


# Machine Learning

## K-Nearest Neighbours Regression

Tarapong Sreenuch

8 February 2024

克明峻德，格物致知



I'm not sure what rent to ask. My building has all kinds of units.

I ask the closest neighbours what they pay. I use their average as my price.

Average your neighbours; get a fair price.

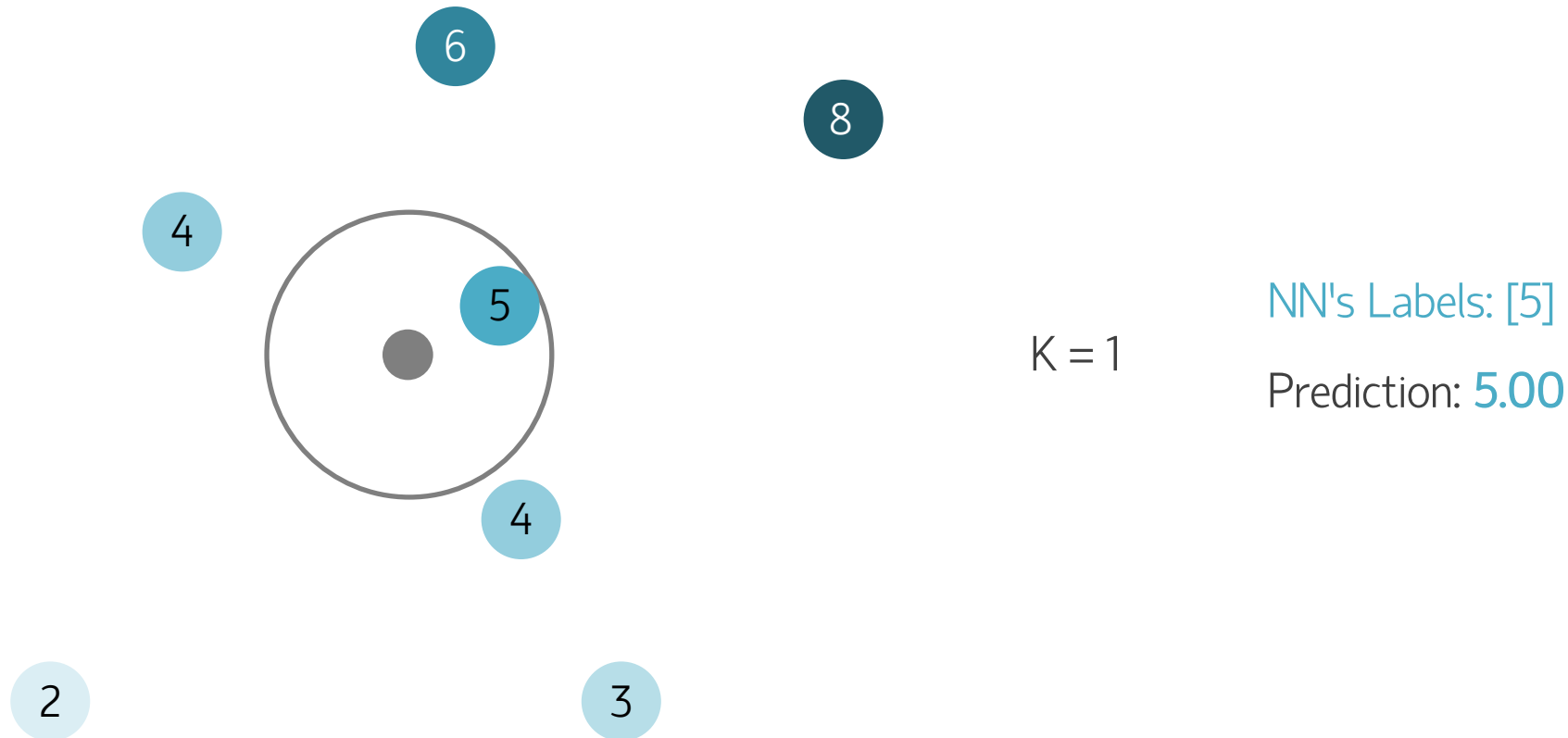
Q: How does KNN estimate a value?

A: We ask the nearest few and Take their average.

That's going to be very close!

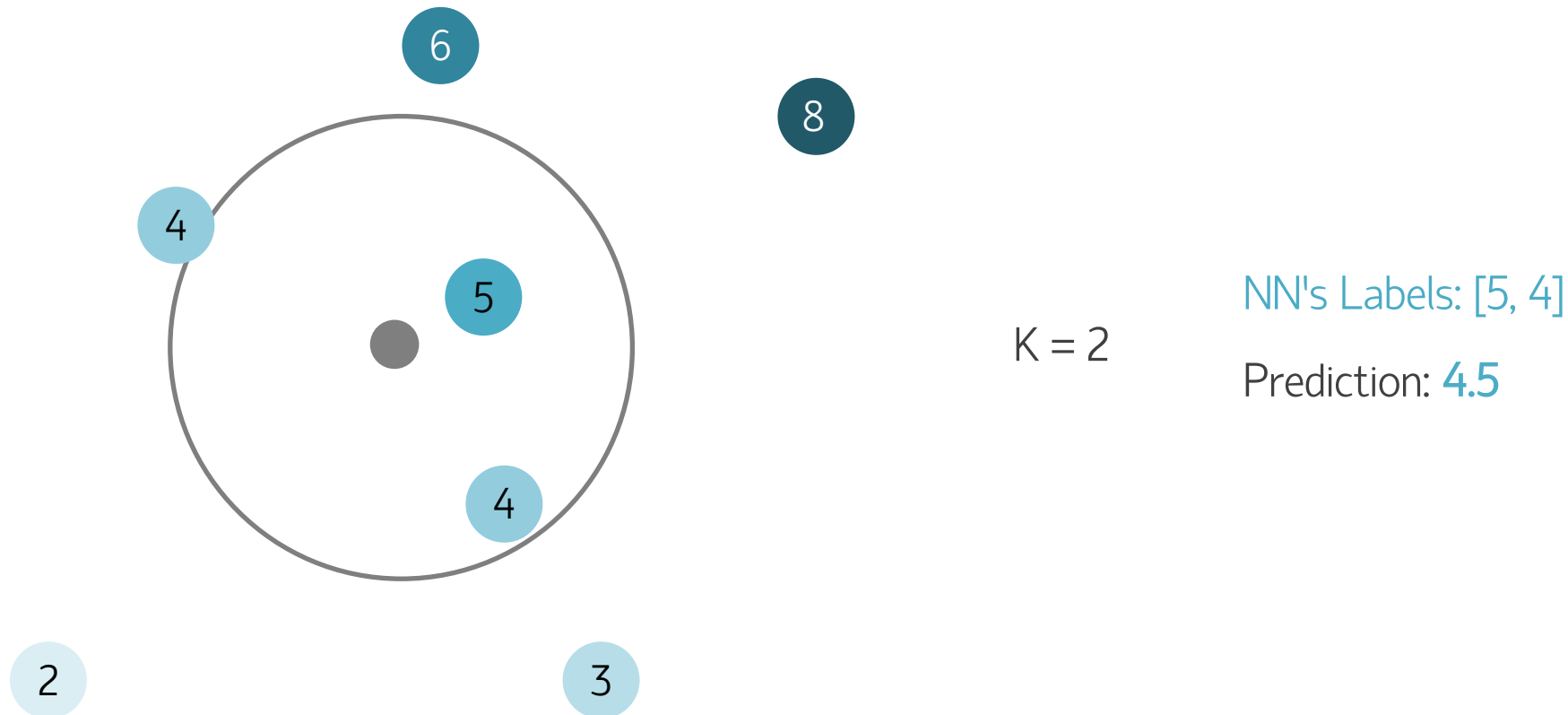
# K-Nearest Neighbours

The KNN algorithm predicts the target value for a test dataset by averaging the labels of its nearest neighbors in the training dataset's feature space. The 'K', representing the number of neighbors considered, is a key hyperparameter that can be adjusted to optimize model performance. Typically, the prediction is the mean or sometimes the median of these neighbors' target values, depending on the application's needs.



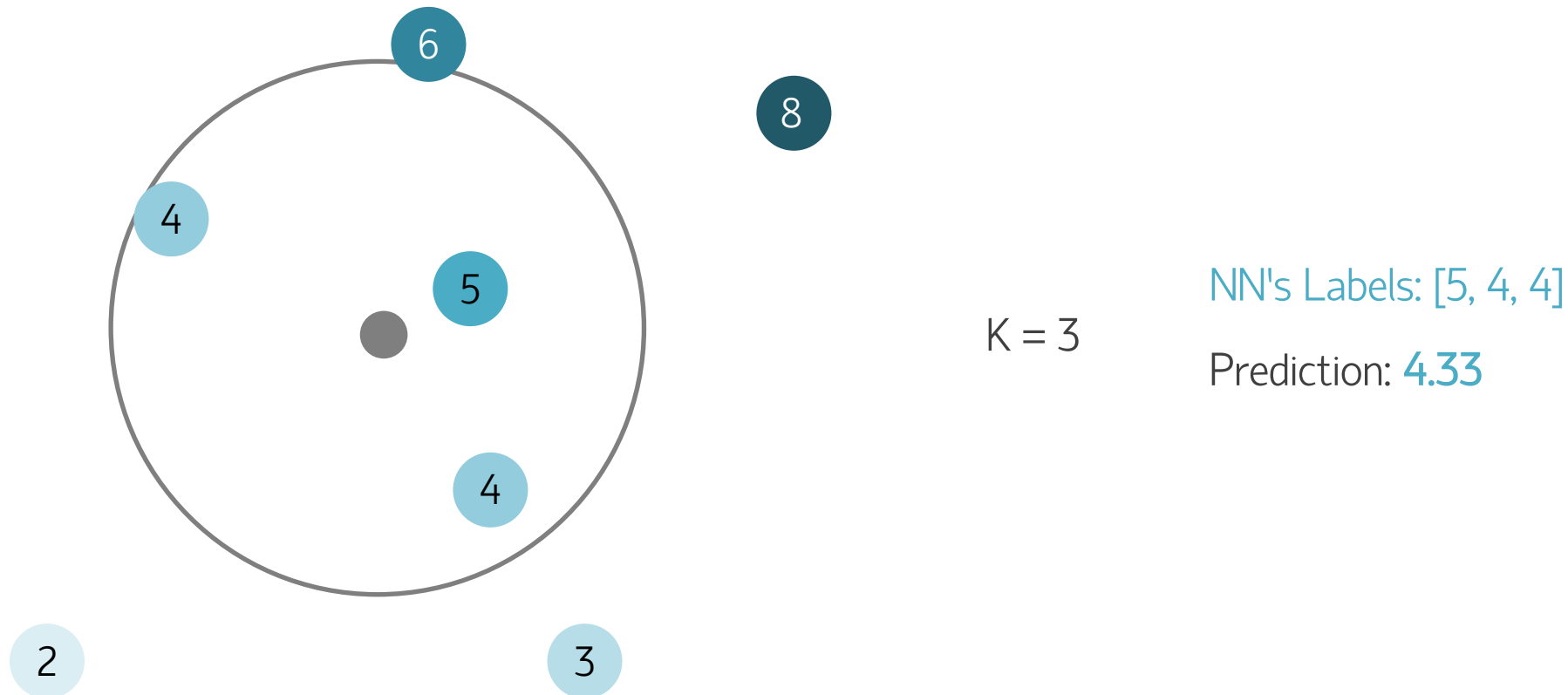
# K-Nearest Neighbours

The KNN algorithm predicts the target value for a test dataset by averaging the labels of its nearest neighbors in the training dataset's feature space. The 'K', representing the number of neighbors considered, is a key hyperparameter that can be adjusted to optimize model performance. Typically, the prediction is the mean or sometimes the median of these neighbors' target values, depending on the application's needs.



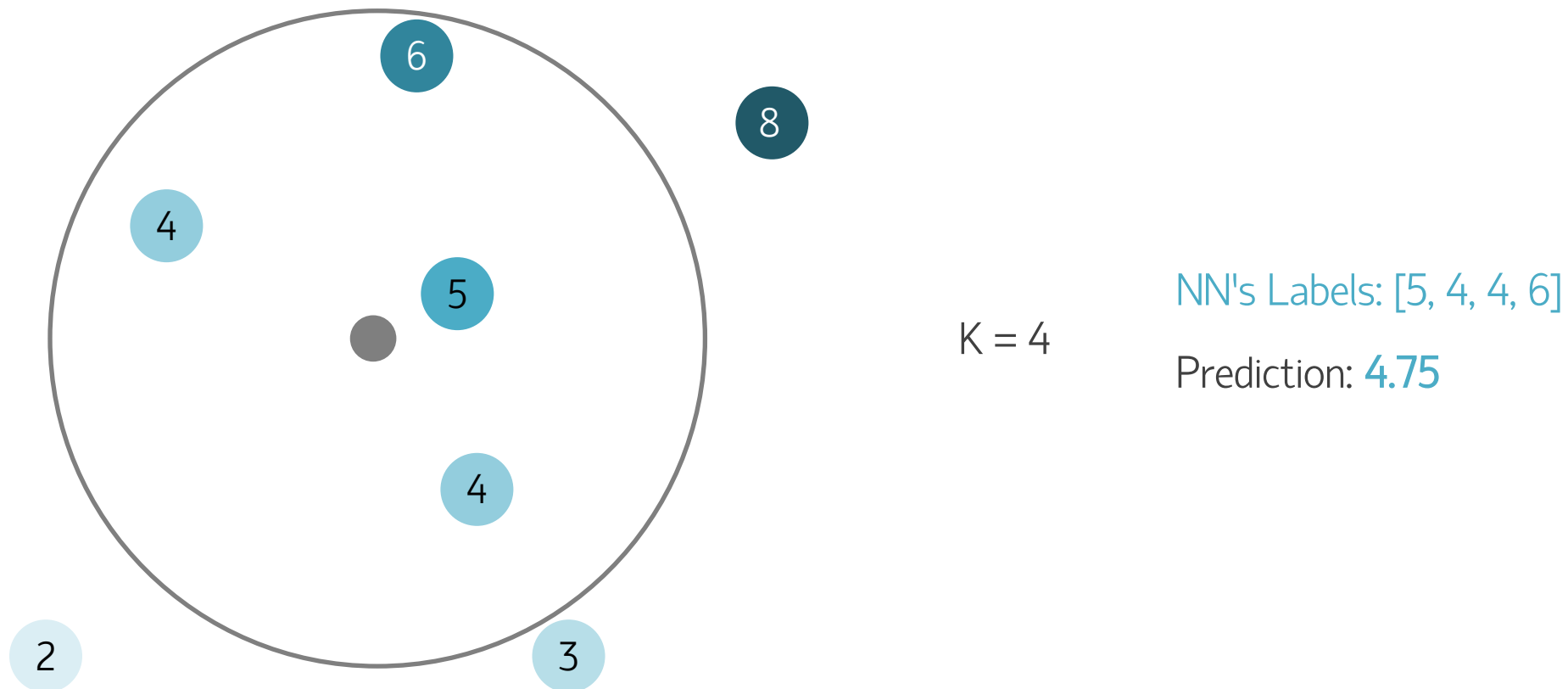
# K-Nearest Neighbours

The KNN algorithm predicts the target value for a test dataset by averaging the labels of its nearest neighbors in the training dataset's feature space. The 'K', representing the number of neighbors considered, is a key hyperparameter that can be adjusted to optimize model performance. Typically, the prediction is the mean or sometimes the median of these neighbors' target values, depending on the application's needs.



