#SOURCE CODE

```
#AI-06-MLB2
#dataset - /content/Bengaluru_House_Data.csv


import pandas as pd
import numpy as np
import matplotlib.pyplot as plt


df = pd.read_csv('/content/Bengaluru_House_Data.csv')
df


df.shape


df.groupby('area_type')['area_type'].agg('count')


df = df.drop(['area_type','society','balcony','availability'],axis ='columns')


df.shape


df.head()


#data cleaning prosses
df.isnull().sum()


df2 = df.dropna()
df2.isnull().sum()


df2['size'].unique()


#we will create a new column called bhk
```

```
df2['bhk']= df2['size'].apply(lambda x:int(x.split(' ')[0]))

df2.head()

df2['bhk'].unique()

df2[df2.bhk>20]

df2.total_sqft.unique()

#to know if a given value in total sqf is float or not
def isfloat(x):
   try:
      float(x)
   except:
      return False
   return True

#~ is a nigget operation and return a dataframe back to me
df2[~df2['total_sqft'].apply(isfloat)].head(10)

#python code to take the range and return the average
def convert_sqft_tonum(x):
   token=x.split('-')
   if len(token)==2:
      return (float(token[0])+float(token[1]))/2
   try:
      return float(x)
   except:
      return None
```

```
convert_sqft_tonum('1120 - 1145')

#applying this function to our total_sqft column and create a new dataframe
df3 = df2.copy()
df3['total_sqft']=df3['total_sqft'].apply(convert_sqft_tonum)
df3.head()

df3.loc[30]#loc gives the location

df3.head()

#feature engeeniaring
#weare gonna create a price per sqft column, and this will help us do the outlier cleening
df4=df3.copy()
df4['price_per_sqft']=df4['price']*100000/df4['total_sqft']
df4.head()

#location is a catagorial feature
len(df4.location.unique())
#to handle the text data we convert into dummy column

df4.location = df4.location.apply(lambda x: x.strip())

location_stats = df4.groupby('location')['location'].agg('count').sort_values(ascending=False)
location_stats

len(location_stats[location_stats<=10])

location_stats_less_than_10 = location_stats[location_stats<=10]
location_stats_less_than_10
```

```
len(df4.location.unique())


df4.location = df4.location.apply(lambda x: 'other' if x in location_stats_less_than_10 else x)
len(df4.location.unique())


df4.head(10)


df4[df4.total_sqft/df4.bhk<300].head()#these are the outliers and weare gonna remove them


df4.shape


df5 = df4[~(df4.total_sqft/df4.bhk<300)]
df5.shape


df5.price_per_sqft.describe()


def remove_pps_outliers(df):
   df_out=pd.DataFrame()
   for key,subdf in df.groupby('location'):
      m=np.mean(subdf.price_per_sqft)
      st=np.std(subdf.price_per_sqft)
      reduced_df=subdf[(subdf.price_per_sqft>(m-st))& (subdf.price_per_sqft<(m+st))]
      df_out=pd.concat([df_out,reduced_df],ignore_index=True)
   return df_out
df6=remove_pps_outliers(df5)
df6.shape


#it is drawing a scatter plot for 2 beedroom and 3 bedreem apartment
import matplotlib.pyplot as plt
def plot_scatter_chart(df,location):
   bhk2=df[(df.location==location)&(df.bhk==2)]
```

```
    bhk3=df[(df.location==location)&(df.bhk==3)]
    plt.rcParams['figure.figsize']=(15,10)
    plt.scatter(bhk2.total_sqft,bhk2.price,color='Blue',label='2 BHK',s=50)
    plt.scatter(bhk3.total_sqft,bhk3.price,color='green',marker='+',label='3 BHK',s=50)
    plt.xlabel('Total Square Foot')
    plt.ylabel('Price')
    plt.title(location)
    plt.legend()
plot_scatter_chart(df6,"Rajaji Nagar")


def remove_bhk_outliers(df):
    exclude_indices=np.array([])
    for location, location_df in df.groupby('location'):#going through every location dataframe
        bhk_sats={}
        for bhk,bhk_df in location_df.groupby('bhk'):#creating new dataframe named as bhk
            bhk_sats[bhk]={#per bhk dataframe i m calculating mean , std and count
                'mean':np.mean(bhk_df.price_per_sqft),
                'std':np.std(bhk_df.price_per_sqft),
                'count':bhk_df.shape[0]
            }
        for bhk,bhk_df in location_df.groupby('bhk'):
            stats=bhk_sats.get(bhk-1)
            if stats and stats['count']>5:

exclude_indices=np.append(exclude_indices,bhk_df[bhk_df.price_per_sqft<(stats['mean'])].index.va
lues)
    return df.drop(exclude_indices,axis='index')


df7=remove_bhk_outliers(df6)
df7.shape


plot_scatter_chart(df7,"Rajaji Nagar")
```

```
plt.rcParams['figure.figsize']=(20,15)
plt.hist(df7.price_per_sqft,rwidth=0.6)
plt.xlabel("Price Per Square Foor")
plt.ylabel("Count")

df7.bath.unique()

df7[df7.bath>10]

plt.hist(df7.bath,rwidth=0.8)
plt.xlabel("Number of bathrooms")
plt.ylabel("count")

#anytime we have bathroom greater than the number of bedroom we can just mart it as a outlier
df7[df7.bath>df7.bhk+2]#all of these are outliers

df8 = df7[df7.bath<df7.bhk+2]
df8.shape

#we can drop size(we have bhk column for it) and price_per_sqft(used only for outlier detection)
df9 = df8.drop(['size','price_per_sqft'],axis='columns')
df9

#Machine learning cannot interprete text data so we have to convert thus into a numeric column
#so we gonna use dummies
dummies = pd.get_dummies(df9.location)#for each of the location it will create a new column
dummies.head(10)

#to avoid a dummy variable trap u should have one less dummies column so we afre dropping the lest column other
```

```
df10 = pd.concat([df9,dummies.drop('other',axis='columns')],axis='columns')
df10.head(10)

df11 = df10.drop('location',axis='columns')
df11.head(10)

df11.shape

#x =independent variables
#y = dependent variables
x=df11.drop('price',axis='columns')
x.head()

y = df11.price
y.head()

#Dividing our dataset into train and test
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test = train_test_split(x,y,random_state = 10)

print(x.shape) #150,4 - 150 rows,4 is cols
print(x_train.shape) #75% of x
print(x_test.shape)#25% of x

print(y.shape) # 150 rows and 1 column
print(y_train.shape) #75%
print(y_test.shape) #25%

"""**HYPER PARAMETER TUNING**

# RANDOM FOREST REGRESSOR
```

```
"""

import sys
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import random
from sklearn.decomposition import PCA
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import BaggingRegressor
from sklearn.model_selection import KFold
from sklearn.model_selection import RandomizedSearchCV
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import LabelEncoder
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from sklearn.tree import DecisionTreeRegressor

rfr_model=RandomForestRegressor(n_estimators=200)
rfr_model.fit(x_train,y_train)
print("Random Forest: ",rfr_model.score(x_test,y_test))

l=[i for i in range(1,101)]
```

```
kfold = KFold(n_splits=10, random_state=None)
parameter= {"max_depth": [2,7,9,11,13,15,None],
            "max_features":['auto', 'sqrt', 'log2',None],
            "max_leaf_nodes":l,
            'min_samples_leaf':l}
rfr_model1= RandomForestRegressor()
rfr_model1_tuning= RandomizedSearchCV(rfr_model1, parameter, cv = 5)

rfr_model1_tuning.fit(x_train, y_train)
print("Tuned Random forest classifier Parameters: {}".format(rfr_model1_tuning.best_params_))
print("Best score is {}".format(rfr_model1_tuning.best_score_))

"""# DECISION TREE"""

l=[i for i in range(1,101)]
kfold1 = KFold(n_splits=10, random_state=None)
parameter1= {"max_depth":l,
           "criterion":["squared_error","friedman_mse","absolute_error","poisson"],
           "splitter":['best','random'],
           "max_leaf_nodes":l,
           "min_samples_leaf":l}
dec_tree=DecisionTreeRegressor()
dec_tree_tuning= RandomizedSearchCV(dec_tree, parameter1, cv = 5)
dec_tree_tuning.fit(x_train, y_train)
print("Tuned decision tree Parameters: {}".format(dec_tree_tuning.best_params_))
print("Best score is {}".format(dec_tree_tuning.best_score_))

"""MACHINE LEARNING ALG"""

accuracy=[]
model=[]
```

```
with_pca=[]
mse=[]

x_train1, x_test1, y_train1, y_test1 = train_test_split(x,y,test_size=0.10)

"""DECISION TREE"""

decision_tree=DecisionTreeRegressor(min_samples_leaf=.01)
decision_tree.fit(x_train1,y_train1)
y_preds=decision_tree.predict(x_test1)
accuracy.append(r2_score(y_test1,y_preds)*100)
model.append('Decision Tree')
with_pca.append(0)
mse.append(mean_squared_error(y_test1,y_preds))
print("Accuracy of Decision Tree Regressor without PCA: ",r2_score(y_test1,y_preds)*100)
print("The mean squared error of Decision tree regressor without pca is: ",mean_squared_error(y_test1,y_preds))
print(" ")

df=pd.DataFrame({'y':y_test1,'y_predicted':y_preds})
df

sns.lmplot('y','y_predicted',data=df)

"""#linear regression"""

mlrm =LinearRegression()
mlrm.fit(x_train1, y_train1)
y_preds1= mlrm.predict(x_test1)
print("the accuracy of linear regression model without PCA is: ",r2_score(y_test1,y_preds1)*100)
print("The mean squared error of Linear regression without pca is: ", mean_squared_error(y_test1,y_preds1))
```

```
model.append('Linear regression')
with_pca.append(0)
mse.append(mean_squared_error(y_test1,y_preds1))
accuracy.append(r2_score(y_test1,y_preds1)*100)

df1=pd.DataFrame({'y':y_test1,'y_predicted':y_preds1})
df1

sns.lmplot('y','y_predicted',data=df1)

"""#KNN REGRESSOR """

neigh= KNeighborsRegressor(n_neighbors=3)
neigh.fit(x_train1, y_train1)
y_preds2= neigh.predict(x_test1)
print("the accuracy of this model is: ",r2_score(y_test1,y_preds2)*100)
print("The mean squared error of KNN regressor without pca is: ",
mean_squared_error(y_test1,y_preds2))
model.append('KNN regressor')
with_pca.append(0)
mse.append(mean_squared_error(y_test1,y_preds2))
accuracy.append(r2_score(y_test1,y_preds2)*100)

df2=pd.DataFrame({'y':y_test1,'y_predicted':y_preds2})
df2

sns.lmplot('y','y_predicted',data=df2)

"""#PCA +BAGGING"""

pca = PCA(n_components =4)
```

```
x = pca.fit_transform(x)


"""#DECISION TREE WITH PCA BAGGING"""


x_train2, x_test2, y_train2, y_test2 = train_test_split(x,y,test_size=0.10)


decision_tree1=DecisionTreeRegressor()
num=90
model1= BaggingRegressor(base_estimator=decision_tree1, n_estimators=num)
model1.fit(x_train2,y_train2)
y_preds3=model1.predict(x_test2)
print('Accuracy of Decision Tree Classifier is ',r2_score(y_test2,y_preds3)*100)
print("the mean squared error of the decision tree classifier with PCA and bagging is: ",mean_squared_error(y_test2,y_preds3))
mse.append(mean_squared_error(y_test2,y_preds3))
model.append('Decision Tree')
with_pca.append(1)
accuracy.append(r2_score(y_test2,y_preds3)*100)


df3=pd.DataFrame({'y':y_test2,'y_predicted':y_preds3})
df3


sns.lmplot('y','y_predicted',data=df3)


"""#LINEAR REGRESSION WITH PCA AND BAGGING"""


mlrm1=LinearRegression()
num1=90
model2= BaggingRegressor(base_estimator=mlrm1, n_estimators=num1)
model2.fit(x_train2, y_train2)
y_preds4= model2.predict(x_test2)
```

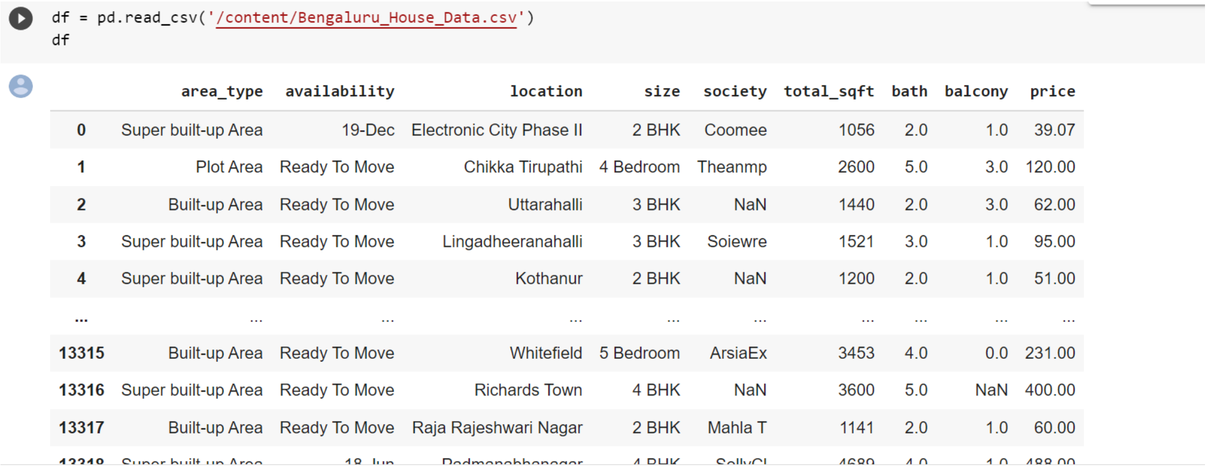
```
print('Accuracy of linear regression method is ',r2_score(y_test2,y_preds4)*100)

print("the mean squared error of the decision tree classifier with PCA and bagging is:
",mean_squared_error(y_test2,y_preds4))

mse.append(mean_squared_error(y_test2,y_preds4))

model.append('Linear regression')

with_pca.append(1)

accuracy.append(r2_score(y_test2,y_preds4)*100)


df4=pd.DataFrame({'y':y_test2,'y_predicted':y_preds4})

df4


sns.lmplot('y','y_predicted',data=df4)


"""#KNN WITH PCA AND BAGGING"""


neigh1= KNeighborsRegressor(n_neighbors=3)

num2=90

model3= BaggingRegressor(base_estimator=neigh1, n_estimators=num2)

model3.fit(x_train2, y_train2)

y_preds5= model3.predict(x_test2)

print("the accuracy of this model is: ",r2_score(y_test2,y_preds5)*100)

print("The mean squared error of KNN regressor without pca is: ",
mean_squared_error(y_test2,y_preds5))

model.append('KNN regressor')

with_pca.append(1)

mse.append(mean_squared_error(y_test2,y_preds5))

accuracy.append(r2_score(y_test2,y_preds5)*100)


df5=pd.DataFrame({'y':y_test2,'y_predicted':y_preds5})

df5


sns.lmplot('y','y_predicted',data=df5)
```

```
df6=pd.DataFrame()

df6['model']=model

df6['with_pca']=with_pca

df6['accuracy']=accuracy

df6['mean_squared_error']=mse

df6


sns.barplot(x=df6['model'],y=df6['accuracy'],hue=df6['with_pca'],palette='Greys')


"""***LINEAR REGRESSION HAS THE HIGHEST ACCURACY WITH PCA, DECISSION TREE HAS THE
HIGHEST ACCURACY WITHOUT PCA***"""
```

#P.T.O FOR OUTPUT

#OUTPUT

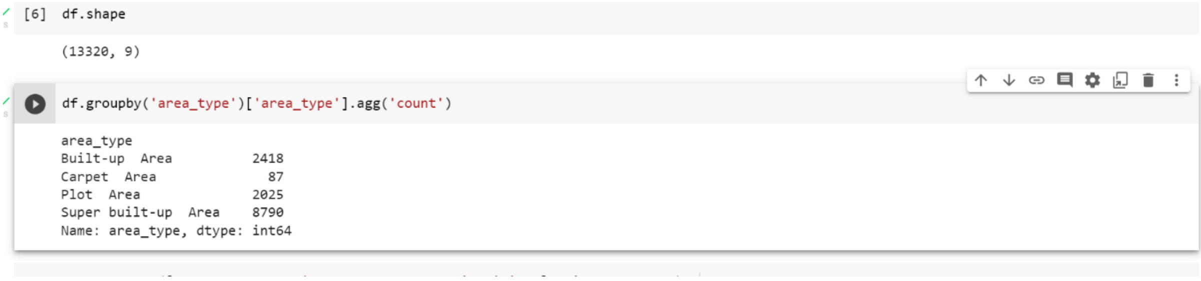
1)



```
df = pd.read_csv('/content/Bengaluru_House_Data.csv')
df
```

	area_type	availability	location	size	society	total_sqft	bath	balcony	price
0	Super built-up Area	19-Dec	Electronic City Phase II	2 BHK	Coomee	1056	2.0	1.0	39.07
1	Plot Area	Ready To Move	Chikka Tirupathi	4 Bedroom	Theanmp	2600	5.0	3.0	120.00
2	Built-up Area	Ready To Move	Uttarahalli	3 BHK	NaN	1440	2.0	3.0	62.00
3	Super built-up Area	Ready To Move	Lingadheeranahalli	3 BHK	Soiewre	1521	3.0	1.0	95.00
4	Super built-up Area	Ready To Move	Kothanur	2 BHK	NaN	1200	2.0	1.0	51.00
...	...	...	...	...	...	...	...	...	...
13315	Built-up Area	Ready To Move	Whitefield	5 Bedroom	ArsiaEx	3453	4.0	0.0	231.00
13316	Super built-up Area	Ready To Move	Richards Town	4 BHK	NaN	3600	5.0	NaN	400.00
13317	Built-up Area	Ready To Move	Raja Rajeshwari Nagar	2 BHK	Mahla T	1141	2.0	1.0	60.00
13318	Super built-up Area	18-Jun	Padmanabhanagar	4 BHK	SollyCl	4689	4.0	1.0	488.00

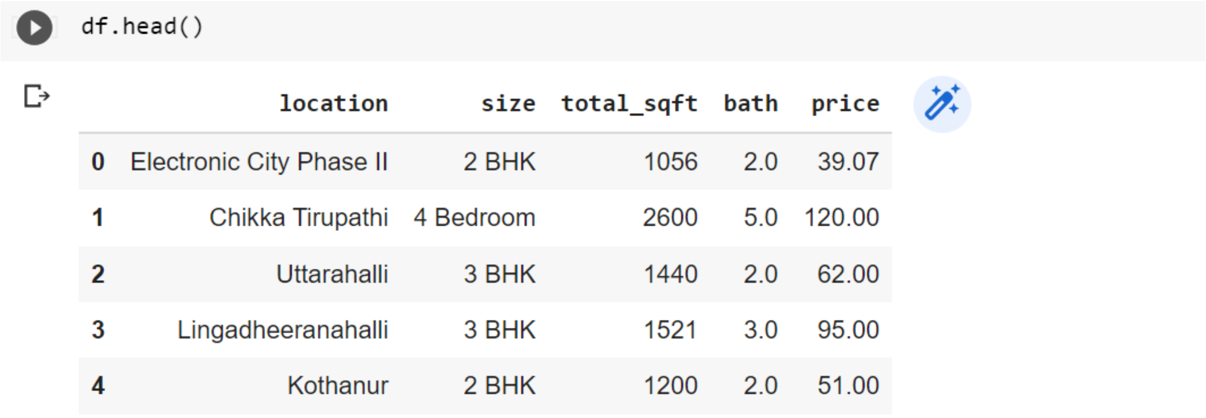
2)



```
[6] df.shape

(13320, 9)
```

```
df.groupby('area_type')['area_type'].agg('count')

area_type
Built-up  Area          2418
Carpet  Area              87
Plot  Area              2025
Super built-up  Area    8790
Name: area_type, dtype: int64
```

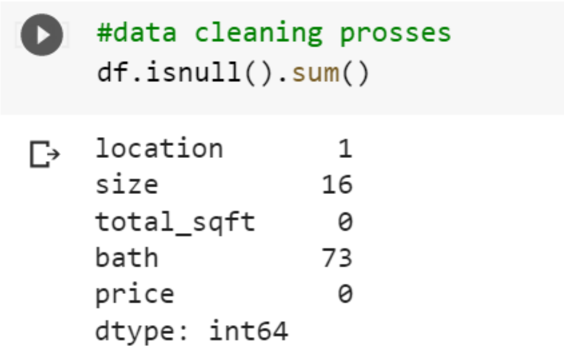
3)



```
df.head()
```

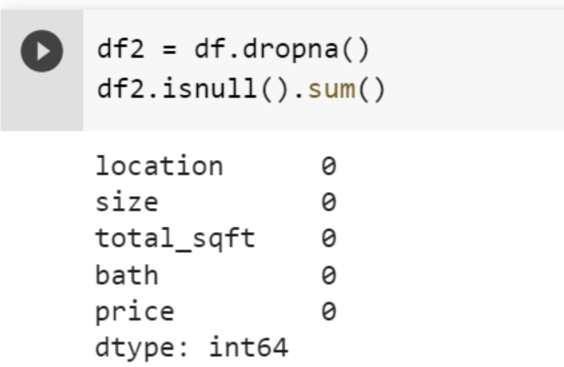
	location	size	total_sqft	bath	price
0	Electronic City Phase II	2 BHK	1056	2.0	39.07
1	Chikka Tirupathi	4 Bedroom	2600	5.0	120.00
2	Uttarahalli	3 BHK	1440	2.0	62.00
3	Lingadheeranahalli	3 BHK	1521	3.0	95.00
4	Kothanur	2 BHK	1200	2.0	51.00

4)



```
#data cleaning prosses
df.isnull().sum()
```

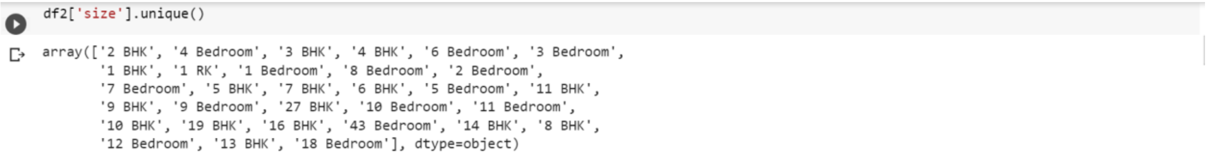
```
location       1
size          16
total_sqft     0
bath          73
price          0
dtype: int64
```



```
df2 = df.dropna()
df2.isnull().sum()
```

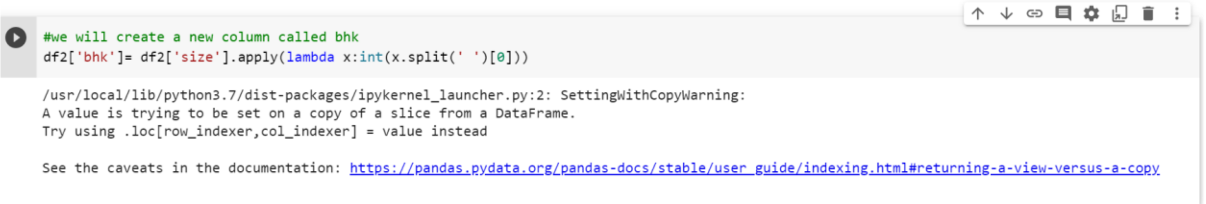
```
location      0
size          0
total_sqft    0
bath          0
price         0
dtype: int64
```

5)



```
df2['size'].unique()
```

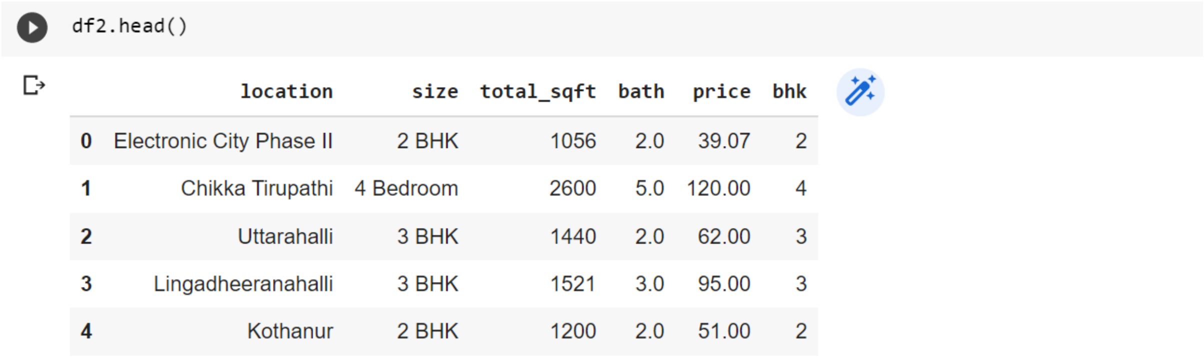
```
array(['2 BHK', '4 Bedroom', '3 BHK', '4 BHK', '6 Bedroom', '3 Bedroom',
       '1 BHK', '1 RK', '1 Bedroom', '8 Bedroom', '2 Bedroom',
       '7 Bedroom', '5 BHK', '7 BHK', '6 BHK', '5 Bedroom', '11 BHK',
       '9 BHK', '9 Bedroom', '27 BHK', '10 Bedroom', '11 Bedroom',
       '10 BHK', '19 BHK', '16 BHK', '43 Bedroom', '14 BHK', '8 BHK',
       '12 Bedroom', '13 BHK', '18 Bedroom'], dtype=object)
```



```
#we will create a new column called bhk
df2['bhk']= df2['size'].apply(lambda x:int(x.split(' ')[0]))
```

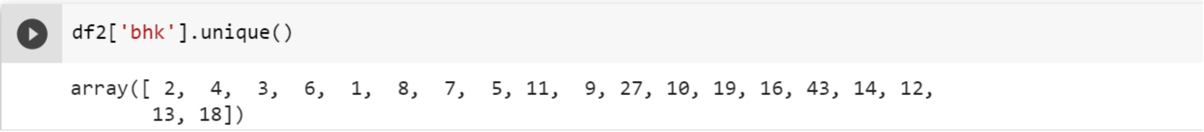
```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:2: SettingWithCopyWarning: 
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

6)



```
df2.head()
```

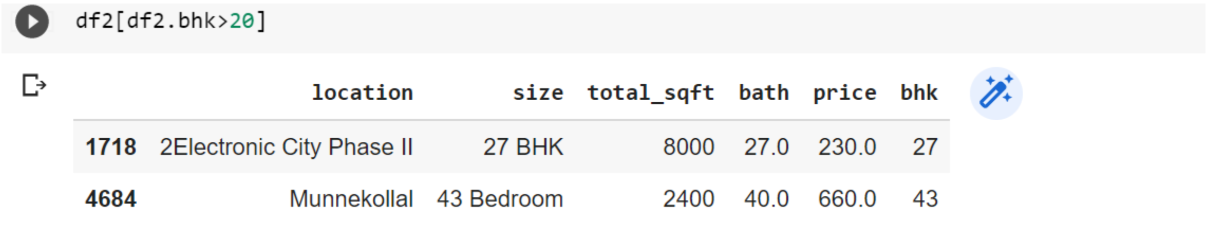
	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600	5.0	120.00	4
2	Uttarahalli	3 BHK	1440	2.0	62.00	3
3	Lingadheeranahalli	3 BHK	1521	3.0	95.00	3
4	Kothanur	2 BHK	1200	2.0	51.00	2



```
df2['bhk'].unique()
```

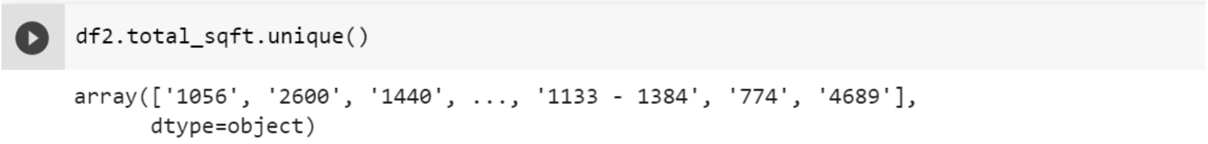
```
array([ 2,  4,  3,  6,  1,  8,  7,  5, 11,  9, 27, 10, 19, 16, 43, 14, 12,
       13, 18])
```

7)



```
df2[df2.bhk>20]
```

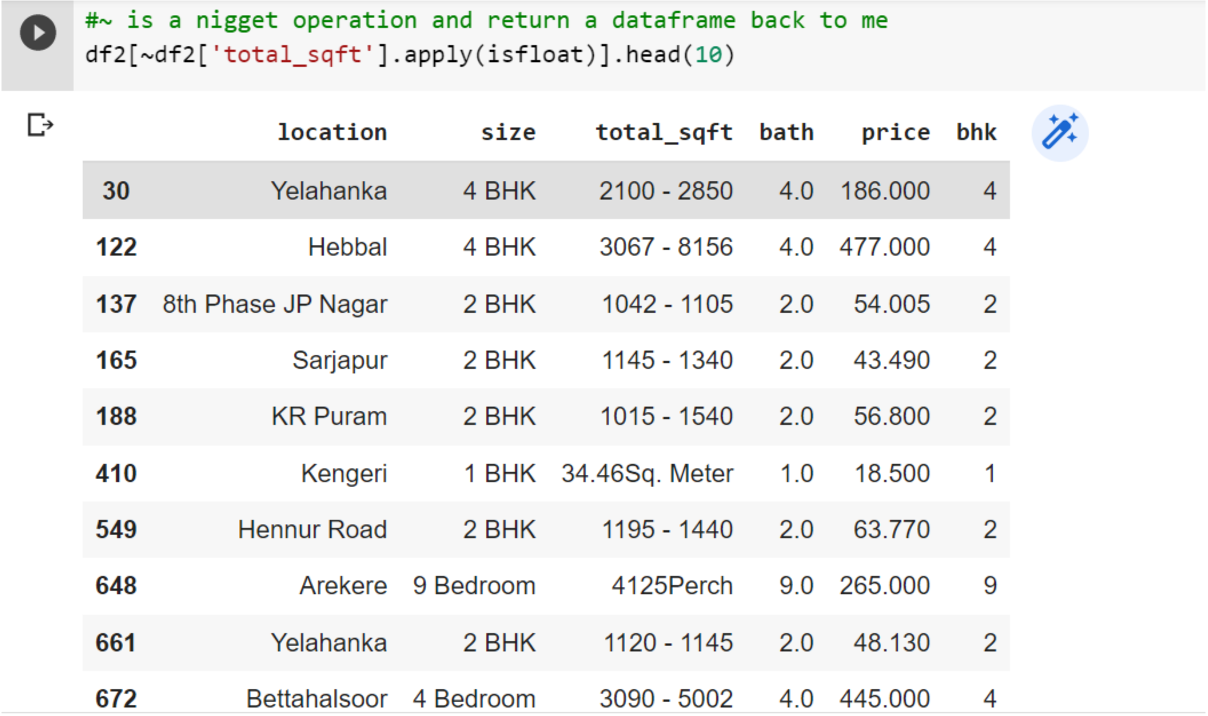
	location	size	total_sqft	bath	price	bhk
1718	2Electronic City Phase II	27 BHK	8000	27.0	230.0	27
4684	Munnekollal	43 Bedroom	2400	40.0	660.0	43



```
df2.total_sqft.unique()
```

```
array(['1056', '2600', '1440', ..., '1133 - 1384', '774', '4689'],
      dtype=object)
```

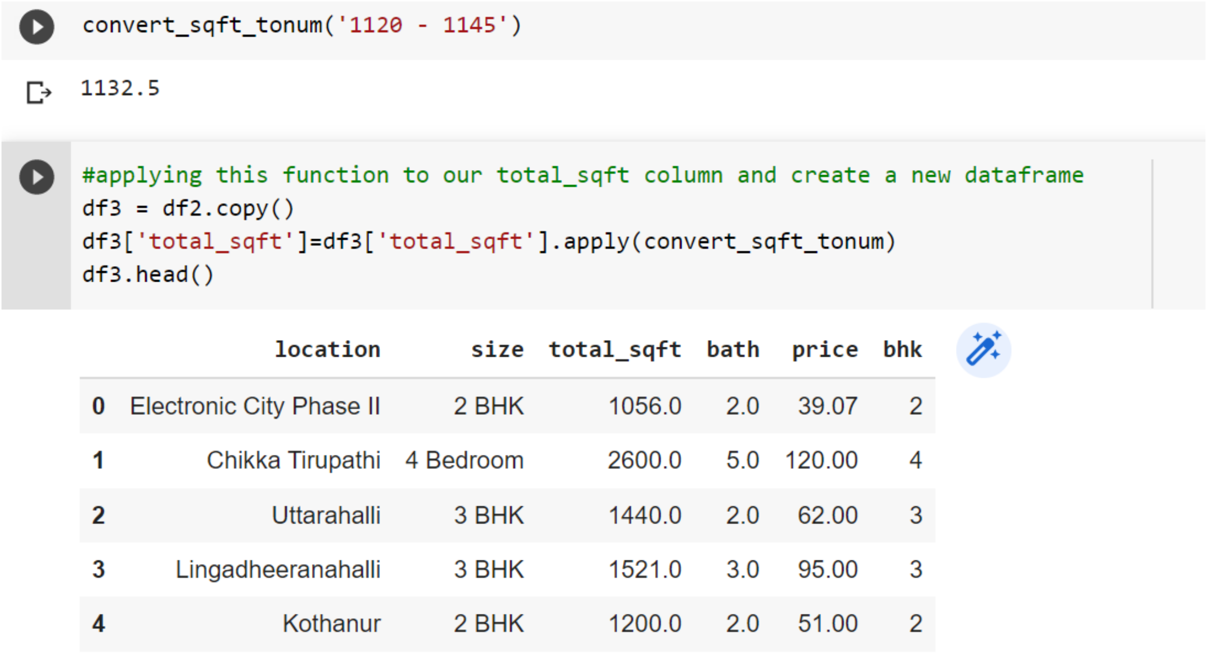
8)



```
#~ is a nigget operation and return a dataframe back to me
df2[~df2['total_sqft'].apply(isfloat)].head(10)
```

	location	size	total_sqft	bath	price	bhk
30	Yelahanka	4 BHK	2100 - 2850	4.0	186.000	4
122	Hebbal	4 BHK	3067 - 8156	4.0	477.000	4
137	8th Phase JP Nagar	2 BHK	1042 - 1105	2.0	54.005	2
165	Sarjapur	2 BHK	1145 - 1340	2.0	43.490	2
188	KR Puram	2 BHK	1015 - 1540	2.0	56.800	2
410	Kengeri	1 BHK	34.46Sq. Meter	1.0	18.500	1
549	Hennur Road	2 BHK	1195 - 1440	2.0	63.770	2
648	Arekere	9 Bedroom	4125Perch	9.0	265.000	9
661	Yelahanka	2 BHK	1120 - 1145	2.0	48.130	2
672	Bettahalsoor	4 Bedroom	3090 - 5002	4.0	445.000	4

9)



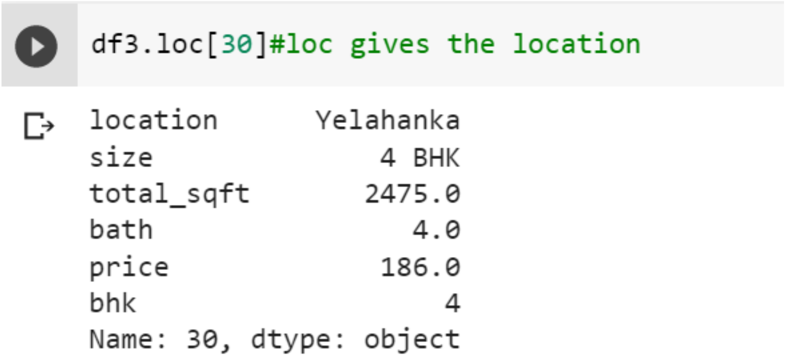
```
convert_sqft_tonum('1120 - 1145')
```

```
1132.5
```

```
#applying this function to our total_sqft column and create a new dataframe
df3 = df2.copy()
df3['total_sqft']=df3['total_sqft'].apply(convert_sqft_tonum)
df3.head()
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3
4	Kothanur	2 BHK	1200.0	2.0	51.00	2

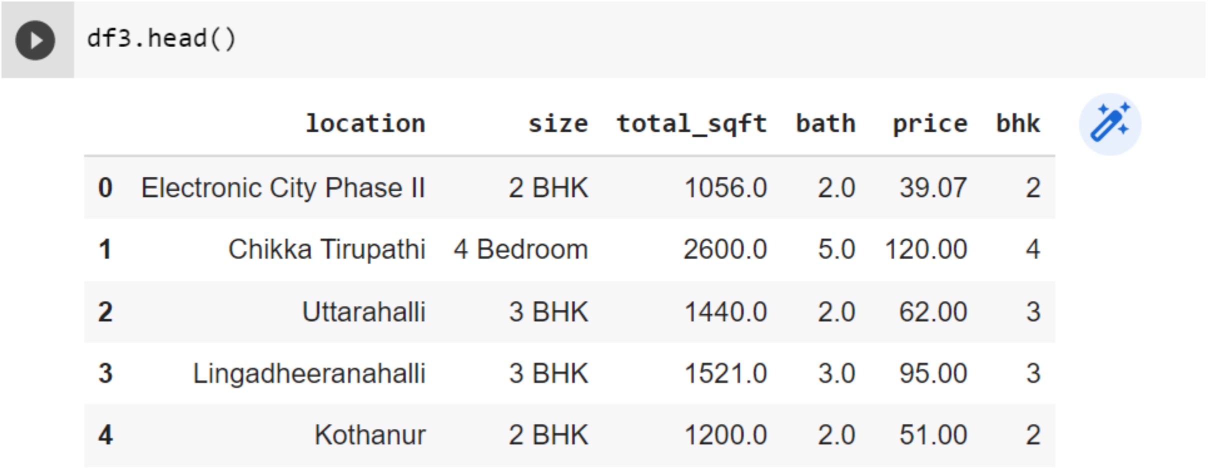
10)



```
df3.loc[30]#loc gives the location
```

```
location        Yelahanka
size                4 BHK
total_sqft         2475.0
bath                  4.0
price               186.0
bhk                     4
Name: 30, dtype: object
```

