Lab 01: Data Cleaning

Author: Nigel Nelson

Introduction:

- This lab acts as an exercise in using Pandas to interpret and clean a provided data set. Specifically, the Pandas library is used to import a data set containing information on real estate transactions for the city of Sacramento, and other surrounding cities in Sacramento. A DataFrame is created from this data set, which is then analyzed for the inferred types, and then using knowledge of the problem space more appropriate types are assigned where necessary. Next, categorical and continuous variables are visualized to gain greater insight on the entries that are contained in the data set. Useful variables are then engineered from the existing data and added to DataFrame to aid in future analysis of this data set. Finally, outliers in the data set are identified, removed, and a cleaned version of the original data set is saved to a new .csv file.

Imports:

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import re
```

1. Loading the Data and Initial Assessment

In [2]:
```
real_estate_df = pd.read_csv('Sacramentorealestatetransactions.csv')
real_estate_df.head()
```

Out[2]:

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

- What are the variables?
    - The *Sacramentorealestatetransactions* data set has 12 variables describing real estate transactions in California. This includes the **street** address, the **city**, the **zip**, the **state**, the numbers of **beds**, the number of **baths**, the square footage of residential space(**sq_ft**), the **type** of real estate, the **sale date**, the sale **price**, the **latitude**, and the **longitude**.

In [3]:
```
real_estate_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 985 entries, 0 to 984
Data columns (total 12 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   street     985 non-null    object
 1   city       985 non-null    object
 2   zip        985 non-null    int64
 3   state      985 non-null    object
 4   beds       985 non-null    int64
 5   baths      985 non-null    int64
 6   sq__ft     985 non-null    int64
 7   type       985 non-null    object
 8   sale_date  985 non-null    object
 9   price      985 non-null    int64
 10  latitude   985 non-null    float64
 11  longitude  985 non-null    float64
dtypes: float64(2), int64(5), object(5)
memory usage: 92.5+ KB
```

- What are their inferred types?
Variable Name	Inferred Type
street	object
city	object
zip	int64
state	object
beds	int64
baths	int64
sq_ft	int64
type	object
sale_date	object
price	int64
latitude	float64
longitude	float64

In [4]:
```
real_estate_df.isnull().any()
```

Out[4]:
```
street       False
city         False
zip          False
state        False
beds         False
baths        False
sq__ft       False
type         False
sale_date    False
price        False
latitude     False
longitude    False
dtype: bool
```

- Do any of the columns have null values?
  - No, none of the columns have null values.

2. Representing Categorical Variables

Counting the number of unique values for the streets, zip codes, and beds:

In [5]:
```
street_suffixes = real_estate_df['street'].apply(lambda address: address.spli
num_unique_suffixes = street_suffixes.nunique()
num_unique_zips = real_estate_df['zip'].nunique()
num_unique_beds = real_estate_df['beds'].nunique()
print(f'Unique number of street suffixes: {num_unique_suffixes}')
print(f'Unique number of zip codes: {num_unique_zips}')
print(f'Unique number of bed counts: {num_unique_beds}')
```

```
Unique number of street suffixes: 49
Unique number of zip codes: 68
Unique number of bed counts: 8
```

Converting the following variables to categorical variables: city, state, zip, beds, baths, and type:

In [6]:
```
real_estate_df['city'] = real_estate_df['city'].astype('category')
real_estate_df['state'] = real_estate_df['state'].astype('category')
real_estate_df['zip'] = real_estate_df['zip'].astype('category')
real_estate_df['beds'] = real_estate_df['beds'].astype('category')
real_estate_df['baths'] = real_estate_df['baths'].astype('category')
real_estate_df['type'] = real_estate_df['type'].astype('category')

real_estate_df.head()
```

Out[6]:

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

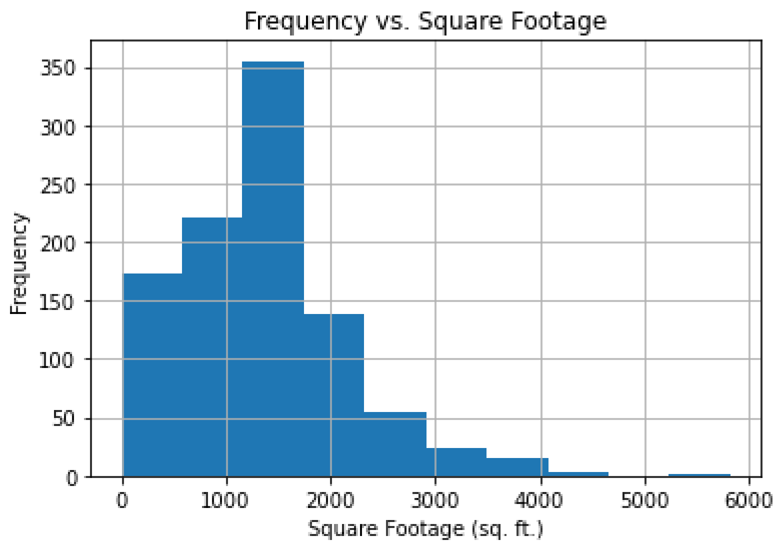
- Do you think it is more appropriate to represent these three variables as categorical or integer variables? Why or why not?
