In [100…

```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
```

In [101…

```python
data=pd.read_csv("f:abc/house price prediction1.csv")
```

In [102…

```python
data
```

Out[102]:

	beds	baths	size	size_units	lot_size	lot_size_units	zip_code	price
0	3	3.0	2850.0	sqft	4200.00	sqft	98119	1175000.0
1	4	5.0	3040.0	sqft	5002.00	sqft	98106	1057500.0
2	3	1.0	1290.0	sqft	6048.00	sqft	98125	799000.0
3	3	2.0	2360.0	sqft	0.28	acre	98188	565000.0
4	3	3.5	1942.0	sqft	1603.00	sqft	98107	1187000.0
...	...	...	...	...	...	...	...	...
500	5	4.5	5580.0	sqft	0.30	acre	98146	3800000.0
501	3	2.5	1390.0	sqft	1570.00	sqft	98126	575000.0
502	3	2.5	2950.0	sqft	0.47	acre	98118	3105000.0
503	5	5.0	3010.0	sqft	4887.00	sqft	98115	1807000.0
504	3	2.0	1301.0	sqft	3000.00	sqft	98103	895000.0

505 rows × 8 columns

find the start 5 rows (it takes automatically 5 rows)

In [103…

```python
data.head()
```

Out[103]:

	beds	baths	size	size_units	lot_size	lot_size_units	zip_code	price
0	3	3.0	2850.0	sqft	4200.00	sqft	98119	1175000.0
1	4	5.0	3040.0	sqft	5002.00	sqft	98106	1057500.0
2	3	1.0	1290.0	sqft	6048.00	sqft	98125	799000.0
3	3	2.0	2360.0	sqft	0.28	acre	98188	565000.0
4	3	3.5	1942.0	sqft	1603.00	sqft	98107	1187000.0

find the last 5 rows (# it takes automatically 5 rows)

In [104…

```python
data.tail()
```

Out[104]:

	beds	baths	size	size_units	lot_size	lot_size_units	zip_code	price
500	5	4.5	5580.0	sqft	0.30	acre	98146	3800000.0
501	3	2.5	1390.0	sqft	1570.00	sqft	98126	575000.0
502	3	2.5	2950.0	sqft	0.47	acre	98118	3105000.0
503	5	5.0	3010.0	sqft	4887.00	sqft	98115	1807000.0
504	3	2.0	1301.0	sqft	3000.00	sqft	98103	895000.0

find the how many rows and columns in dataset

In [105…

```python
data.shape
```

Out[105]:

```
(505, 8)
```

get information total number in rows and total number in columns and datatypes each rows and columns memory requirements

In [106…

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 505 entries, 0 to 504
Data columns (total 8 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   beds            505 non-null    int64  
 1   baths           505 non-null    float64
 2   size            505 non-null    float64
 3   size_units      505 non-null    object 
 4   lot_size        428 non-null    float64
 5   lot_size_units  428 non-null    object 
 6   zip_code        505 non-null    int64  
 7   price           505 non-null    float64
dtypes: float64(4), int64(2), object(2)
memory usage: 31.7+ KB
```

check null values in dataset

In [107…

```python
data.isnull()
```

Out[107]:

	beds	baths	size	size_units	lot_size	lot_size_units	zip_code	price
0	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...
500	False	False	False	False	False	False	False	False
501	False	False	False	False	False	False	False	False
502	False	False	False	False	False	False	False	False
503	False	False	False	False	False	False	False	False
504	False	False	False	False	False	False	False	False

505 rows × 8 columns

In [108…

```python
data.isnull().sum()
```

Out[108]:

```
beds               0
baths              0
size               0
size_units         0
lot_size          77
lot_size_units    77
zip_code           0
price              0
dtype: int64
```

In [109…

```python
for column in data.columns:
    print(data[column].value_counts())
    print("*" * 20)
```