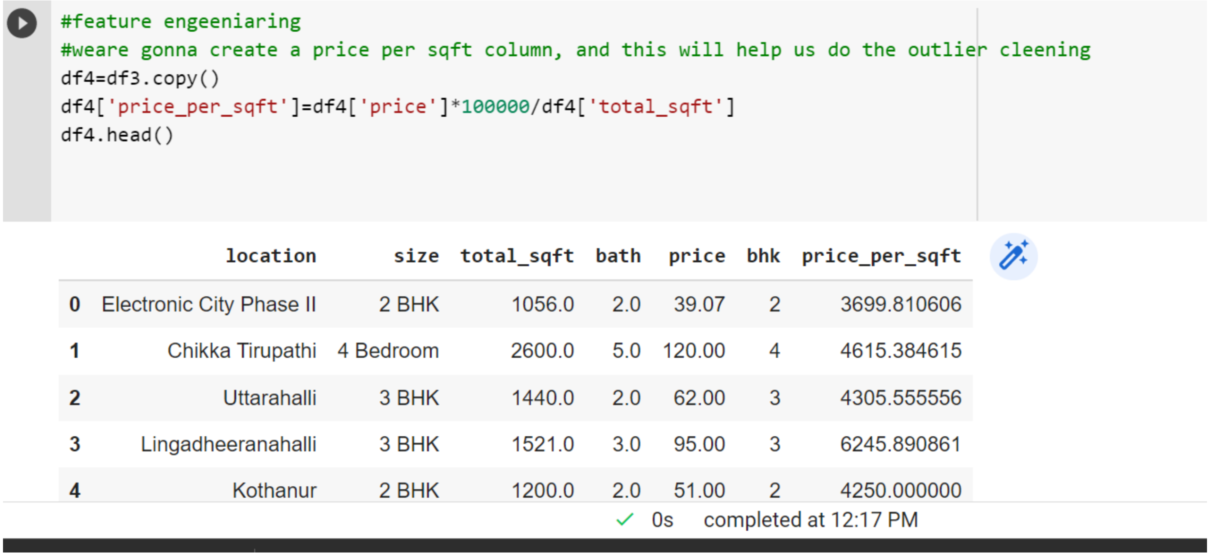
11)



```
df3.head()
```

	location	size	total_sqft	bath	price	bhk
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3
4	Kothanur	2 BHK	1200.0	2.0	51.00	2

12)

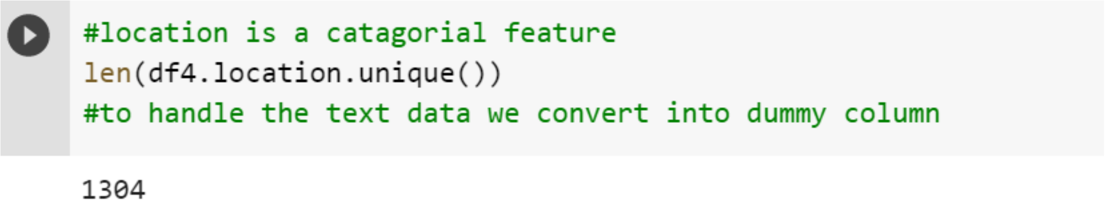


```
#feature engeeniaring
#weare gonna create a price per sqft column, and this will help us do the outlier cleening
df4=df3.copy()
df4['price_per_sqft']=df4['price']*100000/df4['total_sqft']
df4.head()
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	3699.810606
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4	4615.384615
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	4305.555556
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3	6245.890861
4	Kothanur	2 BHK	1200.0	2.0	51.00	2	4250.000000

✓ 0s completed at 12:17 PM

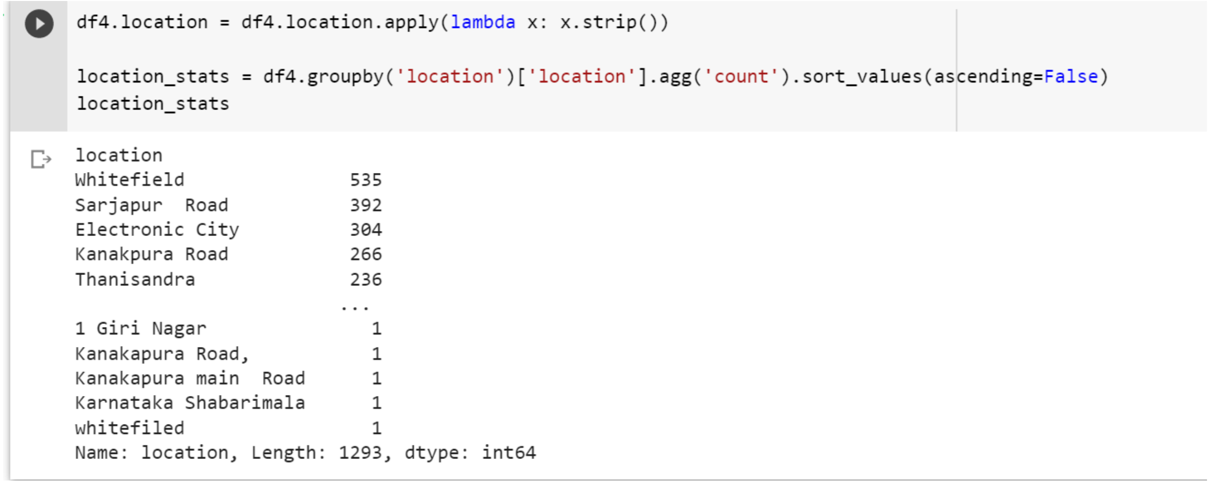
13)



```
#location is a catagorial feature
len(df4.location.unique())
#to handle the text data we convert into dummy column
```

```
1304
```

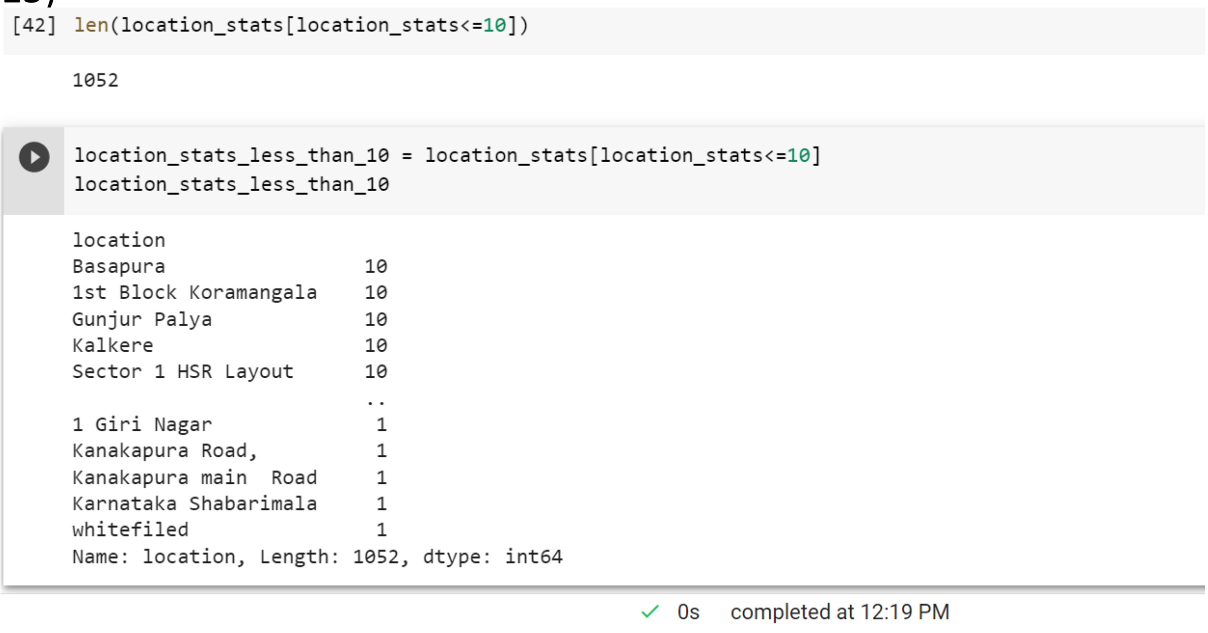
14)



```
df4.location = df4.location.apply(lambda x: x.strip())

location_stats = df4.groupby('location')['location'].agg('count').sort_values(ascending=False)
location_stats
```

```
location
Whitefield                535
Sarjapur  Road            392
Electronic City           304
Kanakpura Road            266
Thanisandra               236
                         ...
1 Giri Nagar                1
Kanakapura Road,            1
Kanakapura main  Road       1
Karnataka Shabarimala       1
whitefiled                  1
Name: location, Length: 1293, dtype: int64
```

15)

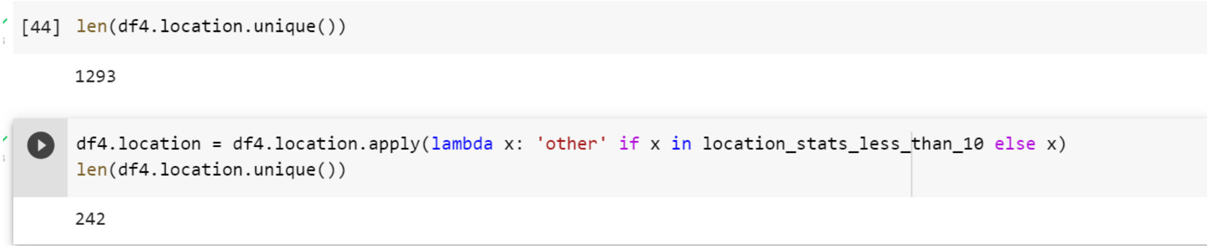


```
[42] len(location_stats[location_stats<=10])

     1052
```

```
location_stats_less_than_10 = location_stats[location_stats<=10]
location_stats_less_than_10

location
Basapura                 10
1st Block Koramangala    10
Gunjur Palya             10
Kalkere                  10
Sector 1 HSR Layout      10
                         ..
1 Giri Nagar              1
Kanakapura Road,          1
Kanakapura main  Road     1
Karnataka Shabarimala     1
whitefiled                1
Name: location, Length: 1052, dtype: int64
```

✓ 0s completed at 12:19 PM

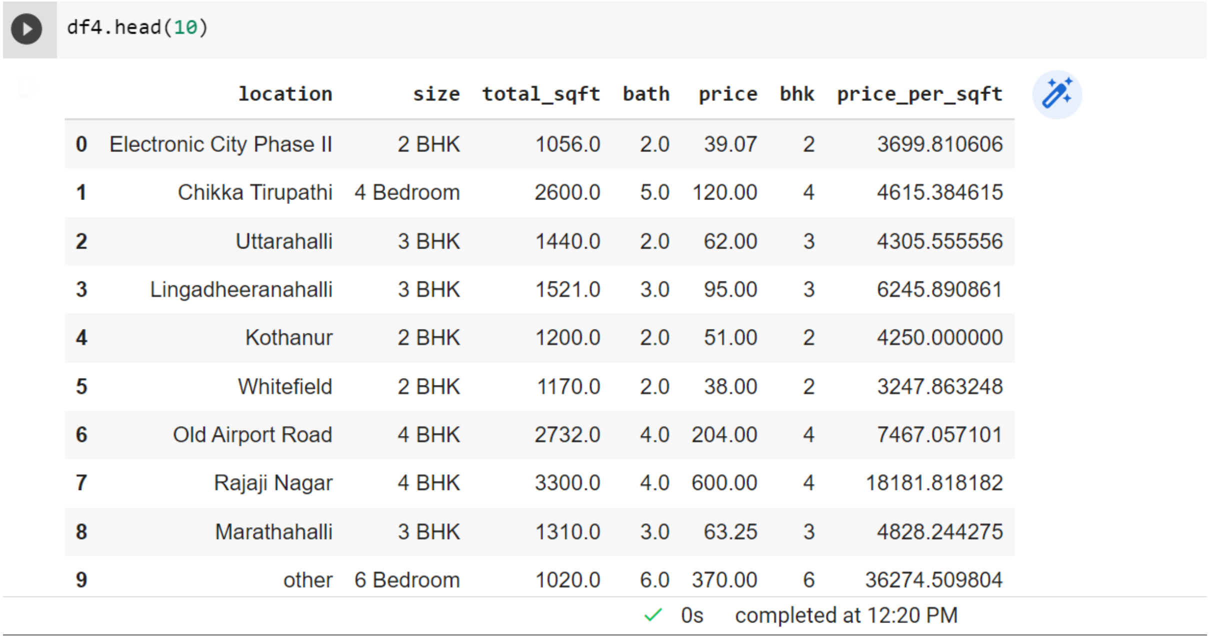
16)



```
[44] len(df4.location.unique())

     1293
```

```
df4.location = df4.location.apply(lambda x: 'other' if x in location_stats_less_than_10 else x)
len(df4.location.unique())

242
```

17)

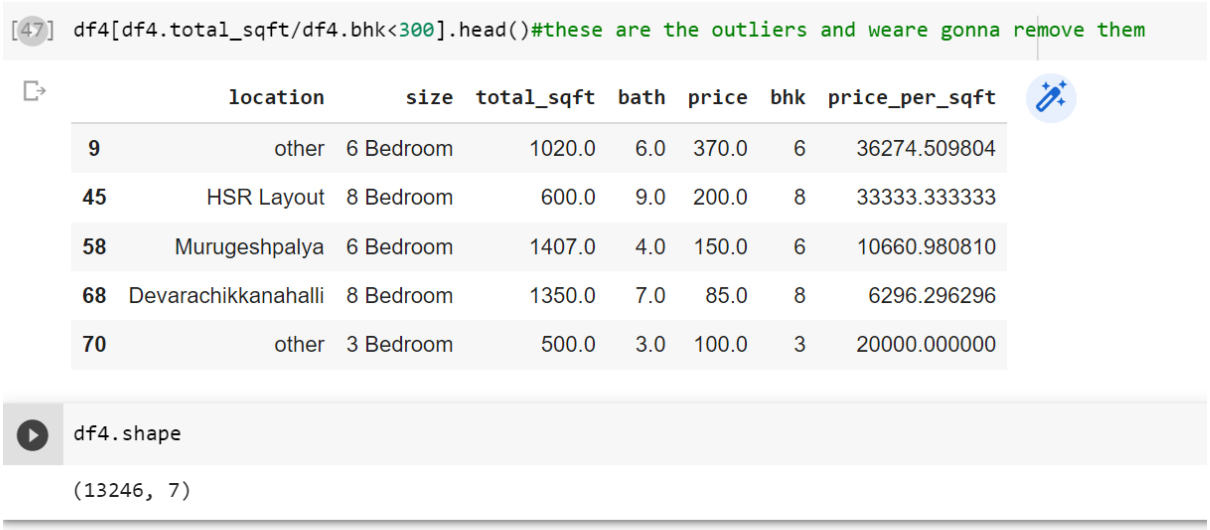


```
df4.head(10)
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
0	Electronic City Phase II	2 BHK	1056.0	2.0	39.07	2	3699.810606
1	Chikka Tirupathi	4 Bedroom	2600.0	5.0	120.00	4	4615.384615
2	Uttarahalli	3 BHK	1440.0	2.0	62.00	3	4305.555556
3	Lingadheeranahalli	3 BHK	1521.0	3.0	95.00	3	6245.890861
4	Kothanur	2 BHK	1200.0	2.0	51.00	2	4250.000000
5	Whitefield	2 BHK	1170.0	2.0	38.00	2	3247.863248
6	Old Airport Road	4 BHK	2732.0	4.0	204.00	4	7467.057101
7	Rajaji Nagar	4 BHK	3300.0	4.0	600.00	4	18181.818182
8	Marathahalli	3 BHK	1310.0	3.0	63.25	3	4828.244275
9	other	6 Bedroom	1020.0	6.0	370.00	6	36274.509804

✓ 0s completed at 12:20 PM

18)



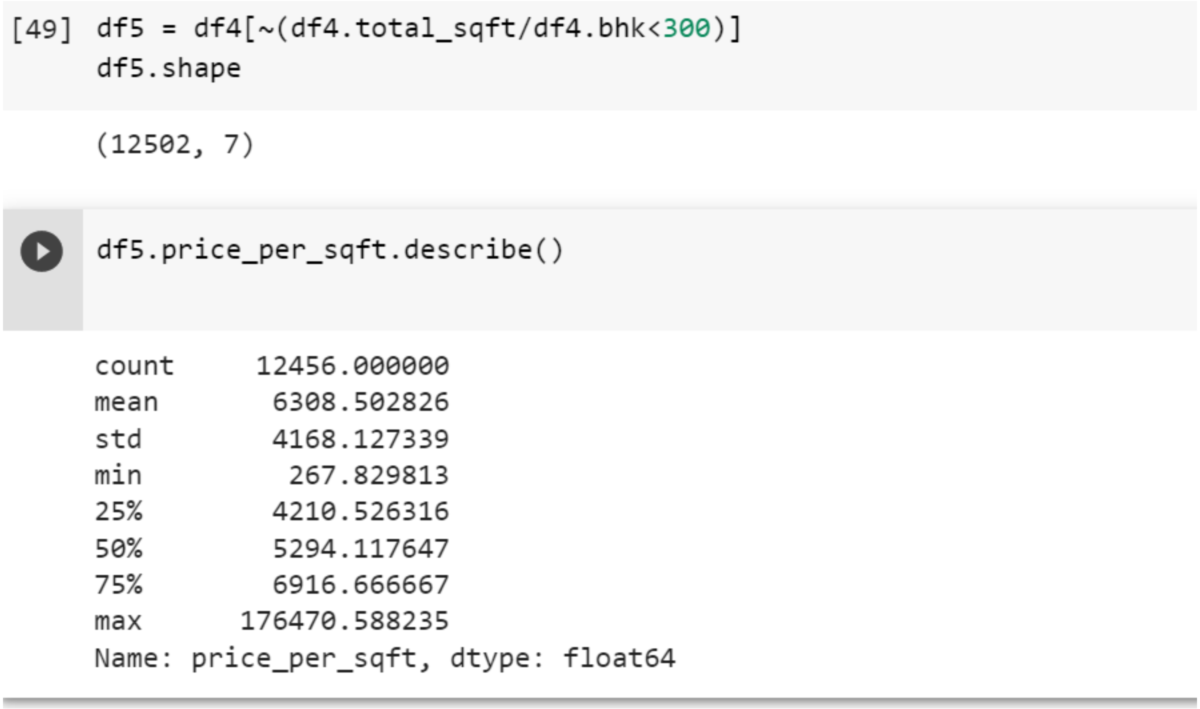
```
[47] df4[df4.total_sqft/df4.bhk<300].head()#these are the outliers and weare gonna remove them
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
9	other	6 Bedroom	1020.0	6.0	370.0	6	36274.509804
45	HSR Layout	8 Bedroom	600.0	9.0	200.0	8	33333.333333
58	Murugeshpalya	6 Bedroom	1407.0	4.0	150.0	6	10660.980810
68	Devarachikkanahalli	8 Bedroom	1350.0	7.0	85.0	8	6296.296296
70	other	3 Bedroom	500.0	3.0	100.0	3	20000.000000

