**REAL ESTATE**

**Overview of the Dataset:**

The dataset for "Real Estate" contains the following features:

1. PropertyID: Unique identifier for each property.
2. xrCompositeLandUseID: Identifier for the composite land use category of the property.
3. xrBuildingTypeID: Identifier for the type of building on the property.
4. ParcelID: Identifier for the parcel associated with the property.
5. LocationStartNumber: Starting number of the property's location.
6. ApartmentUnitNumber: Number of the apartment unit, if applicable.
7. StreetNameAndWay: Name of the street or way where the property is located.
8. xrPrimaryNeighborhoodID: Identifier for the primary neighborhood where the property is located.
9. LandSF: Area of land in square feet.
10. TotalFinishedArea: Total finished area of the property.
11. LivingUnits: Number of living units in the property.
12. OwnerLastName: Last name of the property owner.
13. OwnerFirstName: First name of the property owner.
14. PrimaryGrantor: Grantor associated with the property.
15. SaleDate: Date of the property sale.
16. SalePrice: Price at which the property was sold.
17. TotalAppraisedValue: Total appraised value of the property.
18. LegalReference: Legal reference associated with the property.
19. xrSalesValidityID: Identifier for the sales validity status.
20. xrDeedID: Identifier for the deed associated with the property.

This dataset appears to provide information about various aspects of real estate properties, including their identifiers, location, land use, building type, area, ownership, sales details, and appraised values.

Top of Form

**Overview of the Tool used: Python**

Python is a versatile and widely used programming language that is extensively utilized for data analysis, including the analysis of the "Telecom Users" dataset.:

1. Versatile and Easy to Use
2. Abundance of Libraries
3. Data Manipulation and Exploration
4. Data Visualization
5. Machine Learning and Statistical Analysis
6. Integration and Collaboration
7. Scalability and Performance

**Overview of Libraries Used:**

1. **Numpy:**

* Numpy is a powerful numerical computing library in Python.
* It provides support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently.
* It is widely used for tasks involving numerical computations, data manipulation, and scientific computing.
* It offers various functionalities such as array creation, indexing, slicing, mathematical operations, linear algebra operations, random number generation, and more.
* Numpy is known for its performance and optimized algorithms, making it a fundamental library for data analysis and scientific computing in Python.

1. **Pandas:**

* Pandas is a versatile data manipulation and analysis library in Python.
* It provides powerful data structures, such as Data Frames and Series, which allow for efficient handling and analysis of structured data.
* Pandas excels in data cleaning, transformation, merging, reshaping, and aggregation tasks.
* It offers a wide range of functions and methods for data exploration, data pre-processing, and data wrangling.
* Pandas is extensively used in data analysis, data pre-processing, and data preparation workflows, as well as in data visualization and statistical analysis.

**Matplotlib:**

Matplotlib is a popular plotting library in Python.

* It provides a comprehensive set of tools for creating static, animated, and interactive visualizations in Python.
* Matplotlib enables the creation of various types of plots, including line plots, scatter plots, bar plots, histograms, pie charts, and more.
* It offers a high level of customization and control over plot elements such as axes, labels, colors, markers, and legends.
* Matplotlib is widely used for data visualization, exploration, and presentation purposes in scientific, statistical, and data analysis projects.

**Seaborn:**

* Seaborn is a statistical data visualization library built on top of Matplotlib.
* It provides a higher-level interface for creating attractive and informative statistical graphics.
* Seaborn simplifies the process of creating complex plots, such as heatmaps, violin plots, box plots, and cluster maps, with just a few lines of code.
* It offers a wide range of built-in themes and color palettes that enhance the visual appeal of the plots.
* Seaborn also provides functions for advanced statistical visualizations, such as regression plots, distribution plots, and categorical plots.
* It is commonly used in exploratory data analysis, statistical modeling, and data-driven storytelling.

**Summary:**

Number of Rows: **4735**

Number of Columns: **20**

**Pre-processing:**

import pandas as pd

import numpy as np

df=pd.read\_csv("real-estate-sales.csv")

**# Feature Selection**

df1=df.copy()

df1.drop(df.columns[[0,1,2,3,5,12,18,19]], axis=1, inplace=True)

df1.isnull().sum()

**#Filling Null values**

df1['LocationStartNumber']=df1['LocationStartNumber'].fillna(0)

df1['TotalFinishedArear']=df1['TotalFinishedArea'].fillna(0)

df1['LivingUnitsr']=df1['LivingUnits'].fillna(0)

df1['LandSFr']=df1['LandSF'].fillna(0)

**#Finding Mean Values**

df1['LandSF'].fillna(df1['LandSF'].mean())

df1['TotalFinishedArea'].fillna(df1['TotalFinishedArea'].mean())

df1['LivingUnits'].fillna(df1['LivingUnits'].mean())

1**. Total Sales per Quarter of the Real Estate**

**Code:**

df['SaleDate'] = pd.to\_datetime(df['SaleDate'])

df1['Quarter'] = df1['SaleDate'].dt.quarter

quarterly\_sales = df1.groupby('Quarter')['SalePrice'].sum()

import matplotlib.pyplot as plt

plt.figure(figsize=(6,6))

plt.barh(quarterly\_sales.index, quarterly\_sales)

plt.xlabel("Amount of Sales")

plt.ylabel("Quarter")

plt.title("Quarterly Sale")

