Visualization - Aided Exploration of Real Estate of Northeastern Region of USA

Preface:

I have heard bunch of variations like, " Computers hate me" or " I don't do any CS stuffs". As of someone who did not understand programming before enrolling into the Computer Science major, this does sound familiar. Over time taking Honors Seminar (Music and Arts) and 2D Design (Arts), helped me understand that human beings are visual creatures and process images at an alarming rate. Combining both these aspects of learning - tech and images was challenging and way out of my comfort (Especially while adjusting to different language - Python). I believe crunching down data and findings in a digestible way i.e. images can help any fellow Honors students to not fear using technical aspects in their project, no matter what their majors are. With that, I have tried using simpler explanation so that anyone can follow along.

In this project, I will be visualizing the real estate data of Northeastern region of the United States. Visualized data helps in analyzing and exploring data that can help identify the areas of improvement within the set of data. Real estate in Northeastern region of the States are affected by myriad of factors, like commercial capitals (New York, Boston), government regulations, demographics of the area, financial stability and many more. Data used for this project is broken down at the state level of the Northeastern region.

First, the modules and library of the Python is imported and date sets are read. For different functions, different libraries are used, whose application is commented in the code below.

In [1]:
```
import numpy as np ## for linear algebra
import datetime # for cleaning data
import pandas as pd ## for processing data, csv files I/O
import seaborn as sns ## for statistical graphics
import matplotlib.pyplot as plt ## for manipulating elements of a figure, creating plotting area, lines etc

import bar_chart_race as bcr # visual way of displaying data changing over time

import warnings
warnings.filterwarnings("ignore") # suppresing Python warning
```

Data used below has been collected from Kaggle ranging from year 2016 August to 2021 August. That way there will not be incomplete year count and will be relevant to the current time. Generally, the first half of the year is the market for seller. Source of data: https://www.realtor.com/research/data/

After the data has been read, data is cleaned and formatted. Only the northeastern states are extracted out from the dataset for visualization.

In [2]:
```
df_kaggle = pd.read_csv("MM_DataNEUSA_Sales.csv")
print ("In this dataset, we have", df_kaggle.shape[0], " records and", df_kaggle.shape[1],  " columns.")

#Data Cleaning : Formating date and renaming state column as Northeastern States

df_kaggle['month_date_yyyymm'] = pd.to_datetime(df_kaggle['month_date_yyyymm'],format = '%Y%m')
df_kaggle.rename({"month_date_yyyymm":"Date", "state": "Northeastern_states" }, axis = 'columns', inplace = True)

#extract only northeastern states
northeastern_states = ["maine","new hampshire","new york","new jersey","pennsylvania","connecticut",
                       "rhode island","vermont","massachusetts"]
df_nestate = df_kaggle.loc[df_kaggle["Northeastern_states"].isin(northeastern_states)] # selecting all rows from given options of states
print ("After filtered dataset to northeastern region, we have", df_nestate.shape[0], " records and ", df_nestate.shape[1],  " columns.")
df_nestate.head(10) ## prints first 10 rows of filtered dataframe
```

```
In this dataset, we have 3112  records and 39  columns.
After filtered dataset to northeastern region, we have 549  records and  39  columns.
```

Out[2]:

	Date	Northeastern_states	state_id	median_listing_price	median_listing_price_mm	median_listing_price_yy	active_listing_count	active_listing_count_mm	active_listing_count_yy	median_days_on_market	...	median_square_feet_yy	average_listing_price	average_listing_price_mm
2	2021-07-01	vermont	vt	350000.0	-0.0210	0.0888	1692	0.0702	-0.4540	46.0	...	-0.0562	5.383490e+05	-0.0319
7	2021-07-01	pennsylvania	pa	275000.0	-0.0177	0.0599	21342	0.1037	-0.2037	41.0	...	-0.0564	4.012947e+05	-0.0296
13	2021-07-01	rhode island	ri	419900.0	0.0344	-0.0107	1417	0.1400	-0.2811	30.0	...	-0.0962	7.778582e+05	-0.0137
26	2021-07-01	massachusetts	ma	649000.0	-0.0159	0.0469	9209	0.0205	-0.2615	31.0	...	-0.0911	1.182808e+06	-0.0149
28	2021-07-01	maine	me	339900.0	0.0034	0.1368	3536	0.1802	-0.3571	33.0	...	-0.0070	5.076741e+05	0.0029
32	2021-07-01	connecticut	ct	489900.0	-0.0092	0.2295	7764	0.0774	-0.5490	36.0	...	0.0021	1.143219e+06	-0.0216
41	2021-07-01	new hampshire	nh	424900.0	0.0130	0.0827	2321	0.1105	-0.2664	28.0	...	-0.0486	5.736333e+05	0.0162
44	2021-07-01	new jersey	nj	445000.0	-0.0089	0.0023	25084	0.0599	-0.1966	36.0	...	-1.0000	6.511915e+05	-0.0212
50	2021-07-01	new york	ny	560000.0	-0.0300	0.0626	57328	0.0123	-0.1335	73.0	...	-0.1042	1.215842e+06	-0.0419
62	2021-06-01	maine	me	338750.0	-0.0007	0.1701	2996	0.2249	-0.4651	28.0	...	-0.0047	5.062248e+05	-0.0132

10 rows × 39 columns

In [3]:
```
#df_nestate.describe().T #describes the statistics of the dataframe, transpose for better viewing
```

By removing the hashtags(##) or uncommenting the code, we can see what each column of data displays. Some valuable information like the count of values, mean, standard deviation, min/max and quartiles of each columns are interpreted.

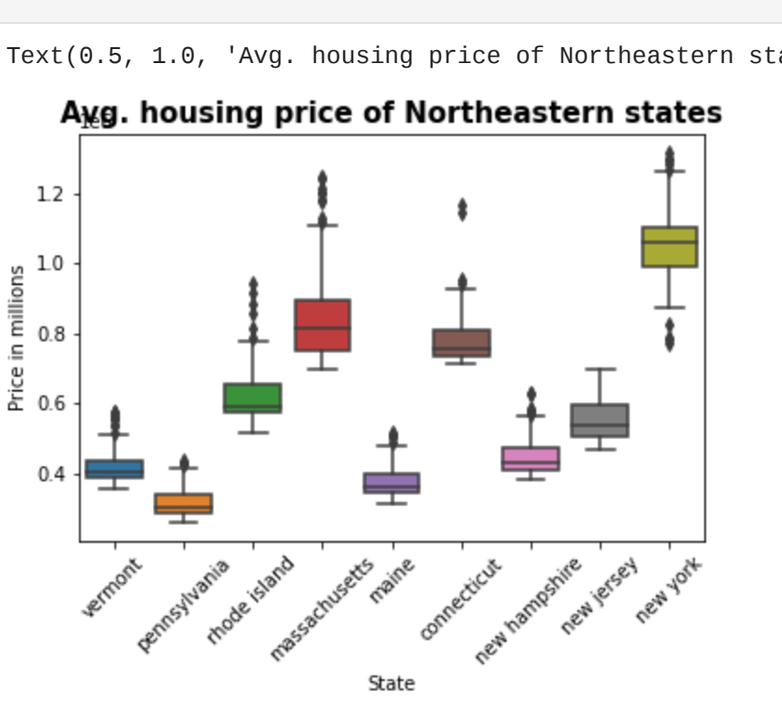
In [4]:
```
#sorting the data by state and date to simplify visualization
df_sort = df_nestate.sort_values (["Northeastern_states","Date"],axis = 0, ascending = [True, True]).reset_index(drop = True)
```

BoxPlot Diagram for these States

One way to plot boxplot is using seaborn library. Vertical box plot is grouped by categorical variables (States). The code below passes the panda dataframe (our data) into seaborn's boxplot.

In [5]:
```
#Visualization
# compute mean per group and find index after sorting in ascending order
box_plot = sns.boxplot(x = 'Northeastern_states', y = 'average_listing_price', data = df_nestate)
plt.ylabel('Price in millions')
plt.xlabel('State')
plt.xticks(rotation = 45)
plt.title("Avg. housing price of Northeastern states", fontsize = 15, fontweight = 'bold')
```

Out[5]:
```
Text(0.5, 1.0, 'Avg. housing price of Northeastern states')
```



Let's take example of red box plot that represents Massachusetts. The bottom black horizontal line is minimum housing price value (somewhere between 0.6 - 0.8) and top black horizontal line (between 1.0 - 1.2) is maximum value for that red box. The first black horizontal line of the rectangular red box plot is first quartile (25%), the line in between the box plot is second quartile (50%) and third black line of the box represents the third quartile (75%). The diamond shaped value present at the top of the horizontal line (in this case) is outlier data or erroneous data.

Bar Chart Race of housing price (2016-2021)

Bar Chart Race graph is a comprehensible representation of dataset over time. Below the average housing price of the real estate from mid 2016 to mid 2021 is viualized in an animated bar chart.

In [6]:
```
df_race = df_sort[['Date','Northeastern_states','average_listing_price']]
#create spreadsheet style pivot table. Values display the statistical summary of feature, index is the featured
#passed in values argument. Columns aggregates values according to the features.
df_race=df_race.pivot_table(values = 'average_listing_price',index=['Date'],columns = 'Northeastern_states')
df_race
```

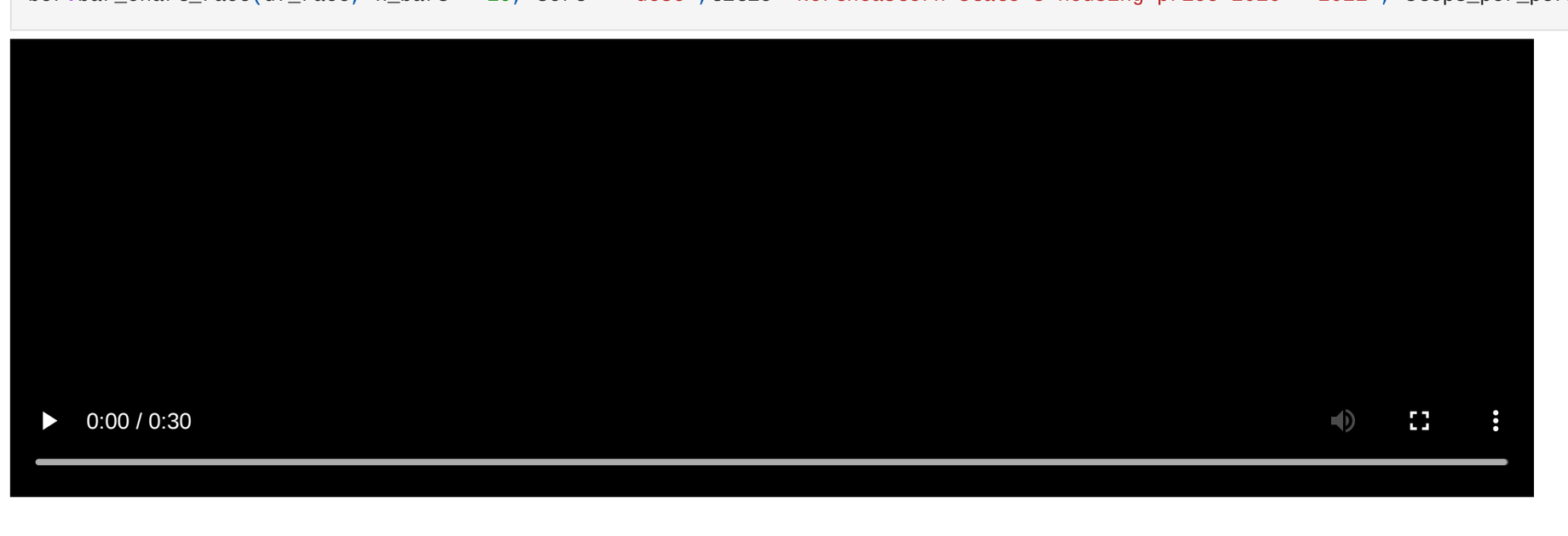
Out[6]:

Northeastern_states	connecticut	maine	massachusetts	new hampshire	new jersey	new york	pennsylvania	rhode island	vermont
Date									
2016-07-01	7.598300e+05	331254.6000	6.964768e+05	388414.6333	482513.6020	7.862448e+05	272366.6627	539143.5216	358648.3840
2016-08-01	7.506584e+05	332513.9006	6.950805e+05	384805.7158	478760.4428	7.734834e+05	270407.8052	535634.2990	358848.4227
2016-09-01	7.565740e+05	330161.7855	6.952674e+05	387083.3655	480283.8117	7.847256e+05	271526.0676	524995.7656	360148.8716
2016-10-01	7.657705e+05	328398.3357	6.980840e+05	401398.3458	482912.0250	8.247852e+05	271621.7676	517423.0132	363225.2592
2016-11-01	7.722720e+05	328159.4010	7.037814e+05	393724.1198	483809.9124	8.744485e+05	270294.0491	522212.3088	369521.7021
...	...	...	...	...	...	...	...	...	...
2021-03-01	9.436239e+05	489828.8378	1.212968e+06	629367.8929	695102.2495	1.295947e+06	432894.3086	917678.3864	575378.2998
2021-04-01	9.165335e+05	515714.0586	1.248178e+06	587768.8035	691526.6734	1.315247e+06	438361.1749	882530.8173	577795.3699
2021-05-01	8.984697e+05	512987.3666	1.239836e+06	571768.7245	680342.4981	1.297325e+06	426755.0953	813859.0833	558232.2729
2021-06-01	1.168469e+06	506224.8076	1.200666e+06	564468.8900	665323.9338	1.269033e+06	413517.8805	788673.6553	556068.4730
2021-07-01	1.143219e+06	507674.1075	1.182808e+06	573633.2588	651191.5334	1.215842e+06	401294.6952	777858.2086	538348.9867

61 rows × 9 columns

In [7]:
```
bcr.bar_chart_race(df_race, n_bars = 10, sort = 'desc',title="Northeastern state's housing price 2016 - 2021", steps_per_period = 20, figsize = (6,2) )
```

Out[7]:



Models

We will now be observing the dataset which was manually populated in Excel. Majority of data were collected from US Census, followed by Zillow, Wikipedia. 21 features like median listing price, Home Value Index, housing units are populated according to states.

In [8]:
```
#Read in data
df_reg = pd.read_csv(r"Analysis_Test.csv")
df_reg.fillna(df_reg.median(), inplace=True) # data wranging, filling NA values with median of data column
df_reg
```

Out[8]:

	States	Crime rate per 100,000 population	ZHVI (Zillow Home Value Index)	Total housing Units	Median Income	median_listing_price	active_listing_count	new_listing_count	price_increased_count	price_reduced_count	...	median_square_feet	average_listing_price	total_listing_count	pending_ratio	nielsen_hh_rank	hotness_score	supply_score
0	Connecticut	181.6	247706	1491786	79043	385000.0	11930.0	2860.0	136.0	1240.0	...	2016.0	847461.2371	12088.0	0.013244	54	64.806595	83.069119
1	Connecticut	184.6	244109	1524992	88380	353688.0	17249.0	2112.0	68.0	1828.0	...	2083.5	724776.1123	17340.0	0.005276	337	31.261890	32.022828
2	Connecticut	209.6	244017	1521019	72812	339900.0	20677.0	2848.0	84.0	2336.0	...	1888.0	732724.4979	20747.0	0.003385	60	59.321497	81.420419
3	Connecticut	229.2	243569	1517251	74304	339900.0	19895.0	2644.0	116.0	2216.0	...	1579.0	715891.4913	20147.0	0.012666	374	51.870640	46.290425
4	Connecticut	227.7	242921	1512528	75923	339900.0	21450.0	3252.0	92.0	2320.0	...	1976.0	768576.4007	25152.0	0.172587	450	73.208624	72.669626
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
94	Vermont	102.6	227102	328991	60708	339450.0	14294.0	2644.0	92.0	1828.0	...	1677.0	522433.7284	16375.0	0.205254	387	56.436271	33.481294
95	Vermont	114.9	227340	327315	54842	339450.0	14294.0	2644.0	92.0	1828.0	...	1677.0	522433.7284	16375.0	0.205254	823	14.838301	1.775523
96	Vermont	142.3	227048	327315	55582	339450.0	14294.0	2644.0	92.0	1828.0	...	1677.0	522433.7284	16375.0	0.205254	1295	31.388713	1.965758
97	Vermont	135.2	227889	324434	51862	339450.0	14294.0	2644.0	92.0	1828.0	...	1677.0	522433.7284	16375.0	0.205254	801	19.942930	5.326569
98	Vermont	130.2	229681	322975	55928	339450.0	14294.0	2644.0	92.0	1828.0	...	1677.0	522433.7284	16375.0	0.205254	958	16.930881	9.194673

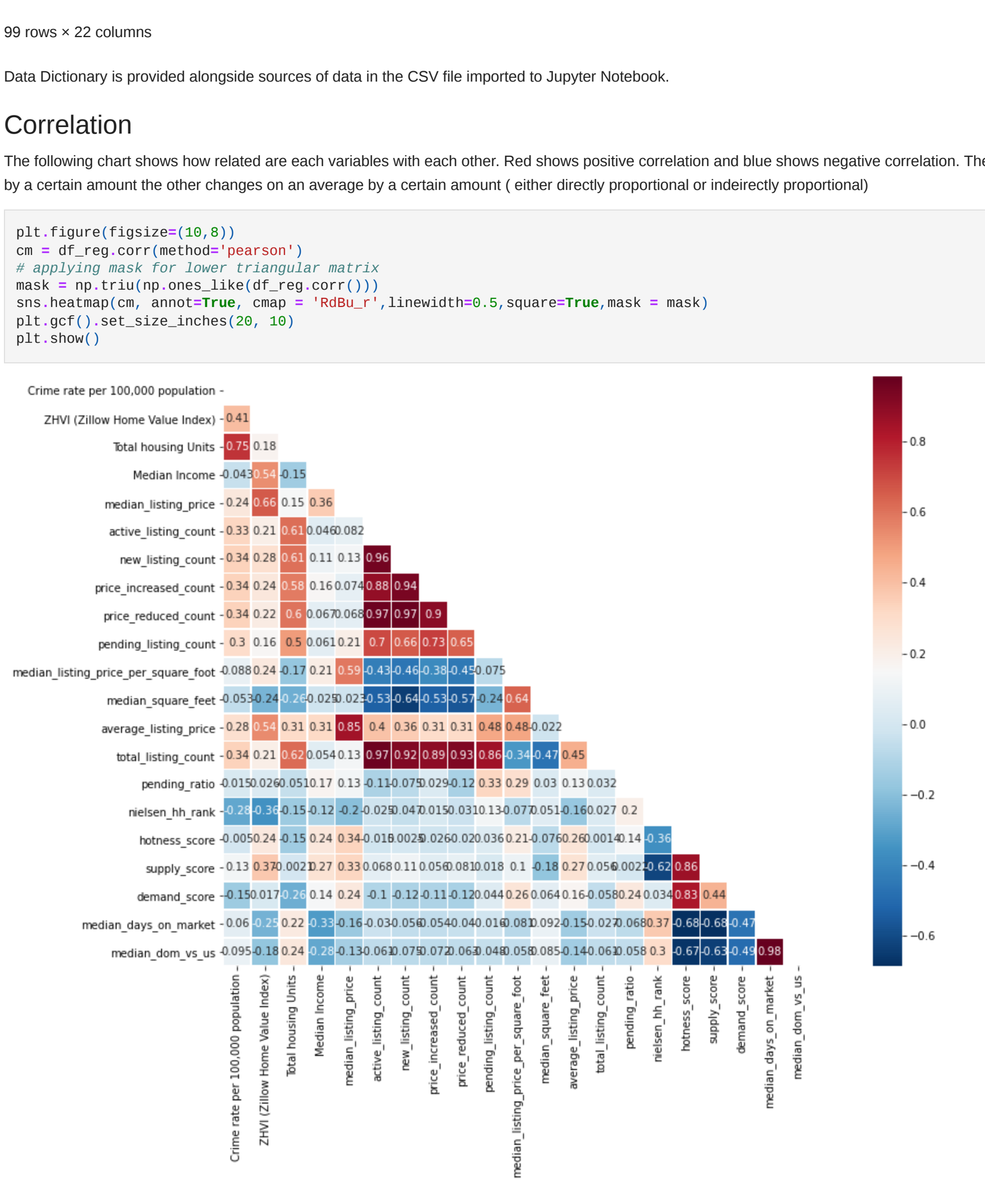
99 rows × 22 columns

Data Dictionary is provided alongside sources of data in the CSV file imported to Jupyter Notebook.

Correlation

The following chart shows how related are each variables with each other. Red shows positive correlation and blue shows negative correlation. The darker the shade of color, the stronger the relation between these variables. If the two variables are related it means that when one changes by a certain amount the other changes on an average by a certain amount (either directly proportional or indeierctly proportional)

In [9]:
```
plt.figure(figsize=(10,8))
cm = df_reg.corr(method='pearson')
# applying mask for lower triangular matrix
mask = np.triu(np.ones_like(df_reg.corr()))
sns.heatmap(cm, annot=True, cmap = 'RdBu_r',linewidth=0.5,square=True,mask = mask)
plt.gcf().set_size_inches(20, 10)
plt.show()
```



Multiclass logistic regression (Softmax regression)

As my target model is States, where there are more than 2 number of classes (i.e. 9 states) , I am using Softmax Regression to handle muliple states. The objective is to classify all features like median housing price, average listing price into one of ninth States. To understand the performance of the model, the dataset is divided into training and test set and splitted in a ratio of 80:20, meaning 80% of data is used for model training and the rest is for model testing. This way, we can predict how accurate is our data.

In [10]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [11]:
```
# Define the features and target (X and y)
pd.unique(df_reg["States"]) # target class

X = df_reg.drop(["States"], axis = 1).astype(float)

#creating target vector and converting to numeric codes (for visualization)
y = df_reg["States"]
y_numeric_dict={'Connecticut':1,'Maine':2,'Massachusetts':3, 'New Hampshire': 4,
               'New Jersey' : 5, 'New York': 6, 'Pennsylvania': 7, 'Rhode Island':8,
               'Vermont': 9}
y = y.apply(lambda x: y_numeric_dict[x])
```

We are going to measure the quality of our model predictions. First, we create k-nearest-neighbors model to classify data. K-nearest neighbors estimates the likelihood of data point based on what group the data points is nearest to.

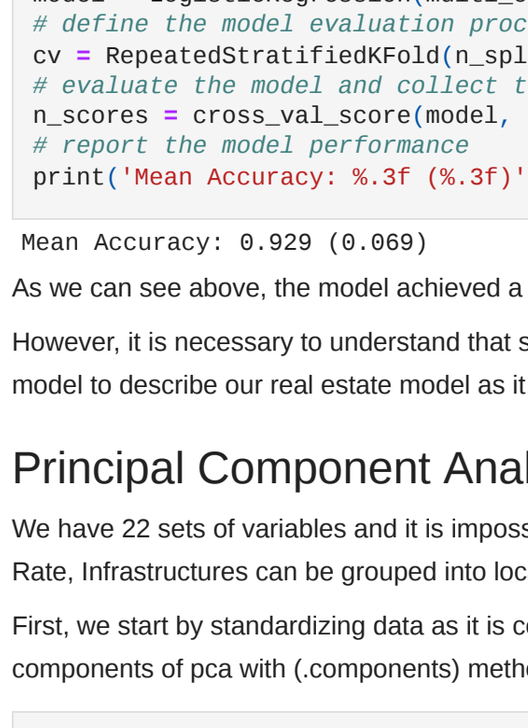
In [12]:
```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import KFold
from sklearn.metrics import plot_confusion_matrix
# Set up the model we will use
# general rule of thumb for n_neighbors : square root of samples (eg:79 for this project)
knn_model = KNeighborsClassifier(n_neighbors=10)
```

The function run_model splits data into training and test set. It generates predictions on the test set and train training set to measure the accuracy of our model.

In [13]:
```
def run_model(X,y,model):
    '''
    Train model and generate test set predictions
    Args:
        X: input feature matrix
        y: target array
        model: instantiated model
    Returns:
        X_test: test set inputs
        y_test: test set targets
        preds: predictions on test set
    '''
    # Split into training and test sets, using 20% of our data for a test set
    X_train,X_test,y_train,y_test = train_test_split(X, y, random_state=0,test_size=0.20)
    model.fit(X_train,y_train)
    preds = model.predict(X_test)

    return X_test, y_test, preds

    raise NotImplementedError()
```

In [14]:
```
# Test cell for 'run_model'
X_test, y_test, preds = run_model(X,y,knn_model)

print('Confusion matrix for our model:')
plot_confusion_matrix(knn_model, X_test, y_test,cmap=plt.cm.Blues,normalize=None)
plt.show()
```

```
Confusion matrix for our model:
```



The model shows that Massachusetts (labeled as 3) data is mostly favored to determine the numerical relationship between set of variables.

Testing model accuracy

Logistic Regression is supported by the scikit-learn library from LogisticRegression class. Using repeated stratified kfold, we evaluate the classification models.The stratification ensures that each cross-validation fold has approximately the same distribution of examples in each class as the whole training dataset.Repeated k-fold cross-validation provides a way to reduce the error in the estimate of mean model performance. We will use three repeats with 10 folds, which is a good default, and evaluate model performance using classification accuracy given that the classes are balanced.

In [15]:
```
from sklearn.datasets import make_classification
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import RepeatedStratifiedKFold

model = LogisticRegression(multi_class='multinomial', solver='lbfgs')
# define the model evaluation procedure
cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
# evaluate the model and collect the scores
n_scores = cross_val_score(model, X, y, scoring='accuracy', cv=cv, n_jobs=-1)
# report the model performance
print('Mean Accuracy: %.3f (%.3f)' % (n_scores.mean(), n_scores.std()))
```

```
Mean Accuracy: 0.929 (0.069)
```

As we can see above, the model achieved a mean classification accuracy of 92.9 % on our real estate dataset. This is an impressive accuracy on the test set.

However, it is necessary to understand that since there is a huge variation of the real estate values within the same geographical region (eg: New York has huge disparity in real house price in regards to its location as stated in our visualization). The Softmax Regression was not the best model to describe our real estate model as it does not give idea on variations of prices of the state rather gives an idea on what states was most favored by our data. This results in biasedness as not all zip codes were taken into considerations but an average or median value of the state.

Principal Component Analysis

We have 22 sets of variables and it is impossible to fit each variables with the listing price of real estate. This is when Principal Componenet Analysis (PCA) comes handy as it reduces the related data together. For eg: Area of room, No of rooms can be grouped into size feature. Crime Rate, Infrastructures can be grouped into location features to reduce dimensionality of data.

First, we start by standardizing data as it is common practice to normalize the data before further application of machine learning. We scale data by fit_transform() on feature data. Each feature will be normally distributed - mean of zero and std devidation of one. We can also access the components of pca with (.components) method.

In [16]:
```
from sklearn.preprocessing import StandardScaler
#fitting the PCA

df_norm = df_reg.drop("States", axis = 1)

df_PCA = StandardScaler().fit_transform(df_norm) # normalizing the features

df_PCA = pd.DataFrame(df_PCA)

from sklearn.decomposition import PCA

pca = PCA(n_components = 4)  # project from 21 to 4 dimensions

pca.fit(df_PCA)

#Accessing the components
pca.components_
```

Out[16]:
```
array([[ 1.83978410e-01,  1.54260863e-01,  2.66303792e-01,
         5.98240902e-02,  1.03087678e-01,  3.62521104e-01,
         3.69726953e-01,  3.52691427e-01,  3.59405929e-01,
         2.92397809e-01, -1.23612317e-01, -2.23163508e-01,
         1.93136928e-01,  3.64561577e-01, -2.47936053e-05,
        -4.94753779e-02,  3.54905839e-02,  7.97936196e-02,
        -2.42335583e-02, -4.39818693e-02, -4.65495100e-02],
       [-6.63891596e-03, -2.25715490e-01,  1.15467479e-01,
        -2.23033592e-01, -2.67646721e-01,  7.51042544e-02,
         6.04975419e-02,  6.57173421e-02,  7.85795628e-02,
         1.72169962e-02, -2.30651615e-01, -4.06485988e-02,
        -2.09758546e-01,  6.03790218e-02, -8.87477486e-02,
         2.11954285e-01, -3.99434118e-01, -3.71355081e-01,
        -3.02956713e-01,  3.61754026e-01,  3.44685251e-01],
       [ 2.56561183e-01,  2.44194226e-01,  1.78033703e-01,
         9.40170173e-02,  3.89139106e-01, -7.36918653e-02,
        -8.45402242e-02, -6.22582772e-02, -9.60698899e-02,
         9.70052594e-02,  4.34617139e-01,  2.55150921e-01,
         3.60801468e-01, -1.94213278e-02,  1.36949620e-01,
         2.89667234e-02, -1.75234188e-01, -1.64383466e-01,
        -1.31304684e-01,  2.79717452e-01,  3.00221741e-01],
       [-3.09681672e-01, -2.24221570e-01, -2.04831140e-01,
         6.93012364e-02, -6.20928206e-04,  4.08315394e-02,
         1.25107550e-02,  9.13367413e-02,  1.11944411e-02,
         3.24258546e-01,  1.19297172e-01,  7.31741830e-02,
         8.67800088e-02,  1.42877180e-01,  4.59761790e-01,
         5.27522305e-01,  5.52300368e-02, -2.07633972e-01,
         3.24966735e-01,  7.27496210e-04, -5.16749618e-02]])
```

In [28]:
```
#Apply the transformation and convert the result into a DataFrame.
columns = ["PCA"+str(x) for x in range(1,pca.n_components_+1)]
df_pca = pd.DataFrame(pca.transform(df_PCA), columns=columns)
df_pca.tail()
```

Out[28]:

	PCA1	PCA2	PCA3	PCA4
94	-1.244742	-0.162538	-0.804357	0.587608
95	-1.523454	2.784093	0.630199	0.604719
96	-1.536873	2.347702	0.429860	1.715031
97	-1.453186	2.368779	0.365540	0.571030
98	-1.425499	2.361646	0.351001	0.696570

We will find explained_variance_ratio. Highest variance causing feature will be the first principal component and so on.

In [18]:
```
print('Explained variation per principal component: {}'.format(pca.explained_variance_ratio_))
```

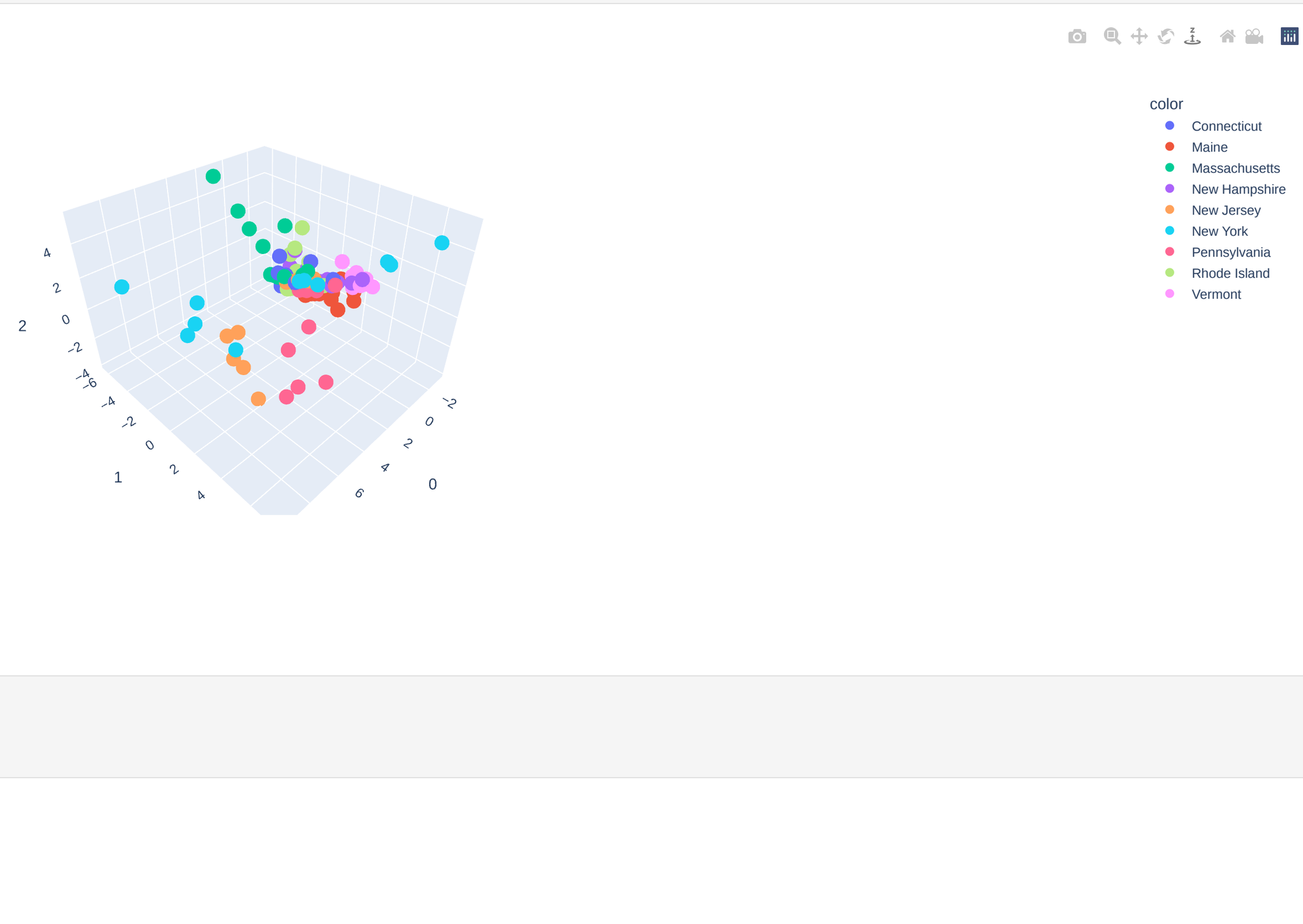
```
Explained variation per principal component: [0.32330413 0.22143267 0.12261058 0.08474567]
```

It shows that principal component 1 holds 32% of information, principal component 2 holds 22% of information, 12% by principal component 3 and 8% by principal component 4. About 25% of data information was lost during the analysis. Now let's visualize the samples along 4 principal components to understand how our samples are distributed among nine classes.

3D Plot of Principal Components

Below figure helps us observe classes (listed in legend) and how they are separated up to some extent.The points belonging to the same class are close to each other, and the points or images that are very different semantically are further away from each other. New York's housing price is spread out compared to any other states.

In [19]:
```
import plotly.express as px

fig = px.scatter_3d(pca.transform(df_PCA), x=0, y=1, z = 2,color=df_reg['States'])
fig.show()
```



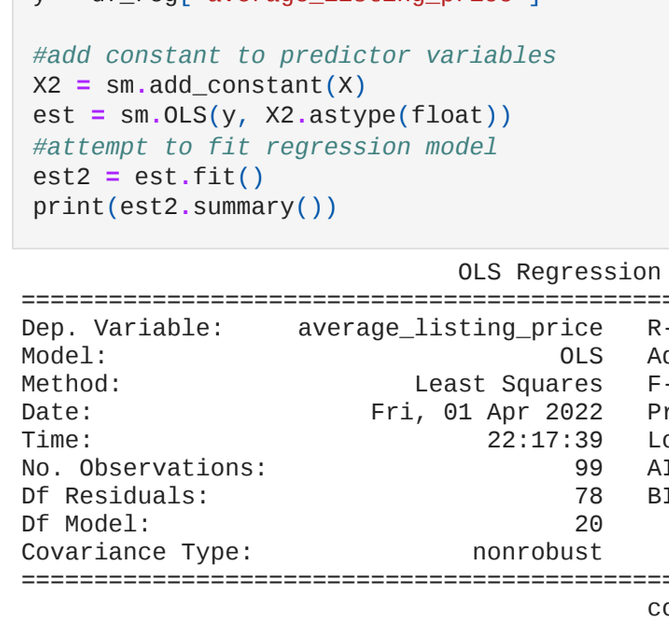
Scree Plot

In [20]:
```
plt.plot(pca.explained_variance_)
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance')
plt.show()
```

The plot shows is an evidence that most of the features are explained by PCA0 and PCA1.

Effect of variables on each components

In [32]:
```
ax = sns.heatmap(pca.components_,
                 cmap='YlGnBu',
                 yticklabels=[ "PCA"+str(x) for x in range(1,pca.n_components_+1)],
                 xticklabels=list(df_PCA.columns),
                 cbar_kws={"orientation": "vertical"})
try:
    ax.set_aspect("equal", figsize(50,30))
except Exception:
    pass
```



The above heatmap shows influence of each PCA components on each features. Features are listed in range of 0 to 20 where 0 stands for States, 1 for Crime rate per 100,000 population and so on.

Linear Regression

A regression line is a straight line that describes how a response variable y (Dependent variable) changes as an explanatory variable x(Independent)changes. This is used to predict the value of y for a given value of x. Here y is average listing price of real estate of each state.

AIC method is used to determine which multiple model is most likely to be the best model of the dataset. I have to import the file after coding in R as Python does not support the functionality. However, I coded some forms in Python, which does not necessarily agree with R and has lesser accuracy than R. I will be sticking with prediction made by R as the result is more reliable.

In [31]:
```
import statsmodels.api as sm

# load data

X = df_reg.drop(["States", "average_listing_price"], axis = 1)
y = df_reg["average_listing_price"]

#add constant to predictor variables
X2 = sm.add_constant(X)
est = sm.OLS(y, X2.astype(float))
#attempt to fit regression model
est2 = est.fit()
print(est2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:     average_listing_price   R-squared:                       0.955
Model:                               OLS   Adj. R-squared:                  0.943
Method:                    Least Squares   F-statistic:                     82.51
Date:                   Fri, 01 Apr 2022   Prob (F-statistic):           3.75e-44
Time:                           22:17:39   Log-Likelihood:                -1176.5
No. Observations:                     99   AIC:                             2395.
Df Residuals:                         78   BIC:                             2449.
Df Model:                             20
Covariance Type:               nonrobust
========================================================================================================
                                          coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------------------------------
const                               -1.946e+05   7.19e+04     -2.707      0.008   -3.38e+05   -5.15e+04
Crime rate per 100,000 population     193.0974     93.671      2.061      0.043       6.612     379.583
ZHVI (Zillow Home Value Index)         -0.5119      0.145     -3.531      0.001      -0.801      -0.223
Total housing Units                    -0.0080      0.004     -2.135      0.036      -0.015      -0.001
Median Income                          -0.4015      0.649     -0.619      0.538      -1.694       0.891
median_listing_price                    1.8584      0.188      9.892      0.000       1.484       2.232
active_listing_count                 2.018e+04   5117.459      3.944      0.000    9996.676    3.04e+04
new_listing_count                      75.7659     17.841      4.247      0.000      40.246     111.285
price_increased_count                -616.7901    144.399     -4.271      0.000    -904.267    -329.314
price_reduced_count                  -114.4701     13.753     -8.323      0.000    -141.851     -87.090
pending_listing_count                2.018e+04   5117.879      3.942      0.000    9986.367    3.04e+04
median_listing_price_per_square_foot  785.4590    281.635      2.789      0.007     224.767    1346.151
median_square_feet                     79.8258     30.304      2.634      0.010      19.495     140.157
total_listing_count                 -2.017e+04   5117.905     -3.942      0.000   -3.04e+04   -9983.878
pending_ratio                       -1.049e+04    1.8e+04     -0.581      0.563   -4.64e+04    2.54e+04
nielsen_hh_rank                         7.3074     18.046      0.405      0.687     -28.620      43.235
hotness_score                        6.436e+09   1.38e+10      0.466      0.643   -2.11e+10    3.39e+10
supply_score                        -3.218e+09   6.91e+09     -0.466      0.643    -1.7e+10    1.05e+10
demand_score                        -3.218e+09   6.91e+09     -0.466      0.643    -1.7e+10    1.05e+10
median_days_on_market                -891.8093    561.038     -1.590      0.116   -2008.750     225.131
median_dom_vs_us                      760.5292    523.908      1.452      0.151    -282.492    1803.551
==============================================================================
Omnibus:                       11.044   Durbin-Watson:                   1.448
Prob(Omnibus):                  0.004   Jarque-Bera (JB):               18.340
Skew:                           0.441   Prob(JB):                     0.000104
Kurtosis:                       4.915   Cond. No.                     1.59e+13
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The smallest eigenvalue is 5.45e-12. This might indicate that there are
strong multicollinearity problems or that the design matrix is singular.
```

We will now drop the variables with p-value higher than 0.05 as they are meaningless variables to our model. That leaves us with : ZHVI,Total housing units, median_listing_price, active_listing_count, new_listing_count, price_increased_count, price_reduced_count, pending_listing_count, median_listing_price_per_square_foot, median_square_feet, total_listing_count. In regression we omits some independent variables that do not have much impact on the dependent variable, the intercept tells the average value of these omitted variables and noise present in model.

R predictions of variables

	Step	Df	Deviance	Resid. Df	Resid. Dev	AIC
1				18	9505845216	916.5830
2	- demand_score	0	0.0	18	9505845216	916.5830
3	- total_listing_count	0	0.0	18	9505845216	916.5830
4	- Median.Income	1	630267.2	19	9506475483	914.5859
5	- Crime.rate.per.100.000.population	1	1420993.6	20	9507896477	912.5927
6	- hotness_score	1	21290918.0	21	9529187395	910.6933
7	- ZHVI..Zillow.Home.Value.Index.	1	59933329.2	22	9589120724	908.9755
8	- pending_ratio	1	95646327.4	23	9684767051	907.4221
9	- active_listing_count	1	127036868.9	24	9811803920	906.0085
10	- median_days_on_market	1	145685925.8	25	9957489846	904.6718
11	- nielsen_hh_rank	1	253892553.3	26	10211382399	903.8048
12	- price_increased_count	1	349805158.9	27	10561187558	903.3205

Similarly, in R 12 data sets were considered valuable attributes to our real estate pricing estimates.

In this way, the project visualized data in an appealing way so that not just people of technology domain understands it. We also predicted some variables like Median Income, ZHVI as essential attributes to average listing price of the real estate of that particular states.