High-Dimensional Visualization

04/04/2019

Overview

- Data Sense-making by feature
- Preprocessing
- Data Sense-making based on Transformation

High Dimension is a challenge!

- Hard to have a direct look at the data space
- Curse of dimensionality in the data analysis process
    - Distortion
    - More data are required

Prepare for the lab...

Libraries we need in this lab:

- pandas
- numpy
- sklearn
- seaborn
- matplotlib

General Data processing

Data Manipulation (Machine Learning)

Visualization

```
pip install [package]
```

Data

Ames, Iowa Housing Dataset

- Download it here: https://github.com/nyuvis/visual_analytics_course/blob/master/hd_vis_lab/AmesHousing.csv
- Not truly "high" dimension, but more than those numerical dataset we usually used in data mining projects
- Originally a dataset for regression (housing price)

```
filename = "./AmesHousing.csv"
df = pd.read_csv(filepath_or_buffer=filename, sep=',', header=0,
index_col=None)

df.head()
```

Feature Extraction

- We analyze numerical data only in this lab

- We need to do other transformation to the categorical data (think of what we have done for the text data)

- You can check the jupyter notebook for this data set here to learn more about how to deal with the categorical and numerical data: https://www.kaggle.com/leeclemmer/exploratory-data-analysis-of-housing-in-ames-iowa/data

Feature Extraction (numerical and categorical)

```
def get_feature_groups(df):

    """ Returns a list of numerical and categorical features,

    excluding SalePrice and Id. """

    # Numerical Features

    num_features = df.select_dtypes(include=['int64','float64']).columns

    num_features = num_features.drop(['SalePrice']) # drop ID and SalePrice

    # Categorical Features

    cat_features = df.select_dtypes(include=['object']).columns

    return list(num_features), list(cat_features)
```

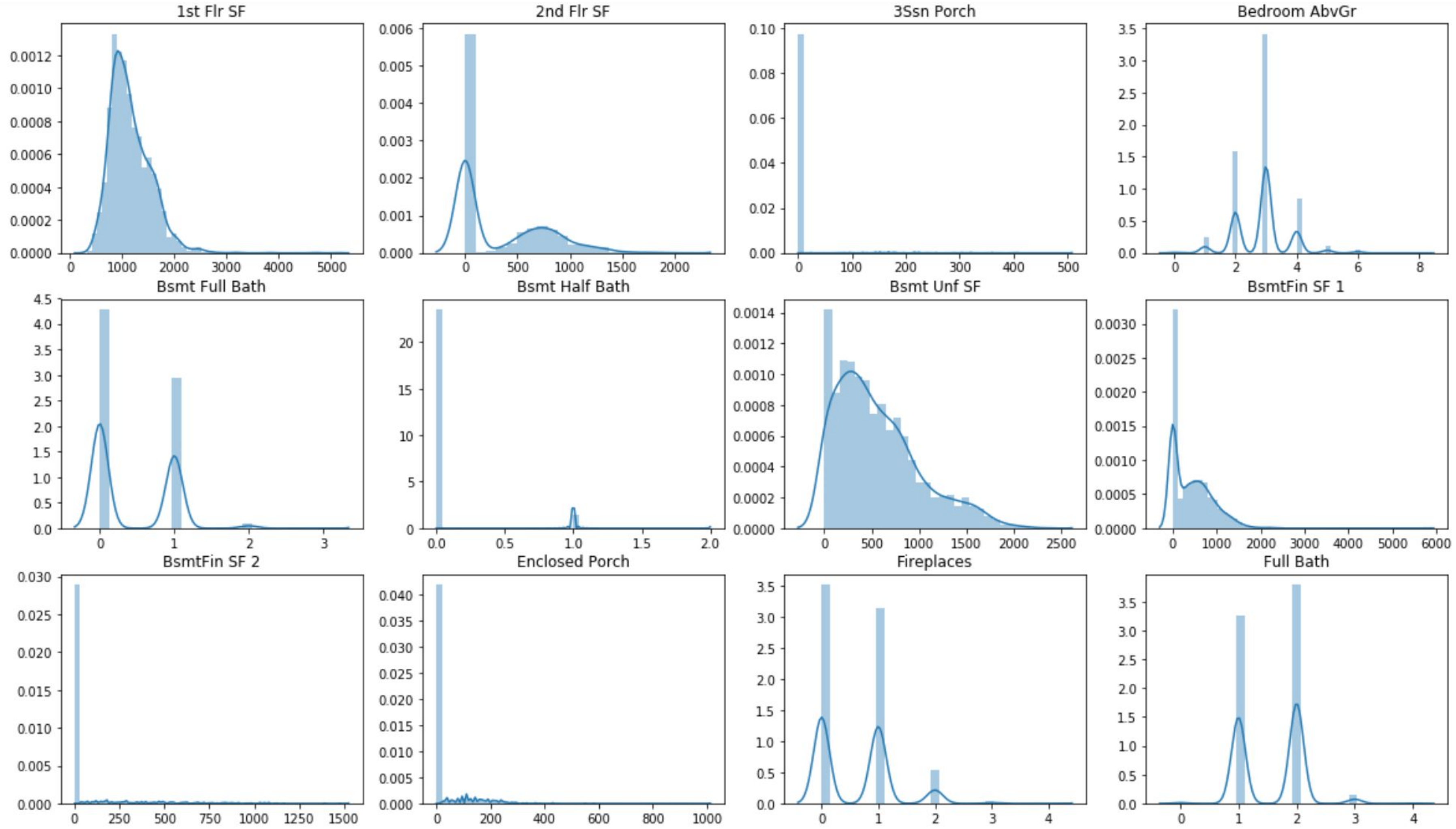
Feature Extraction

```
df = df.fillna(0)

num_features, cat_features = get_feature_groups(df)
```