**Output:**

Quarter

1 682912951

2 247882984

3 653169968

4 1887341730

**Chart:**

****

1. **Average Sale price per Area of the Real Estate**

**Code:**

avg\_sale = df1.groupby(['StreetNameAndWay'])['SalePrice'].mean()

plt.figure(figsize=(10,6))

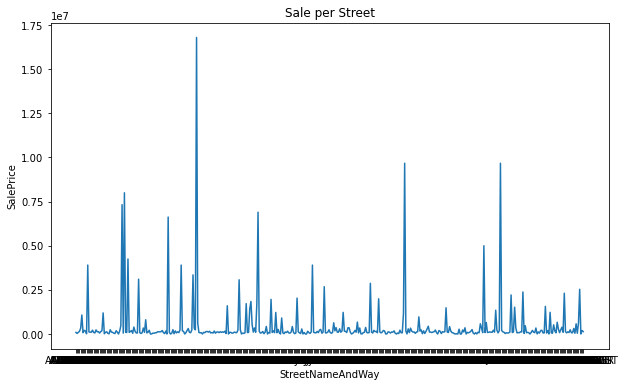
plt.plot(avg\_sale)

plt.xlabel("StreetNameAndWay")

plt.ylabel("SalePrice")

plt.title("Sale per Street")

**Chart:**

****

1. **Which month has the biggest Sale**

**Code:**

from datetime import datetime

import calendar

df1['month'] = df1['SaleDate'].dt.month

month\_number = df1['month'].loc[df1['SalePrice'].idxmax()]

Max\_amount = df1['SalePrice'].loc[df1['SalePrice'].idxmax()]

max\_sales\_month=calendar.month\_name[month\_number]

print("The Higest sale is->", Max\_amount, "->in the Month of--> ", max\_sales\_month)

**Output:**

The Highest sale is-> 70500000 ->in the Month of--> May

1. **Maximum sales per Area**

**Code:**

df1.reset\_index(drop=True)

Max\_Sale = df1.groupby(['StreetNameAndWay'])['SalePrice'].max().reset\_index(name='max')

print(Max\_Sale)

plt.figure(figsize=(10,6))

plt.bar(range(len(Max\_Sale)), Max\_Sale["max"])

plt.xlabel("StreetNameAndWay")

plt.ylabel("SalePrice")

plt.title("Maximum Sale per Street")

**Chart:**

|  |  |
| --- | --- |
|  |  |

1. **Total Finished Area VS Sale Price**

**Code:**

# Extract the variables

x = df['TotalFinishedArea']

y = df['SalePrice']

# Create the scatter plot

plt.scatter(x, y, s=10, alpha=0.5, color='blue')

# Set labels and title

plt.xlabel('Total Finished Area')

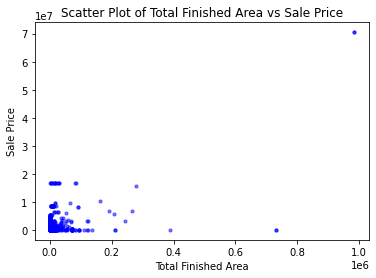
plt.ylabel('Sale Price')

plt.title('Scatter Plot of Total Finished Area vs Sale Price')

# Display the plot

plt.show()

**Chart:**



1. **StreetNameAndWay VS Sale Price**

**Code:**

# Extract the variables

x = df['StreetNameAndWay']

y = df['SalePrice']

# Create the scatter plot

plt.scatter(x, y, s=10, alpha=0.5, color='blue')

# Set labels and title

plt.xlabel('StreetNameAndWay')

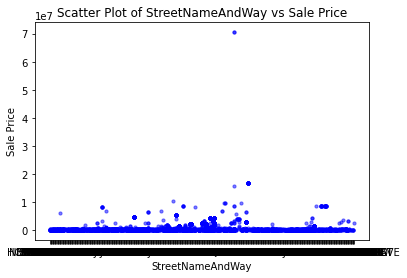
plt.ylabel('Sale Price')

plt.title('Scatter Plot of StreetNameAndWay vs Sale Price')

# Display the plot

plt.show()

**Chart:**

****

1. **Top 5 Sales**

**Code:**

Max\_sale= df1.sort\_values(by ='SalePrice', ascending=False)[['StreetNameAndWay','SalePrice']]

Top\_sale = Max\_sale.head(5)

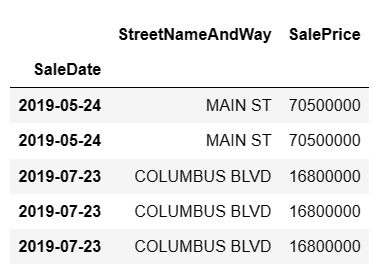
Top\_sale

(0r)

Sort\_Max\_sale= Max\_sale.sort\_values(by ='SalePrice', ascending=False)

Sort\_Max\_sale.nlargest(5,'SalePrice')

**Output:**

****

**ADDITIONAL SUMMARY**

**1.** Determine the total number of properties in the dataset

**Code:**

# Determine the total number of properties in the dataset.

total\_properties = len(df['PropertyID'])

print("Total number of Properties",total\_properties )

**Output:**

Total number of Properties 4735

2. Analyze the distribution of composite land use types (xrCompositeLandUseID) to identify the most common land use categories.

