UIC
BUSINESS

Business Data Mining

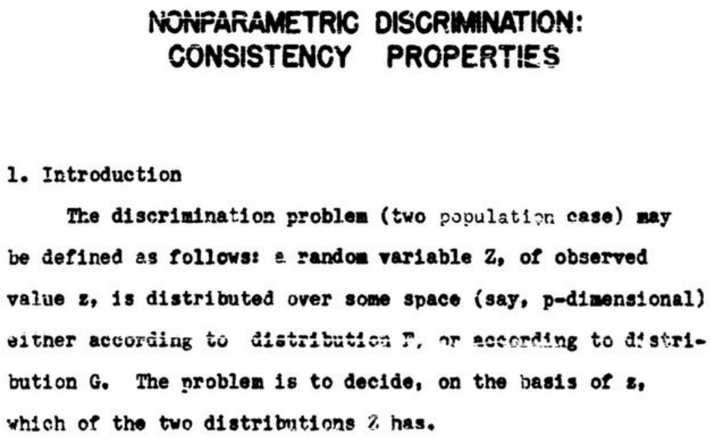
IDS 472 (Spring 2024)

Instructor: Wenxin Zhou

k-Nearest Neighbors

UIC BUSINESS

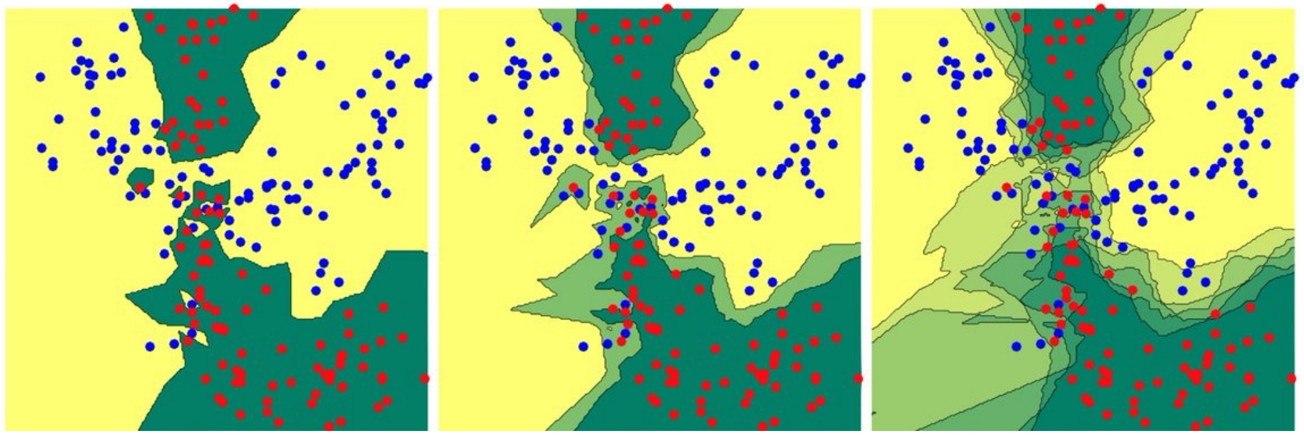
- In this chapter, we formalize the kNN mechanism for both classification and regression



NONPARAMETRIC DISCRIMINATION:
CONSISTENCY PROPERTIES

1. Introduction

The discrimination problem (two population case) may be defined as follows: a random variable Z, of observed value z, is distributed over some space (say, p-dimensional) either according to distribution F, or according to distribution G. The problem is to decide, on the basis of z, which of the two distributions Z has.



Classification

UIC BUSINESS

- Binary classification: $y \in \{0, 1\}$
- Training set: $\mathbf{S}_{tr} = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_n, y_n)\}$
- Given a feature vector $\mathbf{x}$ (test observation), denote the i^{th} nearest observation to $\mathbf{x}$ as $\mathbf{x}_{(i)}(\mathbf{x})$ with label $y_{(i)}(\mathbf{x})$
- Standard kNN classifier $\psi(\mathbf{x})$ is given by

$$\hat{y} = \psi(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_{i=1}^{k} \frac{1}{k} I_{\{y_{(i)}(\mathbf{x})=1\}} > \sum_{i=1}^{k} \frac{1}{k} I_{\{y_{(i)}(\mathbf{x})=0\}}\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

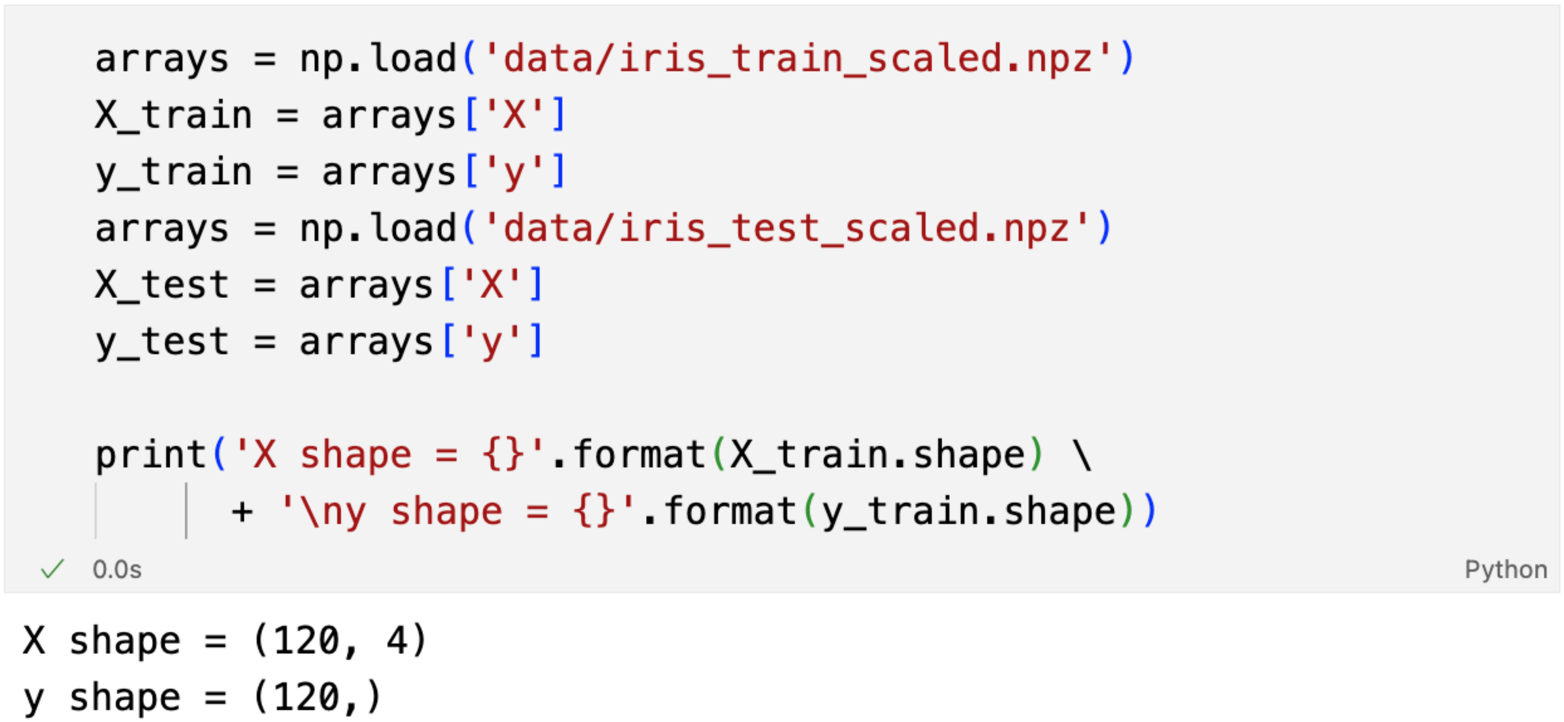
  where k is the number of nearest neighbors wrt $\mathbf{x}$
- For multiclass classification with c classes, $y \in \{0, 1, \ldots, c-1\}$, the kNN classifier is

$$\hat{y} = \psi(\mathbf{x}) = \underset{j}{\operatorname{argmax}} \sum_{i=1}^{k} \frac{1}{k} I_{\{y_{(i)}(\mathbf{x})=j\}}$$

Classification

UIC BUSINESS

- Avoid an even k to prevent ties (3NN, 5NN)
- The larger k, the smoother are the *decision boundaries* (boundaries between *decision regions*)
- Below we examine the effect of k on the decision boundaries and accuracy of kNN on scaled Iris data



```
arrays = np.load('data/iris_train_scaled.npz')
X_train = arrays['X']
y_train = arrays['y']
arrays = np.load('data/iris_test_scaled.npz')
X_test = arrays['X']
y_test = arrays['y']

print('X shape = {}'.format(X_train.shape) \
        + '\ny shape = {}'.format(y_train.shape))
```
✓ 0.0s Python

```
X shape = (120, 4)
y shape = (120,)
```

Classification

UIC BUSINESS

- For illustration purposes, consider the first two features in data

```
X_train = X_train[:,[0,1]]
X_test = X_test[:,[0,1]]
X_train.shape
```

```
(120, 2)
```

- Implement 4 kNN classifiers for $k = 1, 3, 9, 36$
- Visualize the resulting decision regions:
  - great a grid
  - train a classifier using training data
  - classify each point in the grid using trained classifier
  - plot the decision regions based on the assigned labels to each point in the grid

Classification

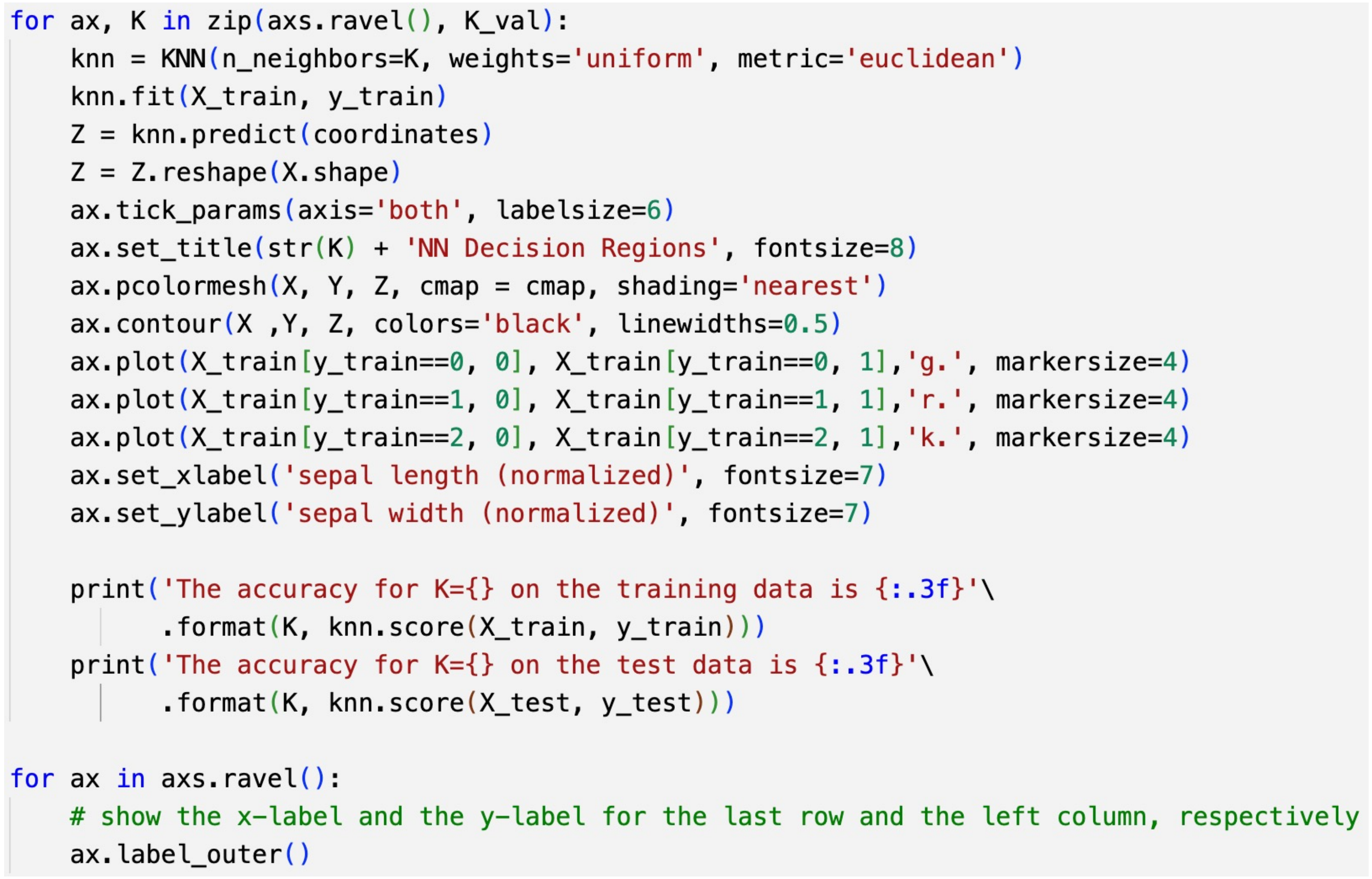
UIC
BUSINESS

import required functionality for this chapter

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
from sklearn.neighbors import KNeighborsClassifier as KNN
```

```
color = ['aquamarine', 'bisque', 'lightgrey']
cmap = ListedColormap(color)

mins = X_train.min(axis=0) - 0.1
maxs = X_train.max(axis=0) + 0.1
x = np.arange(mins[0], maxs[0], 0.01)
y = np.arange(mins[1], maxs[1], 0.01)
X, Y = np.meshgrid(x, y)
coordinates = np.array([X.ravel(), Y.ravel()]).T

fig, axs = plt.subplots(2, 2, figsize=(6, 4), dpi = 200)
fig.tight_layout()
K_val = [1, 3, 9, 36]
```

Classification



```python
for ax, K in zip(axs.ravel(), K_val):
    knn = KNN(n_neighbors=K, weights='uniform', metric='euclidean')
    knn.fit(X_train, y_train)
    Z = knn.predict(coordinates)
    Z = Z.reshape(X.shape)
    ax.tick_params(axis='both', labelsize=6)
    ax.set_title(str(K) + 'NN Decision Regions', fontsize=8)
    ax.pcolormesh(X, Y, Z, cmap = cmap, shading='nearest')
    ax.contour(X ,Y, Z, colors='black', linewidths=0.5)
    ax.plot(X_train[y_train==0, 0], X_train[y_train==0, 1],'g.', markersize=4)
    ax.plot(X_train[y_train==1, 0], X_train[y_train==1, 1],'r.', markersize=4)
    ax.plot(X_train[y_train==2, 0], X_train[y_train==2, 1],'k.', markersize=4)
    ax.set_xlabel('sepal length (normalized)', fontsize=7)
    ax.set_ylabel('sepal width (normalized)', fontsize=7)

    print('The accuracy for K={} on the training data is {:.3f}'\
          .format(K, knn.score(X_train, y_train)))
    print('The accuracy for K={} on the test data is {:.3f}'\
          .format(K, knn.score(X_test, y_test)))

for ax in axs.ravel():
    # show the x-label and the y-label for the last row and the left column, respectively
    ax.label_outer()
```

Classification

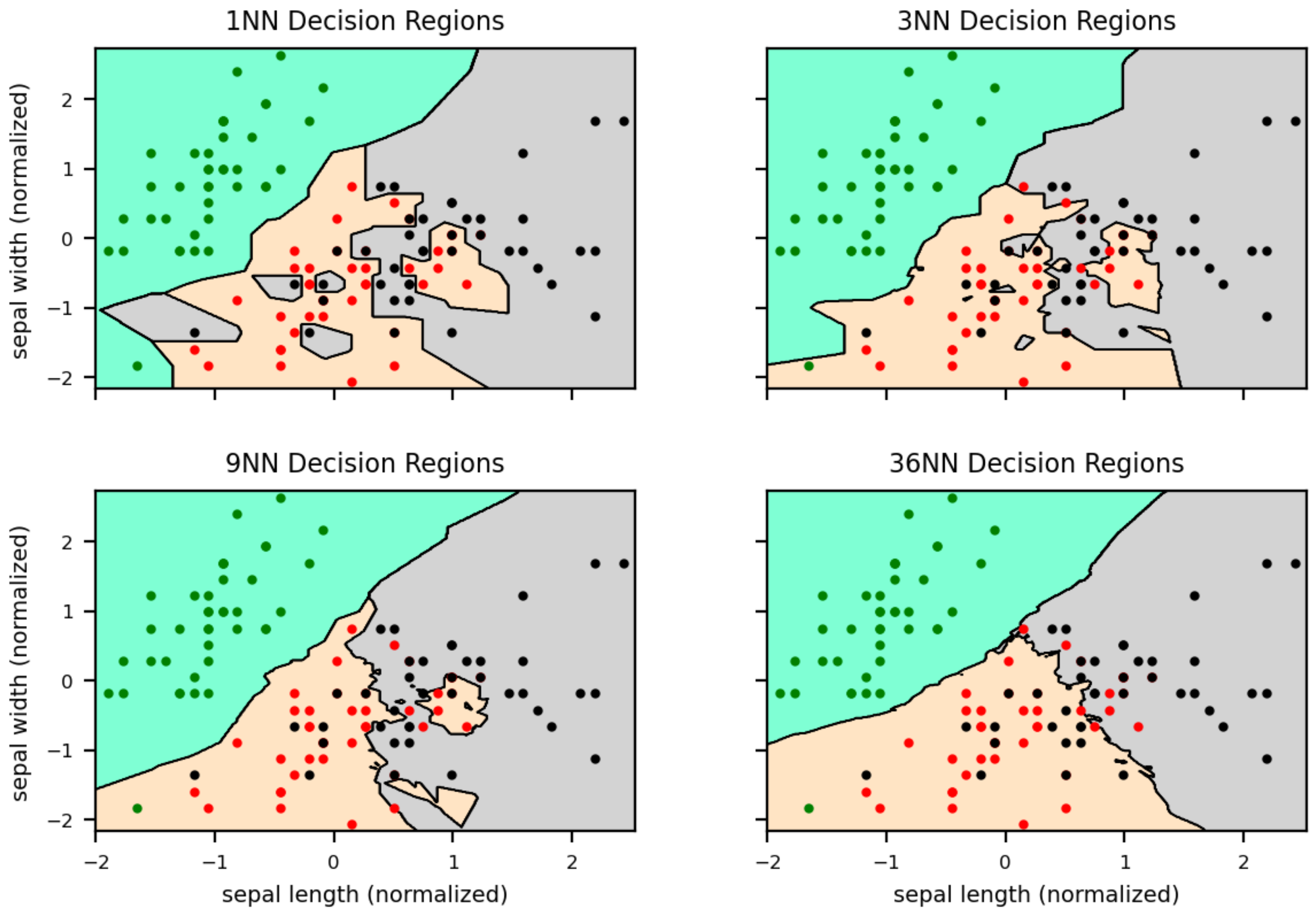
similarly to interpolator: low bias, high variance

```
The accuracy for K=1 on the training data is 0.933
The accuracy for K=1 on the test data is 0.667
The accuracy for K=3 on the training data is 0.833
The accuracy for K=3 on the test data is 0.767
The accuracy for K=9 on the training data is 0.858
The accuracy for K=9 on the test data is 0.800
The accuracy for K=36 on the training data is 0.775
The accuracy for K=36 on the test data is 0.800
```

similarly to a smooth curve: larger bias, low variance

Classification

UIC BUSINESS



Distance-weighted kNN

UIC BUSINESS

- Another form of kNN is distance-weighted kNN (DW-kNN)
  - the k nearest neighbors of a test observation $\mathbf{x}$ are weighted according to their distances from $\mathbf{x}$
  - observations that are closer to $\mathbf{x}$ should impose a higher influence on decision making
  - use the inverse of distance to $\mathbf{x}$ as weight

$$\psi(\mathbf{x}) = \begin{cases} 1 & \text{if } \frac{1}{D}\sum_{i=1}^{k} \frac{1}{d[\mathbf{x}_{(i)}(\mathbf{x}), \mathbf{x}]} I_{\{y_{(i)}(\mathbf{x})=1\}} > \frac{1}{D}\sum_{i=1}^{k} \frac{1}{d[\mathbf{x}_{(i)}(\mathbf{x}), \mathbf{x}]} I_{\{y_{(i)}(\mathbf{x})=0\}}\,, \\ 0 & \text{otherwise}\,, \end{cases}$$

$$D = \sum_{i=1}^{k} \frac{1}{d[\mathbf{x}_{(i)}(\mathbf{x}), \mathbf{x}]}$$

where $d[\mathbf{x}_{(i)}(\mathbf{x}), \mathbf{x}]$ is the distance of $\mathbf{x}_{(i)}(\mathbf{x})$ from $\mathbf{x}$

- In practice, the choice of using DW-kNN, standard kNN, or the choice of k will be decided in the model selection phase

Distance-weighted kNN

UIC BUSINESS

- DW-kNN is implemented by setting `weights='distance'` in `KNeighborsClassifier` (default is `'uniform'`)

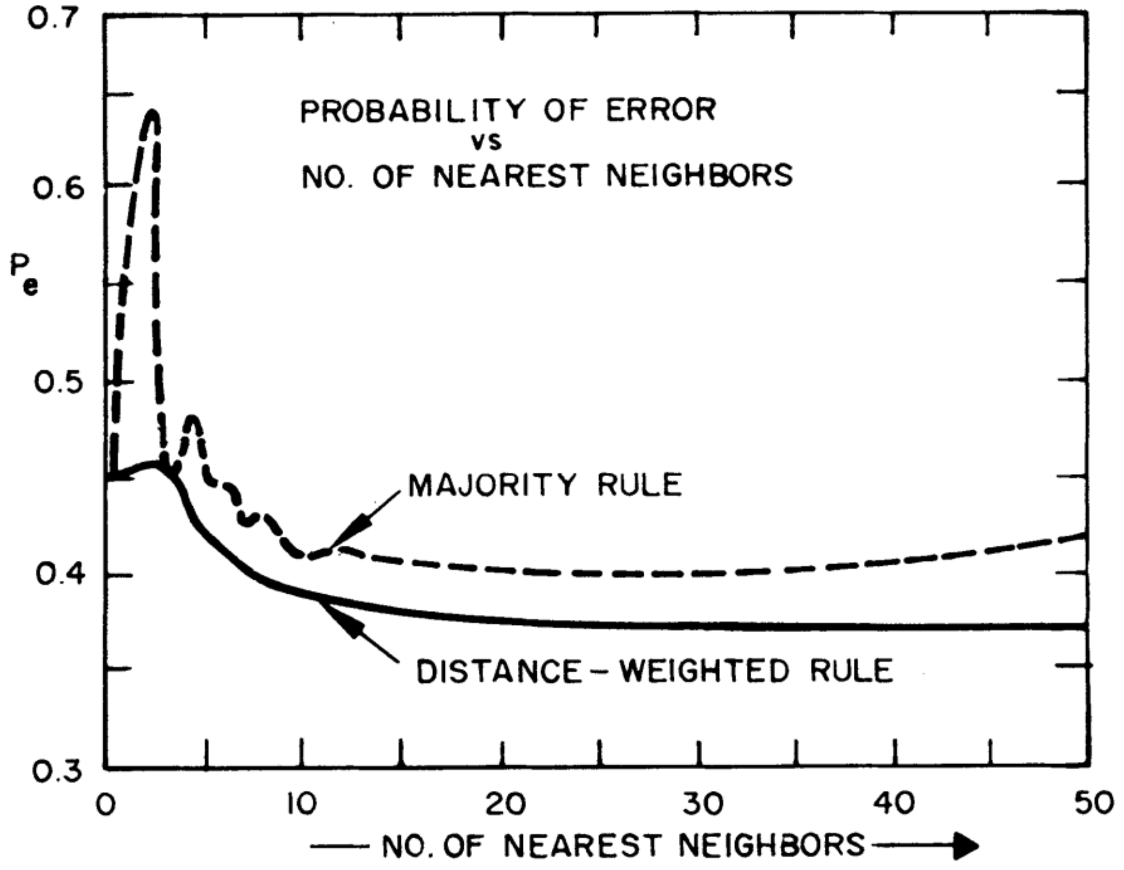
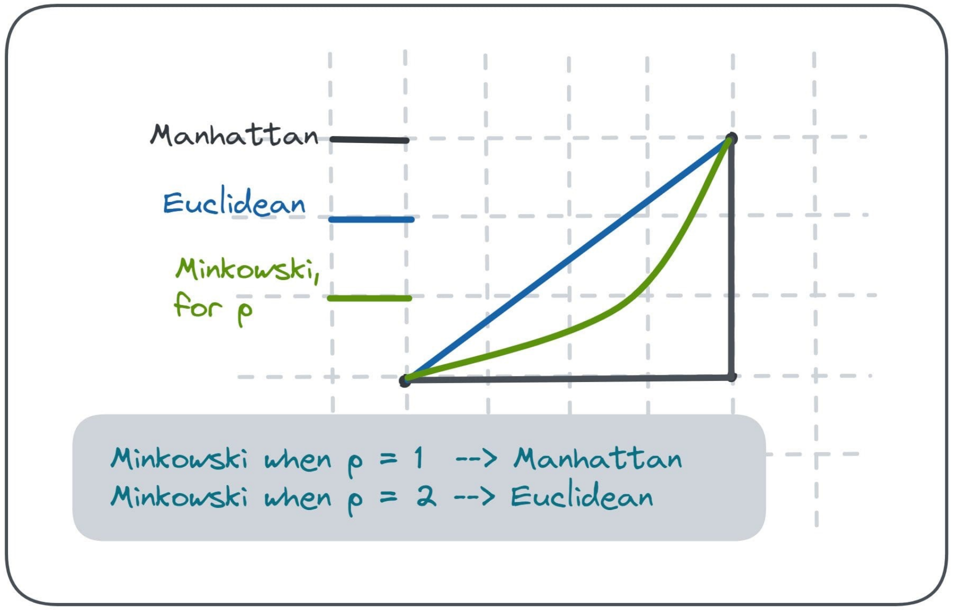


Fig. 1. Plots of probability of error with respect to *k* for the two nearest-neighbor type rules.

Choice of Distance

UIC BUSINESS

- Let $\mathbf{x}_i = [x_{1i}, x_{2i}, \ldots, x_{qi}]^T$ and $\mathbf{x}_j = [x_{1j}, x_{2j}, \ldots, x_{qj}]^T$
- Three popular choices of distances are Euclidean, Manhattan and Minkowski


Manhattan
Euclidean
Minkowski, for p
Minkowski when p = 1 --> Manhattan
Minkowski when p = 2 --> Euclidean

$$d_{\mathrm{E}}[\mathbf{x}_i, \mathbf{x}_j] = \sqrt{\sum_{l=1}^{q} (x_{li} - x_{lj})^2}$$

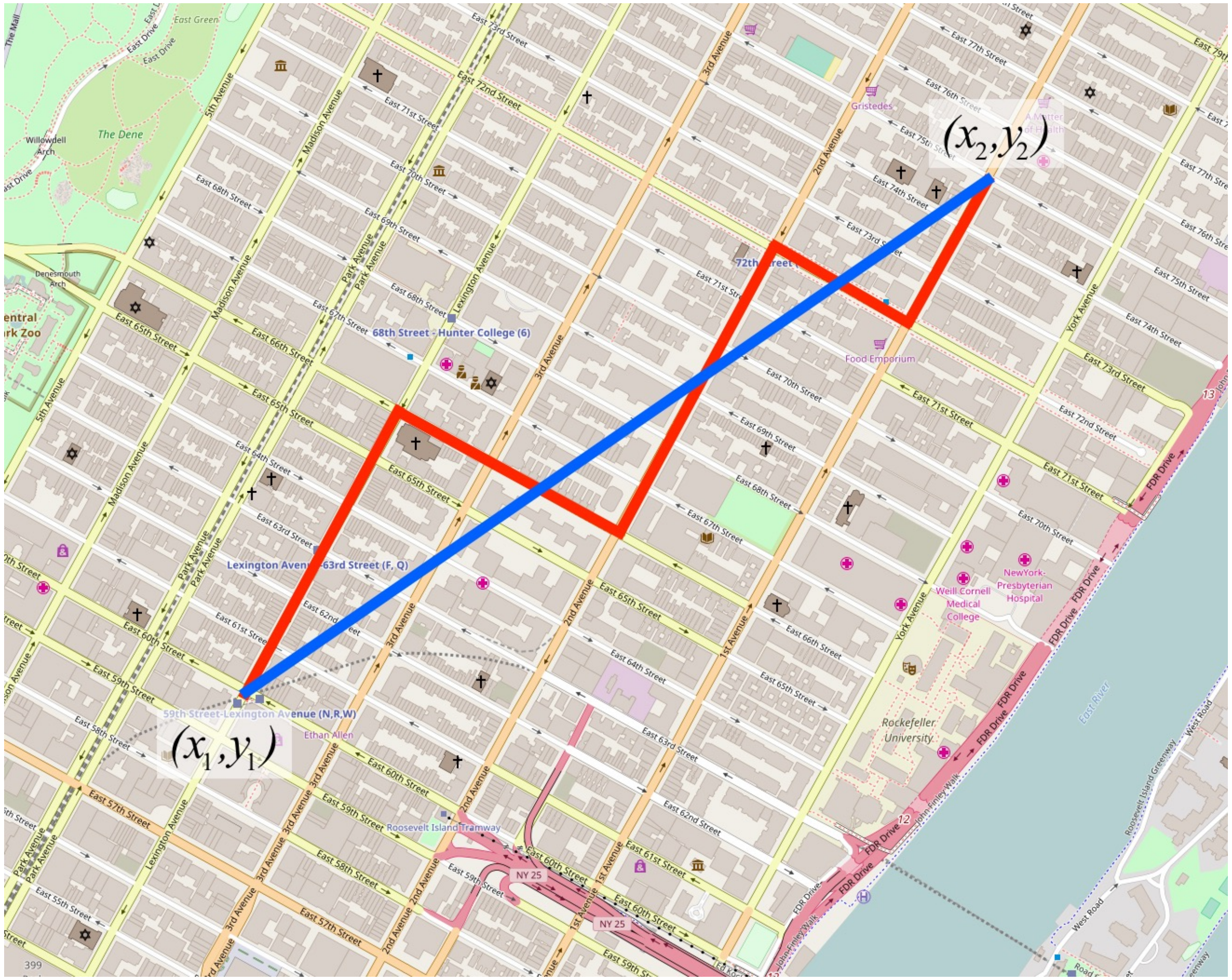
$$d_{\mathrm{Ma}}[\mathbf{x}_i, \mathbf{x}_j] = \sum_{l=1}^{q} |x_{li} - x_{lj}|$$

$$d_{\mathrm{Mi}}[\mathbf{x}_i, \mathbf{x}_j] = \Big(\sum_{l=1}^{q} |x_{li} - x_{lj}|^p \Big)^{\frac{1}{p}}$$

- In `KNeighborsClassifier`, the choice of distance is determined by the `metric` parameter

Manhattan vs Euclidean distance

UIC BUSINESS



Regression

UIC BUSINESS

- The kNN regressor $f(\mathbf{x})$ is given by

$$\hat{y} = f(\mathbf{x}) = \sum_{i=1}^{k} \frac{1}{k} y_{(i)}(\mathbf{x})$$

- The standard kNN regressor estimates the target of a given $\mathbf{x}$ as the average of k targets of the nearest neighbors of $\mathbf{x}$

- Hereafter, denote regressors simply as $f(\mathbf{x})$

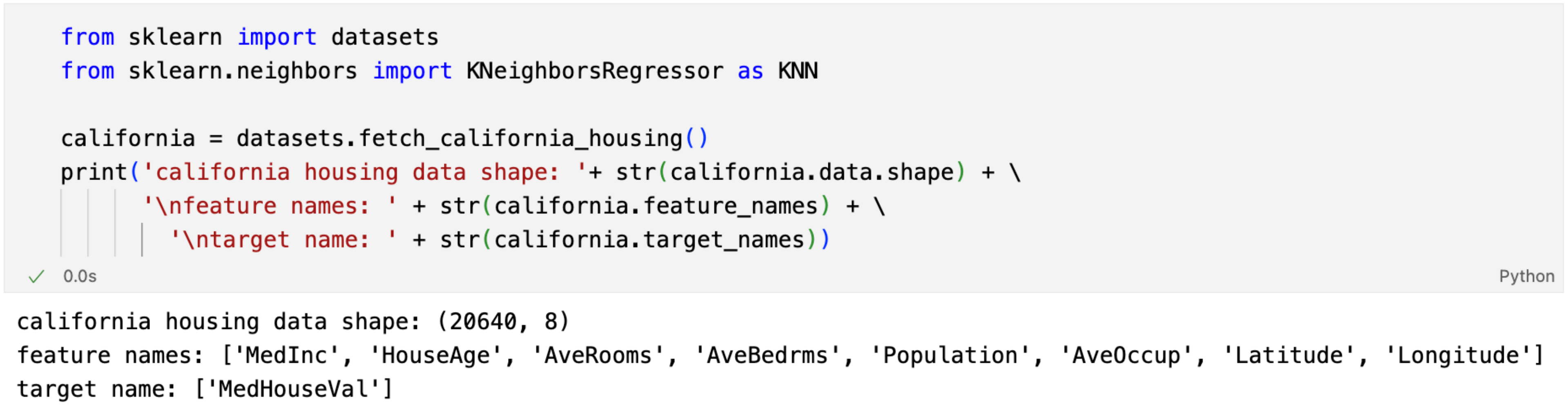
- In this course, a regressor refers to an estimator of the *conditional mean function*

$$E(Y|\mathbf{X} = \mathbf{x}), \mathbf{x} \in \mathbb{R}^{\boldsymbol{p}}$$

A Regression Application

UIC BUSINESS

- California Housing dataset
  - median house price (in $100,000) of 20640 CA districts
  - use 8 features to predict the house price
  - the target is the median house price of a district ('MedHouseVal' in the data)



