42

The above visualization portrays a scatter plot visualizing the relationship between the total units in properties and their sale prices, along with a line indicating the trend of increasing prices with more units. Notably, there appears to be a positive correlation between the size of a property, in terms of total units, and its sale price. Additionally, summary statistics are provided below the graph, including an average sale price of approximately $138.7 million, a median sale price of about $11.7 million, and a total of 42 sales captured in the dataset. This information can help stakeholders gauge market value trends related to property size.

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

st.subheader('Unit Type Distribution')

# 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',
             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)

```

The above code utilizes Streamlit alongside Pandas and Plotly Express to create a pie chart for displaying the distribution of unit types in a property dataset. The Streamlit header indicates the focus of the visualization on 'Unit Type Distribution'. The script calculates the total counts of residential and commercial units within the dataset. A DataFrame is constructed with these totals, which is then used to generate a pie chart. This chart is designed to be interactive, with hover labels and a donut style (indicated by the 'hole' parameter set to 0.3), providing a visual breakdown of the proportion of residential to commercial units. Percentage labels are added to each segment for clarity. The resulting plot is displayed within the Streamlit app, offering a quick and easy understanding of the composition of property types.

## Unit Type Distribution

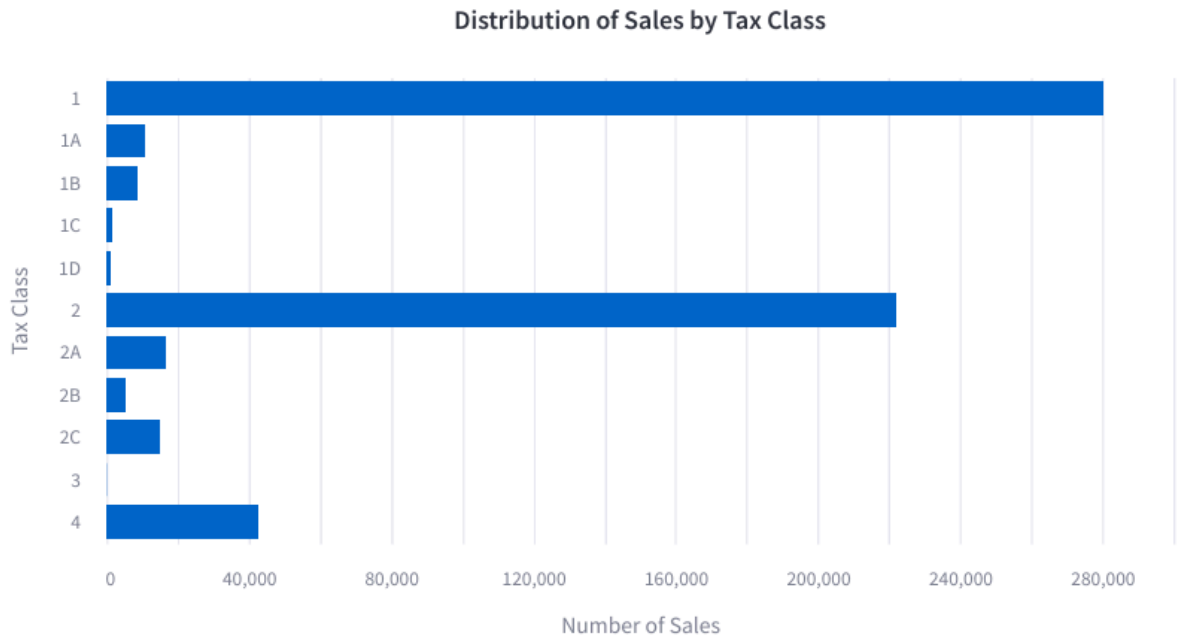



The image displays a donut chart illustrating the distribution of unit types within a property dataset. The chart shows a predominant share of residential units, accounting for 90.9% of the total, while commercial units make up a smaller fraction at 9.1%. This visual representation emphasizes the significantly higher proportion of residential units in comparison to commercial spaces, offering an at-a-glance understanding of the composition of the properties analyzed. The chart serves as an analytical tool to quickly assess the balance of residential versus commercial property types within the market.

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


st.subheader('Distribution of Property Sales by Tax Class')
# 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'] <= end_date)]

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

# Plotting with Altair
bar_chart = alt.Chart(tax_class_counts).mark_bar().encode(
    y=alt.Y('Tax Class:O', title='Tax Class'),
    x=alt.X('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)

```

This above code outlines the process of creating an interactive bar chart that visualizes the distribution of property sales by tax class in a Streamlit web application. The chart is generated using Altair and is based on data filtered by a user-defined sale date range. After filtering, the data is grouped by the 'TAX CLASS AS OF FINAL ROLL' column, and the count of sales for each tax class is computed. The resulting bar chart presents the number of sales per tax class, complete with tooltips for additional detail. It's designed for user interaction and enhanced readability with customized fonts and labels, providing an informative visual summary of sales distribution which can easily be incorporated into a report for stakeholders.