**Code:**

# Analyze the distribution of composite land use types

land\_use\_counts = df['xrCompositeLandUseID'].value\_counts()

# Get the top 5 most common land use categories

top\_land\_use\_categories = land\_use\_counts.head(10)

# Plot the distribution of composite land use types

plt.figure(figsize=(15, 6))

land\_use\_counts.plot(kind='bar')

plt.xlabel('Composite Land Use Type')

plt.ylabel('Count')

plt.title('Distribution of Composite Land Use Types')

plt.xticks(rotation=45)

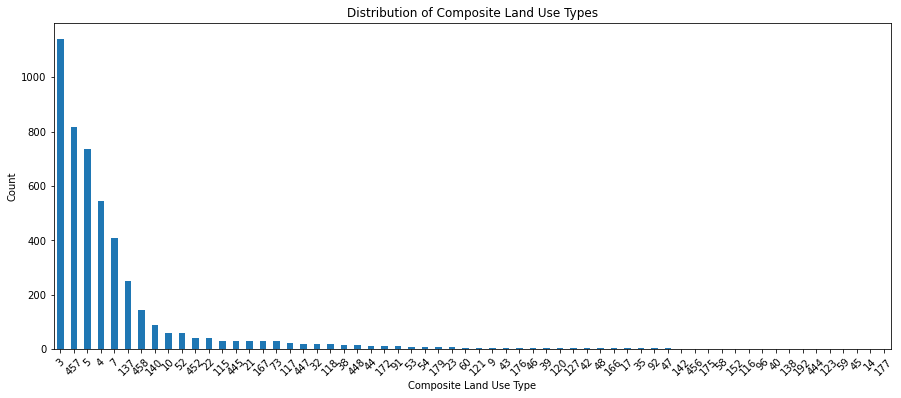
plt.show()

# Print the top 5 most common land use categories

print("Top 5 most common land use categories:")

print(top\_land\_use\_categories)

**Chart:**

****

**3.** Explore the distribution of building types (xrBuildingTypeID) to understand the diversity of property structures.

**Code:**

# Explore the distribution of building types

building\_type\_counts = df['xrBuildingTypeID'].value\_counts()

# Plot the distribution of building types

plt.figure(figsize=(15, 6))

building\_type\_counts.plot(kind='bar')

plt.xlabel('Building Type')

plt.ylabel('Count')

plt.title('Distribution of Building Types')

plt.xticks(rotation=45)

plt.show()

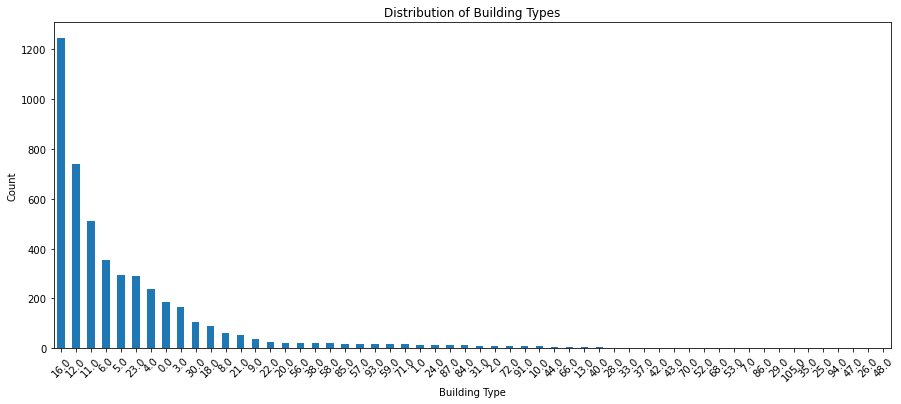
# Print the top 5 most common building types

top\_buidling\_type = building\_type\_counts.head(10)

print("Top 5 most common building types:")

print(top\_buidling\_type)

**Chart:**

****

**4.** Calculate the average and median sale prices (SalePrice) to assess the overall pricing trends.

**Code:**

# Calculate the average sale price

average\_price = df['SalePrice'].mean()

# Calculate the median sale price

median\_price = df['SalePrice'].median()

# Print the results

print("Average sale price:", average\_price)

print("Median sale price:", median\_price)

**Output:**

Average sale price: 733116.7123548046

Median sale price: 107000.0

5. Identify properties with the highest and lowest sale prices to understand the range of property values.

**Code:**

# Identify properties with the highest and lowest sale prices

highest\_price\_property = df['SalePrice'].max()

lowest\_price\_property = df['SalePrice'].min()

# Print the results

print("Property with the highest sale price:", highest\_price\_property)

print("\nProperty with the lowest sale price:", lowest\_price\_property )

**Output:**

Property with the highest sale price: 70500000

Property with the lowest sale price: 0

6. Investigate the relationship between land area (LandSF) and sale prices to determine if there is a correlation.

**Code:**

# Create a scatter plot of land area vs. sale prices

plt.scatter(df['LandSF'], df['SalePrice'])

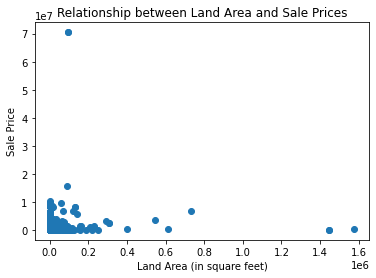
plt.xlabel('Land Area (in square feet)')

plt.ylabel('Sale Price')

plt.title('Relationship between Land Area and Sale Prices')

plt.show()