Feature Distribution

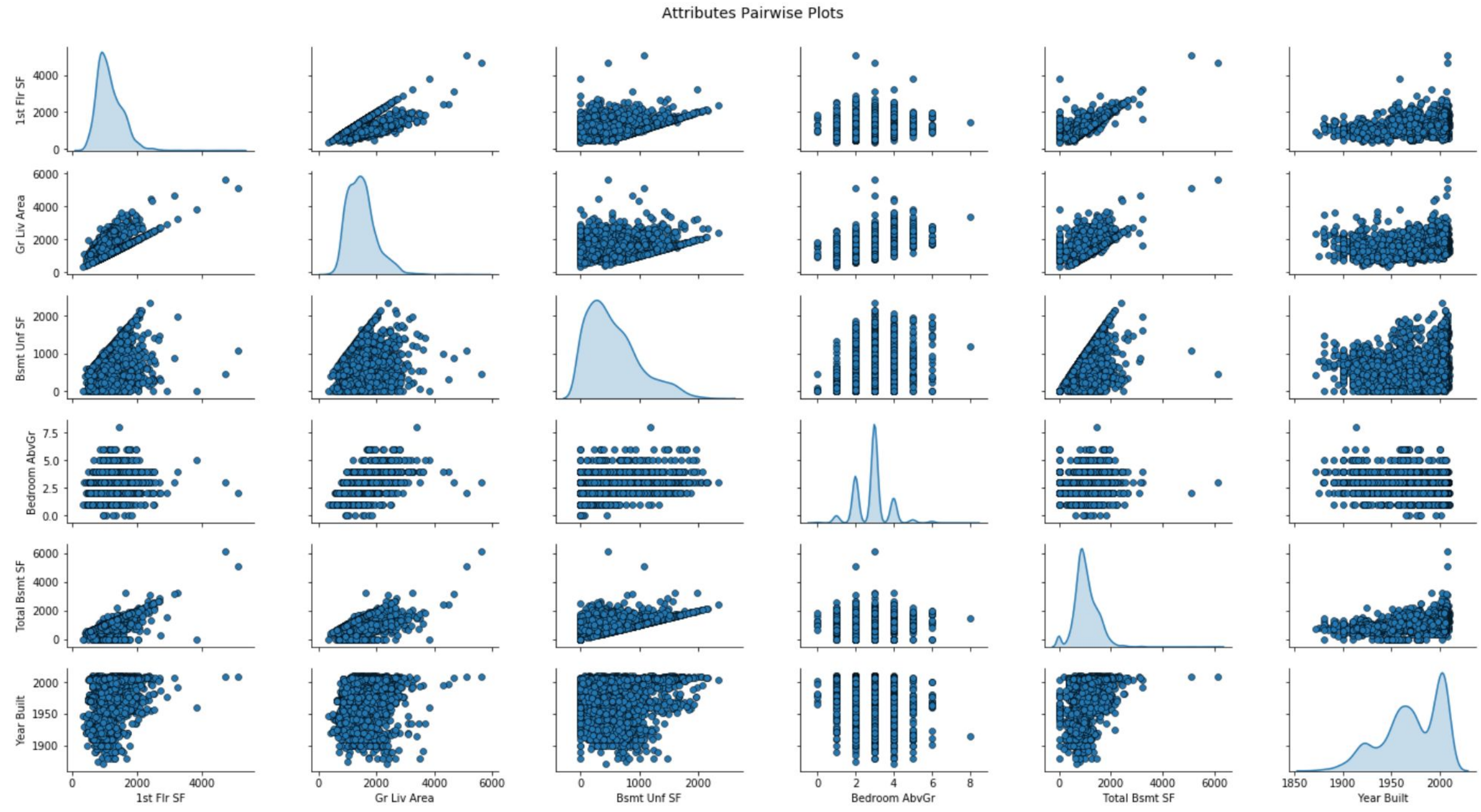
```
def distribution(df, num_features):

    f = pd.melt(df, value_vars=sorted(num_features))

    g = sns.FacetGrid(f, col='variable', col_wrap=4,
sharex=False, sharey=False)

    g = g.map(sns.distplot, 'value')
```

Visualizing


1st Flr SF
2nd Flr SF
3Ssn Porch
Bedroom AbvGr
Bsmt Full Bath
Bsmt Half Bath
Bsmt Unf SF
BsmtFin SF 1
BsmtFin SF 2
Enclosed Porch
Fireplaces
Full Bath

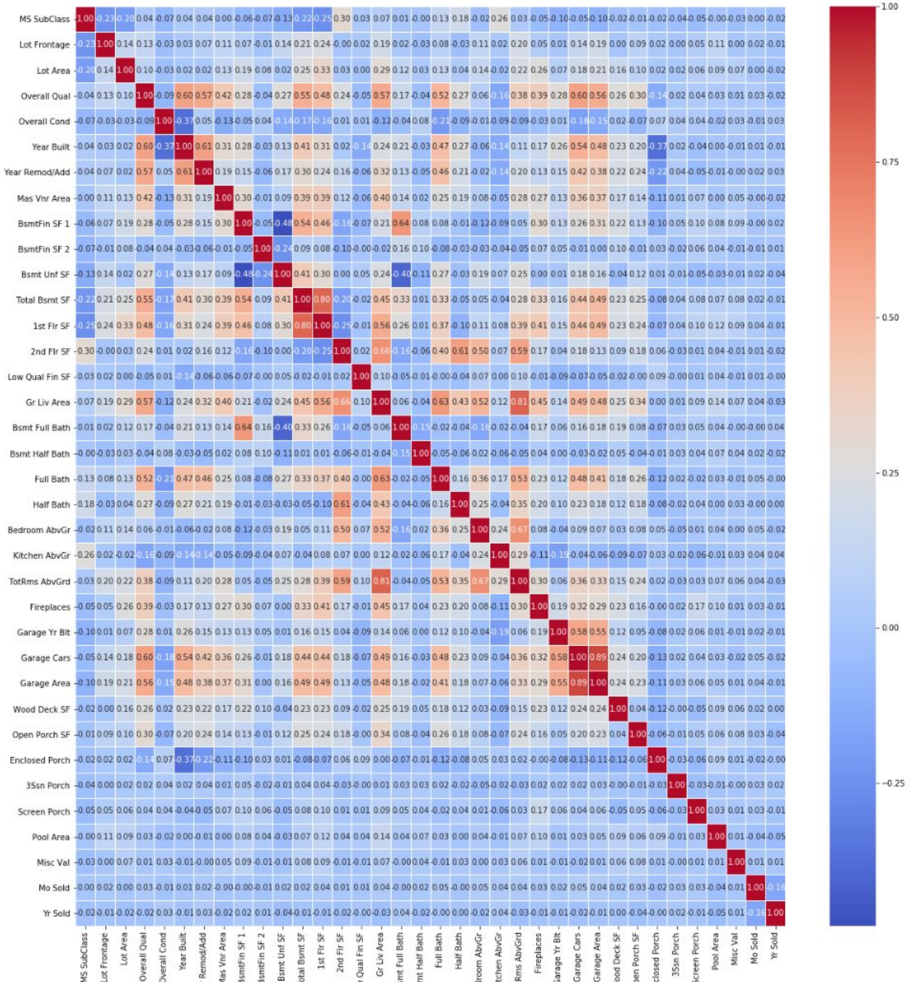
Feature correlation: Scatter Matrix (pair-wise scatter plots)

```
cols = ['1st Flr SF', 'Gr Liv Area', 'Bsmt Unf SF', 'Bedroom AbvGr', 'Total Bsmt SF', 'Year
Built']

pp = sns.pairplot(df[cols], size=1.8, aspect=1.8,

                  plot_kws=dict(edgecolor="k", linewidth=0.5),

                  diag_kind="kde", diag_kws=dict(shade=True))



fig = pp.fig

fig.subplots_adjust(top=0.93, wspace=0.3)

t = fig.suptitle('Wine Attributes Pairwise Plots', fontsize=14)
```


Attributes Pairwise Plots
1st Flr SF
Gr Liv Area
Bsmt Unf SF
Bedroom AbvGr
Total Bsmt SF
Year Built

