Eedris_Busari_Fraugster_Assignment

November 24, 2020

1 QUESTION ONE (1)

2 DATA LOADING

[1]:
```python
# I would have required to use dask which is a parallel form of data loading if
# the size of the data were heaavier to increase time efficiciency and avoiding␣
↪loading
# all the data into the memory. An alternative is to chunk the data but it is␣
↪not as efficient,comparatively
# because of the concatenation required at the end of the chunk process.
import pandas as pd
data=pd.read_csv('realestate_fraugster_case.csv',sep=';',index_col=False)
data.head(10)
```

[1]:

	street	city	zip	state	beds	baths
0	3526 HIGH ST	SACRAMENTO	95838	CA	2.0	1.0
1	51 OMAHA CT	sacramento	95823	CA	3.0	1.0
2	2796 BRANCH ST	SACRAMENTO	95815	CA	2.0	1.0
3	2805 JANETTE WAY	SACRAMENTO	95815	CA	2.0	1.0
4	6001 MCMAHON DR	SACRAMENTO	95824	CA	2.0	1.0
5	5828 PEPPERMILL CT	SACRAMENTO	95841	CA	3.0	1.0
6	6048 OGDEN NASH WAY	SACRAMENTO	95842	CA	3.0	2.0
7	2561 19TH AVE	SACRAMENTO	95820	CA	3.0	1.0
8	11150 TRINITY RIVER DR Unit 114	RANCHO CORDOVA	95670	CA	2.0	2.0
9	7325 10TH ST	RIO LINDA	95673	CA	3.0	2.0

\

	sq__ft	type	sale_date	price	latitude	longitude
0	836.0	Residential	1943-01-09 11:56:01	59222	38.631913	-121.434879
1	1167.0	Residential	1996-11-08 23:09:38	68212	38.478902	-121.431028
2	796.0	Residential	1915-01-05 07:31:45	68880	38.618305	-121.443839
3	852.0	Residential	1998-10-22 04:46:05	69307	38.616835	-121.439146
4	797.0	Residential	1972-01-05 20:52:32	81900	38.51947	-121.435768
5	1122.0	Condo	1918-01-13 23:10:18	89921	38.662595	-121.327813
6	1104.0	Residential	1949-06-16 12:35:50	90895	38.681659	-121.351705
7	1177.0	Residential	1971-01-31 00:55:56	91002	38.535092	-121.481367
8	941.0	Condo	1955-12-30 14:44:20	94905	38.621188	-121.270555
9	1146.0	Residential	1977-06-03 09:55:18	98937	38.700909	-121.442979

.

3 QUESTION TWO (2)

4 DATA CLEANING

The data cleaning steps would be done in three phases as:

5 PHASE 1: THE GENERAL OUTLOOK AND PROFILE OF THE DATASET

6 (a) Statistical Description

The "describe" method of panda's dataframe gives the statistical description of the dataset.This helps to see the count of unique values,most frequent value,how the values deviate or vary from one another percentile, among others.

[2]:
```
data.describe(include='all')
```

[2]:

	street	city	zip	state	beds	baths
count	985	985	985	985	985.000000	985.000000
unique	981	42	69	3	NaN	NaN
top	7 CRYSTALWOOD CIR	SACRAMENTO	95648	CA	NaN	NaN
freq	2	437	72	983	NaN	NaN
mean	NaN	NaN	NaN	NaN	2.911675	1.776650
std	NaN	NaN	NaN	NaN	1.307932	0.895371
min	NaN	NaN	NaN	NaN	0.000000	0.000000
25%	NaN	NaN	NaN	NaN	2.000000	1.000000
50%	NaN	NaN	NaN	NaN	3.000000	2.000000
75%	NaN	NaN	NaN	NaN	4.000000	2.000000
max	NaN	NaN	NaN	NaN	8.000000	5.000000

	sq__ft	type	sale_date	price	latitude
count	985.000000	985	986	985	985
unique	NaN	7	986	606	969
top	NaN	Residential	1999-05-22 19:42:16	4897	38.423251
freq	NaN	914	1	49	5
mean	1314.916751	NaN	NaN	NaN	NaN
std	853.048243	NaN	NaN	NaN	NaN
min	0.000000	NaN	NaN	NaN	NaN
25%	952.000000	NaN	NaN	NaN	NaN
50%	1304.000000	NaN	NaN	NaN	NaN
75%	1718.000000	NaN	NaN	NaN	NaN
max	5822.000000	NaN	NaN	NaN	NaN

longitude

```
count           985
unique          967
top      -121.444489
freq              5
mean            NaN
std             NaN
min             NaN
25%             NaN
50%             NaN
75%             NaN
max             NaN
```

