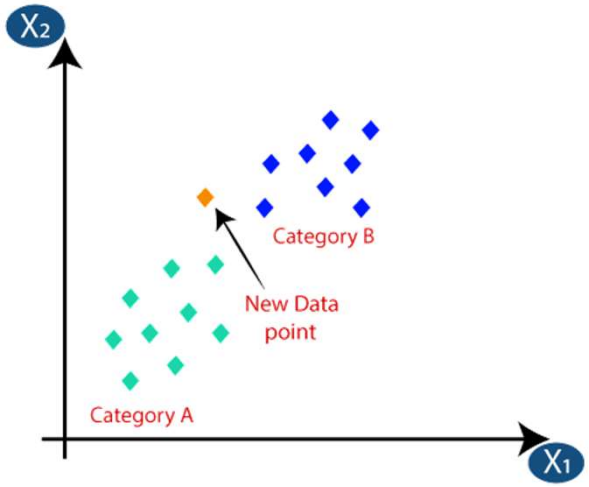
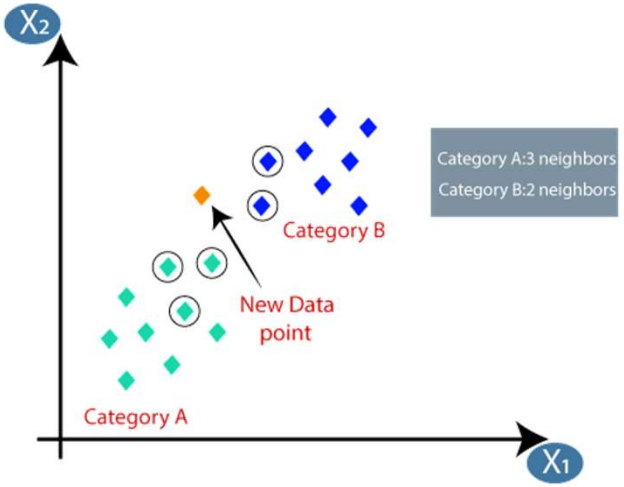
# *k*-Nearest Neighbors (k-NN)

# The *k*-NN Classifier

1. Select the number K of the neighbors
2. Calculate the distance of ***k* number of neighbors**
3. Take the *K* nearest neighbors
4. Among these k neighbors, count the number of the data points in each category.
5. Assign the new data points to that category for which the number of the neighbor is maximum.




Euclidian Distance

$$\sqrt{(x_1 - y_1)^2 + \cdots + (x_n - y_n)^2}$$

It is important to normalize the numerical variables

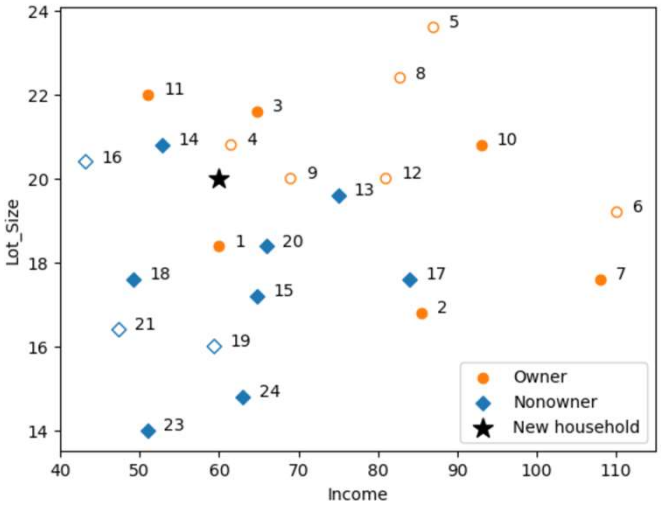
# Riding Mowers

A riding-mower manufacturer would like to find a way of classifying families in a city into those likely to purchase a riding mower and those not likely to buy one. A pilot random sample is undertaken of 12 owners and 12 nonowners in the city.

```
mower_df = pd.read_csv('RidingMowers.csv')
mower_df['Number'] = mower_df.index + 1

trainData, validData = train_test_split(mower_df, test_size=0.4, random_state=123)
mower_df.head()
```

| | Income | Lot_Size | Ownership | Number |
|---|---|---|---|---|
| **0** | 60.0 | 18.4 | Owner | 1 |
| **1** | 85.5 | 16.8 | Owner | 2 |
| **2** | 64.8 | 21.6 | Owner | 3 |
| **3** | 61.5 | 20.8 | Owner | 4 |
| **4** | 87.0 | 23.6 | Owner | 5 |



# Riding Mowers

```
predictors = ['Income', 'Number']

scaler = preprocessing.StandardScaler()
scaler.fit(trainData[['Income', 'Lot_Size']])

# Transform the full dataset
mowerNorm = pd.concat([pd.DataFrame(scaler.transform(mower_df[['Income', 'Lot_Size']]),
                                    columns=['zIncome', 'zLot_Size']), mower_df[['Ownership', 'Number']]], axis=1)
mowerNorm.head()
```

```
trainNorm = mowerNorm.iloc[trainData.index]
validNorm = mowerNorm.iloc[validData.index]
newHouseholdNorm = pd.DataFrame(scaler.transform(newHousehold),
    columns=['zIncome', 'zLot_Size'])

knn = NearestNeighbors(n_neighbors=3)
knn.fit(trainNorm.iloc[:, 0:2])
distances, indices = knn.kneighbors(newHouseholdNorm)

# indices is a list of lists, we are only interested in the first element
trainNorm.iloc[indices[0], :]
```

Neighbors

| | zIncome | zLot_Size | Ownership | Number |
|---|---|---|---|---|
| **13** | -0.956146 | 1.053566 | Nonowner | 14 |
| **0** | -0.535091 | 0.012395 | Owner | 1 |
| **2** | -0.254387 | 1.400623 | Owner | 3 |

# Riding Mowers

```
predictors = ['Income', 'Number']

scaler = preprocessing.StandardScaler()
scaler.fit(trainData[['Income', 'Lot_Size']])

# Transform the full dataset
mowerNorm = pd.concat([pd.DataFrame(scaler.transform(mower_df[['Income', 'Lot_Size']]),
                                    columns=['zIncome', 'zLot_Size']), mower_df[['Ownership', 'Number']]], axis=1)
mowerNorm.head()
```

```
trainNorm = mowerNorm.iloc[trainData.index]
validNorm = mowerNorm.iloc[validData.index]
newHouseholdNorm = pd.DataFrame(scaler.transform(newHousehold),
    columns=['zIncome', 'zLot_Size'])

knn = NearestNeighbors(n_neighbors=3)
knn.fit(trainNorm.iloc[:, 0:2])
distances, indices = knn.kneighbors(newHouseholdNorm)

# indices is a list of lists, we are only interested in the first element
trainNorm.iloc[indices[0], :]
```

Neighbors

| | zIncome | zLot_Size | Ownership | Number |
|---|---|---|---|---|
| **13** | -0.956146 | 1.053566 | Nonowner | 14 |
| **0** | -0.535091 | 0.012395 | Owner | 1 |
| **2** | -0.254387 | 1.400623 | Owner | 3 |

# Riding Mowers

```
train_X = trainNorm[['zIncome', 'zLot_Size']]
train_y = trainNorm['Ownership']
valid_X = validNorm[['zIncome', 'zLot_Size']]
valid_y = validNorm['Ownership']

# Train a classifier for different values of k
results = []
for k in range(1, 15):
    knn = KNeighborsClassifier(n_neighbors=k).fit(train_X, train_y)
    results.append({
        'k': k,
        'accuracy': accuracy_score(valid_y, knn.predict(valid_X))
    })

# Convert results to a pandas data frame
results = pd.DataFrame(results)
print(results)
```

| | k | accuracy |
|---|---|---|
| 0 | 1 | 0.8 |
| 1 | 2 | 0.7 |
| 2 | 3 | 0.8 |
| 3 | 4 | 0.7 |
| 4 | 5 | 0.7 |
| 5 | 6 | 0.4 |
| 6 | 7 | 0.5 |
| 7 | 8 | 0.4 |
| 8 | 9 | 0.7 |
| 9 | 10 | 0.5 |
| 10 | 11 | 0.7 |
| 11 | 12 | 0.4 |
| 12 | 13 | 0.4 |
| 13 | 14 | 0.4 |

All data

```
# Retrain with full dataset
mower_X = mowerNorm[['zIncome', 'zLot_Size']]
mower_y = mowerNorm['Ownership']
knn = KNeighborsClassifier(n_neighbors=3).fit(mower_X, mower_y)

distances, indices = knn.kneighbors(newHouseholdNorm)
print(knn.predict(newHouseholdNorm))
print('Distances',distances)
print('Indices', indices)
print(mowerNorm.iloc[indices[0], :])
```

```
['Owner']
Distances [[0.35797119 0.52631868 0.54565179]]
Indices [[ 3  8 13]]
     zIncome  zLot_Size Ownership  Number
3  -0.447371   1.053566     Owner       4
8  -0.008772   0.706509     Owner       9
13 -0.956146   1.053566  Nonowner      14
```

# The $k$-NN Classifier: other versions

**More than two classes**

1) new record is classified as a member of the majority class of its k neighbors, or

2) when there is a specific class that we are interested in identifying, the proportion of the k neighbors that belong to this class of interest can be used as an estimate of the probability (propensity) that the new record belongs to that class, and then comparing with an user-specified cutoff value we decide whether to assign the new record to that class.

**Categorical variables**

1) Should be replaced by dummies variables

2) All categories of the categorical variable should be considered

# The *k*-NN Classifier for a Numerical Outcome

1) Determine the *k* neighbors by computing distances

2) Calculate the (weighted) average outcome value of the k-nearest neighbors to determine the prediction

The weights of the weighted average decrease with increasing distance from the point at which the prediction is required.