**Chart:**



1. Analyze the distribution of living units (LivingUnits) to assess the density of properties.

**Code:**

# Calculate the distribution of living units

living\_units\_counts = df['LivingUnits'].value\_counts()

# Plot the distribution

plt.bar(living\_units\_counts.index, living\_units\_counts.values)

plt.xlabel('Number of Living Units')

plt.ylabel('Frequency')

plt.title('Distribution of Living Units')

plt.show()

# Top 10 living units

top\_living\_units = living\_units\_counts.head(10)

top\_living\_units

**Chart:**

|  |  |
| --- | --- |
|  |  |

8. Examine the distribution of property owners (OwnerLastName, OwnerFirstName) to identify the most frequent owners.

**Code:**

# Combine owner's first and last names

df['OwnerFullName'] = df['OwnerFirstName'] + ' ' + df['OwnerLastName']

# Calculate the distribution of property owners

owner\_counts = df['OwnerFullName'].value\_counts()

# Plot the distribution

plt.figure(figsize=(15,6))

plt.bar(owner\_counts.index, owner\_counts.values)

plt.xlabel('Property Owner')

plt.ylabel('Frequency')

plt.title('Distribution of Property Owners')

#plt.xticks(rotation=90)

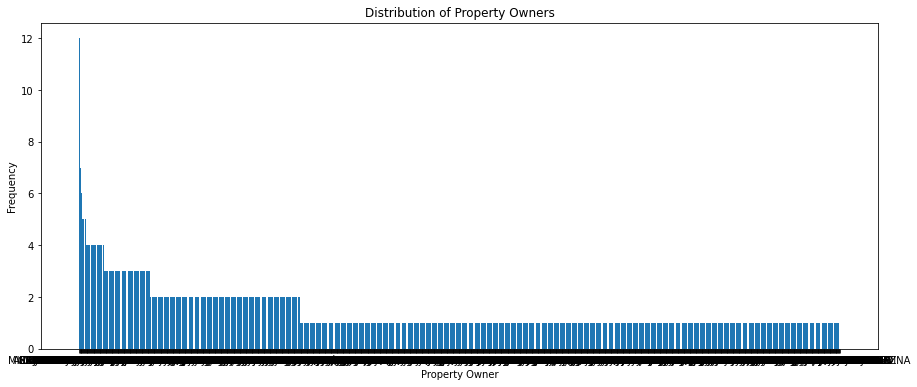
plt.show()

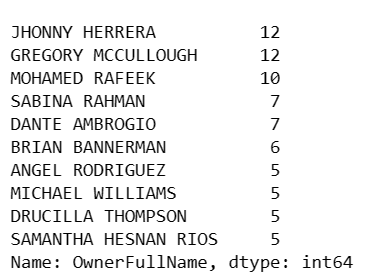
# Top 10 property owners

Top\_Owners = owner\_counts.head(10)

Top\_Owners

**Chart:**





9. Identify the top grantors (PrimaryGrantor) involved in property transactions.

**Code:**

# Calculate the top grantors

top\_grantors = df['PrimaryGrantor'].value\_counts().nlargest(10)

# Plot the top grantors

plt.bar(top\_grantors.index, top\_grantors.values)

plt.xlabel('Primary Grantor')

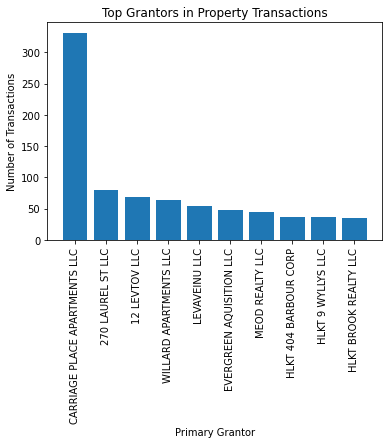
plt.ylabel('Number of Transactions')

plt.title('Top Grantors in Property Transactions')

plt.xticks(rotation=90)

plt.show()

**Chart:**



10. Explore the distribution of sale dates (SaleDate) to identify any seasonality or trends in real estate sales.

**Code:**

# Convert SaleDate to datetime format

df['SaleDate'] = pd.to\_datetime(df['SaleDate'])

# Extract year and month from SaleDate

df['Year'] = df['SaleDate'].dt.year

df['Month'] = df['SaleDate'].dt.month

# Count the number of sales for each year

sales\_per\_year = df['Year'].value\_counts().sort\_index()

# Count the number of sales for each month

sales\_per\_month = df.groupby('Month')['SaleDate'].count()

# Plot the distribution of sales per year

plt.figure(figsize=(10, 6))

plt.bar(sales\_per\_year.index, sales\_per\_year.values)

plt.xlabel('Year')

plt.ylabel('Number of Sales')

plt.title('Distribution of Real Estate Sales per Year')

plt.show()

# Plot the distribution of sales per month

plt.figure(figsize=(10, 6))

plt.bar(sales\_per\_month.index, sales\_per\_month.values)

plt.xlabel('Month')

plt.ylabel('Number of Sales')

plt.title('Distribution of Real Estate Sales per Month')

plt.show()

**Chart:**

|  |  |
| --- | --- |
|  |  |

11. Calculate the total appraised value (TotalAppraisedValue) to understand the overall value of the properties.

Code:

# Calculate the total appraised value

total\_appraised\_value = df['TotalAppraisedValue'].sum()

print("Total Appraised Value:", total\_appraised\_value)

**Output:**

Total Appraised Value: 1093437054

12. Investigate the legal references (LegalReference) associated with property transactions.

**Code:**

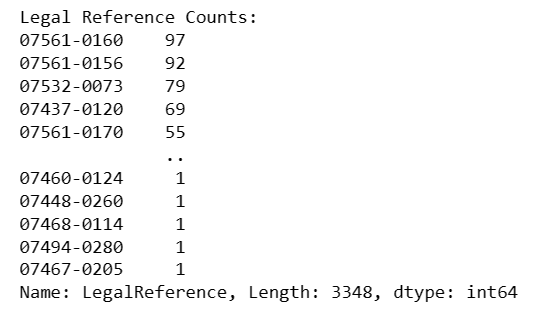
# Count the occurrences of each legal reference

legal\_reference\_counts = df['LegalReference'].value\_counts()

print("Legal Reference Counts:")

print(legal\_reference\_counts)

**Output:**



13. Analyze the sales validity (xrSalesValidityID) to assess the quality and accuracy of sales data.

**Code:**

# Count the occurrences of each sales validity status

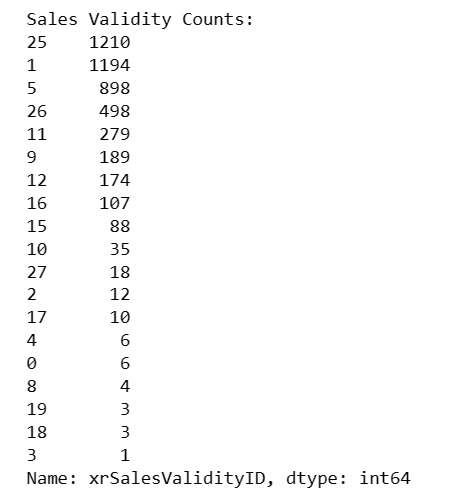
sales\_validity\_counts = df['xrSalesValidityID'].value\_counts()

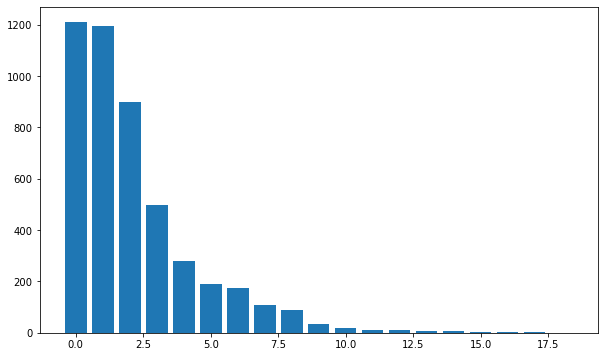
print("Sales Validity Counts:")

print(sales\_validity\_counts)

plt.figure(figsize=(10,6))

plt.bar(range(len(sales\_validity\_counts)), sales\_validity\_counts.values)





14. Examine the types of deeds (xrDeedID) involved in property transactions.

**Code:**

# Get the unique types of deeds

deed\_types = df['xrDeedID'].unique()

deed\_types.sort()

print("Deed Types:")

for deed\_type in deed\_types:

print(deed\_type)

plt.figure(figsize=(10,6))

sns.violinplot(data=df, x='xrDeedID')

plt.xlabel('Deed Types')

plt.ylabel('Frequency')

plt.title('Violin Plot of Deed Types')

# Rotate x-axis labels if needed

plt.xticks(rotation=45)

# Show the plot

plt.show()

**Chart:**

|  |  |
| --- | --- |
|  |  |

15. Calculate the average and median total finished area (TotalFinishedArea) to understand the typical property size.

Code:

# Calculate the average and median total finished area

average\_finished\_area = df['TotalFinishedArea'].mean()

median\_finished\_area = df['TotalFinishedArea'].median()

print("Average Total Finished Area:", average\_finished\_area)

print("Median Total Finished Area:", median\_finished\_area)

Output:

Average Total Finished Area: 4855.434745570989

Median Total Finished Area: 1776.000005

16. Identify the primary neighborhoods (xrPrimaryNeighborhoodID) with the highest number of properties

**Code:**

# Group the dataset by primary neighborhoods and count the number of properties in each neighborhood

neighborhood\_counts = df['xrPrimaryNeighborhoodID'].value\_counts()

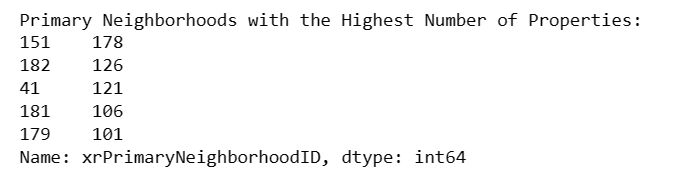
# Get the primary neighborhoods with the highest number of properties

top\_neighborhoods = neighborhood\_counts.head(5) # Adjust the number as per your requirement

print("Primary Neighborhoods with the Highest Number of Properties:")

print(top\_neighborhoods)

**Output:**



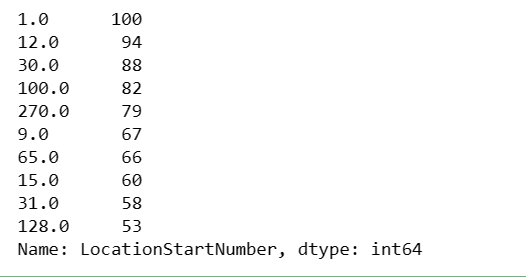
17. Explore the distribution of location start numbers (LocationStartNumber) to identify any spatial patterns.