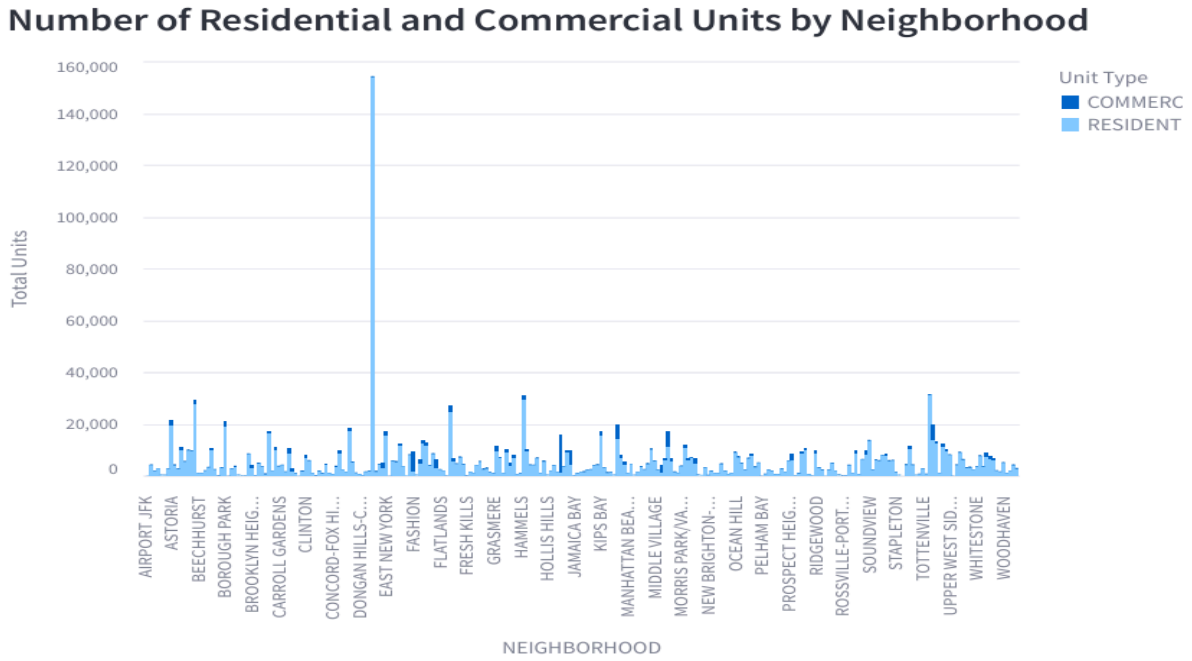
## Distribution of Property Sales by Tax Class




The above visualization presents a horizontal bar chart titled "Distribution of Property Sales by Tax Class", which provides a visual breakdown of the number of property sales grouped by different tax classes. The chart displays a series of tax classes on the y-axis and the corresponding number of sales on the x-axis. Tax class '2' appears to have the highest number of sales, suggesting it represents a significant portion of the property market. The visualization simplifies complex data into an easily interpretable format, enabling quick insights into which tax classes are most common among sold properties, information that can be valuable for market analysis and reporting.

```python
#10

st.subheader('Number of Residential and Commercial Units by Neighborhood')

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

# Melt the DataFrame to long format for easier plotting
units_by_neighborhood_melted = units_by_neighborhood.melt(id_vars='NEIGHBORHOOD', var_name='Unit Type', value_name='Total Units')

# 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()

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

The above code is set up to visualize the distribution of residential and commercial units across neighborhoods within a Streamlit application. It starts by grouping the data by neighborhood and summing up the residential and commercial units within each one. To prepare for visualization, it reshapes the grouped data into a long format suitable for plotting. An Altair stacked bar chart is then created, showing the total number of units for each neighborhood differentiated by unit type. The chart is configured to be interactive and is finally displayed in the Streamlit app, allowing for an intuitive comparison of residential and commercial properties across different neighborhoods.



The above visualization depicts a bar chart visualizing the number of residential and commercial units within various neighborhoods. The bars are segmented by unit type, displaying a comparison between the two. This visualization clearly shows that residential units are more prevalent than commercial units across most neighborhoods. A particular neighborhood stands out with a significantly higher number of residential units compared to others. Such a chart is useful for identifying residential and commercial property distribution patterns, which can inform real estate investment strategies, urban planning, and market analysis.

## Conclusion

In conclusion, the project report presented a detailed exploration of New York City's property sales data from 2016 to 2022, employing a variety of data processing and visualization techniques. Geocoding enriched the dataset with precise location coordinates, facilitating a geospatial analysis that led to interactive maps showcasing sales distributions across neighborhoods. Further, time-series analyses highlighted market dynamics over time, while data cleaning and preparation ensured the reliability of the subsequent analyses. Visualizations, including box plots and pie charts, illustrated the disparities in property features such as unit types, square footage, and the association of these features with sale prices.

This comprehensive analysis provided insights into the NYC real estate market's complexity, underlined by residential dominance over commercial units and a clear preference for certain neighborhoods and property types. The Streamlit web application's interactive nature allowed users to engage directly with the data, ensuring the project's findings were both accessible and actionable. The data-driven approach laid a strong foundation for future market predictions and investment strategies within the ever-changing landscape of New York City real estate.