```
3    173
2    126
4    108
1     55
5     33
7      6
6      3
9      1
Name: beds, dtype: int64
********************
2.0    140
1.0    110
2.5     96
3.0     52
1.5     40
3.5     36
4.0     11
5.0      8
4.5      5
6.0      3
5.5      2
7.0      1
6.5      1
Name: baths, dtype: int64
********************
2480.0    4
1240.0    4
1540.0    4
1200.0    4
1800.0    4
         ..
921.0     1
1892.0    1
2230.0    1
2867.0    1
1301.0    1
Name: size, Length: 375, dtype: int64
********************
sqft    505
Name: size_units, dtype: int64
********************
5000.00    14
4000.00    14
6000.00     8
1.00        6
0.25        5
           ..
6656.00     1
8057.00     1
6755.00     1
9130.00     1
4887.00     1
Name: lot_size, Length: 314, dtype: int64
********************
sqft    369
acre     59
Name: lot_size_units, dtype: int64
********************
98103    48
98115    39
98117    38
98125    29
98122    29
98199    25
98126    22
98144    21
98118    20
98119    19
98102    19
98136    17
98107    17
98178    16
98106    15
98146    15
98105    15
98121    14
98109    13
98108    13
98116    12
98112    12
98133    12
98177     8
98168     7
98101     5
98104     3
98188     1
98164     1
Name: zip_code, dtype: int64
********************
875000.0     8
700000.0     7
750000.0     7
1025000.0    6
900000.0     6
            ..
692500.0     1
1108000.0    1
2334500.0    1
545000.0     1
1807000.0    1
Name: price, Length: 322, dtype: int64
********************
```

In [110…

```python
data.isna().sum()
```

Out[110]:

```
beds               0
baths              0
size               0
size_units         0
lot_size          77
lot_size_units    77
zip_code           0
price              0
dtype: int64
```

In [111…

```python
data.drop(columns=['lot_size', 'size_units'], inplace=True)
```

In [112…

```python
data.describe()
```

Out[112]:

	beds	baths	size	zip_code	price
count	505.000000	505.000000	505.000000	505.000000	5.050000e+02
mean	2.954455	2.219802	1851.843564	98125.366337	9.795822e+05
std	1.214947	1.013404	922.556090	24.875054	6.084759e+05
min	1.000000	1.000000	376.000000	98101.000000	1.700000e+05
25%	2.000000	1.500000	1171.000000	98108.000000	6.199900e+05
50%	3.000000	2.000000	1690.000000	98118.000000	8.400000e+05
75%	4.000000	2.500000	2400.000000	98126.000000	1.155000e+06
max	9.000000	7.000000	6139.000000	98199.000000	6.250000e+06

In [113…

```python
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 505 entries, 0 to 504
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   beds            505 non-null    int64  
 1   baths           505 non-null    float64
 2   size            505 non-null    float64
 3   lot_size_units  428 non-null    object 
 4   zip_code        505 non-null    int64  
 5   price           505 non-null    float64
dtypes: float64(3), int64(2), object(1)
memory usage: 23.8+ KB
```

In [114…

```python
data['beds'].value_counts()
```

Out[114]:

```
3    173
2    126
4    108
1     55
5     33
7      6
6      3
9      1
Name: beds, dtype: int64
```

In [115…

```python
data.head()
```

Out[115]:

	beds	baths	size	lot_size_units	zip_code	price
0	3	3.0	2850.0	sqft	98119	1175000.0
1	4	5.0	3040.0	sqft	98106	1057500.0
2	3	1.0	1290.0	sqft	98125	799000.0
3	3	2.0	2360.0	acre	98188	565000.0
4	3	3.5	1942.0	sqft	98107	1187000.0

price per sq feet

In [116…

```python
data['price_per_sqft'] = 100000 / data['size']
```

In [117…

```python
data['price_per_sqft']
```

Out[117]:

```
0      35.087719
1      32.894737
2      77.519380
3      42.372881
4      51.493306
         ...    
500    17.921147
501    71.942446
502    33.898305
503    33.222591
504    76.863951
Name: price_per_sqft, Length: 505, dtype: float64
```

In [118…

