In [1]:

```
#PREDICTION OF SALES PRICE
import pandas as pd
```

In [2]:

```
df=pd.read_csv('1. Regression - Module - (Housing Prices).csv')
```

In [3]:

```
df['Condition of the House'].head(5)
```

Out[3]:

```
0         Fair
1         Fair
2         Fair
3    Excellent
4         Fair
Name: Condition of the House, dtype: object
```

In [4]:

```
df['Sale Price'].describe()
```

Out[4]:

```
count    2.160900e+04
mean     5.401984e+05
std      3.673890e+05
min      7.500000e+04
25%      3.219500e+05
50%      4.500000e+05
75%      6.450000e+05
max      7.700000e+06
Name: Sale Price, dtype: float64
```

In [5]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column                                     Non-Null Count  Dtype  
---  ------                                     --------------  -----  
 0   ID                                         21613 non-null  int64  
 1   Date House was Sold                        21613 non-null  object 
 2   Sale Price                                 21609 non-null  float64
 3   No of Bedrooms                             21613 non-null  int64  
 4   No of Bathrooms                            21609 non-null  float64
 5   Flat Area (in Sqft)                        21604 non-null  float64
 6   Lot Area (in Sqft)                         21604 non-null  float64
 7   No of Floors                               21613 non-null  float64
 8   Waterfront View                            21613 non-null  object 
 9   No of Times Visited                        21613 non-null  object 
 10  Condition of the House                     21613 non-null  object 
 11  Overall Grade                              21613 non-null  int64  
 12  Area of the House from Basement (in Sqft)  21610 non-null  float64
 13  Basement Area (in Sqft)                    21613 non-null  int64  
 14  Age of House (in Years)                    21613 non-null  int64  
 15  Renovated Year                             21613 non-null  int64  
 16  Zipcode                                    21612 non-null  float64
 17  Latitude                                   21612 non-null  float64
 18  Longitude                                  21612 non-null  float64
 19  Living Area after Renovation (in Sqft)     21612 non-null  float64
 20  Lot Area after Renovation (in Sqft)        21613 non-null  int64  
dtypes: float64(10), int64(7), object(4)
memory usage: 3.5+ MB
```

In [6]:

```
df.describe()
```

Out[6]:

	ID	Sale Price	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	Overall Grade	Area of the House from Basement (in Sqft)	E Area
count	2.161300e+04	2.160900e+04	21613.000000	21609.000000	21604.000000	2.160400e+04	21613.000000	21613.000000	21610.000000	2161
mean	4.580302e+09	5.401984e+05	3.370842	2.114732	2079.931772	1.510776e+04	1.494309	7.623467	1788.344193	29
std	2.876566e+09	3.673890e+05	0.930062	0.770138	918.487597	4.142827e+04	0.539989	1.105439	827.982604	44
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	1.000000	290.000000	
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1429.250000	5.040000e+03	1.000000	7.000000	1190.000000	
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.617500e+03	1.500000	7.000000	1560.000000	
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068825e+04	2.000000	8.000000	2210.000000	56
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	10.000000	9410.000000	482

In [7]:

```
zip_cond=df.groupby(['Condition of the House'])['Sale Price'].mean()
```

In [8]:

```
zip_cond
```

Out[8]:

```
Condition of the House
Bad          334431.666667
Excellent    612577.742504
Fair         542130.611206
Good         521277.510567
Okay         327316.215116
Name: Sale Price, dtype: float64
```

In [9]:

```
l=[]
for i in zip_cond:
    l.append(i)
```

In [10]:

```
l
```

Out[10]:

```
[334431.6666666667,
 612577.7425044092,
 542130.6112061591,
 521277.51056710107,
 327316.2151162791]
```

In [11]:

```
labels=df['Condition of the House'].unique()
```

In [12]:

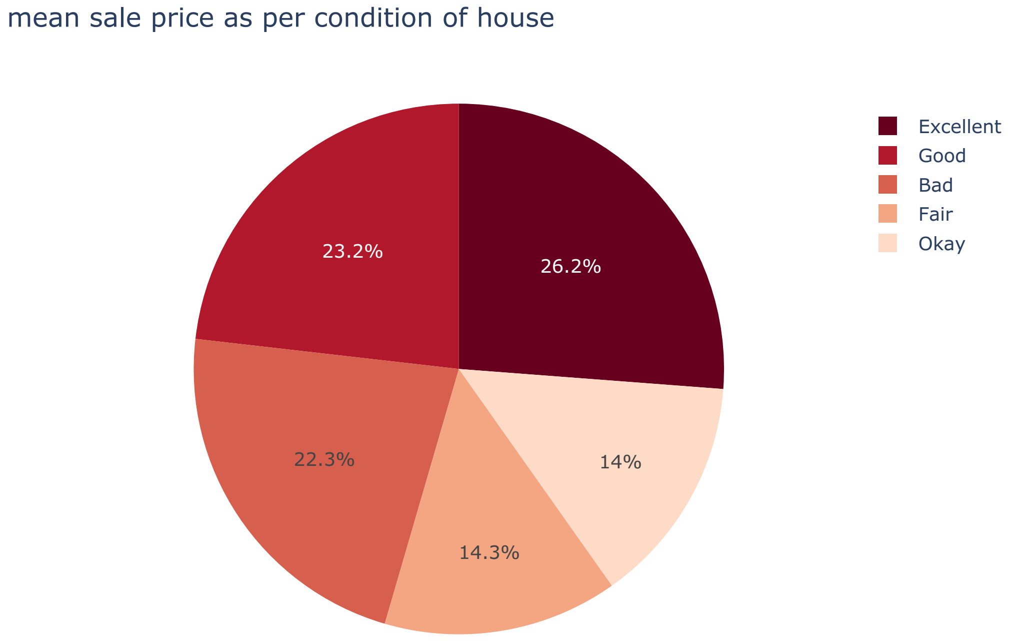
```
labels
```

Out[12]:

```
array(['Fair', 'Excellent', 'Good', 'Bad', 'Okay'], dtype=object)
```

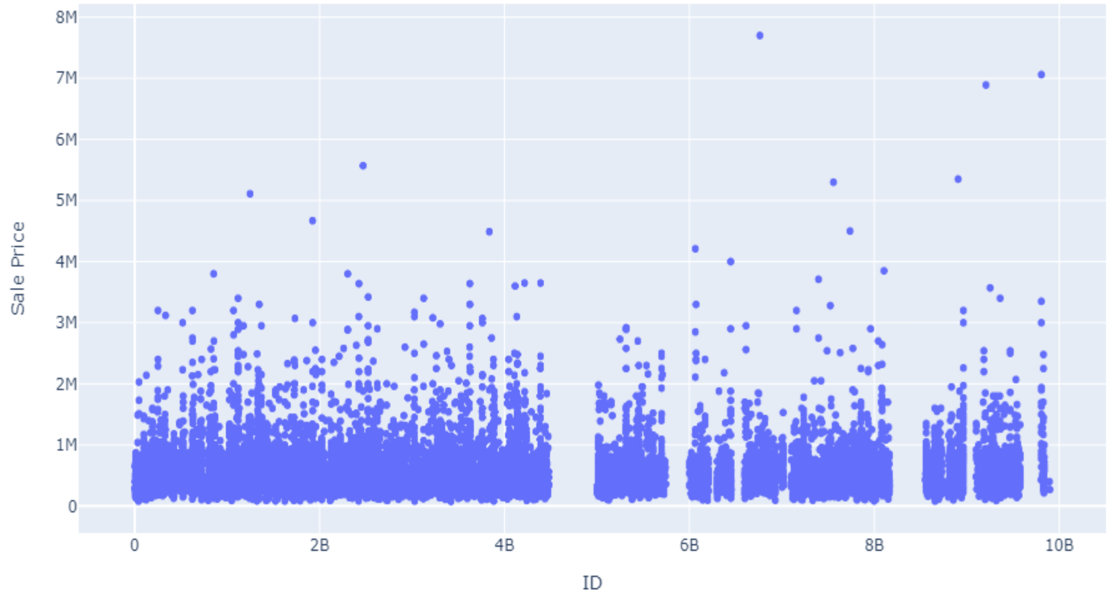
In [13]:

```
import numpy as np
import plotly.express as px
# This dataframe has 244 lines, but 4 distinct values for `day`
fig = px.pie(df, values=l, names=labels, title='mean sale price as per condition of house', color_discrete_sequ
fig.show()
```



In [14]:

```
#OUTLIERS IN GRAPHICAL VIEW
fig = px.scatter(df, x='ID', y="Sale Price")
fig.show()
```



In [15]:

```
#IDENTIFY OUTLIERS IN MATHEMATICAL FORM with inter quarntile method
q3=df['Sale Price'].quantile(.75)
q1=df['Sale Price'].quantile(.25)
```

In [16]:

```
iqr=q3-q1
```

In [17]:

```
iqr
```

Out[17]:

```
323050.0
```

In [18]:

```
lower_limit=q1-1.5*iqr
```

In [19]:

```
lower_limit
```

Out[19]:

```
-162625.0
```

In [20]:

```
upper_limit=q1+1.5*iqr
```

In [21]:

```
#perform imputing method to treat with outliers(independent variables)
def limit_imputer(value):
    if value>upper_limit:
        return upper_limit
    if value<lower_limit:
        return lower_limit
    else:
        return value
df['Sale Price']=df['Sale Price'].apply(limit_imputer)
```

In [22]:

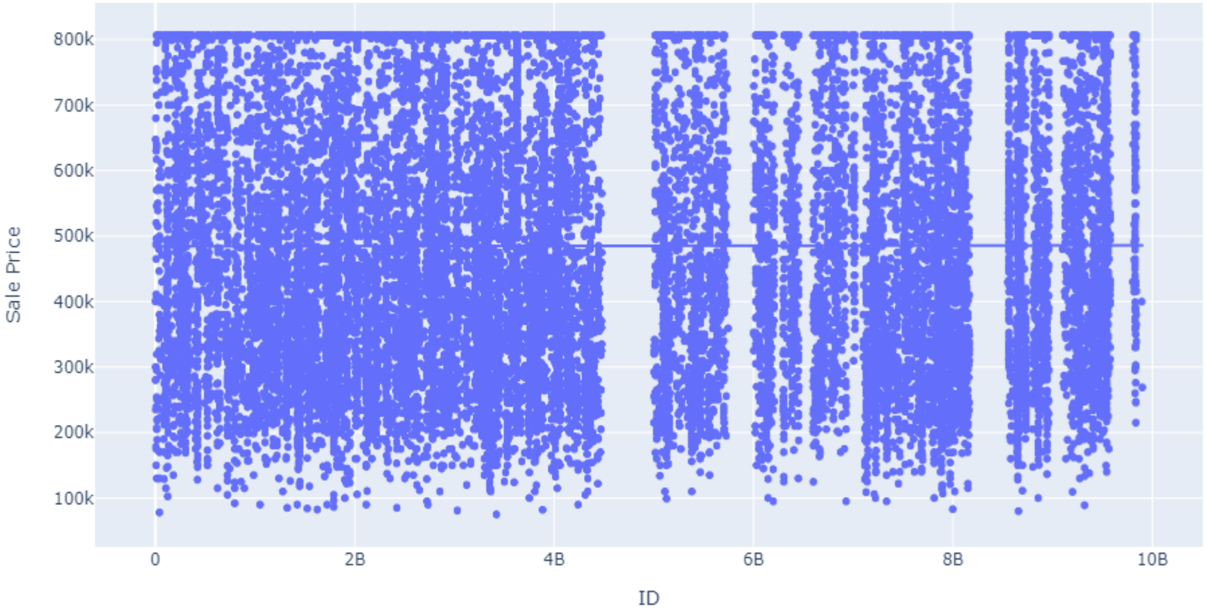
```
df['Sale Price'].describe()
```

Out[22]:

```
count     21609.000000
mean     485048.221667
std      197301.596234
min       75000.000000
25%      321950.000000
50%      450000.000000
75%      645000.000000
max      806525.000000
Name: Sale Price, dtype: float64
```

In [23]:

```
fig = px.scatter(df, x='ID', y="Sale Price",trendline='ols')
fig.show()
```



In [24]:
```
#treat with missing values(target variables)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column                                     Non-Null Count  Dtype  
---  ------                                     --------------  -----  
 0   ID                                         21613 non-null  int64  
 1   Date House was Sold                        21613 non-null  object 
 2   Sale Price                                 21609 non-null  float64
 3   No of Bedrooms                             21613 non-null  int64  
 4   No of Bathrooms                            21609 non-null  float64
 5   Flat Area (in Sqft)                        21604 non-null  float64
 6   Lot Area (in Sqft)                         21604 non-null  float64
 7   No of Floors                               21613 non-null  float64
 8   Waterfront View                            21613 non-null  object 
 9   No of Times Visited                        21613 non-null  object 
 10  Condition of the House                     21613 non-null  object 
 11  Overall Grade                              21613 non-null  int64  
 12  Area of the House from Basement (in Sqft)  21610 non-null  float64
 13  Basement Area (in Sqft)                    21613 non-null  int64  
 14  Age of House (in Years)                    21613 non-null  int64  
 15  Renovated Year                             21613 non-null  int64  
 16  Zipcode                                    21612 non-null  float64
 17  Latitude                                   21612 non-null  float64
 18  Longitude                                  21612 non-null  float64
 19  Living Area after Renovation (in Sqft)     21612 non-null  float64
 20  Lot Area after Renovation (in Sqft)        21613 non-null  int64  
dtypes: float64(10), int64(7), object(4)
memory usage: 3.5+ MB
```

In [25]:
```
df.dropna(inplace=True,axis=0,subset=['Sale Price'])
```

In [26]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21609 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column                                     Non-Null Count  Dtype  
---  ------                                     --------------  -----  
 0   ID                                         21609 non-null  int64  
 1   Date House was Sold                        21609 non-null  object 
 2   Sale Price                                 21609 non-null  float64
 3   No of Bedrooms                             21609 non-null  int64  
 4   No of Bathrooms                            21605 non-null  float64
 5   Flat Area (in Sqft)                        21600 non-null  float64
 6   Lot Area (in Sqft)                         21600 non-null  float64
 7   No of Floors                               21609 non-null  float64
 8   Waterfront View                            21609 non-null  object 
 9   No of Times Visited                        21609 non-null  object 
 10  Condition of the House                     21609 non-null  object 
 11  Overall Grade                              21609 non-null  int64  
 12  Area of the House from Basement (in Sqft)  21606 non-null  float64
 13  Basement Area (in Sqft)                    21609 non-null  int64  
 14  Age of House (in Years)                    21609 non-null  int64  
 15  Renovated Year                             21609 non-null  int64  
 16  Zipcode                                    21608 non-null  float64
 17  Latitude                                   21608 non-null  float64
 18  Longitude                                  21608 non-null  float64
 19  Living Area after Renovation (in Sqft)     21608 non-null  float64
 20  Lot Area after Renovation (in Sqft)        21609 non-null  int64  
dtypes: float64(10), int64(7), object(4)
memory usage: 3.6+ MB
```

In [30]:

```python
#treat misiing values of (independent variables)
num_col=['No of Bathrooms','Flat Area (in Sqft)','Lot Area (in Sqft)','Area of the House from Basement (in Sqft
```

In [31]:

```python
from sklearn.impute import SimpleImputer
imputer=SimpleImputer(missing_values=np.nan,strategy='median')
df[num_col]=imputer.fit_transform(df[num_col])
```

In [32]:

```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21609 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column                                     Non-Null Count  Dtype  
---  ------                                     --------------  -----  
 0   ID                                         21609 non-null  int64  
 1   Date House was Sold                        21609 non-null  object 
 2   Sale Price                                 21609 non-null  float64
 3   No of Bedrooms                             21609 non-null  int64  
 4   No of Bathrooms                            21609 non-null  float64
 5   Flat Area (in Sqft)                        21609 non-null  float64
 6   Lot Area (in Sqft)                         21609 non-null  float64
 7   No of Floors                               21609 non-null  float64
 8   Waterfront View                            21609 non-null  object 
 9   No of Times Visited                        21609 non-null  object 
 10  Condition of the House                     21609 non-null  object 
 11  Overall Grade                              21609 non-null  int64  
 12  Area of the House from Basement (in Sqft)  21609 non-null  float64
 13  Basement Area (in Sqft)                    21609 non-null  int64  
 14  Age of House (in Years)                    21609 non-null  int64  
 15  Renovated Year                             21609 non-null  int64  
 16  Zipcode                                    21608 non-null  float64
 17  Latitude                                   21609 non-null  float64
 18  Longitude                                  21609 non-null  float64
 19  Living Area after Renovation (in Sqft)     21609 non-null  float64
 20  Lot Area after Renovation (in Sqft)        21609 non-null  int64  
dtypes: float64(10), int64(7), object(4)
memory usage: 3.6+ MB
```

