# Assignment 4 – Siddharth Bahekar – 002417718

## Question 1 -

1. The lack of a data dictionary is a significant issue. Without clear descriptions of each variable, it's challenging to understand the meaning and potential impact of various features. This can lead to misinterpretation and suboptimal model performance. There are some missing values in the 'mvsf_br_amt_current_month' column of the master target dataset. It's crucial to understand why these values are missing and decide on an appropriate strategy to handle them. There's no clear time component in the data, which could be valuable for churn prediction. If possible, including historical data or time-based features could improve the model's predictive power.

2. Given that both Logistic Regression and SVM have achieved the highest accuracy (96.09%), you can consider either of these models as the best choice for this specific problem. If computational efficiency is a concern, you might prefer Logistic Regression since it is generally faster to train compared to SVM. For this dataset, Logistic Regression or SVM would be the best choice due to their high accuracy and generalizability.

3. According to the logistic regression model the features affecting churn the most are :

| | |
|---|---|
| rgnl_cust_prov_state_cd_mb_ind_current_month | -0.334036 |
| billg_prov_state_cd_on_ind_current_month | 0.298677 |
| cbu_cust_prov_state_cd_bc_ind_current_month | -0.255307 |
| billg_prov_state_cd_qc_ind_current_month | 0.218998 |
| mvsf_mrc_current_month | 0.197715 |

4. Jupyter notebook is attached named “Question 1.ipynb”.

## Question 2 -

Both algorithms could learn the linear boundary, a decision tree would likely do so more efficiently and produce a more compact, efficient model that closely matches the known structure of the target function. The decision tree could learn the exact linear boundary with just a few splits, whereas K-NN would require storing all points to approximate the boundary. If the dataset was very small or if there was reason to believe the true boundary might deviate slightly from a perfect line, K-NN could potentially offer some advantages in terms of flexibility. Given the stated knowledge that the target function is a line, the decision tree's ability to directly model this structure makes it the best choice for this specific problem.

## Question 3 -

1. L1 (Lasso) and L2 (Ridge) regularization are techniques used to prevent overfitting in machine learning models. The key differences are:
   - Penalty term: L1 adds the sum of absolute values of coefficients, while L2 adds the sum of squared values
   - Effect on coefficients: L1 tends to force some coefficients to exactly zero, while L2 shrinks coefficients towards zero but rarely to exactly zero
   - Feature selection: L1 performs implicit feature selection by eliminating less important features, while L2 retains all features

   L1 regularization results in sparse models because:
   - The L1 penalty is not differentiable at zero, creating a sharper gradient that pushes coefficients to exactly zero more readily
   - L1 regularization causes coefficients to converge to zero more quickly than L2, as the derivative of the L1 penalty is constant ($\lambda$) while for L2 it's proportional to the coefficient ($2\lambda w$)
   - This faster convergence to zero effectively performs feature selection, eliminating less informative features and creating a sparse model

2. Bias-variance trade-off with two examples:
   The bias-variance trade-off refers to the balance between a model's ability to fit the training data (low bias) and its ability to generalize to new data (low variance)

Example 1: Polynomial Regression
Consider fitting a curve to a dataset using polynomial regression of different degrees.
Low-degree polynomial (**high bias, low variance**):
A linear regression (degree 1 polynomial) will have high bias as it's too simple to capture the true underlying pattern in the data. It will have low variance as it's less sensitive to fluctuations in the training data.
High-degree polynomial (**low bias, high variance**):
A high-degree polynomial (e.g., degree 10) will have low bias as it can capture complex patterns in the data. It will have high variance as it's very sensitive to small changes in the training data and may overfit.

Example 2: Decision Trees vs Random Forests
Single Decision Tree (**low bias, high variance**):
A deep, unpruned decision tree can capture complex patterns in the data, resulting in low bias. It's prone to overfitting, leading to high variance as it's sensitive to small changes in the training data.
Random Forest (**moderate bias, lower variance**):
A random forest, which averages predictions from multiple decision trees, introduces some bias as individual trees are typically not grown to full depth. It significantly reduces variance through averaging and randomization techniques.

3. **Curse of dimensionality** :

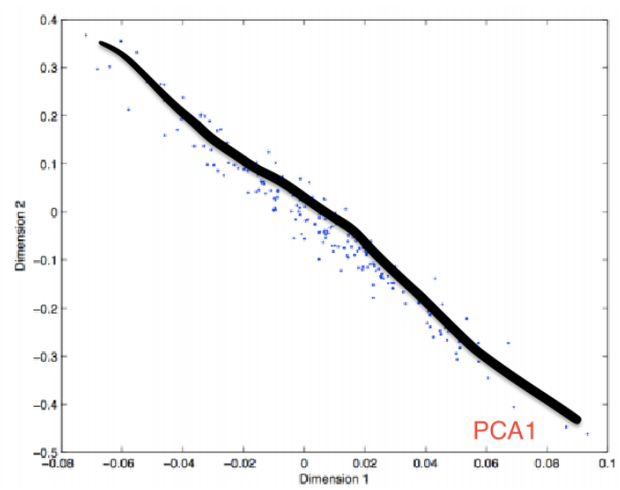
The curse of dimensionality refers to various phenomena that arise when analyzing data in high-dimensional spaces, which do not occur in lower-dimensional settings.

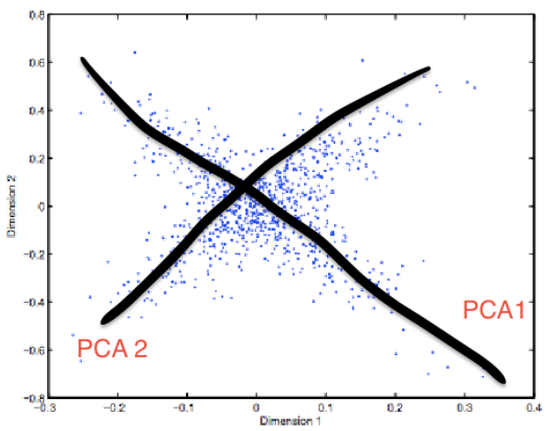
Example: Consider a dataset of 200 individuals with 2000 genetic features (genes)

- As the number of features increases, the volume of the space grows exponentially, making data sparse.
- The number of possible feature combinations grows factorially, increasing computational complexity.
- With many features, the risk of spurious correlations and overfitting increases.
- Distance measures become less meaningful in high dimensions, affecting clustering and classification algorithms.

This curse impacts machine learning by requiring exponentially more data to train models effectively, increasing the risk of overfitting, and making it harder to identify truly relevant features.

## Question 4 -



## Question 5 -

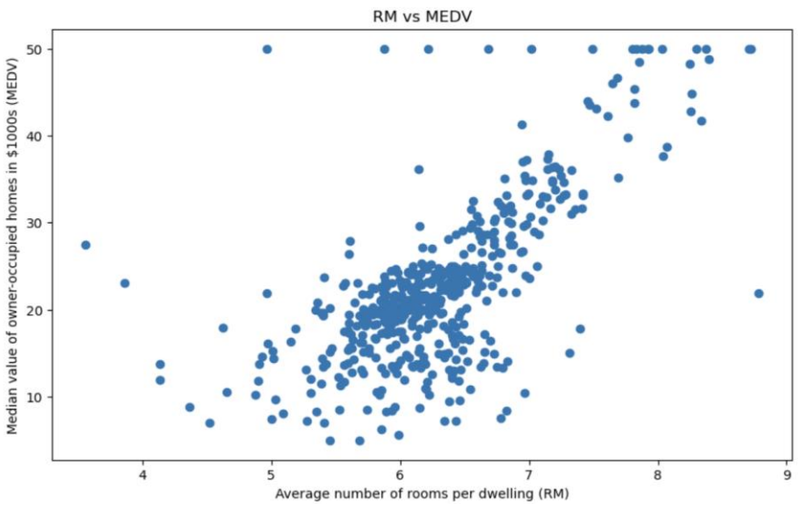
a. **Hierarchical clustering with single link** would be the most appropriate method for this dataset. Single link clustering excels at identifying elongated or chain-like cluster shapes, as shown in the two parallel, stretched-out clusters in the image, because it defines cluster similarity based on the closest pair of points between clusters. Other methods like K-means or complete link would struggle with these non-spherical clusters, as K-means assumes circular clusters, and complete link tends to break up elongated clusters into smaller, more compact segments.