[3]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 986 entries, 0 to 985
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    object
 3   state      985 non-null    object
 4   beds       985 non-null    float64
 5   baths      985 non-null    float64
 6   sq__ft     985 non-null    float64
 7   type       985 non-null    object
 8   sale_date  986 non-null    object
 9   price      985 non-null    object
 10  latitude   985 non-null    object
 11  longitude  985 non-null    object
dtypes: float64(3), object(9)
memory usage: 92.6+ KB
```

7 (b) Data Type Formats

When trying to convert to specific datatypes, the rows that do not comply to the rules of this datatype are identified as errors.These would help in making suitable corrections on the identified observations.

Also,possible operations on the columns depend on the datatype.The correct datatypes would also help to identify errors in the columns.In this section, emphasis would be made on the numeric columns while the non-numeric features would form the basis for the Inconsistency check in phase two

The above information could help determine the need for type conversion The columns with 'object' datatypes need to be investigated to determine which ones would require conversion

8 i) The 'city','state', 'street' and 'type' object columns are non-numeric values

The 'city','state', and 'type' look tempting to convert to the category dtypes for memory efficiency and optimization. However, they would be left as object because the dataset is not large enough to cause memory issues.Also, if converted to category dtype, the addition of new distinct value into the columns would generate 'NaN' error.

9 ii) The 'sale_date' column being a date would be converted to date datatype.

data['sale_date'] = pd.to_datetime(data.sale_date, format='%Y-%m-%d %H:%M:%S')

data['sale_date'] = pd.to_datetime(data.sale_date, format='%Y-%m-%d %H:%M:%S')

Running the above line gives errors such as the one identified below

ValueError: time data 1917-07-24 08:12:24% doesn't match format specified

[4]:
```
# The error causing rows were identified and corrected as follows
data["sale_date"].replace({"2013-12-19 04:05:22A": "2013-12-19 04:05:22",␣
 ↪"1917-07-24 08:12:24%":"1917-07-24 08:12:24","1918-02-25 20:36:13&":
 ↪"1918-02-25 20:36:13"}, inplace=True)
```

10 iii) The 'zip' and 'price' object columns have numeric values. These are supposed to be integer values.This is checked and the rows with errors are identified

[5]:
```
# The error causing rows were identified in the zip column
for j, value in enumerate(data['zip']):
   try:
      int(value)
   except ValueError:
      print('The identified error index {}: {!r}'.format(j, value))
```

```
The identified error index 30: '957f58'
The identified error index 985: nan
```

[6]:
```
# The error causing rows were identified in the price column
for j, value in enumerate(data['price']):
   try:
      int(value)
   except ValueError:
      print('The identified error index {}: {!r}'.format(j, value))
```

```
The identified error index 115: '298000D'
The identified error index 985: nan
```

[7]:
```
# The typographical error were corrected intuitively as follows
data["zip"].replace({"957f58": "95758"}, inplace=True)
data["price"].replace({"298000D": "298000"}, inplace=True)
```

11 iv) The 'longitude' and 'latitude' object columns have floating values. These are checked and the rows with errors identified

[8]:
```
for j, value in enumerate(data['longitude']):
   try:
      float(value)
   except ValueError:
      print('Index error for Longitude  {}: {!r}'.format(j, value))
```

```
Index error for Longitude  121: '-121.2286RT'
Index error for Longitude  147: '-121.363757$'
```

[9]:
```
for j, value in enumerate(data['latitude']):
   try:
      float(value)
   except ValueError:
    print('Index error for Latitude  {}: {!r}'.format(j, value))
```

```
Index error for Latitude  109: '38.410992C'
```

[10]:
```
# The typographical error were replaced intuitively as follows
data["longitude"].replace({"-121.2286RT": "-121.228678","-121.363757$": "-121.
 ↪363757"}, inplace=True)
data["latitude"].replace({"38.410992C": "38.410992"}, inplace=True)
```

[11]:
```
#data = data.astype({'longitude': 'float64', 'latitude': 'float64','price':
 ↪'int64','zip':'int64'})
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 986 entries, 0 to 985
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    object
 3   state      985 non-null    object
 4   beds       985 non-null    float64
 5   baths      985 non-null    float64
 6   sq__ft     985 non-null    float64
 7   type       985 non-null    object
```

```
 8   sale_date  986 non-null    object
 9   price      985 non-null    object
 10  latitude   985 non-null    object
 11  longitude  985 non-null    object
dtypes: float64(3), object(9)
memory usage: 92.6+ KB
```

12 PHASE 2: THE INCONSISTENCY CHECK

(The data consistency check is used for the following:

Redundancy such as duplicates,irrelevant datapoints, format error among others in both the columns and the rows

To do this, we check the consistency of non-numeric features (type, state, city and street) by:

(i) Capitalization Consistency Check

[12]:
```
#The solution to the inconsistency in the case format (lower and upper cases)␣
↪can be solved by either making all the letters
# The upper case would be used in this case
data= data.apply(lambda x: x.astype(str).str.upper() if x.name in ['street',␣
↪'type','city','state'] else x)
data
```

[12]:
```
                 street              city    zip state  beds  baths  sq__ft  \
0          3526 HIGH ST        SACRAMENTO  95838    CA   2.0    1.0   836.0
1           51 OMAHA CT        SACRAMENTO  95823    CA   3.0    1.0  1167.0
2         2796 BRANCH ST        SACRAMENTO  95815    CA   2.0    1.0   796.0
3       2805 JANETTE WAY        SACRAMENTO  95815    CA   2.0    1.0   852.0
4        6001 MCMAHON DR        SACRAMENTO  95824    CA   2.0    1.0   797.0
..                  …                 …      …     …     …      …       …
981      6932 RUSKUT WAY        SACRAMENTO  95823    CA   3.0    2.0  1477.0
982    7933 DAFFODIL WAY    CITRUS HEIGHTS  95610    CA   3.0    2.0  1216.0
983      8304 RED FOX WAY         ELK GROVE  95758    CA   4.0    2.0  1685.0
984  3882 YELLOWSTONE LN   EL DORADO HILLS  95762    CA   3.0    2.0  1362.0
985                  NAN               NAN    NaN   NAN   NaN    NaN     NaN

            type            sale_date   price   latitude    longitude
0    RESIDENTIAL  1943-01-09 11:56:01   59222  38.631913  -121.434879
1    RESIDENTIAL  1996-11-08 23:09:38   68212  38.478902  -121.431028
2    RESIDENTIAL  1915-01-05 07:31:45   68880  38.618305  -121.443839
3    RESIDENTIAL  1998-10-22 04:46:05   69307  38.616835  -121.439146
4    RESIDENTIAL  1972-01-05 20:52:32   81900   38.51947  -121.435768
..           …                    …       …         …            …
981  RESIDENTIAL  1955-09-26 15:13:23  234000  38.499893   -121.45889
982  RESIDENTIAL  1950-09-13 20:59:20  235000  38.708824  -121.256803
983  RESIDENTIAL  1933-04-10 20:13:38  235301     38.417  -121.397424
```

```
984  RESIDENTIAL  1934-06-05 04:16:37  235738  38.655245  -121.075915
985          NAN  1940-02-11 05:29:17     NaN        NaN          NaN

[986 rows x 12 columns]
```

(ii) Duplicate Row Check

[13]:
```python
# Duplicate row check would result into repetition with no new information in
↪the dataset.
# Therefore, observations that have been earlier recorded should be deleted. It
↪could happen as a result of double submission
# file merging, among others
data.drop_duplicates(inplace=True)
data
```

[13]:
```
                  street              city    zip state  beds  baths  sq__ft  \
0           3526 HIGH ST        SACRAMENTO  95838    CA   2.0    1.0   836.0
1            51 OMAHA CT        SACRAMENTO  95823    CA   3.0    1.0  1167.0
2         2796 BRANCH ST        SACRAMENTO  95815    CA   2.0    1.0   796.0
3       2805 JANETTE WAY        SACRAMENTO  95815    CA   2.0    1.0   852.0
4        6001 MCMAHON DR        SACRAMENTO  95824    CA   2.0    1.0   797.0
..                   …                 …     …    …    …      …
981      6932 RUSKUT WAY        SACRAMENTO  95823    CA   3.0    2.0  1477.0
982    7933 DAFFODIL WAY    CITRUS HEIGHTS  95610    CA   3.0    2.0  1216.0
983      8304 RED FOX WAY         ELK GROVE  95758    CA   4.0    2.0  1685.0
984  3882 YELLOWSTONE LN  EL DORADO HILLS  95762    CA   3.0    2.0  1362.0
985                  NAN               NAN    NaN   NAN   NaN    NaN     NaN

            type            sale_date   price   latitude   longitude
0    RESIDENTIAL  1943-01-09 11:56:01   59222  38.631913 -121.434879
1    RESIDENTIAL  1996-11-08 23:09:38   68212  38.478902 -121.431028
2    RESIDENTIAL  1915-01-05 07:31:45   68880  38.618305 -121.443839
3    RESIDENTIAL  1998-10-22 04:46:05   69307  38.616835 -121.439146
4    RESIDENTIAL  1972-01-05 20:52:32   81900   38.51947 -121.435768
..           …                    …      …          …           …
981  RESIDENTIAL  1955-09-26 15:13:23  234000  38.499893  -121.45889
982  RESIDENTIAL  1950-09-13 20:59:20  235000  38.708824 -121.256803
983  RESIDENTIAL  1933-04-10 20:13:38  235301     38.417 -121.397424
984  RESIDENTIAL  1934-06-05 04:16:37  235738  38.655245 -121.075915
985          NAN  1940-02-11 05:29:17     NaN        NaN          NaN

[986 rows x 12 columns]
```

(iii) Irrelevant/Redundant Row Check

[14]:
```python
# Since, it is a real estate sales data. Some columns could be seen as unique
↪identifiers.
```

```python
# Unavailability or missingness of this identifiers would render the␣
↪observation(row) redundant
# An identifier here would be the Longitude and Lattitude.
# This is because the house/bed/baths sold would not be identified without this␣
↪information.
# Therefore, rows with this missing values should be removed
import numpy as np
data = data.dropna(axis=0, subset=['longitude','latitude'])
data
```

[14]:

```
                  street              city    zip state  beds  baths  sq__ft  \
0           3526 HIGH ST        SACRAMENTO  95838    CA   2.0    1.0   836.0
1            51 OMAHA CT        SACRAMENTO  95823    CA   3.0    1.0  1167.0
2         2796 BRANCH ST        SACRAMENTO  95815    CA   2.0    1.0   796.0
3       2805 JANETTE WAY        SACRAMENTO  95815    CA   2.0    1.0   852.0
4        6001 MCMAHON DR        SACRAMENTO  95824    CA   2.0    1.0   797.0
..                   …                 …      …     …     …      …       …
980   9169 GARLINGTON CT        SACRAMENTO  95829    CA   4.0    3.0  2280.0
981      6932 RUSKUT WAY        SACRAMENTO  95823    CA   3.0    2.0  1477.0
982    7933 DAFFODIL WAY    CITRUS HEIGHTS  95610    CA   3.0    2.0  1216.0
983      8304 RED FOX WAY         ELK GROVE  95758    CA   4.0    2.0  1685.0
984  3882 YELLOWSTONE LN   EL DORADO HILLS  95762    CA   3.0    2.0  1362.0

            type            sale_date   price   latitude   longitude
0    RESIDENTIAL  1943-01-09 11:56:01   59222  38.631913 -121.434879
1    RESIDENTIAL  1996-11-08 23:09:38   68212  38.478902 -121.431028
2    RESIDENTIAL  1915-01-05 07:31:45   68880  38.618305 -121.443839
3    RESIDENTIAL  1998-10-22 04:46:05   69307  38.616835 -121.439146
4    RESIDENTIAL  1972-01-05 20:52:32   81900   38.51947 -121.435768
..           …                    …       …          …           …
980  RESIDENTIAL  1951-06-03 12:20:20  232425  38.457679  -121.35962
981  RESIDENTIAL  1955-09-26 15:13:23  234000  38.499893  -121.45889
982  RESIDENTIAL  1950-09-13 20:59:20  235000  38.708824 -121.256803
983  RESIDENTIAL  1933-04-10 20:13:38  235301     38.417 -121.397424
984  RESIDENTIAL  1934-06-05 04:16:37  235738  38.655245 -121.075915

[985 rows x 12 columns]
```

(iv) Typographical and Format Errors

The unique values of the non-numeric columns (‘type’,‘state’,‘city’, and ‘street’) as shown in Out[2]: above are free text, which is prone to typographical error and human discretion in its format used. A look at the unique values show these errors.

As can be seen in the state column, there are typographical error as ‘CA’, ‘CA3’, ‘CA-’ is pointing to a singular state ‘CA’.

[1] The solution to the ‘states’ column can be either of:

a) Delete the column since it is a single-valued column and would not help in any ML modelling task.

b) Correct the spelling and typo-errors.

For completeness of the dataset, I will just replace the values with 'CA'

[15]:
```
# Check the unique values in the 'state' column and also save a copy of the␣
↪data with a new name
print(data.state.unique())
new_data=data.copy()
```

```
['CA' 'CA3' 'CA-']
```

[16]:
```
#new_data.loc[new_data['state'] == 'CA']
new_data=data.loc[data['state'] == 'CA']
new_data.state.unique()
```

[16]:
```
array(['CA'], dtype=object)
```

[2] The solution to the 'type' column:

[31]:
```
#The unique values in the type column are replaced appropriately
new_data.type.unique()
new_data["type"].replace({"RESIDENTIAL%": "RESIDENTIAL","RESIDEN_TIAL":␣
↪"RESIDENTIAL","RESIDENTIAL)": "RESIDENTIAL"}, inplace=True)
new_data.type.unique()
```

[31]:
```
array(['RESIDENTIAL', 'CONDO', 'MULTI-FAMILY', 'UNKOWN'], dtype=object)
```

[3] The solution to the 'city' column:

[18]:
```
# To check the count and unique values in the column
print(new_data.city.nunique())
new_data.city.unique()
```