```python
data.describe()
```

Out[118]:

	beds	baths	size	zip_code	price	price_per_sqft
count	505.000000	505.000000	505.000000	505.000000	5.050000e+02	505.000000
mean	2.954455	2.219802	1851.843564	98125.366337	9.795822e+05	70.051146
std	1.214947	1.013404	922.556090	24.875054	6.084759e+05	39.123550
min	1.000000	1.000000	376.000000	98101.000000	1.700000e+05	16.289298
25%	2.000000	1.500000	1171.000000	98108.000000	6.199900e+05	41.666667
50%	3.000000	2.000000	1690.000000	98118.000000	8.400000e+05	59.171598
75%	4.000000	2.500000	2400.000000	98126.000000	1.155000e+06	85.397096
max	9.000000	7.000000	6139.000000	98199.000000	6.250000e+06	265.957447

In [119…

```python
data.shape
```

Out[119]:

```
(505, 7)
```

In [120…

```python
data
```

Out[120]:

	beds	baths	size	lot_size_units	zip_code	price	price_per_sqft
0	3	3.0	2850.0	sqft	98119	1175000.0	35.087719
1	4	5.0	3040.0	sqft	98106	1057500.0	32.894737
2	3	1.0	1290.0	sqft	98125	799000.0	77.519380
3	3	2.0	2360.0	acre	98188	565000.0	42.372881
4	3	3.5	1942.0	sqft	98107	1187000.0	51.493306
...	...	...	...	...	...	...	...
500	5	4.5	5580.0	acre	98146	3800000.0	17.921147
501	3	2.5	1390.0	sqft	98126	575000.0	71.942446
502	3	2.5	2950.0	acre	98118	3105000.0	33.898305
503	5	5.0	3010.0	sqft	98115	1807000.0	33.222591
504	3	2.0	1301.0	sqft	98103	895000.0	76.863951

505 rows × 7 columns

In [121…

```python
data.columns
```

Out[121]:

```
Index(['beds', 'baths', 'size', 'lot_size_units', 'zip_code', 'price',
       'price_per_sqft'],
      dtype='object')
```

In [122…

```python
data.drop(columns=['lot_size_units'], inplace=True)
```

In [123…

```python
data.drop(columns=['price_per_sqft'], inplace=True)
```

In [124…

```python
data.head()
```

Out[124]:

	beds	baths	size	zip_code	price
0	3	3.0	2850.0	98119	1175000.0
1	4	5.0	3040.0	98106	1057500.0
2	3	1.0	1290.0	98125	799000.0
3	3	2.0	2360.0	98188	565000.0
4	3	3.5	1942.0	98107	1187000.0

In [133…

```python
data=pd.read_csv("f:abc/final predicting house dataset.csv")
```

In [134…

```python
data
```

Out[134]:

	Unnamed: 0	beds	baths	size	zip_code	price
0	0	3	3.0	2850.0	98119	1175000.0
1	1	4	5.0	3040.0	98106	1057500.0
2	2	3	1.0	1290.0	98125	799000.0
3	3	3	2.0	2360.0	98188	565000.0
4	4	3	3.5	1942.0	98107	1187000.0
...	...	...	...	...	...	...
500	500	5	4.5	5580.0	98146	3800000.0
501	501	3	2.5	1390.0	98126	575000.0
502	502	3	2.5	2950.0	98118	3105000.0
503	503	5	5.0	3010.0	98115	1807000.0
504	504	3	2.0	1301.0	98103	895000.0

505 rows × 6 columns

In [142…

```python
x= data.drop(columns=['price'])
y= data['price']
```

In [166…

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.linear_model import LinearRegression, Lasso, Ridge
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score
from sklearn.preprocessing import StandardScaler
```

In [ ]:

```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
```

Applying linear regression

In [150…

```python
column_trans = make_column_transformer(
    (OneHotEncoder(sparse=False), ['beds']),
    remainder='passthrough'
)
```

In [154…

```python
scaler = StandardScaler()
```

In [156…

```python
lr=LinearRegression(normalize=True)
```

In [ ]:

```python
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler

#  Assuming x and y are already defined

#  Create and fit the scaler
scaler = StandardScaler()
x_scaled = scaler.fit_transform(x)

#  Create and fit the Linear Regression model
lr = LinearRegression()
lr.fit(x_scaled, y)
```

Out[ ]:

```
LinearRegression()
```