

Big Data - Foundations and Applications

Lesson #13 - Machine Learning Fundamentals

Ivanovitch Silva
November, 2017



Agenda

- Introduction
- Kinds of machine learning algorithms (k-nearest neighbors)
- Problems definition: airbnb
- Univariate KNN
 - Euclidean distance for univariate
 - Function to make predictions
 - Error metrics
- Multivariate KNN
 - Normalize columns
 - Euclidean distance for multivariate
- Introduction to scikit-learn

Introduction

Outcome: Draw a Face

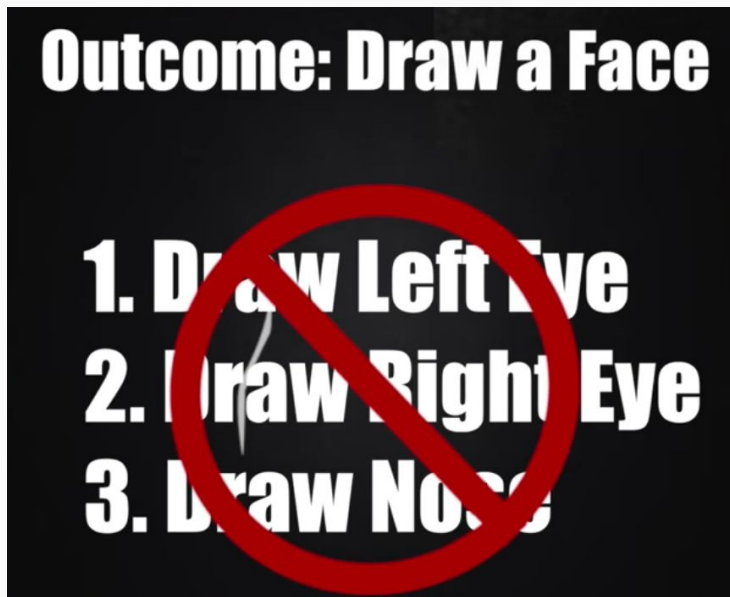
1. Draw Left Eye

2. Draw Right Eye

3. Draw Nose

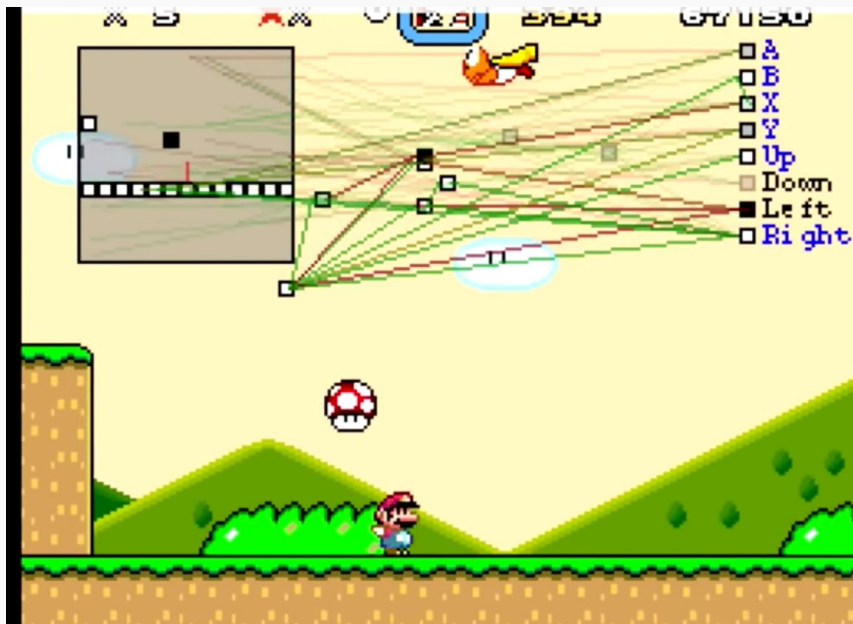
Traditional programming has been about defining every single step for programs to reach outcomes.

Introduction to Machine Learning

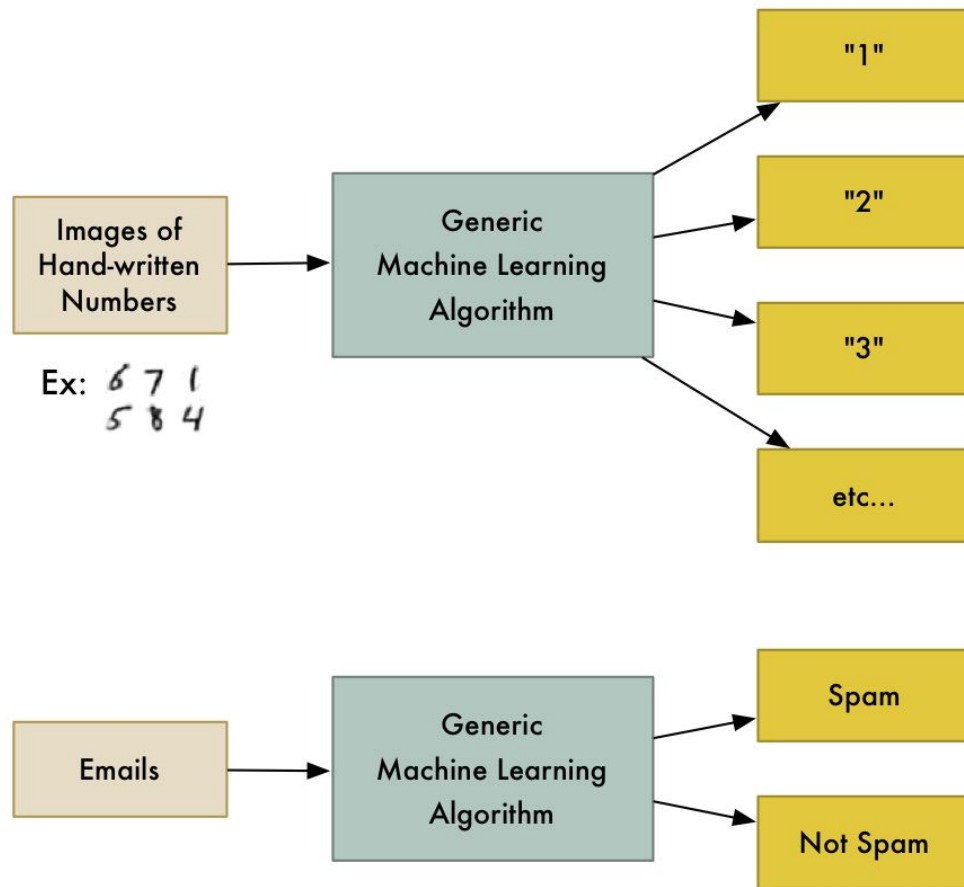


In Machine Learning (ML) we define outcome and the program learns the steps to get there.

Introduction to ML



Instead of writing code for every possible scenario, we say in ML, the goal is to get to endpoint without dying learn the steps to get there.



Style Transfer



Style transfer allows you to take famous paintings, and recreate your own images in their styles!

Style Transfer

Project (Logan Engstrom/MIT): [fast-style-transfer GitHub repo](https://github.com/lengstrom/fast-style-transfer)

```
git clone https://github.com/lengstrom/fast-style-transfer
```

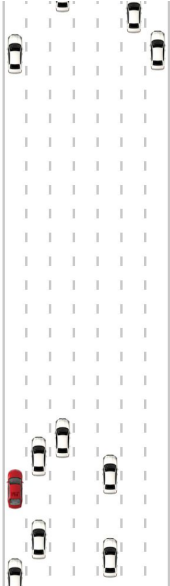
Notebook

```
!pip install tensorflow  
!pip install scipy  
!pip install pillow
```


Deep Traffic

<http://selfdrivingcars.mit.edu/deeptraffics/>

Speed:
34 mph
Cars Passed:
-33



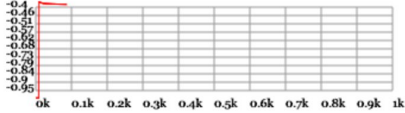
```
1
2 //<![CDATA[
3
4 // a few things don't have var in front of them - they update already
   existing variables the game needs
5 lanesSide = 0;
6 patchesAhead = 1;
7 patchesBehind = 0;
8 trainIterations = 10000;
9
10 var num_inputs = (lanesSide * 2 + 1) * (patchesAhead + patchesBehind);
11 var num_actions = 5;
12 var temporal_window = 3;
13 var network_size = num_inputs * temporal_window + num_actions *
```

Apply Code/Reset Net

Save Code/Net to File

Load Code/Net from File

Submit Model to Competition

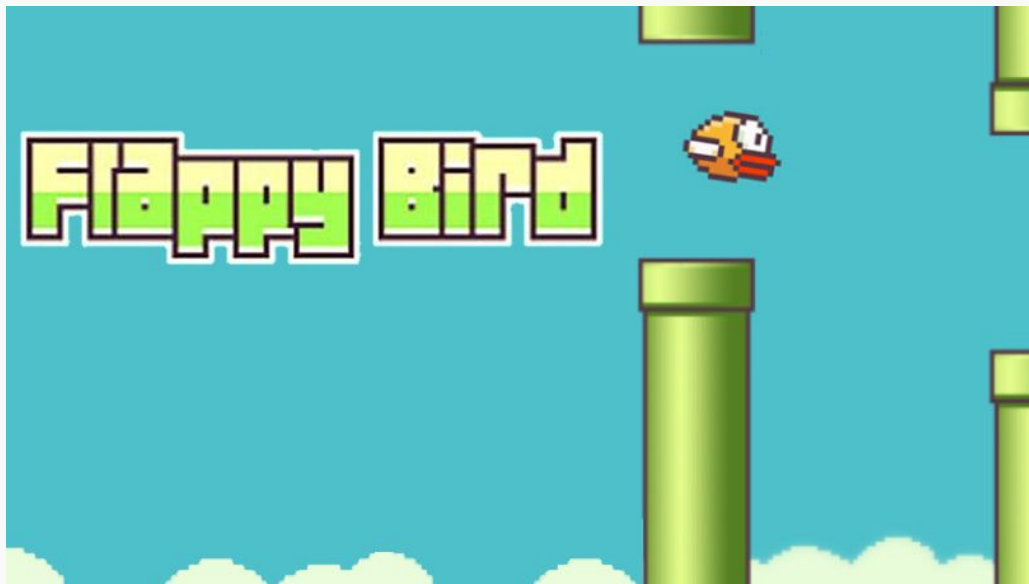


Run Training

Start Evaluation Run

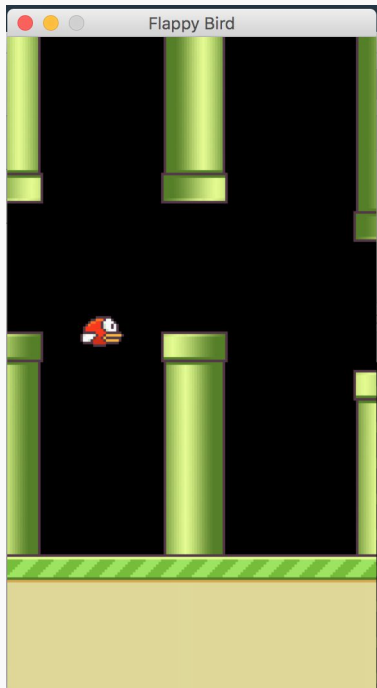
Flappy Bird

<https://github.com/yenchenlin/DeepLearningFlappyBird>

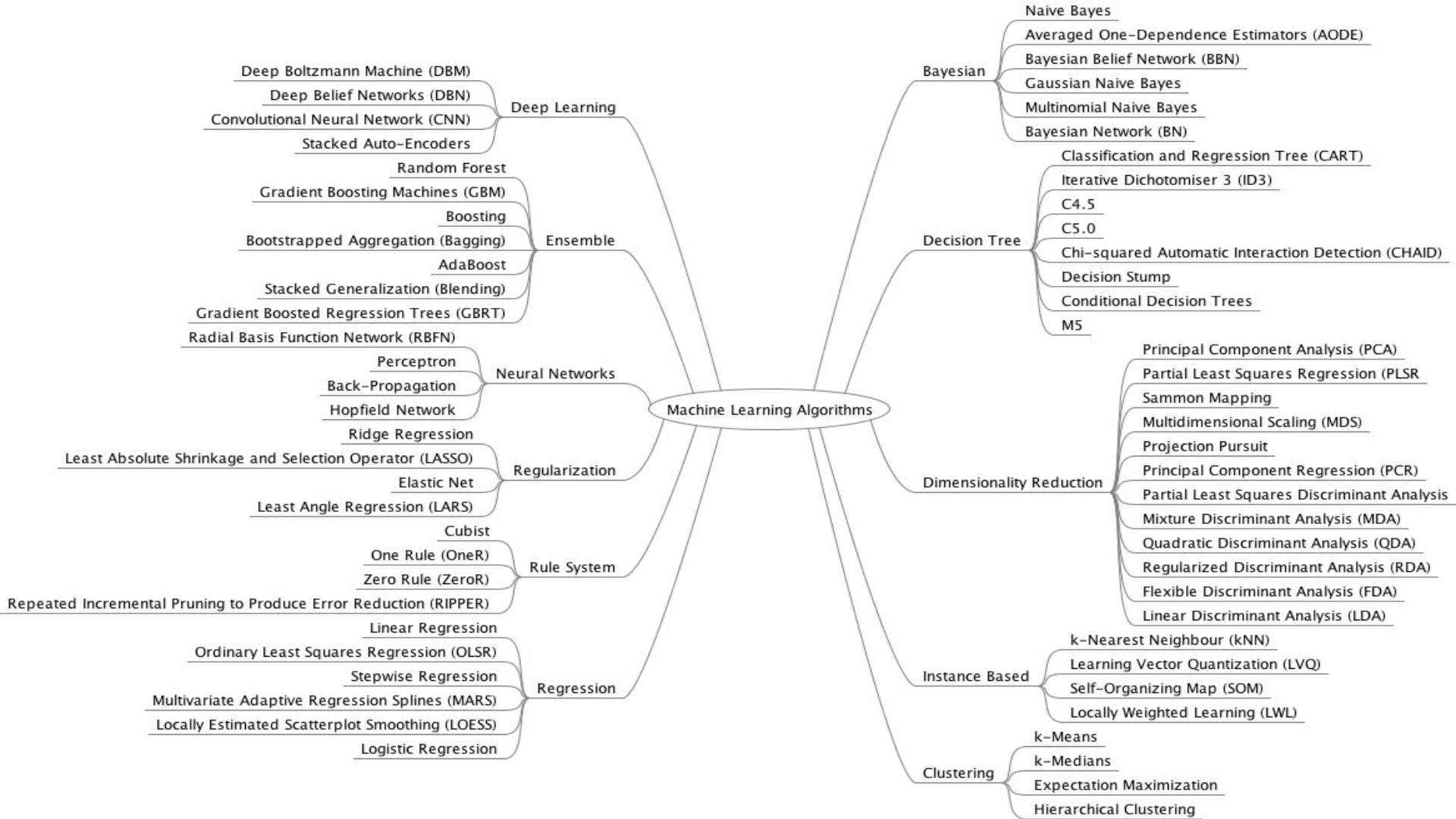


The following agent is able to play without being told any information about the structure of the game or its rules. It automatically discovers the rules of the game by finding out how it did on each iteration.

Flappy Bird



1. `conda install -c menpo opencv3`
2. `pip install pygame`
3. `pip install tensorflow`
4. `git clone`
`https://github.com/yenchenlin/DeepLearningFlappyBird.git`
5. `cd DeepLearningFlappyBird`
6. `python deep_q_network.py`



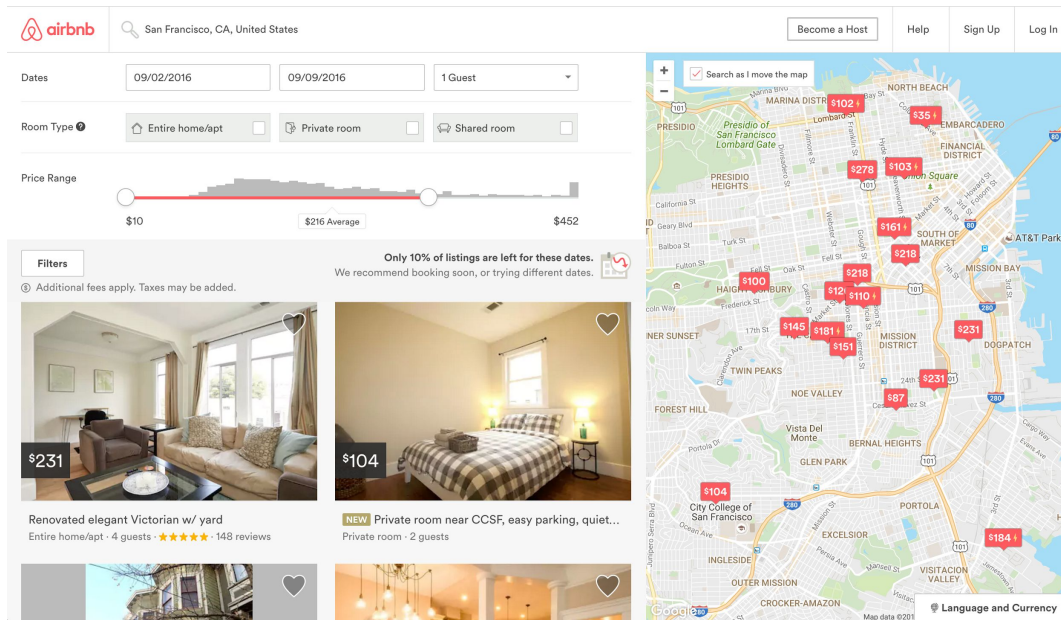
Kinds of ML Algorithms

	Algorithms
Unsupervised Learning	<i>k</i> -Means Clustering Principal Component Analysis Association Rules Social Network Analysis
Supervised Learning	Regression Analysis <i>k</i> -Nearest Neighbors Support Vector Machine Decision Tree Random Forests Neural Networks
Reinforcement Learning	Multi-Armed Bandits



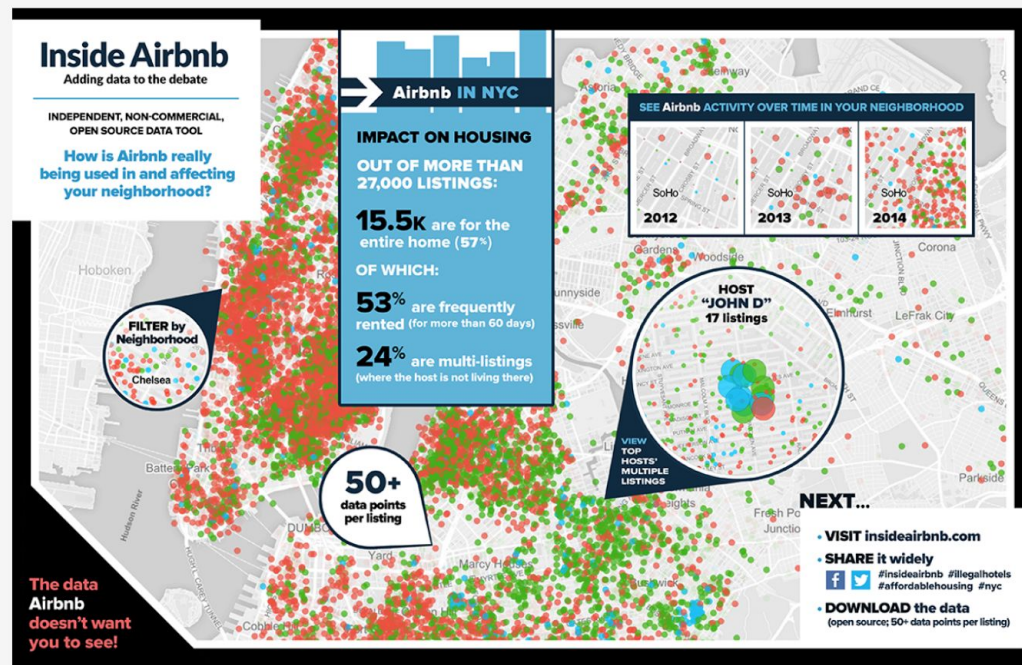
Problem definition

One challenge that hosts looking to rent their living space face is determining the optimal nightly rent price





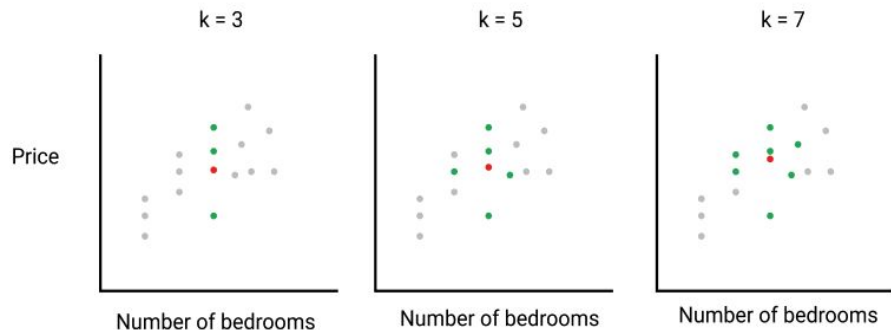
About Inside Airbnb



October 3, 2015 on the
listings from
Washington, D.C

- **host_response_rate**: the response rate of the host
- **host_acceptance_rate**: number of requests to the host that convert to rentals
- **host_listings_count**: number of other listings the host has
- **latitude**: latitude dimension of the geographic coordinates
- **longitude**: longitude part of the coordinates
- **city**: the city the living space resides
- **zipcode**: the zip code the living space resides
- **state**: the state the living space resides
- **accommodates**: the number of guests the rental can accommodate
- **room_type**: the type of living space (Private room, Shared room or Entire home/apt)
- **bedrooms**: number of bedrooms included in the rental
- **bathrooms**: number of bathrooms included in the rental
- **beds**: number of beds included in the rental
- **price**: nightly price for the rental
- **cleaning_fee**: additional fee used for cleaning the living space after the guest leaves
- **security_deposit**: refundable security deposit, in case of damages
- **minimum_nights**: minimum number of nights a guest can stay for the rental
- **maximum_nightss**: maximum number of nights a guest can stay for the rental
- **number_of_reviews**: number of reviews that previous guests have left

Select the number of similar listings, k , you want to compare with.



For each listing, calculate how similar it is to our unpriced listing.



For this example, we'll use 3 for our k value.

K-Nearest Neighbors

Rank each listing by the similarity metric and select the first k listings.

dataset (ordered by similarity)

bedrooms	price	similarity
1	160	0
1	60	0
1	95	0
1	50	0
3	350	2

our unpriced listing

bedrooms	price
1	?

Calculate the mean list price for the k similar listings and use as our list price.



Euclidean distance

$$d = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$$

index	host_listings_count	accommodates	bedrooms	bathrooms	beds
0	26	4	1	1	2
1	1	6	3	3	3

$$(q_1 - p_1) + (q_2 - p_2) + \dots + (q_n - p_n) \quad \text{differences} \quad (26 - 1) + (4 - 6) + (1 - 3) + (1 - 3) + (2 - 3)$$

$$(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2 \quad \text{squared differences} \quad (25)^2 + (-2)^2 + (-2)^2 + (-2)^2 + (-1)^2$$

$$\sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$$

Euclidean distance	= $\sqrt{625 + 4 + 4 + 4 + 1}$
	= $\sqrt{638}$
	= 25.258661

Euclidean distance - example

Univariate case

$$d = \sqrt{(q_1 - p_1)^2}$$

$$d = |q_1 - p_1|$$

	accommodates
our listing	8

	index	accommodates	distance
dc_listings	0	4	$(4 - 8)^2$
	1	6	$(6 - 8)^2$
	2	1	$(1 - 8)^2$
	3	2	$(2 - 8)^2$

Randomizing and sorting

```
dc_listings[dc_listings["distance"] == 0]["accommodates"]
```

```
26      3  
34      3  
36      3  
40      3  
44      3  
45      3  
48      3  
65      3  
66      3  
71      3  
75      3  
86      3  
...
```

Cleaning & preparing data

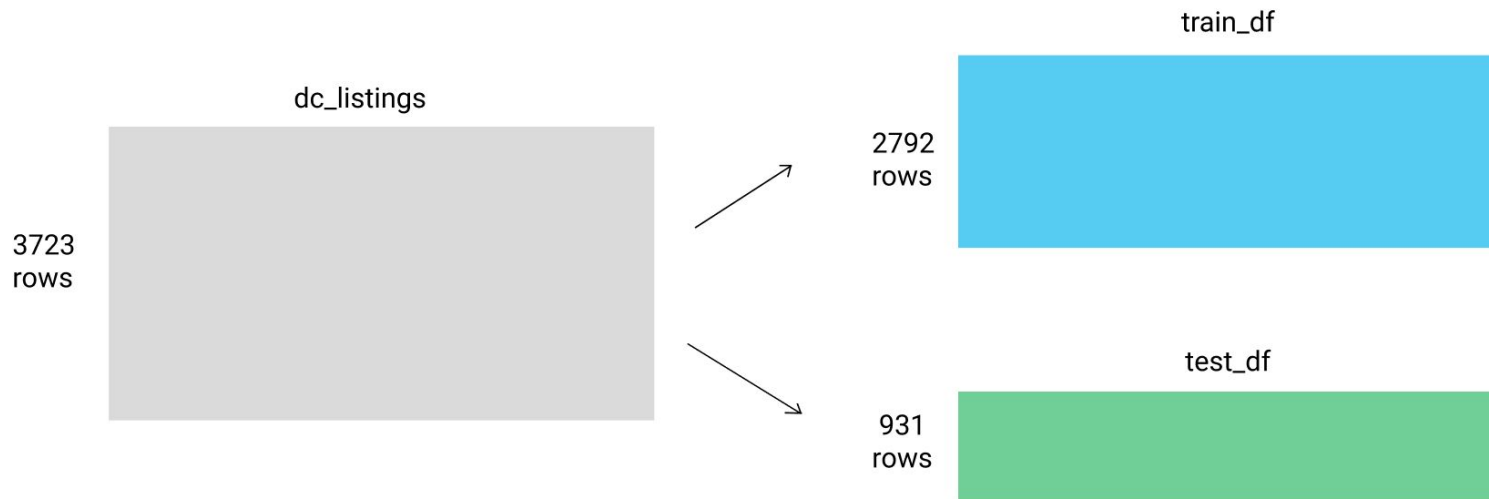
```
# Brought along the changes we made to the `dc_listings` Dataframe.  
dc_listings = pd.read_csv('dc_airbnb.csv')  
stripped_commas = dc_listings['price'].str.replace(',', '')  
stripped_dollars = stripped_commas.str.replace('$', '')  
dc_listings['price'] = stripped_dollars.astype('float')  
dc_listings = dc_listings.loc[np.random.permutation(len(dc_listings))]
```

Function to make predictions

```
def predict_price(new_listing):  
    temp_df = dc_listings  
    temp_df['distance'] = temp_df['accommodates'].apply(lambda x: np.abs(x - new_listing))  
    temp_df = temp_df.sort_values('distance')  
    nearest_neighbors = temp_df.iloc[0:5]['price']  
    predicted_price = nearest_neighbors.mean()  
    return(predicted_price)
```

Testing quality of predictions

Machine Learning Model



Error metrics

- We now need a metric that quantifies how good the predictions were on the test set.

Mean Absolute Error

$$MAE = \frac{|actual_1 - predicted_1| + |actual_2 - predicted_2| + \dots + |actual_n - predicted_n|}{n}$$

Mean Squared Error

$$MSE = \frac{(actual_1 - predicted_1)^2 + (actual_2 - predicted_2)^2 + \dots + (actual_n - predicted_n)^2}{n}$$

Root Mean Squared Error (RMSE)

- While comparing MSE values helps us identify which model performs better on a relative basis, it doesn't help us understand if the performance is good enough in general.
- This is because the units of the MSE metric are squared (in this case, dollars squared)

$$RMSE = \sqrt{MSE}$$



Part I - Introduction to K-Nearest Neighbors.ipynb
Part II - Evaluating Model Performance.ipynb

Improve the accuracy (multivariate knn)

- **Increase the number of attributes** the model uses to calculate similarity when ranking the closest neighbors
- **Increase k**, the number of nearby neighbors the model uses when computing the prediction

Improve the accuracy (multivariate knn)

When selecting more attributes to use in the model, we need to watch out for columns that don't work well with the distance equation.

- **non-numerical values** (e.g. city or state)
 - Euclidean distance equation expects numerical values
- **missing values**
 - distance equation expects a value for each observation and attribute
- **non-ordinal values** (e.g. latitude or longitude)
 - ranking by Euclidean distance doesn't make sense if all attributes aren't ordinal

Removing features

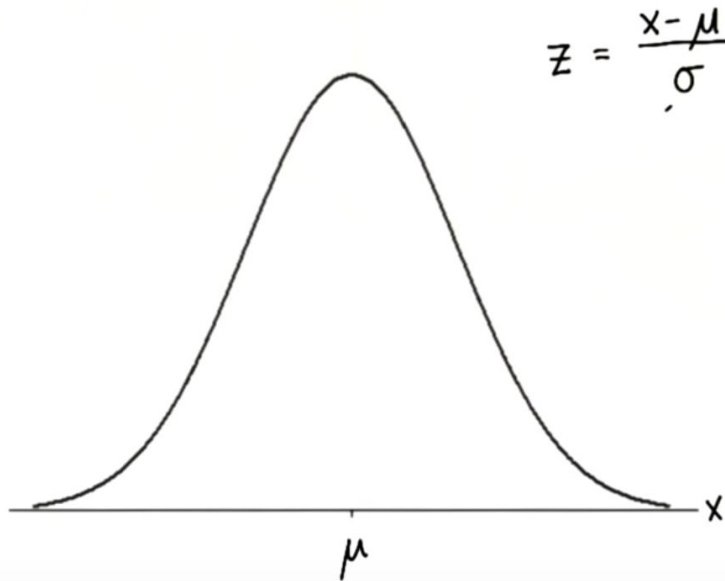
- **room_type**: e.g. **Private room**
- **city**: e.g. **Washington**
- **state**: e.g. **DC**
- **latitude**: e.g. **38.913458**
- **longitude**: e.g. **-77.031**
- **zipcode**: e.g. **20009**
- **host_response_rate**
- **host_acceptance_rate**
- **host_listings_count**

Normalize columns

accommodates	bedrooms	bathrooms	beds	price	minimum_nights	maximum_nights	number_of_reviews
2	1.0	1.0	1.0	125.0	1	4	149
2	1.0	1.5	1.0	85.0	1	30	49
1	1.0	0.5	1.0	50.0	1	1125	1
2	1.0	1.0	1.0	209.0	4	730	2
12	5.0	2.0	5.0	215.0	2	1825	34

Normalize columns

```
normalized_listings = (dc_listings - dc_listings.mean()) / (dc_listings.std())
```



Normalize columns

	accommodates	bedrooms	bathrooms	beds	price	minimum_nights	maximum_nights	number_of_reviews
574	-0.596544	-0.249467	-0.439151	-0.546858	125.0	-0.341375	-0.016604	4.579650
1593	-0.596544	-0.249467	0.412923	-0.546858	85.0	-0.341375	-0.016603	1.159275
3091	-1.095499	-0.249467	-1.291226	-0.546858	50.0	-0.341375	-0.016573	-0.482505
420	-0.596544	-0.249467	-0.439151	-0.546858	209.0	0.487635	-0.016584	-0.448301
808	4.393004	4.507903	1.264998	2.829956	215.0	-0.065038	-0.016553	0.646219

Euclidean distance for multivariate case

accommodates	bathrooms
-0.596544	-0.439151
-0.596544	0.412923

$$(q_1 - p_1) + (q_2 - p_2) + \dots + (q_n - p_n)$$

differences $(-0.596544 + 0.596544) + (-0.439151 - 0.412923)$

$$(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2$$

squared differences $(0)^2 + (-0.852074)^2$

$$\sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2}$$

Euclidean distance

$$= \sqrt{0 + 0.72603}$$

$$= 0.852074$$

Euclidean distance for multivariate case

```
from scipy.spatial import distance
first_listing = [-0.596544, -0.439151]
second_listing = [-0.596544, 0.412923]
dist = distance.euclidean(first_listing, second_listing)
```

Introduction to scikit-learn



scikit-learn
Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.

Algorithms: SVM, nearest neighbors, random forest, ... — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.

Algorithms: SVR, ridge regression, Lasso, ... — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes

Algorithms: k-Means, spectral clustering, mean-shift, ... — Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency

Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning

Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.

Modules: preprocessing, feature extraction. — Examples

Scikit-learn workflow

The scikit-learn workflow consists of 4 main steps:

- instantiate the specific machine learning model you want to use
- fit the model to the training data
- use the model to make predictions
- evaluate the accuracy of the predictions

Instantiate the machine learning model

```
from sklearn.neighbors import KNeighborsRegressor  
knn = KNeighborsRegressor()
```

Fitting the model to the training data

```
# Split full dataset into train and test sets.  
train_df = normalized_listings.iloc[0:2792]  
test_df = normalized_listings.iloc[2792:]  
# Matrix-like object, containing just the 2 columns of interest from training set.  
train_features = train_df[['accommodates', 'bathrooms']]  
# List-like object, containing just the target column, `price`.  
train_target = normalized_listings['price']  
# Pass everything into the fit method.  
knn.fit(train_features, train_target)
```

Make predictions

```
predictions = knn.predict(test_df[['accommodates', 'bathrooms']])
```

Calculating MSE/RMSE using scikit-learn

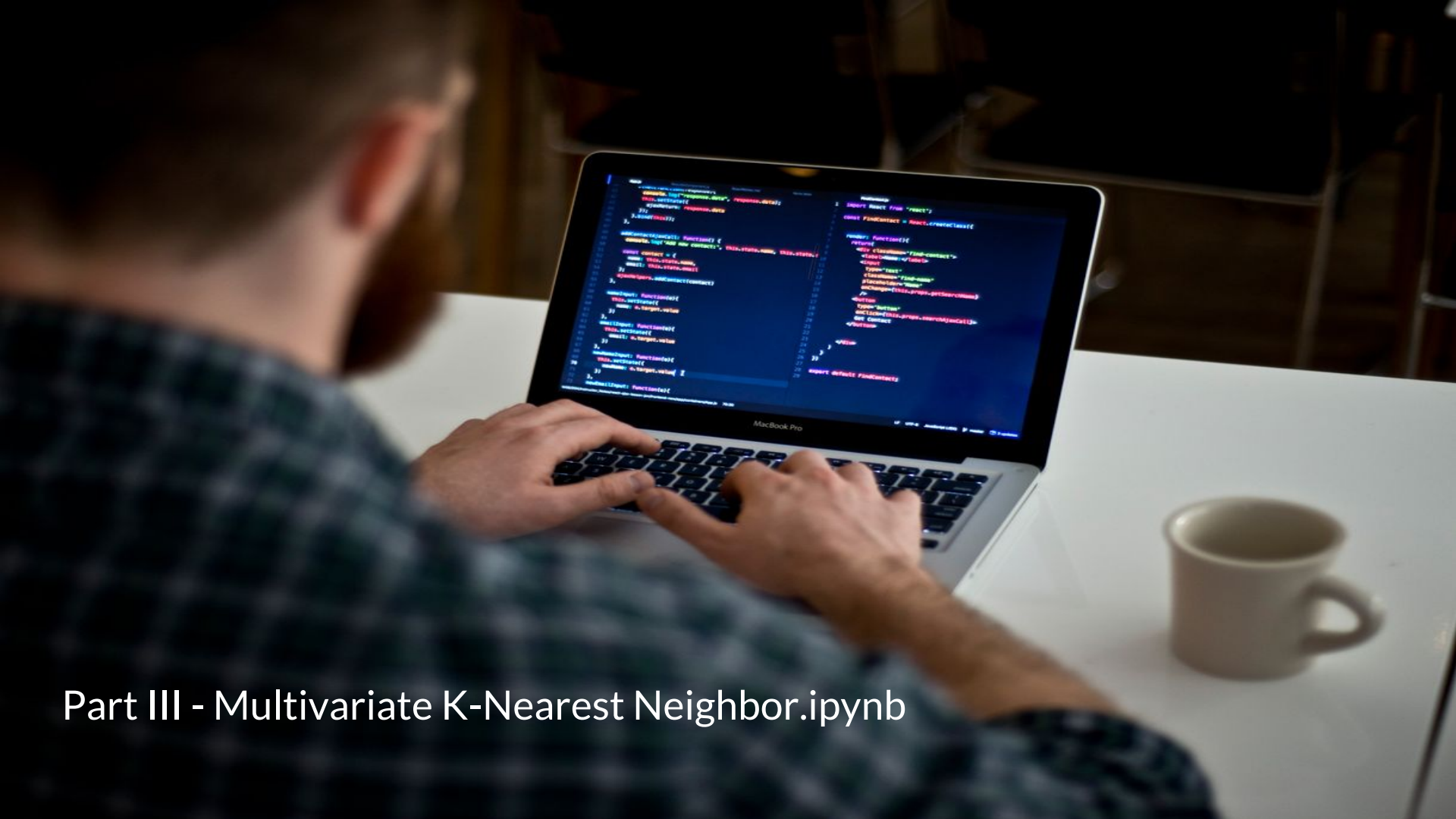
```
all_features_mse = mean_squared_error(test_df['price'], all_features_predictions)
all_features_rmse = all_features_mse ** (1/2)
```


Using more features

feature(s)	MSE	RMSE
accommodates	18646.5	136.6
bathrooms	17333.4	131.7
accommodates, bathrooms	15660.4	125.1
accommodates, bathrooms, bedrooms, number_of_reviews	13320.2	115.4

Using all features

???



Part III - Multivariate K-Nearest Neighbor.ipynb