b. **K-means or EM** (Expectation-Maximization) clustering would be most suitable for this pattern, with K-means being the most likely choice. The clusters appear to be spherical, well-separated, and of roughly equal size and density, which aligns perfectly with K-means' assumptions about cluster properties. Hierarchical clustering methods (single, complete, or average link) would be less appropriate here as they tend to create elongated or chain-like clusters, which doesn't match the clear spherical separation we see in this visualization.

c. The data points show a scattered distribution with significant overlap between the blue and yellow clusters, suggesting that **Hierarchical clustering with average linkage** is the most likely clustering method to be used. In average linkage, we define the distance between two clusters to be the average distance between data points in the first cluster and data points in the second cluster, which is well-defined in this case.

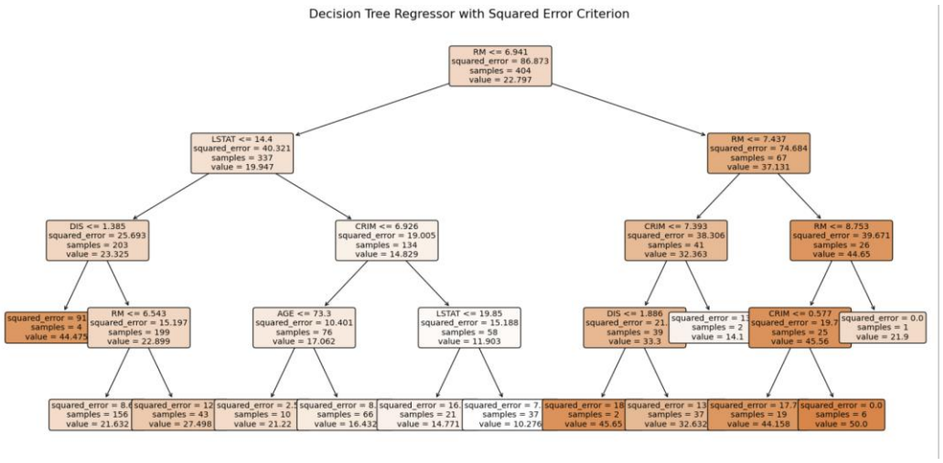
## Question 6 -

Decision Trees for regression work by recursively partitioning the feature space into smaller regions, creating a tree-like structure. At each node, the algorithm selects the feature and split point that minimizes the chosen error criterion. The process continues until a stopping criterion is met, such as reaching a maximum depth or a minimum number of samples in a leaf. The prediction for a new sample is then made by traversing the tree to a leaf node and returning the average target value of the training samples in that leaf.




In the dataset given :

- Most properties in the dataset have between 5 to 7 rooms, with prices clustering most densely in the $15,000 to $30,000 range. Properties with 6 or more rooms show a notably higher price potential, frequently reaching values above $35,000.
- Properties with 4-5 rooms typically fall in the lower price bracket, rarely exceeding $25,000. The mid-range segment, comprising houses with 6-7 rooms, shows greater price variation but generally values between $20,000 and $35,000. The premium segment, consisting of properties with 8 or more rooms, consistently achieves the highest valuations, often exceeding $40,000 and reaching up to $50,000 in some cases.
- The correlation coefficient of 0.695 between room count and house values indicates a strong positive relationship. This suggests that room count alone can explain approximately 48% of the variance in house prices. The average house value across all properties is $22,533, with a standard deviation that increases notably in properties with more rooms, indicating greater price volatility in larger homes.




## Question 7 -

### Part A -

[9]:
```python
import numpy as np

class TwoInputPerceptron:
    def __init__(self):
        self.weights = np.array([0.5, -0.5])  # Weights for A and NOT B
        self.bias = -0.5  # Bias term for the perceptron

    def activation(self, x):
        return 1 if x >= 0 else 0  # Step activation function

    def predict(self, A, B):
        weighted_sum = np.dot(self.weights, [A, 1 - B]) + self.bias  # A AND NOT B
        return self.activation(weighted_sum)
```

[11]:
```python
# Create an instance of the perceptron
perceptron = TwoInputPerceptron()

# Test the perceptron with all possible input combinations
print("A | B | Output")
print("-" * 15)
for A in [0, 1]:
    for B in [0, 1]:
        output = perceptron.predict(A, B)
        print(f"{A} | {B} | {output}")
```

```
A | B | Output
---------------
0 | 0 | 0
0 | 1 | 0
1 | 0 | 0
1 | 1 | 1
```

This implementation creates a two-input perceptron that performs the A ∧ ¬B (A AND NOT B) operation.

Here's a brief explanation of each cell:

- The first cell initializes the perceptron class with weights and bias. The weights [0.5, -0.5] correspond to A and NOT B, respectively, and the bias is set to -0.5 and defines the activation function, which is a simple step function returning 1 for non-negative inputs and 0 otherwise. It calculates the weighted sum of inputs (using 1 - B to represent NOT B) and applies the activation function to determine the output.
- The second cell contains the prediction function and demonstrates how to use the perceptron by testing it with all possible input combinations.

This perceptron implementation correctly models the A ∧ ¬B function, outputting 1 only when A is 1 and B is 0, and 0 for all other input combinations.

### Part B -

[13]:
```python
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers

# XOR input and output data
X = np.array([[0, 0], [0, 1], [1, 0], [1, 1]])
Y = np.array([[0], [1], [1], [0]])

# Create the model
model = keras.Sequential([
    layers.Dense(2, activation='sigmoid', input_shape=(2,)),
    layers.Dense(1, activation='sigmoid')
])
```

```
/Users/sid/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim
` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

This code does the following:

- Import the necessary modules from Keras.
- define our XOR input (X) and output (Y) data.
- create a Sequential model with two layers:
- The first layer has 2 neurons with sigmoid activation and takes 2 inputs.
- The second layer has 1 neuron with sigmoid activation for the output.

```
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
model.fit(X, Y, epochs=1000, verbose=0)

# Test the model
print("XOR Neural Network Output:")
print("--------------------------")
predictions = model.predict(X)
for i in range(len(X)):
    print(f"Input: {X[i]}, Output: {predictions[i][0]:.4f}, Output: {Y[i][0]}")
```

```
XOR Neural Network Output:
--------------------------
WARNING:tensorflow:5 out of the last 5 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at 0x17a2c6340> triggered tf.function retracing. Tracing is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/function for more details.
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 14ms/step
Input: [0 0], Output: 0.0833, Output: 0
Input: [0 1], Output: 0.8761, Output: 1
Input: [1 0], Output: 0.8687, Output: 1
Input: [1 1], Output: 0.1720, Output: 0
```

- compile the model using the Adam optimizer and binary crossentropy loss function.
- train the model for 1000 epochs on our XOR data.
- test the model by predicting outputs for all inputs and print the results.

The model has successfully learned the XOR function, producing outputs close to 0 for inputs [0,0] and [1,1], and outputs close to 1 for inputs [0,1] and [1,0].