```
df4.shape
```

```
(13246, 7)
```

19)



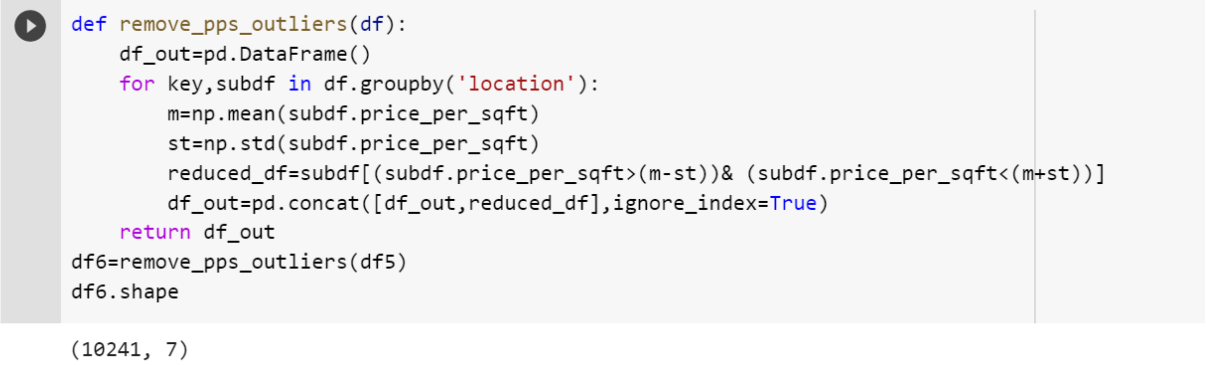
```
[49] df5 = df4[~(df4.total_sqft/df4.bhk<300)]
     df5.shape
```

```
(12502, 7)
```

```
df5.price_per_sqft.describe()
```

```
count     12456.000000
mean       6308.502826
std        4168.127339
min         267.829813
25%        4210.526316
50%        5294.117647
75%        6916.666667
max      176470.588235
Name: price_per_sqft, dtype: float64
```

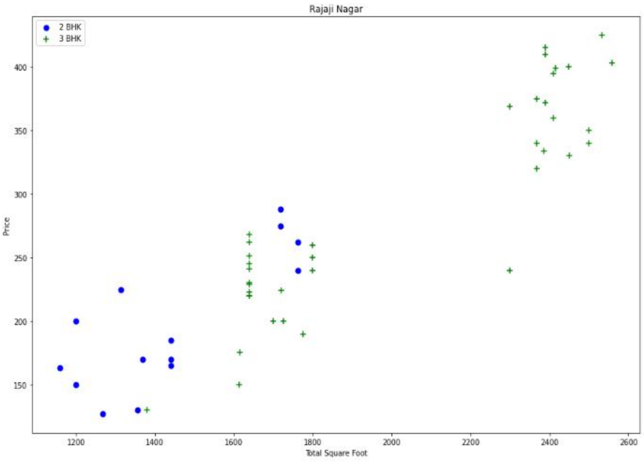
20)

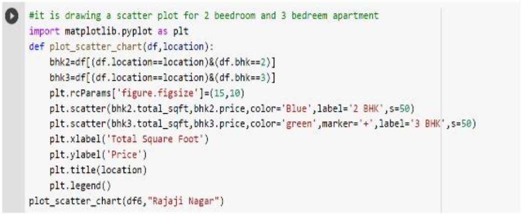


```
def remove_pps_outliers(df):
    df_out=pd.DataFrame()
    for key,subdf in df.groupby('location'):
        m=np.mean(subdf.price_per_sqft)
        st=np.std(subdf.price_per_sqft)
        reduced_df=subdf[(subdf.price_per_sqft>(m-st))& (subdf.price_per_sqft<(m+st))]
        df_out=pd.concat([df_out,reduced_df],ignore_index=True)
    return df_out
df6=remove_pps_outliers(df5)
df6.shape
```

```
(10241, 7)
```

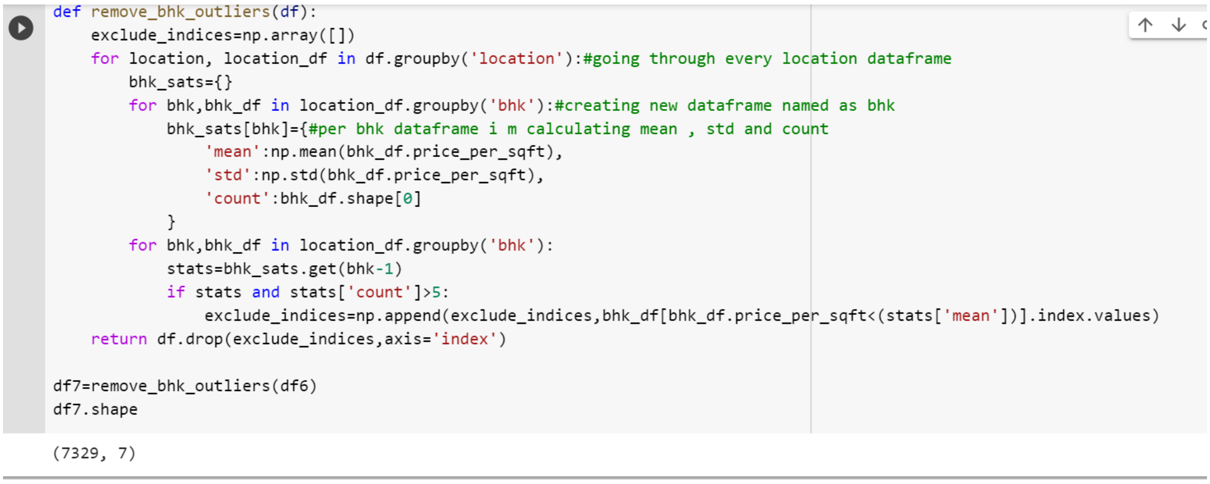
21)





```
#it is drawing a scatter plot for 2 beedroom and 3 bedreem apartment
import matplotlib.pyplot as plt
def plot_scatter_chart(df,location):
    bhk2=df[(df.location==location)&(df.bhk==2)]
    bhk3=df[(df.location==location)&(df.bhk==3)]
    plt.rcParams['figure.figsize']=(15,10)
    plt.scatter(bhk2.total_sqft,bhk2.price,color='Blue',label='2 BHK',s=50)
    plt.scatter(bhk3.total_sqft,bhk3.price,color='green',marker='+',label='3 BHK',s=50)
    plt.xlabel('Total Square Foot')
    plt.ylabel('Price')
    plt.title(location)
    plt.legend()
plot_scatter_chart(df6,"Rajaji Nagar")
```

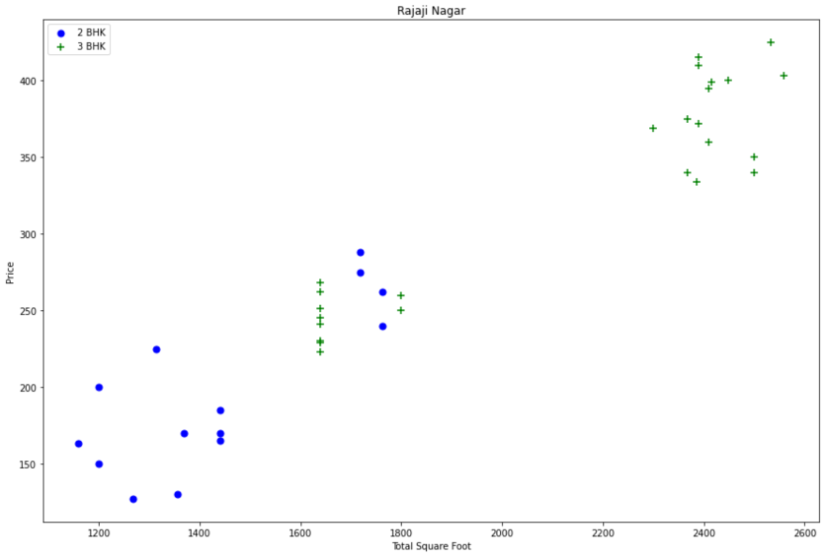
22)

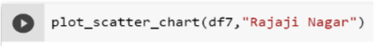


```
def remove_bhk_outliers(df):
    exclude_indices=np.array([])
    for location, location_df in df.groupby('location'):#going through every location dataframe
        bhk_sats={}
        for bhk,bhk_df in location_df.groupby('bhk'):#creating new dataframe named as bhk
            bhk_sats[bhk]={#per bhk dataframe i m calculating mean , std and count
                'mean':np.mean(bhk_df.price_per_sqft),
                'std':np.std(bhk_df.price_per_sqft),
                'count':bhk_df.shape[0]
            }
        for bhk,bhk_df in location_df.groupby('bhk'):
            stats=bhk_sats.get(bhk-1)
            if stats and stats['count']>5:
                exclude_indices=np.append(exclude_indices,bhk_df[bhk_df.price_per_sqft<(stats['mean'])].index.values)
    return df.drop(exclude_indices,axis='index')

df7=remove_bhk_outliers(df6)
df7.shape
```

```
(7329, 7)
```

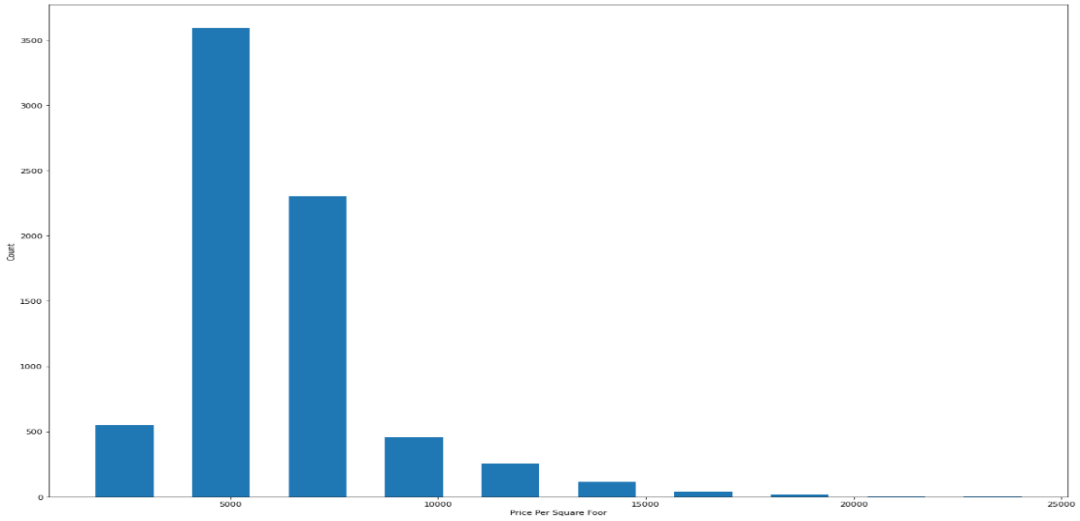
23)

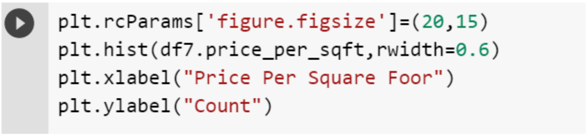




```
plot_scatter_chart(df7,"Rajaji Nagar")
```

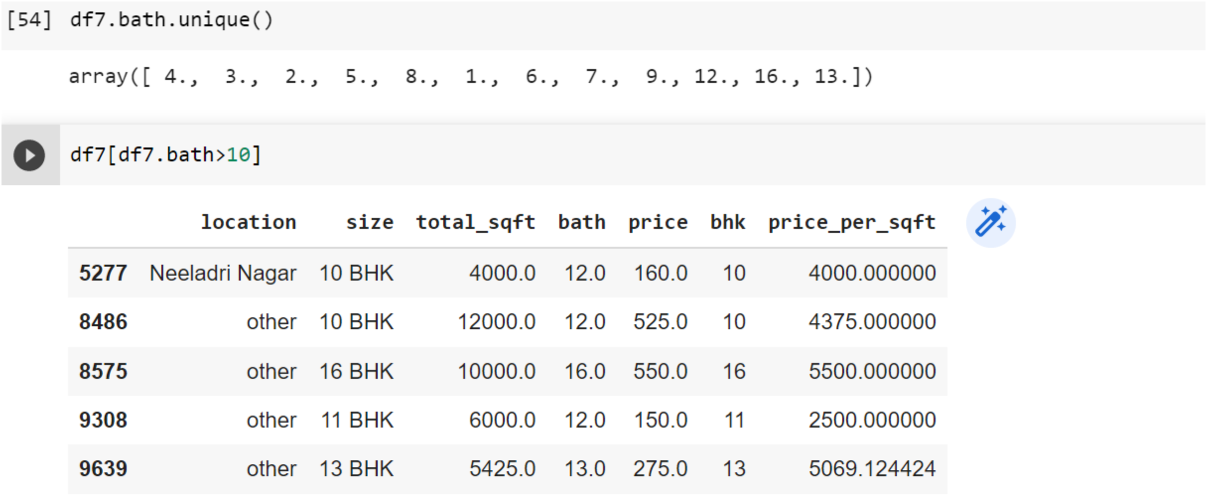
24)





```
plt.rcParams['figure.figsize']=(20,15)
plt.hist(df7.price_per_sqft,rwidth=0.6)
plt.xlabel("Price Per Square Foor")
plt.ylabel("Count")
```

25)



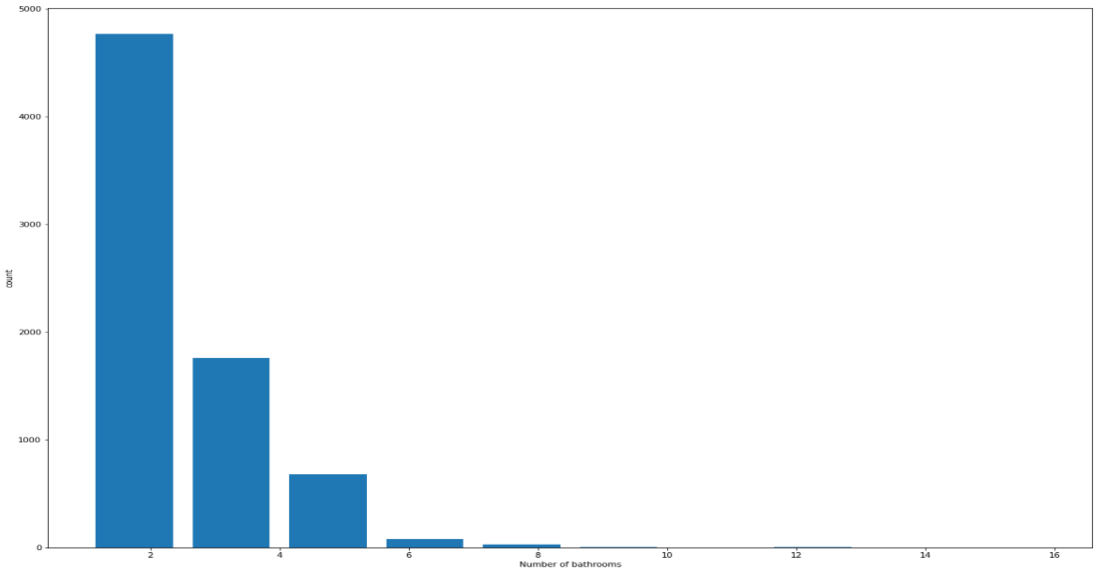
```
[54] df7.bath.unique()
```

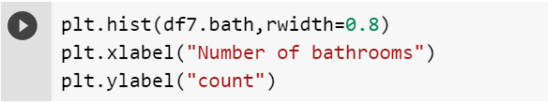
```
array([ 4.,  3.,  2.,  5.,  8.,  1.,  6.,  7.,  9., 12., 16., 13.])
```

```
df7[df7.bath>10]
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
5277	Neeladri Nagar	10 BHK	4000.0	12.0	160.0	10	4000.000000
8486	other	10 BHK	12000.0	12.0	525.0	10	4375.000000
8575	other	16 BHK	10000.0	16.0	550.0	16	5500.000000
9308	other	11 BHK	6000.0	12.0	150.0	11	2500.000000
9639	other	13 BHK	5425.0	13.0	275.0	13	5069.124424

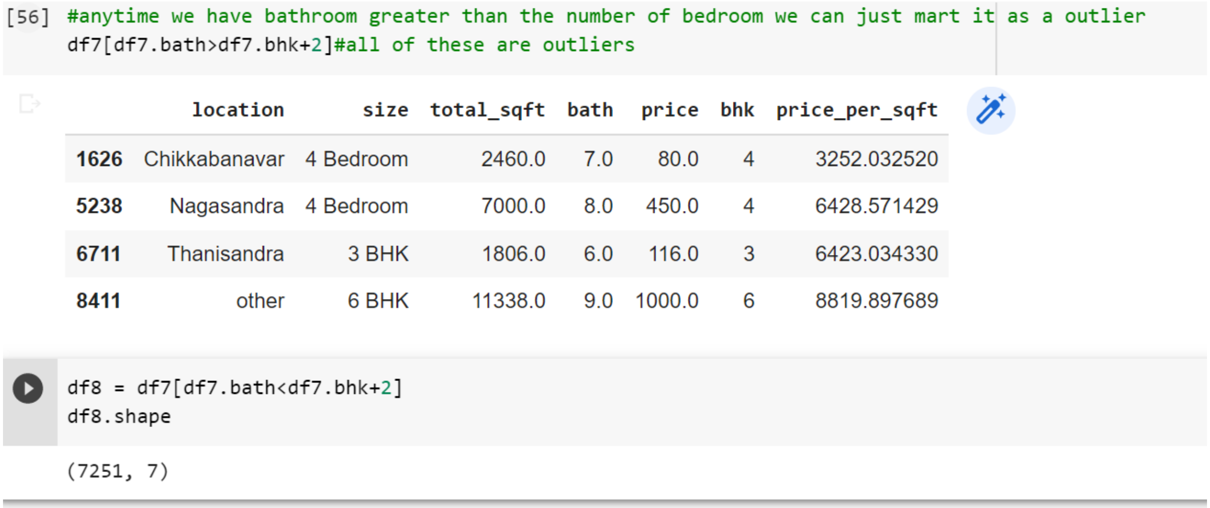
26)





```
plt.hist(df7.bath,rwidth=0.8)
plt.xlabel("Number of bathrooms")
plt.ylabel("count")
```

27)



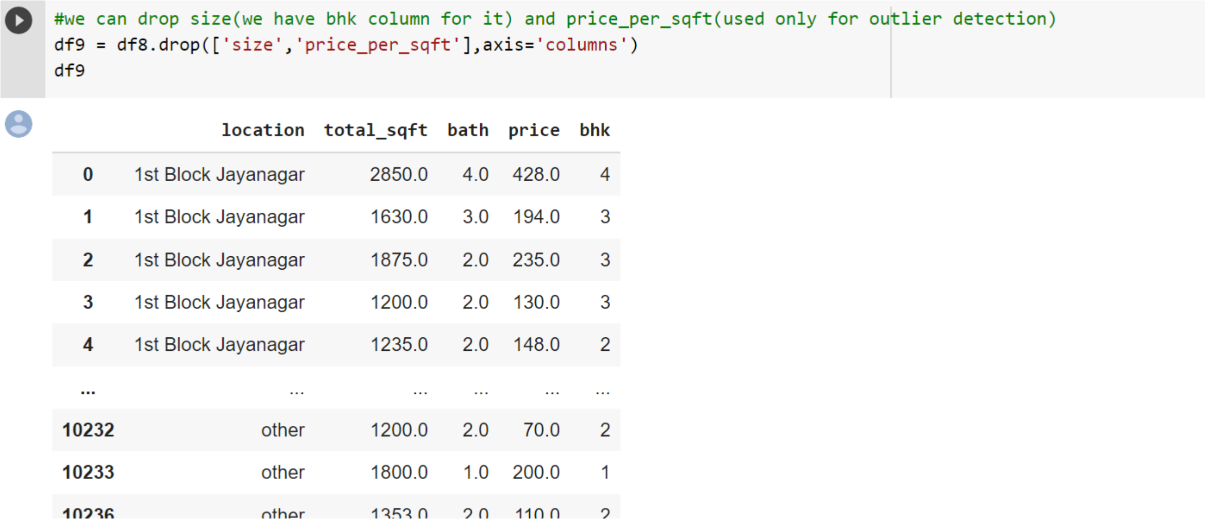
```
[56] #anytime we have bathroom greater than the number of bedroom we can just mart it as a outlier
     df7[df7.bath>df7.bhk+2]#all of these are outliers
```

	location	size	total_sqft	bath	price	bhk	price_per_sqft
1626	Chikkabanavar	4 Bedroom	2460.0	7.0	80.0	4	3252.032520
5238	Nagasandra	4 Bedroom	7000.0	8.0	450.0	4	6428.571429
6711	Thanisandra	3 BHK	1806.0	6.0	116.0	3	6423.034330
8411	other	6 BHK	11338.0	9.0	1000.0	6	8819.897689

```
df8 = df7[df7.bath<df7.bhk+2]
df8.shape
```

```
(7251, 7)
```

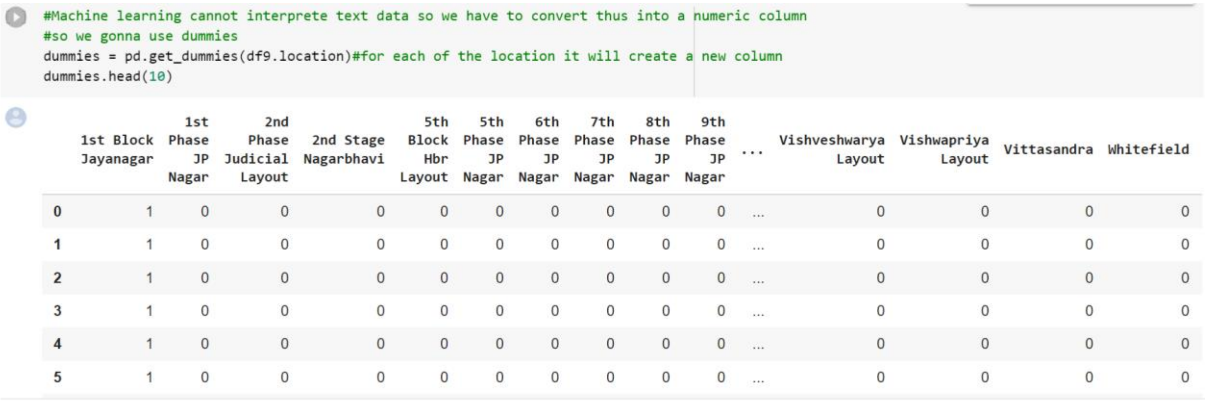
28)



```
#we can drop size(we have bhk column for it) and price_per_sqft(used only for outlier detection)
df9 = df8.drop(['size','price_per_sqft'],axis='columns')
df9
```

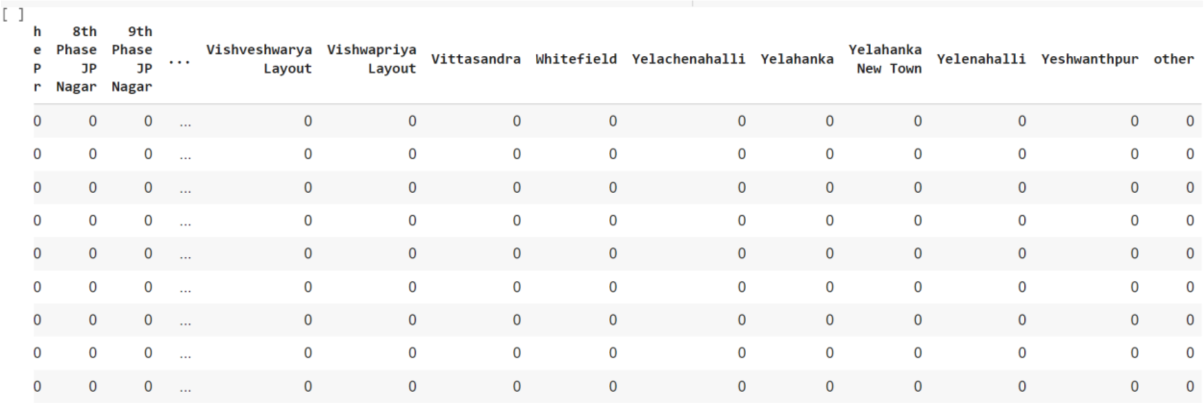
	location	total_sqft	bath	price	bhk
0	1st Block Jayanagar	2850.0	4.0	428.0	4
1	1st Block Jayanagar	1630.0	3.0	194.0	3
2	1st Block Jayanagar	1875.0	2.0	235.0	3
3	1st Block Jayanagar	1200.0	2.0	130.0	3
4	1st Block Jayanagar	1235.0	2.0	148.0	2
...	...	...	...	...	...
10232	other	1200.0	2.0	70.0	2
10233	other	1800.0	1.0	200.0	1
10236	other	1353.0	2.0	110.0	2

29)



```
#Machine learning cannot interprete text data so we have to convert thus into a numeric column
#so we gonna use dummies
dummies = pd.get_dummies(df9.location)#for each of the location it will create a new column
dummies.head(10)
```

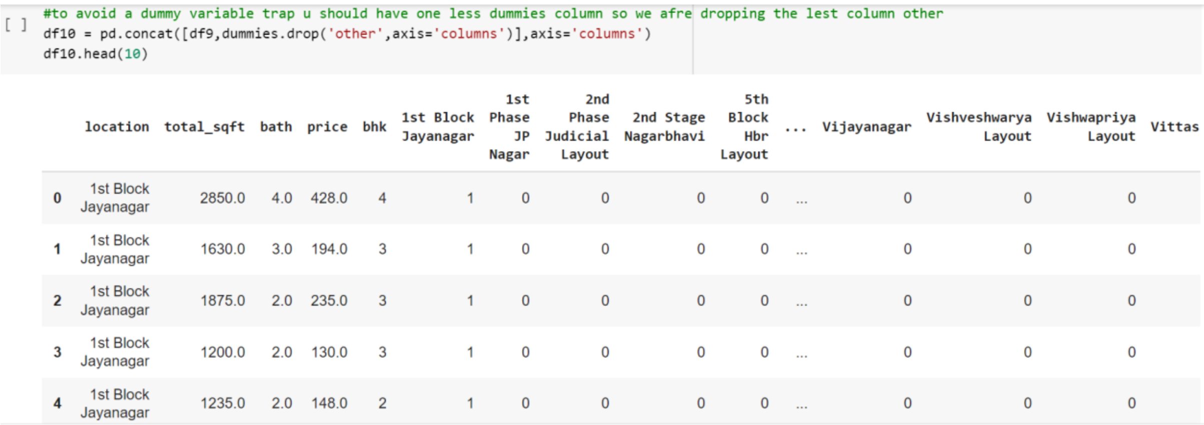
	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	6th Phase JP Nagar	7th Phase JP Nagar	8th Phase JP Nagar	9th Phase JP Nagar	...	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefield
0	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0
2	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0
3	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0
4	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0
5	1	0	0	0	0	0	0	0	0	0	...	0	0	0	0



[ ]

h e P r	8th Phase JP Nagar	9th Phase JP Nagar	...	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefield	Yelachenahalli	Yelahanka	Yelahanka New Town	Yelenahalli	Yeshwanthpur	other
0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	...	0	0	0	0	0	0	0	0	0	0
0	0	0	...	0	0	0	0	0	0	0	0	0	0

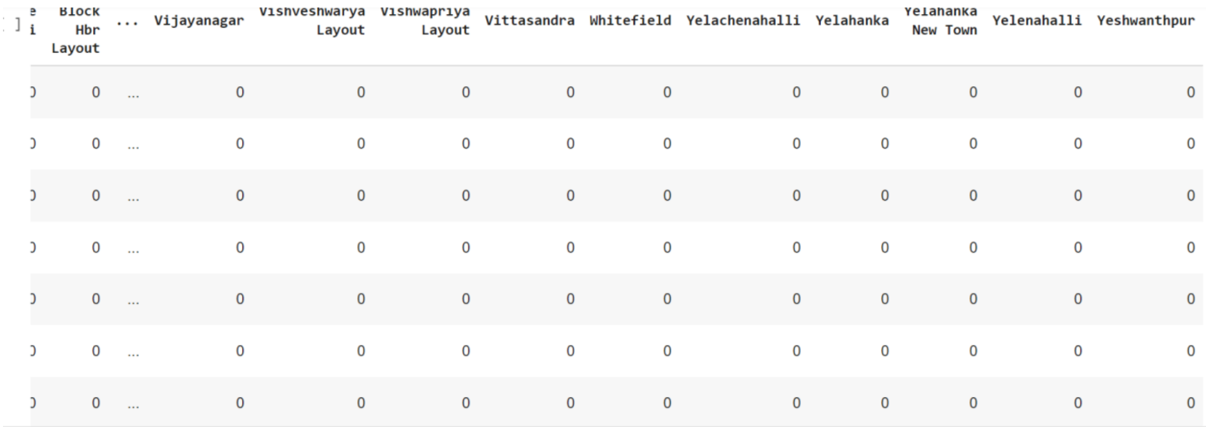
30)



[ ]

```
#to avoid a dummy variable trap u should have one less dummies column so we afre dropping the lest column other
df10 = pd.concat([df9,dummies.drop('other',axis='columns')],axis='columns')
df10.head(10)
```

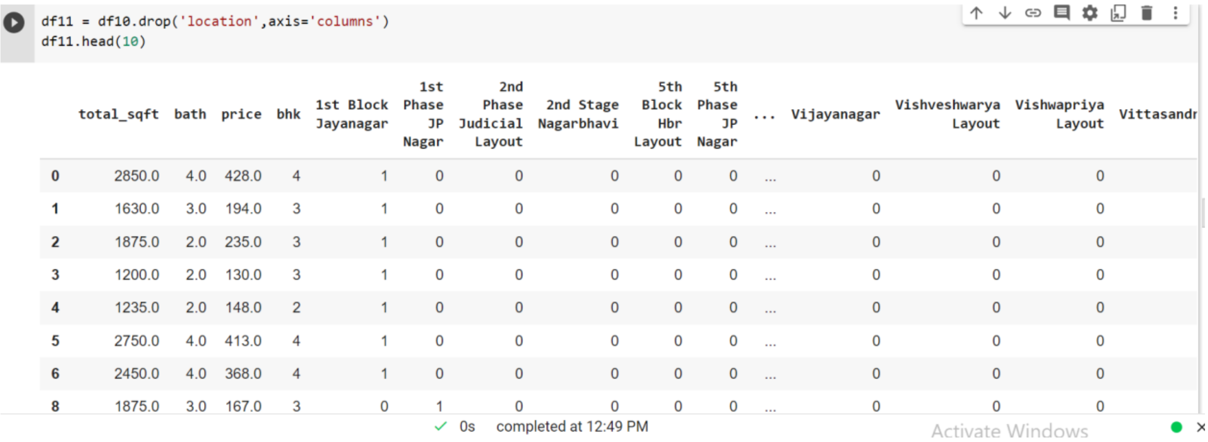
	location	total_sqft	bath	price	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	...	Vijayanagar	Vishveshwarya Layout	Vishwapriya Layout	Vittas
0	1st Block Jayanagar	2850.0	4.0	428.0	4	1	0	0	0	0	...	0	0	0	
1	1st Block Jayanagar	1630.0	3.0	194.0	3	1	0	0	0	0	...	0	0	0	
2	1st Block Jayanagar	1875.0	2.0	235.0	3	1	0	0	0	0	...	0	0	0	
3	1st Block Jayanagar	1200.0	2.0	130.0	3	1	0	0	0	0	...	0	0	0	
4	1st Block Jayanagar	1235.0	2.0	148.0	2	1	0	0	0	0	...	0	0	0	



[ ]

e i	Block Hbr Layout	...	Vijayanagar	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra	Whitefield	Yelachenahalli	Yelahanka	Yelahanka New Town	Yelenahalli	Yeshwanthpur
)	0	...	0	0	0	0	0	0	0	0	0	0
)	0	...	0	0	0	0	0	0	0	0	0	0
)	0	...	0	0	0	0	0	0	0	0	0	0
)	0	...	0	0	0	0	0	0	0	0	0	0
)	0	...	0	0	0	0	0	0	0	0	0	0
)	0	...	0	0	0	0	0	0	0	0	0	0
)	0	...	0	0	0	0	0	0	0	0	0	0

31)



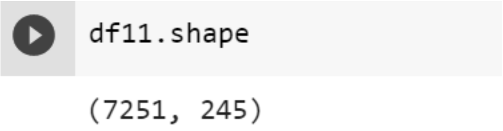
```
df11 = df10.drop('location',axis='columns')
df11.head(10)
```

	total_sqft	bath	price	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	...	Vijayanagar	Vishveshwarya Layout	Vishwapriya Layout	Vittasandr
0	2850.0	4.0	428.0	4	1	0	0	0	0	0	...	0	0	0	
1	1630.0	3.0	194.0	3	1	0	0	0	0	0	...	0	0	0	
2	1875.0	2.0	235.0	3	1	0	0	0	0	0	...	0	0	0	
3	1200.0	2.0	130.0	3	1	0	0	0	0	0	...	0	0	0	
4	1235.0	2.0	148.0	2	1	0	0	0	0	0	...	0	0	0	
5	2750.0	4.0	413.0	4	1	0	0	0	0	0	...	0	0	0	
6	2450.0	4.0	368.0	4	1	0	0	0	0	0	...	0	0	0	
8	1875.0	3.0	167.0	3	0	1	0	0	0	0	...	0	0	0	

✓ 0s completed at 12:49 PM

Activate Windows

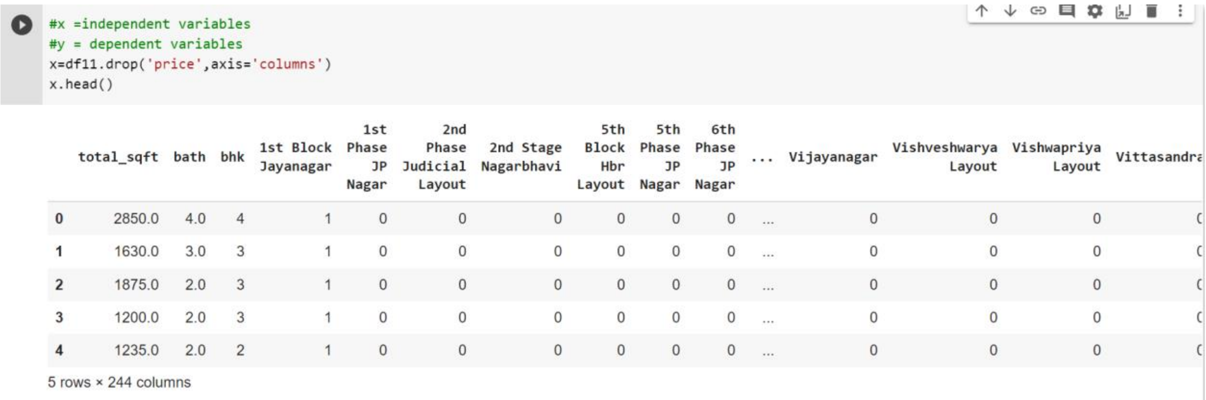
32)



```
df11.shape
```

```
(7251, 245)
```

33)

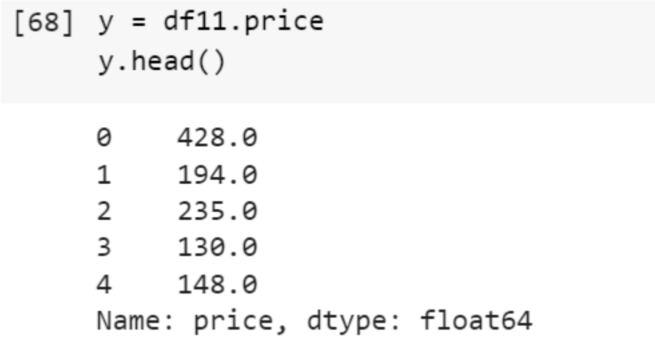


```
#x =independent variables
#y = dependent variables
x=df11.drop('price',axis='columns')
x.head()
```

	total_sqft	bath	bhk	1st Block Jayanagar	1st Phase JP Nagar	2nd Phase Judicial Layout	2nd Stage Nagarbhavi	5th Block Hbr Layout	5th Phase JP Nagar	6th Phase JP Nagar	...	Vijayanagar	Vishveshwarya Layout	Vishwapriya Layout	Vittasandra
0	2850.0	4.0	4	1	0	0	0	0	0	0	...	0	0	0	0
1	1630.0	3.0	3	1	0	0	0	0	0	0	...	0	0	0	0
2	1875.0	2.0	3	1	0	0	0	0	0	0	...	0	0	0	0
3	1200.0	2.0	3	1	0	0	0	0	0	0	...	0	0	0	0
4	1235.0	2.0	2	1	0	0	0	0	0	0	...	0	0	0	0

5 rows × 244 columns

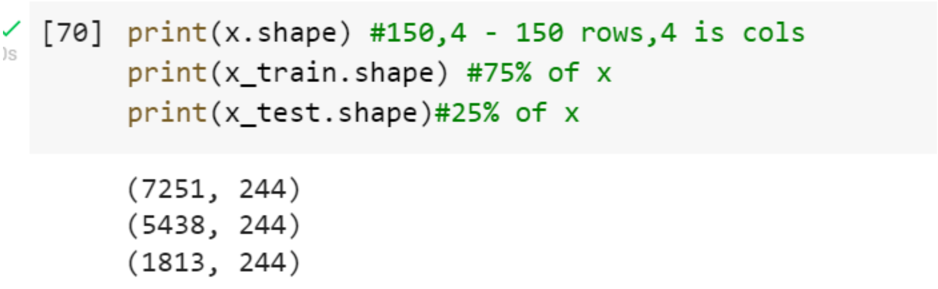
34)



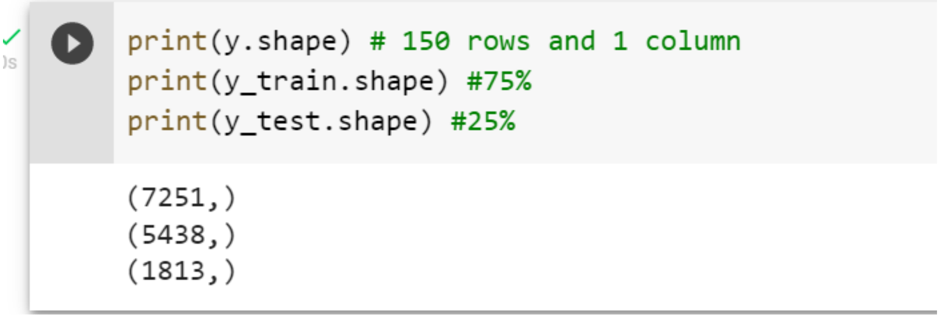
```
[68] y = df11.price
     y.head()
```

```
0    428.0
1    194.0
2    235.0
3    130.0
4    148.0
Name: price, dtype: float64
```

35)

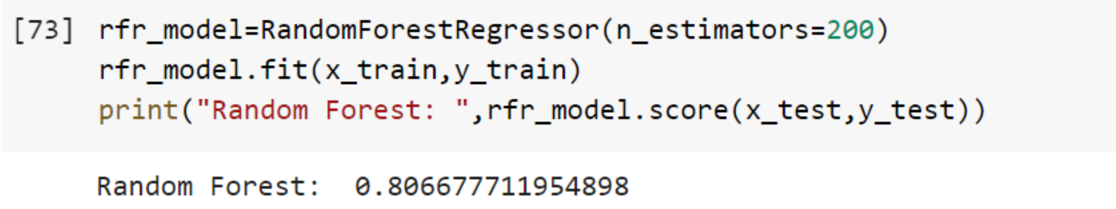


```
[70] print(x.shape) #150,4 - 150 rows,4 is cols
     print(x_train.shape) #75% of x
     print(x_test.shape)#25% of x

     (7251, 244)
     (5438, 244)
     (1813, 244)
```



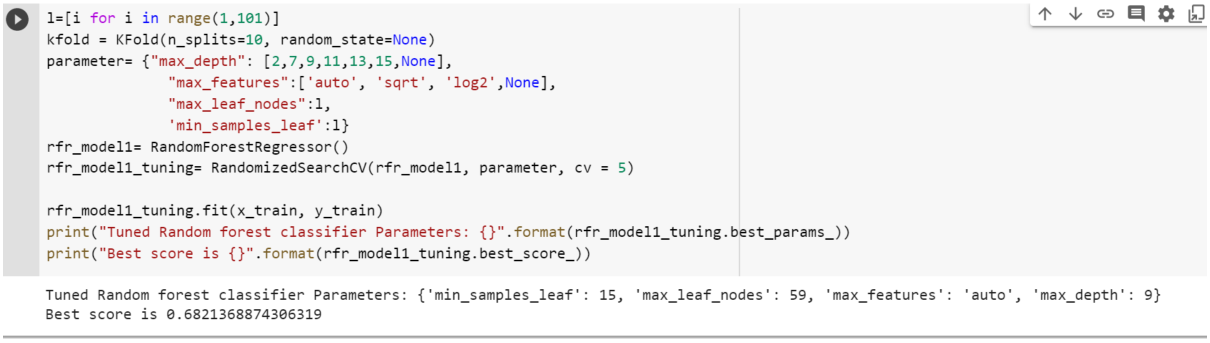
```
print(y.shape) # 150 rows and 1 column
print(y_train.shape) #75%
print(y_test.shape) #25%

(7251,)
(5438,)
(1813,)
```

36)



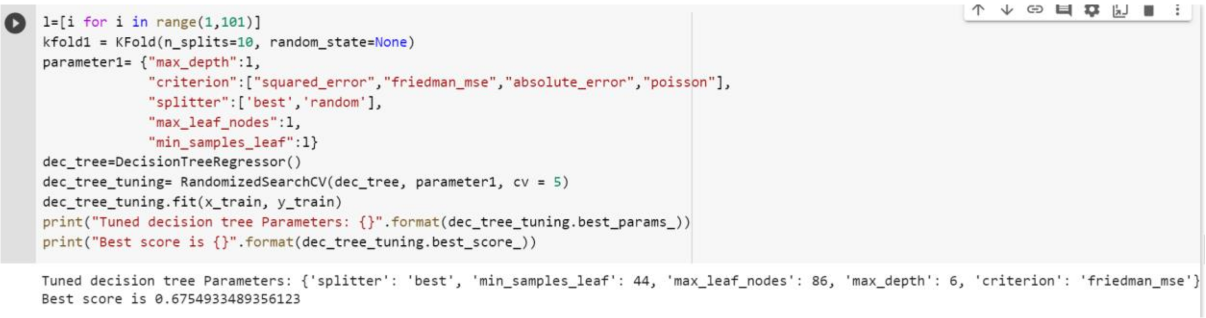
```
[73] rfr_model=RandomForestRegressor(n_estimators=200)
     rfr_model.fit(x_train,y_train)
     print("Random Forest: ",rfr_model.score(x_test,y_test))

     Random Forest:  0.806677711954898
```

37)



```
l=[i for i in range(1,101)]
kfold = KFold(n_splits=10, random_state=None)
parameter= {"max_depth": [2,7,9,11,13,15,None],
            "max_features":['auto', 'sqrt', 'log2',None],
            "max_leaf_nodes":l,
            'min_samples_leaf':l}
rfr_model1= RandomForestRegressor()
rfr_model1_tuning= RandomizedSearchCV(rfr_model1, parameter, cv = 5)

rfr_model1_tuning.fit(x_train, y_train)
print("Tuned Random forest classifier Parameters: {}".format(rfr_model1_tuning.best_params_))
print("Best score is {}".format(rfr_model1_tuning.best_score_))

Tuned Random forest classifier Parameters: {'min_samples_leaf': 15, 'max_leaf_nodes': 59, 'max_features': 'auto', 'max_depth': 9}
Best score is 0.6821368874306319
```

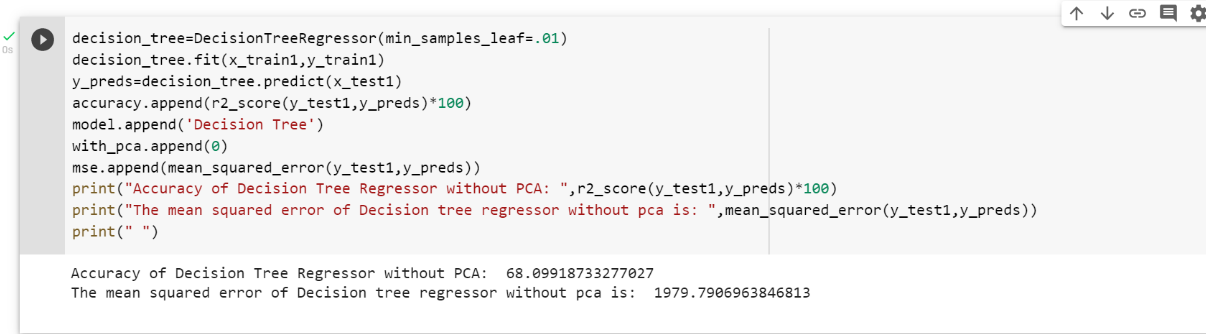
38)



```
l=[i for i in range(1,101)]
kfold1 = KFold(n_splits=10, random_state=None)
parameter1= {"max_depth":l,
            "criterion":["squared_error","friedman_mse","absolute_error","poisson"],
            "splitter":['best','random'],
            "max_leaf_nodes":l,
            "min_samples_leaf":l}
dec_tree=DecisionTreeRegressor()
dec_tree_tuning= RandomizedSearchCV(dec_tree, parameter1, cv = 5)
dec_tree_tuning.fit(x_train, y_train)
print("Tuned decision tree Parameters: {}".format(dec_tree_tuning.best_params_))
print("Best score is {}".format(dec_tree_tuning.best_score_))

Tuned decision tree Parameters: {'splitter': 'best', 'min_samples_leaf': 44, 'max_leaf_nodes': 86, 'max_depth': 6, 'criterion': 'friedman_mse'}
Best score is 0.6754933489356123
```

39)

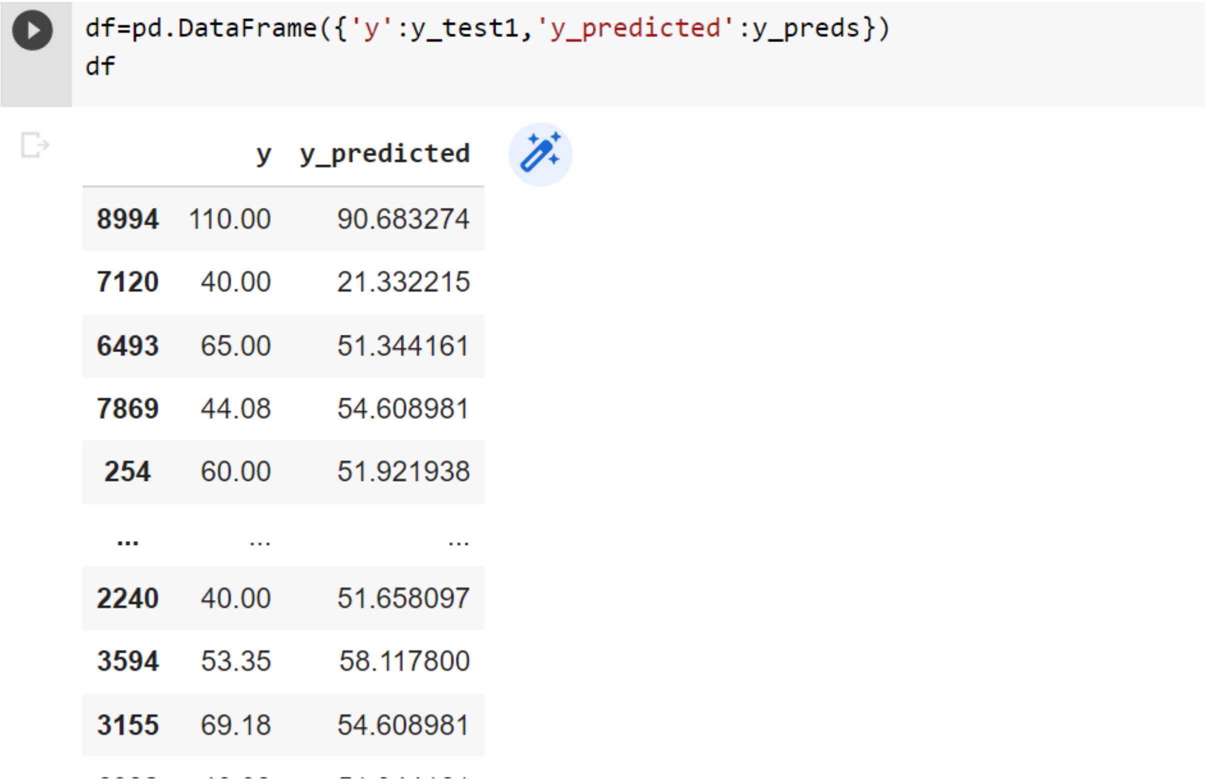
DECISION TREE



```
decision_tree=DecisionTreeRegressor(min_samples_leaf=.01)
decision_tree.fit(x_train1,y_train1)
y_preds=decision_tree.predict(x_test1)
accuracy.append(r2_score(y_test1,y_preds)*100)
model.append('Decision Tree')
with_pca.append(0)
mse.append(mean_squared_error(y_test1,y_preds))
print("Accuracy of Decision Tree Regressor without PCA: ",r2_score(y_test1,y_preds)*100)
print("The mean squared error of Decision tree regressor without pca is: ",mean_squared_error(y_test1,y_preds))
print(" ")
```

```
Accuracy of Decision Tree Regressor without PCA:  68.09918733277027
The mean squared error of Decision tree regressor without pca is:  1979.7906963846813
```

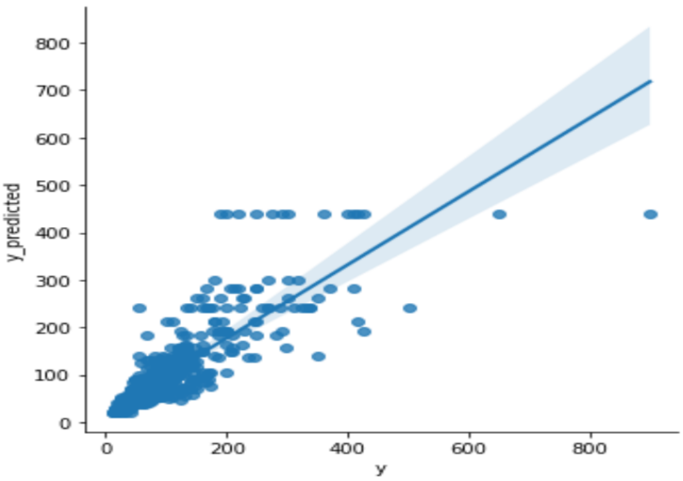
40)

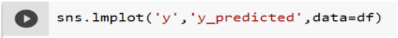


```
df=pd.DataFrame({'y':y_test1,'y_predicted':y_preds})
df
```

	y	y_predicted
8994	110.00	90.683274
7120	40.00	21.332215
6493	65.00	51.344161
7869	44.08	54.608981
254	60.00	51.921938
...	...	...
2240	40.00	51.658097
3594	53.35	58.117800
3155	69.18	54.608981

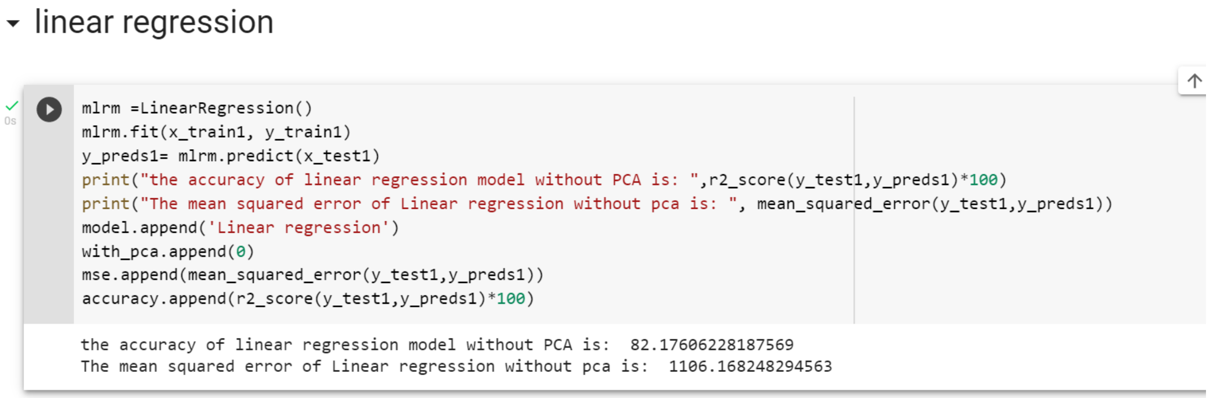
41)





```
sns.lmplot('y','y_predicted',data=df)
```

42)

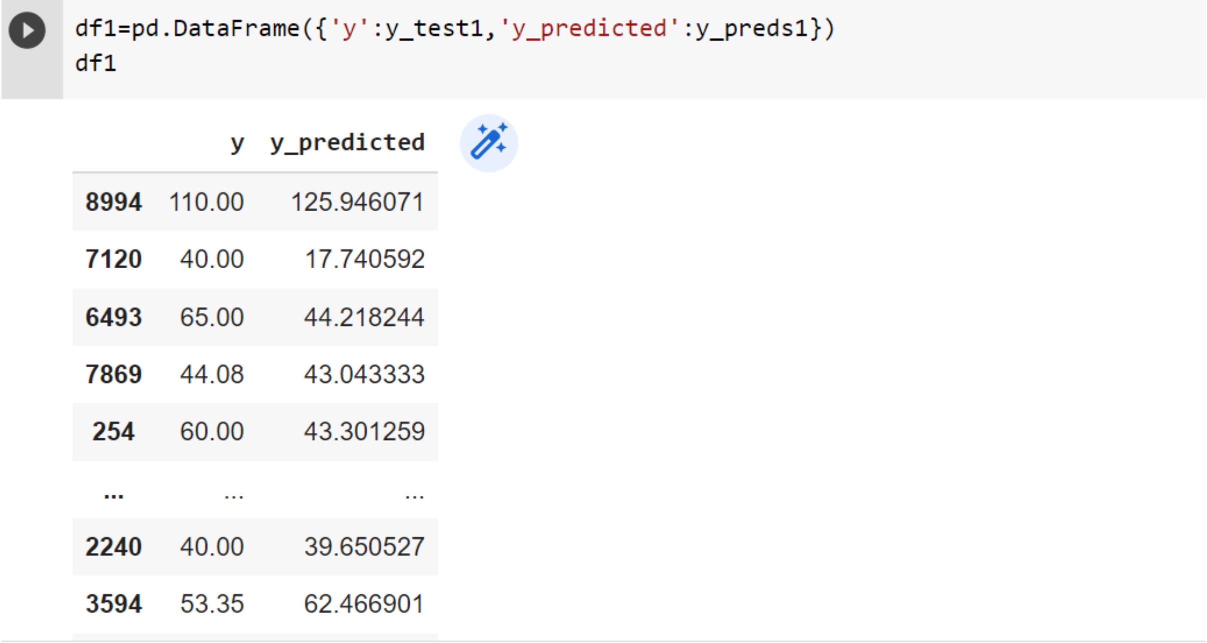


▾ linear regression

```
mlrm =LinearRegression()
mlrm.fit(x_train1, y_train1)
y_preds1= mlrm.predict(x_test1)
print("the accuracy of linear regression model without PCA is: ",r2_score(y_test1,y_preds1)*100)
print("The mean squared error of Linear regression without pca is: ", mean_squared_error(y_test1,y_preds1))
model.append('Linear regression')
with_pca.append(0)
mse.append(mean_squared_error(y_test1,y_preds1))
accuracy.append(r2_score(y_test1,y_preds1)*100)
```

```
the accuracy of linear regression model without PCA is:  82.17606228187569
The mean squared error of Linear regression without pca is:  1106.168248294563
```

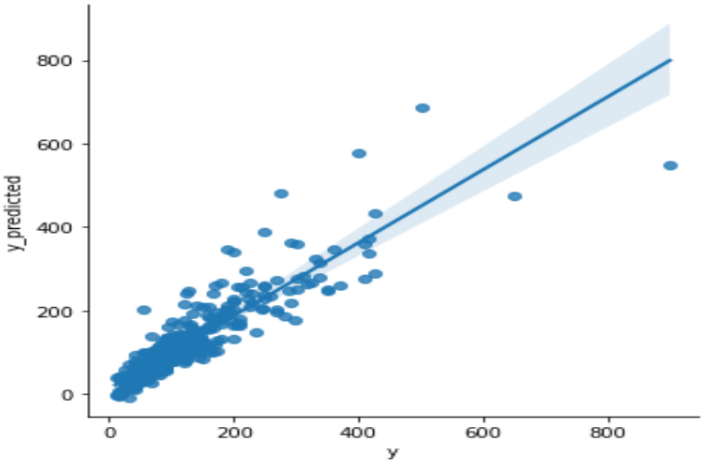
43)



```
df1=pd.DataFrame({'y':y_test1,'y_predicted':y_preds1})
df1
```

	y	y_predicted
8994	110.00	125.946071
7120	40.00	17.740592
6493	65.00	44.218244
7869	44.08	43.043333
254	60.00	43.301259
...	...	...
2240	40.00	39.650527
3594	53.35	62.466901

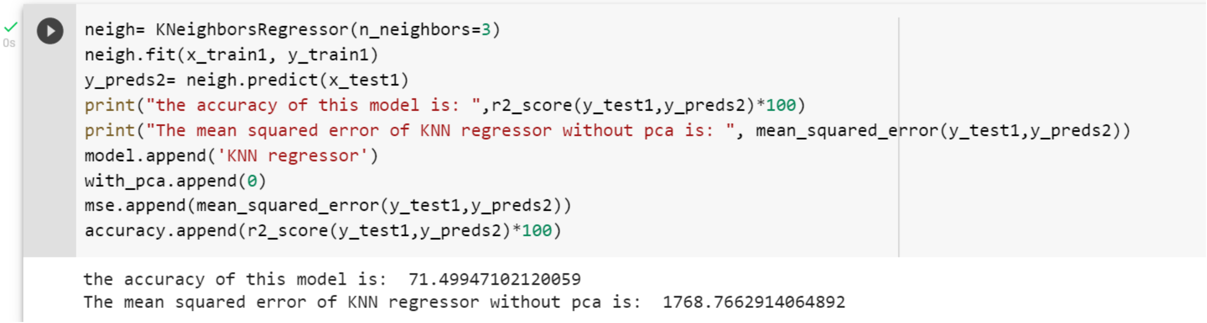
44)



```
sns.lmplot('y','y_predicted',data=df1)
```

45)

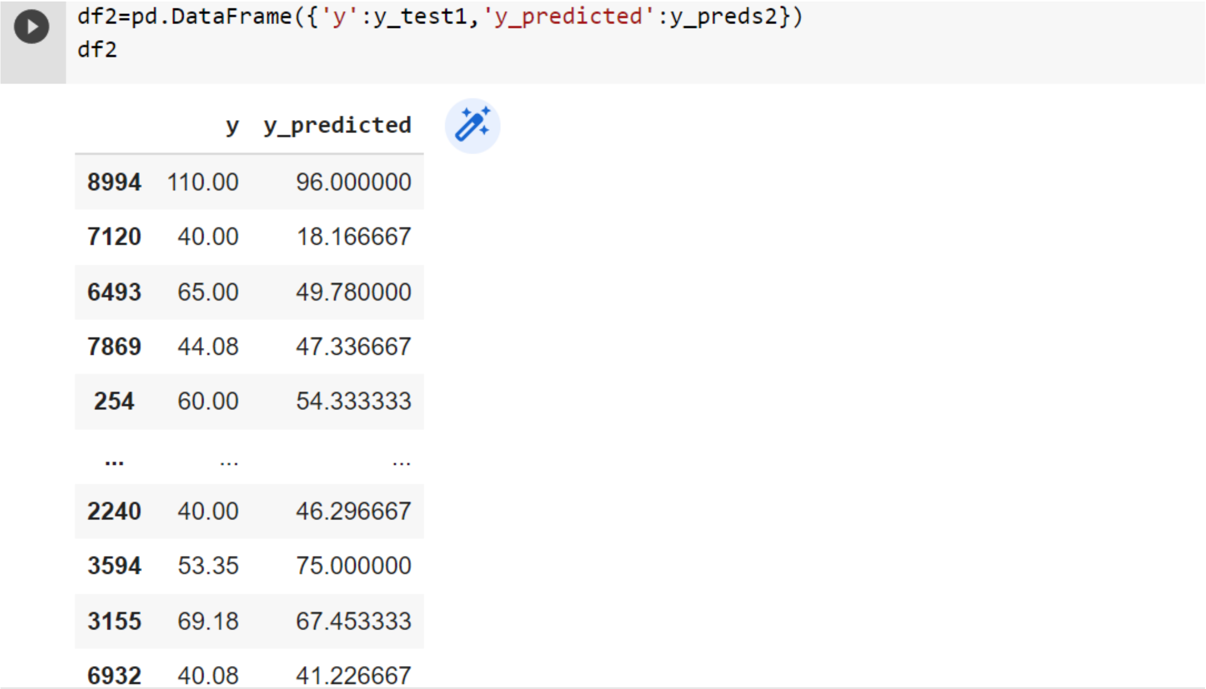
KNN REGRESSOR



```
neigh= KNeighborsRegressor(n_neighbors=3)
neigh.fit(x_train1, y_train1)
y_preds2= neigh.predict(x_test1)
print("the accuracy of this model is: ",r2_score(y_test1,y_preds2)*100)
print("The mean squared error of KNN regressor without pca is: ", mean_squared_error(y_test1,y_preds2))
model.append('KNN regressor')
with_pca.append(0)
mse.append(mean_squared_error(y_test1,y_preds2))
accuracy.append(r2_score(y_test1,y_preds2)*100)
```

```
the accuracy of this model is:  71.49947102120059
The mean squared error of KNN regressor without pca is:  1768.7662914064892
```

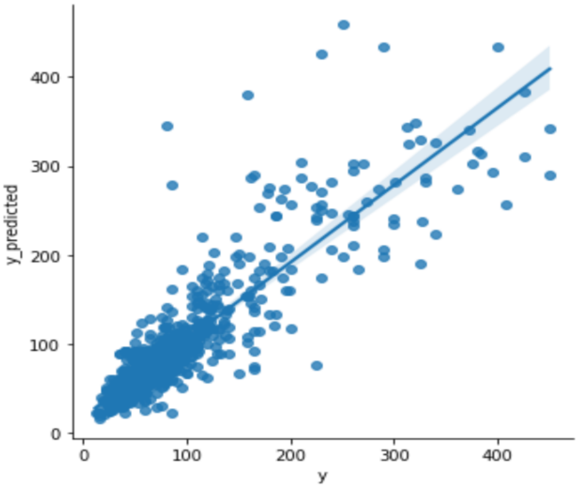
46)

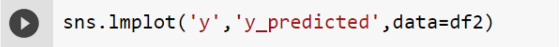


```
df2=pd.DataFrame({'y':y_test1,'y_predicted':y_preds2})
df2
```

	y	y_predicted
8994	110.00	96.000000
7120	40.00	18.166667
6493	65.00	49.780000
7869	44.08	47.336667
254	60.00	54.333333
...	...	...
2240	40.00	46.296667
3594	53.35	75.000000
3155	69.18	67.453333
6932	40.08	41.226667

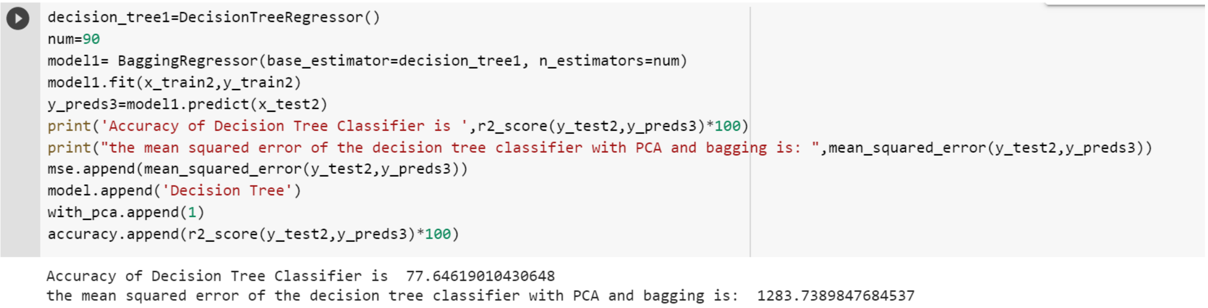
47)





```
sns.lmplot('y','y_predicted',data=df2)
```

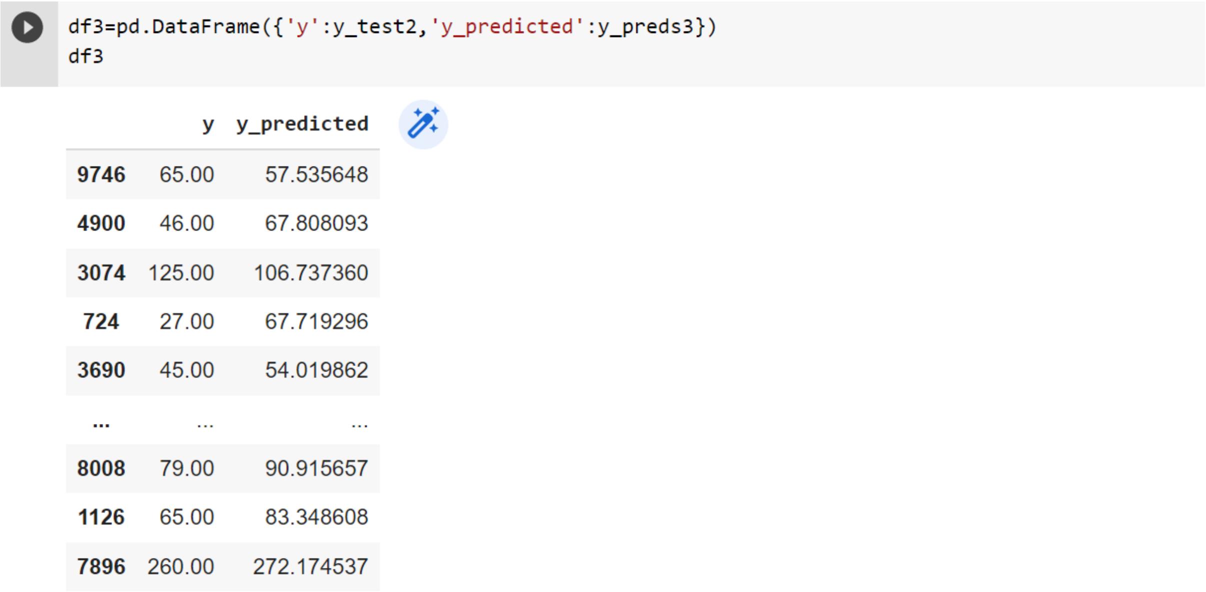
48)



```
decision_tree1=DecisionTreeRegressor()
num=90
model1= BaggingRegressor(base_estimator=decision_tree1, n_estimators=num)
model1.fit(x_train2,y_train2)
y_preds3=model1.predict(x_test2)
print('Accuracy of Decision Tree Classifier is ',r2_score(y_test2,y_preds3)*100)
print("the mean squared error of the decision tree classifier with PCA and bagging is: ",mean_squared_error(y_test2,y_preds3))
mse.append(mean_squared_error(y_test2,y_preds3))
model.append('Decision Tree')
with_pca.append(1)
accuracy.append(r2_score(y_test2,y_preds3)*100)
```

```
Accuracy of Decision Tree Classifier is  77.64619010430648
the mean squared error of the decision tree classifier with PCA and bagging is:  1283.7389847684537
```

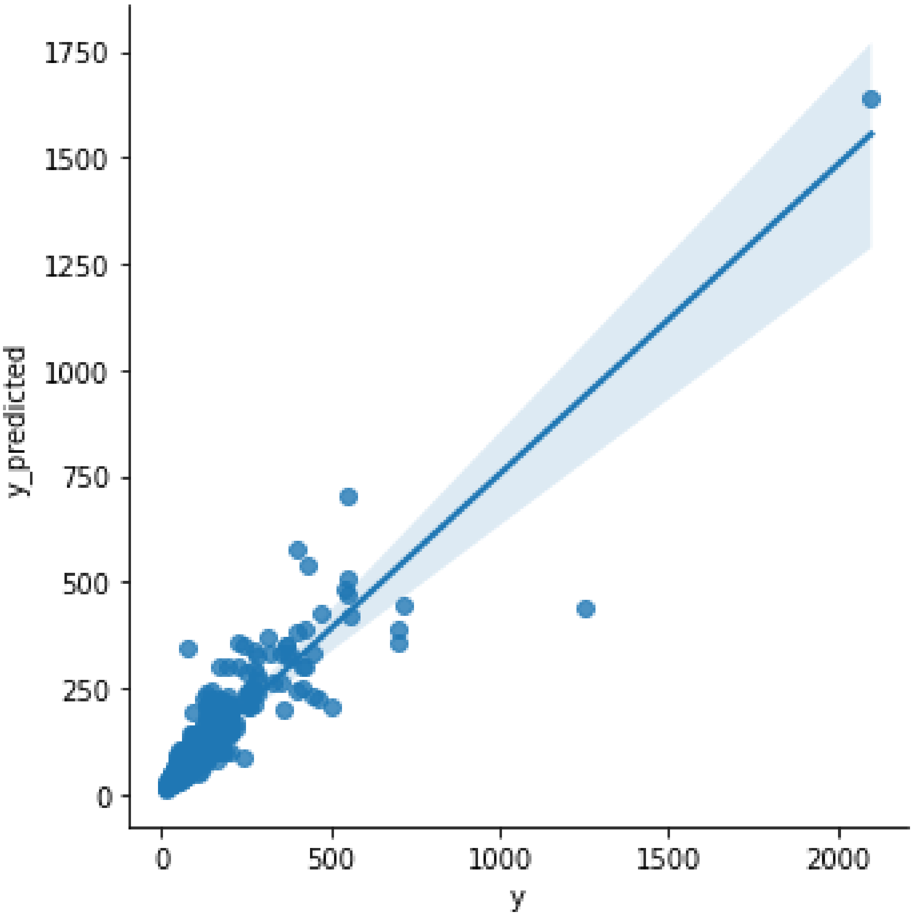
49)



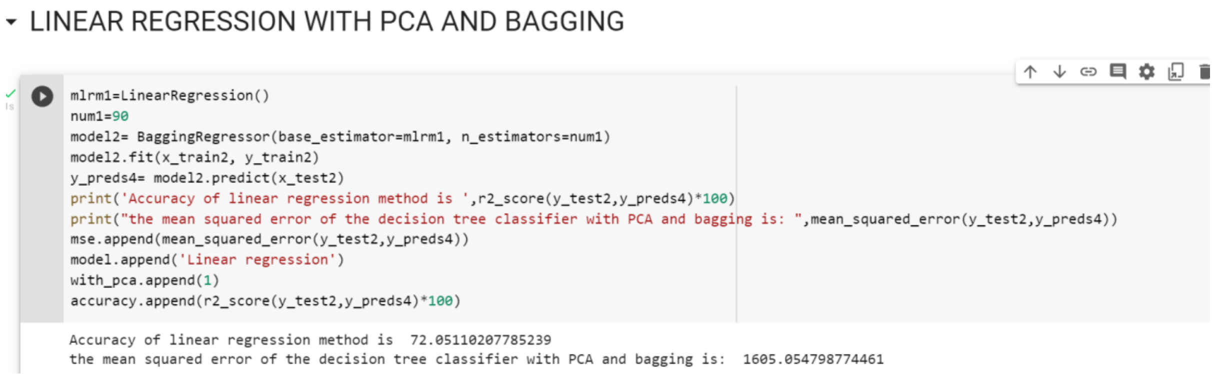
```
df3=pd.DataFrame({'y':y_test2,'y_predicted':y_preds3})
df3
```

	y	y_predicted
9746	65.00	57.535648
4900	46.00	67.808093
3074	125.00	106.737360
724	27.00	67.719296
3690	45.00	54.019862
...	...	...
8008	79.00	90.915657
1126	65.00	83.348608
7896	260.00	272.174537

50)



51)

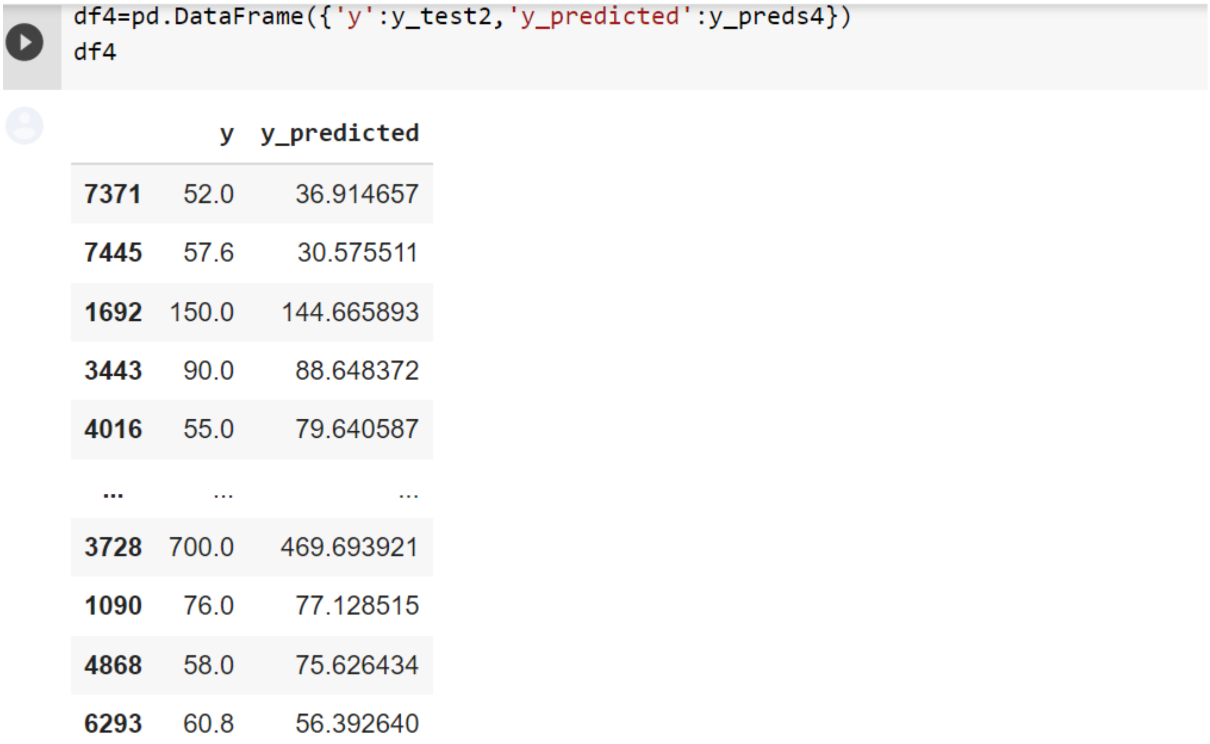


LINEAR REGRESSION WITH PCA AND BAGGING

```
mlrm1=LinearRegression()
num1=90
model2= BaggingRegressor(base_estimator=mlrm1, n_estimators=num1)
model2.fit(x_train2, y_train2)
y_preds4= model2.predict(x_test2)
print('Accuracy of linear regression method is ',r2_score(y_test2,y_preds4)*100)
print("the mean squared error of the decision tree classifier with PCA and bagging is: ",mean_squared_error(y_test2,y_preds4))
mse.append(mean_squared_error(y_test2,y_preds4))
model.append('Linear regression')
with_pca.append(1)
accuracy.append(r2_score(y_test2,y_preds4)*100)
```

```
Accuracy of linear regression method is  72.05110207785239
the mean squared error of the decision tree classifier with PCA and bagging is:  1605.054798774461
```

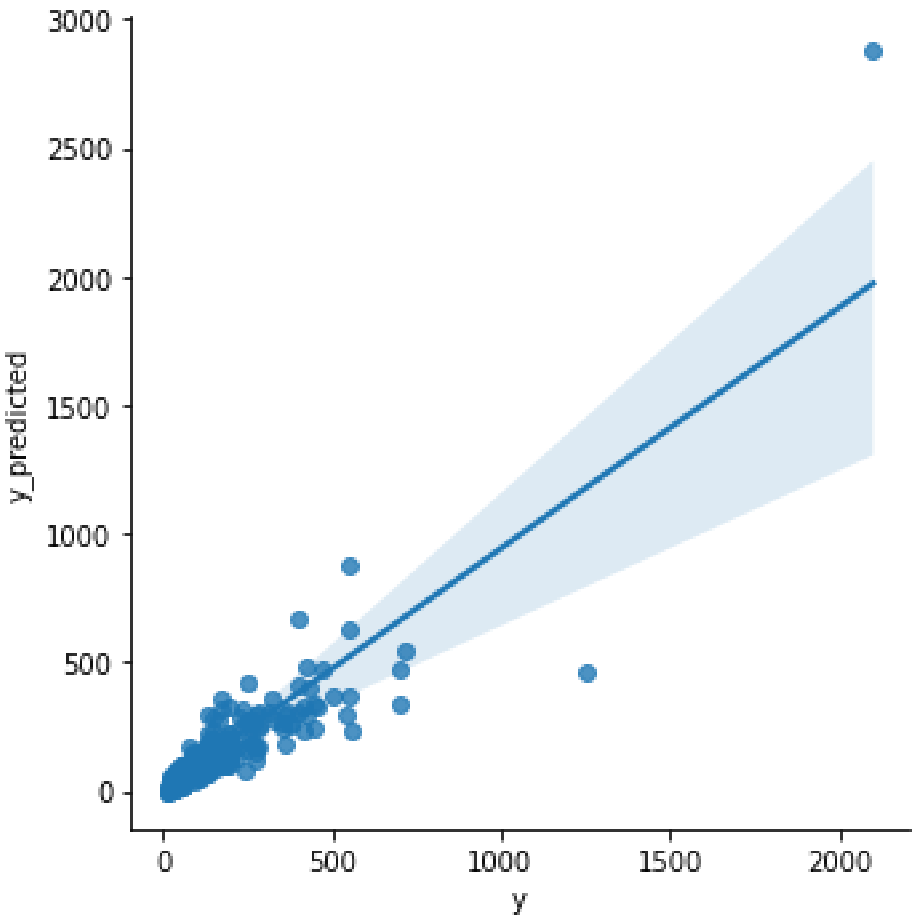
52)



```
df4=pd.DataFrame({'y':y_test2,'y_predicted':y_preds4})
df4
```

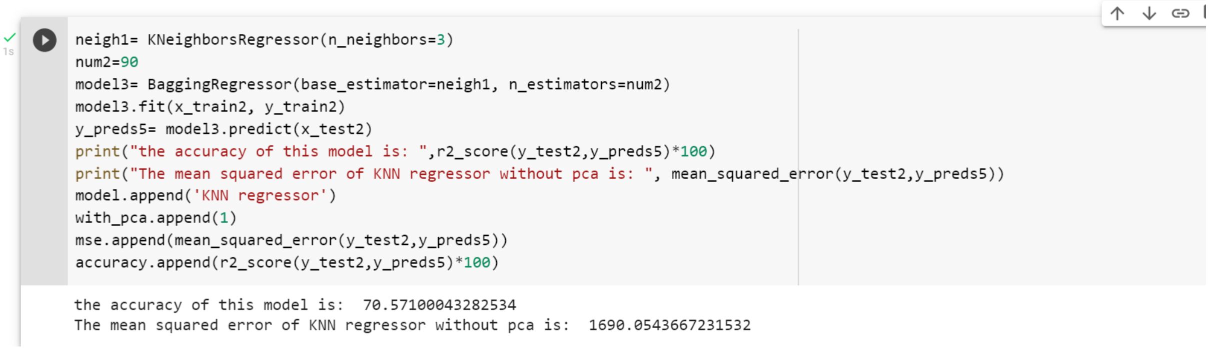
	y	y_predicted
7371	52.0	36.914657
7445	57.6	30.575511
1692	150.0	144.665893
3443	90.0	88.648372
4016	55.0	79.640587
...	...	...
3728	700.0	469.693921
1090	76.0	77.128515
4868	58.0	75.626434
6293	60.8	56.392640

53)



54)

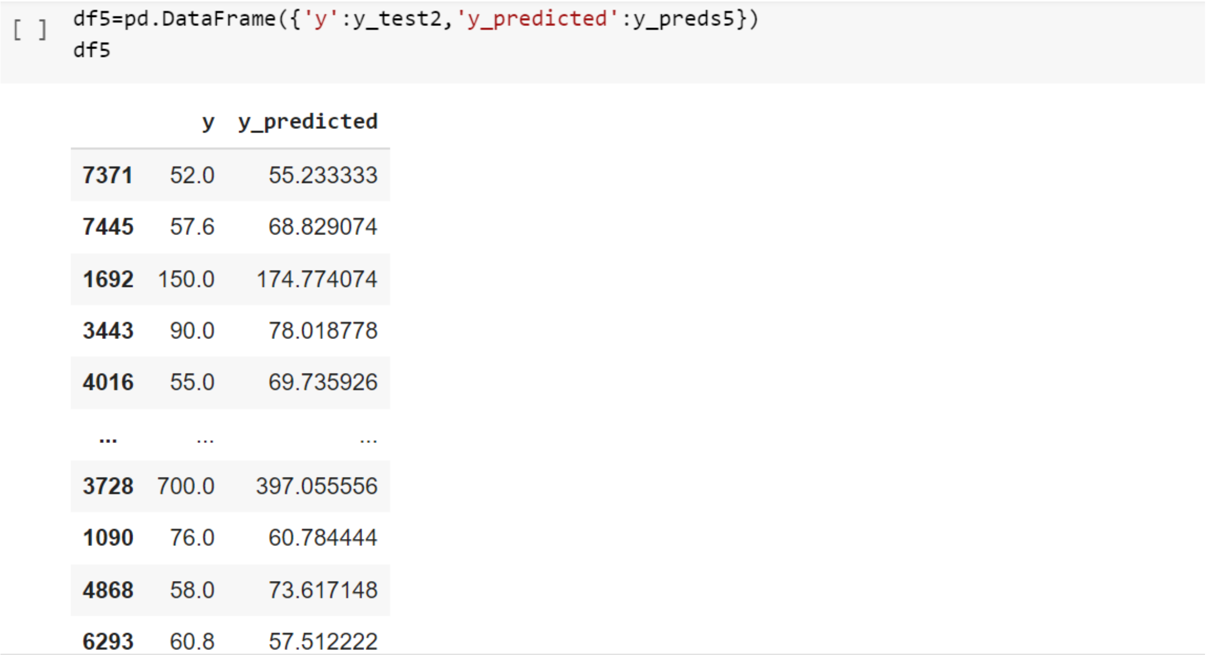
KNN WITH PCA AND BAGGING



```
neigh1= KNeighborsRegressor(n_neighbors=3)
num2=90
model3= BaggingRegressor(base_estimator=neigh1, n_estimators=num2)
model3.fit(x_train2, y_train2)
y_preds5= model3.predict(x_test2)
print("the accuracy of this model is: ",r2_score(y_test2,y_preds5)*100)
print("The mean squared error of KNN regressor without pca is: ", mean_squared_error(y_test2,y_preds5))
model.append('KNN regressor')
with_pca.append(1)
mse.append(mean_squared_error(y_test2,y_preds5))
accuracy.append(r2_score(y_test2,y_preds5)*100)
```

```
the accuracy of this model is:  70.57100043282534
The mean squared error of KNN regressor without pca is:  1690.0543667231532
```

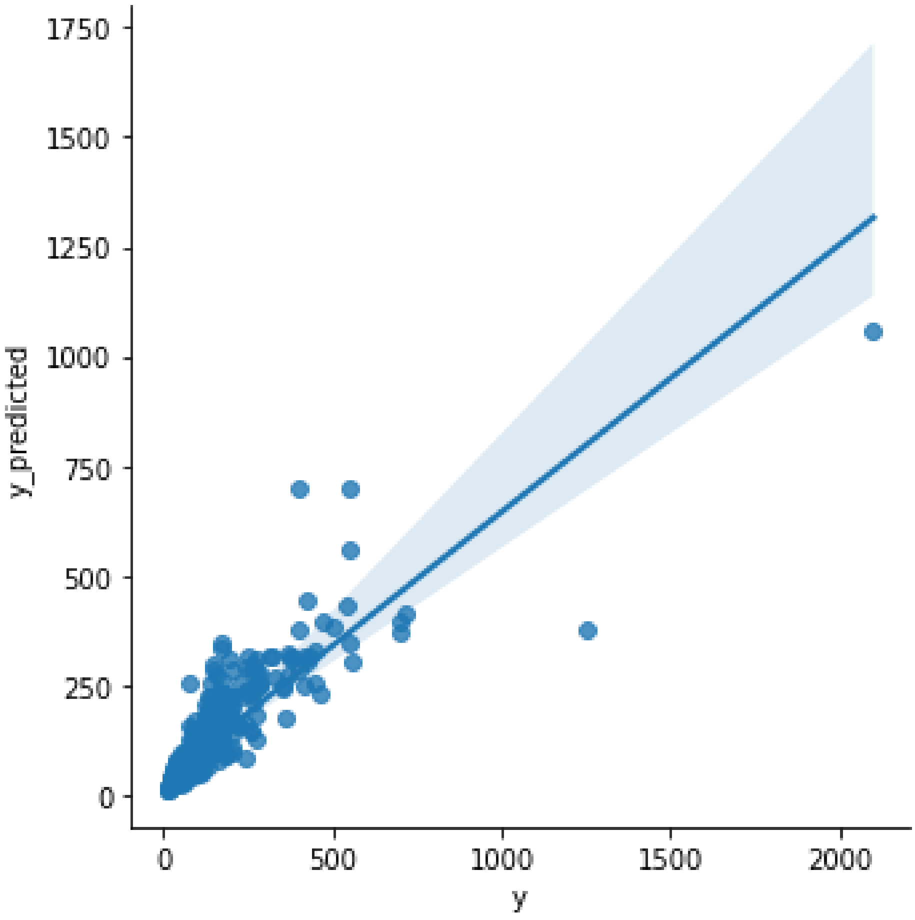
55)



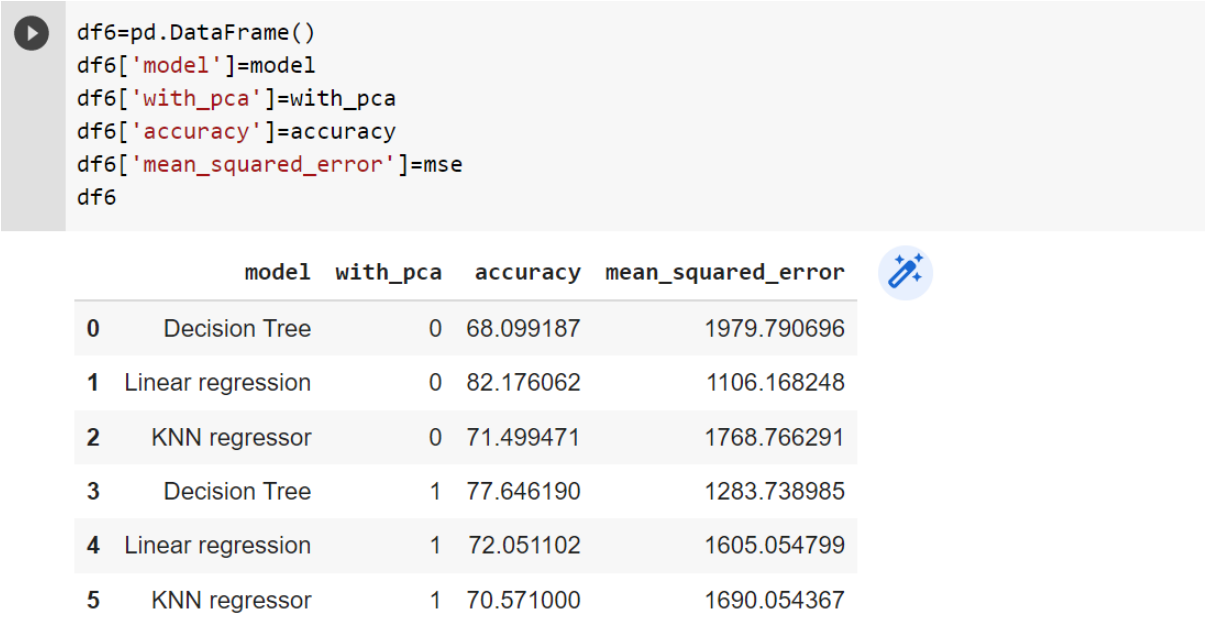
```
[ ]  df5=pd.DataFrame({'y':y_test2,'y_predicted':y_preds5})
     df5
```

	y	y_predicted
7371	52.0	55.233333
7445	57.6	68.829074
1692	150.0	174.774074
3443	90.0	78.018778
4016	55.0	69.735926
...	...	...
3728	700.0	397.055556
1090	76.0	60.784444
4868	58.0	73.617148
6293	60.8	57.512222

56)



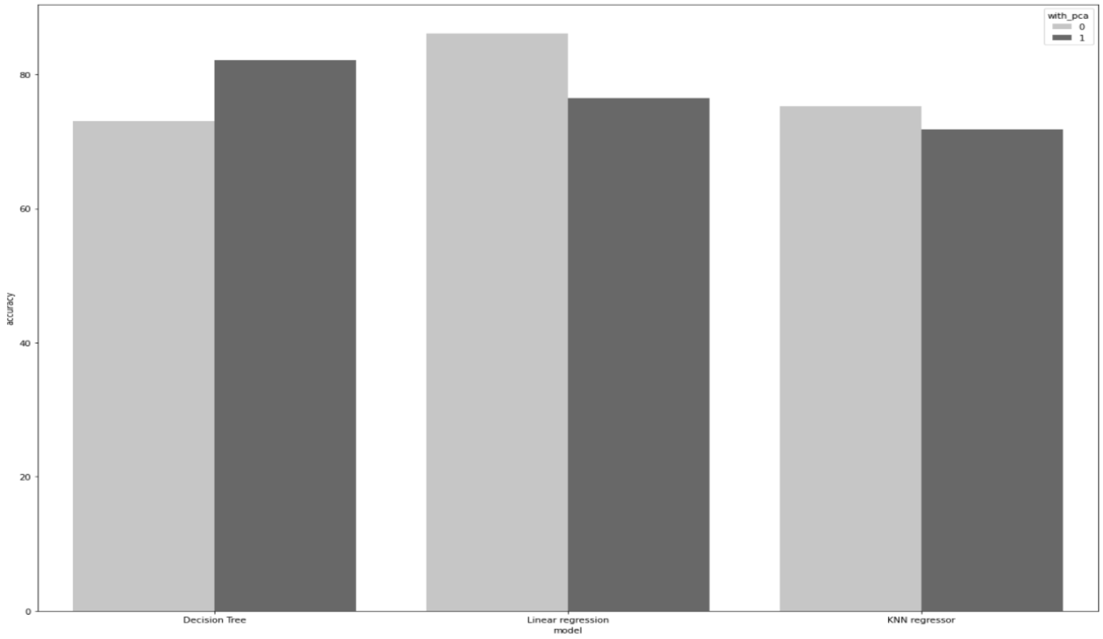
57)

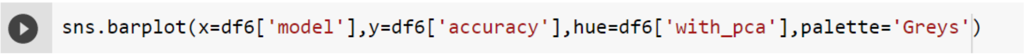


```
df6=pd.DataFrame()
df6['model']=model
df6['with_pca']=with_pca
df6['accuracy']=accuracy
df6['mean_squared_error']=mse
df6
```

	model	with_pca	accuracy	mean_squared_error
0	Decision Tree	0	68.099187	1979.790696
1	Linear regression	0	82.176062	1106.168248
2	KNN regressor	0	71.499471	1768.766291
3	Decision Tree	1	77.646190	1283.738985
4	Linear regression	1	72.051102	1605.054799
5	KNN regressor	1	70.571000	1690.054367

58)





```
sns.barplot(x=df6['model'],y=df6['accuracy'],hue=df6['with_pca'],palette='Greys')
```