  - I think that it is more appropriate to represent the beds and baths as integer values. This is because categorizing these results in values that are less intuitive than their native integer types. By leaving these values as integers, you allow a wider array of operations to be performed on these variables, such as summing the number of beds on a street, or dividing the cost of the home by the number of baths to get an idea of how number of baths impacts the homes cost. However, converting the city, state, zip, and type to categorical is much more intuitive than representing them as integers. This is because for the city, state, and type variables, they are natively represented as strings, and they represent a finite number of categories that should not be used in numerical operations such as adding 2 street integers together, or multiplying the city by the type. Zip code is interpreted as an integer natively, however, it is better served as a categorical type because it is not an ordered value, and it does not make sense to allow greater than comparisons between zip codes, or numerical operations on zip codes such as multiplying one zip code by another.

3. Cleaning Continuous Variables

Plotting histograms of the square footage, latitudes, and longitudes:

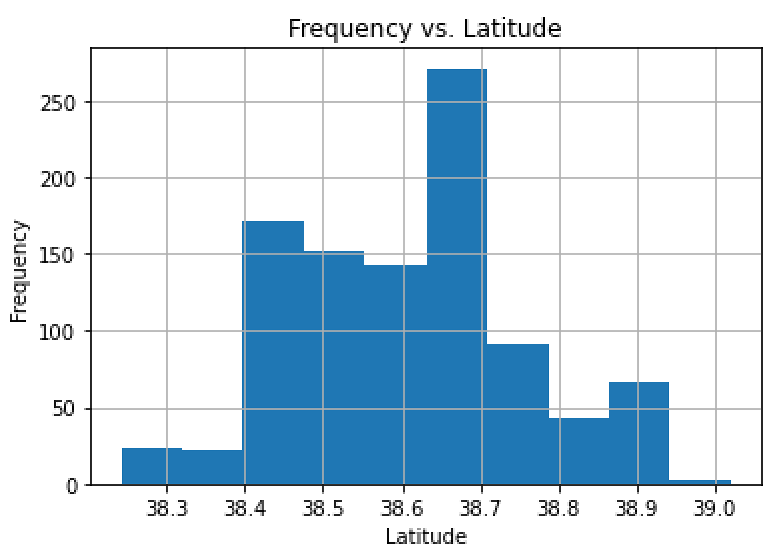
In [7]:
```
real_estate_df['sq__ft'].hist()
plt.xlabel('Square Footage (sq. ft.)')
plt.ylabel('Frequency')
plt.title('Frequency vs. Square Footage')
```

Out[7]: `Text(0.5, 1.0, 'Frequency vs. Square Footage')`



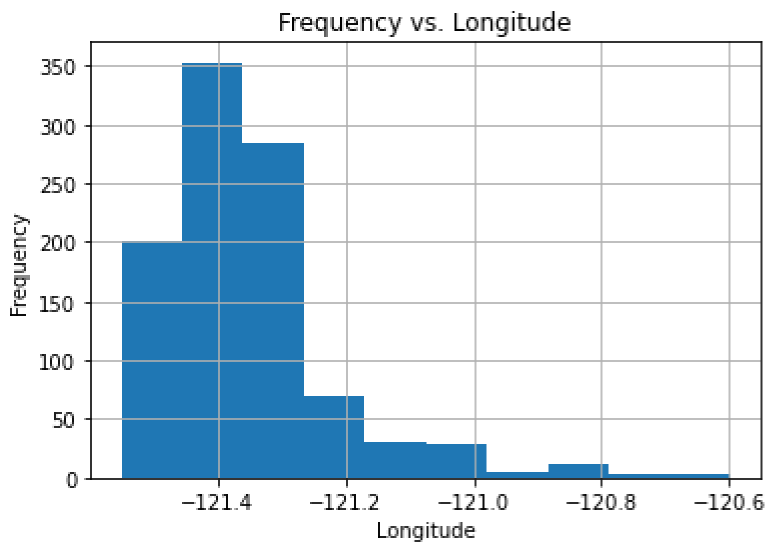
In [8]:
```
real_estate_df['latitude'].hist()
plt.xlabel('Latitude')
plt.ylabel('Frequency')
plt.title('Frequency vs. Latitude')
```

Out[8]: `Text(0.5, 1.0, 'Frequency vs. Latitude')`



In [9]:
```
real_estate_df['longitude'].hist()
plt.xlabel('Longitude')
plt.ylabel('Frequency')
plt.title('Frequency vs. Longitude')
```

Out[9]: `Text(0.5, 1.0, 'Frequency vs. Longitude')`



- Do you notice any “odd” patterns in any of the plots? Do you think they real or artifacts?
  - One odd pattern is that in the plot of square footages, there are over 150 properties that have 0 square footage. This is likely real, and can be explained by the fact that some of these properties may be empty lots, or contain exclusively buildings that don't count towards square footage, such as sheds and parking structures.

4. Cleaning Categorical Variables

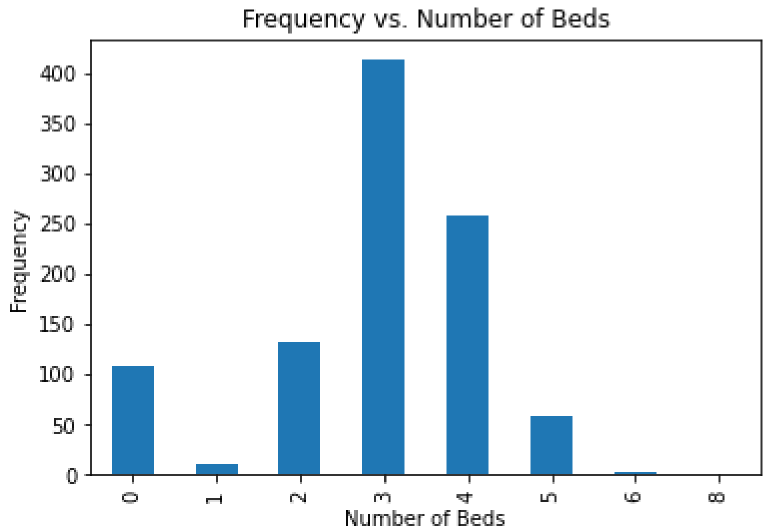
Plotting the beds, baths, type, state, city, and zip codes as count (bar) plots:

In [10]:

```
real_estate_df['beds'].value_counts(sort=False).plot.bar()
plt.xlabel('Number of Beds')
plt.ylabel('Frequency')
plt.title('Frequency vs. Number of Beds')
```

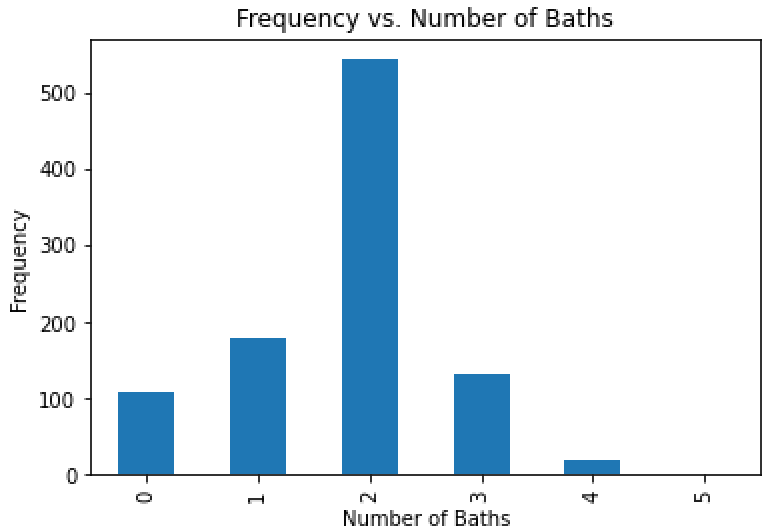
Out[10]:

```
Text(0.5, 1.0, 'Frequency vs. Number of Beds')
```



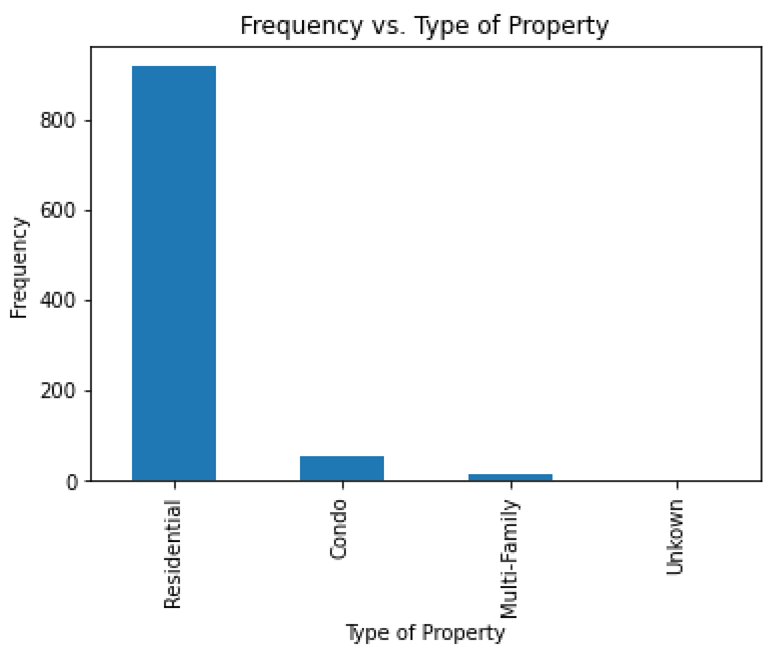
In [11]:
```
real_estate_df['baths'].value_counts(sort=False).plot.bar()
plt.xlabel('Number of Baths')
plt.ylabel('Frequency')