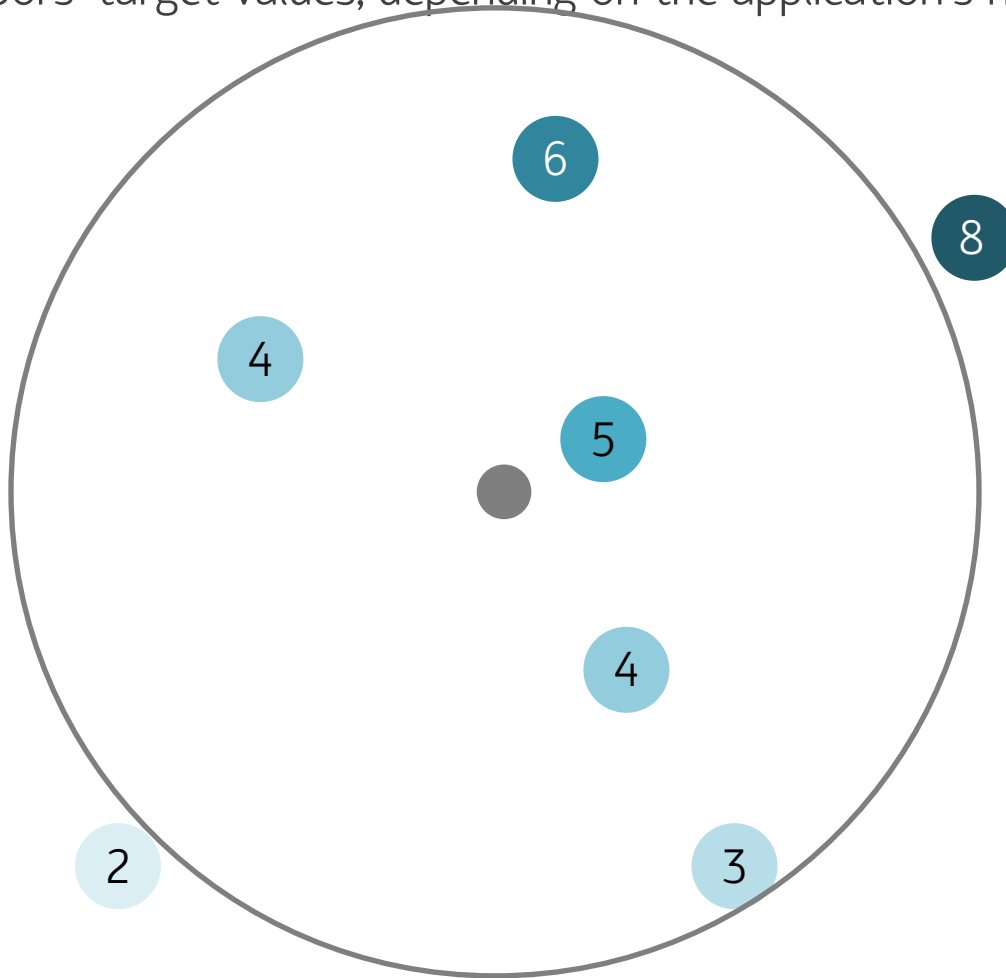
# K-Nearest Neighbours

The KNN algorithm predicts the target value for a test dataset by averaging the labels of its nearest neighbors in the training dataset's feature space. The 'K', representing the number of neighbors considered, is a key hyperparameter that can be adjusted to optimize model performance. Typically, the prediction is the mean or sometimes the median of these neighbors' target values, depending on the application's needs.



# K-Nearest Neighbours

The KNN algorithm predicts the target value for a test dataset by averaging the labels of its nearest neighbors in the training dataset's feature space. Typically, the prediction is the mean or sometimes the median of these neighbors' target values, depending on the application's needs.



$K = 5$

NN's Labels: [5, 4, 4, 6, 3]

Prediction: 4.40

# Pseudocode for KNN Regressor

```
# Inputs
# data      ← training set of N examples (x, y)
# k         ← number of neighbours
# metric    ← distance function (e.g., Euclidean, Manhattan)
# X_query   ← set of examples to predict

# ----- "fit" (lazy) -----
X_train ← data.x
y_train ← data.y

# ----- predict -----
ŷ ← list of length |X_query|           # outputs align 1:1 with X_query

FOR i = 1 TO |X_query| DO
  x* ← X_query[i]
  d  ← distances from x* to all X_train using metric
  J  ← indices of the k smallest values in d
  # regression prediction = average of the targets of the k nearest neighbours
  ŷ[i] ← mean(y_train[J])
END FOR

RETURN ŷ
```



# KNN from Scratch

```
import numpy as np
from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.metrics import pairwise_distances

class MyKNNRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, k=3, metric="euclidean"):
        self.k = k
        self.metric = metric

    def fit(self, X, y):
        self.X_train_ = np.asarray(X)
        self.y_train_ = np.asarray(y).ravel() # enforce 1D target
        return self

    def predict(self, X):
        X = np.asarray(X)
        y_pred = np.empty(X.shape[0], dtype=float)

        for i, x in enumerate(X):
            d = pairwise_distances(self.X_train_, x[None, :], metric=self.metric).ravel()
            J = np.argsort(d)[:self.k]
            y_pred[i] = self.y_train_[J].mean() # average of k nearest targets

        return y_pred
```

# Usage Example

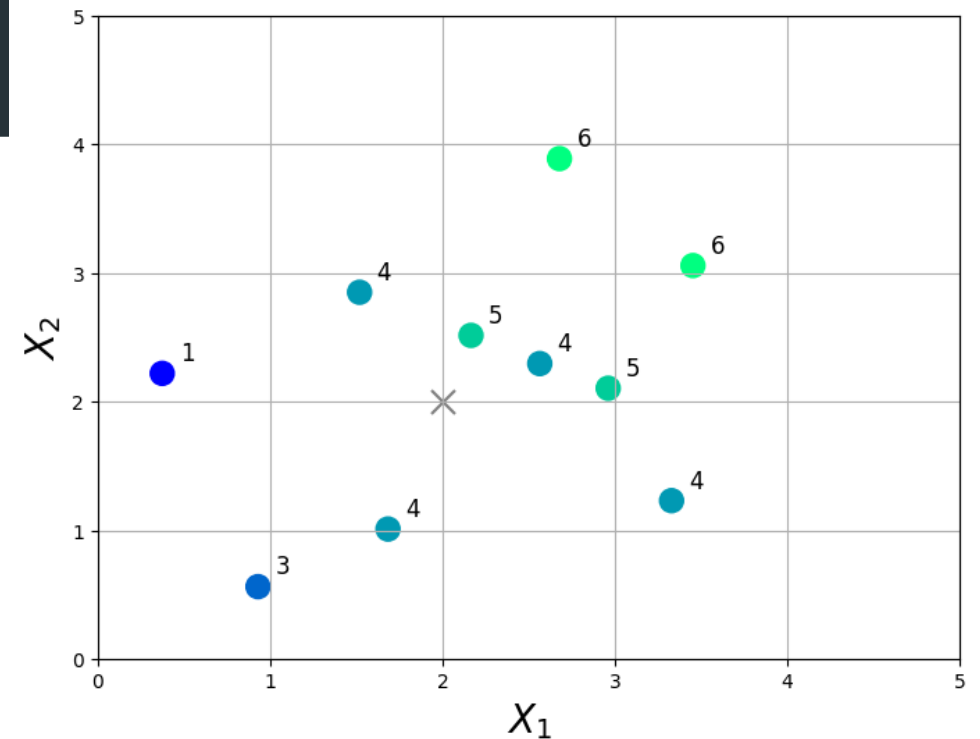
```
# Test data point
X_test = np.array([[2,2]])

# Example usage
K = 5
distance_metric = "euclidean" # Can be "euclidean", "Manhattan", "cosine", etc.

knn = MyKNNRegressor(K=5, metric=distance_metric)
knn.fit(X_train, y_train)

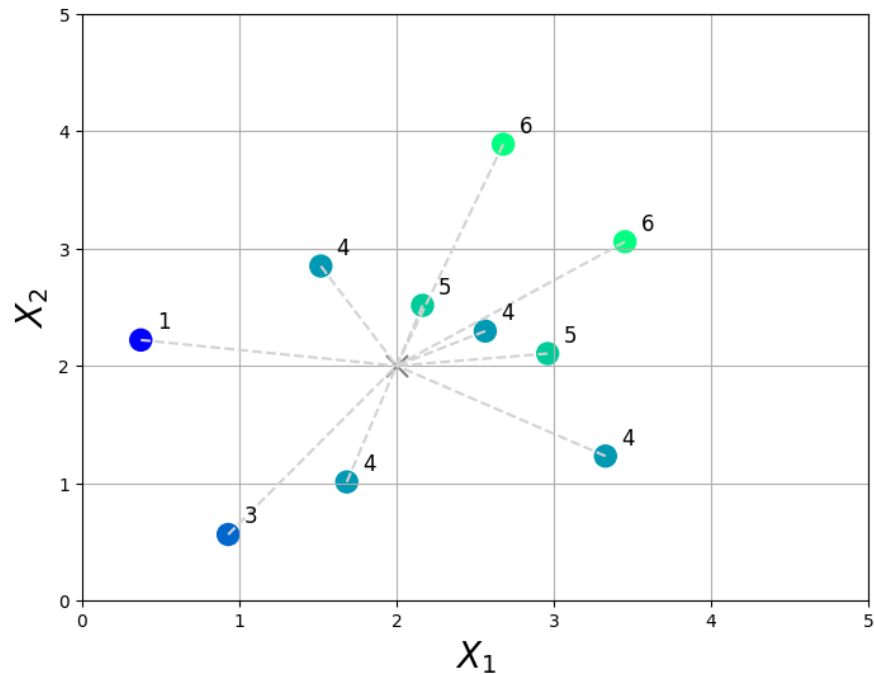
Y_pred = knn.predict()
print("Predicted Value:", y_pred)
```

Predicted Value: 4.40

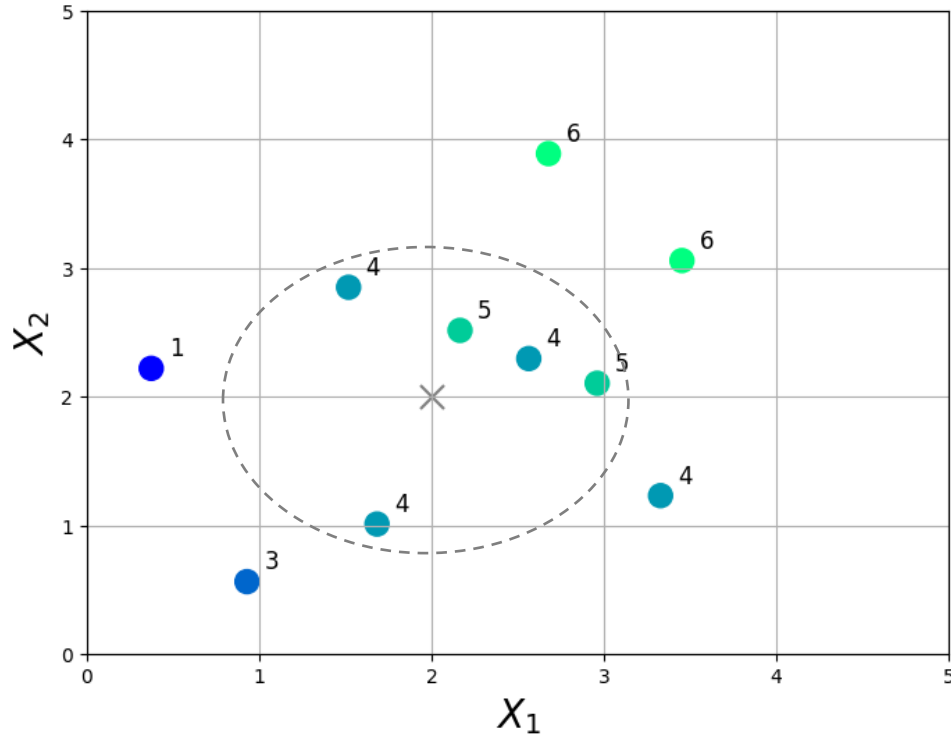


# Step 1: Pairwise Distance Calculation

```
# Step 1: Calculate distances using pairwise_distances
distances = pairwise_distances(data_points, [new_point], metric=distance_metric).ravel()
```



## Step 2 & 3: Finding k Nearest Neighbours

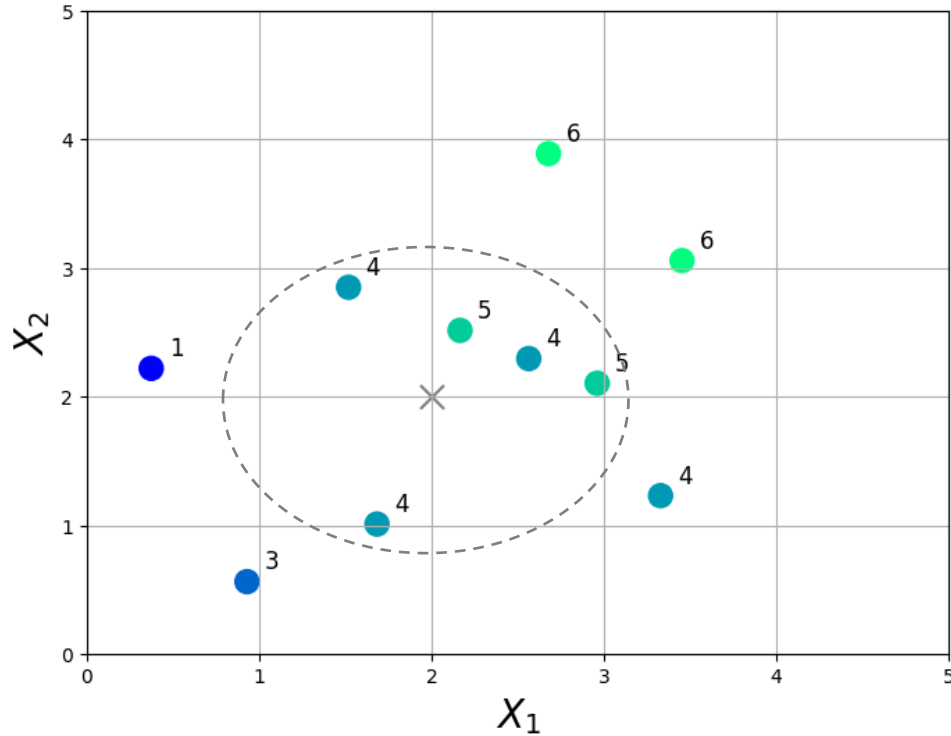


```
# Step 2 & 3: Sort distances and select k nearest neighbors  
J = np.argsort(d)[:self.k]
```

NN's Labels: [5, 4, 5, 4, 4]

K = 5

## Step 4 & 5: Prediction based on Majority Voting



```
# Step 4: Compute average of labels  
y_pred[i] = self.y_train[J].mean()
```

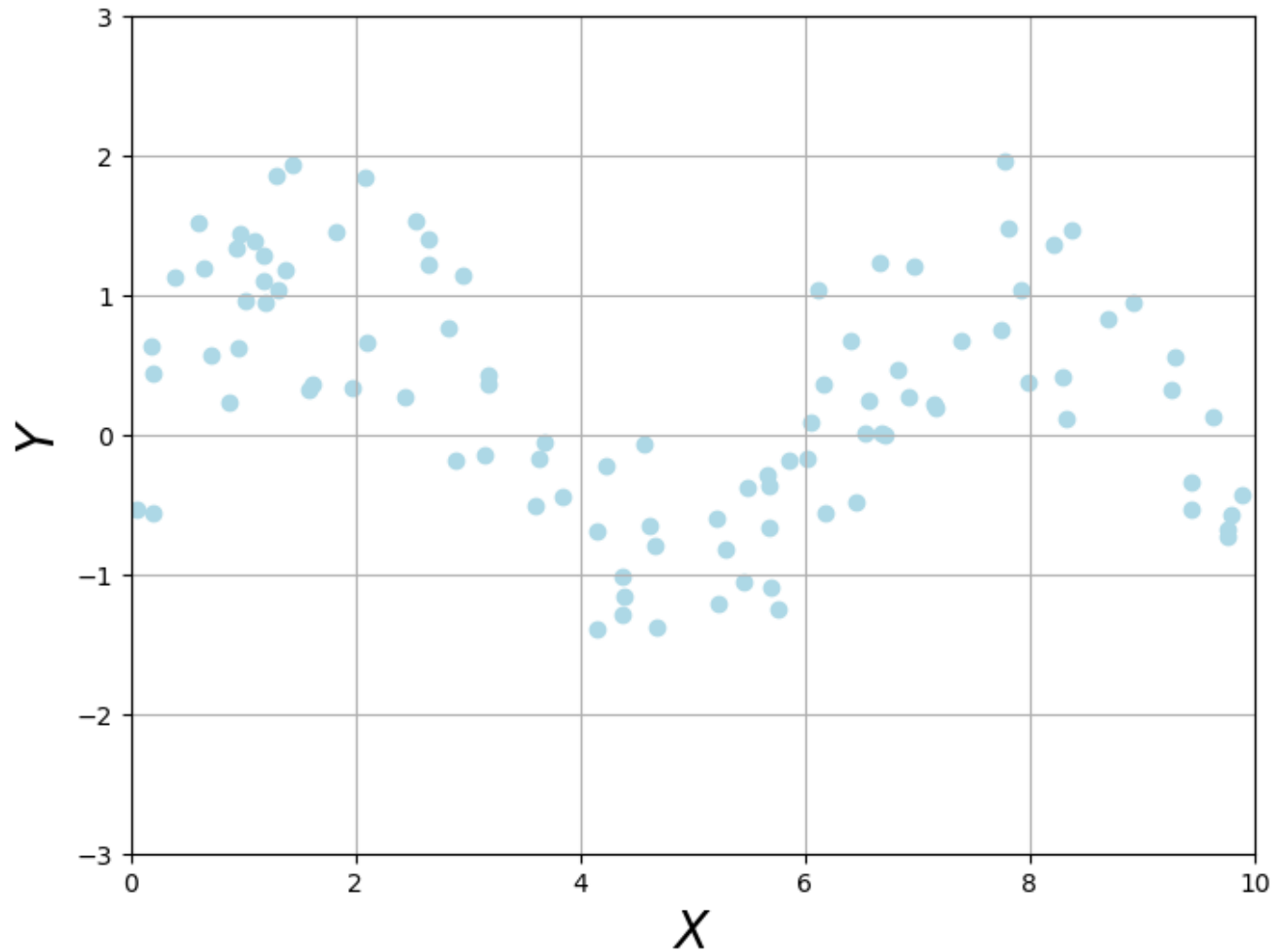
$K = 5$

NN's Labels: [5, 4, 5, 4, 4]

Prediction: 4.40

# Example Dataset

---



# What Makes a Good ML Model?



- We learn through tutorials and practice problems. Hopefully, we develop understanding of the subject.
- To pass an exam, we are likely to be tested on unseen exam questions, so memorising the practice ones will not work.

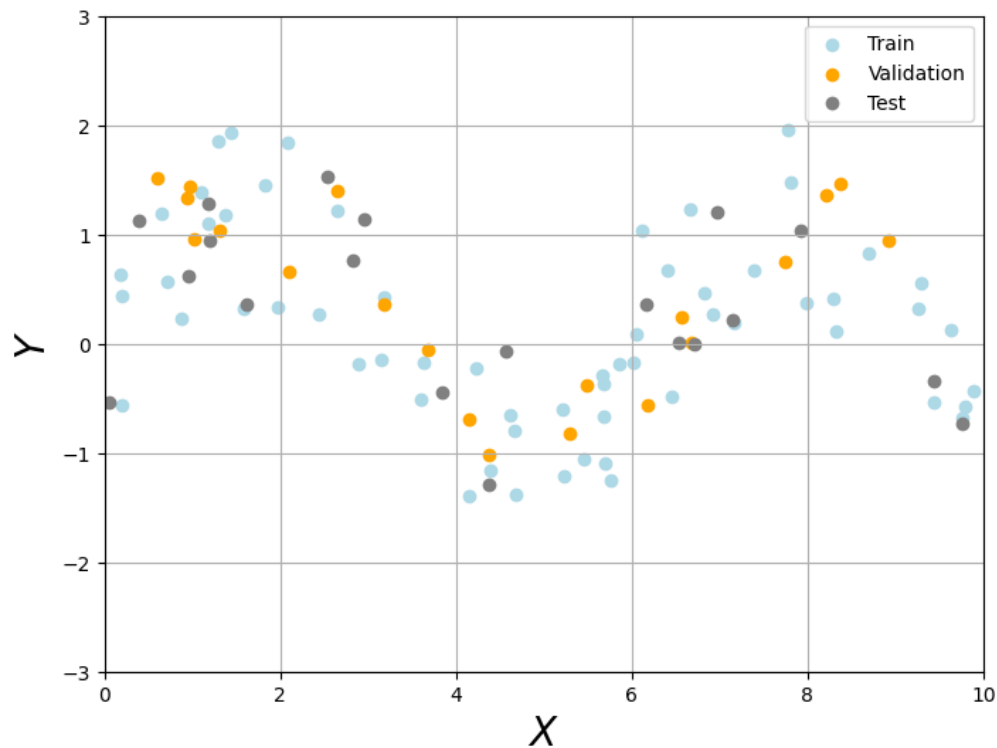
- Like wise, a good classifier is the one that consistently performs well on both training data and also an unseen dataset.
- The model must learn underlying relationships in the data, instead of memorising it.

# Train, Validation and Test Datasets

```
from sklearn.model_selection import train_test_split

# First, split the data into train (60%) and temp (40%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)

# Then split the temp data into validation (50% of 40%  $\Rightarrow$  20% of total) and test (remaining 50% of 40%  $\Rightarrow$  20% of total)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```



- (Train+Validation):Test is typically either 0.8:0.2 or 0.7:0.3.
- **Test** dataset is a proxy of unseen data, and it will only be used in the final evaluation.
- We train our ML model on the **Train** dataset.
- **Validation** dataset is used to fine-tune or optimise the ML model. Here, we find a set of hyper-parameters, e.g.  $k$  in the kNN context, that will result in the best performance on the Validation dataset.



# scikit-learn: KNeighborsRegressor

```
from sklearn.neighbors import KNeighborsRegressor
from sklearn.metrics import mean_squared_error

# Define k and the distance metric
k = 5
distance_metric = 'euclidean'

# Create an instance of KNeighborsRegressor
knn = KNeighborsRegressor(n_neighbors=k, metric=distance_metric)

# Fit the model on the training data
knn.fit(X_train, y_train)

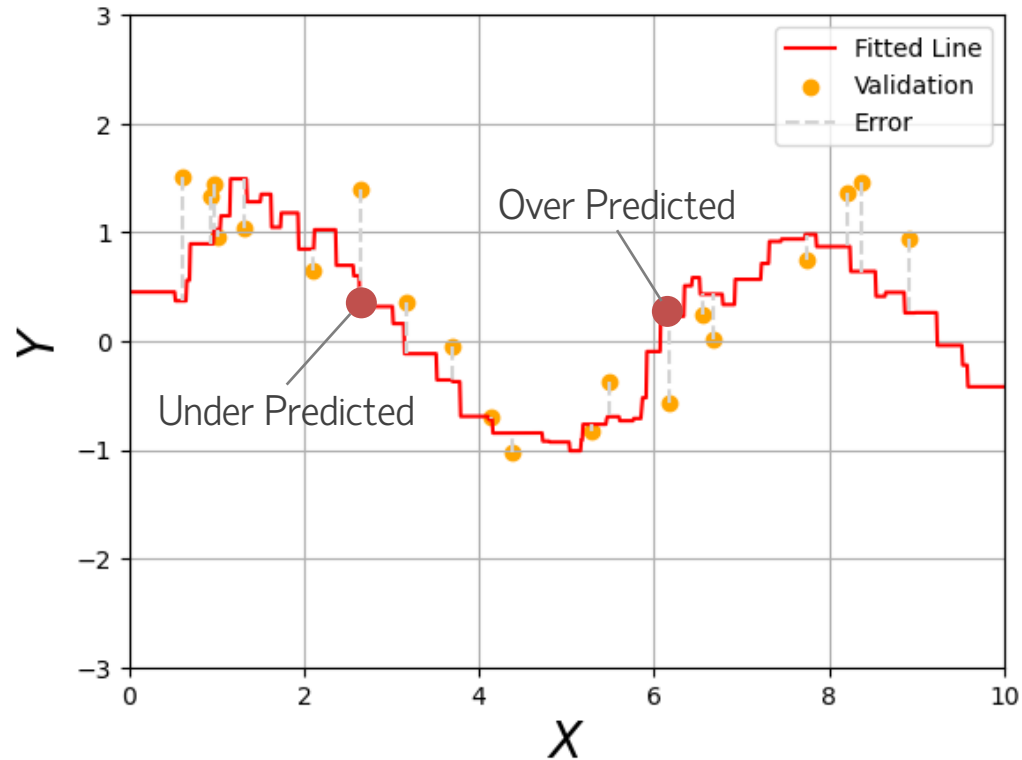
# Predict the target values on the validation set
predictions = knn.predict(X_val)

# Calculate the mean squared error of the predictions
mse = mean_squared_error(y_val, predictions)
print("Mean Squared Error of the KNN regressor on the validation set: {:.2f}".format(mse))
```