In [36]:

```python
#IMP: ZIPCODE CANT BE TREATED AS CONTINUOUS VARIABLE ,IT SHOULD BE TREATED AS CATEGORICAL VARIABLE SO,THIS WAY
imputer=SimpleImputer(missing_values=np.nan,strategy='most_frequent')
df['Zipcode']=imputer.fit_transform(df['Zipcode'])
#BUT HERE IS AN IMP ERROR TO UNDERSTAND
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[36], line 3
      1 #IMP: ZIPCODE CANT BE TREATED AS CONTINUOUS VARIABLE ,IT SHOULD BE TREATED AS CATEGORICAL VARIABLE SO,
      2 imputer=SimpleImputer(missing_values=np.nan,strategy='most_frequent')
----> 3 df['Zipcode']=imputer.fit_transform(df['Zipcode'])

File ~\anaconda3\lib\site-packages\sklearn\utils\_set_output.py:142, in _wrap_method_output.<locals>.wrapped(se
lf, X, *args, **kwargs)
    140 @wraps(f)
    141 def wrapped(self, X, *args, **kwargs):
--> 142     data_to_wrap = f(self, X, *args, **kwargs)
    143     if isinstance(data_to_wrap, tuple):
    144         # only wrap the first output for cross decomposition
    145         return (
    146             _wrap_data_with_container(method, data_to_wrap[0], X, self),
    147             *data_to_wrap[1:],
    148         )

File ~\anaconda3\lib\site-packages\sklearn\base.py:859, in TransformerMixin.fit_transform(self, X, y, **fit_par
ams)
    855 # non-optimized default implementation; override when a better
    856 # method is possible for a given clustering algorithm
    857 if y is None:
    858     # fit method of arity 1 (unsupervised transformation)
--> 859     return self.fit(X, **fit_params).transform(X)
    860 else:
    861     # fit method of arity 2 (supervised transformation)
    862     return self.fit(X, y, **fit_params).transform(X)

File ~\anaconda3\lib\site-packages\sklearn\impute\_base.py:390, in SimpleImputer.fit(self, X, y)
    381 if self.verbose != "deprecated":
    382     warnings.warn(
    383         "The 'verbose' parameter was deprecated in version "
    384         "1.1 and will be removed in 1.3. A warning will "
   (...)
    387         FutureWarning,
    388     )
--> 390 X = self._validate_input(X, in_fit=True)
    392 # default fill_value is 0 for numerical input and "missing_value"
    393 # otherwise
    394 if self.fill_value is None:

File ~\anaconda3\lib\site-packages\sklearn\impute\_base.py:344, in SimpleImputer._validate_input(self, X, in_fi
t)
    342         raise new_ve from None
    343     else:
--> 344         raise ve
    346 if in_fit:
    347     # Use the dtype seen in `fit` for non-`fit` conversion
    348     self._fit_dtype = X.dtype

File ~\anaconda3\lib\site-packages\sklearn\impute\_base.py:327, in SimpleImputer._validate_input(self, X, in_fi
t)
    324     force_all_finite = True
    326 try:
--> 327     X = self._validate_data(
    328         X,
    329         reset=in_fit,
    330         accept_sparse="csc",
    331         dtype=dtype,
    332         force_all_finite=force_all_finite,
    333         copy=self.copy,
    334     )
    335 except ValueError as ve:
    336     if "could not convert" in str(ve):

File ~\anaconda3\lib\site-packages\sklearn\base.py:546, in BaseEstimator._validate_data(self, X, y, reset, vali
date_separately, **check_params)
    544     raise ValueError("Validation should be done on X, y or both.")
    545 elif not no_val_X and no_val_y:
--> 546     X = check_array(X, input_name="X", **check_params)
    547     out = X
    548 elif no_val_X and not no_val_y:

File ~\anaconda3\lib\site-packages\sklearn\utils\validation.py:902, in check_array(array, accept_sparse, accept
_large_sparse, dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_featur
es, estimator, input_name)
    900     # If input is 1D raise error
    901     if array.ndim == 1:
--> 902         raise ValueError(
    903             "Expected 2D array, got 1D array instead:\narray={}.\n"
    904             "Reshape your data either using array.reshape(-1, 1) if "
    905             "your data has a single feature or array.reshape(1, -1) "
    906             "if it contains a single sample.".format(array)
    907         )
    909 if dtype_numeric and array.dtype.kind in "USV":
    910     raise ValueError(
    911         "dtype='numeric' is not compatible with arrays of bytes/strings."
    912         "Convert your data to numeric values explicitly instead."
    913     )

ValueError: Expected 2D array, got 1D array instead:
array=[98178. 98125. 98028. ... 98144. 98027. 98144.].
Reshape your data either using array.reshape(-1, 1) if your data has a single feature or array.reshape(1, -1) i
f it contains a single sample.
```

In [37]:

```
df['Zipcode'].shape
```

Out[37]:

```
(21609,)
```

In [49]:

```
#we need to convert our array in 2d
column=df['Zipcode'].values.reshape(-1,1)
```

In [50]:
```
column.shape
```

Out[50]:
```
(21609, 1)
```

In [59]:
```
#as our zipcode is in 2d,now we can do fit transform again for zipcode column
column=df['Zipcode'].values.reshape(-1,1)
imputer=SimpleImputer(missing_values=np.nan,strategy='most_frequent')
df['Zipcode']=imputer.fit_transform(column)
```

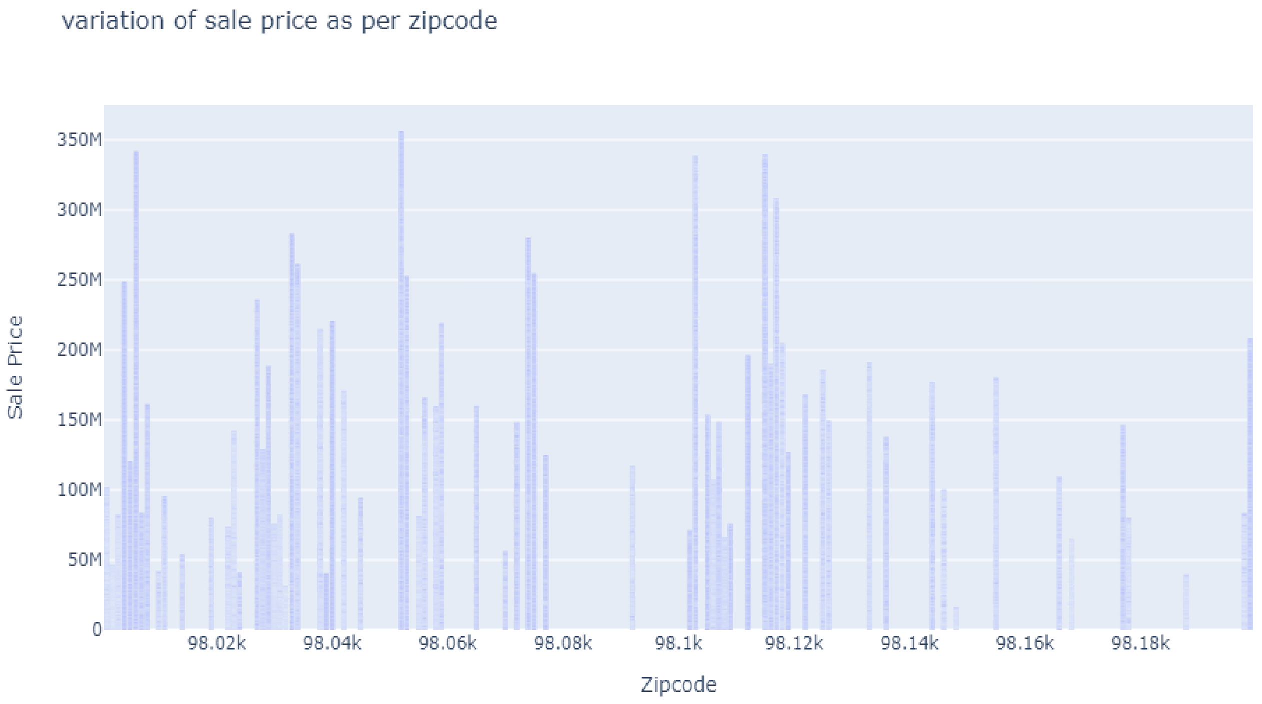
In [60]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21609 entries, 0 to 21612
Data columns (total 22 columns):
 #   Column                                     Non-Null Count  Dtype  
---  ------                                     --------------  -----  
 0   ID                                         21609 non-null  int64  
 1   Date House was Sold                        21609 non-null  object 
 2   Sale Price                                 21609 non-null  float64
 3   No of Bedrooms                             21609 non-null  int64  
 4   No of Bathrooms                            21609 non-null  float64
 5   Flat Area (in Sqft)                        21609 non-null  float64
 6   Lot Area (in Sqft)                         21609 non-null  float64
 7   No of Floors                               21609 non-null  float64
 8   Waterfront View                            21609 non-null  object 
 9   No of Times Visited                        21609 non-null  object 
 10  Condition of the House                     21609 non-null  object 
 11  Overall Grade                              21609 non-null  int64  
 12  Area of the House from Basement (in Sqft)  21609 non-null  float64
 13  Basement Area (in Sqft)                    21609 non-null  int64  
 14  Age of House (in Years)                    21609 non-null  int64  
 15  Renovated Year                             21609 non-null  int64  
 16  Zipcode                                    21609 non-null  float64
 17  Latitude                                   21609 non-null  float64
 18  Longitude                                  21609 non-null  float64
 19  Living Area after Renovation (in Sqft)     21609 non-null  float64
 20  Lot Area after Renovation (in Sqft)        21609 non-null  int64  
 21  zipcode                                    21609 non-null  float64
dtypes: float64(11), int64(7), object(4)
memory usage: 3.8+ MB
```

In [61]:
```
#variable transformaton
#we cannot have direct relation with sale price and zip code so we will but we can check sale price zipcode wis
df['Zipcode']=df['Zipcode'].astype(object)
df.dtypes
```

Out[61]:
```
ID                                             int64
Date House was Sold                           object
Sale Price                                   float64
No of Bedrooms                                 int64
No of Bathrooms                              float64
Flat Area (in Sqft)                          float64
Lot Area (in Sqft)                           float64
No of Floors                                 float64
Waterfront View                               object
No of Times Visited                           object
Condition of the House                        object
Overall Grade                                  int64
Area of the House from Basement (in Sqft)    float64
Basement Area (in Sqft)                        int64
Age of House (in Years)                        int64
Renovated Year                                 int64
Zipcode                                       object
Latitude                                     float64
Longitude                                    float64
Living Area after Renovation (in Sqft)       float64
Lot Area after Renovation (in Sqft)            int64
zipcode                                      float64
dtype: object
```

In [70]:
```
import plotly.express as px
fig = px.bar(df, x = "Zipcode", y = "Sale Price",title='variation of sale price as per zipcode')
fig.show()
```



In [73]:

```
#variable transformation over 'no of times visited' col as it does not represent data properly
df['No of Times Visited'].unique()
```

Out[73]:

```
array(['None', 'Thrice', 'Four', 'Twice', 'Once'], dtype=object)
```

In [74]:

```
mapping={'None':'0', 'Thrice':'3', 'Four':'4', 'Twice':'2', 'Once':'1'}
```

In [75]:

```
df['No of Times Visited']=df['No of Times Visited'].map(mapping)
```

In [76]:

```
df['No of Times Visited'].unique()
```

Out[76]:

```
array(['0', '3', '4', '2', '1'], dtype=object)
```

In [77]:

```
#variable transformation used to create new variable by combining or transforming 2 variables
#by looking at 'renovation year' we can have que when renovation was done and how it can impact sale price
#create 2 variables 'ever renovated' and 'years since renovataion'
df['ever renovated']=np.where(df['Renovated Year']==0,'No','Yes')
```

In [80]:

```
#for 2nd variable fetch year from 'date of sold col'
df['purchase year']=pd.DatetimeIndex(df['Date House was Sold']).year
```

In [82]:

```
df['purchase year'].head(5)
```

Out[82]:

```
0    2017
1    2017
2    2016
3    2017
4    2016
Name: purchase year, dtype: int64
```

In [91]:

```
df['years since renovataion']=np.where(df['ever renovated']=='Yes',abs(df['purchase year']-df['Renovated Year']
```

In [92]:

```
df['years since renovataion'].tail(5)
```

Out[92]:

```
21608    0
21609    0
21610    0
21611    0
21612    0
Name: years since renovataion, dtype: int64
```

In [93]:

```
df.head()
```

Out[93]:

	ID	Date House was Sold	Sale Price	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	Waterfront View	No of Times Visited	...	Renovated Year	Zipcode	Latitude	L
0	7129300520	14 October 2017	221900.0	3	1.00	1180.0	5650.0	1.0	No	0	...	0	98178.0	47.5112	
1	6414100192	14 December 2017	538000.0	3	2.25	2570.0	7242.0	2.0	No	0	...	1991	98125.0	47.7210	
2	5631500400	15 February 2016	180000.0	2	1.00	770.0	10000.0	1.0	No	0	...	0	98028.0	47.7379	
3	2487200875	14 December 2017	604000.0	4	3.00	1960.0	5000.0	1.0	No	0	...	0	98136.0	47.5208	
4	1954400510	15 February 2016	510000.0	3	2.00	1680.0	8080.0	1.0	No	0	...	0	98074.0	47.6168	

5 rows × 25 columns

In [94]:

```
#AS WE SEE WE GOT OUR COLOUMN NOW WE DONT REQUIRE USELESS COLUMNS,WE WILL DELETE THOSE TO KEEP OUR DATA TIDY
df.drop(columns=['Date House was Sold','Renovated Year','purchase year'],inplace=True)
```

In [95]:

```
df.head()
```

Out[95]:

	ID	Sale Price	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	Waterfront View	No of Times Visited	Condition of the House	...	Basement Area (in Sqft)	Age of House (in Years)	Zipcode	Lati
0	7129300520	221900.0	3	1.00	1180.0	5650.0	1.0	No	0	Fair	...	0	63	98178.0	47.
1	6414100192	538000.0	3	2.25	2570.0	7242.0	2.0	No	0	Fair	...	400	67	98125.0	47.
2	5631500400	180000.0	2	1.00	770.0	10000.0	1.0	No	0	Fair	...	0	85	98028.0	47.
3	2487200875	604000.0	4	3.00	1960.0	5000.0	1.0	No	0	Excellent	...	910	53	98136.0	47.
4	1954400510	510000.0	3	2.00	1680.0	8080.0	1.0	No	0	Fair	...	0	31	98074.0	47.

5 rows × 22 columns

In [96]:

```
#correlation calculation
df['Sale Price'].corr(df['Flat Area (in Sqft)'])
```

Out[96]:

```
0.6492472259786739
```

In [97]:

```
#correlation between independent variables
df.drop(columns=['ID']).corr()
```

```
C:\Users\Janhavi\AppData\Local\Temp\ipykernel_8236\3805217886.py:2: FutureWarning:

The default value of numeric_only in DataFrame.corr is deprecated. In a future version, it will default to Fals
e. Select only valid columns or specify the value of numeric_only to silence this warning.
```

Out[97]:

	Sale Price	No of Bedrooms	No of Bathrooms	Flat Area (in Sqft)	Lot Area (in Sqft)	No of Floors	Overall Grade	Area of the House from Basement (in Sqft)	Basement Area (in Sqft)	Age of House (in Years)	Latitude	Longitu
Sale Price	1.000000	0.333635	0.509890	0.649247	0.101291	0.303304	0.664972	0.568595	0.283494	-0.066771	0.453716	0.0631
No of Bedrooms	0.333635	1.000000	0.515813	0.576628	0.031692	0.175536	0.349223	0.477549	0.303294	-0.154113	-0.008708	0.1295
No of Bathrooms	0.509890	0.515813	1.000000	0.754568	0.087732	0.500776	0.635638	0.685088	0.283798	-0.505954	0.024570	0.2231
Flat Area (in Sqft)	0.649247	0.576628	0.754568	1.000000	0.172721	0.354142	0.705725	0.876226	0.435142	-0.318146	0.052538	0.2400
Lot Area (in Sqft)	0.101291	0.031692	0.087732	0.172721	1.000000	-0.005162	0.102314	0.183492	0.015252	-0.053119	-0.085719	0.2294
No of Floors	0.303304	0.175536	0.500776	0.354142	-0.005162	1.000000	0.461368	0.524031	-0.245572	-0.489244	0.049692	0.1256
Overall Grade	0.664972	0.349223	0.635638	0.705725	0.102314	0.461368	1.000000	0.705153	0.145232	-0.456711	0.111226	0.2017
Area of the House from Basement (in Sqft)	0.568595	0.477549	0.685088	0.876226	0.183492	0.524031	0.705153	1.000000	-0.051825	-0.423848	-0.000819	0.3437
Basement Area (in Sqft)	0.283494	0.303294	0.283798	0.435142	0.015252	-0.245572	0.145232	-0.051825	1.000000	0.133072	0.110451	-0.1448
Age of House (in Years)	-0.066771	-0.154113	-0.505954	-0.318146	-0.053119	-0.489244	-0.456711	-0.423848	0.133072	1.000000	0.148083	-0.4095
Latitude	0.453716	-0.008708	0.024570	0.052538	-0.085719	0.049692	0.111226	-0.000819	0.110451	0.148083	1.000000	-0.1355
Longitude	0.063154	0.129569	0.223171	0.240091	0.229449	0.125620	0.201736	0.343793	-0.144822	-0.409515	-0.135551	1.0000
Living Area after Renovation (in Sqft)	0.600540	0.391771	0.568568	0.756185	0.144507	0.280106	0.681362	0.731996	0.200302	-0.326307	0.048836	0.3345
Lot Area after Renovation (in Sqft)	0.092041	0.029264	0.087226	0.183223	0.718527	-0.011204	0.107581	0.194106	0.017263	-0.071016	-0.086420	0.2544
zipcode	-0.033636	-0.152760	-0.203951	-0.199380	-0.129551	-0.059222	-0.185844	-0.261124	0.074933	0.346928	0.267022	-0.5641
years since renovataion	0.054093	-0.007198	0.003551	0.023503	0.013835	-0.000901	-0.024388	0.010491	0.029158	0.203375	0.019739	-0.0550

In [98]:

```
#how to identify categorical variable: variable with datatype object
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 21609 entries, 0 to 21612
Data columns (total 22 columns):
 #   Column                                    Non-Null Count  Dtype  
---  ------                                    --------------  -----  
 0   ID                                        21609 non-null  int64  
 1   Sale Price                                21609 non-null  float64
 2   No of Bedrooms                            21609 non-null  int64  
 3   No of Bathrooms                           21609 non-null  float64
 4   Flat Area (in Sqft)                       21609 non-null  float64
 5   Lot Area (in Sqft)                        21609 non-null  float64
 6   No of Floors                              21609 non-null  float64
 7   Waterfront View                           21609 non-null  object 
 8   No of Times Visited                       21609 non-null  object 
 9   Condition of the House                    21609 non-null  object 
 10  Overall Grade                             21609 non-null  int64  
 11  Area of the House from Basement (in Sqft)  21609 non-null  float64
 12  Basement Area (in Sqft)                   21609 non-null  int64  
 13  Age of House (in Years)                   21609 non-null  int64  
 14  Zipcode                                   21609 non-null  object 
 15  Latitude                                  21609 non-null  float64
 16  Longitude                                 21609 non-null  float64
 17  Living Area after Renovation (in Sqft)    21609 non-null  float64
 18  Lot Area after Renovation (in Sqft)       21609 non-null  int64  
 19  zipcode                                   21609 non-null  float64
 20  ever renovated                            21609 non-null  object 
 21  years since renovataion                   21609 non-null  int64  
dtypes: float64(10), int64(7), object(5)
memory usage: 3.8+ MB
```

In [99]:

```
df['Condition of the House'].value_counts()
```

Out[99]:

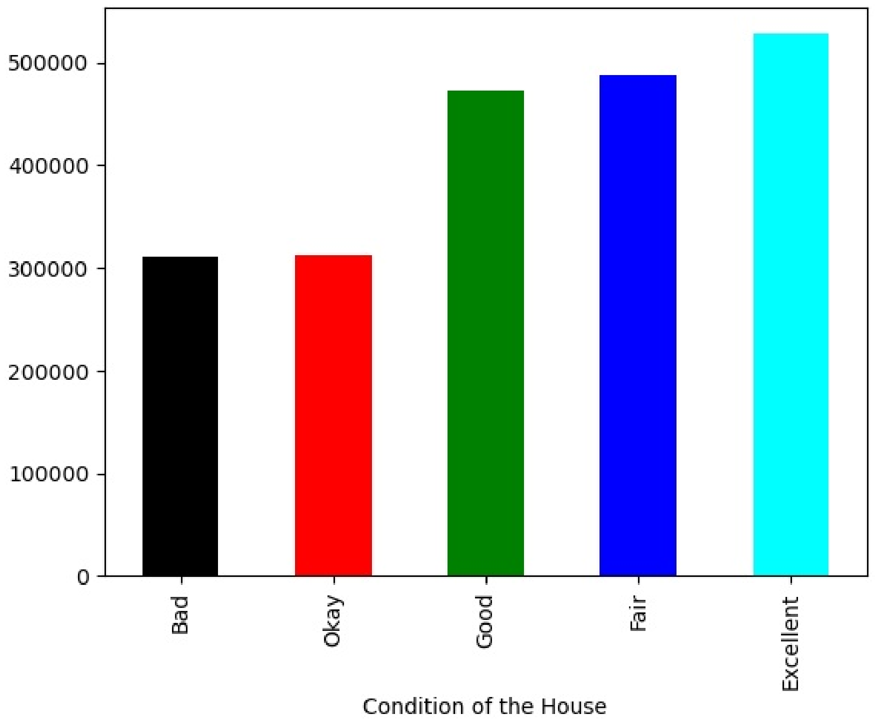
```
Fair         14028
Good          5678
Excellent     1701
Okay           172
Bad             30
Name: Condition of the House, dtype: int64
```

In [114…

```
#relationship of independent with dependent variables
df.groupby('Condition of the House')['Sale Price'].mean().sort_values().plot(kind='bar',color=['black', 'red',
```

Out[114]:

```
<Axes: xlabel='Condition of the House'>
```



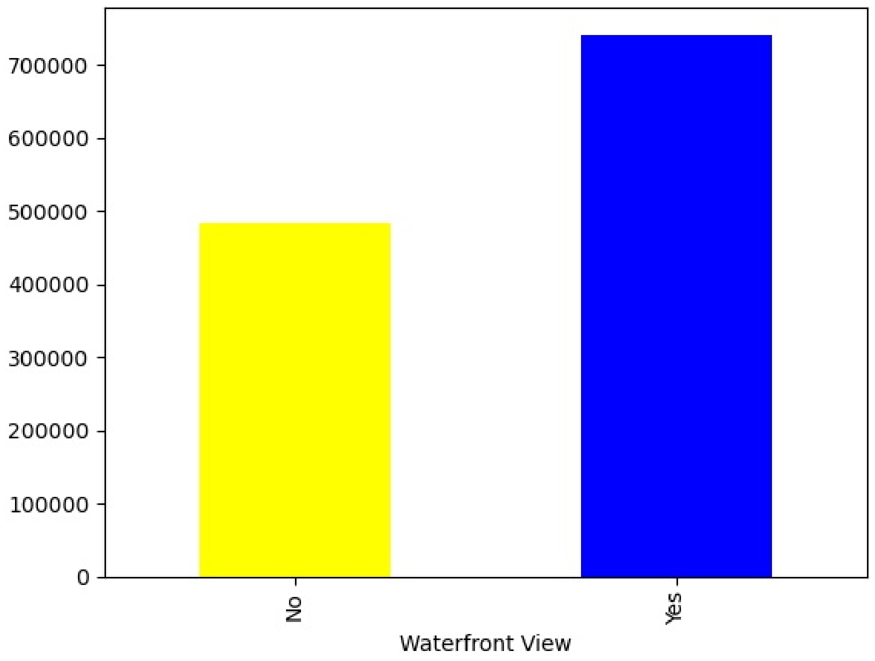
In [ ]:

In [117…

```
df.groupby('Waterfront View')['Sale Price'].mean().sort_values().plot(kind='bar',color=['yellow', 'blue'])
```

Out[117]:

```
<Axes: xlabel='Waterfront View'>
```

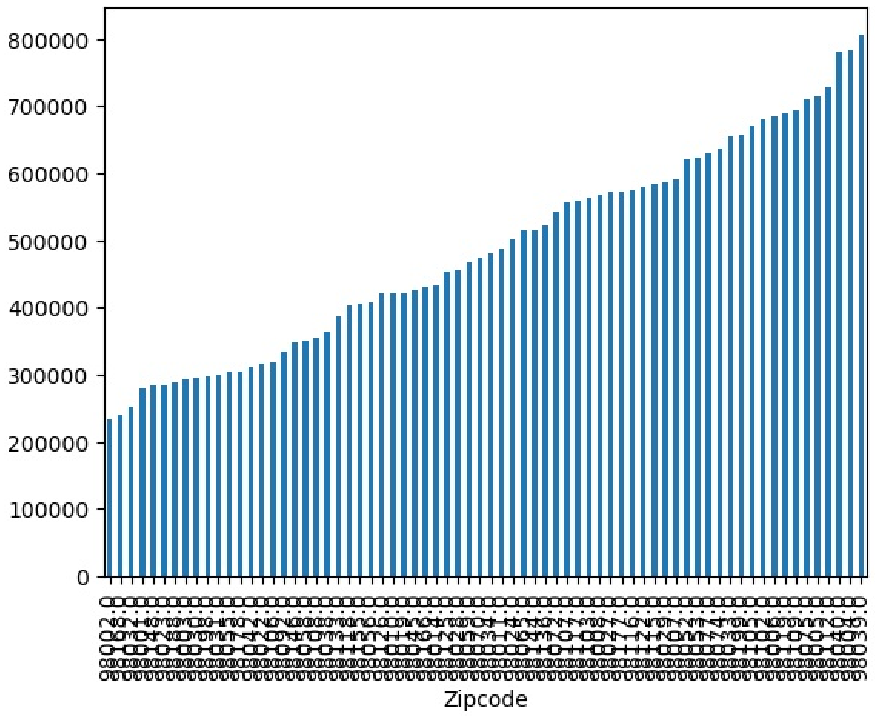


In [118…

```
df.groupby('Zipcode')['Sale Price'].mean().sort_values().plot(kind='bar')
```

Out[118]:

```
<Axes: xlabel='Zipcode'>
```


800000
700000
600000
500000
400000
300000
200000
100000
0
98002.0
98039.0
Zipcode