In [1]:
```
import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
from matplotlib import pyplot
import seaborn as sns
from sklearn.decomposition import PCA
from sklearn.preprocessing import scale
from statsmodels.tsa.arima_model import ARIMA
from datetime import datetime
```

In [2]:
```
#insert data from csv
df = pd.read_csv('/Users/amrita/Desktop/zillow data.csv')
```

In [41]:
```
df.head()
```

Out[41]:

	RegionID	SizeRank	RegionName	RegionType	StateName	1/31/00	2/29/00	3/31/00	4/30/
0	102001	0	United States	Country	NaN	127104.0	127448.0	127809.0	12854
1	394913	1	New York, NY	Msa	NY	223875.0	225213.0	226416.0	22878
2	753899	2	Los Angeles-Long Beach-Anaheim, CA	Msa	CA	231151.0	231956.0	233189.0	23553
3	394463	3	Chicago, IL	Msa	IL	169017.0	169416.0	169932.0	17096
4	394514	4	Dallas-Fort Worth, TX	Msa	TX	130276.0	130380.0	130466.0	13067

5 rows × 268 columns

In [42]:
```
#transpose data
ndf= df.melt(id_vars=["RegionID", "SizeRank","RegionName","RegionType","Sta
        var_name="Date",
        value_name="Price")
```

In [43]:
```
ndf.head()
```

Out[43]:

	RegionID	SizeRank	RegionName	RegionType	StateName	Date	Price
0	102001	0	United States	Country	NaN	1/31/00	127104.0
1	394913	1	New York, NY	Msa	NY	1/31/00	223875.0
2	753899	2	Los Angeles-Long Beach-Anaheim, CA	Msa	CA	1/31/00	231151.0
3	394463	3	Chicago, IL	Msa	IL	1/31/00	169017.0
4	394514	4	Dallas-Fort Worth, TX	Msa	TX	1/31/00	130276.0

In [44]:

```
#summary of the datatype
ndf.dtypes
```

Out[44]:

```
RegionID         int64
SizeRank         int64
RegionName      object
RegionType      object
StateName       object
Date            object
Price          float64
dtype: object
```

In [45]:

```
#no. of missing values by column
ndf.isna().sum()
```

Out[45]:

```
RegionID           0
SizeRank           0
RegionName         0
RegionType         0
StateName        263
Date               0
Price          48727
dtype: int64
```

In [8]:

```
#Checking for the total count of Region name=United States

i= ndf[ndf['RegionName']=='United States']
i.shape
```

Out[8]:

```
(263, 7)
```

In [46]:

```
i.head()
```

Out[46]:

	RegionID	SizeRank	RegionName	RegionType	StateName	Date	Price
0	102001	0	United States	Country	NaN	1/31/00	127104.0
908	102001	0	United States	Country	NaN	2/29/00	127448.0
1816	102001	0	United States	Country	NaN	3/31/00	127809.0
2724	102001	0	United States	Country	NaN	4/30/00	128546.0
3632	102001	0	United States	Country	NaN	5/31/00	129288.0

In [47]:

```
#Dropping all rows with Region Name=United States and creating a new datafr

df_new= ndf[ndf.RegionName != 'United States']
df_new.head()
```

Out[47]:

	RegionID	SizeRank	RegionName	RegionType	StateName	Date	Price
1	394913	1	New York, NY	Msa	NY	1/31/00	223875.0
2	753899	2	Los Angeles-Long Beach-Anaheim, CA	Msa	CA	1/31/00	231151.0
3	394463	3	Chicago, IL	Msa	IL	1/31/00	169017.0
4	394514	4	Dallas-Fort Worth, TX	Msa	TX	1/31/00	130276.0
5	394974	5	Philadelphia, PA	Msa	PA	1/31/00	129615.0

In [69]:

```
df_new.describe()
```

Out[69]:

	RegionID	SizeRank	Price
count	238541.000000	238541.000000	1.898140e+05
mean	415361.502756	458.604190	1.612216e+05
std	83488.890005	267.525459	1.038787e+05
min	394297.000000	1.000000	2.848100e+04
25%	394548.000000	227.000000	9.925225e+04
50%	394804.000000	455.000000	1.320360e+05
75%	395050.000000	687.000000	1.877395e+05
max	753929.000000	933.000000	1.506129e+06

In [78]:

```
df_new.(RegionName=='New York, NY').Date.unique()
```

```
  File "<ipython-input-78-9002fa9f8b66>", line 1
    df_new.(RegionName=='New York, NY').Date.unique()
           ^
SyntaxError: invalid syntax
```

In [11]:

```
df_new.shape
```

Out[11]:

```
(238541, 7)
```

In [12]:
```
df_new.dtypes
```

Out[12]:
```
RegionID        int64
SizeRank        int64
RegionName     object
RegionType     object
StateName      object
Date           object
Price         float64
dtype: object
```

In [48]:
```
#Changing date from object to datetime

df_new['Date'] = pd.to_datetime(df_new['Date'])
df_new.head()
```

```
/Users/amrita/opt/anaconda3/lib/python3.7/site-packages/ipykernel_launcher.py:3: 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 (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  This is separate from the ipykernel package so we can avoid doing imports until
```

Out[48]:

	RegionID	SizeRank	RegionName	RegionType	StateName	Date	Price
1	394913	1	New York, NY	Msa	NY	2000-01-31	223875.0
2	753899	2	Los Angeles-Long Beach-Anaheim, CA	Msa	CA	2000-01-31	231151.0
3	394463	3	Chicago, IL	Msa	IL	2000-01-31	169017.0
4	394514	4	Dallas-Fort Worth, TX	Msa	TX	2000-01-31	130276.0
5	394974	5	Philadelphia, PA	Msa	PA	2000-01-31	129615.0

In [49]:
```
df_new.dtypes
```

Out[49]:
```
RegionID                int64
SizeRank                int64
RegionName             object
RegionType             object
StateName              object
Date           datetime64[ns]
Price                 float64
dtype: object
```

In [50]:

```
#no. of missing values by column in the new dataset

df_new.isna().sum()
```

Out[50]:

```
RegionID          0
SizeRank          0
RegionName        0
RegionType        0
StateName         0
Date              0
Price         48727
dtype: int64
```

In [51]:

```
#percent missing values by each column

percent_missing = df_new.isnull().sum() * 100 / len(df_new)
missing_value_df = pd.DataFrame({'column_name': df_new.columns,
                                 'percent_missing': percent_missing})
missing_value_df.sort_values('percent_missing', inplace=True)
print(missing_value_df)
```

```
           column_name  percent_missing
RegionID      RegionID         0.000000
SizeRank      SizeRank         0.000000
RegionName  RegionName         0.000000
RegionType  RegionType         0.000000
StateName    StateName         0.000000
Date              Date         0.000000
Price            Price        20.427096
```

In [52]:

```
#impute missing values using interpolation method

interpolated = df_new.interpolate(method='linear')
interpolated.head()
```

Out[52]:

	RegionID	SizeRank	RegionName	RegionType	StateName	Date	Price
1	394913	1	New York, NY	Msa	NY	2000-01-31	223875.0
2	753899	2	Los Angeles-Long Beach-Anaheim, CA	Msa	CA	2000-01-31	231151.0
3	394463	3	Chicago, IL	Msa	IL	2000-01-31	169017.0
4	394514	4	Dallas-Fort Worth, TX	Msa	TX	2000-01-31	130276.0
5	394974	5	Philadelphia, PA	Msa	PA	2000-01-31	129615.0

```
In [54]: #no missing values

         interpolated.isna().sum()
```

```
Out[54]: RegionID      0
         SizeRank      0
         RegionName    0
         RegionType    0
         StateName     0
         Date          0
         Price         0
         dtype: int64
```

```
In [55]: #percent missing values by each column

         percentage_missing = interpolated.isnull().sum() * 100 / len(interpolated)
         missing_value = pd.DataFrame({'column_name': interpolated.columns,
                                          'percent_missing': percentage_missing})
         missing_value.sort_values('percent_missing', inplace=True)
         print(missing_value)
```

```
                     column_name  percent_missing
         RegionID      RegionID              0.0
         SizeRank      SizeRank              0.0
         RegionName  RegionName              0.0
         RegionType  RegionType              0.0
         StateName    StateName              0.0
         Date              Date              0.0
         Price            Price              0.0
```

```
In [56]: # total unique region names

         interpolated. RegionName. nunique()
```

```
Out[56]: 907
```

```
In [57]: interpolated['RegionName'].value_counts().head(100)
```

```
Out[57]: Albemarle, NC       263
         Hereford, TX        263
         Scottsbluff, NE     263
         Red Bluff, CA       263
         Indianapolis, IN    263
                            ...
         Fort Morgan, CO     263
         Worcester, MA       263
         Menomonie, WI       263
         Ocean City, NJ      263
         Eau Claire, WI      263
         Name: RegionName, Length: 100, dtype: int64
```

In [66]:

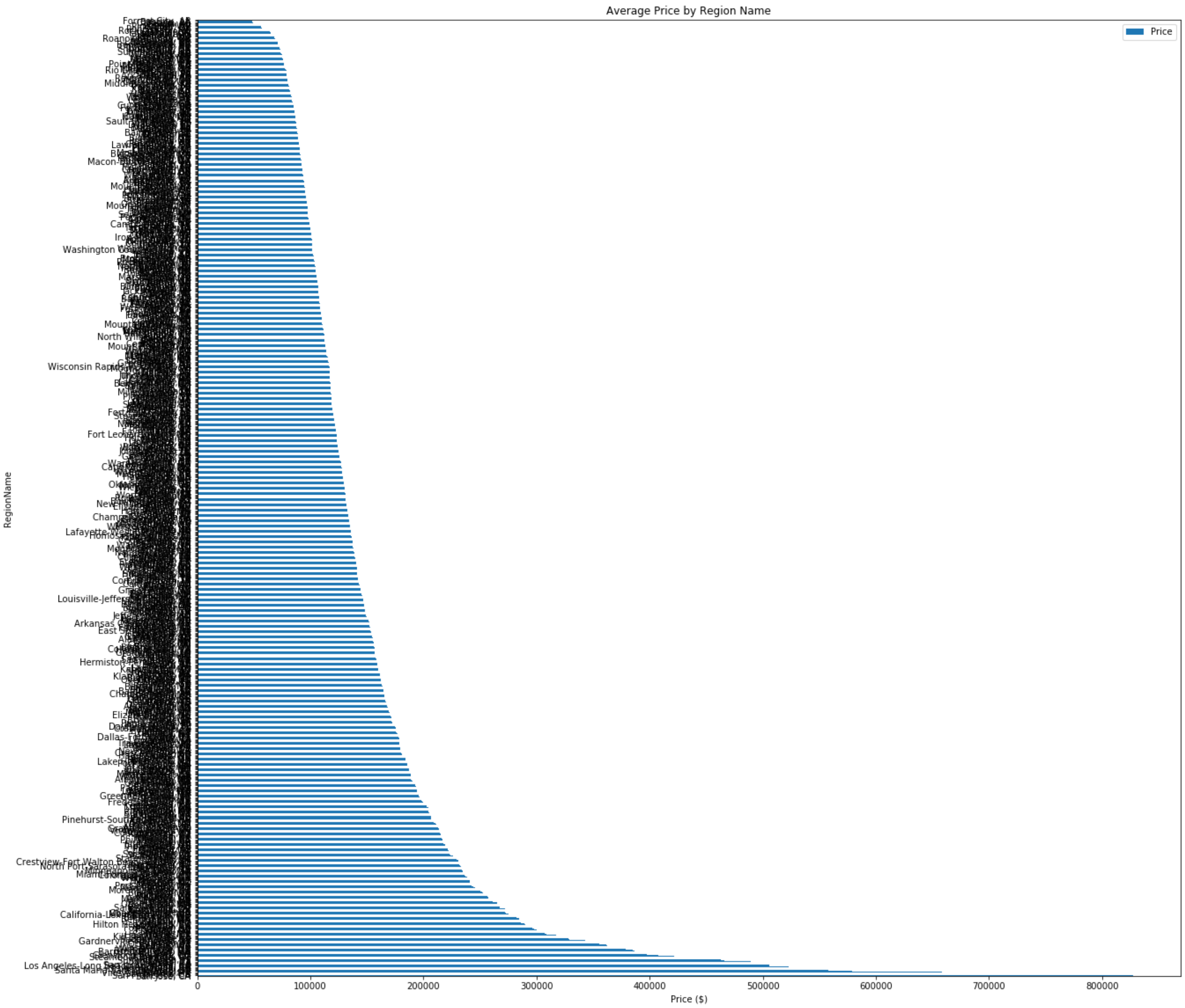
```
#Distribution of price by RegionName
region_price_means = interpolated.groupby("RegionName")[['Price']].mean()
region_price_means.head(90)
```

Out[66]:

```
                          Price
        RegionName
     Aberdeen, SD 131921.124585
     Aberdeen, WA 194096.368061
      Abilene, TX 134786.818931
          Ada, OK  80116.889734
       Adrian, MI 133173.357414
              ...           ...
Big Stone Gap, VA  90701.484791
      Billings, MT 189141.785171
   Binghamton, NY 106751.771863
   Birmingham, AL 147310.996198
     Bismarck, ND 184398.741445

90 rows × 1 columns
```

```
In [24]: (state_price_means.reindex(index=state_price_means.mean(axis=1)
             .sort_values(ascending=False)
             .index)
             .plot(kind='barh', figsize=(20, 20), title='Average Price by Region Nam
         plt.xlabel('Price ($)');
```



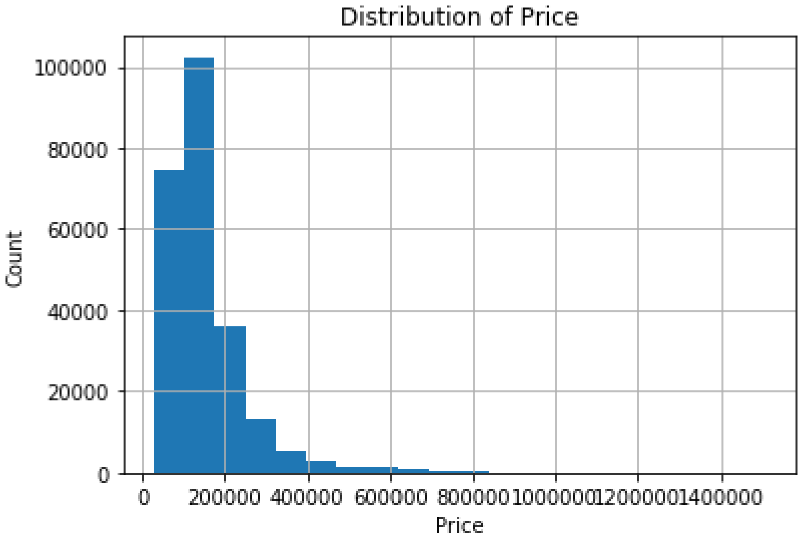
In [59]:
```
# statistical summary of all the numerical columns

interpolated.describe().T
```

Out[59]:

	count	mean	std	min	25%	50%	75%	ma
RegionID	238541.0	415361.502756	83488.890005	394297.0	394548.0	394804.0	395050.0	753929.
SizeRank	238541.0	458.604190	267.525459	1.0	227.0	455.0	687.0	933.
Price	238541.0	154240.877113	98655.340633	28481.0	95525.0	126544.0	179056.0	1506129.

In [60]:
```
interpolated.Price.hist(bins=20)
plt.xlabel('Price')
plt.ylabel('Count')
plt.title('Distribution of Price');
```



In [61]:
```
interpolated['log_base10'] = np.log10(interpolated['Price'])
interpolated.head()
```

Out[61]:

	RegionID	SizeRank	RegionName	RegionType	StateName	Date	Price	log_base10
1	394913	1	New York, NY	Msa	NY	2000-01-31	223875.0	5.350006
2	753899	2	Los Angeles-Long Beach-Anaheim, CA	Msa	CA	2000-01-31	231151.0	5.363896
3	394463	3	Chicago, IL	Msa	IL	2000-01-31	169017.0	5.227930
4	394514	4	Dallas-Fort Worth, TX	Msa	TX	2000-01-31	130276.0	5.114864
5	394974	5	Philadelphia, PA	Msa	PA	2000-01-31	129615.0	5.112655

In [62]:

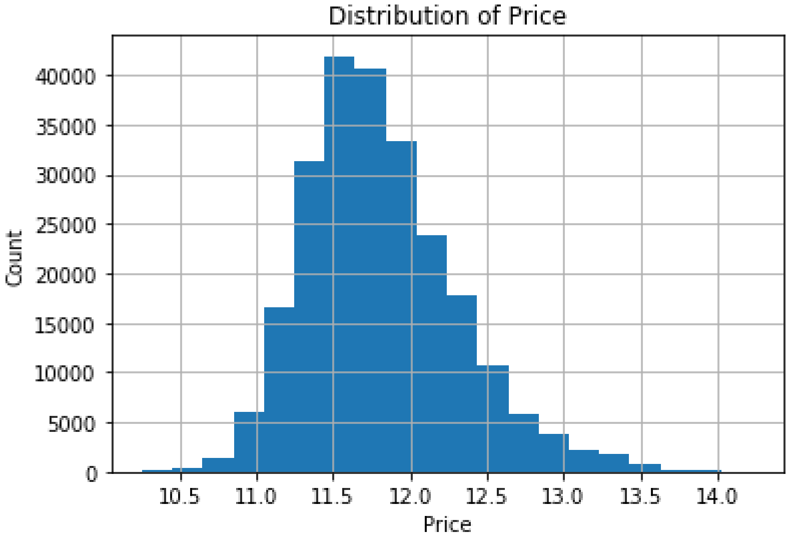
```
#taking natural log to achieve a normal distribution
interpolated['natural_log'] = np.log(interpolated['Price'])
interpolated.head()
```

Out[62]:

	RegionID	SizeRank	RegionName	RegionType	StateName	Date	Price	log_base10	natural_
1	394913	1	New York, NY	Msa	NY	2000-01-31	223875.0	5.350006	12.318
2	753899	2	Los Angeles-Long Beach-Anaheim, CA	Msa	CA	2000-01-31	231151.0	5.363896	12.350
3	394463	3	Chicago, IL	Msa	IL	2000-01-31	169017.0	5.227930	12.037
4	394514	4	Dallas-Fort Worth, TX	Msa	TX	2000-01-31	130276.0	5.114864	11.777
5	394974	5	Philadelphia, PA	Msa	PA	2000-01-31	129615.0	5.112655	11.772

In [63]:

```
interpolated.natural_log.hist(bins=20)
plt.xlabel('Price')
plt.ylabel('Count')
plt.title('Distribution of Price');
```



In [64]:

```
interpolated.drop('log_base10', axis=1, inplace=True)
```

In [65]:

```
interpolated.head()
```

Out[65]:

	RegionID	SizeRank	RegionName	RegionType	StateName	Date	Price	natural_log
1	394913	1	New York, NY	Msa	NY	2000-01-31	223875.0	12.318843
2	753899	2	Los Angeles-Long Beach-Anaheim, CA	Msa	CA	2000-01-31	231151.0	12.350826
3	394463	3	Chicago, IL	Msa	IL	2000-01-31	169017.0	12.037755
4	394514	4	Dallas-Fort Worth, TX	Msa	TX	2000-01-31	130276.0	11.777411
5	394974	5	Philadelphia, PA	Msa	PA	2000-01-31	129615.0	11.772324

In [67]:

```
interpolated.shape
```

Out[67]:

```
(238541, 8)
```

In [68]:

```
interpolated.to_excel('output1.xlsx', engine='xlsxwriter')
```