```
Mean Squared Error of the KNN regressor on the validation set: 0.29
```



# Regression Line



$$\text{Error (Residue)} = y - \hat{y}$$

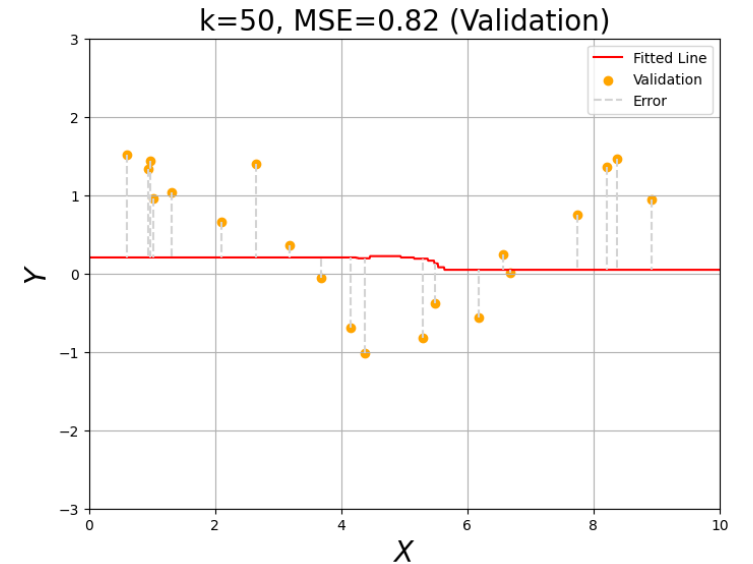
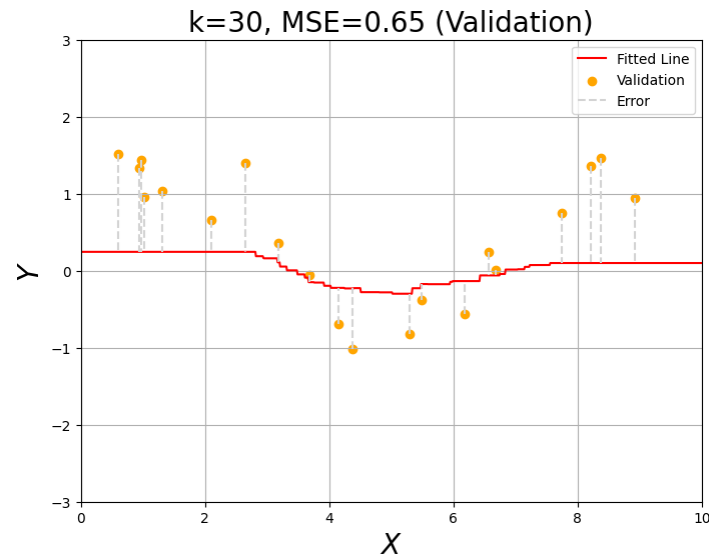
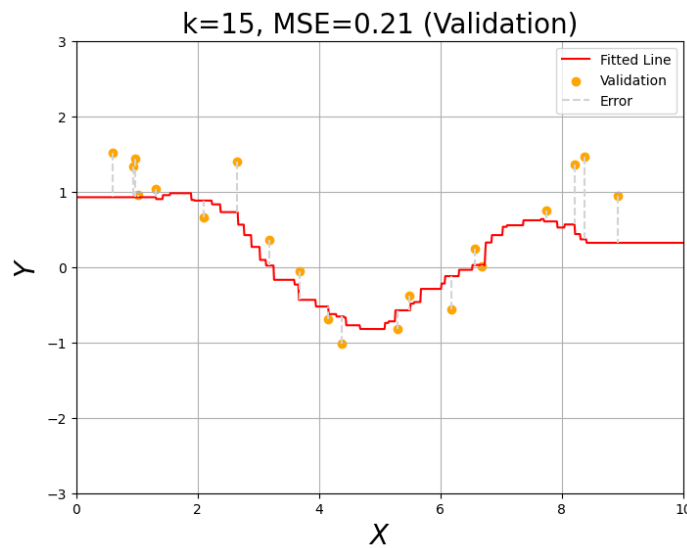
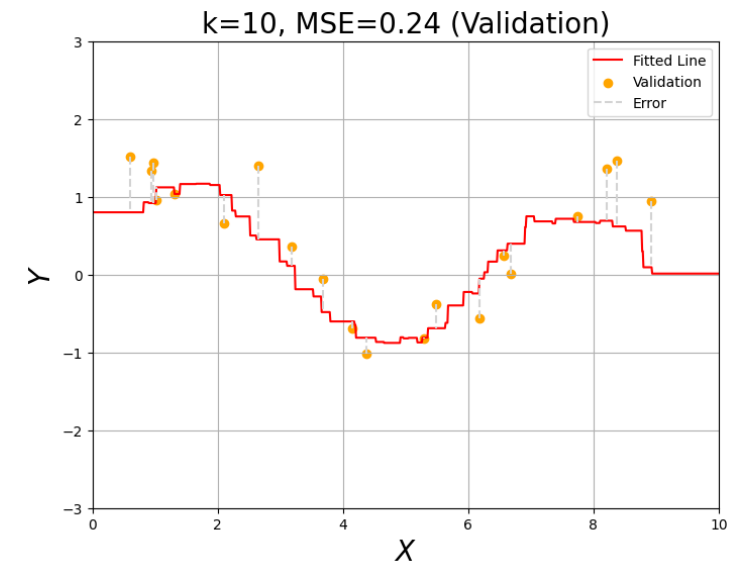
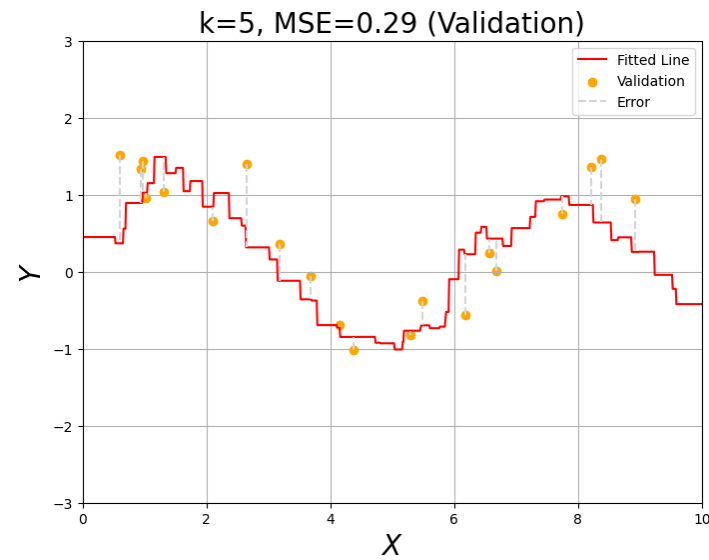
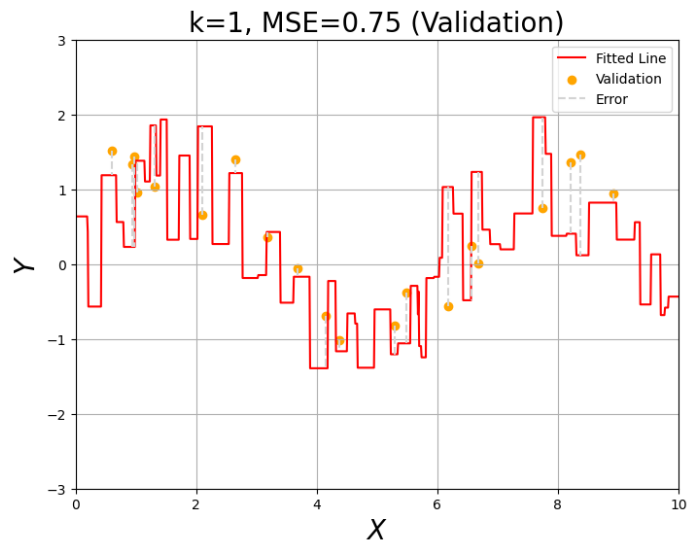
where  $y$  and  $\hat{y}$  are the observed and predicted values, respectively.

Mean Squared Error (MSE):

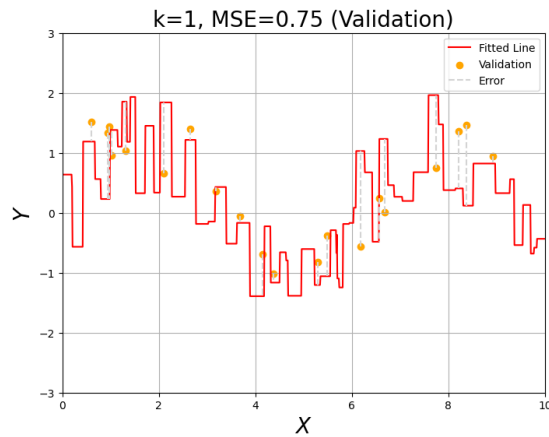
$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y^{(i)} - \hat{y}^{(i)})^2$$

where  $N$  is the total number of data points.

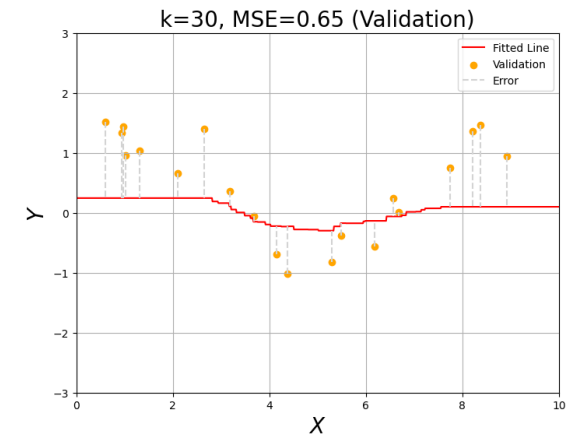
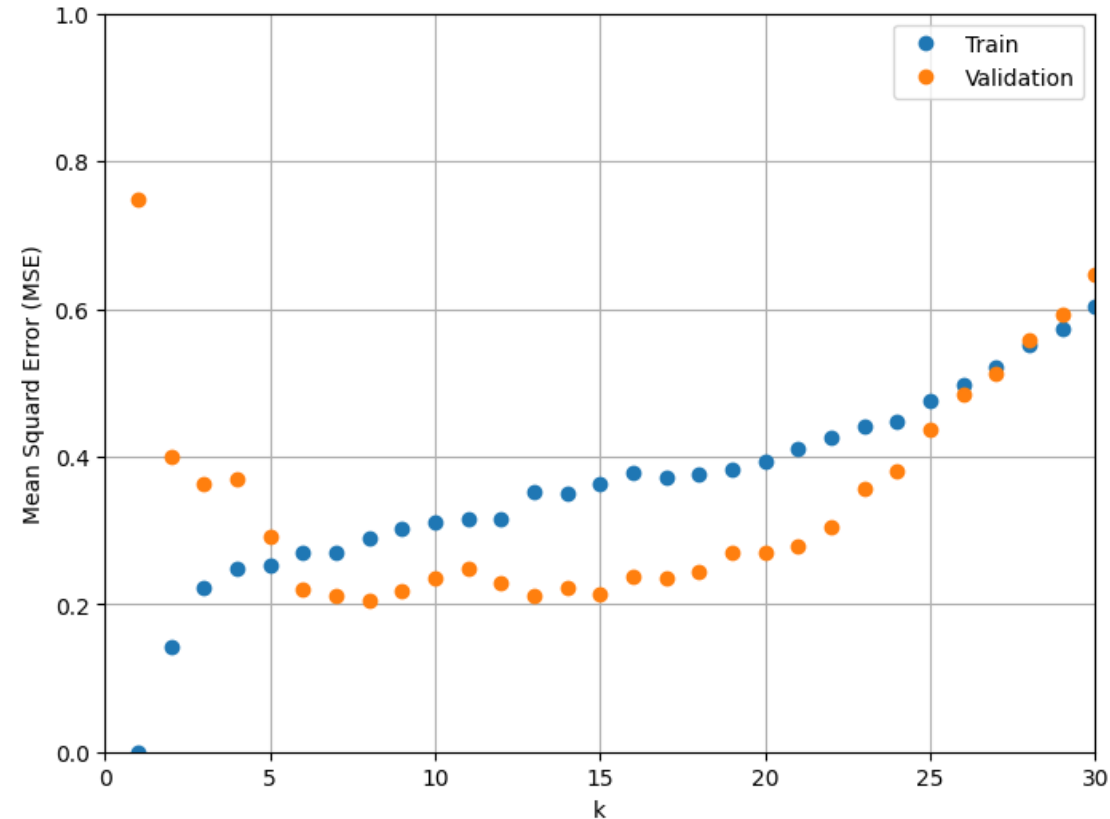
# Model Complexity



# Bias-Variance Trade-Offs



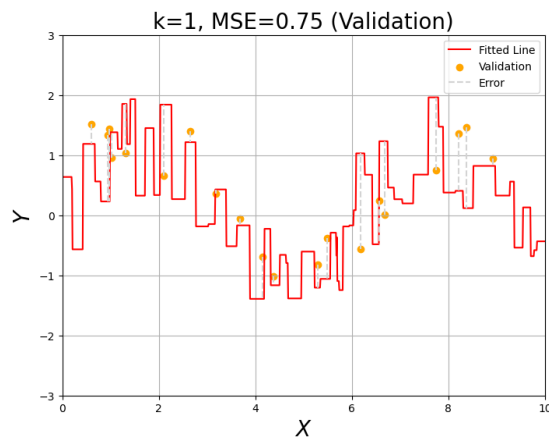
- High Variance, Low Bias
- Complex Fitted Line
- Low Train MSE
- High Validation MSE



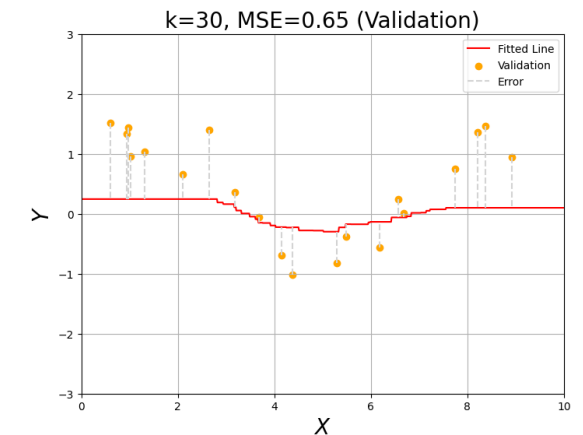
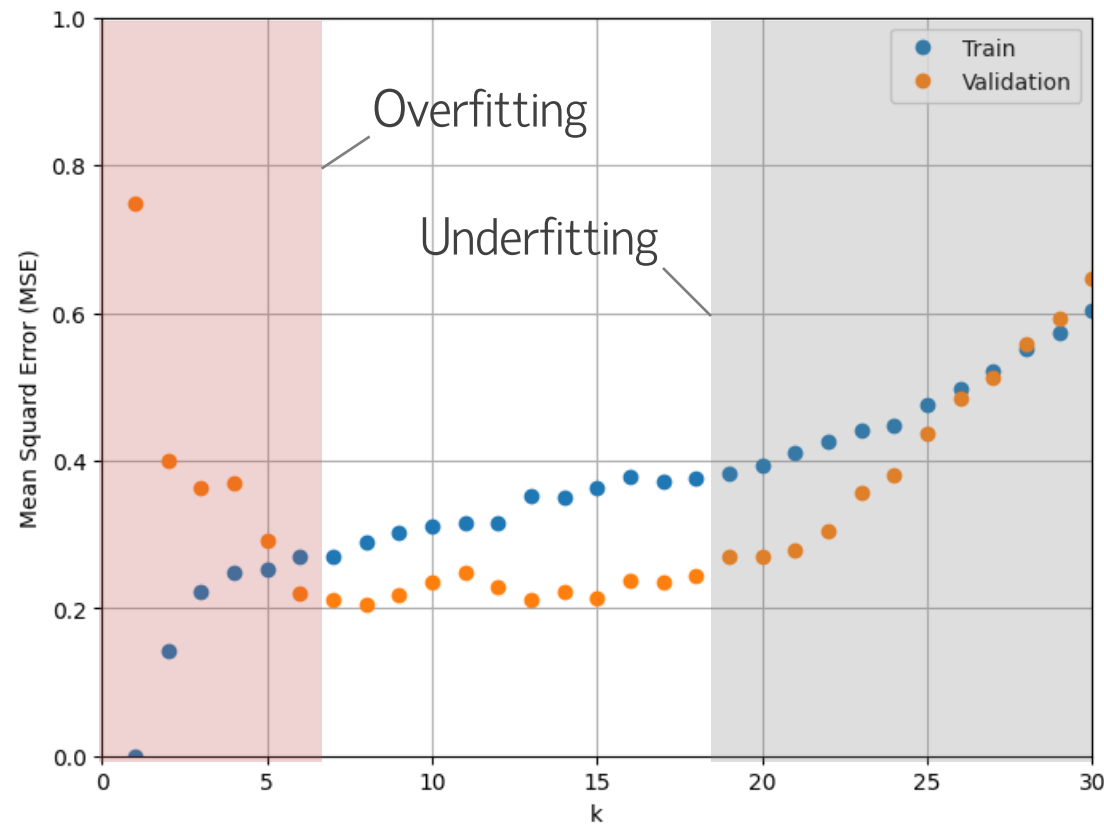
- High Bias, Low Variance
- Simple Fitted Line
- High Train MSE
- High Validation MSE

# Overfitting vs Underfitting

- **Overfitting** occurs with a small  $k$ , where the model captures noise in the training data as if it were significant signal. This results in a model that performs well on the training data but poorly on validation data.
- **Underfitting** arises with a large  $k$ , leading to a model that is too generalized. It overlooks important patterns in the data, causing subpar performance on both the training and validation datasets as it fails to capture the underlying data structure.



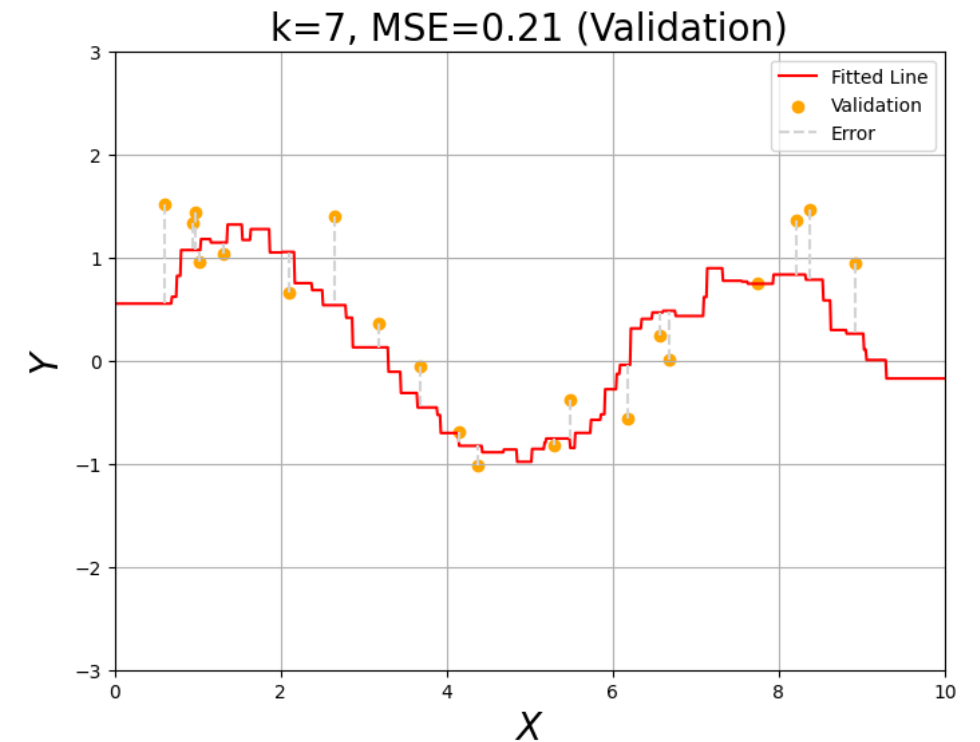
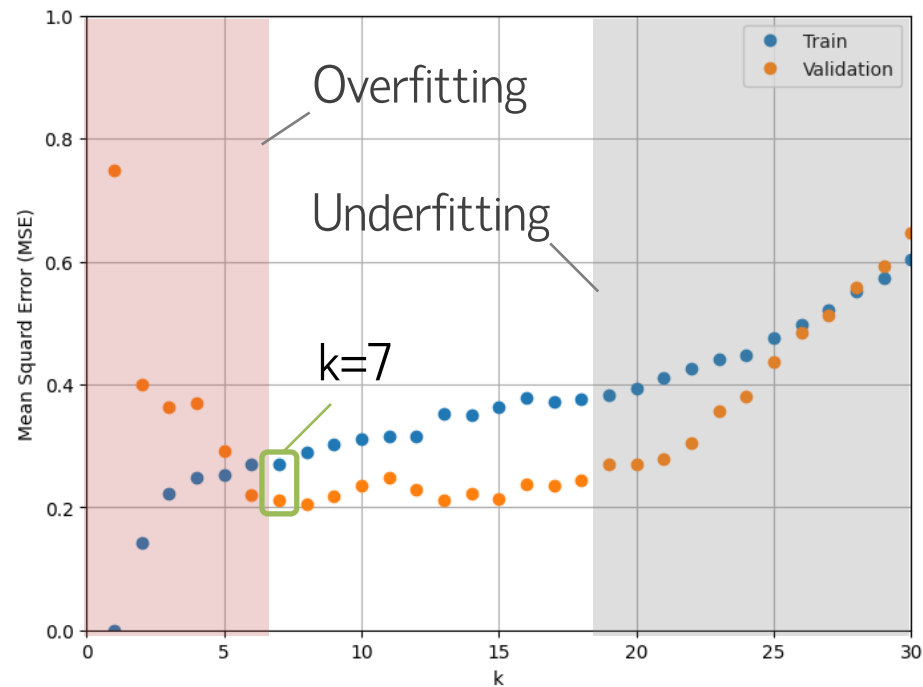
- High Variance, Low Bias
- Complex Fitted Line
- Low Train MSE
- High Validation MSE



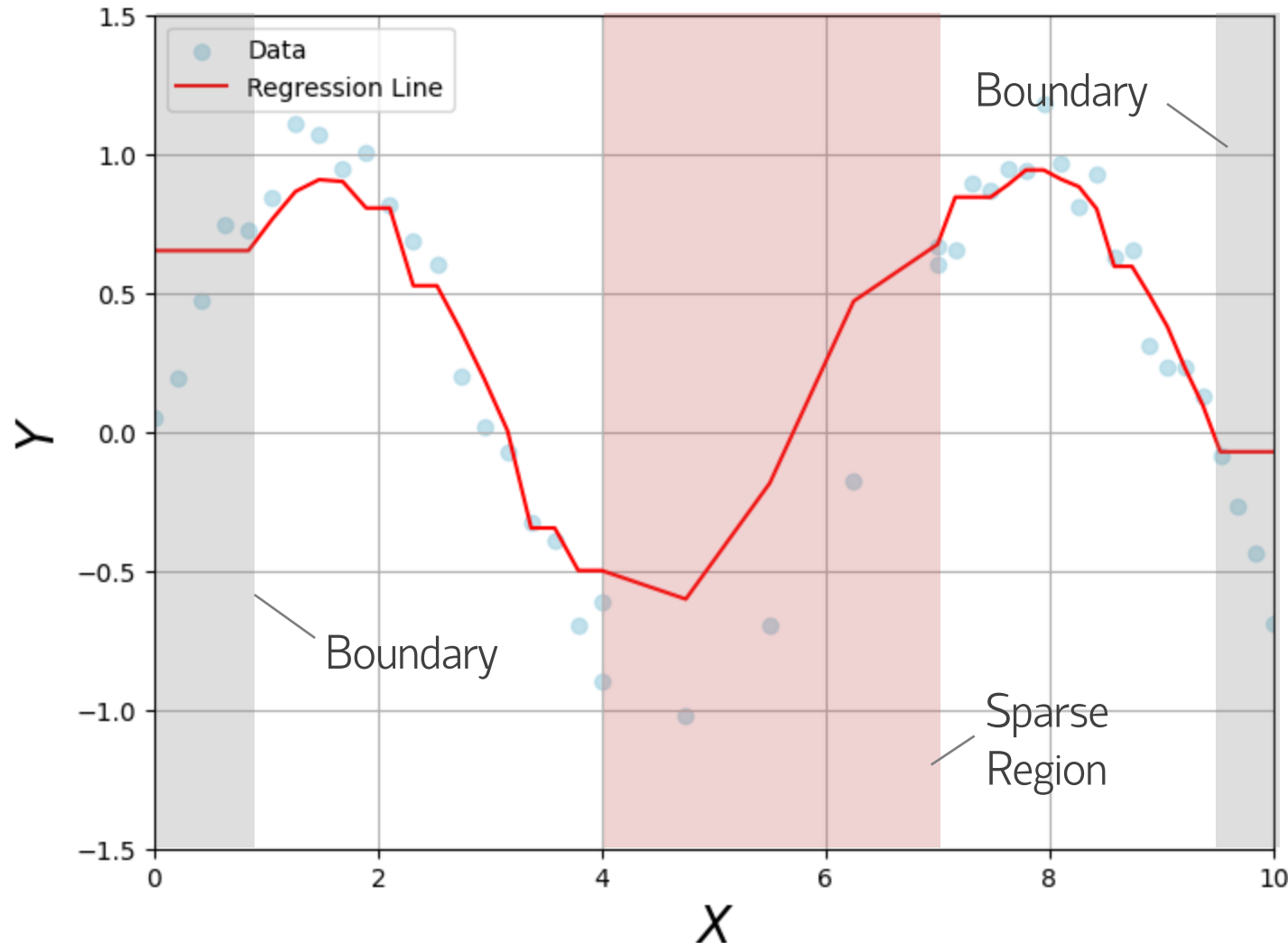
- High Bias, Low Variance
- Simple Fitted Line
- High Train MSE
- High Validation MSE

# Optimal Model

- An optimal k-NN model is one where the choice of k strikes a balance between fitting the training data and generalizing to new data. The right k allows the model to smooth out noise and capture the underlying trends, leading to accurate predictions on both the training and unseen data, indicating it has avoided overfitting and underfitting.

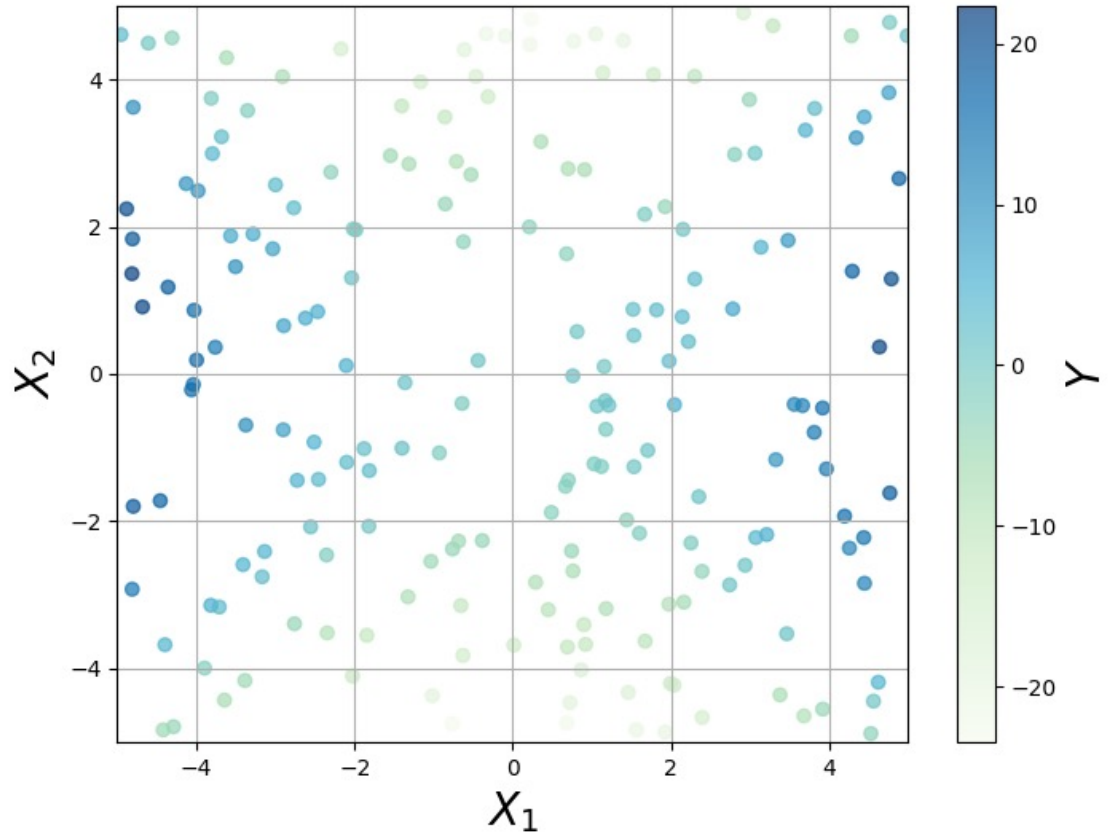
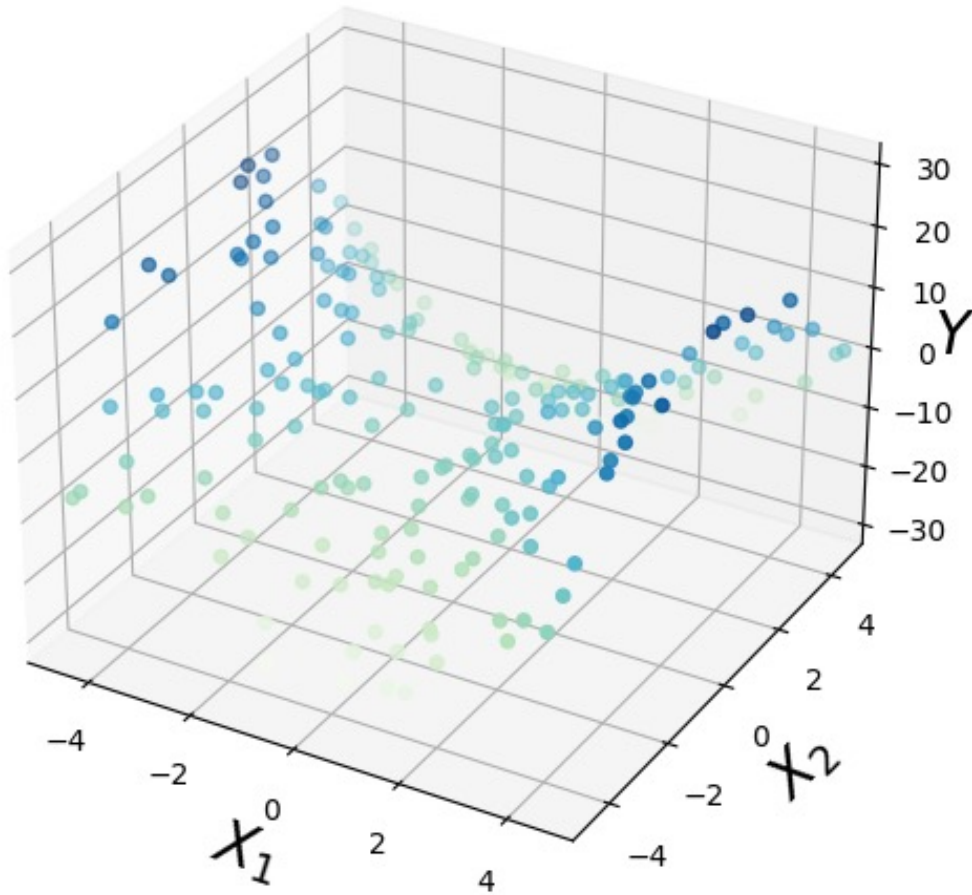


# KNN in Practice



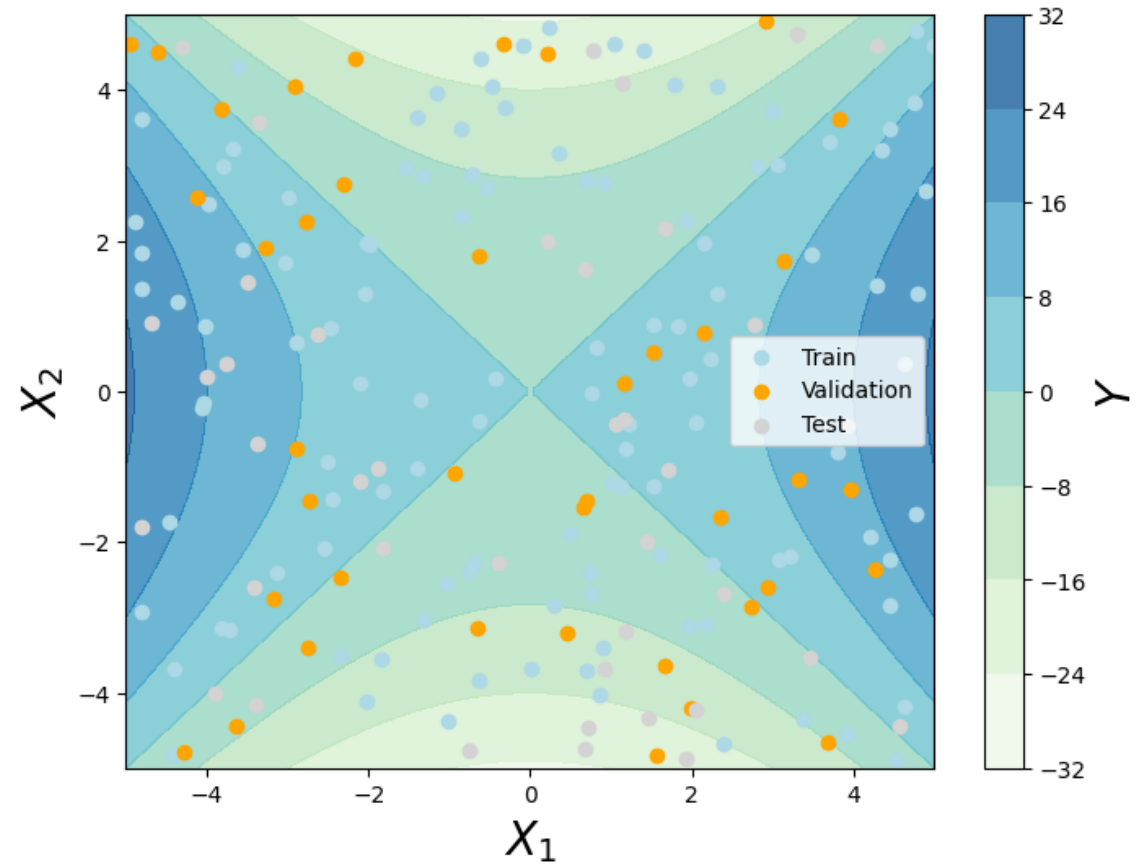
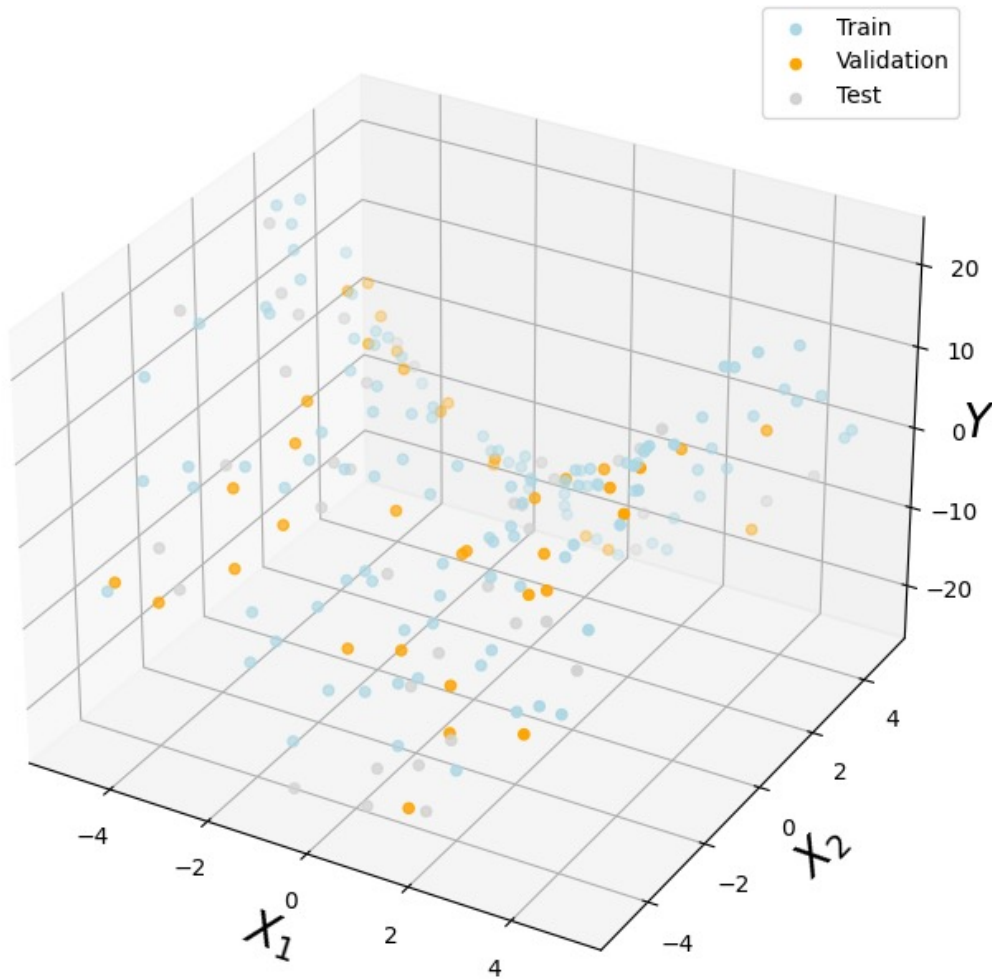
- The area, where there are high output variations among the neighbours, is where the residues what observed and what predicted are likely to to be high. Hence, this is where regression errors are coming from.
- On the contrary, we can predict with low residue errors in the area where this is no (or little) variation in the target variable between the neighbouring points.

# High Dimesional Dataset

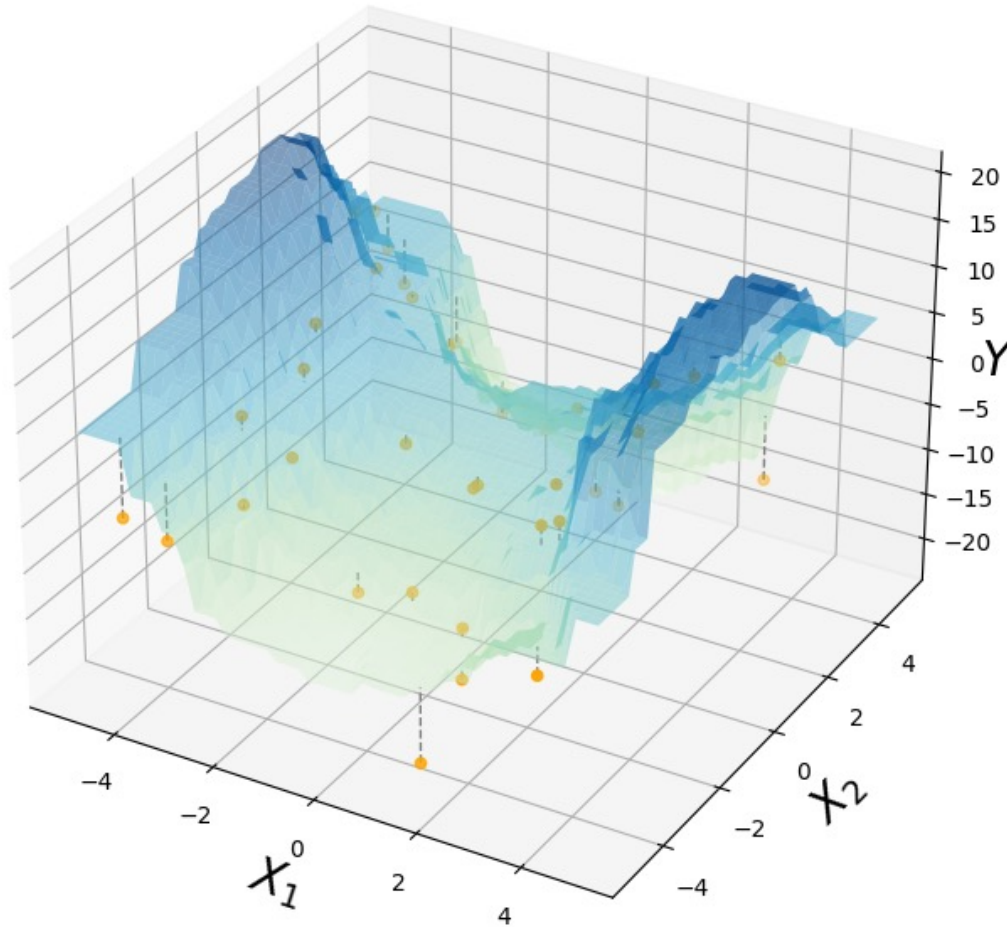




# Train-Test Split



# Regression Surface

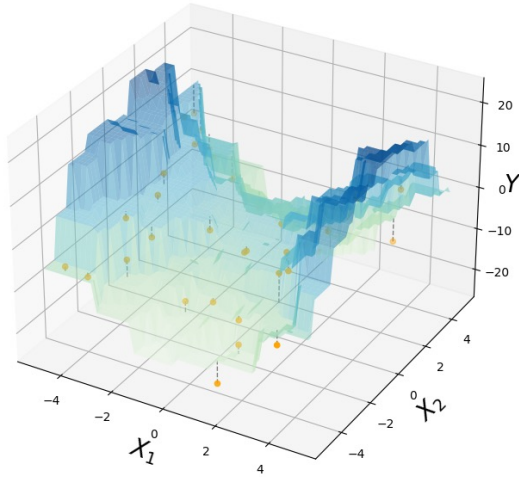


- It is similar to a regression line, but for higher dimensional data. The surface can be curved or flat, depending on the nature of the relationship.

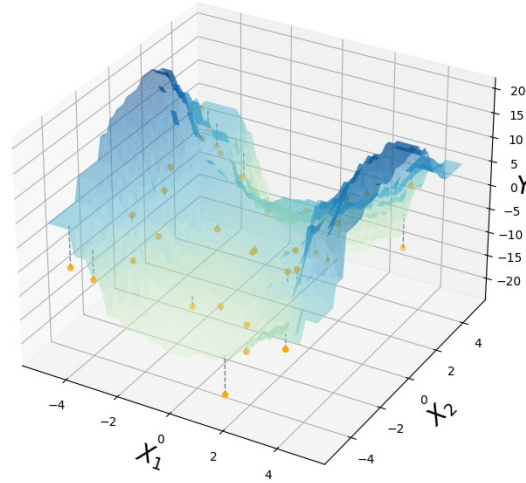
$K = 5$ ,  $MSE = 9.7$  (Validation)

# Model Complexity

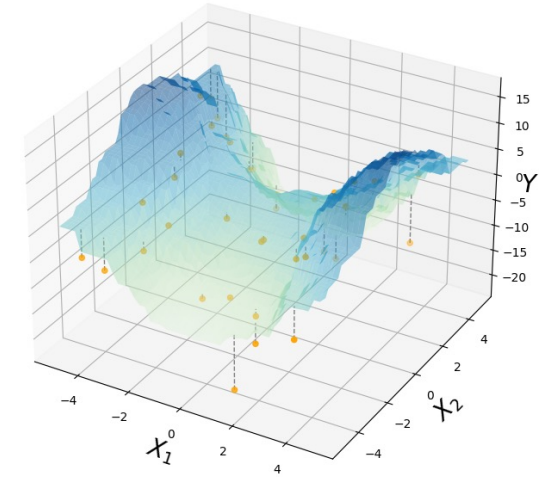
$k=1$ , MSE=12.14 (Validation)



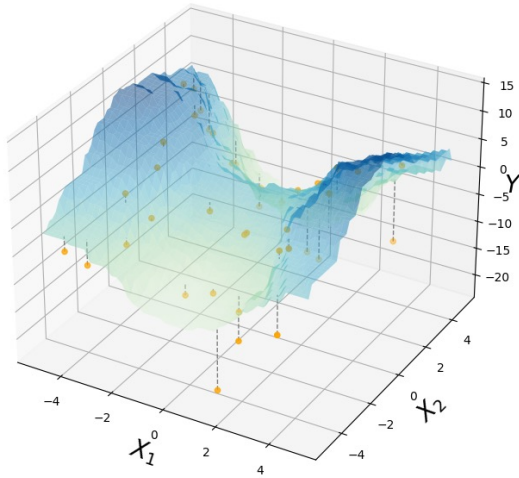
$k=5$ , MSE=9.7 (Validation)



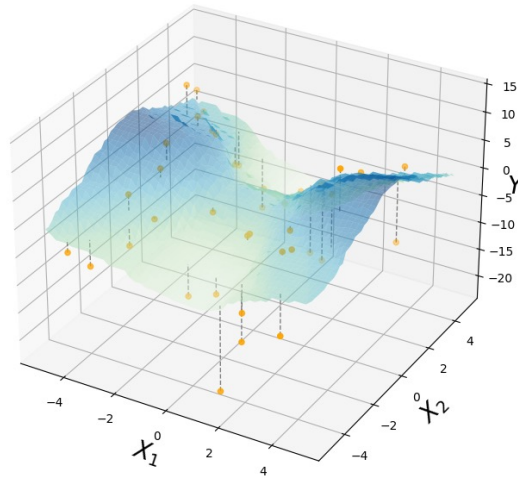
$k=10$ , MSE=17.17 (Validation)



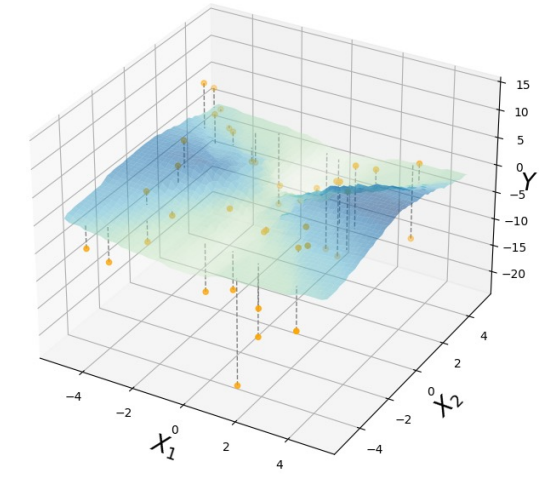
$k=15$ , MSE=17.52 (Validation)



$k=30$ , MSE=33.5 (Validation)

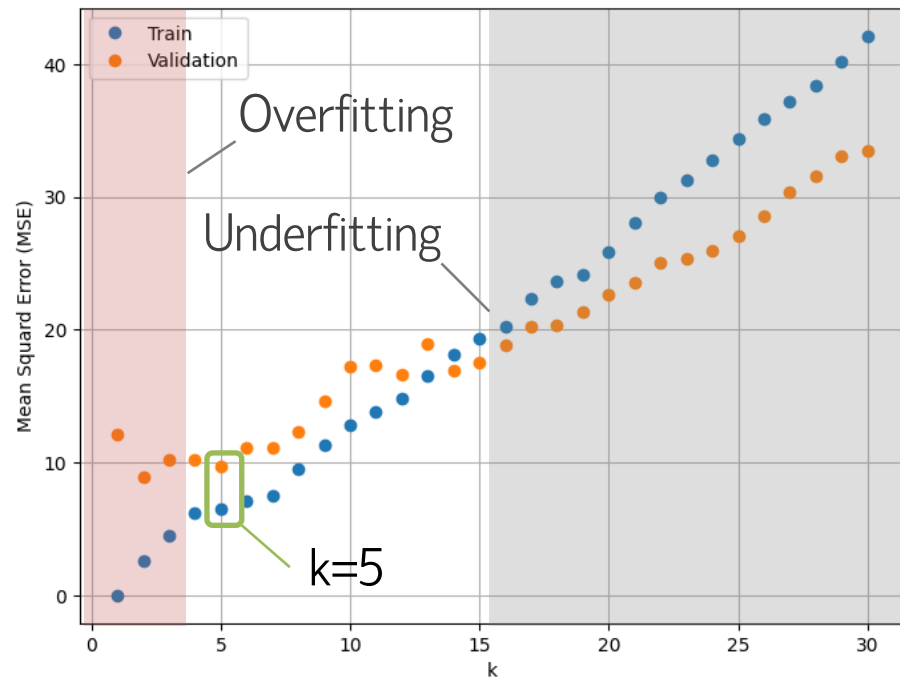


$k=50$ , MSE=59.8 (Validation)

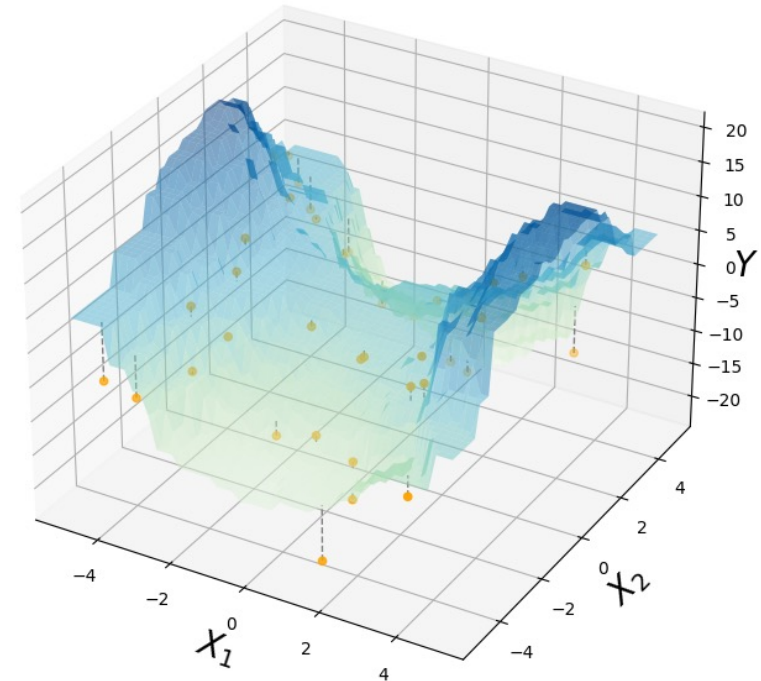


# Optimal Model

- An optimal k-NN model is one where the choice of k strikes a balance between fitting the training data and generalizing to new data. The right k allows the model to smooth out noise and capture the underlying trends, leading to accurate predictions on both the training and unseen data, indicating it has avoided overfitting and underfitting.

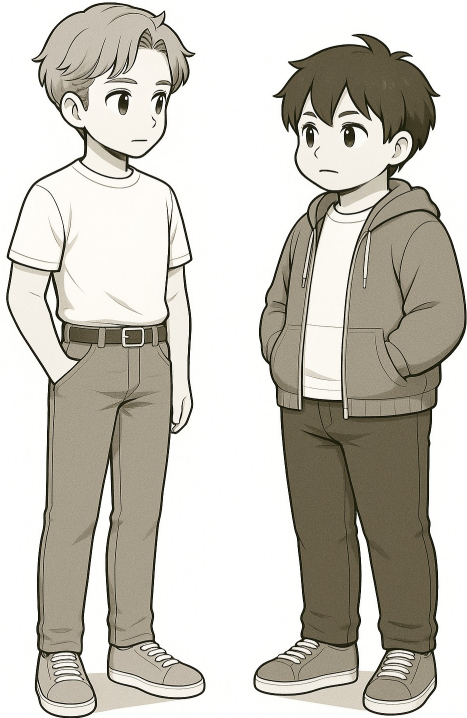


k=5, MSE=9.7 (Validation)



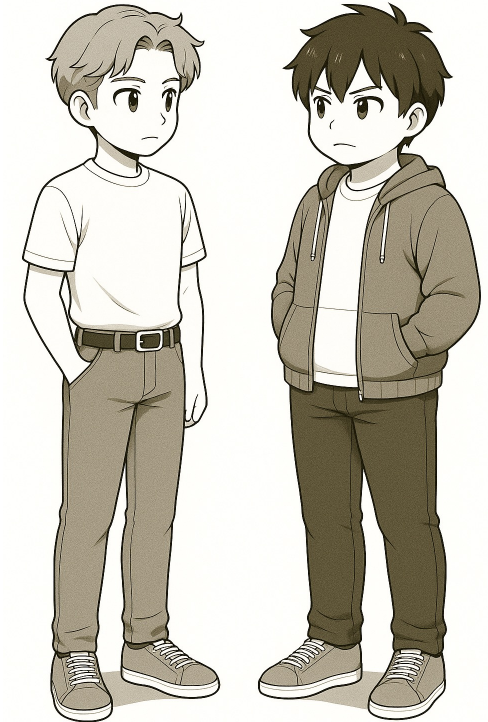


# Scaling Effects



A: (170 cm, 60 kg), B: (160 cm, 60 kg)

Distance(A, B) = 10



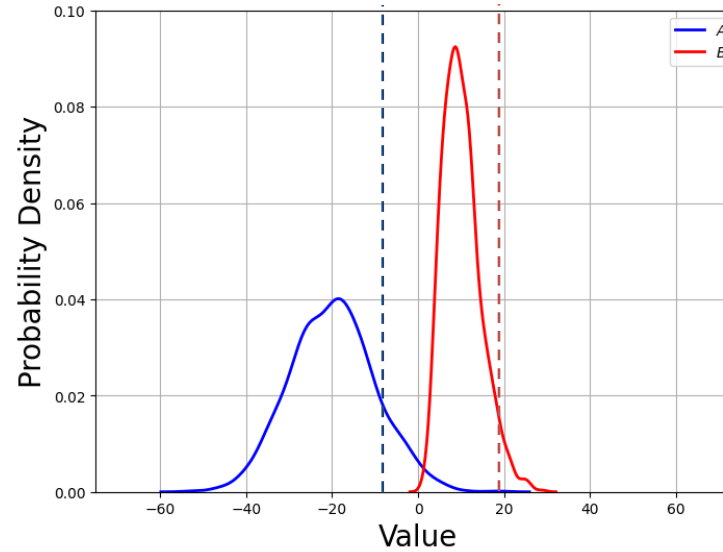
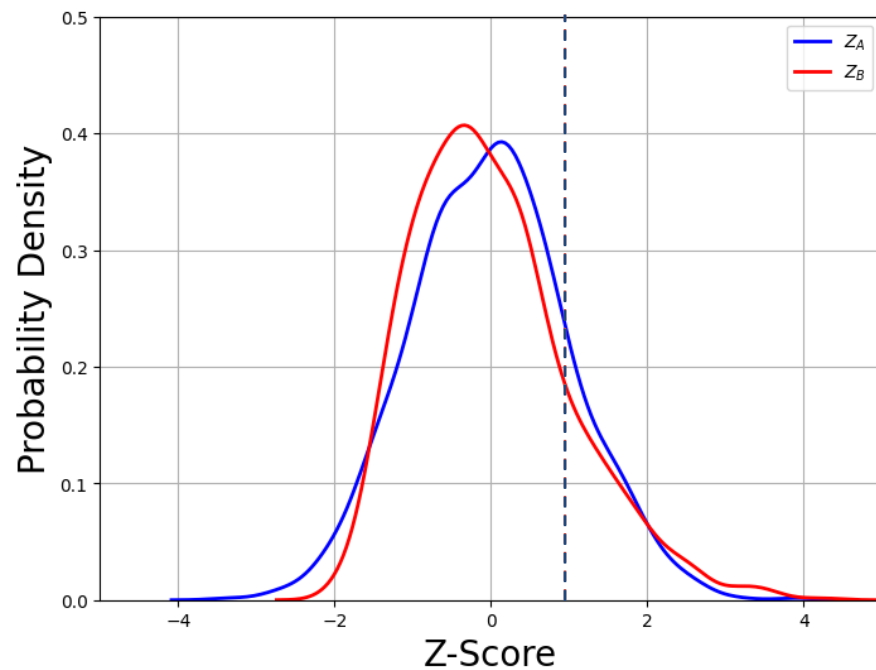
C: (170 cm, 60 kg), D: (170 cm, 70 kg)

Distance(C, D) = 10

- Distance(A, B) = Distance(C, D). Eh?
- We are able to differentiate A from B quite easily. This will not be so obvious between C and D.
- Where possible, we must ensure different features are fairly contributing to the machine learning (ML) model.

# Standardising with Z Scores

$$Z = \frac{x - \mu}{\sigma}$$



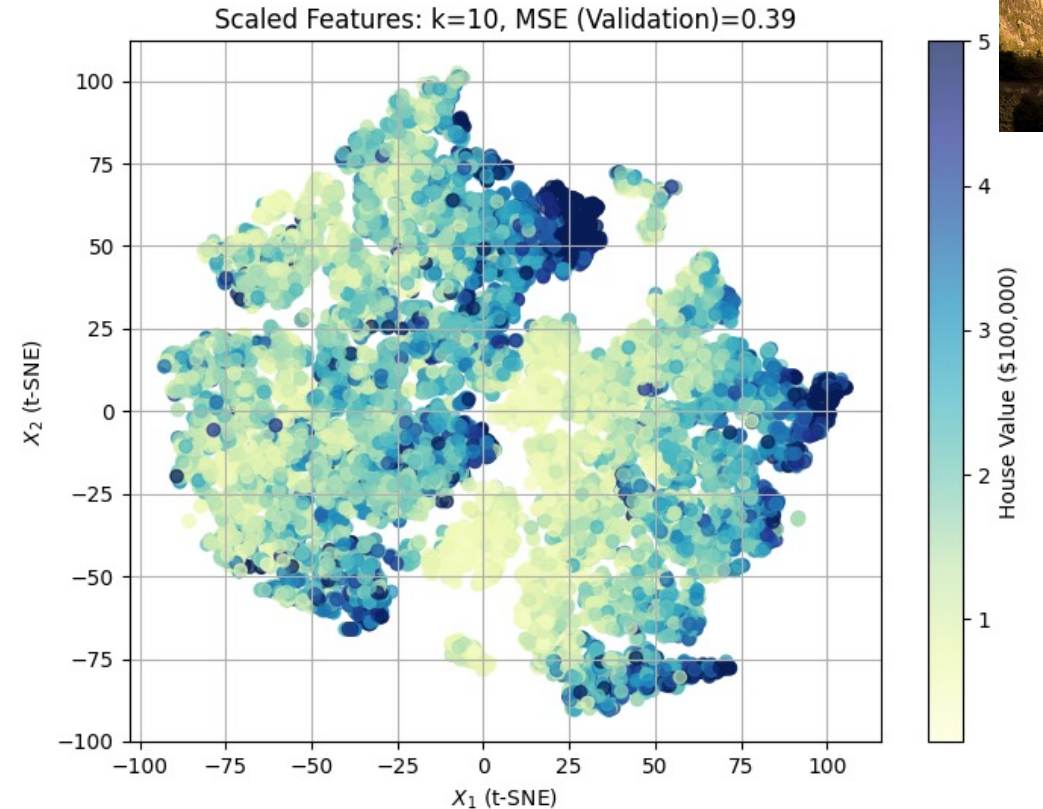
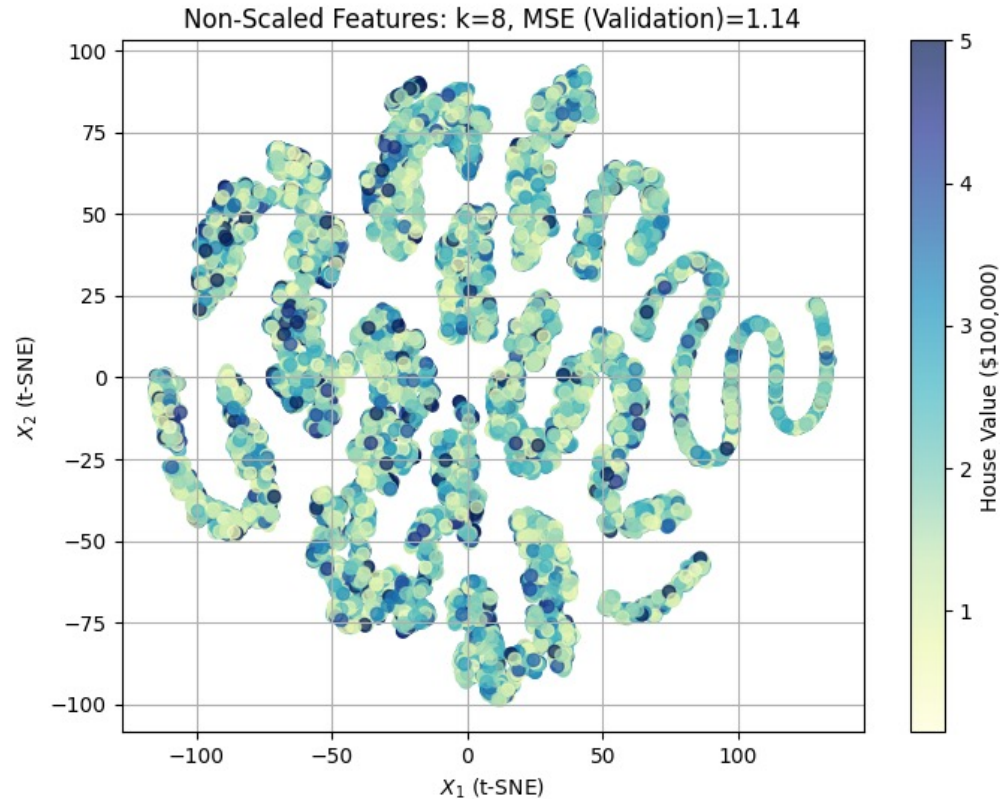
$$\begin{aligned}\mu_A &= -19.81 \\ \sigma_A &= 9.88\end{aligned}$$

$$\begin{aligned}\mu_B &= 10.16 \\ \sigma_B &= 4.40\end{aligned}$$

$$\begin{aligned}Z_A &= \frac{-9.93 - (-19.81)}{9.88} \\ &= 1\end{aligned}$$

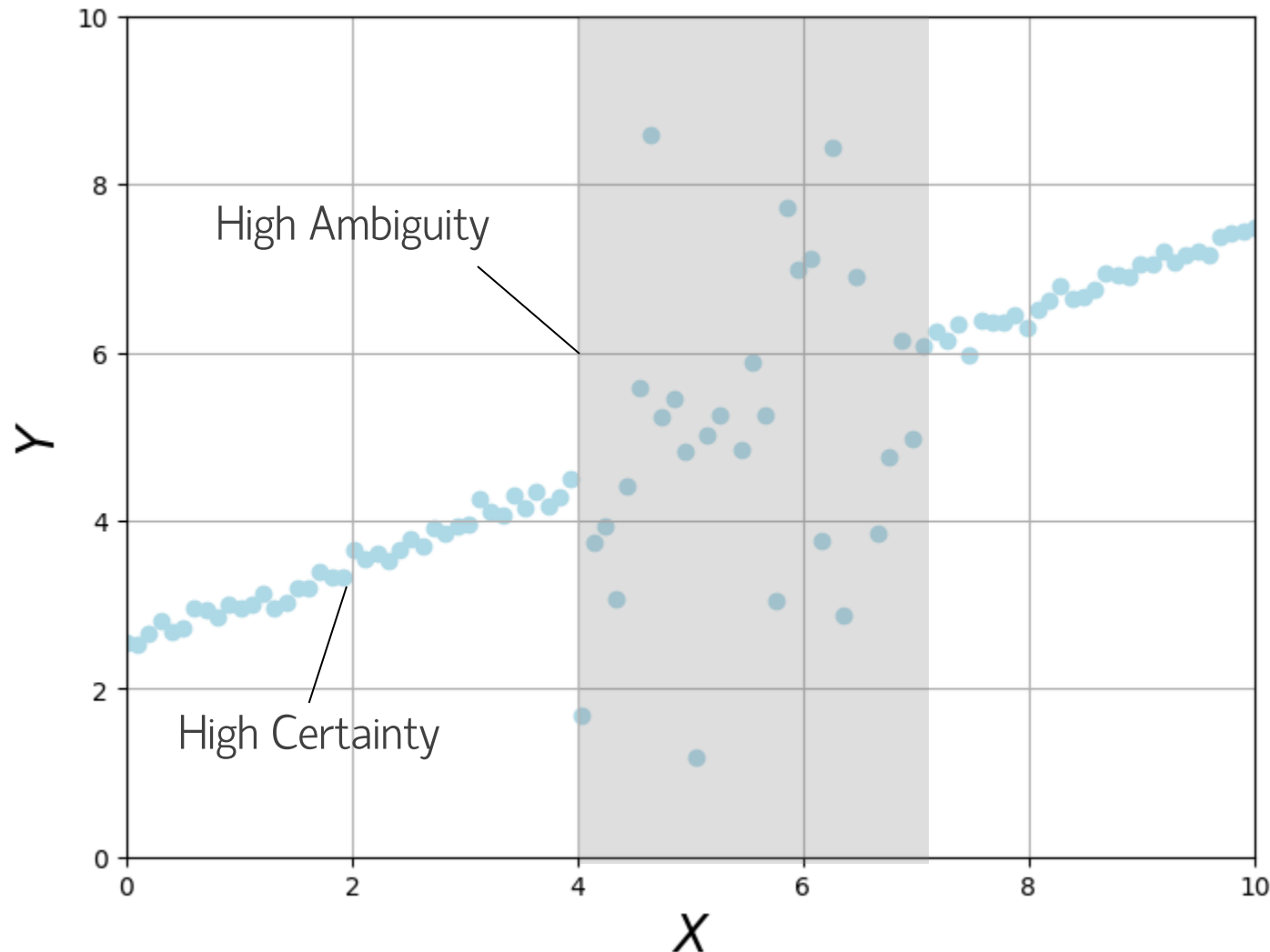
$$\begin{aligned}Z_B &= \frac{14.56 - 10.16}{4.40} \\ &= 1\end{aligned}$$

# California Housing Dataset: Better Predictions with Z-Score Features



- California Housing Dataset: There are 8 features, which include median income, housing age, average rooms and bedrooms per household, population, average occupancy, and geographical coordinates (latitude and longitude) of the block group.
- Standardisation ensures features with different units and scales are contributing fairly into the ML model. This results in better correlation between the distance between 2 data points and their difference in the target values.

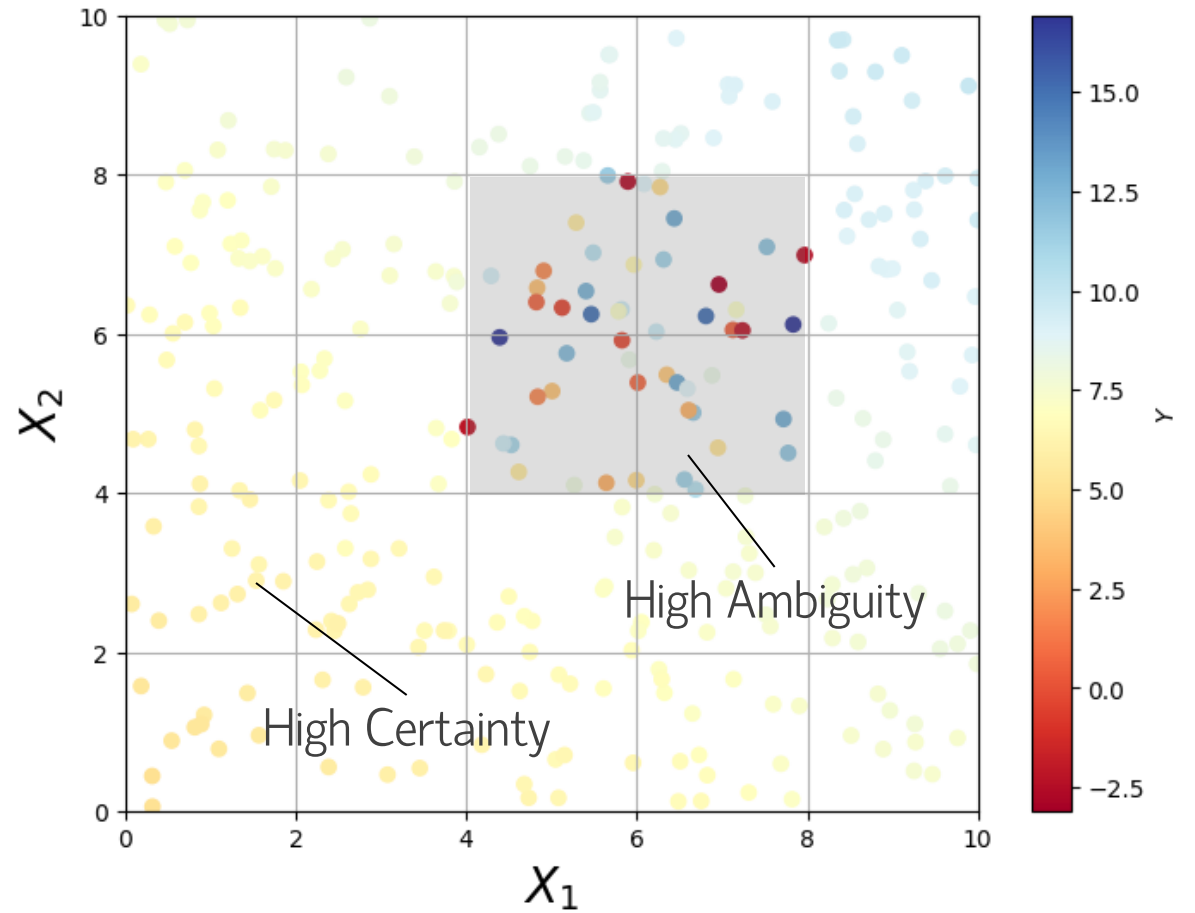
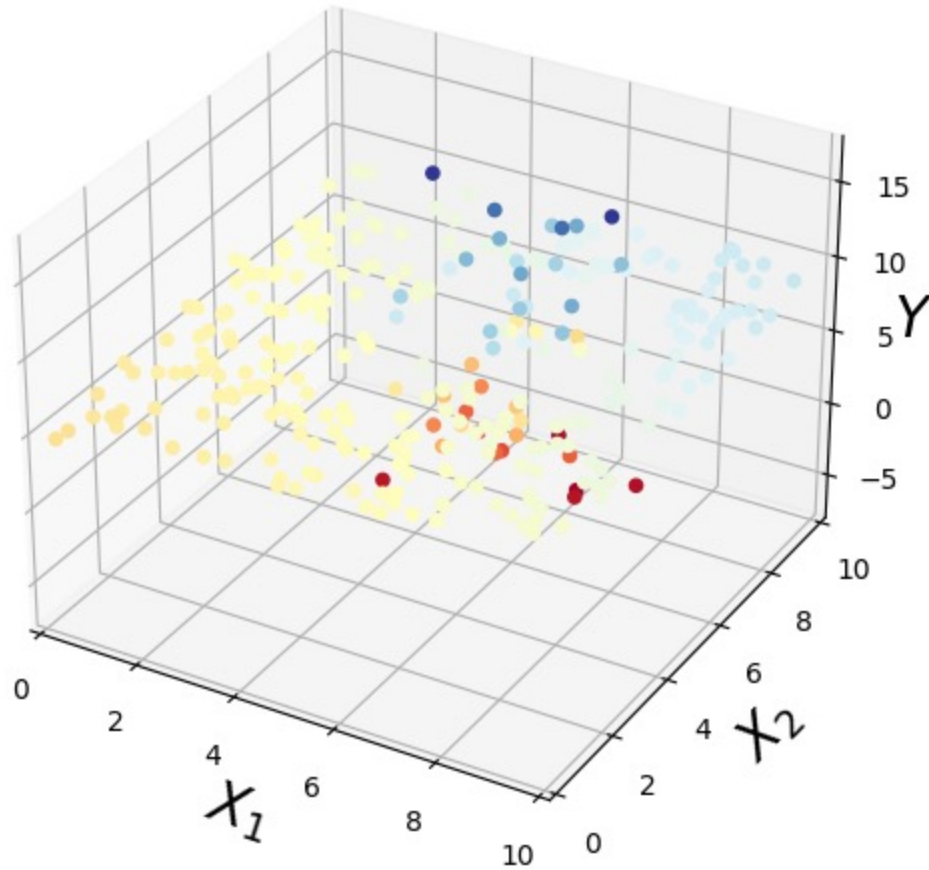
# Where Do Errors Come From?



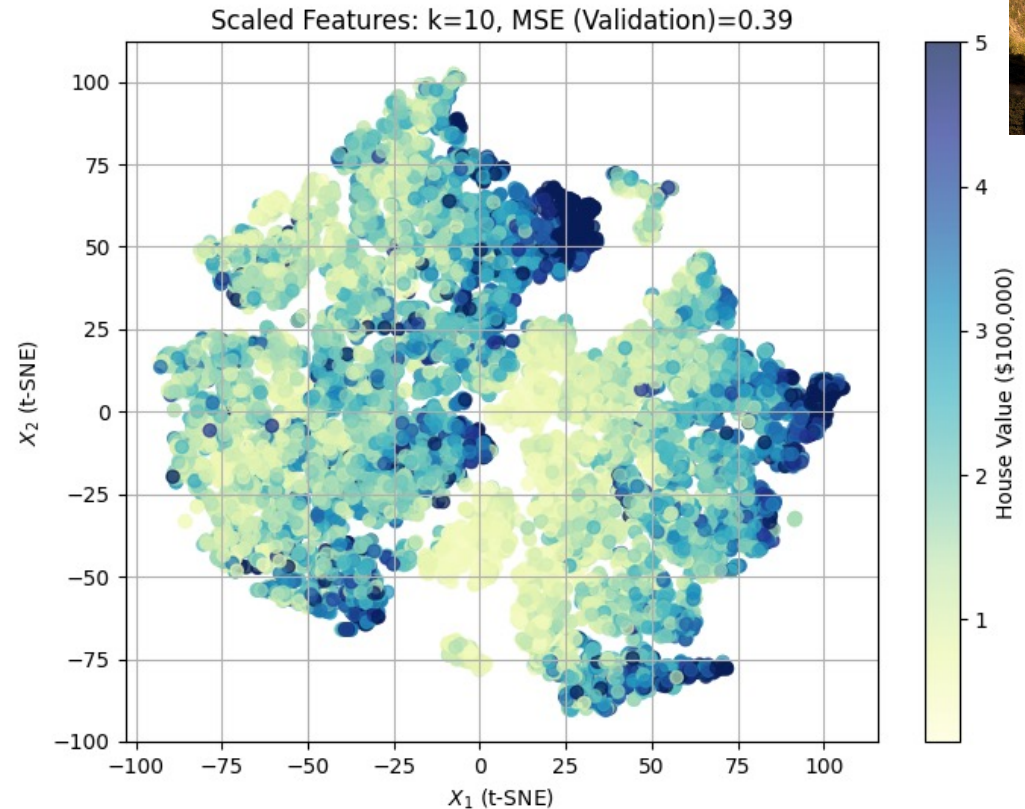
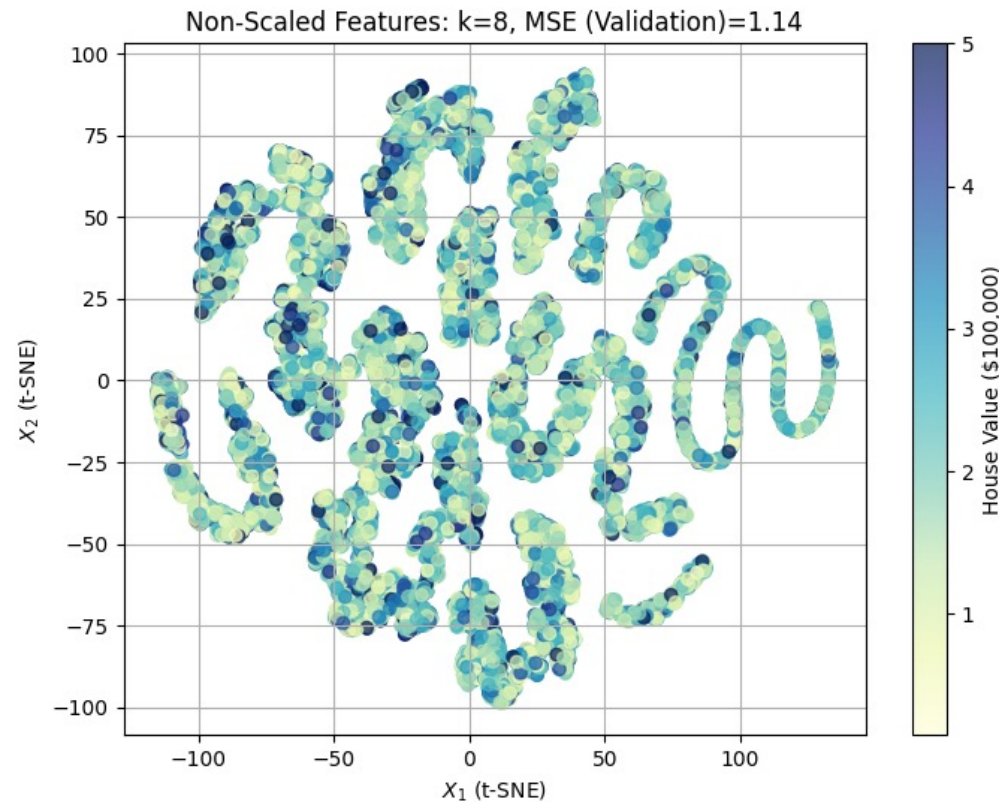
- The area, where there are high output variations among the neighbours. is where the residues what observed and what predicted are likely to to be high. Hence, this is where regression errors are coming from.
- On the contrary, we can predict with low residue errors in the area where this is no (or little) variation in the target variable between the neighbouring points.



# Where Do Errors Come From? (cont.)



# California Housing Dataset: Z-Score Features (Revisit)



- For non-scaled features, there are high variations in term of price among the neighbouring data points. This will result in high prediction errors, hence a high MSE on the validation dataset.
- The house values are similar among the neighbouring data points with the standardised features. The change is also more gradual as the distance increases. We can predict with high certainty, and hence a low MSE.

# Summary

---

- In regression problems, K-Nearest Neighbours (KNN) makes a prediction on query point by averaging the labels of its neighbouring data points. The notion of distance (or similarity) underpins how K-Nearest Neighbours (KNN) works. It defines nearness in the same manner as KNN classification.
- KNN models, or ML in general, are considered good if its prediction performances are consistent between train, validation and test datasets. We need to find a model that is neither overfitting nor underderfitting.
- Low K generally leads to a complex regression surface (or line), i.e. high variance (or low bias). In reverse, K will lead to a simple one, i.e. high bias (or low variance). We need to find K which will result in low validation mean squared error (MSE) and still maintaining low MSE on the training dataset.
- In practice, KNN performs poorly around the boundaries where k neighbouring points are unchanged. It has issues also in the regions where the data points are sparse. Nevertheless, KNN is much more reasonable fit in the presence of noise.
- In practice, we always scale our features before training a model. This often makes neighbouring data points to have comparable labels (or output values), and hence a better prediction performance.