plt.title('Frequency vs. Number of Baths')
```

Out[11]: Text(0.5, 1.0, 'Frequency vs. Number of Baths')



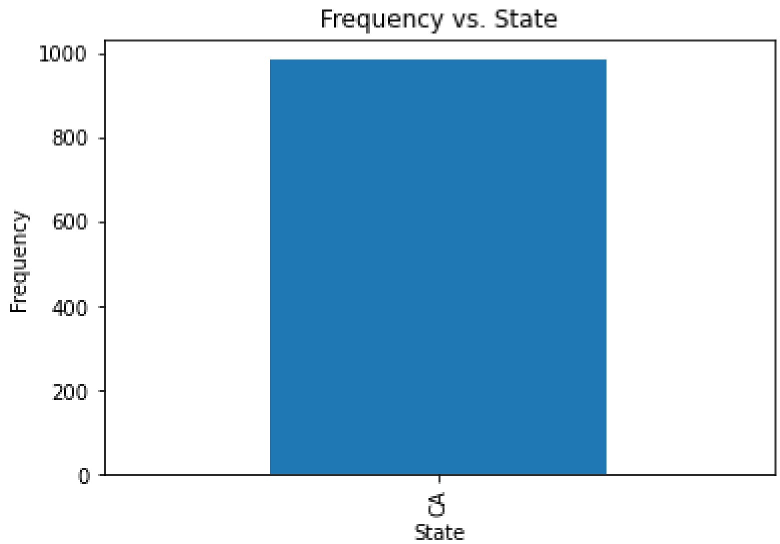
In [12]:
```
real_estate_df['type'].value_counts().plot.bar()
plt.xlabel('Type of Property')
plt.ylabel('Frequency')
plt.title('Frequency vs. Type of Property')
```

Out[12]: Text(0.5, 1.0, 'Frequency vs. Type of Property')



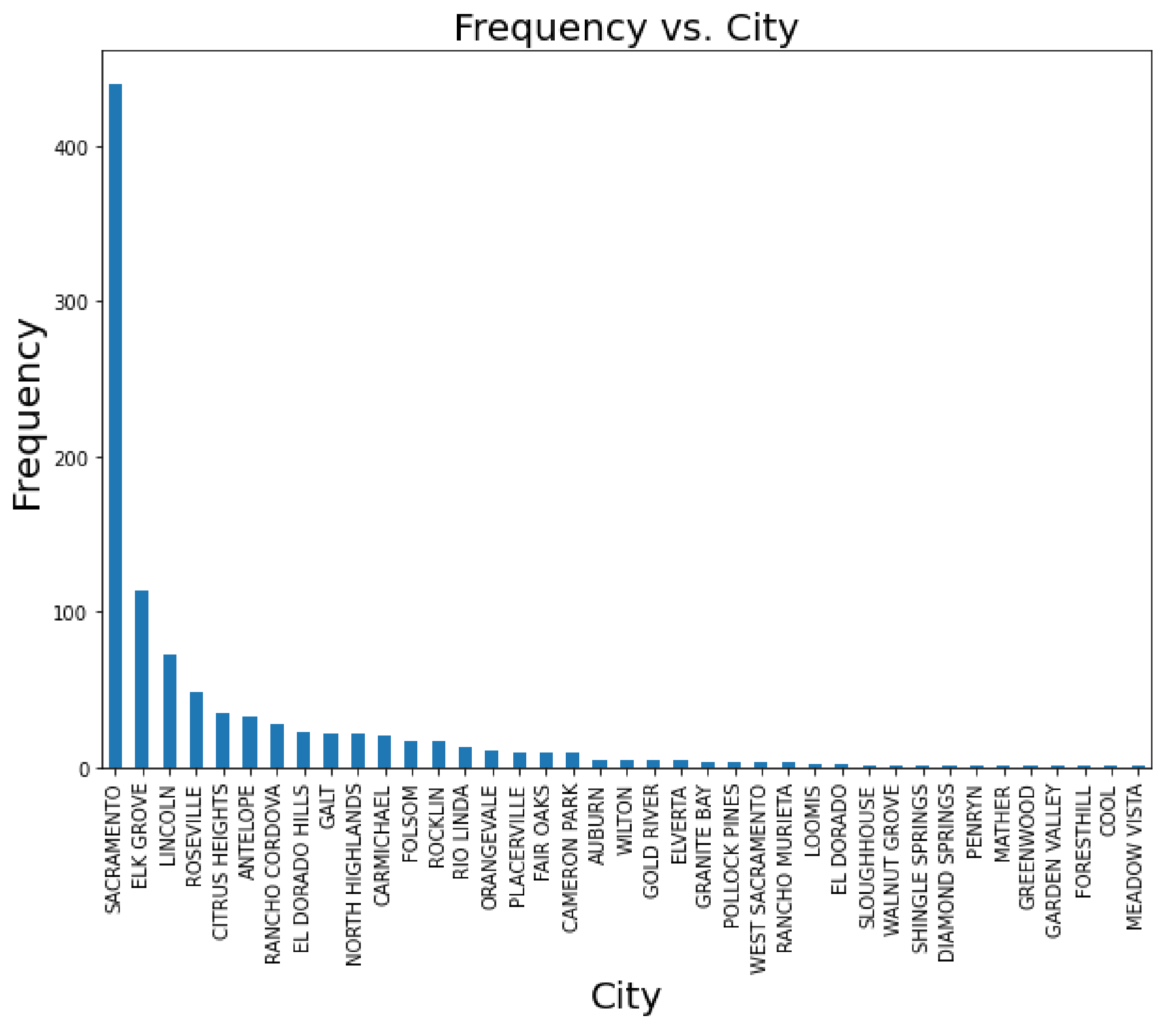
In [13]:
```
real_estate_df['state'].value_counts().plot.bar()
plt.xlabel('State')
plt.ylabel('Frequency')
plt.title('Frequency vs. State')
```

Out[13]:
```
Text(0.5, 1.0, 'Frequency vs. State')
```

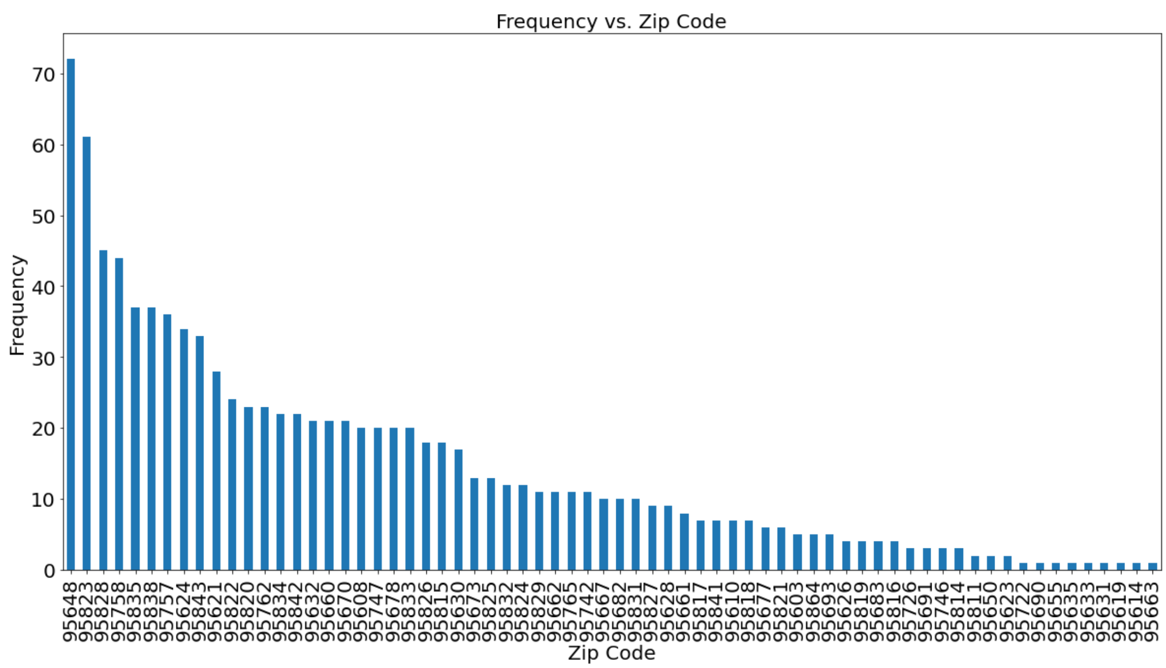


In [14]:
```
plt.figure(figsize=(10,7))
real_estate_df['city'].value_counts().plot.bar()
plt.xlabel('City', fontsize=20)
plt.ylabel('Frequency', fontsize=20)
plt.title('Frequency vs. City', fontsize=20)
```

Out[14]: Text(0.5, 1.0, 'Frequency vs. City')



In [15]:
```
plt.figure(figsize=(20,10))
plt.rc('xtick',labelsize=20)
plt.rc('ytick',labelsize=20)
real_estate_df['zip'].value_counts().plot.bar()
plt.xlabel('Zip Code', fontsize=20)
plt.ylabel('Frequency', fontsize=20)
plt.title('Frequency vs. Zip Code', fontsize=20)
plt.rc('xtick',labelsize=12)
plt.rc('ytick',labelsize=12)
```



- Do you notice any “odd” patterns in any of the plots? Do you think they real or artifacts?
  - One odd pattern was the fact that there were a considerable amount of properties without any beds or bathrooms. As mentioned in section 3, this is likely not an artifact and due to the fact that some properties sold did not contain any residential space.
  - Another odd pattern was that the property type 'Unknown' appears to have 0 transactions associated with this type. This is also not likely an artifact as this type could be useful in real estate transaction records, however this dataset happens to only contains known property types.

5. Engineering New Variables – Part I

Creating a new boolean variable called "empty_lot":

In [16]:
```
real_estate_df['empty_lot'] = real_estate_df['sq__ft'] == 0
real_estate_df.head()
```

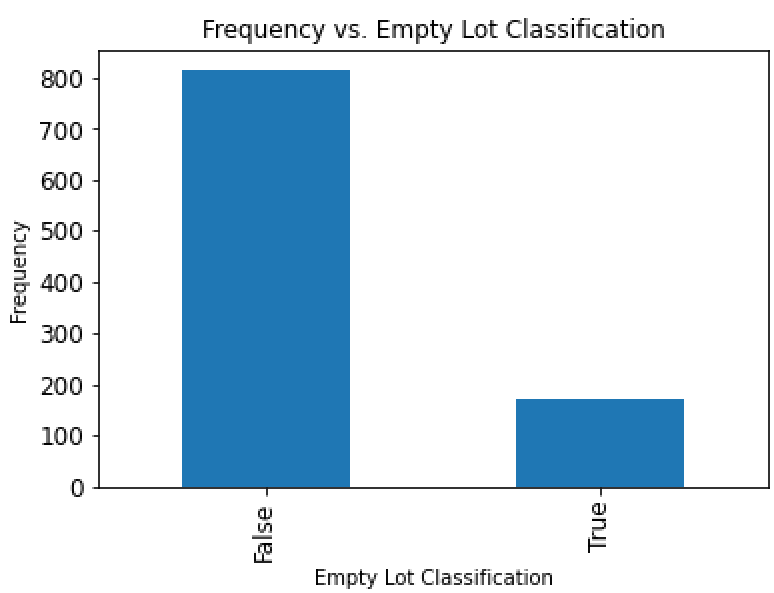
Out[16]:

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

Plotting the counts for the empty_lot variable:

In [17]:
```
real_estate_df['empty_lot'].value_counts().plot.bar()
plt.xlabel('Empty Lot Classification')
plt.ylabel('Frequency')
plt.title('Frequency vs. Empty Lot Classification')
```

Out[17]: `Text(0.5, 1.0, 'Frequency vs. Empty Lot Classification')`



6. Engineering New Variables – Part II

Counting of the number of unique values for the addresses:

In [18]:
```
nunique_address = real_estate_df['street'].nunique()
print(f'Number of unique addresses: {nunique_address}')
```

```
Number of unique addresses: 981
```

- Do you think this variable is useful for analysis or as a feature for a ML model in its current form?
  - No, this is because the street feature acts as a unique identifier the vast majority of the time. As such, in its current form this feature cannot be compared across entries to find commonalities that would allow for underlying patterns to be discovered. If this feature was included in a data set used to train a ML model, it would likely contribute to an increase in variance.

Identifying patterns in street types using the head() command:

In [19]:
```
real_estate_df['street'].head(20)
```

Out[19]:
```
0                     3526 HIGH ST
1                      51 OMAHA CT
2                   2796 BRANCH ST
3                  2805 JANETTE WAY
4                   6001 MCMAHON DR
5                5828 PEPPERMILL CT
6                6048 OGDEN NASH WAY
7                     2561 19TH AVE
8      11150 TRINITY RIVER DR Unit 114
9                     7325 10TH ST
10                 645 MORRISON AVE
11                    4085 FAWN CIR
12                  2930 LA ROSA RD
13                    2113 KIRK WAY
14               4533 LOCH HAVEN WAY
15                  7340 HAMDEN PL
16                     6715 6TH ST
17            6236 LONGFORD DR Unit 1
18                  250 PERALTA AVE
19                  113 LEEWILL AVE
Name: street, dtype: object
```

Writing a function get_street_type(address) that will return the street type (as a String) of an address:

In [20]:
```
def get_street_type(address):
    non_numeric = re.sub('\d+', '', address)
    clean_address = re.split('Unit| MARTINA', non_numeric)[0]
    street_suffix = clean_address.split()[-1]
    return street_suffix
```

Identifying number of unique street types:

In [21]:
```
street_types = real_estate_df['street'].map(lambda address: get_street_type(a
unique_streets = street_types.nunique()
print(f'Number of unique street types: {unique_streets}')
```

```
Number of unique street types: 22
```

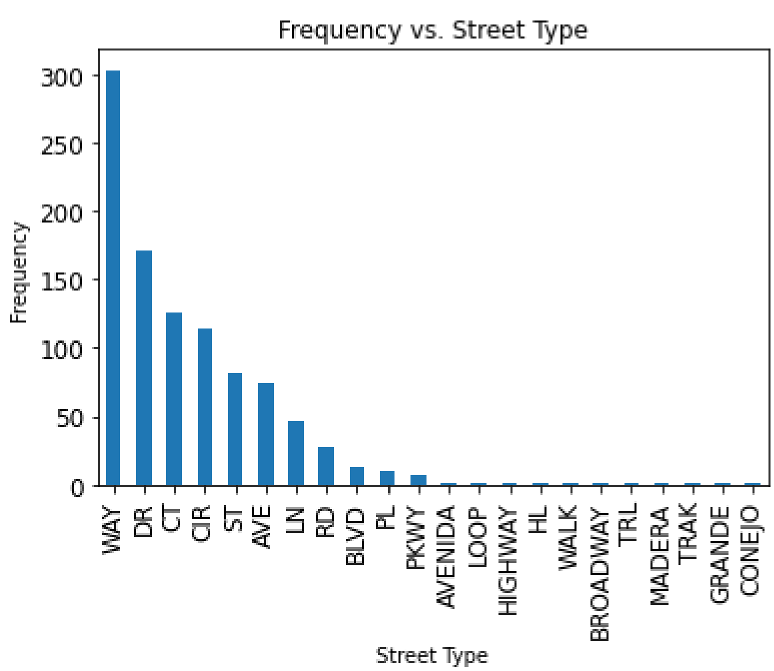
Adding street suffix column to the DataFrame:

In [22]:
```
real_estate_df['street_types'] = street_types
```

Plotting the street types as a count plot:

In [23]:
```
real_estate_df['street_types'].value_counts().plot.bar()
plt.xlabel('Street Type')
plt.ylabel('Frequency')
plt.title('Frequency vs. Street Type')
```

Out[23]:
```
Text(0.5, 1.0, 'Frequency vs. Street Type')
```



7. Identifying Potential Dependent Variables

In [24]:
```
real_estate_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 985 entries, 0 to 984
Data columns (total 14 columns):
 #   Column        Non-Null Count  Dtype   
---  ------        --------------  -----   
 0   street        985 non-null    object  
 1   city          985 non-null    category
 2   zip           985 non-null    category
 3   state         985 non-null    category
 4   beds          985 non-null    category
 5   baths         985 non-null    category
 6   sq__ft        985 non-null    int64   
 7   type          985 non-null    category
 8   sale_date     985 non-null    object  
 9   price         985 non-null    int64   
 10  latitude      985 non-null    float64 
 11  longitude     985 non-null    float64 
 12  empty_lot     985 non-null    bool    
 13  street_types  985 non-null    object  
dtypes: bool(1), category(6), float64(2), int64(2), object(3)
memory usage: 65.5+ KB
```

- What types of variables are appropriate for regression?
  - The variables that are appropriate for regression are the int and float types. Specifically a good dependent variable for a regression problem would be the predicting the price of the property. This is because a property's price is usually dependent on many of the variables contained in this data set, such as number of beds and baths, the zip code, the square footage, and even the street type.
- What types of variables are appropriate for classification?
  - The variables that are appropriate for classification are the bool and the category types. Specifically a good dependent variable for a classification problem would be predicting the number of beds a property has. This is because the number of beds could likely be inferred from combining many of the variables from this data set, such as the square footage, the number of baths, the price, and the type of real estate.

8. Identify and remove 2 outlier records from the dataset

Removing properties that have 0 square feet yet also have non-zero values for beds or baths:

- These values were deemed to be outliers because intuitively it does not make sense that a building does not have any square footage, but somehow has beds and bathrooms in this void space.

In [25]:
```
zero_sq_ft = real_estate_df.query('sq__ft == 0 & (beds != 0 | baths != 0)')
zero_sq_ft.head()
```

Out[25]:

	street	city	zip	state	beds	baths	sq__ft	type	sale_date	pr
132	3020 RICHARDSON CIR	EL DORADO HILLS	95762	CA	3	2	0	Residential	Wed May 21 00:00:00 EDT 2008	352(
154	6030 PALERMO WAY	EL DORADO HILLS	95762	CA	4	3	0	Residential	Wed May 21 00:00:00 EDT 2008	600(
155	4070 REDONDO DR	EL DORADO HILLS	95762	CA	4	3	0	Residential	Wed May 21 00:00:00 EDT 2008	606
157	315 JUMEL CT	EL DORADO HILLS	95762	CA	6	5	0	Residential	Wed May 21 00:00:00 EDT 2008	830(
223	2778 KAWEAH CT	CAMERON PARK	95682	CA	3	1	0	Residential	Tue May 20 00:00:00 EDT 2008	201(

In [26]:
```
real_estate_df = real_estate_df.drop(zero_sq_ft.index)
real_estate_df.count()
```

Out[26]:
```
street          922
city            922
zip             922
state           922
beds            922
baths           922
sq__ft          922
type            922
sale_date       922
price           922
latitude        922
longitude       922
empty_lot       922
street_types    922
dtype: int64
```

9. Saving the Cleaned Data Set

In [27]:
```
real_estate_df.to_csv('CleanedSacramentorealestatetransactions.csv')
```

Conclusion:

- This lab acted as an exercise in using Pandas to interpret and clean a provided data set. Through the use of Pandas, it was discovered that the *Sacramentorealestatetransactions* data set, contained mostly clean, usable data. Through problem space knowledge several of the inferred types of variables were changed to better represent the underlying data. In addition, new variables were added to the data set that were engineered from existing variables to better describe real estate transactions. Lastly, Pandas were used to identify outliers in the data set, which were removed to reduce any bias they may have introduced. The result of this lab was a DataFrame that represented a cleaned and optimized version of the original data set, that was exported to the *CleanedSacramentorealestatetransactions* .csv file.