```python
from sklearn import datasets
from sklearn.neighbors import KNeighborsRegressor as KNN

california = datasets.fetch_california_housing()
print('california housing data shape: '+ str(california.data.shape) + \
      '\nfeature names: ' + str(california.feature_names) + \
        '\ntarget name: ' + str(california.target_names))
```

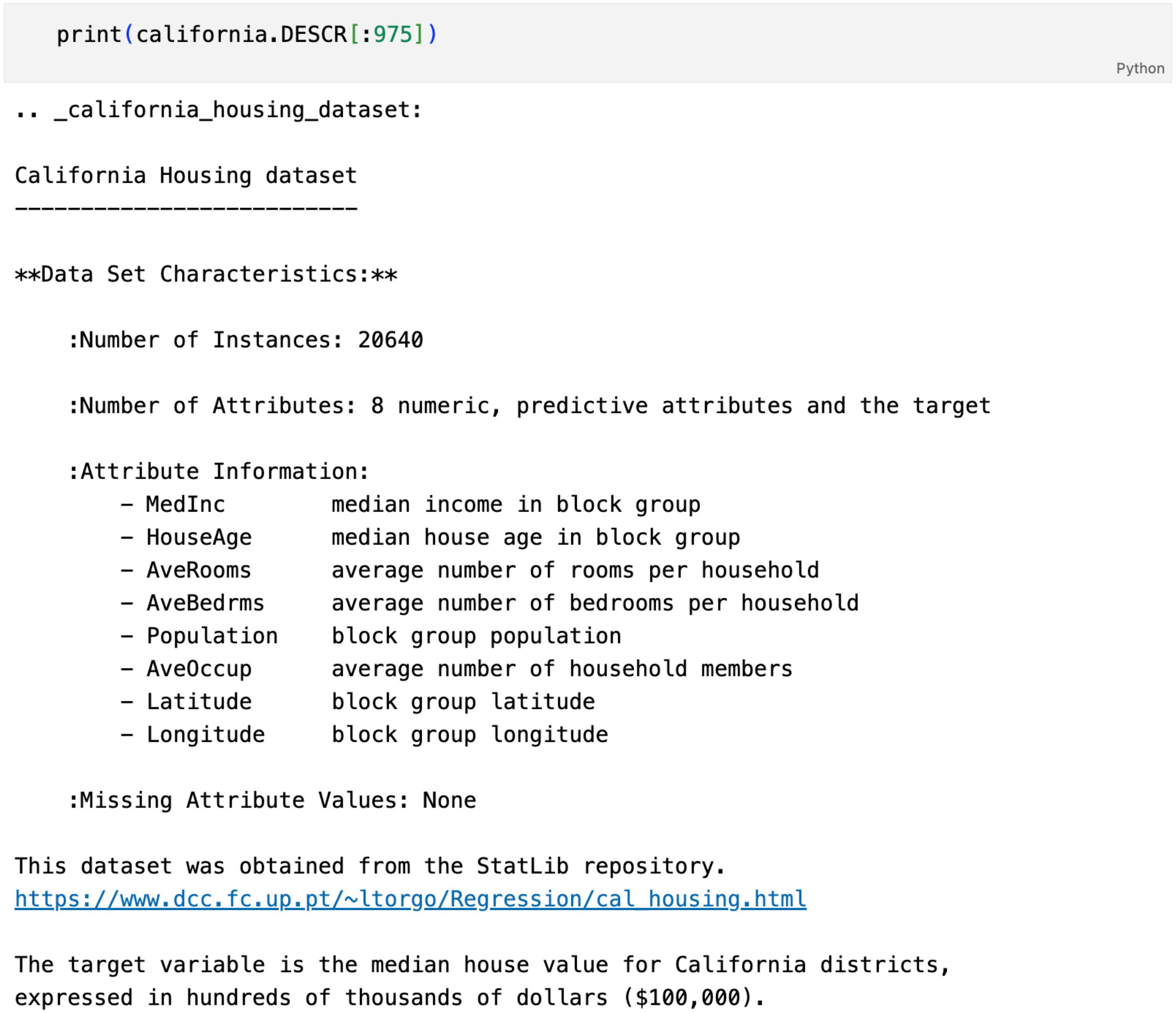
✓ 0.0s Python

```
california housing data shape: (20640, 8)
feature names: ['MedInc', 'HouseAge', 'AveRooms', 'AveBedrms', 'Population', 'AveOccup', 'Latitude', 'Longitude']
target name: ['MedHouseVal']
```

A Regression Application

UIC BUSINESS

- Use the `DESCR` key to check some details about the dataset



```python
print(california.DESCR[:975])
```

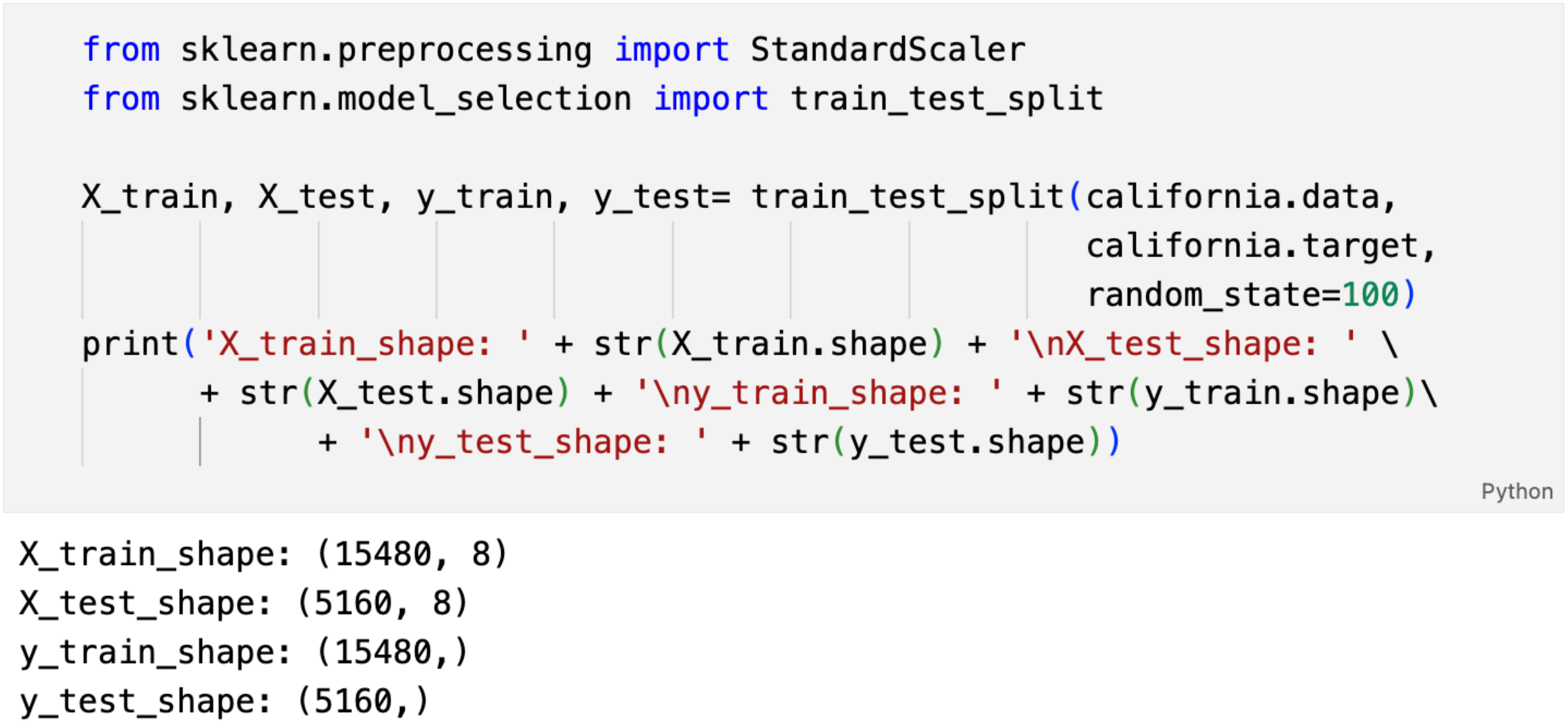
Python

```
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

    :Number of Instances: 20640

    :Number of Attributes: 8 numeric, predictive attributes and the target

    :Attribute Information:
        - MedInc        median income in block group
        - HouseAge      median house age in block group
        - AveRooms      average number of rooms per household
        - AveBedrms     average number of bedrooms per household
        - Population    block group population
        - AveOccup      average number of household members
        - Latitude      block group latitude
        - Longitude     block group longitude

    :Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).
```

Preprocessing

UIC BUSINESS

- Split the data into training and test sets: use the default 0.25 `test_size` and do not set `stratify` to any variable (there are no “classes” in regression)



```python
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test= train_test_split(california.data,
                                                   california.target,
                                                   random_state=100)
print('X_train_shape: ' + str(X_train.shape) + '\nX_test_shape: ' \
      + str(X_test.shape) + '\ny_train_shape: ' + str(y_train.shape)\
            + '\ny_test_shape: ' + str(y_test.shape))
```
Python

```
X_train_shape: (15480, 8)
X_test_shape: (5160, 8)
y_train_shape: (15480,)
y_test_shape: (5160,)
```

Preprocessing

- Use standardization to scale the training and test sets: the scaler object is trained using the training set, and is then used to transform both training and test sets

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

- Previously we used pair plots as an exploratory analysis.
- Another exploratory analysis is to look into *Pearson's correlation coefficients* between predictors and the response or even between predictors themselves
- The Pearson's correlation coefficient between two rv X and Y is

$$\rho = \frac{\mathrm{Cov}[X,Y]}{\sigma_X \sigma_Y} \qquad \mathrm{Cov}[X,Y] = E\left[(X - E[X])(Y - E[Y])\right]$$

Exploratory Analysis

UIC BUSINESS

- Given a sample $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ from (X, Y). The sample Pearson's correlation coefficient is

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}}$$

  - sample correlation coefficient is scale and location invariant
  - it is a measure of the degree of *linear relationship*
  - $-1 \leq \rho \leq 1, -1 \leq r \leq 1$
  - $|\rho| < 1$ indicates the relationship is not completely linear

- `pandas.DataFrame.corr()` calculates pairwise correlation coefficients

Exploratory Analysis

UIC BUSINESS

```python
# np.concatenate((a1, a2)) concatenate arrays
# a1 and a2 along the specified axis
california_arr = np.concatenate((X_train_scaled, y_train.reshape(-1,1)),
                                axis=1)
california_pd = pd.DataFrame(california_arr,
                             columns=[*california.feature_names, 'MEDV'])
california_pd.head().round(3)
```

Python

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MEDV
0	1.604	-0.843	0.974	-0.086	0.068	0.027	-0.865	0.883	2.903
1	-0.921	0.345	-0.197	-0.238	-0.472	-0.068	1.647	-0.999	0.687
2	-0.809	1.849	-0.376	-0.037	-0.516	-0.082	1.675	-0.741	1.097
3	0.597	-0.289	-0.437	-0.119	-0.680	-0.121	1.008	-1.423	4.600
4	0.219	0.107	0.187	0.122	-0.436	-0.057	0.956	-1.283	2.134

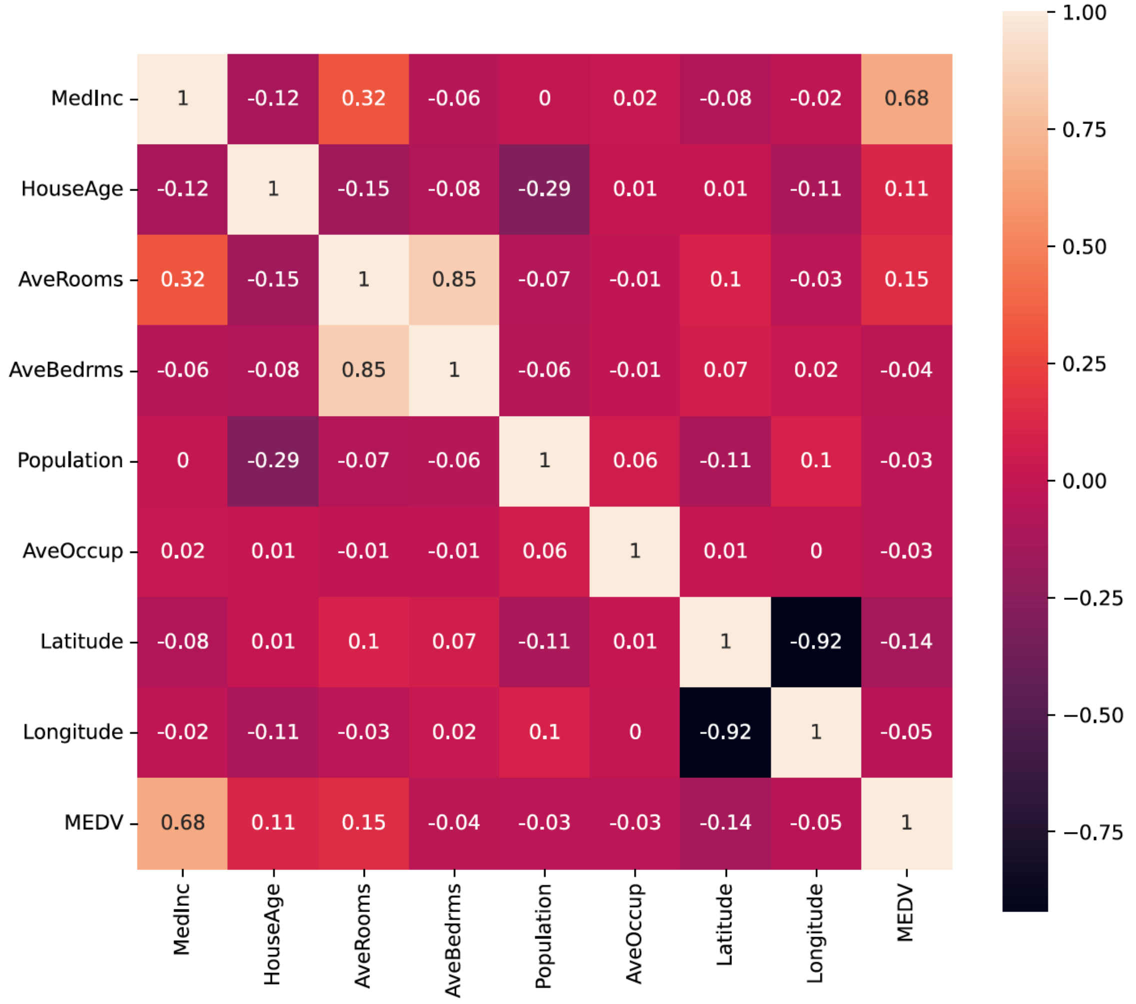
```python
fig, ax = plt.subplots(figsize=(9,5))
sns.heatmap(california_pd.corr().round(2), annot=True, square=True, ax=ax)
```

Python

Exploratory Analysis

UIC BUSINESS

- Calculate and draw a color-coded plot (heatmap) of these correlation coefficients using `seaborn.heatmap()`



	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude	MEDV
MedInc	1	-0.12	0.32	-0.06	0	0.02	-0.08	-0.02	0.68
HouseAge	-0.12	1	-0.15	-0.08	-0.29	0.01	0.01	-0.11	0.11
AveRooms	0.32	-0.15	1	0.85	-0.07	-0.01	0.1	-0.03	0.15
AveBedrms	-0.06	-0.08	0.85	1	-0.06	-0.01	0.07	0.02	-0.04
Population	0	-0.29	-0.07	-0.06	1	0.06	-0.11	0.1	-0.03
AveOccup	0.02	0.01	-0.01	-0.01	0.06	1	0.01	0	-0.03
Latitude	-0.08	0.01	0.1	0.07	-0.11	0.01	1	-0.92	-0.14
Longitude	-0.02	-0.11	-0.03	0.02	0.1	0	-0.92	1	-0.05
MEDV	0.68	0.11	0.15	-0.04	-0.03	-0.03	-0.14	-0.05	1

Exploratory Analysis

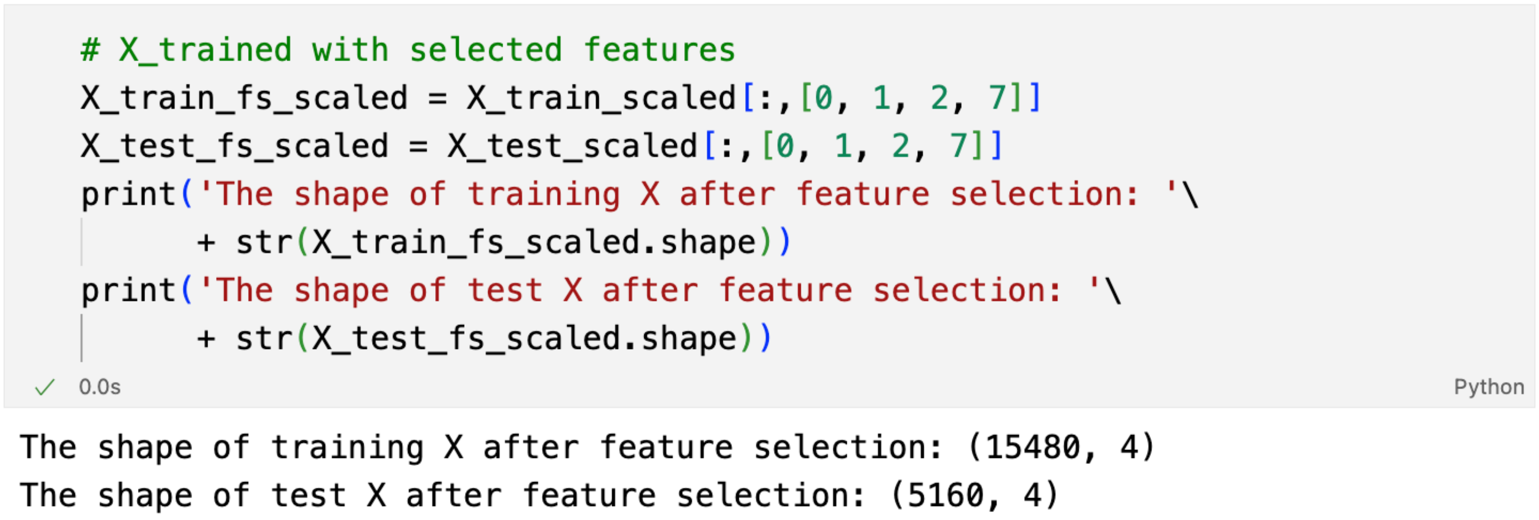
UIC BUSINESS

- One way that we may use this exploratory analysis is in *feature selection*
- In many applications with a limited sample size and moderate to large number of features, using a subset of features could lead to a better performance in predicting the target than using all features
- This is due to what is known as the "curse of dimensionality"
- Using more features in training not only increases the computational burden, but also could lead to a lower performance of the trained models after adding more than a certain number of features
- We may use the correlation matrix to identify a subset of features such that each feature within this subset is strongly or moderately correlated with the response

Exploratory Analysis

UIC BUSINESS

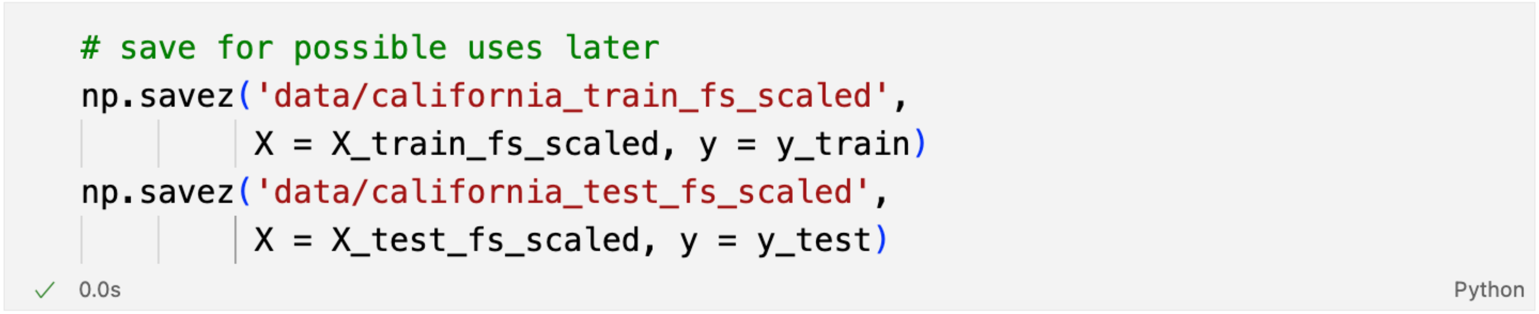
- Defining a strong or moderate correlation is subjective
- For simplicity, we choose variables with a correlation magnitude ≥ 0.1: `MedInc`, `HouseAge`, `AveRooms`, `Latitude`



```
# X_trained with selected features
X_train_fs_scaled = X_train_scaled[:,[0, 1, 2, 7]]
X_test_fs_scaled = X_test_scaled[:,[0, 1, 2, 7]]
print('The shape of training X after feature selection: '\
        + str(X_train_fs_scaled.shape))
print('The shape of test X after feature selection: '\
        + str(X_test_fs_scaled.shape))
```

✓ 0.0s Python

```
The shape of training X after feature selection: (15480, 4)
The shape of test X after feature selection: (5160, 4)
```



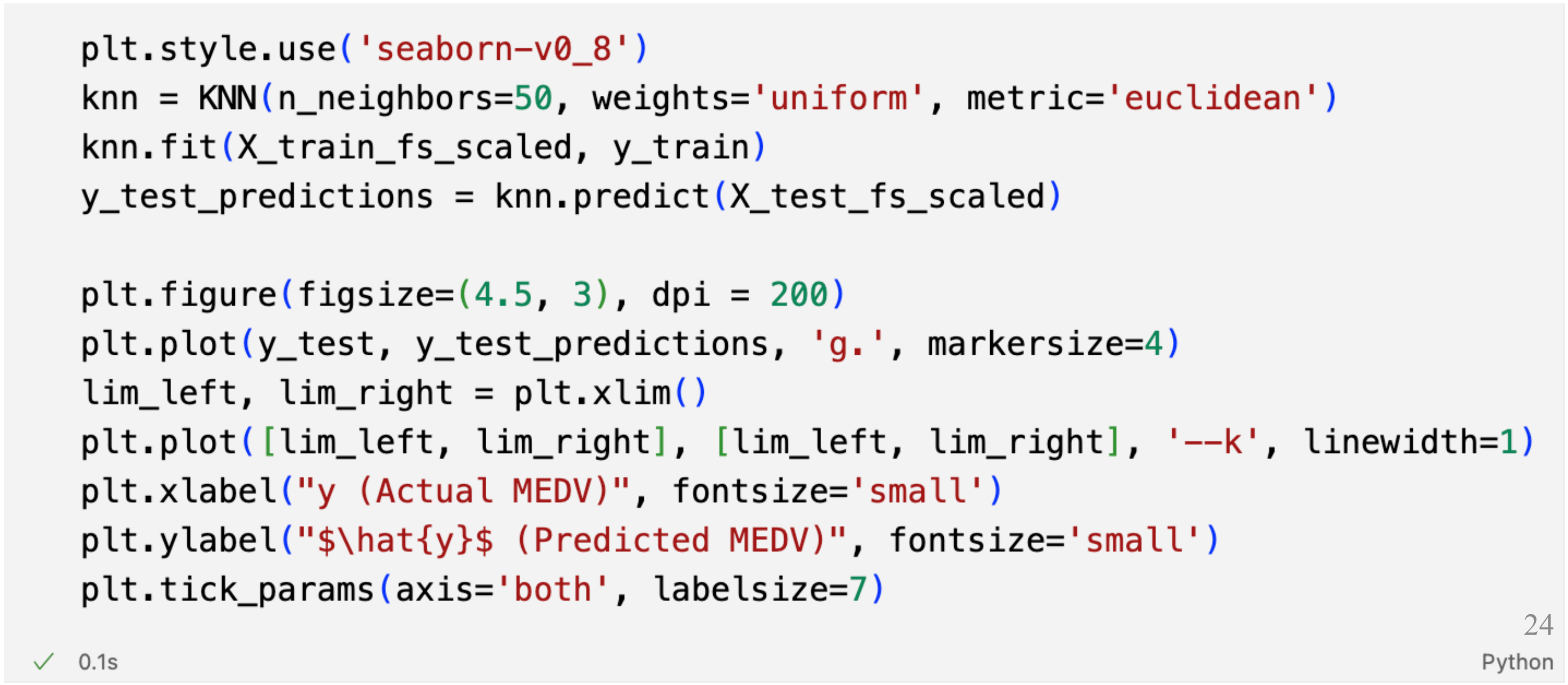
```
# save for possible uses later
np.savez('data/california_train_fs_scaled',
            X = X_train_fs_scaled, y = y_train)
np.savez('data/california_test_fs_scaled',
            X = X_test_fs_scaled, y = y_test)
```

✓ 0.0s Python

Model Training & Evaluation

UIC BUSINESS

- The standard kNN regressor is implemented in the `KNeighborsRegressor` class from `sklearn.neighbors` module
- Train a standard 50NN (Euclidean distance & uniform weights) using the scaled training set (`X_train_fs_scaled`), then use the model to predict the target in the test set (`X_test_fs_scaled`)
- Create scatter plot between predicted targets $\hat{y}$ and actual targets

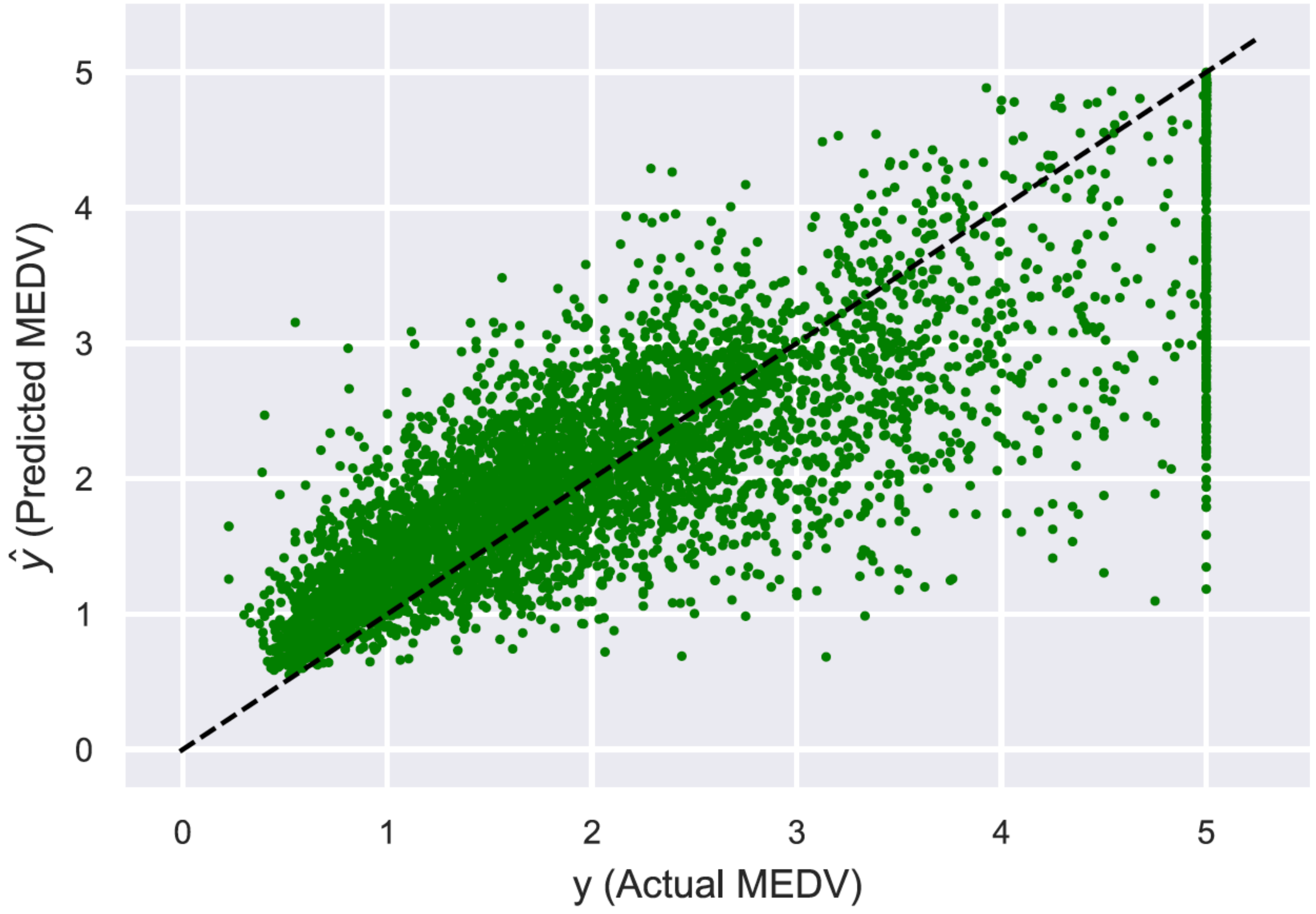


```python
plt.style.use('seaborn-v0_8')
knn = KNN(n_neighbors=50, weights='uniform', metric='euclidean')
knn.fit(X_train_fs_scaled, y_train)
y_test_predictions = knn.predict(X_test_fs_scaled)

plt.figure(figsize=(4.5, 3), dpi = 200)
plt.plot(y_test, y_test_predictions, 'g.', markersize=4)
lim_left, lim_right = plt.xlim()
plt.plot([lim_left, lim_right], [lim_left, lim_right], '--k', linewidth=1)
plt.xlabel("y (Actual MEDV)", fontsize='small')
plt.ylabel("$\hat{y}$ (Predicted MEDV)", fontsize='small')
plt.tick_params(axis='both', labelsize=7)
```

Model Training & Evaluation

UIC BUSINESS

- The scatter plot of predicted targets by a 50NN and the actual targets in the California Housing dataset



Model Training & Evaluation

UIC BUSINESS

- Let $\mathbf{S}_{te}$ be the test set containing m test observations

- If $\hat{y}_i = y_i$ for $\mathbf{x}_i \in \mathbf{S}_{te}$, we commit no error in predictions, meaning all points in the scatter plot between predicted and actual targets should lie on the diagonal line $\hat{y} = y$

- From the plot we see that
  - when MEDV $\leq \$100\text{k}$, the model generally overestimates the target
  - for large values of MEDV around $\$100\text{k}$, the model underestimates the target

Model Training & Evaluation

UIC BUSINESS

- Classifiers in scikit-learn have a `score` method that given a test data and their labels, returns a performance measure
- For regressors, the `score` method estimates *coefficient of determination* (*R-squared* statistic $\hat{R}^2$), given by

$$\hat{R}^2 = 1 - \frac{\text{RSS}}{\text{TSS}}$$

where RSS and TSS are short for *Residual Sum of Squares* and *Total Sum of Squares*, resp:

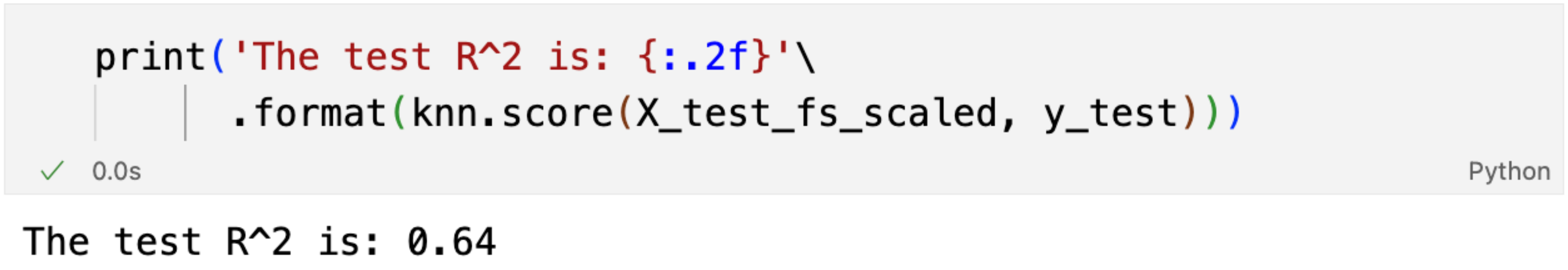
$$\text{RSS} = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

$$\text{TSS} = \sum_{i=1}^{m} (y_i - \bar{y})^2 \qquad \bar{y} = \frac{1}{m} \sum_{i=1}^{m} y_i$$

trivial estimator of the target

Model Training & Evaluation

- $\hat{R}^2$ measures how well the trained regressor is performing when compared with the trivial estimator
  - for perfect predictions, RSS = 0 and $\hat{R}^2 = 1$
  - when trained regressor performs similarly to the trivial one, RSS = TSS and $\hat{R}^2 = 0$



```python
print('The test R^2 is: {:.2f}'\
      .format(knn.score(X_test_fs_scaled, y_test)))
```

✓ 0.0s Python

```
The test R^2 is: 0.64
```

Distance-weighted kNN

UIC BUSINESS

- DW-kNN regressor computes a weighted average of targets for the k nearest neighbors, where the weights are the inverse of the distance of each nearest neighbor $\mathbf{x}$

$$f(\mathbf{x}) = \frac{1}{D} \sum_{i=1}^{K} \frac{1}{d[\mathbf{x}_{(i)}(\mathbf{x}),\ \mathbf{x}]} y_{(i)}$$

- Observations that are closer to a test observation impose a higher influence in the weighted average

- In scikit-learn, DW-kNN regressor is obtained by switching the `weights` parameter of `KNeighborsRegressor` from `'uniform'` to `'distance'`