Feature Correlation: Heatmap

```
f, ax = plt.subplots(figsize=(20, 20))

corr = df[num_features].corr()

hm = sns.heatmap(round(corr,2), annot=True, ax=ax, cmap="coolwarm",fmt='.2f',

                 linewidths=.05)

f.subplots_adjust(top=0.93)

t= f.suptitle('Attributes Correlation Heatmap', fontsize=14)
```


MS SubClass
Lot Frontage
Lot Area
Overall Qual
Overall Cond
Year Built
Year Remod/Add
Mas Vnr Area
BsmtFin SF 1
BsmtFin SF 2
Bsmt Unf SF
Total Bsmt SF
1st Flr SF
2nd Flr SF
Low Qual Fin SF
Gr Liv Area
Bsmt Full Bath
Bsmt Half Bath
Full Bath
Half Bath
Bedroom AbvGr
Kitchen AbvGr
TotRms AbvGrd
Fireplaces
Garage Yr Blt
Garage Cars
Garage Area
Wood Deck SF
Open Porch SF
Enclosed Porch
3Ssn Porch
Screen Porch
Pool Area
Misc Val
Mo Sold
Yr Sold
1.00
0.75
0.50
0.25
0.00
−0.25

Dimension Reduction

Overview

Project original high-dimensional data to a lower dimensional space, e.g., 2 dimension.

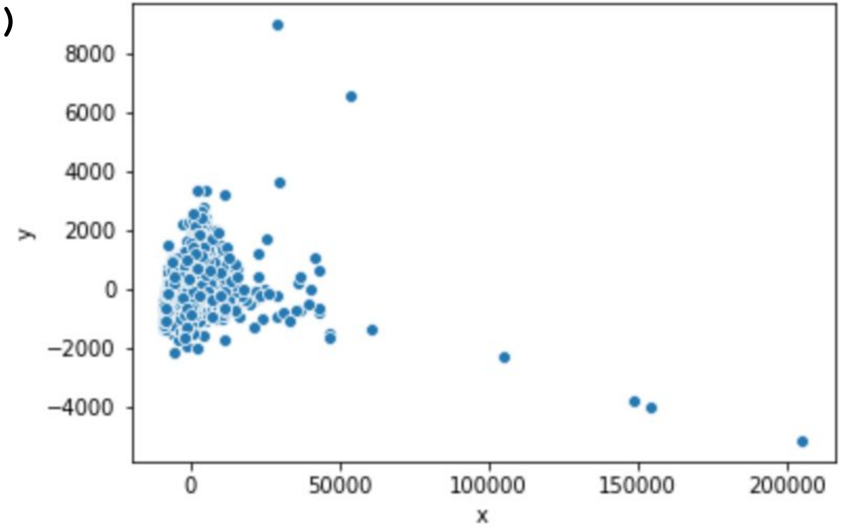
- PCA
- t-SNE
- UMAP (similar to t-SNE)

```
from sklearn.decomposition import PCA

from sklearn.manifold import TSNE
```

PCA

```
result_pca = PCA(n_components=2).fit_transform(df[num_features].values)
```

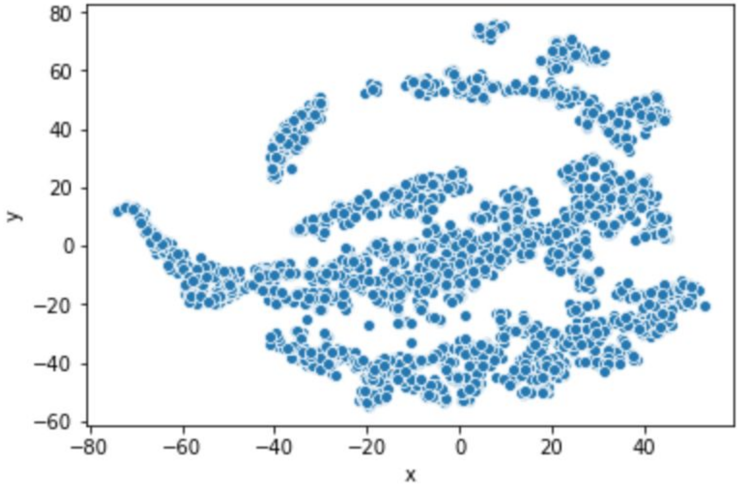
```
pca_df = pd.DataFrame(data=result_pca, columns=['x','y'])

sns.scatterplot(x="x", y="y", data=pca_df)
```



t-SNE

```
result_tsne = TSNE(n_components=2).fit_transform(df[num_features].values)
```

```
tsne_df = pd.DataFrame(data=result_tsne, columns=['x','y'])

sns.scatterplot(x="x", y="y", data=tsne_df`
```



Don’t forget scaling...

Imagine that you have a data set of student information, the feature `year_in_school` ranges from 0 to 8, `weight` ranges from 50 to 100.

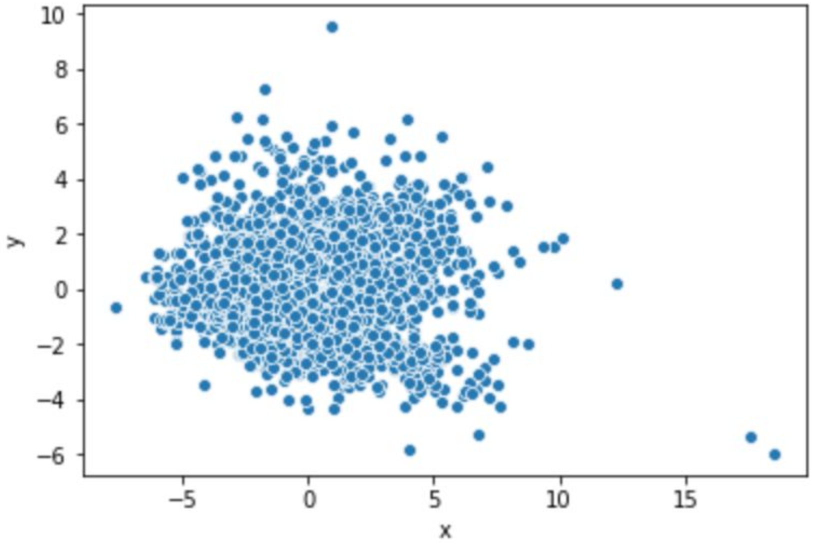
If we calculate the euclidean distance, the distance is highly related to `weight`.

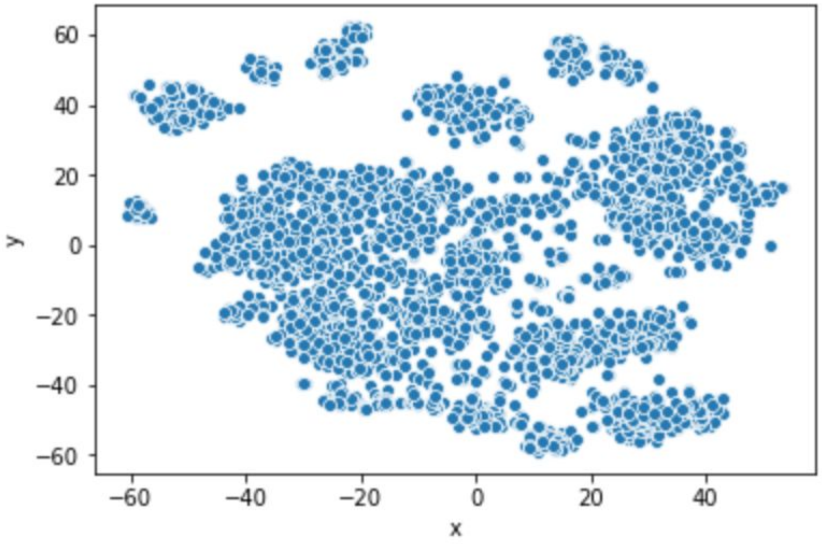
So we need to **`standardize`** our data here.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(df[num_features].values)
scaled = scaler.transform(df[num_features].values)
```

PCA & t-SNE

Run the previous code for PCA and t-SNE again...

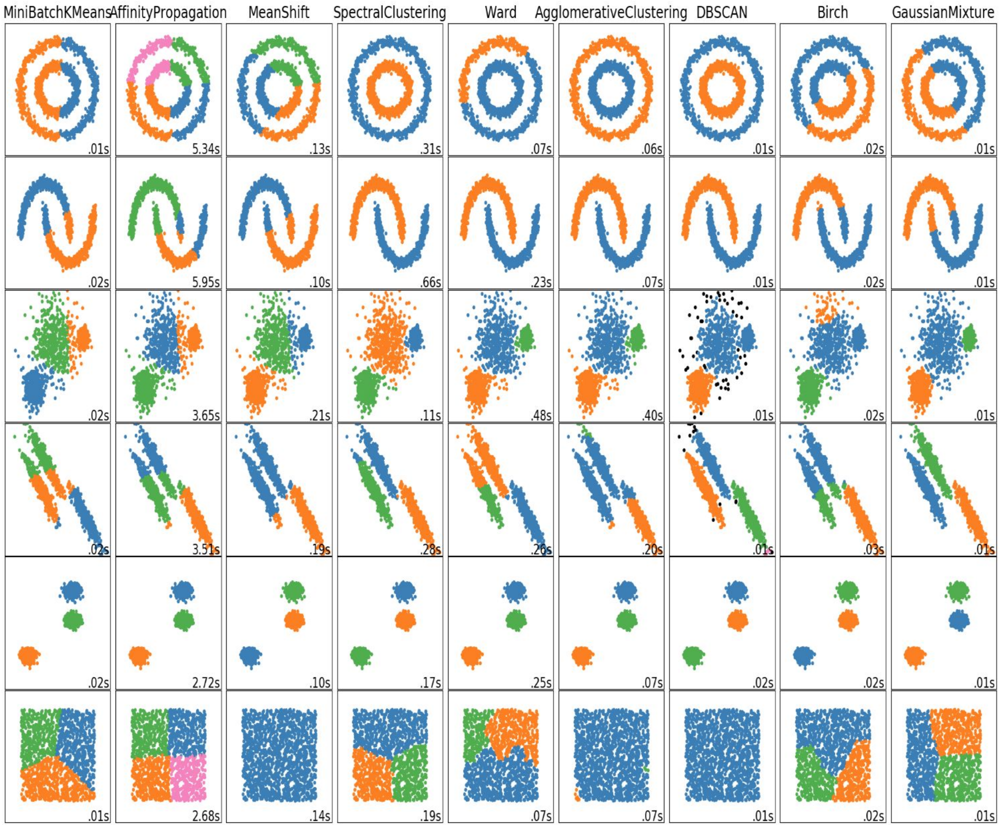




A real-time t-SNE project from Google: https://ai.googleblog.com/2018/06/realtime-tsne-visualizations-with.html
A 3d word projection: https://projector.tensorflow.org/

Clustering

Different Clustering Methods



DBSCAN: (Density Based Spatial Clustering of Applications with Noise)

```
from sklearn import cluster

dbscan = cluster.DBSCAN(eps=3, min_samples=7).fit(scaled)
```

The choice of parameters:

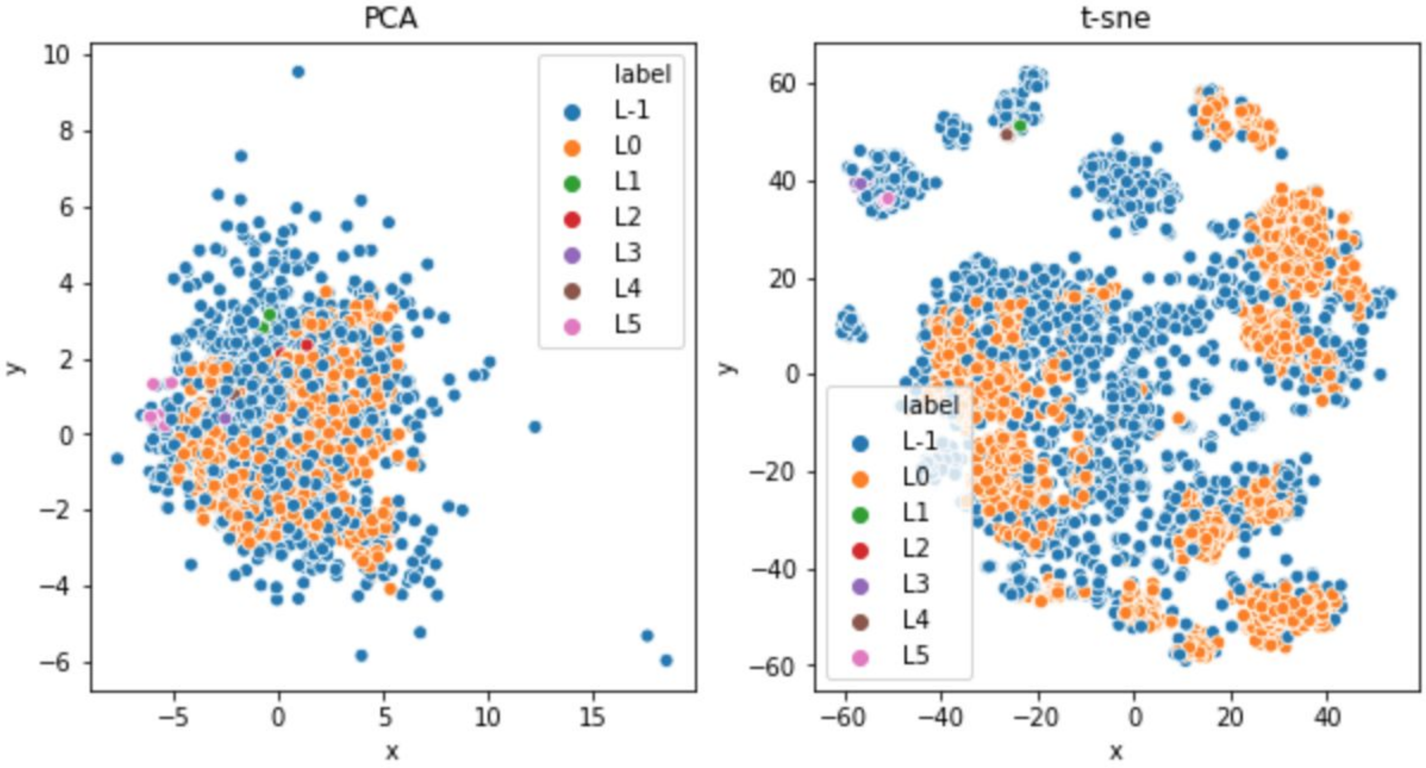
- ε (eps): a parameter specifying the radius of a neighborhood with respect to some point
- min_sample: the minimum number of points required to form a dense region

People are still working on how to get the optimal parameters for DBSCAN

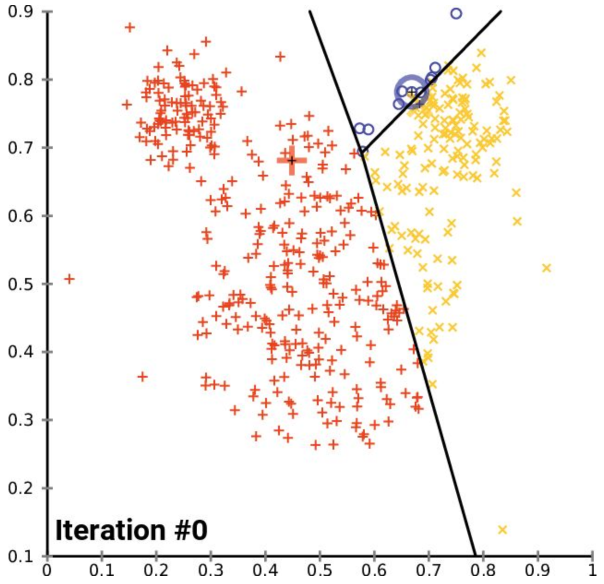
DBSCAN: Plot with projection

```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))


labels = ["L"+str(x) for x in dbscan.labels_]

pca_df2['label'] = labels

tsne_df2['label'] = labels


sns.scatterplot(x='x', y='y', data=pca_df2, ax=axes[0], hue='label').set_title('PCA')

sns.scatterplot(x='x', y='y', data=tsne_df2, ax=axes[1], hue='label').set_title('t-sne')
```

DBSCAN: plot with projection



K-means



https://upload.wikimedia.org/wikipedia/commons/e/ea/K-means_convergence.gif

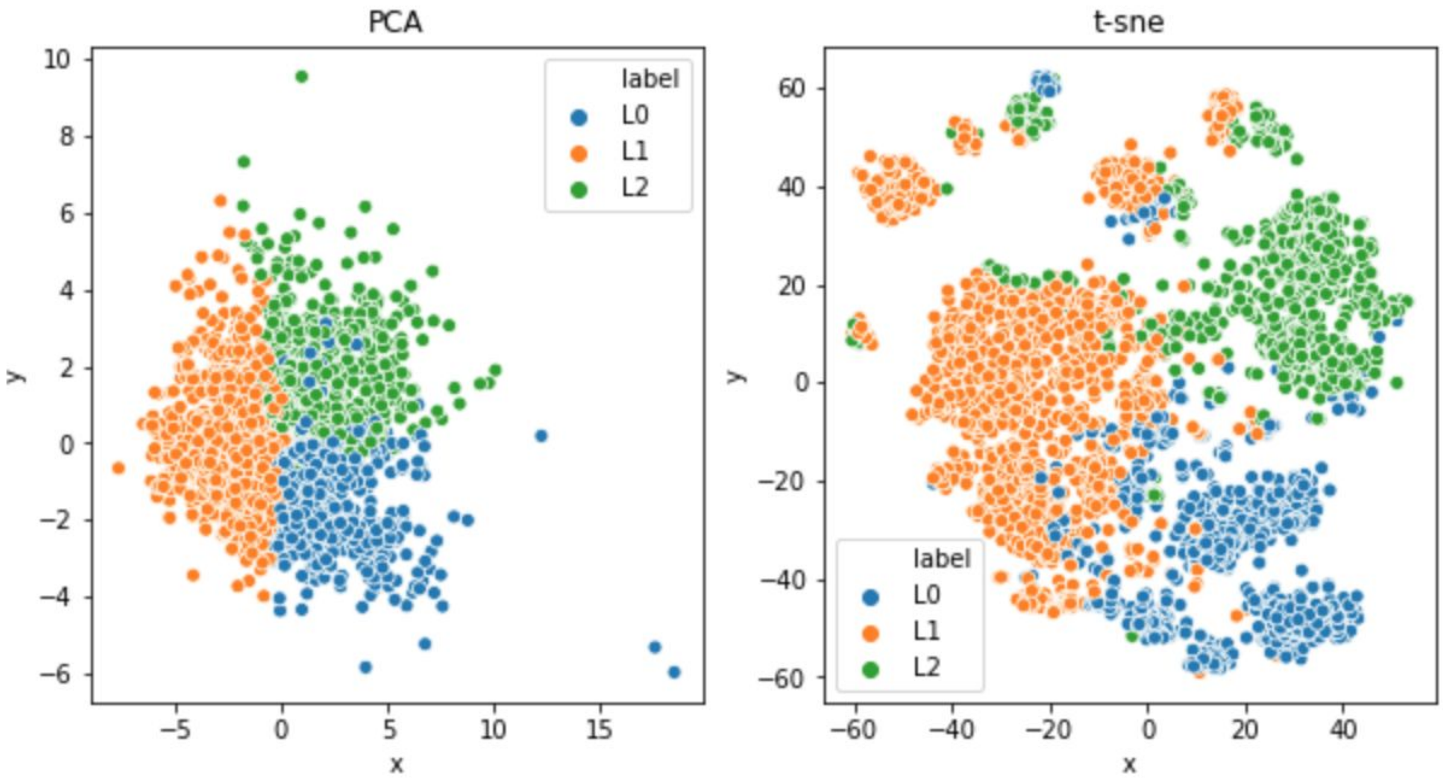
K-means

```
kmeans = cluster.KMeans(n_clusters=3).fit(scaled)
```

K-means: plot with projection

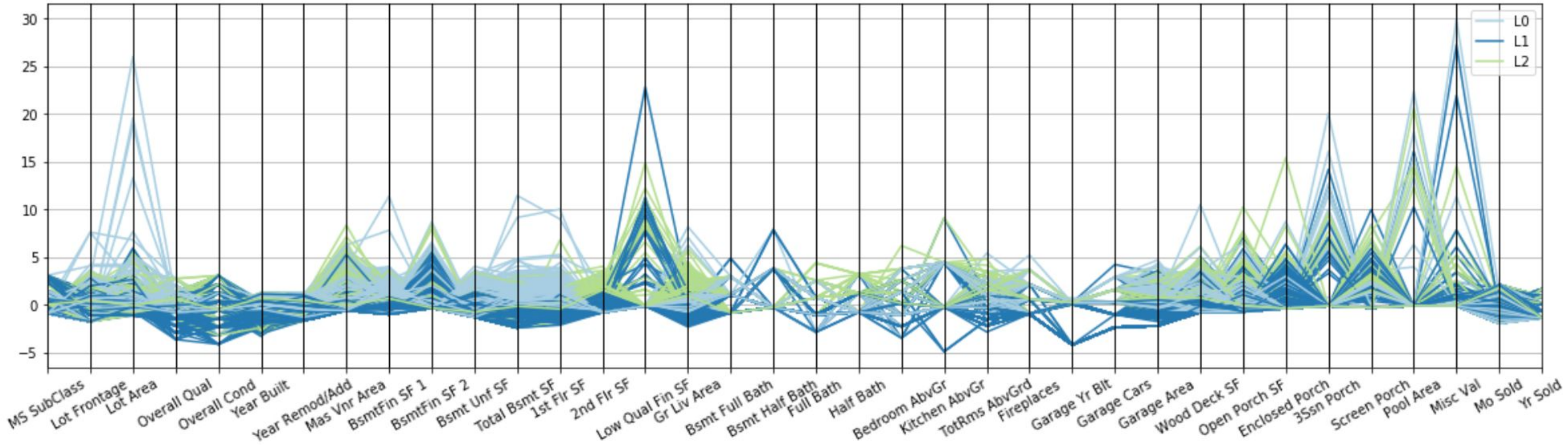
```
fig, axes = plt.subplots(nrows=1, ncols=2, figsize=(10, 5))



km_labels = ["L"+str(x) for x in kmeans.labels_]

pca_df2['label'] = km_labels

tsne_df2['label'] = km_labels



sns.scatterplot(x='x', y='y', data=pca_df2, ax=axes[0], hue='label').set_title('PCA')

sns.scatterplot(x='x', y='y', data=tsne_df2, ax=axes[1], hue='label').set_title('t-sne')
```

K-means


PCA
t-sne
label
L0
L1
L2
x
y
10
8
6
4
2
0
−2
−4
−6
−5
0
5
10
15
60
40
20
0
−20
−40
−60
−60
−40
−20
0
20
40

Parallel Coordinates

Use **`parallel_coordinates`** function in pandas library.



A demo...