**Code:**

location\_count= df['LocationStartNumber'].value\_counts()

location\_count.head(10)

**Output:**



**Code:**

# Plot the histogram of location start numbers

plt.figure(figsize=(10, 6))

plt.hist(df['LocationStartNumber'], bins=30, edgecolor='black')

plt.xlabel('Location Start Number')

plt.ylabel('Frequency')

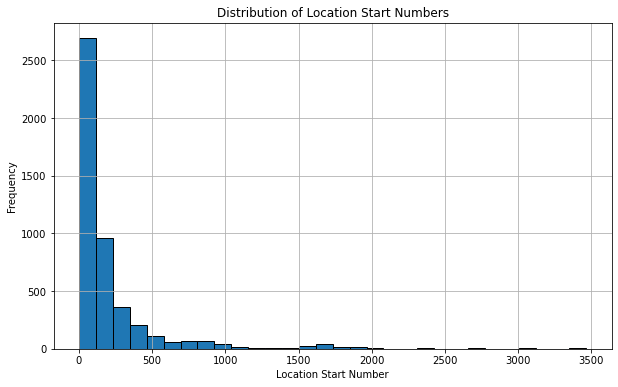
plt.title('Distribution of Location Start Numbers')

plt.grid(True)

# Show the plot

plt.show()

**Chart:**



18. Analyze the presence of apartment units (ApartmentUnitNumber) in the dataset.

**Code:**

# Count the number of properties with and without apartment units

has\_apartment = df['ApartmentUnitNumber'].notnull().sum()

no\_apartment = df['ApartmentUnitNumber'].isnull().sum()

# Create a pie chart to visualize the presence of apartment units

labels = ['Has Apartment', 'No Apartment']

sizes = [has\_apartment, no\_apartment]

colors = ['#ff9999', '#66b3ff']

explode = (0.1, 0)

plt.figure(figsize=(6, 6))

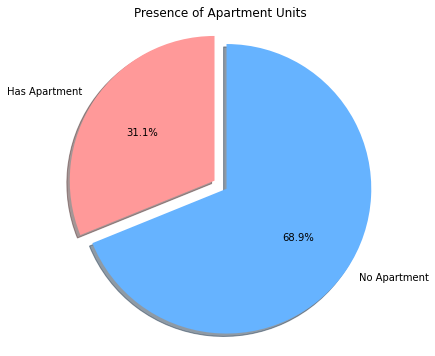
plt.pie(sizes, explode=explode, labels=labels, colors=colors, autopct='%1.1f%%', shadow=True, startangle=90)

plt.axis('equal')

plt.title('Presence of Apartment Units')

plt.show()

**Chart:**



19. Investigate the distribution of street names (StreetNameAndWay) to identify common street names.

**Code:**

# Count the frequency of each street name

street\_counts = df['StreetNameAndWay'].value\_counts()

# Select the top 10 most common street names

top\_10\_streets = street\_counts.head(10)

# Plot the bar chart of street names

plt.figure(figsize=(10, 6))

top\_10\_streets.plot(kind='bar')

plt.xlabel('Street Name')

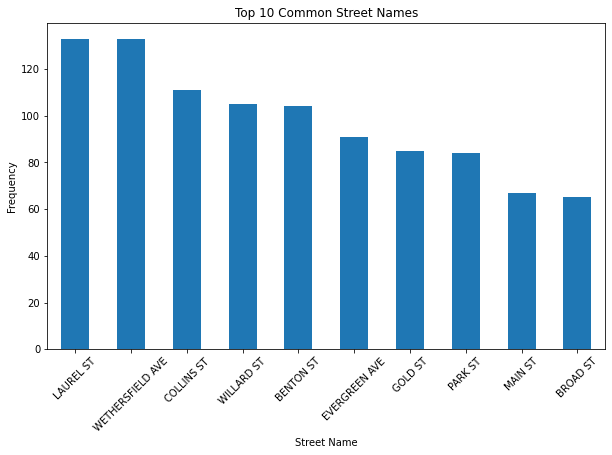
plt.ylabel('Frequency')

plt.title('Top 10 Common Street Names')

plt.xticks(rotation=45)

plt.show()

**Chart:**



20. Assess the relationship between land area and total finished area to understand property utilization.

**Code:**

# Plot scatter plot

plt.figure(figsize=(8, 6))

plt.scatter(df['LandSF'], df['TotalFinishedArea'])

plt.xlabel('Land Area (in sq. ft.)')

plt.ylabel('Total Finished Area (in sq. ft.)')

plt.title('Relationship between Land Area and Total Finished Area')

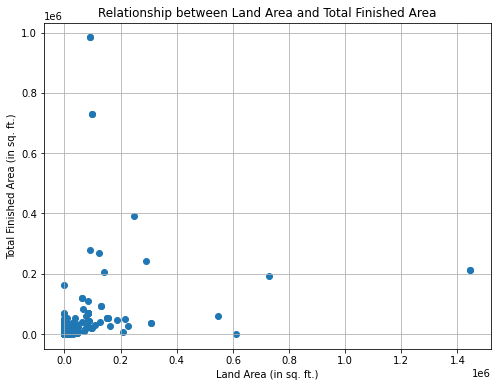
# Add gridlines

plt.grid(True)

# Show the plot

plt.show()

**Chart:**



21. Investigate the relationship between the number of living units and total finished area to understand property density.

**Code:**

# Extract the living units and total finished area columns

living\_units = df['LivingUnits']

finished\_area = df['TotalFinishedArea']

# Create a scatter plot

plt.scatter(finished\_area, living\_units, alpha=0.5)

# Set the labels and title

plt.xlabel('Total Finished Area')

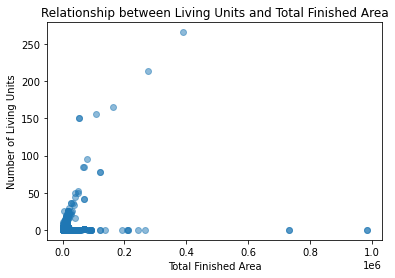
plt.ylabel('Number of Living Units')

plt.title('Relationship between Living Units and Total Finished Area')

# Show the plot

plt.show()

**Chart:**



22. Calculate the average sale price per neighborhood to identify areas with higher property values.

**Code:**

# Group by neighborhood and calculate average sale price

neighborhood\_avg\_price = df.groupby('xrPrimaryNeighborhoodID')['SalePrice'].mean()

# Sort the average prices in descending order

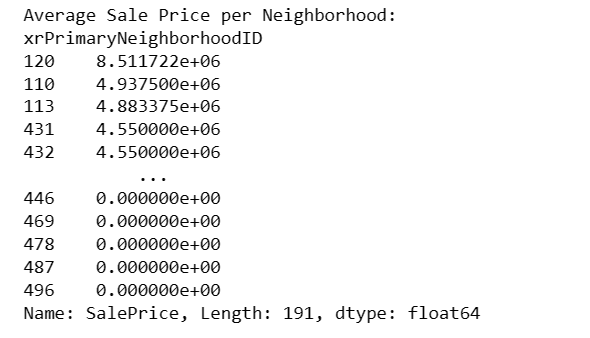
neighborhood\_avg\_price = neighborhood\_avg\_price.sort\_values(ascending=False)

# Display the average sale price per neighborhood

print("Average Sale Price per Neighborhood:")

print(neighborhood\_avg\_price)

**Output:**



23. Analyze the relationship between property ownership and sale prices to identify any patterns.

**Code:**

# Group by ownership and calculate average sale price

ownership\_avg\_price = df.groupby(['OwnerLastName', 'OwnerFirstName'])['SalePrice'].mean()

# Sort the average prices in descending order

ownership\_avg\_price = ownership\_avg\_price.sort\_values(ascending=False)

# Plot the average sale price by ownership

ownership\_avg\_price.plot(kind='bar', figsize=(10, 6))

plt.xlabel('Owner')

plt.ylabel('Average Sale Price')

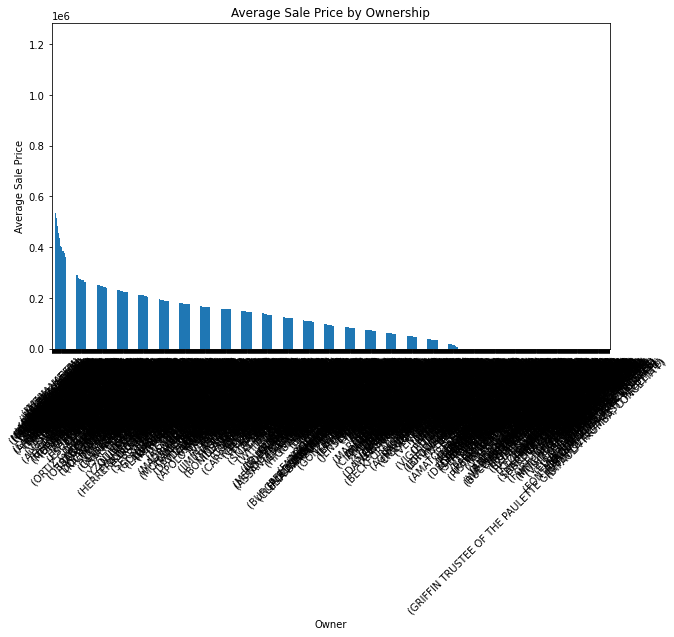
plt.title('Average Sale Price by Ownership')

plt.xticks(rotation=45)

# Display the plot

plt.show()

**Chart:**



24. Explore the distribution of sale prices based on building types to understand price variations.

**Code:**

# Create a box plot of sale prices for each building type

plt.figure(figsize=(10, 6))

sns.boxplot(data=df, x='xrBuildingTypeID', y='SalePrice')

plt.xlabel('Building Type')

plt.ylabel('Sale Price')

plt.title('Distribution of Sale Prices by Building Type')

