Unsupervised Learning

May 23, 2022

```
[1]: # load data
     import pandas as pd
```

```
[2]: df = pd.read_csv("cars.csv")
```

```
[3]: df
```

```
[3]:       mpg  cylinders  cubicinches   hp  weightlbs  time-to-60  year  \
     0    14.0          8          350  165       4209          12  1972
     1    31.9          4           89   71       1925          14  1980
     2    17.0          8          302  140       3449          11  1971
     3    15.0          8          400  150       3761          10  1971
     4    30.5          4           98   63       2051          17  1978
     ..    …          …            …   …         …          …
     256  17.0          8          305  130       3840          15  1980
     257  36.1          4           91   60       1800          16  1979
     258  22.0          6          232  112       2835          15  1983
     259  18.0          6          232  100       3288          16  1972
     260  22.0          6          250  105       3353          15  1977

             brand
     0         US.
     1     Europe.
     2         US.
     3         US.
     4         US.
     ..        …
     256       US.
     257    Japan.
     258       US.
     259       US.
     260       US.

     [261 rows x 8 columns]
```

```
[4]: df.describe()
```

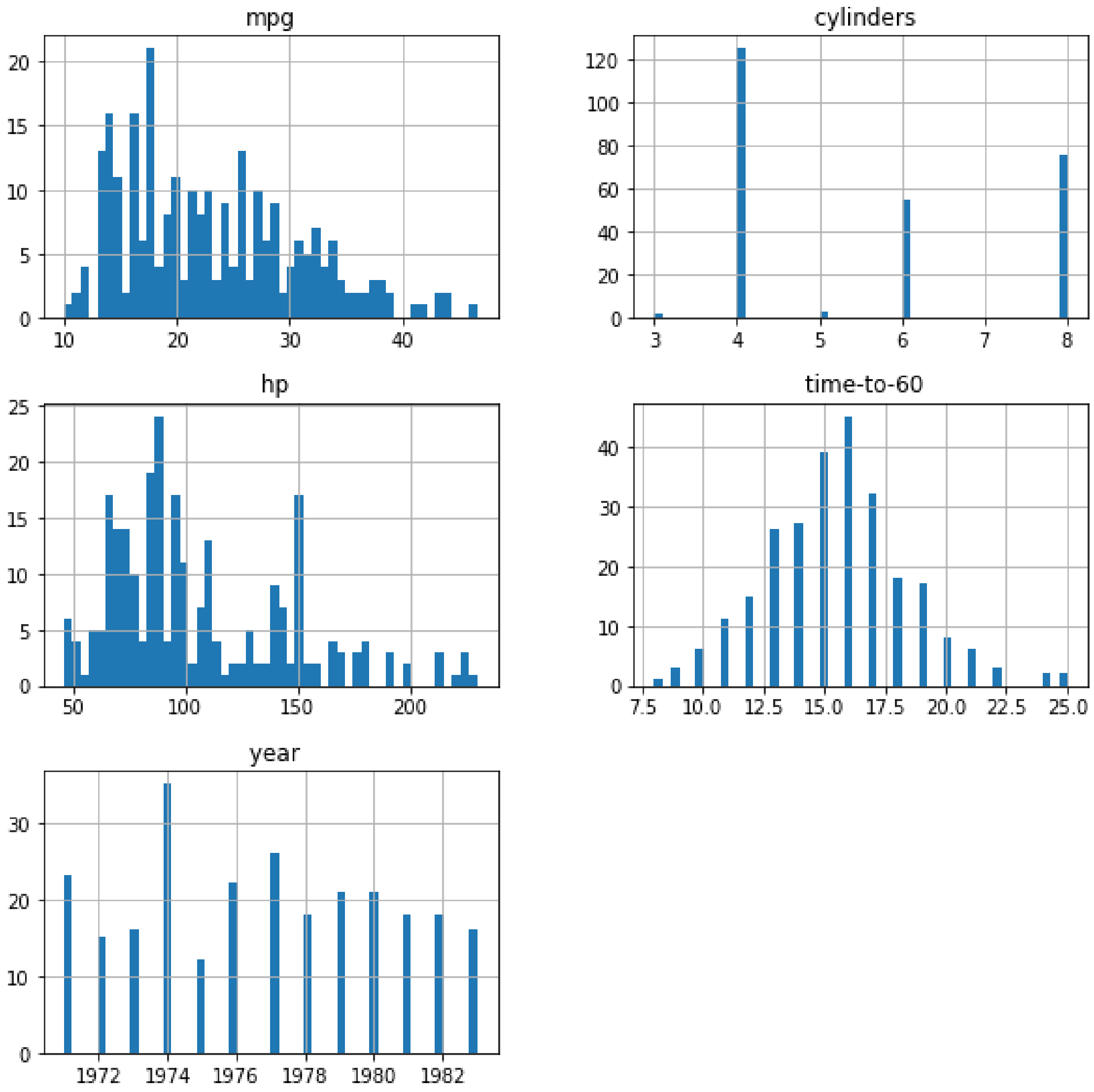
[4]:
```
              mpg   cylinders          hp   time-to-60          year
count  261.000000  261.000000  261.000000   261.000000    261.000000
mean    23.144828    5.590038  106.360153    15.547893   1976.819923
std      7.823570    1.733310   40.499959     2.910625      3.637696
min     10.000000    3.000000   46.000000     8.000000   1971.000000
25%     16.900000    4.000000   75.000000    14.000000   1974.000000
50%     22.000000    6.000000   95.000000    16.000000   1977.000000
75%     28.800000    8.000000  138.000000    17.000000   1980.000000
max     46.600000    8.000000  230.000000    25.000000   1983.000000
```

[5]:
```python
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 261 entries, 0 to 260
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   mpg            261 non-null    float64
 1    cylinders     261 non-null    int64
 2    cubicinches   261 non-null    object
 3    hp            261 non-null    int64
 4    weightlbs     261 non-null    object
 5    time-to-60    261 non-null    int64
 6    year          261 non-null    int64
 7    brand         261 non-null    object
dtypes: float64(1), int64(4), object(3)
memory usage: 16.4+ KB
```

1 EDA

[6]:
```python
_ = df.hist(figsize=(10,10), bins=50)
```



From the above results, there is no need to fill data.

2 Normalize dataset

```
[7]: # fix columns names
     df.columns = list(map(lambda x: x.strip(" "), df.columns))
     df.columns
```

```
[7]: Index(['mpg', 'cylinders', 'cubicinches', 'hp', 'weightlbs', 'time-to-60',
            'year', 'brand'],
           dtype='object')
```

[8]:
```
# Bin age of the car
df["year_bin"] = pd.cut(df["year"], 5)
df["year_bin"].value_counts()
```

[8]:
```
(1975.8, 1978.2]      66
(1970.988, 1973.4]    54
(1980.6, 1983.0]      52
(1973.4, 1975.8]      47
(1978.2, 1980.6]      42
Name: year_bin, dtype: int64
```

[9]:
```
from sklearn.preprocessing import LabelEncoder
```

[10]:
```
le = LabelEncoder()
df["year"] = le.fit_transform(df["year_bin"])
df.drop("year_bin", axis=1, inplace=True)
```

[11]:
```
set(df.brand)
```

[11]:
```
{' Europe.', ' Japan.', ' US.'}
```

[12]:
```
le = LabelEncoder()
df["brand"] = le.fit_transform(df["brand"])
```

[13]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
df = sc.fit_transform(df)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [13], in <cell line: 3>()
      1 from sklearn.preprocessing import StandardScaler
      2 sc = StandardScaler()
----> 3 df = sc.fit_transform(df)

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/sklearn/base.py:867, in␣
  ↪TransformerMixin.fit_transform(self, X, y, **fit_params)
    863 # non-optimized default implementation; override when a better
    864 # method is possible for a given clustering algorithm
    865 if y is None:
    866     # fit method of arity 1 (unsupervised transformation)
--> 867     return self.fit(X, **fit_params).transform(X)
    868 else:
    869     # fit method of arity 2 (supervised transformation)
    870     return self.fit(X, y, **fit_params).transform(X)
```

```
File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/sklearn/preprocessing/
 ↪_data.py:809, in StandardScaler.fit(self, X, y, sample_weight)
    807 # Reset internal state before fitting
    808 self._reset()
--> 809 return self.partial_fit(X, y, sample_weight)

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/sklearn/preprocessing/
 ↪_data.py:844, in StandardScaler.partial_fit(self, X, y, sample_weight)
    812 """Online computation of mean and std on X for later scaling.
    813
    814 All of X is processed as a single batch. This is intended for cases
   (…)
    841     Fitted scaler.
    842 """
    843 first_call = not hasattr(self, "n_samples_seen_")
--> 844 X = self._validate_data(
    845     X,
    846     accept_sparse=("csr", "csc"),
    847     dtype=FLOAT_DTYPES,
    848     force_all_finite="allow-nan",
    849     reset=first_call,
    850 )
    851 n_features = X.shape[1]
    853 if sample_weight is not None:

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/sklearn/base.py:577, in␣
 ↪BaseEstimator._validate_data(self, X, y, reset, validate_separately,␣
 ↪**check_params)
    575     raise ValueError("Validation should be done on X, y or both.")
    576 elif not no_val_X and no_val_y:
--> 577     X = check_array(X, input_name="X", **check_params)
    578     out = X
    579 elif no_val_X and not no_val_y:

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/sklearn/utils/validation
 ↪py:856, in check_array(array, accept_sparse, accept_large_sparse, dtype,␣
 ↪order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples,␣
 ↪ensure_min_features, estimator, input_name)
    854         array = array.astype(dtype, casting="unsafe", copy=False)
    855     else:
--> 856         array = np.asarray(array, order=order, dtype=dtype)
    857 except ComplexWarning as complex_warning:
    858     raise ValueError(
    859         "Complex data not supported\n{}\n".format(array)
    860     ) from complex_warning

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/pandas/core/generic.py:
 ↪2064, in NDFrame.__array__(self, dtype)
```

```
   2063 def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2064     return np.asarray(self._values, dtype=dtype)

ValueError: could not convert string to float: ''
```

[14]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 261 entries, 0 to 260
Data columns (total 8 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   mpg          261 non-null    float64
 1   cylinders    261 non-null    int64
 2   cubicinches  261 non-null    object
 3   hp           261 non-null    int64
 4   weightlbs    261 non-null    object
 5   time-to-60   261 non-null    int64
 6   year         261 non-null    int64
 7   brand        261 non-null    int64
dtypes: float64(1), int64(5), object(2)
memory usage: 16.4+ KB
```

[15]:

```
# cubicinches and weightlbs also may have some strings in the data
```

[16]:

```
import numpy as np
def find_empty_string(data):
    if data == '' or data == ' ':
        return np.nan
    else:
        return data
```

[17]:

```
df.cubicinches = df.cubicinches.apply(find_empty_string)
df.weightlbs = df.weightlbs.apply(find_empty_string)
```

[18]:

```
df.weightlbs.apply(find_empty_string).isna().sum()
```

[18]: 3

[19]:

```
df.cubicinches.apply(find_empty_string).isna().sum()
```

[19]: 2

[24]:

```
df.cubicinches.astype(int)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
```

```
Input In [24], in <cell line: 1>()
----> 1 df.cubicinches.astype(int)

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/pandas/core/generic.py:
 ↪5912, in NDFrame.astype(self, dtype, copy, errors)
   5905     results = [
   5906         self.iloc[:, i].astype(dtype, copy=copy)
   5907         for i in range(len(self.columns))
   5908     ]
   5910 else:
   5911     # else, only a single dtype is given
-> 5912     new_data = self._mgr.astype(dtype=dtype, copy=copy, errors=errors)
   5913     return self._constructor(new_data).__finalize__(self,␣
 ↪method="astype")
   5915 # GH 33113: handle empty frame or series

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/pandas/core/internals/
 ↪managers.py:419, in BaseBlockManager.astype(self, dtype, copy, errors)
    418 def astype(self: T, dtype, copy: bool = False, errors: str = "raise") -␣
 ↪T:
--> 419     return self.apply("astype", dtype=dtype, copy=copy, errors=errors)

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/pandas/core/internals/
 ↪managers.py:304, in BaseBlockManager.apply(self, f, align_keys,␣
 ↪ignore_failures, **kwargs)
    302         applied = b.apply(f, **kwargs)
    303     else:
--> 304         applied = getattr(b, f)(**kwargs)
    305 except (TypeError, NotImplementedError):
    306     if not ignore_failures:

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/pandas/core/internals/
 ↪blocks.py:580, in Block.astype(self, dtype, copy, errors)
    562 """
    563 Coerce to the new dtype.
    564
   (…)
    576 Block
    577 """
    578 values = self.values
--> 580 new_values = astype_array_safe(values, dtype, copy=copy, errors=errors)
    582 new_values = maybe_coerce_values(new_values)
    583 newb = self.make_block(new_values)

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/pandas/core/dtypes/cast.
 ↪py:1292, in astype_array_safe(values, dtype, copy, errors)
   1289     dtype = dtype.numpy_dtype
   1291 try:
```

```
-> 1292     new_values = astype_array(values, dtype, copy=copy)
   1293 except (ValueError, TypeError):
   1294     # e.g. astype_nansafe can fail on object-dtype of strings
   1295     #  trying to convert to float
   1296     if errors == "ignore":

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/pandas/core/dtypes/cast.
 ↪py:1237, in astype_array(values, dtype, copy)
   1234     values = values.astype(dtype, copy=copy)
   1236 else:
-> 1237     values = astype_nansafe(values, dtype, copy=copy)
   1239 # in pandas we don't store numpy str dtypes, so convert to object
   1240 if isinstance(dtype, np.dtype) and issubclass(values.dtype.type, str):

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/pandas/core/dtypes/cast.
 ↪py:1154, in astype_nansafe(arr, dtype, copy, skipna)
   1150 elif is_object_dtype(arr.dtype):
   1151
   1152     # work around NumPy brokenness, #1987
   1153     if np.issubdtype(dtype.type, np.integer):
-> 1154         return lib.astype_intsafe(arr, dtype)
   1156     # if we have a datetime/timedelta array of objects
   1157     # then coerce to a proper dtype and recall astype_nansafe
   1159     elif is_datetime64_dtype(dtype):

File ~/anaconda3/envs/EDIT/lib/python3.9/site-packages/pandas/_libs/lib.pyx:668
 ↪in pandas._libs.lib.astype_intsafe()

ValueError: cannot convert float NaN to integer
```

[25]:

```
data = []
for i in df.cubicinches:
    try:
        data.append(int(i))
    except:
        print(i)
```

```
nan
nan
```

[26]:

```
cilinder_average = sum(data)/len(data)
```

[27]:

```
df.cubicinches = df.cubicinches.fillna(cilinder_average).astype(int)
```

[28]:

```
data = []
for i in df.weightlbs:
    try:
        data.append(int(i))
    except:
        print(i)
```

```
nan
nan
nan
```

```
[29]: weightlbs_average = sum(data)/len(data)
```

```
[30]: weightlbs_average
```

```
[30]: 3009.8333333333335
```

```
[31]: df.weightlbs = df.weightlbs.fillna(weightlbs_average).astype(int)
```

```
[32]: df
```

```
[32]:       mpg  cylinders  cubicinches   hp  weightlbs  time-to-60  year  brand
      0    14.0          8          350  165       4209          12     0      2
      1    31.9          4           89   71       1925          14     3      0
      2    17.0          8          302  140       3449          11     0      2
      3    15.0          8          400  150       3761          10     0      2
      4    30.5          4           98   63       2051          17     2      2
      ..    …          …          …  …    …          …    …     …
      256  17.0          8          305  130       3840          15     3      2
      257  36.1          4           91   60       1800          16     3      1
      258  22.0          6          232  112       2835          15     4      2
      259  18.0          6          232  100       3288          16     0      2
      260  22.0          6          250  105       3353          15     2      2

      [261 rows x 8 columns]
```

```
[33]: sc = StandardScaler()
      df = sc.fit_transform(df)
```

3 Apply TSNE

```
[34]: from sklearn.manifold import TSNE
```

```
[35]: # We want to get TSNE embedding with 2 dimensions
      n_components = 2
      tsne = TSNE(n_components)
      tsne_result = tsne.fit_transform(df)
      tsne_result.shape
```

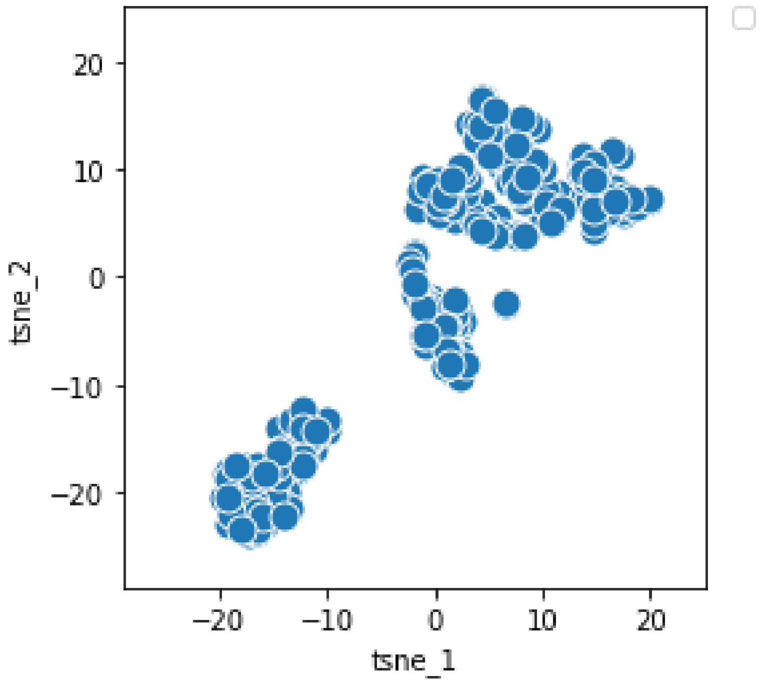
```
/home/local/FARFETCH/tiago.cabo/anaconda3/envs/EDIT/lib/python3.9/site-
packages/sklearn/manifold/_t_sne.py:795: FutureWarning: The default
initialization in TSNE will change from 'random' to 'pca' in 1.2.
  warnings.warn(
/home/local/FARFETCH/tiago.cabo/anaconda3/envs/EDIT/lib/python3.9/site-
packages/sklearn/manifold/_t_sne.py:805: FutureWarning: The default learning
rate in TSNE will change from 200.0 to 'auto' in 1.2.
  warnings.warn(
```

[35]:
```
(261, 2)
```

[36]:
```python
import seaborn as sns
import matplotlib.pyplot as plt
tsne_result_df = pd.DataFrame({'tsne_1': tsne_result[:,0], 'tsne_2':␣
 ↪tsne_result[:,1]})
fig, ax = plt.subplots(1)
sns.scatterplot(x='tsne_1', y='tsne_2', data=tsne_result_df, ax=ax,s=120)
lim = (tsne_result.min()-5, tsne_result.max()+5)
ax.set_xlim(lim)
ax.set_ylim(lim)
ax.set_aspect('equal')
ax.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.0)
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```

[36]:
```
<matplotlib.legend.Legend at 0x7f6a1c700970>
```



From the T-SNE plot, we can see that 3 or 4 clusters may exist

4 Let's Apply k-means

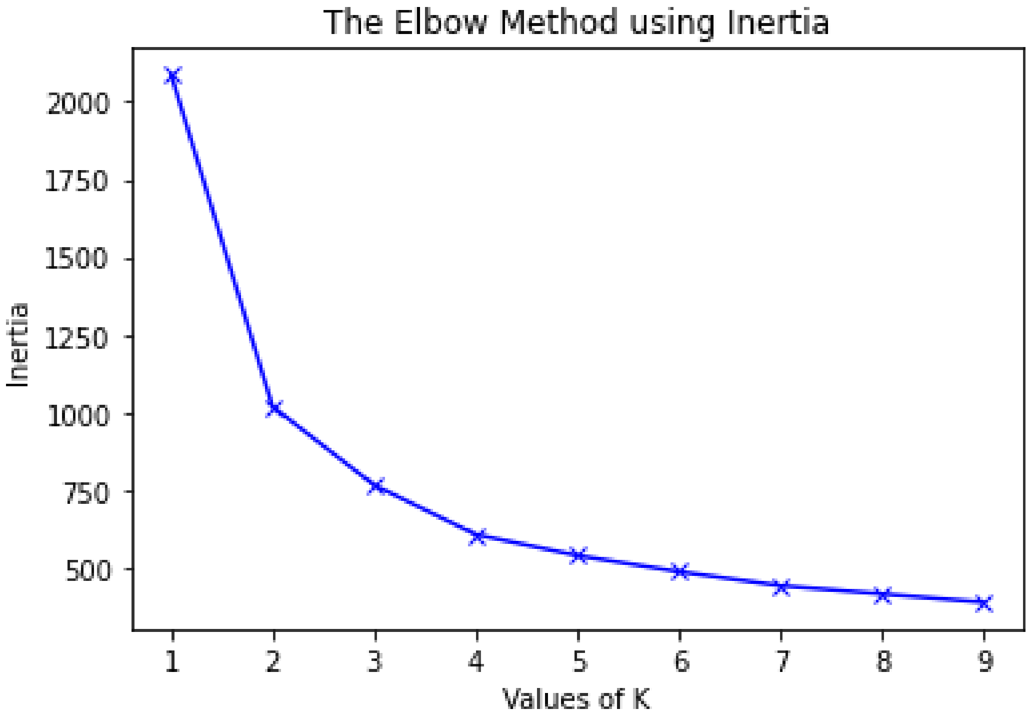
[37]:
```
from sklearn.cluster import KMeans
```

[38]:
```
# Let's just confirm that  the data is normal distributed
pd.DataFrame(df).describe()
```

[38]:
```
                  0             1             2             3             4  \
count  2.610000e+02  2.610000e+02  2.610000e+02  2.610000e+02  2.610000e+02
mean   3.143505e-16  2.124737e-16  8.443650e-17  9.831429e-17 -2.475670e-16
std    1.001921e+00  1.001921e+00  1.001921e+00  1.001921e+00  1.001921e+00
min   -1.683385e+00 -1.497144e+00 -1.223544e+00 -1.493239e+00 -1.648229e+00
25%   -7.997404e-01 -9.191048e-01 -9.197567e-01 -7.758132e-01 -8.918596e-01
50%   -1.466117e-01  2.369740e-01 -4.134443e-01 -2.810368e-01 -1.248703e-01
75%    7.242265e-01  1.393053e+00  9.305848e-01  7.827325e-01  7.719173e-01
max    3.003774e+00  1.393053e+00  2.339054e+00  3.058704e+00  2.344835e+00

                  5             6             7
count  2.610000e+02  2.610000e+02  2.610000e+02
mean  -2.043916e-16  2.911145e-17  1.795073e-16
std    1.001921e+00  1.001921e+00  1.001921e+00
min   -2.598203e+00 -1.402393e+00 -1.833878e+00
25%   -5.328295e-01 -6.888950e-01 -5.574989e-01
50%    1.556284e-01  2.460339e-02  7.188801e-01
75%    4.998573e-01  7.381017e-01  7.188801e-01
max    3.253689e+00  1.451600e+00  7.188801e-01
```

[39]:
```
inertias = []
mapping2 = {}
K = range(1, 10)
for k in K:
    # Building and fitting the model
    kmeanModel = KMeans(n_clusters=k).fit(df)
    kmeanModel.fit(df)

    inertias.append(kmeanModel.inertia_)

    mapping2[k] = kmeanModel.inertia_
```

[40]:
```
plt.plot(K, inertias, 'bx-')
plt.xlabel('Values of K')
plt.ylabel('Inertia')
plt.title('The Elbow Method using Inertia')
plt.show()
```



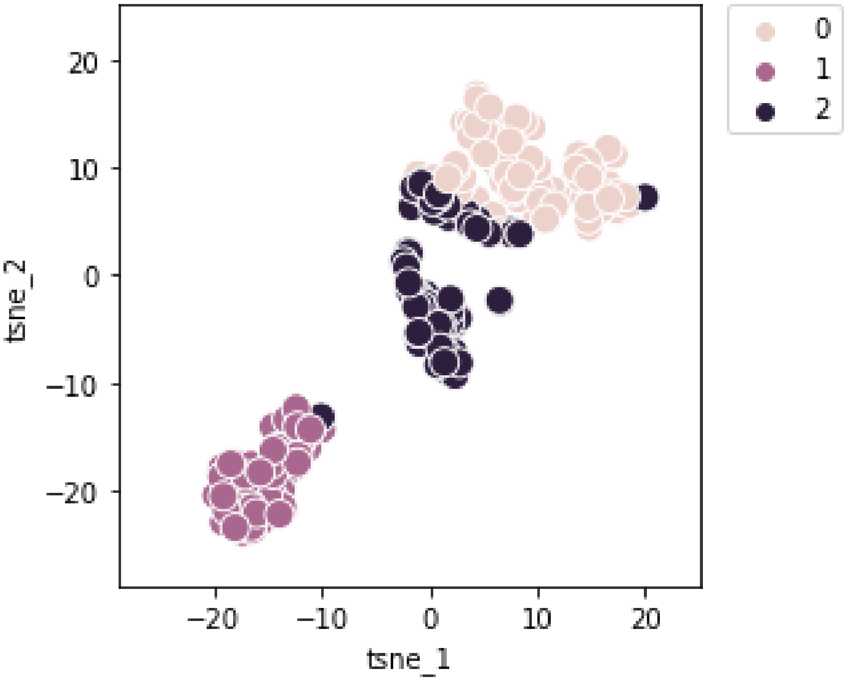
[41]:
```
# plot data with 3 clusters
kmeanModel = KMeans(n_clusters=3).fit(df)
kmeanModel.fit(df)
```

[41]:
```
KMeans(n_clusters=3)
```

[42]:
```
labels = kmeanModel.labels_
```

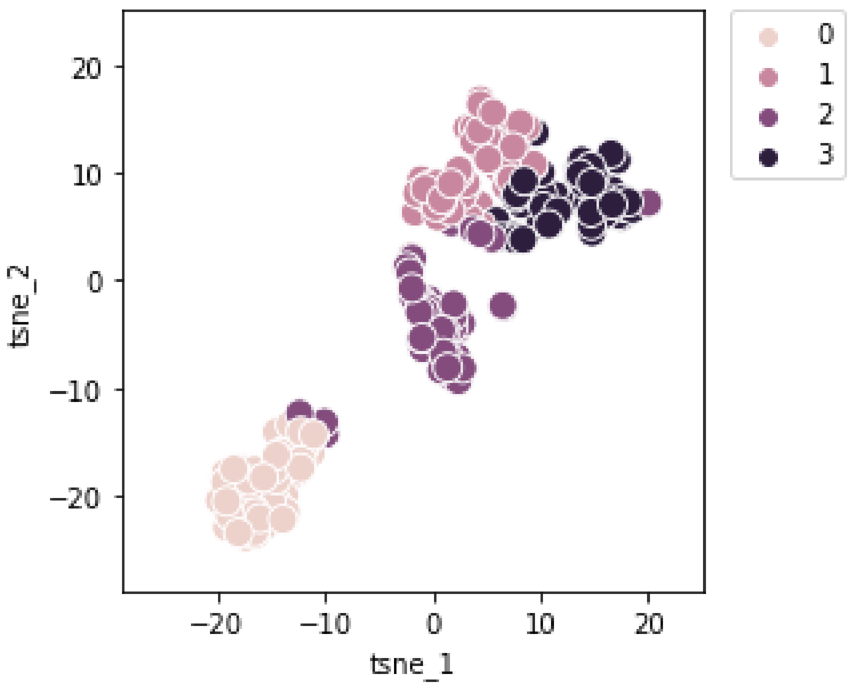
[43]:
```
import seaborn as sns
import matplotlib.pyplot as plt
tsne_result_df = pd.DataFrame({'tsne_1': tsne_result[:,0], 'tsne_2':␣
 ↪tsne_result[:,1], 'label':labels})
fig, ax = plt.subplots(1)
sns.scatterplot(x='tsne_1', y='tsne_2',hue='label', data=tsne_result_df,␣
 ↪ax=ax,s=120)
lim = (tsne_result.min()-5, tsne_result.max()+5)
ax.set_xlim(lim)
ax.set_ylim(lim)
ax.set_aspect('equal')
ax.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.0)
```

```
[43]: <matplotlib.legend.Legend at 0x7f6a1c34c7c0>
```



```
[44]: # Let's try with 4 clusters
      # plot data with 3 clusters
      kmeanModel = KMeans(n_clusters=4).fit(df)
      kmeanModel.fit(df)

      # store results
      labels = kmeanModel.labels_

      tsne_result_df = pd.DataFrame({'tsne_1': tsne_result[:,0], 'tsne_2':␣
       ↪tsne_result[:,1], 'label':labels})
      fig, ax = plt.subplots(1)
      sns.scatterplot(x='tsne_1', y='tsne_2',hue='label', data=tsne_result_df,␣
       ↪ax=ax,s=120)
      lim = (tsne_result.min()-5, tsne_result.max()+5)
      ax.set_xlim(lim)
      ax.set_ylim(lim)
      ax.set_aspect('equal')
      ax.legend(bbox_to_anchor=(1.05, 1), loc=2, borderaxespad=0.0)
```

```
[44]: <matplotlib.legend.Legend at 0x7f6a1c27e130>
```


20
10
0
−10
−20
tsne_2
−20
−10
0
10
20
tsne_1
0
1
2
3