Name: Sidhant satapathy
Roll No: 21-27-06
M.Tech: Data Science

In [1]:
```
import pandas as pd
import numpy as np
import re

import warnings
warnings.filterwarnings("ignore")
```

1. Read the housingdata.csv file into pandas DataFrame and display first six rows of the DataFrame

In [2]:
```
df = pd.read_csv("housingdata.csv")
df
```

Out[2]:

	PID	STATE	PRICE	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	MP	321654	3	1	1000
1	100002000.0	MAHARASHTRA	21325	3	1.5	--
2	100003000.0	AP	2541654	NaN	1	850
3	100004000.0	TN	321321	1	NaN	700
4	NaN	TN	589465645	3	2	1600
5	100006000.0	TN	65465466	NaN	1	800
6	100007000.0	ASSAM	3222321	2	HURLEY	950
7	100008000.0	HP	23131	1	1	NaN
8	100009000.0	HP	21212	na	2	1800

2. Display the column names and row index in separate cells.

In [3]:
```
print(f'Column: {df.columns}')
```

```
Column: Index(['PID', 'STATE', 'PRICE', 'NUM_BEDROOMS', 'NUM_BATH', 'SQ_FT'], dtype='object')
```

In [4]:
```
df.index
```

Out[4]:
```
RangeIndex(start=0, stop=9, step=1)
```

3. a) How many numbers of "STATE" exist in the dataset. Hint: use shape b) How many unique states exist in the dataset

In [5]:
```
print(f'No. of states in dataset: {len(df) - 1}')
```

```
No. of states in dataset: 8
```

In [6]:
```
X = df.STATE.unique()
print(f'Unique States: {X}')
print(f'No. of unique states: {len(X)}')
```

```
Unique States: ['MP' 'MAHARASHTRA' 'AP' 'TN' 'ASSAM' 'HP']
No. of unique states: 6
```

4. Retrieve the list of all NaN/Null/Empty cells in the form of Boolean list

In [7]:
```
# A zip object yielding tuples until an input is exhausted.
[df.iloc[i,j] for i,j in zip(*np.where(pd.isnull(df)))]

'''
Using * and ** to pass arguments to a function
Using * and ** to capture arguments passed into a function
Using * to accept keyword-only arguments
Using * to capture items during tuple unpacking
Using * to unpack iterables into a list/tuple
```

```
Using ** to unpack dictionaries into other dictionaries
'''
```

Out[7]:
```
'\nUsing * and ** to pass arguments to a function\nUsing * and ** to capture arguments passed into a funct
ion\nUsing * to accept keyword-only arguments\nUsing * to capture items during tuple unpacking\nUsing * to
unpack iterables into a list/tuple\nUsing ** to unpack dictionaries into other dictionaries\n'
```

In [8]:
```
X = np.array(np.where(pd.isnull(df)))
X
```

Out[8]:
```
array([[2, 3, 4, 5, 7],
       [3, 4, 0, 3, 5]], dtype=int64)
```

In [9]:
```
df.iloc[2,3]
```

Out[9]:
```
nan
```

In [10]:
```
y = df.isnull()
y
```

Out[10]:

	PID	STATE	PRICE	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	False	False	False	False	False	False
1	False	False	False	False	False	False
2	False	False	False	True	False	False
3	False	False	False	False	True	False
4	True	False	False	False	False	False
5	False	False	False	True	False	False
6	False	False	False	False	False	False
7	False	False	False	False	False	True
8	False	False	False	False	False	False

In [11]:
```
y = y.values
y.tolist
y
```

Out[11]:
```
array([[False, False, False, False, False, False],
       [False, False, False, False, False, False],
       [False, False, False,  True, False, False],
       [False, False, False, False,  True, False],
       [ True, False, False, False, False, False],
       [False, False, False,  True, False, False],
       [False, False, False, False, False, False],
       [False, False, False, False, False,  True],
       [False, False, False, False, False, False]])
```

5. Drop all rows with N/A,NA,na values in Num_Bedrooms

In [12]:
```
df['NUM_BEDROOMS'] = df['NUM_BEDROOMS'].replace('na', np.NaN)
df

# Re Module
# df = df.replace(r'^\s*$', np.NaN, regex=True)
```

Out[12]:

	PID	STATE	PRICE	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	MP	321654	3	1	1000
1	100002000.0	MAHARASHTRA	21325	3	1.5	--
2	100003000.0	AP	2541654	NaN	1	850
3	100004000.0	TN	321321	1	NaN	700
4	NaN	TN	589465645	3	2	1600
5	100006000.0	TN	65465466	NaN	1	800

	PID	STATE	PRICE	NUM_BEDROOMS	NUM_BATH	SQ_FT
6	100007000.0	ASSAM	3222321	2	HURLEY	950
7	100008000.0	HP	23131	1	1	NaN
8	100009000.0	HP	21212	NaN	2	1800

In [13]:
```
df = df[df['NUM_BEDROOMS'].notna()]
df
```

Out[13]:

	PID	STATE	PRICE	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	MP	321654	3	1	1000
1	100002000.0	MAHARASHTRA	21325	3	1.5	--
3	100004000.0	TN	321321	1	NaN	700
4	NaN	TN	589465645	3	2	1600
6	100007000.0	ASSAM	3222321	2	HURLEY	950
7	100008000.0	HP	23131	1	1	NaN

6. Replace the NaN and String entries in Num_Bath with previous entry

In [14]:
```
df['NUM_BATH'] = df['NUM_BATH'].str.replace('^[a-z]+', '' , regex = True, flags = re.I)
df['NUM_BATH'] = df['NUM_BATH'].replace('', np.NaN)
df['NUM_BATH'] = df['NUM_BATH'].fillna(method= 'ffill')

df['NUM_BATH'] = df['NUM_BATH'].fillna(method='ffill')
df['NUM_BATH'] = df['NUM_BATH'].replace(df['NUM_BATH'].str, method= 'ffill')

df
```

Out[14]:

	PID	STATE	PRICE	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	MP	321654	3	1	1000
1	100002000.0	MAHARASHTRA	21325	3	1.5	--
3	100004000.0	TN	321321	1	1.5	700
4	NaN	TN	589465645	3	2	1600
6	100007000.0	ASSAM	3222321	2	2	950
7	100008000.0	HP	23131	1	1	NaN

7. Replace the empty values in “SQ_FT” with the mean of the all entries

In [15]:
```
df['SQ_FT'] = df['SQ_FT'].replace('--',np.NaN)
df['SQ_FT'] = df['SQ_FT'].astype(float)
df['SQ_FT'].fillna(int(df['SQ_FT'].mean()), inplace=True)
df
```

Out[15]:

	PID	STATE	PRICE	NUM_BEDROOMS	NUM_BATH	SQ_FT
0	100001000.0	MP	321654	3	1	1000.0
1	100002000.0	MAHARASHTRA	21325	3	1.5	1062.0
3	100004000.0	TN	321321	1	1.5	700.0
4	NaN	TN	589465645	3	2	1600.0
6	100007000.0	ASSAM	3222321	2	2	950.0
7	100008000.0	HP	23131	1	1	1062.0

8. What is the sum of SQ_FT of all entries in the dataset

In [16]:
```
sum = np.sum(df['SQ_FT'])
sum
```

Out[16]: 6374.0

9. Add a new column "NUM_STORE" and with all the values as 1 for all rows

In [17]:
```
NUM_STORE = np.ones(6).astype(int)
NUM_STORE
df['NUM_STORE'] = NUM_STORE
df
```

Out[17]:

	PID	STATE	PRICE	NUM_BEDROOMS	NUM_BATH	SQ_FT	NUM_STORE
0	100001000.0	MP	321654	3	1	1000.0	1
1	100002000.0	MAHARASHTRA	21325	3	1.5	1062.0	1
3	100004000.0	TN	321321	1	1.5	700.0	1
4	NaN	TN	589465645	3	2	1600.0	1
6	100007000.0	ASSAM	3222321	2	2	950.0	1
7	100008000.0	HP	23131	1	1	1062.0	1

10. Create a DataFrame containing all entries of TN state only

In [18]:
```
TN = df
TN = TN.loc[TN['STATE'] == 'TN']
TN
```

Out[18]:

	PID	STATE	PRICE	NUM_BEDROOMS	NUM_BATH	SQ_FT	NUM_STORE
3	100004000.0	TN	321321	1	1.5	700.0	1
4	NaN	TN	589465645	3	2	1600.0	1

11. Create a DataFrame having SQ_FT area greater than 1000 and display the DataFrame

In [19]:
```
SQT = df

SQT['SQ_FT'] = SQT['SQ_FT'].replace('--', np.NaN)
SQT['SQ_FT'] = SQT['SQ_FT'].fillna(method= 'ffill')

SQT['SQ_FT'] = SQT['SQ_FT'].astype(float)

# SQT = SQT['SQ_FT'].loc[SQT['SQ_FT'] > 1000]
SQT = SQT.loc[SQT['SQ_FT'] > 1000]
SQT
```

Out[19]:

	PID	STATE	PRICE	NUM_BEDROOMS	NUM_BATH	SQ_FT	NUM_STORE
1	100002000.0	MAHARASHTRA	21325	3	1.5	1062.0	1
4	NaN	TN	589465645	3	2	1600.0	1
7	100008000.0	HP	23131	1	1	1062.0	1

12. Create a DataFrame having only first Three columns and First Three rows

In [20]:
```
df3 = df
df3 = df3.iloc[:3,:3]
df3
```

Out[20]:

	PID	STATE	PRICE
0	100001000.0	MP	321654
1	100002000.0	MAHARASHTRA	21325
3	100004000.0	TN	321321

13. Display the state having highest average price per square foot of housing area. Hint: Average of (TotalPrice/Sq_Ft)

In [21]:
```
per = df
per
```

Out[21]:

	PID	STATE	PRICE	NUM_BEDROOMS	NUM_BATH	SQ_FT	NUM_STORE
0	100001000.0	MP	321654	3	1	1000.0	1
1	100002000.0	MAHARASHTRA	21325	3	1.5	1062.0	1
3	100004000.0	TN	321321	1	1.5	700.0	1
4	NaN	TN	589465645	3	2	1600.0	1
6	100007000.0	ASSAM	3222321	2	2	950.0	1
7	100008000.0	HP	23131	1	1	1062.0	1

In [22]:
```
per = df.groupby('STATE').mean()
per
```

Out[22]:

STATE	PID	PRICE	SQ_FT	NUM_STORE
ASSAM	100007000.0	3222321.0	950.0	1.0
HP	100008000.0	23131.0	1062.0	1.0
MAHARASHTRA	100002000.0	21325.0	1062.0	1.0
MP	100001000.0	321654.0	1000.0	1.0
TN	100004000.0	294893483.0	1150.0	1.0

In [23]:
```
per['Average'] = per['PRICE']/per['SQ_FT']
per
```

Out[23]:

STATE	PID	PRICE	SQ_FT	NUM_STORE	Average
ASSAM	100007000.0	3222321.0	950.0	1.0	3391.916842
HP	100008000.0	23131.0	1062.0	1.0	21.780603
MAHARASHTRA	100002000.0	21325.0	1062.0	1.0	20.080038
MP	100001000.0	321654.0	1000.0	1.0	321.654000
TN	100004000.0	294893483.0	1150.0	1.0	256429.115652

In [24]:
```
per['Average'].idxmax()
```

Out[24]:
```
'TN'
```

In [ ]: