Problem Statement and Data Description

Chennai House Price Prediction (Regression)

ChennaiEstate is a real estate firm based in Chennai that is involved in the property business for the past 5 years. Since, they are in the business for so long, they have enough data of all the real estate transactions in the city.

They decided to venture into Analytics and have now started a division called “Chennai Estate Analytics” to give consumers as much information as possible about housings and the real estate market in Chennai. A home is often the largest and most expensive purchase a person makes in his or her lifetime. Ensuring real-estate owners have a trusted way to monitor the asset is incredibly important. Hence, they have hired you as a consultant to help them give insights and develop a model to accurately predict real estate prices.

Based on the train dataset, you will need to develop a model that accurately predicts the real estate price in Chennai.

Data Description

House Features

- INT_SQFT – The interior Sq. Ft of the property
- N_BEDROOM – The number of Bed rooms
- N_BATHROOM - The number of bathrooms
- N_ROOM – Total Number of Rooms
- QS_ROOMS – The quality score assigned for rooms based on buyer reviews
- QS_BATHROOM – The quality score assigned for bathroom based on buyer reviews
- QS_BEDROOM – The quality score assigned for bedroom based on buyer reviews
- QS_OVERALL – The Overall quality score assigned for the property
- SALE_COND – The Sale Condition
  - Normal: Normal Sale
  - Abnorml: Abnormal Sale - trade, foreclosure, short sale
  - AdjLand: Adjoining Land Purchase
  - Family: Sale between family members
  - Partial: Home was not completed when last assessed
- BUILDTYPE – The type of building
  - House
  - Commercial
  - Others

Surrounding and Locality

- AREA – The property in which the real estate is located
- DIST_MAINROAD – The distance of the property to the main road
- PARK_FACIL – Whether parking facility is available
- UTILITY_AVAIL
  - AllPub: All public Utilities (E,G,W,& S)
  - NoSewr: Electricity, Gas, and Water (Septic Tank)

- NoSeWa: Electricity and Gas Only
- ELO: Electricity only
- STREET
  - Gravel
  - Paved
  - No Access
- MZZONE
  - A: Agriculture
  - C: Commercial
  - I: Industrial
  - RH: Residential High Density
  - RL: Residential Low Density
  - RM: Residential Medium Density

House Sale Price

- PRT_ID – The Property Transaction ID assigned by ChennaiEstate
- COMMIS – The Commission paid to the agent
- SALES_PRICE – The total sale price of the property

Loading the Dataset

In [1]:

```
import pandas as pd
import numpy as np

import matplotlib.pyplot as plt
%matplotlib inline
```

In [2]:

```
df = pd.read_csv("chennai_house_price_prediction.csv")
df.shape
```

Out[2]:

```
(7109, 19)
```

In [3]:

```
df.head()
```

Out[3]:

	PRT_ID	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM
0	P03210	Karapakkam	1004	131	1.0	1.0	3
1	P09411	Anna Nagar	1986	26	2.0	1.0	5
2	P01812	Adyar	909	70	1.0	1.0	3
3	P05346	Velachery	1855	14	3.0	2.0	5
4	P06210	Karapakkam	1226	84	1.0	1.0	3

Data Exploration

Describe function

In [4]:

```
df.describe()
```

Out[4]:

	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	QS_ROOMS
count	7109.000000	7109.000000	7108.000000	7104.000000	7109.000000	7109.000000
mean	1382.073006	99.603179	1.637029	1.213260	3.688704	3.517471
std	457.410902	57.403110	0.802902	0.409639	1.019099	0.891972
min	500.000000	0.000000	1.000000	1.000000	2.000000	2.000000
25%	993.000000	50.000000	1.000000	1.000000	3.000000	2.700000
50%	1373.000000	99.000000	1.000000	1.000000	4.000000	3.500000
75%	1744.000000	148.000000	2.000000	1.000000	4.000000	4.300000
max	2500.000000	200.000000	4.000000	2.000000	6.000000	5.000000

- The describe function works only for continuous variables
- We can identify the number of missing values from the 'count' given
- Comparing the 75% and the max value, determine presence of outliers

In [5]:

```
df.describe(include='all')
```

Out[5]:

	PRT_ID	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_
count	7109	7109	7109.000000	7109.000000	7108.000000	7104.000000	7109.(
unique	7109	17	NaN	NaN	NaN	NaN	
top	P08550	Chrompet	NaN	NaN	NaN	NaN	
freq	1	1681	NaN	NaN	NaN	NaN	
mean	NaN	NaN	1382.073006	99.603179	1.637029	1.213260	3.6
std	NaN	NaN	457.410902	57.403110	0.802902	0.409639	1.(
min	NaN	NaN	500.000000	0.000000	1.000000	1.000000	2.(
25%	NaN	NaN	993.000000	50.000000	1.000000	1.000000	3.(
50%	NaN	NaN	1373.000000	99.000000	1.000000	1.000000	4.(
75%	NaN	NaN	1744.000000	148.000000	2.000000	1.000000	4.(
max	NaN	NaN	2500.000000	200.000000	4.000000	2.000000	6.(

- Count can be used to find out missing values
- Gives unique values for categorical variables

Isnull function

In [6]:

```
df.isnull().sum()
```

Out[6]:

```
PRT_ID            0
AREA              0
INT_SQFT          0
DIST_MAINROAD     0
N_BEDROOM         1
N_BATHROOM        5
N_ROOM            0
SALE_COND         0
PARK_FACIL        0
BUILDTYPE         0
UTILITY_AVAIL     0
STREET            0
MZZONE            0
QS_ROOMS          0
QS_BATHROOM       0
QS_BEDROOM        0
QS_OVERALL       48
COMMIS            0
SALES_PRICE       0
dtype: int64
```

Data types

In [7]:

```
df.dtypes
```

Out[7]:

```
PRT_ID            object
AREA              object
INT_SQFT           int64
DIST_MAINROAD      int64
N_BEDROOM        float64
N_BATHROOM       float64
N_ROOM             int64
SALE_COND         object
PARK_FACIL        object
BUILDTYPE         object
UTILITY_AVAIL     object
STREET            object
MZZONE            object
QS_ROOMS         float64
QS_BATHROOM      float64
QS_BEDROOM       float64
QS_OVERALL       float64
COMMIS             int64
SALES_PRICE        int64
dtype: object
```

In [8]:

```
temp = pd.DataFrame(index=df.columns)
temp['data_type'] = df.dtypes
temp['null_count'] = df.isnull().sum()
temp['unique_count'] = df.nunique()
```

In [9]:

```
temp
```

Out[9]:

	data_type	null_count	unique_count
PRT_ID	object	0	7109
AREA	object	0	17
INT_SQFT	int64	0	1699
DIST_MAINROAD	int64	0	201
N_BEDROOM	float64	1	4
N_BATHROOM	float64	5	2
N_ROOM	int64	0	5
SALE_COND	object	0	9
PARK_FACIL	object	0	3
BUILDTYPE	object	0	5
UTILITY_AVAIL	object	0	5
STREET	object	0	5
MZZONE	object	0	6
QS_ROOMS	float64	0	31
QS_BATHROOM	float64	0	31
QS_BEDROOM	float64	0	31
QS_OVERALL	float64	48	479
COMMIS	int64	0	7011
SALES_PRICE	int64	0	7057

Univariate Analysis

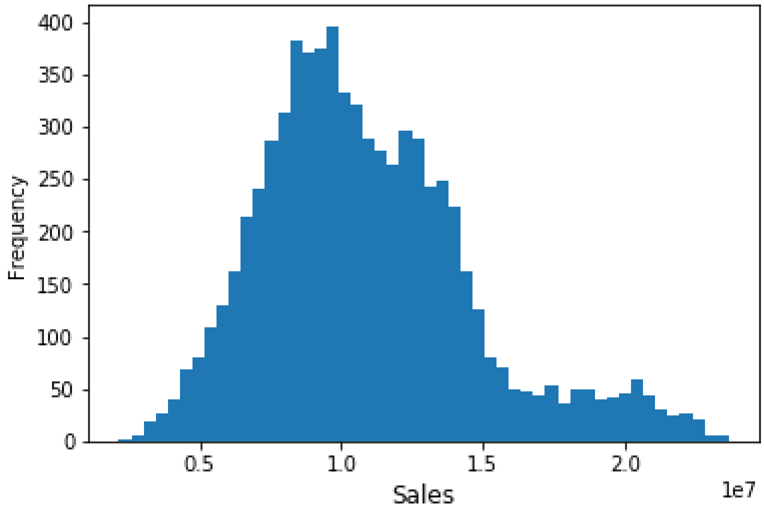
Histogram

In [10]:

```
## target variable

df['SALES_PRICE'].plot.hist(bins = 50)
plt.xlabel('Sales', fontsize=12)
```

Out[10]:

```
Text(0.5,0,'Sales')
```



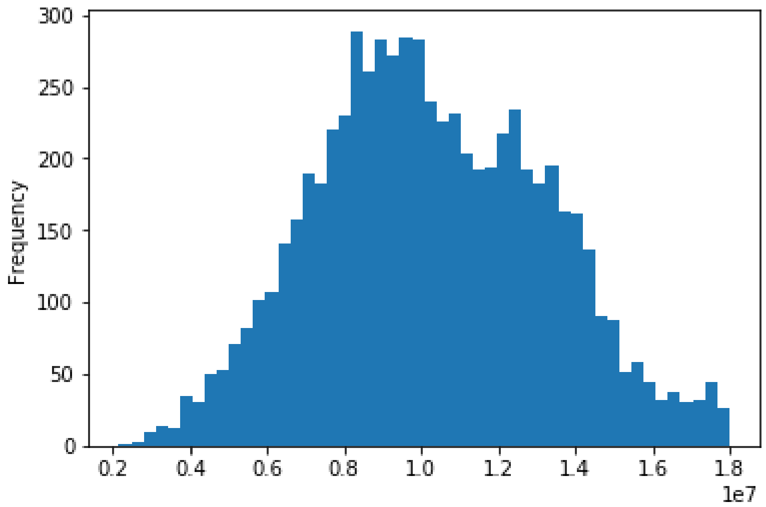
- The distribution of the target variable is slightly right skewed.
- We can see a small number of houses with a very high price.

In [11]:

```
(df['SALES_PRICE'].loc[df['SALES_PRICE']<18000000]).plot.hist(bins=50)
```

Out[11]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f293c8bd2b0>
```

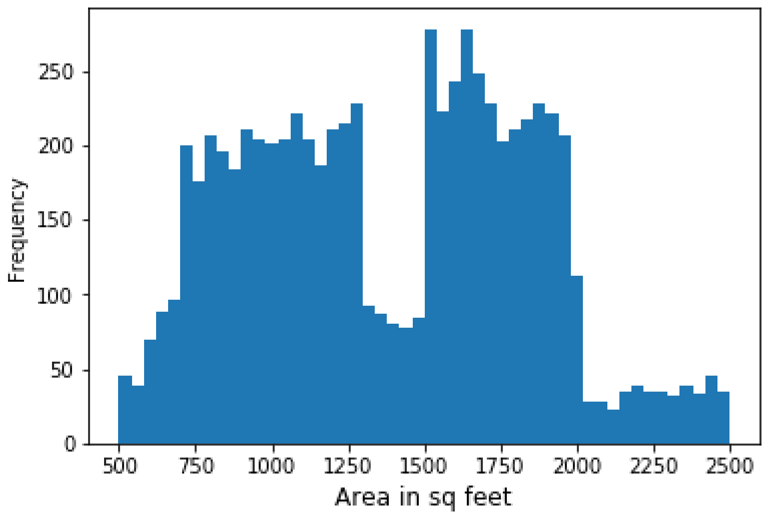


In [12]:

```
## Area of house in Square feet

df['INT_SQFT'].plot.hist(bins = 50)
plt.xlabel('Area in sq feet', fontsize=12)
```

Out[12]:

```
Text(0.5,0,'Area in sq feet')
```



- Most houses have the area between **750 sq feet to 1250 sq feet** or around **1500 sq feet to 2000 sq feet**
- Very less number of houses have area more than 2000 sq feet or less than 750 sq feet

Value counts

In [13]:

```
# number of bedrooms

df['N_BEDROOM'].value_counts()
```

Out[13]:

```
1.0    3795
2.0    2352
3.0     707
4.0     254
Name: N_BEDROOM, dtype: int64
```

- It has four different categories
- This variable should be object and not integer

In [14]:

```
df['N_BEDROOM'].value_counts()/len(df)*100
```

Out[14]:

```
1.0    53.383036
2.0    33.084822
3.0     9.945140
4.0     3.572936
Name: N_BEDROOM, dtype: float64
```

- About 53% houses have one bedroom
- 33% have 2 bedrooms
- Less than 10% houses have 3 bedrooms
- Only 3.5% have 4 bedrooms

In [15]:

```
df['N_ROOM'].value_counts()
```

Out[15]:

```
4    2563
3    2125
5    1246
2     921
6     254
Name: N_ROOM, dtype: int64
```

- The 'Rooms' might have number of kitchen, hall, dinning area etc.
- No house with 1 room, and a very few that have 2

In [16]:

```
df['N_BATHROOM'].value_counts()/len(df)
```

Out[16]:

```
1.0    0.786187
2.0    0.213110
Name: N_BATHROOM, dtype: float64
```

- 78% houses have 1 bathroom and 21% have 2 bathrooms
- The same can be represented using bar plots

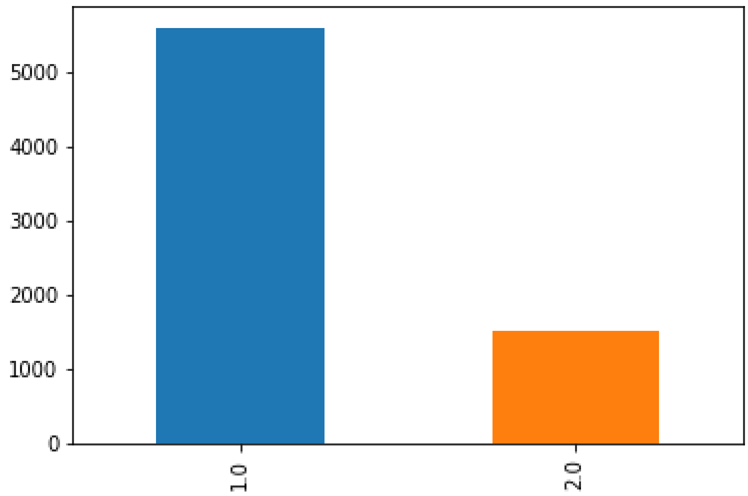
Bar Plot

In [17]:

```
df['N_BATHROOM'].value_counts().plot(kind = 'bar')
```

Out[17]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f293c72e2e8>
```

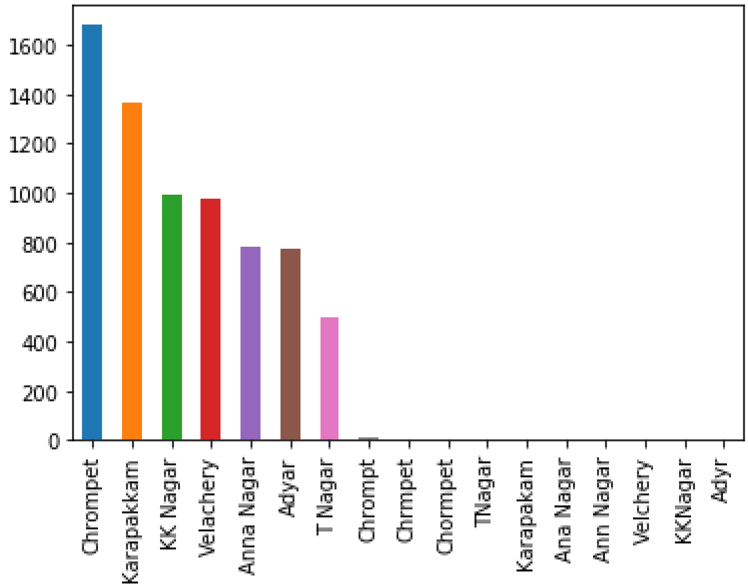


In [18]:

```
df['AREA'].value_counts().plot(kind = 'bar')
```

Out[18]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f293a414a20>
```



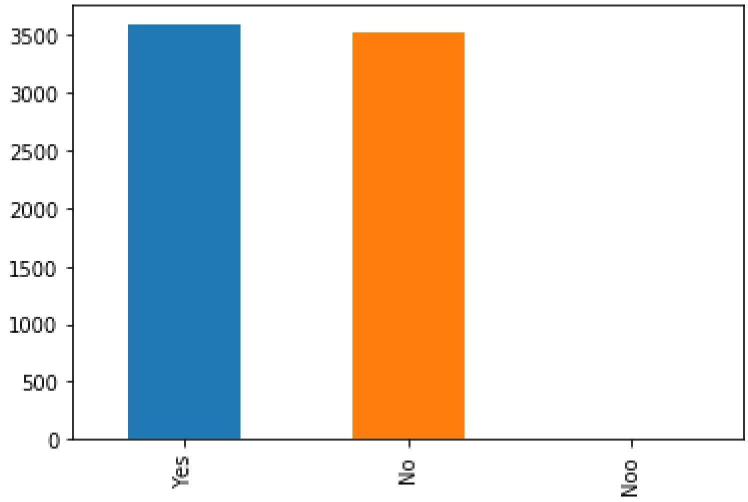
- There are 17 different categories in the 'AREA' variable
- Only 7 unique area name
- maximum houses are in the area Chrompet, followed by Karapakkam

In [19]:

```
# houses with parking facility

df['PARK_FACIL'].value_counts().plot(kind = 'bar')
```

Out[19]:

```
&lt;matplotlib.axes._subplots.AxesSubplot at 0x7f293c7ab2b0&gt;
```



In [20]:

```
df['PARK_FACIL'].value_counts()
```

Out[20]:

```
Yes     3587
No      3520
Noo        2
Name: PARK_FACIL, dtype: int64
```

- There are only two unique categories
- The number of houses with parking facility in both the cases is almost the same

Data Manipulation

1. Drop Duplicates (if any)
2. Fill the missing Values
3. Correct the data types
4. Fix the spelling errors in variables

Drop Duplicates (if any)

In [21]:

```
df.drop_duplicates()
```

Out[21]:

	PRT_ID	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_CON
0	P03210	Karapakkam	1004	131	1.0	1.0	3	AbNorm
1	P09411	Anna Nagar	1986	26	2.0	1.0	5	AbNorm
2	P01812	Adyar	909	70	1.0	1.0	3	AbNorm
3	P05346	Velachery	1855	14	3.0	2.0	5	Fam
4	P06210	Karapakkam	1226	84	1.0	1.0	3	AbNorm
5	P00219	Chrompet	1220	36	2.0	1.0	4	Parti
6	P09105	Chrompet	1167	137	1.0	1.0	3	Parti
7	P09679	Velachery	1847	176	3.0	2.0	5	Fam
8	P03377	Chrompet	771	175	1.0	1.0	2	AdjLar

In [22]:

```
df.drop_duplicates(subset=['AREA']).shape
```

Out[22]:

(17, 19)

In [23]:

```
df.shape
```

Out[23]:

(7109, 19)

- We have no duplicates. Hence the shape did not change here.

Missing Values

In [24]:

```
# missing values

df.isnull().sum()
```

Out[24]:

```
PRT_ID            0
AREA              0
INT_SQFT          0
DIST_MAINROAD     0
N_BEDROOM         1
N_BATHROOM        5
N_ROOM            0
SALE_COND         0
PARK_FACIL        0
BUILDTYPE         0
UTILITY_AVAIL     0
STREET            0
MZZONE            0
QS_ROOMS          0
QS_BATHROOM       0
QS_BEDROOM        0
QS_OVERALL       48
COMMIS            0
SALES_PRICE       0
dtype: int64
```

Different ways deal with the missing values

- Remove the rows with missing values
- Mean or median in case of continuous variable
- With mode in case of categorical variable
- Using other independent variables

Drop rows with missing values

In [25]:

```
df.dropna(axis=0, how='any')
```

Out[25]:

	PRT_ID	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM	SALE_CON
0	P03210	Karapakkam	1004	131	1.0	1.0	3	AbNorm
1	P09411	Anna Nagar	1986	26	2.0	1.0	5	AbNorm
2	P01812	Adyar	909	70	1.0	1.0	3	AbNorm
3	P05346	Velachery	1855	14	3.0	2.0	5	Fam
4	P06210	Karapakkam	1226	84	1.0	1.0	3	AbNorm
5	P00219	Chrompet	1220	36	2.0	1.0	4	Parti
6	P09105	Chrompet	1167	137	1.0	1.0	3	Parti
7	P09679	Velachery	1847	176	3.0	2.0	5	Fam
8	P03377	Chrompet	771	175	1.0	1.0	2	AdjLar

- To make changes to original data, use inplace=True
- In this case, 54 rows removed

In [26]:

```
df.dropna(axis=1, how='any')
```

Out[26]:

	PRT_ID	AREA	INT_SQFT	DIST_MAINROAD	N_ROOM	SALE_COND	PARK_FACIL	BUILDTYPE
0	P03210	Karapakkam	1004	131	3	AbNormal	Yes	Commercial
1	P09411	Anna Nagar	1986	26	5	AbNormal	No	Commercial
2	P01812	Adyar	909	70	3	AbNormal	Yes	Commercial
3	P05346	Velachery	1855	14	5	Family	No	Others
4	P06210	Karapakkam	1226	84	3	AbNormal	Yes	Others
5	P00219	Chrompet	1220	36	4	Partial	No	Commercial
6	P09105	Chrompet	1167	137	3	Partial	No	Other
7	P09679	Velachery	1847	176	5	Family	No	Commercial
8	P03377	Chrompet	771	175	2	AdjLand	No	Others

- When axis is set to 1, columns are dropped.
- For given data, 3 columns has missing values hence three columns dropped
- To avoid loss of data, we can use other ways of imputation

1. N_BEDROOM

In [27]:

```
df['N_BEDROOM'].mode()
```

Out[27]:

```
0    1.0
dtype: float64
```

In [28]:

```
df['N_BEDROOM'].fillna(value = (df['N_BEDROOM'].mode()[0]), inplace=True)
```

2. N_BATHROOM

In [29]:

```
df.loc[df['N_BATHROOM'].isnull()==True]
```

Out[29]:

	PRT_ID	AREA	INT_SQFT	DIST_MAINROAD	N_BEDROOM	N_BATHROOM	N_ROOM
70	P05304	Anna Nagar	1589	39	1.0	NaN	4
5087	P01333	Chrompet	1016	105	1.0	NaN	3
6134	P01332	Chormpet	916	173	1.0	NaN	3
6371	P01189	Chrompet	1035	90	1.0	NaN	3
6535	P09189	Anna Nagar	1864	184	2.0	NaN	5

In [30]:

```
for i in range(0, len(df)):
    if pd.isnull(df['N_BATHROOM'][i])==True:
        if (df['N_BEDROOM'][i] == 1.0):
            df['N_BATHROOM'][i] = 1.0
        else:
            df['N_BATHROOM'][i] = 2.0
```

```
/home/aishwarya/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:
4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/s
table/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pand
as-docs/stable/indexing.html#indexing-view-versus-copy)
  after removing the cwd from sys.path.
/home/aishwarya/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:
6: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/s
table/indexing.html#indexing-view-versus-copy (http://pandas.pydata.org/pand
as-docs/stable/indexing.html#indexing-view-versus-copy)
```

3. QS_OVERALL

In [31]:

```
df[[ 'QS_ROOMS','QS_BATHROOM', 'QS_BEDROOM', 'QS_OVERALL']].head()
```

Out[31]:

	QS_ROOMS	QS_BATHROOM	QS_BEDROOM	QS_OVERALL
0	4.0	3.9	4.9	4.330
1	4.9	4.2	2.5	3.765
2	4.1	3.8	2.2	3.090
3	4.7	3.9	3.6	4.010
4	3.0	2.5	4.1	3.290

In [32]:

```
temp = (df['QS_ROOMS'] + df['QS_BATHROOM'] + df['QS_BEDROOM'])/3
pd.concat([df['QS_ROOMS'], df['QS_BATHROOM'], df['QS_BEDROOM'], temp], axis=1).head(10)
```

Out[32]:

	QS_ROOMS	QS_BATHROOM	QS_BEDROOM	0
0	4.0	3.9	4.9	4.266667
1	4.9	4.2	2.5	3.866667
2	4.1	3.8	2.2	3.366667
3	4.7	3.9	3.6	4.066667
4	3.0	2.5	4.1	3.200000
5	4.5	2.6	3.1	3.400000
6	3.6	2.1	2.5	2.733333
7	2.4	4.5	2.1	3.000000
8	2.9	3.7	4.0	3.533333
9	3.1	3.1	3.3	3.166667

- Imputing missing values with the help of other 'quality score' columns
- Additionally we can assign higher weights to n_bedroom and lower to n_bathroom

In [33]:

```
df.loc[df['QS_OVERALL'].isnull()==True].shape
```

Out[33]:

```
(48, 19)
```

In [34]:

```
def fill_na(x):
    return ((x['QS_ROOMS'] + x['QS_BATHROOM'] + x['QS_BEDROOM'])/3)
```

In [35]:

```
df['QS_OVERALL'] =  df.apply(lambda x: fill_na(x) if pd.isnull(x['QS_OVERALL']) else x['QS_
```

In [36]:

```
df.isnull().sum()
```

Out[36]:

```
PRT_ID           0
AREA             0
INT_SQFT         0
DIST_MAINROAD    0
N_BEDROOM        0
N_BATHROOM       0
N_ROOM           0
SALE_COND        0
PARK_FACIL       0
BUILDTYPE        0
UTILITY_AVAIL    0
STREET           0
MZZONE           0
QS_ROOMS         0
QS_BATHROOM      0
QS_BEDROOM       0
QS_OVERALL       0
COMMIS           0
SALES_PRICE      0
dtype: int64
```

Data Types

In [37]:

```
df.dtypes
```

Out[37]:

```
PRT_ID            object
AREA              object
INT_SQFT           int64
DIST_MAINROAD      int64
N_BEDROOM        float64
N_BATHROOM       float64
N_ROOM             int64
SALE_COND         object
PARK_FACIL        object
BUILDTYPE         object
UTILITY_AVAIL     object
STREET            object
MZZONE            object
QS_ROOMS         float64
QS_BATHROOM      float64
QS_BEDROOM       float64
QS_OVERALL       float64
COMMIS             int64
SALES_PRICE        int64
dtype: object
```

In [38]:

```
# data type of n_bedroom, n_room, n_bathroom

df = df.astype({'N_BEDROOM': 'object', 'N_ROOM': 'object', 'N_BATHROOM': 'object'})
```

Replace categories

In [39]:

```
temp = ['AREA','N_BEDROOM','N_BATHROOM','N_ROOM','SALE_COND','PARK_FACIL','BUILDTYPE','UTIL
for i in temp:
    print('************ Value Count in', i, '************')
    print(df[i].value_counts())
    print('')
```

```
************ Value Count in AREA ************
Chrompet      1681
Karapakkam    1363
KK Nagar       996
Velachery      979
Anna Nagar     783
Adyar          773
T Nagar        496
Chrompt          9
Chrmpet          6
Chormpet         6
TNagar           5
Karapakam        3
Ana Nagar        3
Ann Nagar        2
Velchery         2
KKNagar          1
Adyr             1
Name: AREA, dtype: int64

************ Value Count in N_BEDROOM ************
1.0    3796
2.0    2352
3.0     707
4.0     254
Name: N_BEDROOM, dtype: int64

************ Value Count in N_BATHROOM ************
1.0    5593
2.0    1516
Name: N_BATHROOM, dtype: int64

************ Value Count in N_ROOM ************
4    2563
3    2125
5    1246
2     921
6     254
Name: N_ROOM, dtype: int64

************ Value Count in SALE_COND ************
AdjLand        1433
Partial        1429
Normal Sale    1423
AbNormal       1406
Family         1403
Adj Land          6
Ab Normal         5
Partiall          3
PartiaLl          1
Name: SALE_COND, dtype: int64

************ Value Count in PARK_FACIL ************
```

```
Yes    3587
No     3520
Noo       2
Name: PARK_FACIL, dtype: int64

************ Value Count in BUILDTYPE ************
House         2444
Commercial    2325
Others        2310
Other           26
Comercial        4
Name: BUILDTYPE, dtype: int64

************ Value Count in UTILITY_AVAIL ************
AllPub     1886
NoSeWa     1871
NoSewr     1829
ELO        1522
All Pub       1
Name: UTILITY_AVAIL, dtype: int64

************ Value Count in STREET ************
Paved        2560
Gravel       2520
No Access    2010
Pavd           12
NoAccess        7
Name: STREET, dtype: int64

************ Value Count in MZZONE ************
RL    1858
RH    1822
RM    1817
C      550
A      537
I      525
Name: MZZONE, dtype: int64
```

Update names in column

- AREA
- SALE_COND
- PARK_FACIL
- BUILDTYPE
- UTILITY_AVAIL
- STREET

In [40]:

```
df['PARK_FACIL'].replace({'Noo':'No'}, inplace = True)
df['PARK_FACIL'].value_counts()
```

Out[40]:

```
Yes    3587
No     3522
Name: PARK_FACIL, dtype: int64
```

In [41]:

```
df['AREA'].replace({'TNagar':'T Nagar', 'Adyr': 'Adyar', 'KKNagar': 'KK Nagar',
                    'Chrompt': 'Chrompet', 'Chormpet': 'Chrompet','Chrmpet': 'Chrompet',
                    'Ana Nagar': 'Anna Nagar', 'Ann Nagar': 'Anna Nagar',
                     'Karapakam': 'Karapakkam' , 'Velchery': 'Velachery'}, inplace = True)
```

In [42]:

```
df['AREA'].value_counts()
```

Out[42]:

```
Chrompet      1702
Karapakkam    1366
KK Nagar       997
Velachery      981
Anna Nagar     788
Adyar          774
T Nagar        501
Name: AREA, dtype: int64
```

In [43]:

```
df['SALE_COND'].replace({'PartiaLl':'Partial', 'Partiall': 'Partial',
                         'Adj Land': 'AdjLand',
                         'Ab Normal': 'AbNormal'}, inplace = True)
df['SALE_COND'].value_counts()
```

Out[43]:

```
AdjLand        1439
Partial        1433
Normal Sale    1423
AbNormal       1411
Family         1403
Name: SALE_COND, dtype: int64
```

In [44]:

```
df['BUILDTYPE'].replace({'Comercial':'Commercial', 'Other': 'Others'},inplace = True)
df['UTILITY_AVAIL'].replace({'All Pub':'AllPub'},inplace = True)
df['STREET'].replace({'NoAccess':'No Access', 'Pavd':'Paved'},inplace = True)
```

BIVARIATE ANALYSIS

House Features

- INT_SQFT – The interior Sq. Ft of the property
- N_BEDROOM – The number of Bed rooms
- N_BATHROOM - The number of bathrooms
- N_ROOM – Total Number of Rooms
- QS_ROOMS – The quality score assigned for rooms based on buyer reviews
- QS_BATHROOM – The quality score assigned for bathroom based on buyer reviews
- QS_BEDROOM – The quality score assigned for bedroom based on buyer reviews
- QS_OVERALL – The Overall quality score assigned for the property
- SALE_COND – The Sale Condition
  - Normal: Normal Sale
  - Abnorml: Abnormal Sale - trade, foreclosure, short sale
  - AdjLand: Adjoining Land Purchase
  - Family: Sale between family members
  - Partial: Home was not completed when last assessed
- BUILDTYPE – The type of building
  - House
  - Commercial
  - Others

Surrounding and Locality

- AREA – The property in which the real estate is located
- DIST_MAINROAD – The distance of the property to the main road
- PARK_FACIL – Whether parking facility is available
- UTILITY_AVAIL
  - AllPub: All public Utilities (E,G,W,& S)
  - NoSewr: Electricity, Gas, and Water (Septic Tank)
  - NoSeWa: Electricity and Gas Only
  - ELO: Electricity only
- STREET
  - Gravel
  - Paved
  - No Access
- MZZONE
  - A: Agriculture
  - C: Commercial
  - I: Industrial
  - RH: Residential High Density
  - RL: Residential Low Density
  - RM: Residential Medium Density

House Sale Price

- PRT_ID – The Property Transaction ID assigned by ChennaiEstate
- COMMIS – The Commission paid to the agent
- SALES_PRICE – The total sale price of the property

In [45]:

```
df.columns
```

Out[45]:

```
Index(['PRT_ID', 'AREA', 'INT_SQFT', 'DIST_MAINROAD', 'N_BEDROOM',
       'N_BATHROOM', 'N_ROOM', 'SALE_COND', 'PARK_FACIL', 'BUILDTYPE',
       'UTILITY_AVAIL', 'STREET', 'MZZONE', 'QS_ROOMS', 'QS_BATHROOM',
       'QS_BEDROOM', 'QS_OVERALL', 'COMMIS', 'SALES_PRICE'],
      dtype='object')
```

Hypothesis -

- Sales price should increase with increase in interior square feet
- The sales price would depend on the area where house is located
- Higher the number of rooms, bathrooms in the house, more should be the price

1. House Features

- INT_SQFT – The interior Sq. Ft of the property
- N_BEDROOM – The number of Bed rooms
- N_BATHROOM - The number of bathrooms
- N_ROOM – Total Number of Rooms
- QS_ROOMS – The quality score assigned for rooms based on buyer reviews
- QS_BATHROOM – The quality score assigned for bathroom based on buyer reviews
- QS_BEDROOM – The quality score assigned for bedroom based on buyer reviews
- QS_OVERALL – The Overall quality score assigned for the property
- SALE_COND – The Sale Condition
- BUILDTYPE – The type of building

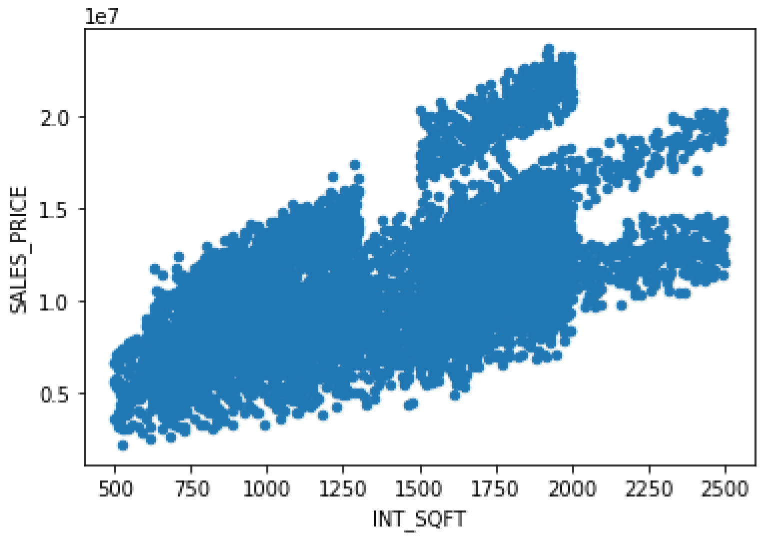
1. Interior area and sales price (target)

In [46]:

```
# interior area and sales price (target)

df.plot.scatter('INT_SQFT','SALES_PRICE')
```

Out[46]:

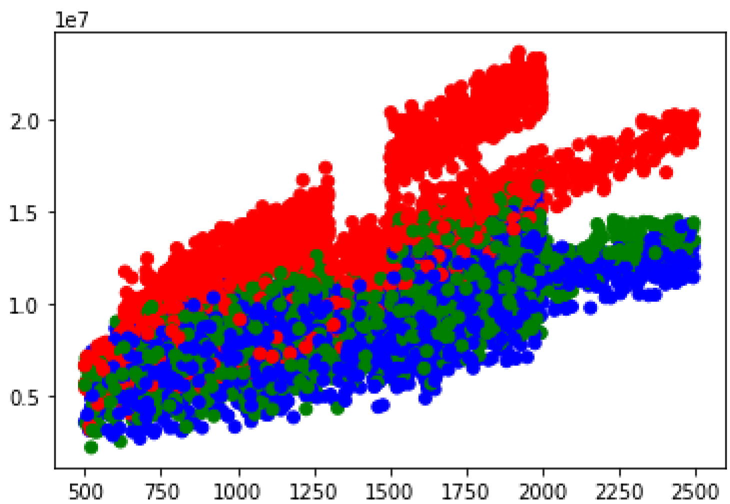
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f293a308748>
```



- A very clear linear relationship can be seen between the interior area and sales price
- These variables have a positive correlation

In [47]:

```
fig, ax = plt.subplots()
colors = {'Commercial':'red', 'House':'blue', 'Others':'green'}
ax.scatter(df['INT_SQFT'], df['SALES_PRICE'], c=df['BUILDTYPE'].apply(lambda x: colors[x]))
plt.show()
```



** 2. Sales Price against no of bedroom and bathroom**

In [48]:

```
# sale price of houses wrt number of bedrooms and bathroomms
df.pivot_table(values='SALES_PRICE', index='N_BEDROOM', columns='N_BATHROOM', aggfunc='medi
```

Out[48]:

N_BATHROOM	1.0	2.0
N_BEDROOM		
1.0	9168740.0	NaN
2.0	12129780.0	9125250.0
3.0	NaN	11663490.0
4.0	NaN	13172000.0

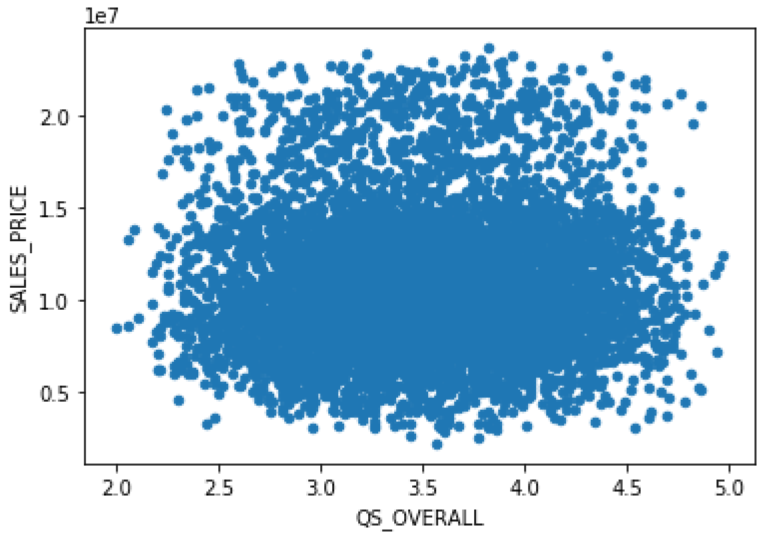
*3. QS_OVERALL and sales price *

In [49]:

```
#QS_OVERALL and sales price

df.plot.scatter('QS_OVERALL', 'SALES_PRICE')
```

Out[49]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f293a1e22e8>
```

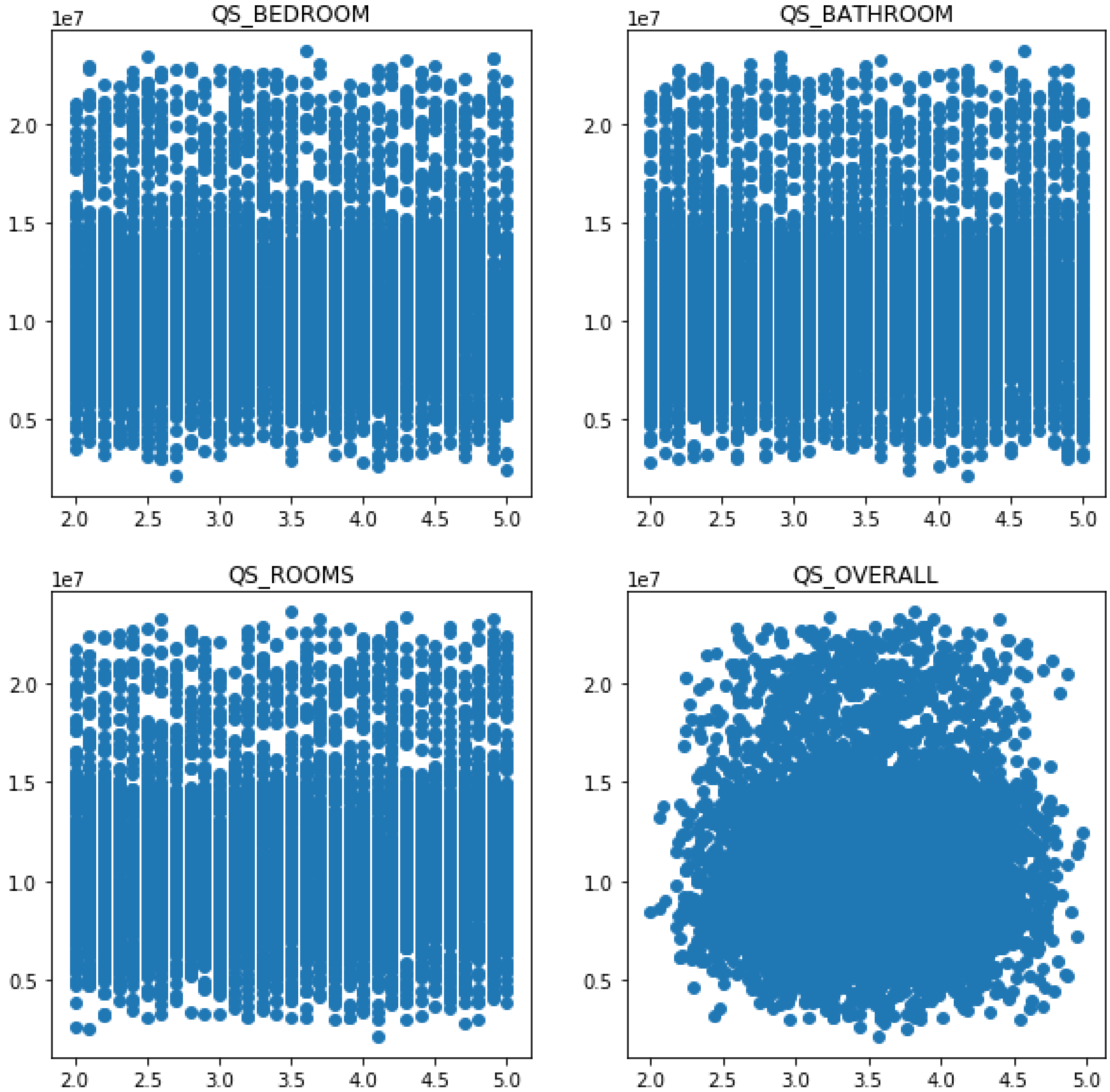


In [50]:

```
fig, axs = plt.subplots(2, 2)

fig.set_figheight(10)
fig.set_figwidth(10)

axs[0, 0].scatter(df['QS_BEDROOM'], df['SALES_PRICE'])    # QS_BEDROOM and sale price
axs[0, 0].set_title('QS_BEDROOM')

axs[0, 1].scatter(df['QS_BATHROOM'], df['SALES_PRICE'])   # QS_BATHROOM and price
axs[0, 1].set_title('QS_BATHROOM')

axs[1, 0].scatter(df['QS_ROOMS'], df['SALES_PRICE'])    # QS_ROOMS and sale price
axs[1, 0].set_title('QS_ROOMS')

axs[1, 1].scatter(df['QS_OVERALL'], df['SALES_PRICE'])    # QS_OVERALL and sale price
axs[1, 1].set_title('QS_OVERALL')
```

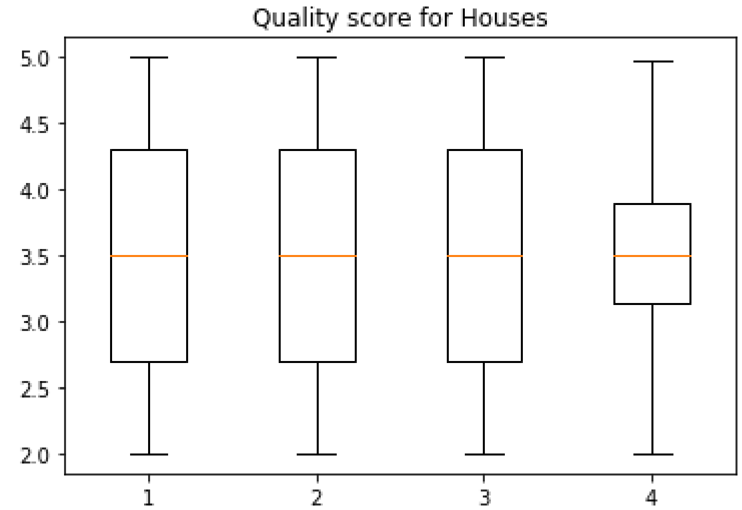
Out[50]:

```
Text(0.5,1,'QS_OVERALL')
```



In [51]:

```
# Create an axes instance
ax = plt.figure().add_subplot(111)
ax.set_title('Quality score for Houses')

# Create the boxplot
bp = ax.boxplot([df['QS_BEDROOM'], df['QS_ROOMS'], df['QS_BATHROOM'], df['QS_OVERALL']])
```



- Distribution of number of houses in each quartile is same for 'QS_ROOMS', 'QS_BATHROOM','QS_BEDROOM'
- For QS_OVERALL, 50 % of values lie in a very small range of ~3.2 to 3.7 score

** 4. Building type and sales price**

In [52]:

```
# SALE PRICE based on building type

df.groupby('BUILDTYPE').SALES_PRICE.median()
```

Out[52]:

```
BUILDTYPE
Commercial    13356200
House          8985370
Others         9637260
Name: SALES_PRICE, dtype: int64
```

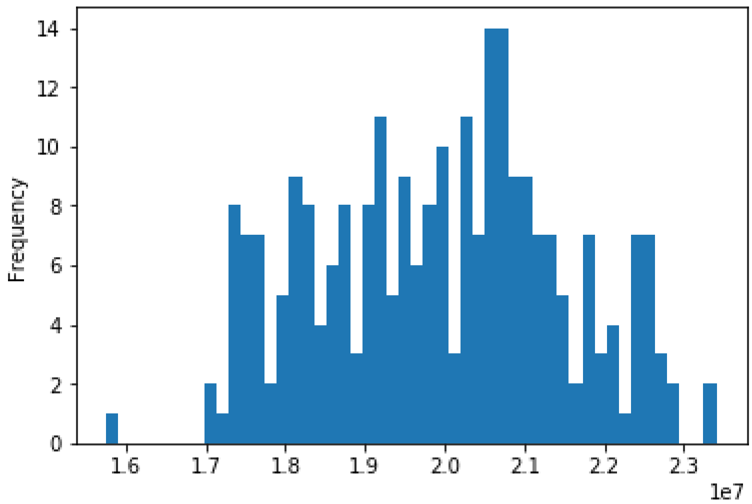
- Houses built for commercial purposes have a considerably higher sale price
- Houses with additional facility should have higher price

In [53]:

```
temp_df = df.loc[(df['BUILDTYPE']=='Commercial')&(df['AREA']=='Anna Nagar')]
temp_df['SALES_PRICE'].plot.hist(bins=50)
```

Out[53]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f293880fef0>
```

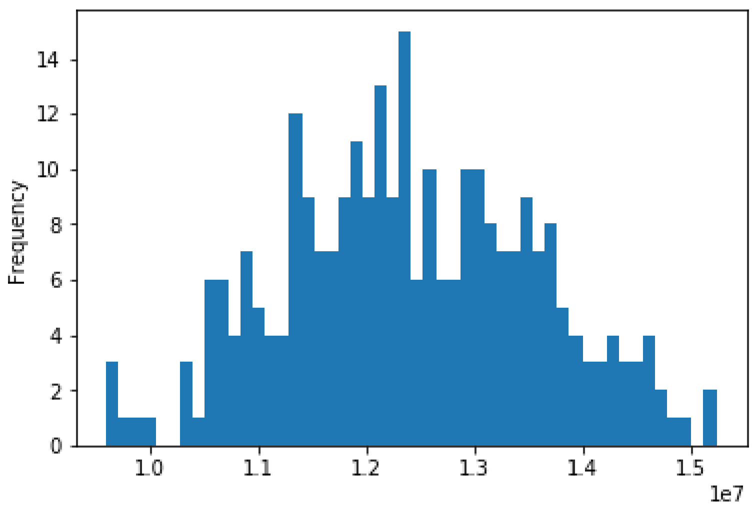


In [54]:

```
temp_df = df.loc[(df['BUILDTYPE']=='House')&(df['AREA']=='Anna Nagar')]
temp_df['SALES_PRICE'].plot.hist(bins=50)
```

Out[54]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f29387562b0>
```



Surrounding and Locality

- AREA – The property in which the real estate is located
- DIST_MAINROAD – The distance of the property to the main road
- PARK_FACIL – Whether parking facility is available
- UTILITY_AVAIL
- STREET
- MZZONE

5. Building type and parking facility

In [55]:

```
# building type and parking facility

df.groupby(['BUILDTYPE', 'PARK_FACIL']).SALES_PRICE.median()
```

Out[55]:

```
BUILDTYPE   PARK_FACIL
Commercial  No            12692985
            Yes           13920600
House       No             8514140
            Yes            9468150
Others      No             9104645
            Yes           10039405
Name: SALES_PRICE, dtype: int64
```

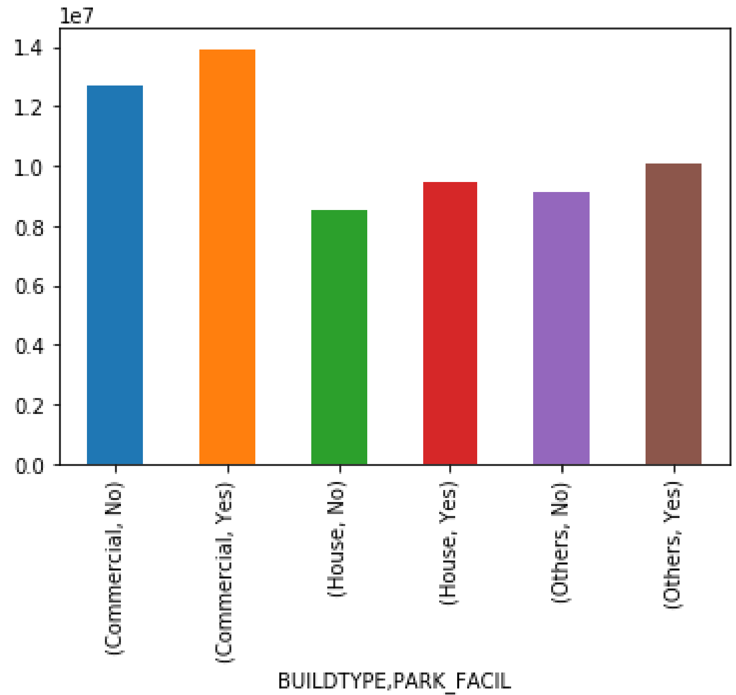
- For all three categories, houses with park facility have a higher price
- we can use groupby function to generate a plot for better comparison

In [56]:

```
temp = df.groupby(['BUILDTYPE', 'PARK_FACIL']).SALES_PRICE.median()
temp.plot(kind = 'bar', stacked = True)
```

Out[56]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f29386d17f0>
```



6. Area-wise price for houses

In [57]:

```
# average price for each area category

df.pivot_table(values='SALES_PRICE', index='AREA', aggfunc='median')
```

Out[57]:

AREA	SALES_PRICE
Adyar	8878350
Anna Nagar	13727895
Chrompet	9606725
KK Nagar	12146740
Karapakkam	7043125
T Nagar	14049650
Velachery	10494410

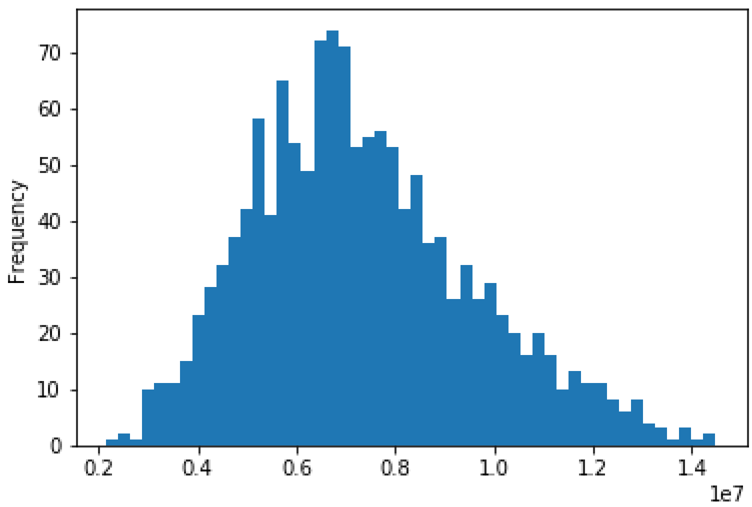
- Anna Nagar and T Nagar are comparatively more expensive
- The least priced are among the 7 is karapakkam

In [58]:

```
temp_df = df.loc[(df['AREA']=='Karapakkam')]
temp_df['SALES_PRICE'].plot.hist(bins=50)
```

Out[58]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f2938402f28>
```

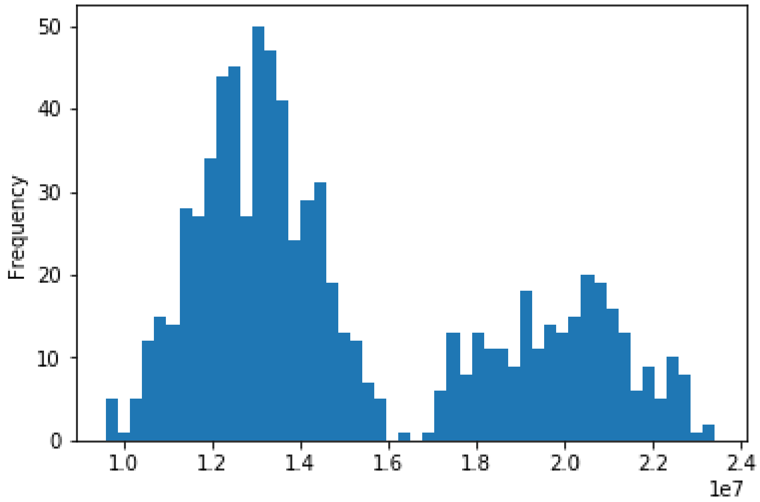


In [59]:

```
temp_df = df.loc[(df['AREA']=='Anna Nagar')]
temp_df['SALES_PRICE'].plot.hist(bins=50)
```

Out[59]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f293839da90>
```



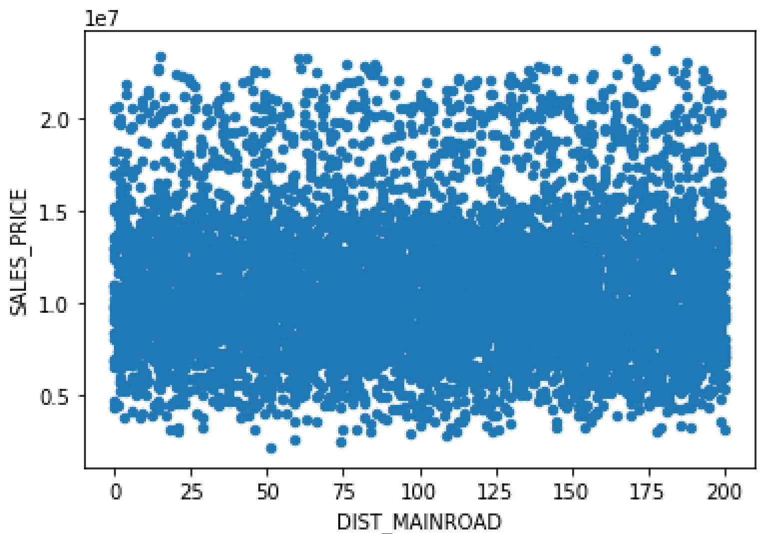
*7. Distance from main road *

In [60]:

```
df.plot.scatter('DIST_MAINROAD', 'SALES_PRICE')
```

Out[60]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f29382b1eb8>
```



8. Type of street around the house

In [61]:

```
df.groupby(['STREET']).SALES_PRICE.median()
```

Out[61]:

```
STREET
Gravel       10847225
No Access     9406050
Paved        10470070
Name: SALES_PRICE, dtype: int64
```

- Both gravel and paved roads have approximately same sale price
- Houses marked with 'no access' have a lower sale price

House Sale Price

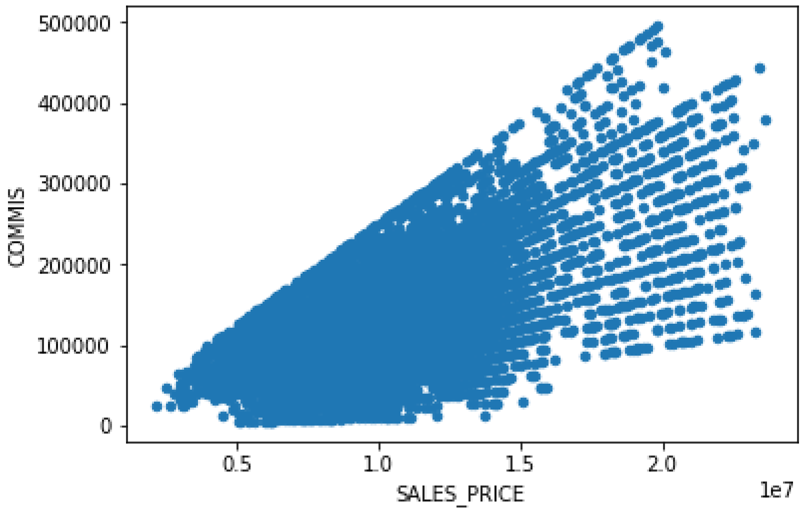
- PRT_ID – The Property Transaction ID assigned by ChennaiEstate
- COMMIS – The Commission paid to the agent
- SALES_PRICE – The total sale price of the property

In [62]:

```
# commission and sales price

df.plot.scatter('SALES_PRICE', 'COMMIS')
```

Out[62]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f293a2b8940>
```



In [63]:

```
df[['SALES_PRICE', 'COMMIS']].corr()
```

Out[63]:

	SALES_PRICE	COMMIS
SALES_PRICE	1.000000	0.626275
COMMIS	0.626275	1.000000

Linear Regression Model

In [64]:

```
df.drop(['PRT_ID'], axis=1, inplace = True)
```

In [65]:

```
df = pd.get_dummies(df)
```

In [66]:

```
x = df.drop('SALES_PRICE', axis=1)
y= df['SALES_PRICE']
```

Train Test Split

In [67]:

```
from sklearn.model_selection import train_test_split
train_x, valid_x, train_y, valid_y = train_test_split(x, y, test_size = 0.3, random_state =

train_x.shape, valid_x.shape, train_y.shape, valid_y.shape
```

Out[67]:

```
((4976, 48), (2133, 48), (4976,), (2133,))
```

In [68]:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_log_error
```

In [69]:

```
lreg = LinearRegression()
lreg.fit(train_x, train_y)
```

Out[69]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=None, normalize=Fal
se)
```

Model Evaluation - RMSLE

In [70]:

```
pred_train = lreg.predict(train_x)
train_score = np.sqrt(mean_squared_log_error(train_y,pred_train))
```

In [71]:

```
pred_test = lreg.predict(valid_x)
valid_score=np.sqrt(mean_squared_log_error(valid_y,pred_test))
```

In [72]:

```
print('Training score:', train_score)
print('Validation score:', valid_score)
```

```
Training score: 0.09097022122982201
Validation score: 0.0946013502150473
```

In [ ]:

```

```