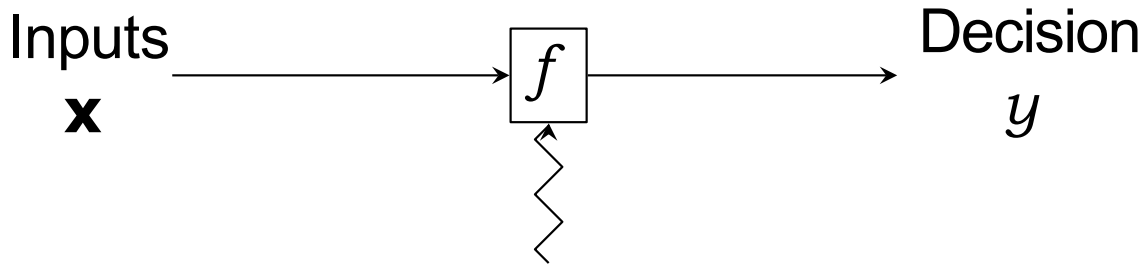
Lecture 11
K- Nearest Neighbors

What is Machine Learning?

Learning: devising a rule for making a decision based on inputs.

What is Machine Learning?

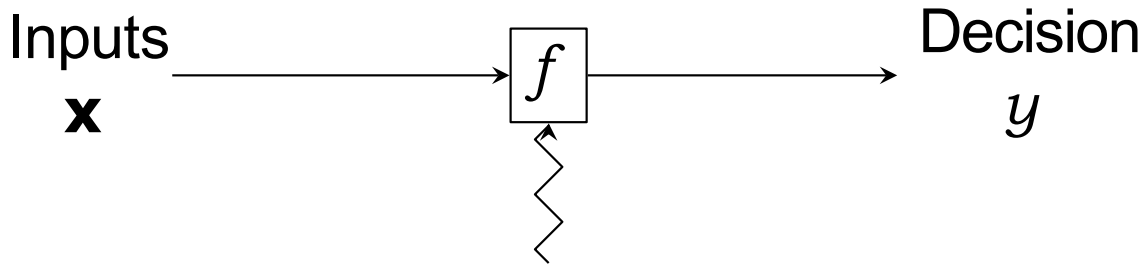
Learning: devising a rule for making a decision based on inputs.



Goal of Machine Learning:
Estimate the rule f
from data $(\mathbf{x}_i, y_i)$.

What is Machine Learning?

Learning: devising a rule for making a decision based on inputs.



Goal of Machine Learning:
Estimate the rule f
from data $(\mathbf{x}_i, y_i)$.

The decision y is typically called the **target** or the **label**.

Two Types of Machine Learning Problems

Machine learning problems are grouped into two types, based on the type of y:

Two Types of Machine Learning Problems

Machine learning problems are grouped into two types, based on the type of y:

Regression: The label y is quantitative.

Two Types of Machine Learning Problems

Machine learning problems are grouped into two types, based on the type of y:

Regression: The label y is quantitative.

Classification: The label y is categorical.

Two Types of Machine Learning Problems

- Machine learning problems are grouped into two types, based on the type of y:
- **Regression:** The label y is quantitative.
- **Classification:** The label y is categorical.
- Note that the input features $\mathbf{x}$ may be categorical, quantitative, or a mix of the two.

Two Types of Machine Learning Problems

- Machine learning problems are grouped into two types, based on the type of y:
- **Regression:** The label y is quantitative.
- **Classification:** The label y is categorical.
- Note that the input features $\mathbf{x}$ may be categorical, quantitative, or a mix of the two.

- We will initially focus on regression problems.

Predicting Wine Quality

Orley Ashenfelter, an economics professor, used summer temperature and winter rainfall to predict the price of Bordeaux wines.

Predicting Wine Quality

Orley Ashenfelter, an economics professor, used summer temperature and winter rainfall to predict the price of Bordeaux wines.

```
import pandas as pd

df = pd.read_csv("https://dlsun.github.io/pods/data/bordeaux.csv",
                 index_col="year")
df
```

year	price	summer	har	sep	win	age
1952	37.0	17.1	160	14.3	600	40
1953	63.0	16.7	80	17.3	690	39
1955	45.0	17.1	130	16.8	502	37
1957	22.0	16.1	110	16.2	420	35
...	...	...	...	...	...	...
1988	NaN	17.1	59	16.8	808	4
1989	NaN	18.6	82	18.4	443	3
1990	NaN	18.7	80	19.3	468	2
1991	NaN	17.7	183	20.4	570	1

38 rows × 6 columns

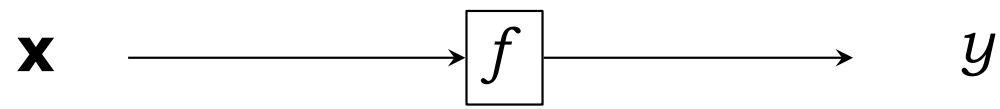
Predicting Wine Quality

Orley Ashenfelter, an economics professor, used summer temperature and winter rainfall to predict the price of Bordeaux wines.

```
import pandas as pd

df = pd.read_csv("https://dlsun.github.io/pods/data/bordeaux.csv",
                 index_col="year")
df
```

year	price	summer	har	sep	win	age
1952	37.0	17.1	160	14.3	600	40
1953	63.0	16.7	80	17.3	690	39
1955	45.0	17.1	130	16.8	502	37
1957	22.0	16.1	110	16.2	420	35
...	...	...	...	...	...	...
1988	NaN	17.1	59	16.8	808	4
1989	NaN	18.6	82	18.4	443	3
1990	NaN	18.7	80	19.3	468	2
1991	NaN	17.7	183	20.4	570	1

38 rows × 6 columns



summer temp,

winter rainfall

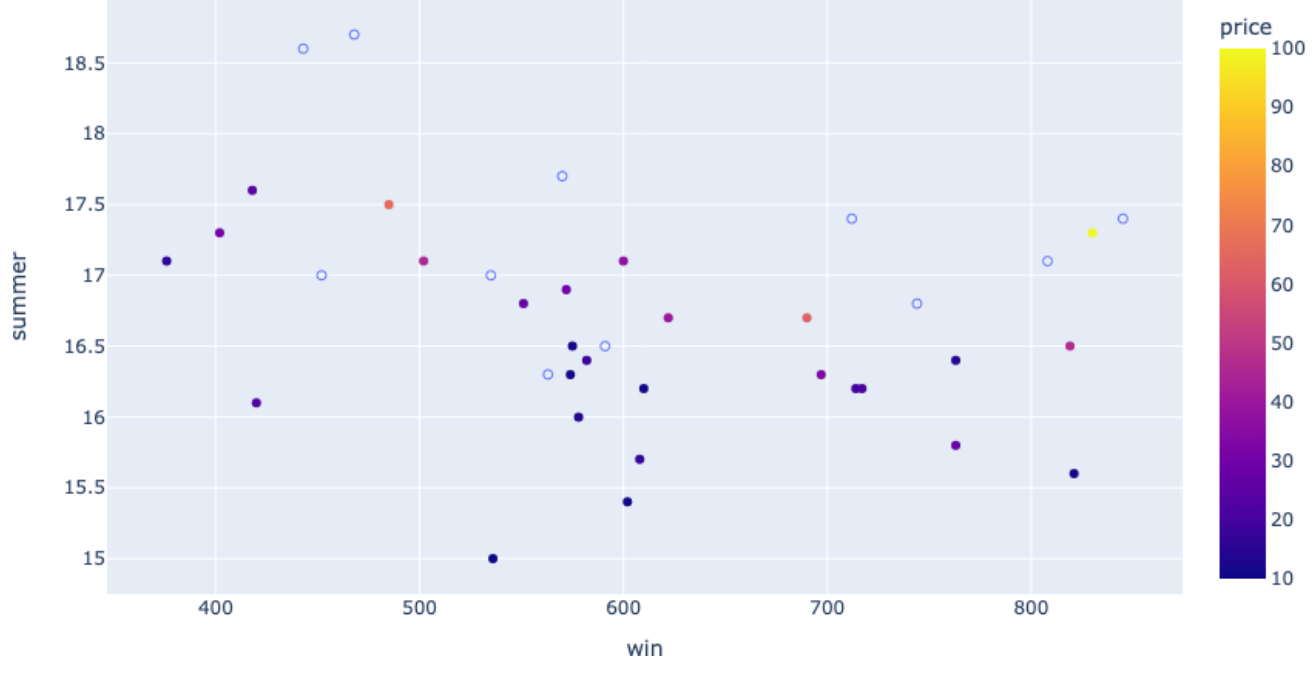
price

Visualizing the Data

```
import plotly.express as px
import plotly.graph_objects as go

fig1 = px.scatter(df[~df["price"].isnull()],
                  x="win", y="summer", color="price")
fig2 = px.scatter(df[df["price"].isnull()],
                  x="win", y="summer", symbol_sequence=["circle-open"])

go.Figure(data=fig1.data + fig2.data, layout=fig1.layout)
```

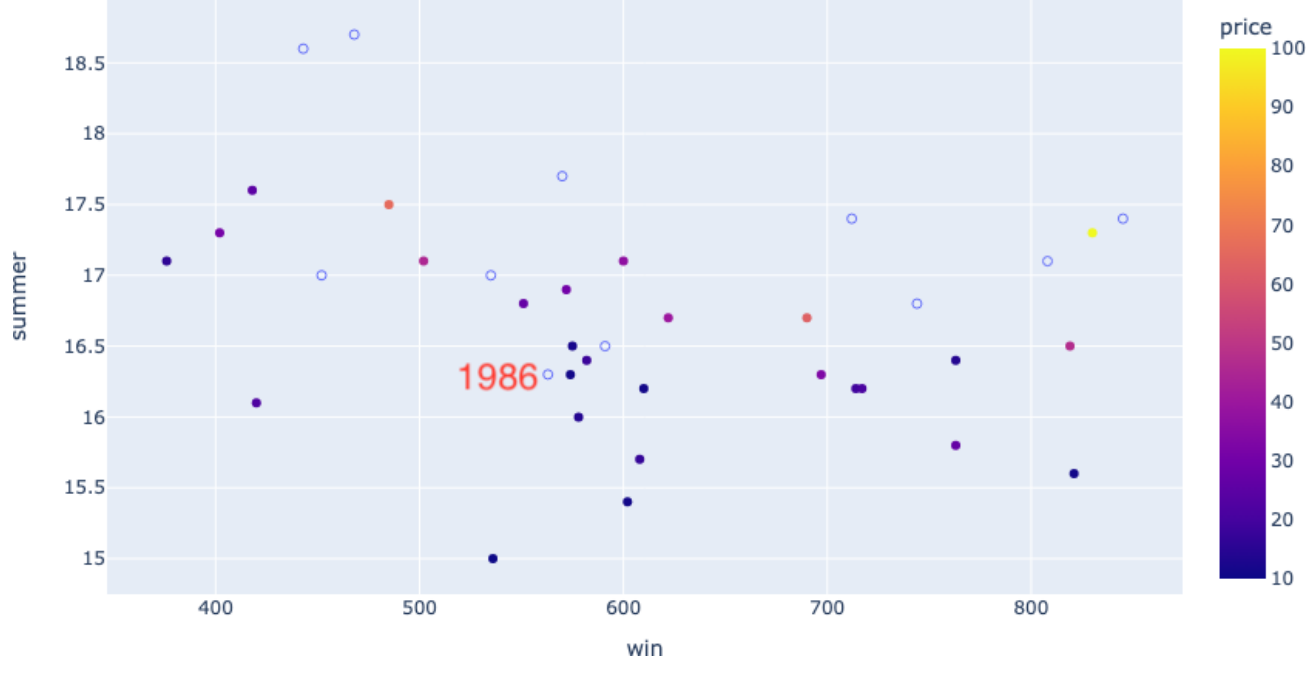
Visualizing the Data

```
import plotly.express as px
import plotly.graph_objects as go

fig1 = px.scatter(df[~df["price"].isnull()],
                  x="win", y="summer", color="price")
fig2 = px.scatter(df[df["price"].isnull()],
                  x="win", y="summer", symbol_sequence=["circle-open"])

go.Figure(data=fig1.data + fig2.data, layout=fig1.layout)
```



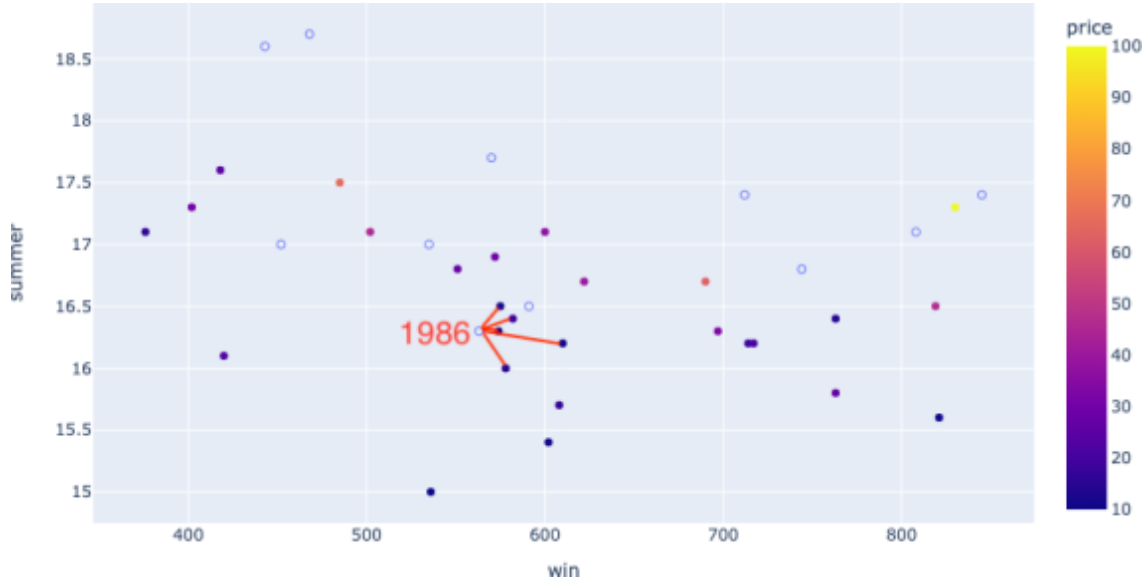
Visualizing the Data

```
import plotly.express as px
import plotly.graph_objects as go

fig1 = px.scatter(df[~df["price"].isnull()],
                  x="win", y="summer", color="price")
fig2 = px.scatter(df[df["price"].isnull()],
                  x="win", y="summer", symbol_sequence=["circle-open"])

go.Figure(data=fig1.data + fig2.data, layout=fig1.layout)
```



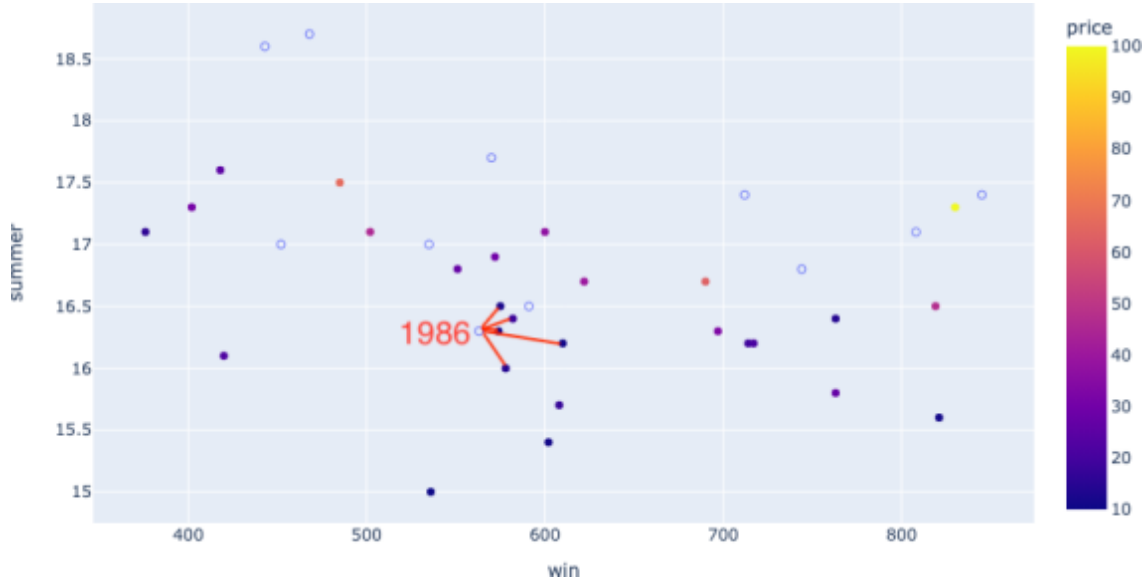
What would you predict is the quality of the 1986 wine?

Visualizing the Data



Insight: The “closest” wines are low quality, so the 1986 vintage is probably low quality as well.

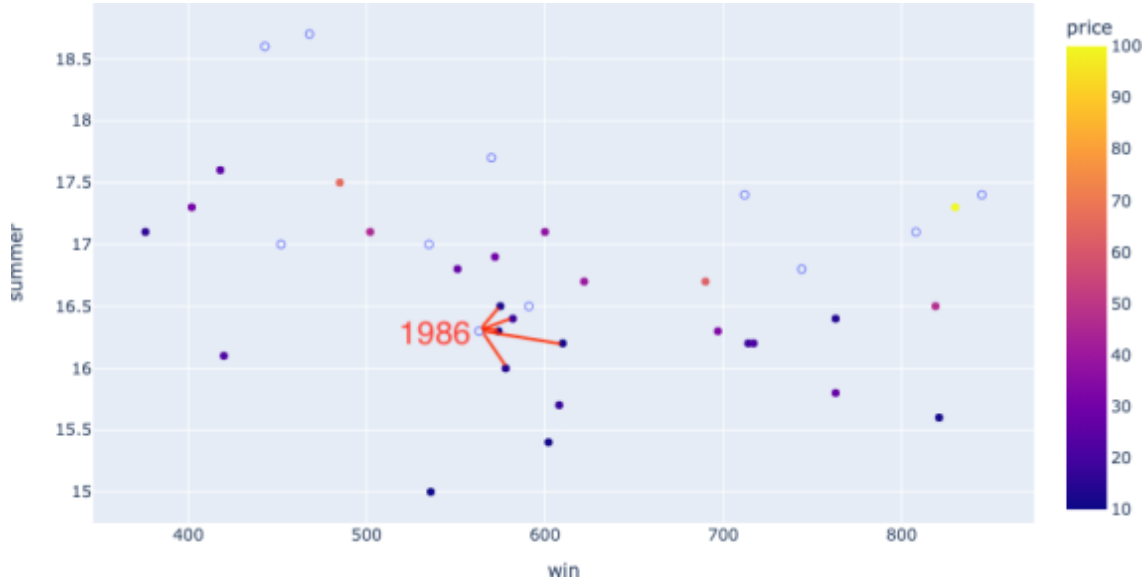
Visualizing the Data



Insight: The "closest" wines are low quality, so the 1986 vintage is probably low quality as well.

This is the intuition behind k-**nearest neighbors** regression.

Visualizing the Data



Insight: The "closest" wines are low quality, so the 1986 vintage is probably low quality as well.

This is the intuition behind k-**nearest neighbors** regression.

Today: implementing k-nearest neighbors

K- Nearest Neighbors

The data for which we know the label y is called the training data.

K-Nearest Neighbors

The data for which we know the label y is called the **training data**.

The data for which we don't know y (and want to predict it) is called the **test data**.

K-Nearest Neighbors

The data for which we know the label y is called the **training data**.

The data for which we don't know y (and want to predict it) is called the **test data**.

```
df_train = df.loc[:1980].copy()
df_test = df.loc[1981:].copy()
```

K-Nearest Neighbors

The data for which we know the label y is called the **training data**.

The data for which we don't know y (and want to predict it) is called the **test data**.

```
df_train = df.loc[:1980].copy()
df_test = df.loc[1981:].copy()
```

K-Nearest Neighbors

K-Nearest Neighbors

The data for which we know the label y is called the **training data**.

The data for which we don't know y (and want to predict it) is called the **test data**.

```
df_train = df.loc[:1980].copy()
df_test = df.loc[1981:].copy()
```

K-Nearest Neighbors

1. For each observation in the test data, find the k "nearest" observations in the training data based on input features $\mathbf{x}$ (e.g., summer temperature and winter rainfall).

K-Nearest Neighbors

The data for which we know the label y is called the **training data**.

The data for which we don't know y (and want to predict it) is called the **test data**.

```
df_train = df.loc[:1980].copy()
df_test = df.loc[1981:].copy()
```

K-Nearest Neighbors

1. For each observation in the test data, find the k "nearest" observations in the training data based on input features $\mathbf{x}$ (e.g., summer temperature and winter rainfall).
2. To predict the label y (e.g., price) for the test observation, average the labels of these k "nearest" training observations.

K- Nearest Neighbors from Scratch

K-Nearest Neighbors from Scratch

Before computing distances, we should scale the variables.

```
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]

# Standardize the features.
X_train_mean = X_train.mean()
X_train_sd = X_train.std()
X_train_scaled = (X_train - X_train_mean) / X_train_sd
```

K-Nearest Neighbors from Scratch

Before computing distances, we should scale the variables.

```
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]

# Standardize the features.
X_train_mean = X_train.mean()
X_train_sd = X_train.std()
X_train_scaled = (X_train - X_train_mean) / X_train_sd
```

We should scale the test data in exactly same way.

```
X_test = df_test[["win", "summer"]]
X_test_scaled = (X_test - X_train_mean) / X_train_sd
X_test_scaled
```

K-Nearest Neighbors from Scratch

Before computing distances, we should scale the variables.

```
X_train = df_train[["win", "summer"]]
y_train = df_train["price"]

# Standardize the features.
X_train_mean = X_train.mean()
X_train_sd = X_train.std()
X_train_scaled = (X_train - X_train_mean) / X_train_sd
```

We should scale the test data in exactly same way.

```
X_test = df_test[["win", "summer"]]
X_test_scaled = (X_test - X_train_mean) / X_train_sd
X_test_scaled
```

year	win	summer
1981	-0.568896	0.812183
1982	0.802826	1.425581
1983	1.833554	1.425581
1984	-0.134905	0.045437
...	...	...
1988	1.546810	0.965533
1989	-1.281881	3.265772
1990	-1.088135	3.419122
1991	-0.297651	1.885629

11 rows × 2 columns

K-Nearest Neighbors from Scratch

Next, we calculate the (Euclidean) distances between a vintage in the test set and the vintages in the training data.

```
import numpy as np
dists = np.sqrt(
    ((X_test_scaled.loc[1986] - X_train_scaled) ** 2).sum(axis=1))
dists
```

axis = 0 → sums down the rows (column-wise operation)

axis = 1 → sums across the columns

K-Nearest Neighbors from Scratch

Next, we calculate the (Euclidean) distances between a vintage in the test set and the vintages in the training data.

```
import numpy as np
dists = np.sqrt(
    ((X_test_scaled.loc[1986] - X_train_scaled) ** 2).sum(axis=1))
dists
```

```
year
1952    1.259860
1953    1.159726
1955    1.314727
1957    1.149883
          ...
1977    2.269387
1978    1.729248
1979    1.203287
1980    0.474508
Length: 27, dtype: float64
```

K-Nearest Neighbors from Scratch

Next, we calculate the (Euclidean) distances between a vintage in the test set and the vintages in the training data.

```
import numpy as np
dists = np.sqrt(
    ((X_test_scaled.loc[1986] - X_train_scaled) ** 2).sum(axis=1))
dists
```

```
year
1952     1.259860
1953     1.159726
1955     1.314727
1957     1.149883
            ...
1977     2.269387
1978     1.729248
1979     1.203287
1980     0.474508
Length: 27, dtype: float64
```

Now we just need to sort these distances and take the first $k = 5$.

K-Nearest Neighbors from Scratch

Next, we calculate the (Euclidean) distances between a vintage in the test set and the vintages in the training data.

```
import numpy as np
dists = np.sqrt(
    ((X_test_scaled.loc[1986] - X_train_scaled) ** 2).sum(axis=1))
dists
```

```
year
1952      1.259860
1953      1.159726
1955      1.314727
1957      1.149883
             ...
1977      2.269387
1978      1.729248
1979      1.203287
1980      0.474508
Length: 27, dtype: float64
```

Now we just need to sort these distances and take the first $k = 5$.

```
index_nearest = dists.sort_values().index[:5]
index_nearest
```

K-Nearest Neighbors from Scratch

Next, we calculate the (Euclidean) distances between a vintage in the test set and the vintages in the training data.

```
import numpy as np
dists = np.sqrt(
    ((X_test_scaled.loc[1986] - X_train_scaled) ** 2).sum(axis=1))
dists
```

```
year
1952     1.259860
1953     1.159726
1955     1.314727
1957     1.149883
            ...
1977     2.269387
1978     1.729248
1979     1.203287
1980     0.474508
Length: 27, dtype: float64
```

Now we just need to sort these distances and take the first $k = 5$.

```
index_nearest = dists.sort_values().index[:5]
index_nearest
```

```
Int64Index([1974, 1958, 1969, 1968, 1980], dtype='int64', name='year')
```

K-Nearest Neighbors from Scratch

Finally, to make a prediction, we average the labels y of these $k = 5$ nearest vintages in the training data.

K-Nearest Neighbors from Scratch

Finally, to make a prediction, we average the labels y of these $k = 5$ nearest vintages in the training data.

```
y_train[index_nearest].mean()
```

K-Nearest Neighbors from Scratch

Finally, to make a prediction, we average the labels y of these $k = 5$ nearest vintages in the training data.

```
y_train[index_nearest].mean()
```

13.2

K-Nearest Neighbors from Scratch

Finally, to make a prediction, we average the labels y of these $k = 5$ nearest vintages in the training data.

```
y_train[index_nearest].mean()
```

```
13.2
```

That's $13.20 for a bottle of wine. So 1986 is not a good vintage.

K-Nearest Neighbors from Scratch

Finally, to make a prediction, we average the labels y of these $k = 5$ nearest vintages in the training data.

```
y_train[index_nearest].mean()
```

```
13.2
```

That's $13.20 for a bottle of wine. So 1986 is not a good vintage.

How do we do this for every vintage in the test set?

Let’s go into Colab and predict the price for every vintage in the test set.

K- Nearest Neighbors in Scikit-Learn

Scikit-learn provides a built-in model `KNeighborsRegressor` that fits k-nearest neighbors regression models.

K- Nearest Neighbors in Scikit-Learn

Scikit-learn provides a built-in model `KNeighborsRegressor` that fits k-nearest neighbors regression models.

But first, we need to scale the training and test data.

K-Nearest Neighbors in Scikit-Learn

Scikit-learn provides a built-in model `KNeighborsRegressor` that fits k-nearest neighbors regression models.

But first, we need to scale the training and test data.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)

# Scale the test data using a scaler that was fit to the training data!
X_test_scaled = scaler.transform(X_test)
```

K-Nearest Neighbors in Scikit-Learn

Scikit-learn provides a built-in model `KNeighborsRegressor` that fits k-nearest neighbors regression models.

But first, we need to scale the training and test data.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)

# Scale the test data using a scaler that was fit to the training data!
X_test_scaled = scaler.transform(X_test)
```

Now, we fit k-nearest neighbors using `KNeighborsRegressor`.

K-Nearest Neighbors in Scikit-Learn

Scikit-learn provides a built-in model `KNeighborsRegressor` that fits *k*-nearest neighbors regression models.

But first, we need to scale the training and test data.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)

# Scale the test data using a scaler that was fit to the training data!
X_test_scaled = scaler.transform(X_test)
```

Now, we fit *k*-nearest neighbors using `KNeighborsRegressor`.

```
from sklearn.neighbors import KNeighborsRegressor

model = KNeighborsRegressor(n_neighbors=5)
model.fit(X=X_train_scaled, y=y_train)
model.predict(X=X_test_scaled)
```

K-Nearest Neighbors in Scikit-Learn

Scikit-learn provides a built-in model `KNeighborsRegressor` that fits *k*-nearest neighbors regression models.

But first, we need to scale the training and test data.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)

# Scale the test data using a scaler that was fit to the training data!
X_test_scaled = scaler.transform(X_test)
```

Now, we fit *k*-nearest neighbors using `KNeighborsRegressor`.

```
from sklearn.neighbors import KNeighborsRegressor

model = KNeighborsRegressor(n_neighbors=5)
model.fit(X=X_train_scaled, y=y_train)
model.predict(X=X_test_scaled)
```

```
array([35.8, 54. , 52.2, 18.4, 35.6, 13.2, 37. , 51.4, 36.6, 36.6, 40.6])
```

Pipelines in Scikit-Learn

In the code above, we had to be careful to standardize the training and test data in exactly the same way.

Pipelines in Scikit-Learn

In the code above, we had to be careful to standardize the training and test data in exactly the same way.

Machine learning models typically involve many more preprocessing steps.

Pipelines in Scikit-Learn

In the code above, we had to be careful to standardize the training and test data in exactly the same way.

Machine learning models typically involve many more preprocessing steps.

Scikit-Learn's `Pipeline` allows us to chain steps together.

Pipelines in Scikit-Learn

In the code above, we had to be careful to standardize the training and test data in exactly the same way.

Machine learning models typically involve many more preprocessing steps.

Scikit-Learn's `Pipeline` allows us to chain steps together.

```
from sklearn.pipeline import make_pipeline

pipeline = make_pipeline(
           StandardScaler(),
           KNeighborsRegressor(n_neighbors=5))
```

Pipeline
a structure that runs multiple processing steps (e.g., scaling + model) sequentially as if they were a single model.

Pipelines in Scikit-Learn

In the code above, we had to be careful to standardize the training and test data in exactly the same way.

Machine learning models typically involve many more preprocessing steps.

Scikit-Learn's `Pipeline` allows us to chain steps together.

```
from sklearn.pipeline import make_pipeline

pipeline = make_pipeline(
           StandardScaler(),
           KNeighborsRegressor(n_neighbors=5))
```

We can use this `Pipeline` like any other machine learning model.

Pipelines in Scikit-Learn

In the code above, we had to be careful to standardize the training and test data in exactly the same way.

Machine learning models typically involve many more preprocessing steps.

Scikit-Learn's `Pipeline` allows us to chain steps together.

```
from sklearn.pipeline import make_pipeline

pipeline = make_pipeline(
           StandardScaler(),
           KNeighborsRegressor(n_neighbors=5))
```

We can use this `Pipeline` like any other machine learning model.

```
pipeline.fit(X=X_train, y=y_train)
pipeline.predict(X=X_test)
```

```
array([35.8, 54. , 52.2, 18.4, 35.6, 13.2, 37. , 51.4, 36.6, 36.6, 40.6])
```

Was Ashenfelter Right?

Here is how the vintages rank today:

Vintage	Score	Drink Window	Description
1994	85	Drink	Medium-bodied, with good fruit and firm tannins
1993	82	Past peak	Good color, perfumy, fruity and balanced
1992	72	Past peak	Light, simple and often diluted; early maturing
1991	72	Past peak	Lean, tough and light; stick to top names
1990	97	Drink	Opulent, well-structured and harmonious
1989	98	Drink	Bold, dramatic fruit character; tannic and long-aging
1988	93	Drink	Racy, fruity wines, with firm tannins and typical structure; best in Pessac-Léognan, Pomerol and Pauillac
1987	76	Past peak	Delicate, ripe yet diluted
1986	95	Drink	Powerful, intense and tannic; best in Médoc
1985	93	Drink	Balanced, supple and fruity; defines finesse
1984	70	Past peak	Unripe, astringent and dry

Was Ashenfelter Right?

Here is how the vintages rank today:

Vintage	Score	Drink Window	Description
1994	85	Drink	Medium-bodied, with good fruit and firm tannins
1993	82	Past peak	Good color, perfumy, fruity and balanced
1992	72	Past peak	Light, simple and often diluted; early maturing
1991	72	Past peak	Lean, tough and light; stick to top names
1990	97	Drink	Opulent, well-structured and harmonious
1989	98	Drink	Bold, dramatic fruit character; tannic and long-aging
1988	93	Drink	Racy, fruity wines, with firm tannins and typical structure; best in Pessac-Léognan, Pomerol and Pauillac
1987	76	Past peak	Delicate, ripe yet diluted
1986	95	Drink	Powerful, intense and tannic; best in Médoc
1985	93	Drink	Balanced, supple and fruity; defines finesse
1984	70	Past peak	Unripe, astringent and dry

Ashenfelter was wrong that 1986 would be a disappointing vintage!