```
41
```

[18]:
```
array(['SACRAMENTO', 'RANCHO CORDOVA', 'RIO LINDA', 'CITRUS HEIGHTS',
       'NORTH HIGHLANDS', 'ANTELOPE', 'SACRAMENTO@', 'ELK GROVE',
       'ELVERTA', 'GALT', 'CARMICHAEL', 'ORANGEVALE', 'FOLSOM',
       'ELK GROVE<>', 'MATHER', 'POLLOCK PINES', 'GOLD RIVER',
       'EL DORADO HILLS', 'RANCHO MURIETA', 'WILTON', 'GREENWOOD',
       'FAIR OAKS', 'CAMERON PARK', 'LINCOLN', 'PLACERVILLE',
       'MEADOW VISTA', 'ROSEVILLE', 'ROCKLIN', 'AUBURN', 'LOOMIS',
       'EL DORADO', 'PENRYN', 'GRANITE BAY', 'FORESTHILL',
       'DIAMOND SPRINGS', 'SHINGLE SPRINGS', 'COOL', 'WALNUT GROVE',
       'GARDEN VALLEY', 'SLOUGHHOUSE', 'WEST SACRAMENTO'], dtype=object)
```

[19]:
```python
# One way to do this is to create a list of valid cities in California
# Then, check the "city" column with this list.
# Any value that is present in the 'city' column but not available in the actual
# city list would be investigated
actual_city=['SACRAMENTO', 'RANCHO CORDOVA', 'RIO LINDA', 'CITRUS HEIGHTS',
       'NORTH HIGHLANDS', 'ANTELOPE', 'ELK GROVE',
       'ELVERTA', 'GALT', 'CARMICHAEL', 'ORANGEVALE', 'FOLSOM',
       'ELK GROVE', 'MATHER', 'POLLOCK PINES', 'GOLD RIVER',
       'EL DORADO HILLS', 'RANCHO MURIETA', 'WILTON', 'GREENWOOD',
       'FAIR OAKS', 'CAMERON PARK', 'LINCOLN', 'PLACERVILLE',
       'MEADOW VISTA', 'ROSEVILLE', 'ROCKLIN', 'AUBURN', 'LOOMIS',
       'EL DORADO', 'PENRYN', 'GRANITE BAY', 'FORESTHILL',
       'DIAMOND SPRINGS', 'SHINGLE SPRINGS', 'COOL', 'WALNUT GROVE',
       'GARDEN VALLEY', 'SLOUGHHOUSE', 'WEST SACRAMENTO']
check_this= new_data[~new_data.city.isin(actual_city)].city
check_this
```

[19]:
```
28    SACRAMENTO@
76    ELK GROVE<>
Name: city, dtype: object
```

[20]:
```python
#The unique values in the type column are replaced appropriately
new_data["city"].replace({"SACRAMENTO@": "SACRAMENTO","ELK GROVE<>": "ELK␣
 ↪GROVE"}, inplace=True)
print(new_data.city.nunique())
new_data.city.unique()
```

```
39
```

[20]:
```
array(['SACRAMENTO', 'RANCHO CORDOVA', 'RIO LINDA', 'CITRUS HEIGHTS',
       'NORTH HIGHLANDS', 'ANTELOPE', 'ELK GROVE', 'ELVERTA', 'GALT',
       'CARMICHAEL', 'ORANGEVALE', 'FOLSOM', 'MATHER', 'POLLOCK PINES',
       'GOLD RIVER', 'EL DORADO HILLS', 'RANCHO MURIETA', 'WILTON',
       'GREENWOOD', 'FAIR OAKS', 'CAMERON PARK', 'LINCOLN', 'PLACERVILLE',
       'MEADOW VISTA', 'ROSEVILLE', 'ROCKLIN', 'AUBURN', 'LOOMIS',
       'EL DORADO', 'PENRYN', 'GRANITE BAY', 'FORESTHILL',
       'DIAMOND SPRINGS', 'SHINGLE SPRINGS', 'COOL', 'WALNUT GROVE',
       'GARDEN VALLEY', 'SLOUGHHOUSE', 'WEST SACRAMENTO'], dtype=object)
```

[32]:
```python
# Other possible typo-error that can be checked are whitespace, fullstop, among␣
 ↪others
new_data['city'] = new_data['city'].str.strip() # delete whitespace.
new_data['city'] = new_data['city'].str.replace('\\.', '') # delete dot/full␣
 ↪stop.
print(new_data.city.nunique())
new_data.city.unique()
```

39

[32]:
```
array(['SACRAMENTO', 'RANCHO CORDOVA', 'RIO LINDA', 'CITRUS HEIGHTS',
       'NORTH HIGHLANDS', 'ANTELOPE', 'ELK GROVE', 'ELVERTA', 'GALT',
       'CARMICHAEL', 'ORANGEVALE', 'FOLSOM', 'MATHER', 'POLLOCK PINES',
       'GOLD RIVER', 'EL DORADO HILLS', 'RANCHO MURIETA', 'WILTON',
       'GREENWOOD', 'FAIR OAKS', 'CAMERON PARK', 'LINCOLN', 'PLACERVILLE',
       'MEADOW VISTA', 'ROSEVILLE', 'ROCKLIN', 'AUBURN', 'LOOMIS',
       'EL DORADO', 'PENRYN', 'GRANITE BAY', 'FORESTHILL',
       'DIAMOND SPRINGS', 'SHINGLE SPRINGS', 'COOL', 'WALNUT GROVE',
       'GARDEN VALLEY', 'SLOUGHHOUSE', 'WEST SACRAMENTO'], dtype=object)
```

[4] The solution to the 'street' column:

[22]:
```python
# To check the count of unique values in the column
new_data.street.nunique()
```

[22]:
```
979
```

[33]:
```python
# There is actually less to e done here because the unique values almost equal␣
↪the number of observations
# Therefore,one way to clean the data is to emove blanks,dots,abbreviate some␣
↪words, etc
new_data['street'] = new_data['street'].str.strip() # delete blankspaces
new_data['street'] = new_data['street'].str.replace('\\.', '') # delete dot/
↪full stop.
print(new_data.street.nunique())
```

```
979
```

[24]:
```python
#changing the datatypes after the corrections have been effected
datatype= {'price': int, 'zip': int,'longitude':float,'latitude':float}
new_data = new_data.astype(datatype)
new_data['sale_date'] =  pd.to_datetime(new_data.sale_date, format='%Y-%m-%d %H:
↪%M:%S')
print(new_data.dtypes)
```

```
street                 object
city                   object
zip                     int32
state                  object
beds                  float64
baths                 float64
sq__ft                float64
type                   object
sale_date      datetime64[ns]
price                   int32
latitude              float64
```

```
longitude           float64
dtype: object
```

13 PHASE 3: HANDLING THE MISSING VALUES

[25]:
```
new_data.isnull().values.any()
```

[25]:
```
False
```

There are no missing values in the refined data. However, there are 'zero' valued cells which could also mean that the missing values have been replaced with zero.If the zero values actually represent missing values. Then, there are a number of ways to handle this:

(i) Single-Value Imputation(SI) which involves replacing the missing cells with a single value. It could be the mean,highest occuring values,among others.

(ii) Multiple/Multivariate Imputation(MI) which involves the use of different values to replace the missing cell based on the distribution of the data. There are several state of the art methods to do this.

My master thesis research was based on Classification with data irregularities (missing values and class imbalance).I implemented and compared different sota imputation algorithms such as Generative Adversarial Network (GAN) for building prediction. This could be a good alternatives to handling the missing values. The link to my thesis can be found here https://github.com/busarieedris/Classification-with-Data-Irregularities (There may be some restrictions on some data due to privacy concerns.It is a collaborative research with a foremost research institute in Germany)

.

14 QUESTION THREE (3)

15 DATA SAVING

[26]:
```
# Save the cleaned data with a better interactive name. This can be done with␣
↪the '.to_csv' command
# But the instruction says 'write a new csv with a similar name with the␣
↪cleaned data'.That is the reason for changing the cleaned data
# with a better name first.
clean_realestate_fraugster_case=new_data.copy()
clean_realestate_fraugster_case.to_csv('clean_realestate_fraugster_case.
↪csv',index=False,sep=';')
```

[27]:
```
clean_realestate_fraugster_case.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 983 entries, 0 to 984
Data columns (total 12 columns):
```

```
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   street     983 non-null    object
 1   city       983 non-null    object
 2   zip        983 non-null    int32
 3   state      983 non-null    object
 4   beds       983 non-null    float64
 5   baths      983 non-null    float64
 6   sq__ft     983 non-null    float64
 7   type       983 non-null    object
 8   sale_date  983 non-null    datetime64[ns]
 9   price      983 non-null    int32
 10  latitude   983 non-null    float64
 11  longitude  983 non-null    float64
dtypes: datetime64[ns](1), float64(5), int32(2), object(4)
memory usage: 92.2+ KB
```

.

16 QUESTION FOUR (4)

17 OUTPUT THE FOLLOWING ANSWERS:

(A) what is the distance (in meters) between the cheapest sale and the most recent sale?

SOLUTION / APPROACH

To do this:

STEP 1: You need the location (Longitude and Latitude) of the two points (The cheapest sale and the most recent sale).

[28]:
```
# LET X BE THE CHEAPEST SALE (i.e The least value in the 'price' column)
lon_x=new_data.loc[new_data['price'].idxmin()]['longitude'] # The corresponding␣
 ↪longitude for X
lat_x=new_data.loc[new_data['price'].idxmin()]['latitude']  # The corresponding␣
 ↪latitude for X
```

[29]:
```
# LET Y REPRESENT THE MOST RECENT SALE (i.e The most recent date in the␣
 ↪'sale_date' column)
lon_y=new_data.loc[new_data.sale_date.idxmax()]['longitude'] # The␣
 ↪corresponding longitude for the most recent sale
lat_y=new_data.loc[new_data.sale_date.idxmax()]['latitude']  # The␣
 ↪corresponding latitude for the most recent sale
```

STEP 2: Calculate the difference in distance between these two points

In order to get the distance between two coordinate points, there are quite some formulars for such calculations with varying degree of accuracy.

Some of the methods are:

1) Haversine formula: It is used to determine the distance between two points based on the law of Haversine.
2) Vincenty Formula: It is a distance calculation based on the fact that the earth is oblate spherical.It has an accuracy of almost 1mm

Step (i): Converting the trigonometrical values of the longitude and latitude into radian.

Step (ii): Find the difference in the coordinates.

Step (iii): Use one of the formulars above to calculate the distance between two points.

[34]:
```
import math
from math import sin, cos, sqrt, atan2, radians
R = 6373.0 # Mean Radius of the Earth

# Step(i) Converting the trigonometrical values of the longitude and latitude␣
↪into radian.
lat_x_rad = math.radians(lat_x)
lon_x_rad= math.radians(lon_x)
lat_y_rad = math.radians(lat_y)
lon_y_rad= math.radians(lon_y)

