In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
house_sales=pd.read_csv(r"C:\Users\s323\Desktop\Gatherings\Data Science\ML\Amit Mi
```

In [3]:
```
house_sales
```

Out[3]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floor
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.(
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.(
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.(
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.(
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.(
...	...	...	...	...	...	...	...	..
21608	263000018	20140521T000000	360000.0	3	2.50	1530	1131	3.(
21609	6600060120	20150223T000000	400000.0	4	2.50	2310	5813	2.(
21610	1523300141	20140623T000000	402101.0	2	0.75	1020	1350	2.(
21611	291310100	20150116T000000	400000.0	3	2.50	1600	2388	2.(
21612	1523300157	20141015T000000	325000.0	2	0.75	1020	1076	2.(

21613 rows × 21 columns

In [4]:
```
house_sales.head()

# without head function it will show first five and last five transcation
```

Out[4]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	wa
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	

5 rows × 21 columns

In [5]:
```
house_sales.shape
```

Out[5]:
```
(21613, 21)
```

In [6]:
```
house_sales.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           21613 non-null  object 
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64  
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64  
 6   sqft_lot       21613 non-null  int64  
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64  
 9   view           21613 non-null  int64  
 10  condition      21613 non-null  int64  
 11  grade          21613 non-null  int64  
 12  sqft_above     21613 non-null  int64  
 13  sqft_basement  21613 non-null  int64  
 14  yr_built       21613 non-null  int64  
 15  yr_renovated   21613 non-null  int64  
 16  zipcode        21613 non-null  int64  
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64  
 20  sqft_lot15     21613 non-null  int64  
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

In [7]:
```
house_sales.columns
```

Out[7]:
```
Index(['id', 'date', 'price', 'bedrooms', 'bathrooms', 'sqft_living',
       'sqft_lot', 'floors', 'waterfront', 'view', 'condition', 'grade',
       'sqft_above', 'sqft_basement', 'yr_built', 'yr_renovated', 'zipcode',
       'lat', 'long', 'sqft_living15', 'sqft_lot15'],
      dtype='object')
```

Data Wrangling

- Check if there is any missing value

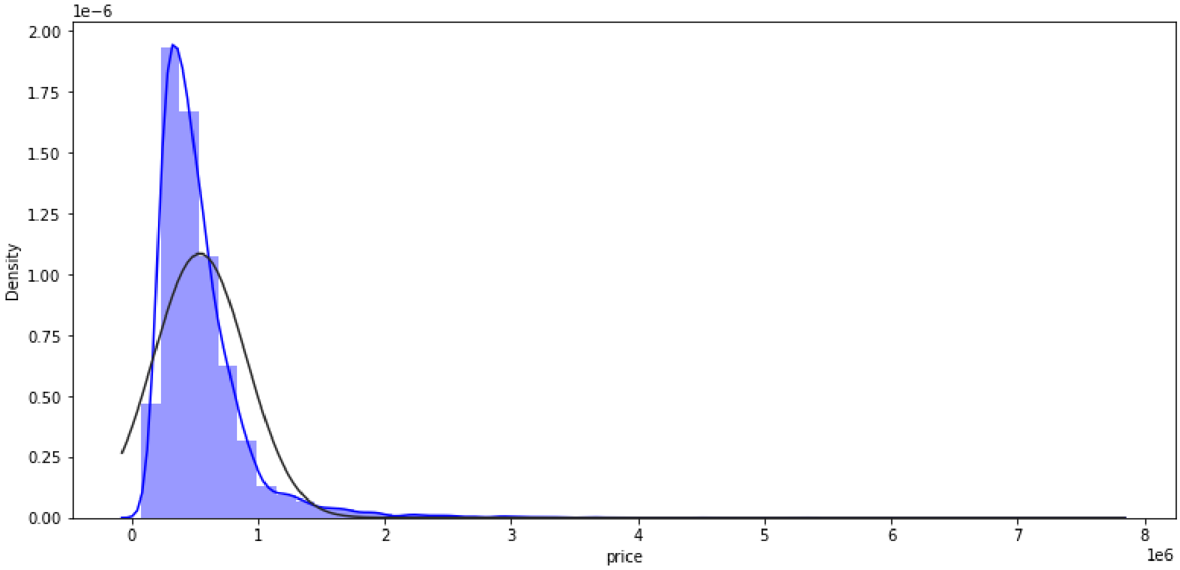
In [8]:
```
house_sales.isnull().sum()
```

Out[8]:
```
id               0
date             0
price            0
bedrooms         0
bathrooms        0
sqft_living      0
sqft_lot         0
floors           0
waterfront       0
view             0
condition        0
grade            0
sqft_above       0
sqft_basement    0
yr_built         0
yr_renovated     0
zipcode          0
lat              0
long             0
sqft_living15    0
sqft_lot15       0
dtype: int64
```

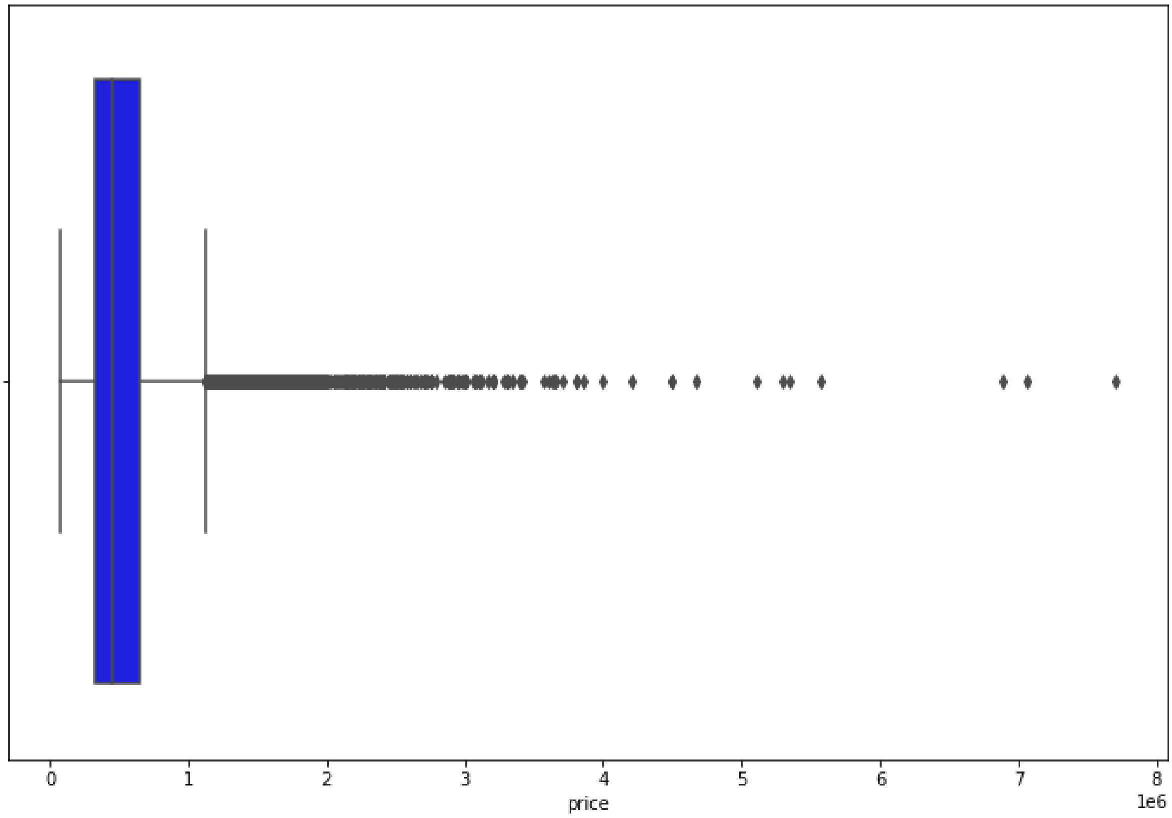
In [9]:
```
# We have to predict house price, In linear regression - Assumption 1. Our target s
# Check the normality
```

Assumptions

In [10]:
```
from scipy.stats import norm
sns.set_style=("whitegrid")
plt.figure(figsize=(13,6))
sns.distplot(house_sales["price"], fit=norm,color="blue")
plt.show()
```