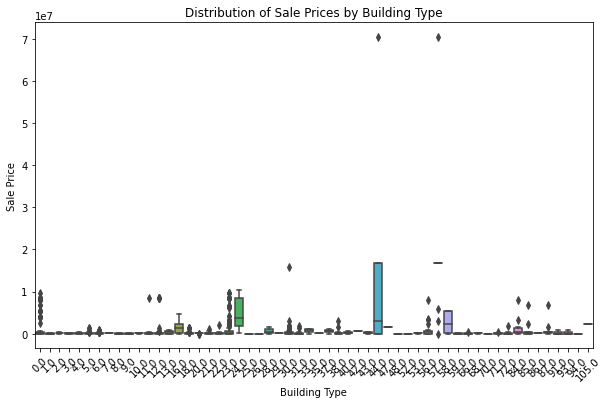
# Rotate x-axis labels if needed

plt.xticks(rotation=45)

# Display the plot

plt.show()

**Chart:**



25. Identify the most common land use types for properties with the highest sale prices.

**Code:**

# Filter the dataset for properties with highest sale prices

highest\_sale\_prices = df[df['SalePrice'] == df['SalePrice'].max()]

# Group by land use types and count occurrences

land\_use\_counts = highest\_sale\_prices.groupby('xrCompositeLandUseID').size().reset\_index(name='Count')

# Print the results

print("Most Common Land Use Types for Properties with Highest Sale Prices:")

print(land\_use\_counts)

**Output:**

Most Common Land Use Types for Properties with Highest Sale Prices:

xrCompositeLandUseID Count

1. 53 2

26. Identify any temporal patterns or trends in real estate sales based on the sale dates.

**Code:**

# Convert the SaleDate column to datetime format

df['SaleDate'] = pd.to\_datetime(df['SaleDate'])

# Extract the year and month from the SaleDate column

df['Year'] = df['SaleDate'].dt.year

df['Month'] = df['SaleDate'].dt.month

# Count the number of sales per year and month

sales\_count = df.groupby(['Year', 'Month']).size().reset\_index(name='SalesCount')

# Sort the data by year and month

sales\_count = sales\_count.sort\_values(['Year', 'Month'])

# Plot the sales count over time

plt.figure(figsize=(10, 6))

plt.plot(sales\_count['SalesCount'])

plt.xlabel('Time')

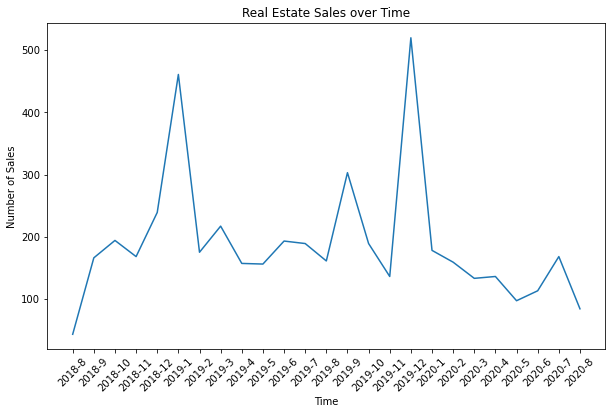
plt.ylabel('Number of Sales')

plt.title('Real Estate Sales over Time')

plt.xticks(range(len(sales\_count)), sales\_count['Year'].astype(str) + '-' + sales\_count['Month'].astype(str), rotation=45)

plt.show()

**Chart:**



**Insights:**

Based on the output, the deed types present in the dataset are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 18, 19, 22. The deed types could represent different legal classifications or types of documents involved in property transactions, such as warranty deeds, quitclaim deeds, or special warranty deeds.

The most common sales validity type is 25, with a count of 1210. This indicates that a significant number of property sales have been classified under this sales validity type.

Sales validity types 1 and 5 also have relatively high counts, with 1194 and 898 respectively. These are commonly observed in property transactions as well.

On the other hand, sales validity types such as 3, 4, 8, 18, and 19 have very low counts, ranging from 1 to 6. This suggests that these sales validity types are less frequently encountered in the dataset.

The dataset's most common number of living units is 1.0, with a count of 2494. This indicates that a significant number of properties have a single living unit. Living units with counts of 3.0 and 2.0 are also relatively common, with 766 and 598 respectively. This suggests that there are properties with multiple living units, such as apartments or multi-family dwellings.

There are a considerable number of properties with 0.0 living units, which could indicate vacant land or non-residential properties. Living units with counts of 6.0, 4.0, and 12.0 are present in smaller numbers, suggesting the existence of properties with a higher number of living units. Living units with counts of 5.0, 8.0, and 9.0 are less frequent, indicating a lower occurrence of properties with these specific numbers of living units.

Building Type 16.0: This is the most common building type with a count of 1245. Building Type 0.0: With a count of 187, this building type is relatively common as well.

Land Use Category 3: This is the most common land use category with a count of 1142. It suggests a specific type of land use that appears frequently in the dataset. Land Use Category 457: This is the second most common land use category with a count of 816. It indicates another prevalent type of land use. Land Use Category 5: With a count of 736, this land use category is also relatively common. Land Use Category 4: This category appears 544 times, suggesting its presence in the dataset. Land Use Category 7: This category has a count of 409, indicating its frequency in the dataset.

Location 15158: This location has 18 properties associated with it, indicating a relatively high concentration of properties in that area. Location 19120: With a count of 12, this location is also home to a considerable number of properties. Location 10396: Similar to the previous location, it has 12 properties associated with it. Location 10393: This location also has 12 properties, suggesting a significant presence in the dataset. Location 6130: Although relatively lower than the previous locations, it still has 8 properties associated with it.