# Step(ii) Find the difference in the coordinates.
diff_lon = lon_y_rad - lon_x_rad
diff_lat = lat_y_rad - lat_x_rad

# Step(iii)  For the purpose of this assignment,the Haversine formula would be␣
↪used.
# Using Haversine formula to calculate the distance between two points.
a = math.sin(diff_lat / 2)**2 + math.cos(lat_x_rad) * math.cos(lat_y_rad) *␣
↪math.sin(diff_lon / 2)**2
c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))
dist = R * c

print("The distance (in meters) between the cheapest sale and the most recent␣
↪sale:", dist* 1000, 'metres')
```

```
The distance (in meters) between the cheapest sale and the most recent sale:
6233.4089687788455 metres
```

.

(B) what is the median street number, in multi-family houses, sold between 05/11/1933 and 03/12/1998 , in Sacramento?

SOLUTION / APPROACH

To do this:

STEP 1: Filter out the rows with ‘city= SACRAMENTO’ and ‘type= MULTI-FAMILY’

[35]:
```python
# Filter out the rows with 'city= SACRAMENTO' and 'type= MULTI-FAMILY' ,
data_add=new_data[(new_data['type']=='MULTI-FAMILY') &␣
↪(new_data['city']=='SACRAMENTO')]
data_add
```

[35]:
```
                street        city    zip state  beds  baths   sq__ft  \
56      4520 BOMARK WAY  SACRAMENTO  95842    CA   4.0    2.0  1943.0
68        7624 BOGEY CT  SACRAMENTO  95828    CA   4.0    4.0  2162.0
108    2912 NORCADE CIR  SACRAMENTO  95826    CA   8.0    4.0  3612.0
113  10158 CRAWFORD WAY  SACRAMENTO  95827    CA   4.0    4.0  2213.0
353  2820 DEL PASO BLVD  SACRAMENTO  95815    CA   4.0    2.0  1404.0
366         7342 GIGI PL  SACRAMENTO  95828    CA   4.0    4.0  1995.0
527       398 LINDLEY DR  SACRAMENTO  95815    CA   4.0    2.0  1744.0
648  8198 STEVENSON AVE  SACRAMENTO  95828    CA   6.0    4.0  2475.0
716      1139 CLINTON RD  SACRAMENTO  95825    CA   4.0    2.0  1776.0
923         7351 GIGI PL  SACRAMENTO  95828    CA   4.0    2.0  1859.0

             type           sale_date   price   latitude   longitude
56   MULTI-FAMILY 1923-09-30 03:05:24  179580  38.665724 -121.358576
68   MULTI-FAMILY 2002-09-29 22:46:40  195000  38.480090 -121.415102
108  MULTI-FAMILY 2000-07-17 06:06:41  282400  38.559505 -121.364839
113  MULTI-FAMILY 1953-04-21 02:32:12  297000  38.570300 -121.315735
353  MULTI-FAMILY 1944-01-09 17:10:33  100000  38.617718 -121.440089
366  MULTI-FAMILY 2016-02-06 00:58:59  120000  38.490704 -121.410176
527  MULTI-FAMILY 1984-11-07 08:49:22  416767  38.622359 -121.457582
648  MULTI-FAMILY 1915-11-08 07:50:42  159900  38.465271 -121.404260
716  MULTI-FAMILY 1910-02-14 13:44:04  221250  38.585291 -121.406824
923  MULTI-FAMILY 1932-02-20 19:07:20  170000  38.490606 -121.410173
```

STEP 2: Filter the date that falls between '05/11/1933' and '03/12/1998' in step 1

[36]:
```python
# From the data_add gotten above, fiter the date thaat falls between 05/11/1933␣
↪and 03/12/1998
date_filter = (data_add['sale_date'] > '1933-11-05 00:00:00') &␣
↪(data_add['sale_date'] <= '1998-12-03 00:00:00') # Filter date 05/11/1933␣
↪and 03/12/1998
data_ctd= data_add.loc[date_filter] # data with filtered city= SACRAMENTO,␣
↪type=MULTI-FAMILY and date=05/11/1933 and 03/12/1998.
data_ctd
```

[36]:
```
                street        city    zip state  beds  baths   sq__ft  \
113  10158 CRAWFORD WAY  SACRAMENTO  95827    CA   4.0    4.0  2213.0
353  2820 DEL PASO BLVD  SACRAMENTO  95815    CA   4.0    2.0  1404.0
527       398 LINDLEY DR  SACRAMENTO  95815    CA   4.0    2.0  1744.0

             type           sale_date   price   latitude   longitude
113  MULTI-FAMILY 1953-04-21 02:32:12  297000  38.570300 -121.315735
```

```
353  MULTI-FAMILY 1944-01-09 17:10:33  100000  38.617718 -121.440089
527  MULTI-FAMILY 1984-11-07 08:49:22  416767  38.622359 -121.457582
```

STEP 3: From the 'street' column, extract the characters before the first blankspace. This corresponds to the street numbers .Then, find the median of these numbers

[37]:
```
# Extract street numbers from the street column (by splitting the content of␣
↪the column by blank spaces and extracting the first value)
# The result is passed to the median value method
street_num = (data_ctd['street'].apply(lambda x: x.split()[0])).median()
print('The median street number, in multi-family houses, sold between 05/11/
↪1933 and 03/12/1998 , in Sacramento is: ',street_num)
```

```
The median street number, in multi-family houses, sold between 05/11/1933 and
03/12/1998 , in Sacramento is:  2820.0
```

.

(C) What is the city name, and its 3 most common zip codes, that has the 2nd highest amount of beds sold?

SOLUTION / APPROACH

To do this:

STEP 1: Get the name of the city that has the 2nd highest amount of beds sold This is achieved by summing the number of beds per city.

The name of the city with the second highest number of sold beds is gotten

[38]:
```
# Step 1: Get the name of the city that has the 2nd highest amount of beds sold
# This is achieved by
k=new_data.groupby('city')['beds'].sum()
k.nlargest(2).iloc[[-1]]
```

[38]:
```
city
ELK GROVE    383.0
Name: beds, dtype: float64
```

STEP 2: Filter out the ELK GROVE rows from th original data since we established that ELK GROVE is the city of interest.

[39]:
```
# Filter out ELK GROVE rows from th original data since we established that ELK␣
↪GROVE is the city of interest.
data_elk=new_data[(new_data['city']=='ELK GROVE')]
data_elk
```

[39]:
```
                          street       city    zip state  beds  baths  \
30   5201 LAGUNA OAKS DR UNIT 140  ELK GROVE  95758    CA   2.0    2.0
34   5201 LAGUNA OAKS DR UNIT 162  ELK GROVE  95758    CA   2.0    2.0
42                  8718 ELK WAY  ELK GROVE  95624    CA   3.0    2.0
```

```
50                   9417 SARA ST  ELK GROVE  95624    CA   3.0    2.0
66               7005 TIANT % WAY  ELK GROVE  95758    CA   3.0    2.0
..                             …         …     …     …    …     …
964               10085 ATKINS DR  ELK GROVE  95757    CA   3.0    2.0
965           9185 CERROLINDA CIR  ELK GROVE  95758    CA   3.0    2.0
975          5024 CHAMBERLIN CIR  ELK GROVE  95757    CA   3.0    2.0
979              1909 YARNELL WAY  ELK GROVE  95758    CA   3.0    2.0
983              8304 RED FOX WAY  ELK GROVE  95758    CA   4.0    2.0

     sq__ft         type            sale_date   price   latitude   longitude
30   1039.0        CONDO 1995-06-27 11:07:11  133000  38.423251 -121.444489
34   1039.0        CONDO 1995-02-14 02:55:54  141000  38.423251 -121.444489
42   1056.0  RESIDENTIAL 1964-12-06 07:32:56  156896  38.416530 -121.379653
50   1188.0  RESIDENTIAL 1913-11-26 14:12:17  170000  38.415518 -121.370527
66   1586.0  RESIDENTIAL 1911-11-05 06:37:27  194000  38.422811 -121.423285
..      …            …                  …     …         …          …
964  1302.0  RESIDENTIAL 1965-08-27 15:55:54  219794  38.390893 -121.437821
965  1418.0  RESIDENTIAL 2003-11-28 09:52:55  220000  38.424497 -121.426595
975  1450.0  RESIDENTIAL 1919-02-03 11:26:02  228000  38.389756 -121.446246
979  1262.0  RESIDENTIAL 1979-04-17 04:20:28  230000  38.417382 -121.484325
983  1685.0  RESIDENTIAL 1933-04-10 20:13:38  235301  38.417000 -121.397424

[114 rows x 12 columns]
```

STEP 3 Find the three (3) most common zip codes of the ELK GROVE city

Do a groupby of zip with the GROVE city.This gives all the unique zip codes belonging to ELK GROVE

Then, count the number of occurrences(using the size()) of the unique ELK GROVE's zip codes and rename the resulting column as frequency (using the reset_index(name='frequency')

Rearrange the table in descending order (sort_values by ascending=[0,1])so that the most frequent is the topmost in the column.

Use the .head(3) to select the first three rows as the three most frequent

[40]:
```python
data_elk.groupby(['city','zip']).size().reset_index(name='frequency').
 ↪sort_values(['frequency','zip'],ascending=[0,1]).groupby('city').head(3)
```

[40]:

	city	zip	frequency
2	ELK GROVE	95758	44
1	ELK GROVE	95757	36
0	ELK GROVE	95624	34

[41]:
```python
# Print the solution
stg='''Therefore, the city name, and the 3 most common zip codes, that has the␣
 ↪2nd highest amount of beds sold: \n
city name: ELK GROVE \n
```

```
Zip codes: 95758,95757 and 95624'''
print(stg)
```

Therefore, the city name, and the 3 most common zip codes, that has the 2nd highest amount of beds sold:

city name: ELK GROVE

Zip codes: 95758,95757 and 95624