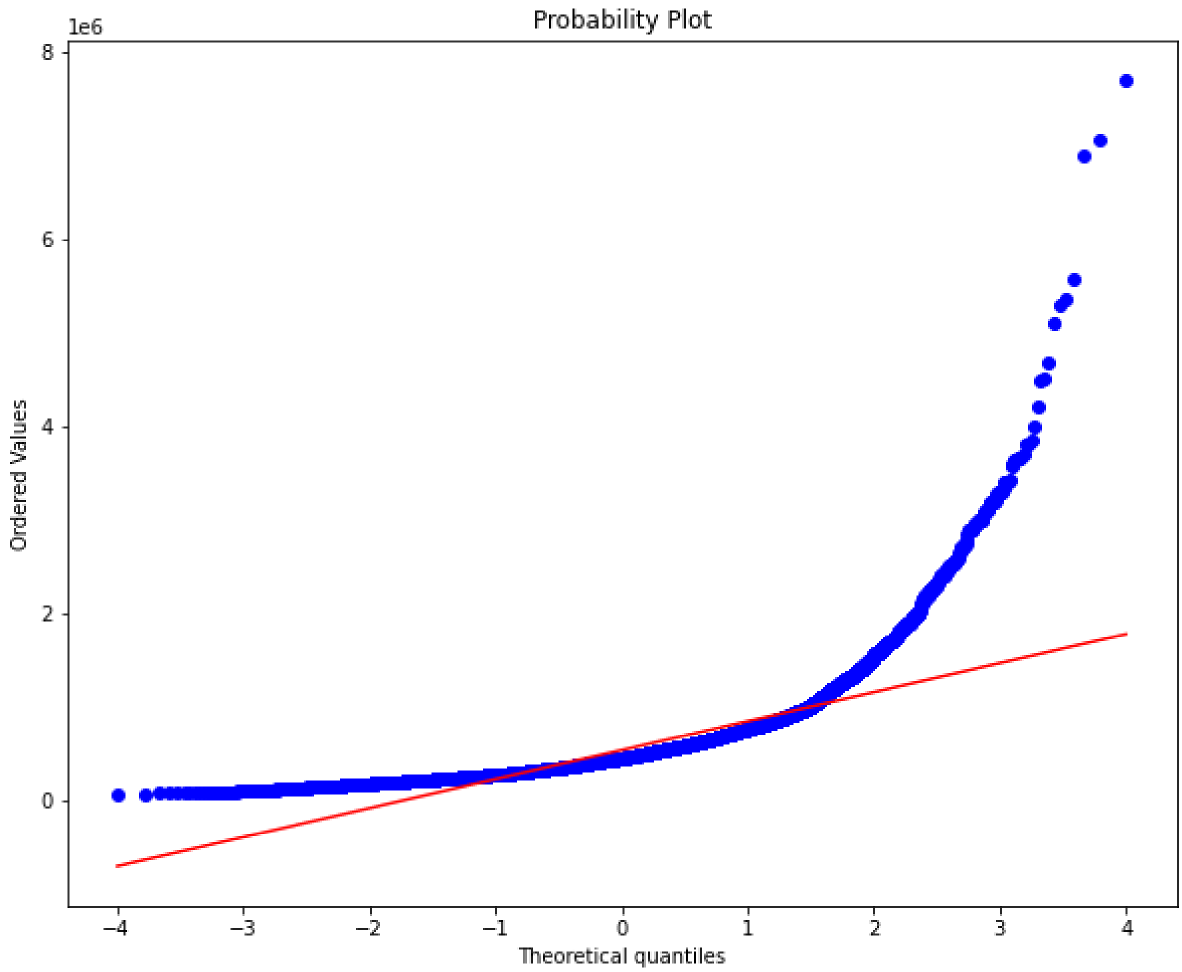
In [11]:
```
# Since the data is +vely skewed, need to exclude the outliers
# need to check if there is any outliers
plt.figure(figsize=(12,8))
sns.set_style=("whitegrid")
sns.boxplot(house_sales["price"], color="blue")
plt.show()
```



In [12]:
```
# we can't remove all outliers since there are alot and we will loose the data
```

In [13]:
```
# QQ PLOT

from scipy import stats
plt.figure(figsize=(10,8))
sns.set_style=("whitegrid")
stats.probplot(house_sales["price"], plot=plt)
plt.show()
```



Handling outliers= we can do it by Z score

- We can do it by IQR as well, but it targets only one column at a time while Z-score is applied on whole data range

In [14]:
```
house_sales.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           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

In [15]:

```
# Since there are 2 coluns which are not playing any major role in data, will drop
house_df=house_sales.drop(["id","date"],axis=1)
```

In [16]:

```
house_df.head()
```

Out[16]:

	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grac
0	221900.0	3	1.00	1180	5650	1.0	0	0	3	
1	538000.0	3	2.25	2570	7242	2.0	0	0	3	
2	180000.0	2	1.00	770	10000	1.0	0	0	3	
3	604000.0	4	3.00	1960	5000	1.0	0	0	5	
4	510000.0	3	2.00	1680	8080	1.0	0	0	3	

In [17]:

```
from scipy import stats
z=stats.zscore(house_df)

# for outliers z score should be greater than value - +3 or -3
```

In [18]:

```
np.where(np.abs(z>4))
# np.where will return the row and col value
```

Out[18]:

```
(array([    1,     5,     21, ..., 21576, 21576, 21576], dtype=int64),
 array([13, 17,  7, ...,  0,  6,  7], dtype=int64))
```

In [19]:

```
# let's see how much data we are loosing
len(np.where(np.abs(z>4))[0])
```

Out[19]:

```
2527
```

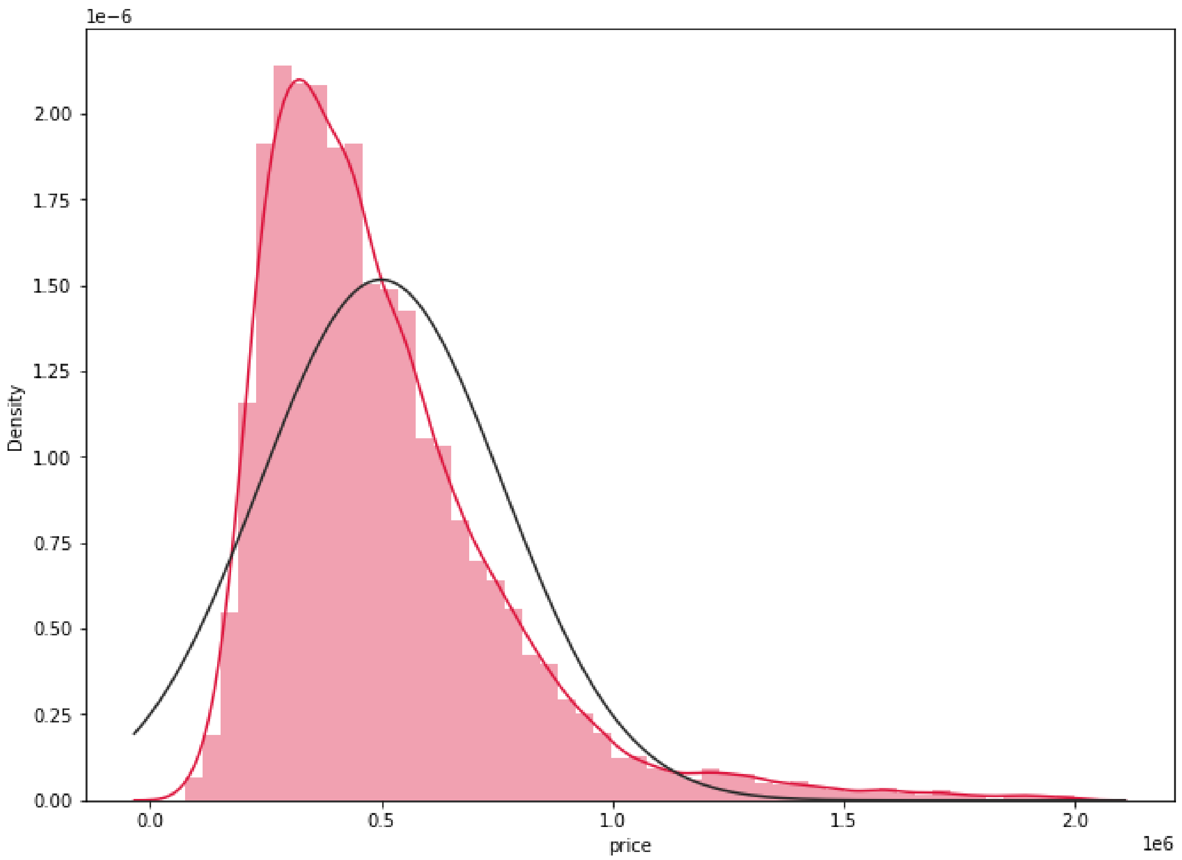
In [20]:
```
# in percentage
len(np.where(np.abs(z>4))[0])/ len(house_df)

# ideal case is 3, but here we are doing it at 4 but can't increase more than 4
```

Out[20]:
```
0.11692037199833434
```

In [21]:
```
house_df.drop(np.where(np.abs(z>4))[0], axis = 0, inplace=True)
```

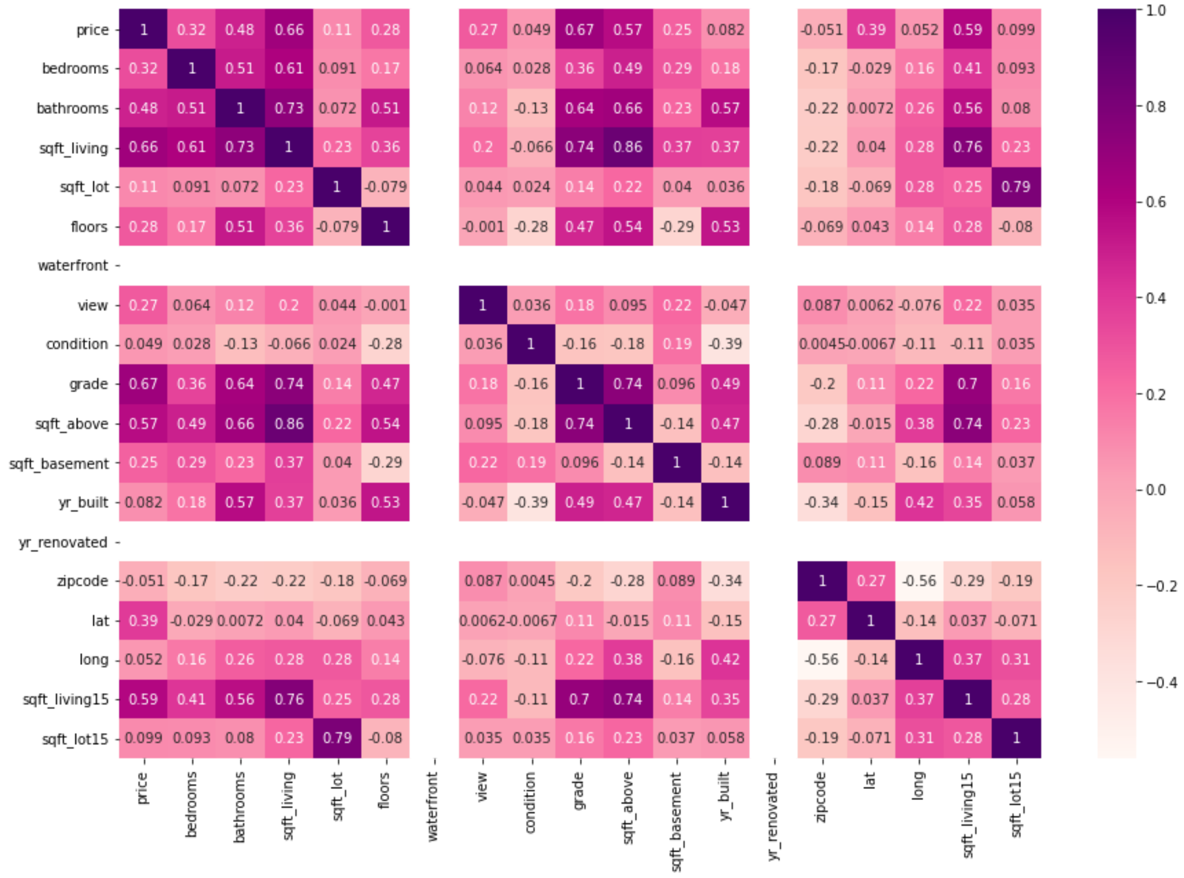
In [22]:
```
from scipy.stats import norm
sns.set_style=("whitegrid")
plt.figure(figsize=(11,8))
sns.distplot(house_df["price"],fit=norm, color="crimson")
plt.show()
```



Corelation

In [23]:
```
plt.figure(figsize=(15,10))
sns.heatmap(house_df.corr(),annot=True,cmap="RdPu")
```

Out[23]:
```
<AxesSubplot:>
```



In [24]:
```
# from the above diagram we can feel, two colomns isnot playing any role in corelat
```

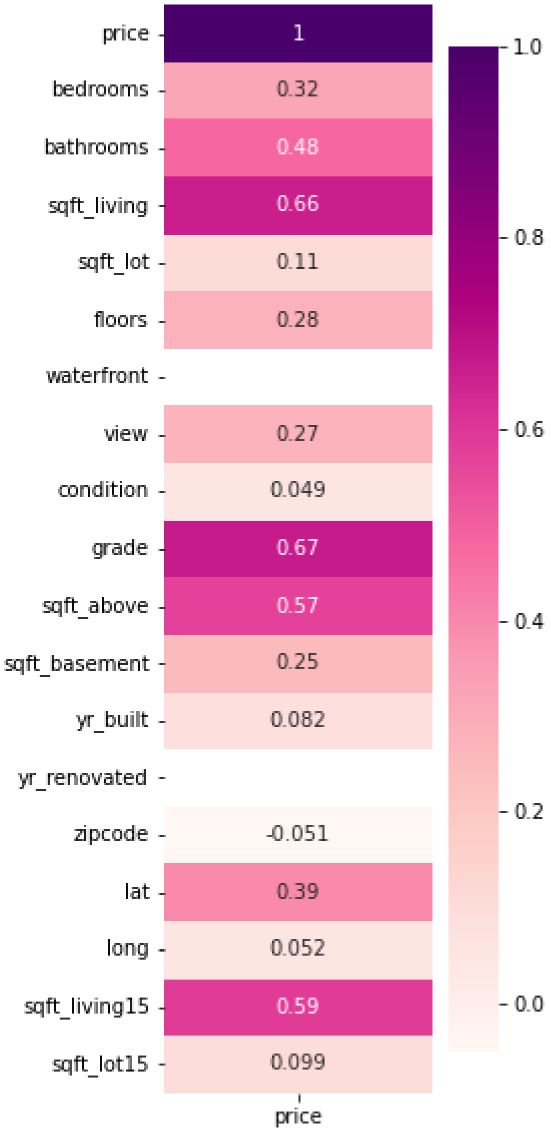
In [25]:
```
# unique value to know if it consists any of the value
house_df['waterfront'].unique()
```

Out[25]:
```
array([0], dtype=int64)
```

In [26]:
```
house_df['yr_renovated'].unique()
```

Out[26]:
```
array([0], dtype=int64)
```

In [27]:
```
plt.figure(figsize=(3,10))
sns.heatmap(house_df.corr()[["price"]],annot=True,cmap="RdPu")
plt.show()
```



In [28]:
```
# Any feature closer to 0, will be eliminated
```

Split X & Y

In [29]:
```
X = house_df.drop(["waterfront","yr_renovated","price"], axis=1)

# Rest we will not drop weaker features now, because we want to do lassoon it, it
```

In [30]:
```
Y = house_df["price"]
```

Train and Test Split

In [31]:
```
from sklearn.model_selection import train_test_split

# choose a random state (0-11) 80% train , 20% test
# random_state = (1-11)
# X is denoting feature and y is representing target

x_train,x_test,y_train, y_test= train_test_split(X,Y, test_size=0.2, random_state=
```

Data Preprocessing, Data Scalar

In [32]:
```
from sklearn.preprocessing import MinMaxScaler
# MinMaxScaler - scale all the features with range (0,1)
scaler=MinMaxScaler(feature_range=(0,1))
# fit + transform
x_train_scaler = scaler.fit_transform(x_train)
# only transform for test
x_test_scaler = scaler.transform(x_test)
```

LASSO Regression

In [33]:
```
from sklearn.linear_model import Lasso
# hyperparameters : alpha = (0.00001-1), max_iter = higher if sample size is small
lasso_model=Lasso(alpha = 0.0001, max_iter = 100000)

# Lasso model is created afterwards we do need to fit that
```

In [34]:
```
lasso_model.fit(x_train_scaler, y_train)
```

Out[34]:
```
Lasso(alpha=0.0001, max_iter=100000)
```

In [35]:
```
# Algorithm is untrained and later we trained them and make them model to predict
lasso_model.score(x_test_scaler, y_test)
```

Out[35]:
```
0.7049161200498151
```

Regression Metrics

- It is used for check the performance of regression algorithm (also called accuracy matrix)
- MSE, MSA

In [36]:
```
# making new Predictions (yhat)
yhat = lasso_model.predict(x_test_scaler)
```

In [37]:
```
# r2 SCORE, MSE, MAE
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
# coefficient of determination (0 - 1)
r2_score(y_test, yhat)
```

Out[37]:
```
0.7049161200498151
```

In [38]:
```
mean_absolute_error(y_test, yhat)
```

Out[38]:
```
102494.56640321515
```

In [39]:
```
# RMSE
np.sqrt(mean_squared_error(y_test, yhat))

# here we can see that the RMSE value is way bigger but still we need to see actua
```

Out[39]:
```
147902.42875551208
```

In [40]:
```
Y.mean()
# quite away from orginal mean
```

Out[40]:
```
499567.6842822532
```

Feature selection using Lasso

- how does lasso useful in feature selection

In [41]:
```
# lasso.coef_ is beta value such as
# Lasso model will shrink the least impt feature's coefficent (b0, b1,b2) closer to
lasso_model.coef_
```

Out[41]:
```
array([-134752.6994601 ,  159438.26741008, 1101684.57475157,
         36468.41885377,   69724.91743785,  130568.19229457,
        111048.62667001,  994031.93381119, -435980.42470607,
       -224314.29930833, -269815.46292146,  -79402.67035199,
        346568.24409264,  -91777.0024419 ,  166977.06157302,
       -104014.7823389 ])
```

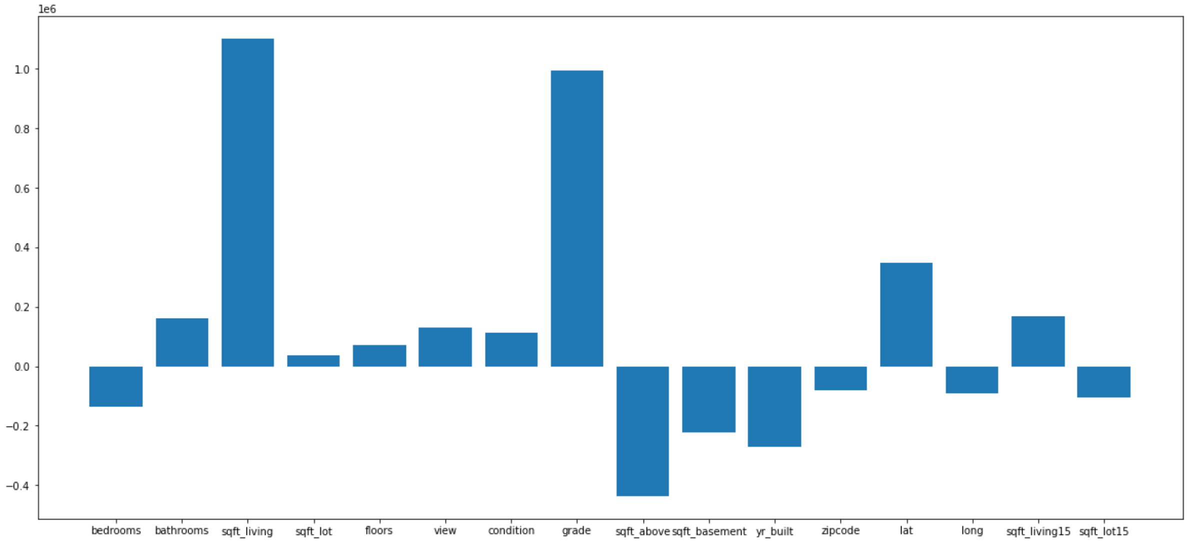
In [42]:
```
# in order to know that we had to create a dataframe, of columns and columns coeff
# creating empty data frame and then adding columns with values
lasso_coef = pd.DataFrame()
lasso_coef['Columns'] = x_train.columns
lasso_coef['Coefficient Estimate'] = pd.Series(lasso_model.coef_)
print(lasso_coef)

# b0 won't be there since it is for intializing
```

```
          Columns  Coefficient Estimate
0        bedrooms         -1.347527e+05
1       bathrooms          1.594383e+05
2     sqft_living          1.101685e+06
3        sqft_lot          3.646842e+04
4          floors          6.972492e+04
5            view          1.305682e+05
6       condition          1.110486e+05
7           grade          9.940319e+05
8      sqft_above         -4.359804e+05
9   sqft_basement         -2.243143e+05
10       yr_built         -2.698155e+05
11        zipcode         -7.940267e+04
12            lat          3.465682e+05
13           long         -9.177700e+04
14  sqft_living15          1.669771e+05
15     sqft_lot15         -1.040148e+05
```

In [43]:
```
# columns vs their coeff and closer to 0, we will eliminate them
# we can play with alpha value
plt.figure(figsize=(20,9))
plt.bar(lasso_coef['Columns'],lasso_coef['Coefficient Estimate'])
```

Out[43]:
```
<BarContainer object of 16 artists>
```



In [44]:
```
# closer to 0, squarefit plot and zipcode can be eliminated
# we can change with alpha value as well
# alpha is learning rate
```

Lasso Regression with alpha = 1

- Basically we will retrain our model with lasso at alpha = 1

In [45]:
```
from sklearn.linear_model import Lasso
# hyperparameters : alpha = (0.00001-1), max_iter = higher if sample size is small
lasso_model=Lasso(alpha = 1, max_iter = 100000)
```

In [46]:
```
lasso_model.fit(x_train_scaler, y_train)
```

Out[46]:
```
Lasso(alpha=1, max_iter=100000)
```

In [47]:
```
# Algorithm is untrained and later we trained them and make them model to predict
lasso_model.score(x_test_scaler, y_test)
```

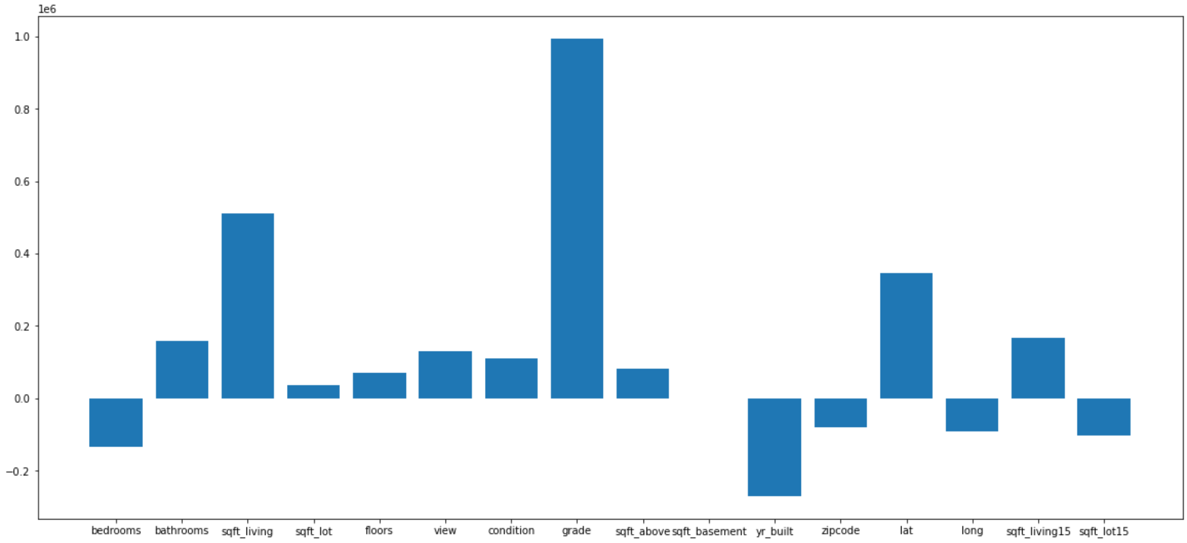
Out[47]:
```
0.7049159098093531
```

In [48]:
```
lasso_coef = pd.DataFrame()
lasso_coef['Columns'] = x_train.columns
lasso_coef['Coefficient Estimate'] = pd.Series(lasso_model.coef_)
print(lasso_coef)
```

```
          Columns  Coefficient Estimate
0        bedrooms        -134585.475691
1       bathrooms         159338.157479
2     sqft_living         510405.741327
3        sqft_lot          35770.376987
4          floors          69717.404177
5            view         130561.199316
6       condition         111009.768583
7           grade         993950.584686
8      sqft_above          82304.299707
9   sqft_basement              0.000000
10       yr_built        -269764.784051
11        zipcode         -79363.261753
12            lat         346558.381581
13           long         -91721.988286
14  sqft_living15         166949.531808
15     sqft_lot15        -103392.745685
```

In [49]:
```
plt.figure(figsize=(20,9))
plt.bar(lasso_coef['Columns'],lasso_coef['Coefficient Estimate'])
```

Out[49]: <BarContainer object of 16 artists>



In [50]:
```
# waterfront, yearrenovated, price are from starting and remaining 2 features from
X = house_df.drop(['waterfront','sqft_basement','sqft_lot15','yr_renovated','price
Y = house_df['price']
```

In [51]:
```
x_train,x_test,y_train,y_test=train_test_split(X,Y, test_size=0.2, random_state=8)
```

In [52]:
```
scaler=MinMaxScaler(feature_range=(0,1))
#fit + transform
x_train.scaler=scaler.fit_transform(x_train)
# only transform for test
x_test.scaler=scaler.transform(x_test)
```

In [53]:
```
lasso_model = Lasso(alpha = 1, max_iter = 100000)
```

In [54]:
```
lasso_model.fit(x_train_scaler, y_train)
```

Out[54]: Lasso(alpha=1, max_iter=100000)

In [55]:
```
lasso_model.score(x_test_scaler, y_test)
```

Out[55]: 0.7049159098093531

In [56]:
```
# We can observe very minor change is with dropping the feature and without droppi
```

Ridge Regression

In [57]:
```
from sklearn.linear_model import Ridge
# hyperparameters : alpha = (0.00001-1), max_iter = higher if sample size is small
ridge_model = Ridge(alpha = 0.01, max_iter = 100000)
```

In [58]:
```
ridge_model.fit(x_train.scaler,y_train)
```

Out[58]: Ridge(alpha=0.01, max_iter=100000)

In [59]:
```
ridge_model.score(x_test.scaler,y_test)
```

Out[59]:
```
0.7044084249552689
```

Ridge Regression Metrics

In [60]:
```python
# making new Predictions (yhat)
#yhat = ridge_model.predict(x_test_scaler)

#yhat = ridge_model.predict(x_test_scaler)
```

In [61]:
```python
# r2-score , mean_squared_error, mean_absolute_error
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
# coefficient of determination (0 - 1)
r2_score(y_test, yhat)
```

Out[61]:
```
0.7049161200498151
```

In [62]:
```python
mean_absolute_error(y_test, yhat)
```

Out[62]:
```
102494.56640321515
```

In [63]:
```python
# RMSE
np.sqrt(mean_squared_error(y_test, yhat))
```

Out[63]:
```
147902.42875551208
```

Elastic net- Combination of both Lasso and Ridge model

In [64]:
```python
from sklearn.linear_model import ElasticNet
elasticNet_model = ElasticNet(alpha = 0.004, max_iter=100000)
```

In [65]:
```python
elasticNet_model.fit(x_train_scaler, y_train)
```

Out[65]:
```
ElasticNet(alpha=0.004, max_iter=100000)
```

In [66]:
```python
elasticNet_model.score(x_test_scaler, y_test)
```

Out[66]:
```
0.6993882344502862
```

In [ ]: