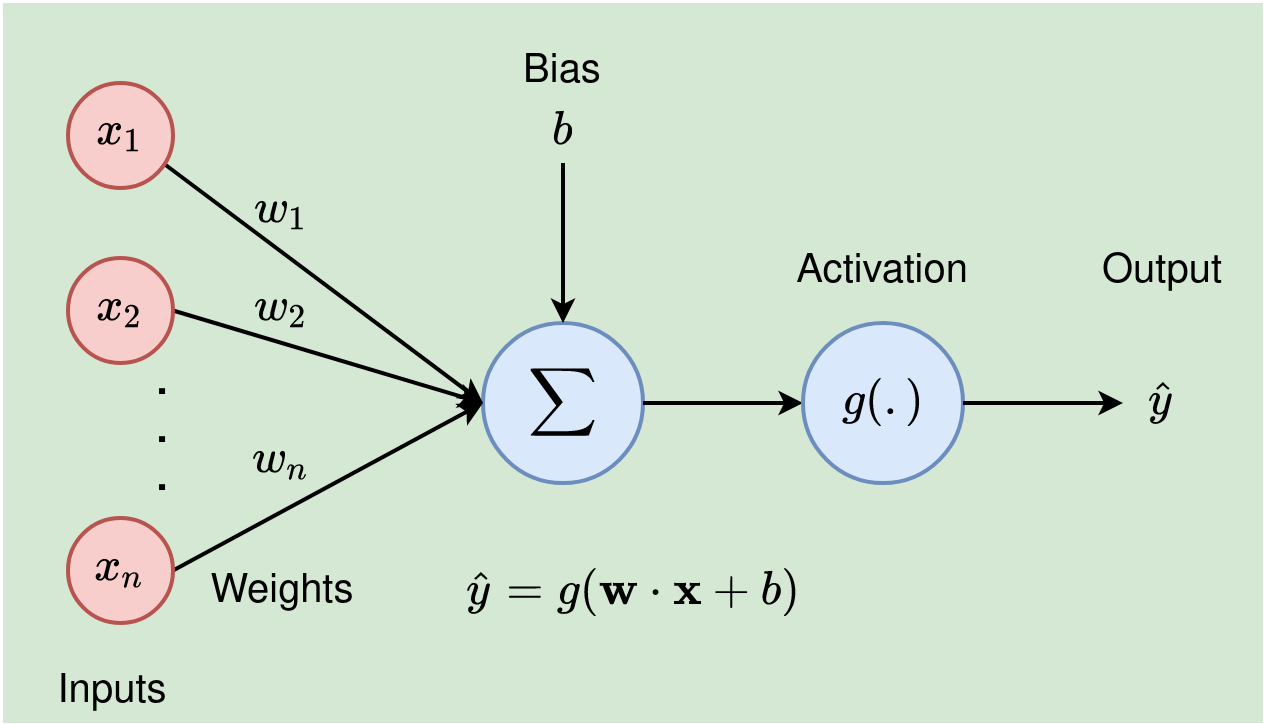
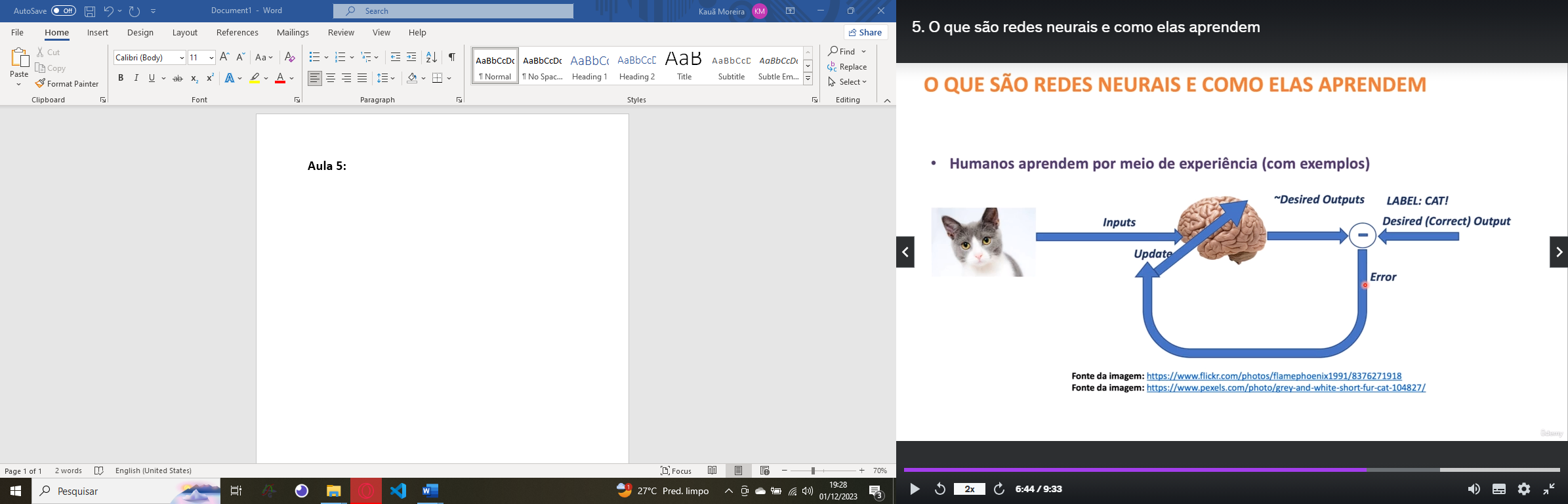
**Lesson 5:**

Basicly, the idea of a Artificial Neural Network is make the computer learn like a human using algorithms that works like a neuron.

In this example we have the ***Inputs***, ***Bias*** and a math function that works using the received values.





In the context of artificial intelligence and machine learning:

***X values (features):*** These are the input variables or features used to make predictions. They represent the data attributes or characteristics that the model uses to learn patterns and make predictions.

***Y values (target or dependent variable):*** This is the output variable that the model aims to predict or estimate based on the input features (X values). The model learns from the relationship between X values and Y values during the training process, and once trained, it makes predictions or estimations of Y based on new or unseen X values.

So, to summarize:

X values (features) are the predictors or input data.

Y values (target or dependent variable) are the values the model aims to predict or estimate based on the input features.

**Lesson 6 - 8:**

In artificial intelligence and machine learning, a “***model***” refers to the learned representation of patterns or relationships within data. Talking about TensorFlow or Keras, a model is an abstract entity that encapsulates the structure and learned parameters (weights and biases) from the training data.  
In the context of building neural networks and deep learning models, ***Keras*** is an open-source high-level neural networks API written in Python. It provides a user-friendly interface for creating, training, evaluating, and deploying deep learning models.

In the line of code model = tf.keras.Sequential(), a model is being instantiated using Keras. This particular model is a sequential model, which is a simple linear stack of layers where each layer has connections only to the next layer in the stack.

In the line of code model.compile(optimizer=tf.keras.optimizers.Adam(0.1), loss='mean\_squared\_error'). It prepares the model for the training process by specifying additional settings necessary for training, such as the optimizer and loss function.

1. optimizer:

An optimizer, in the context of training neural networks, is an algorithm or method used to adjust the attributes (weights and biases) of the neural network's parameters during the training process to minimize the loss function.

1. loss:

The loss parameter refers to the function used to quantify the difference between the predicted output of the model and the actual target output during the training process.

In the line of code model.add(tf.keras.layers.Dense(units=64, activation='relu', input\_shape=(input\_size,))) you add a layer in the model.

Besides the tf.keras.layer.Dense, there are a lot of kind of ***layers***, here are some examples:

1. Convolutional Layers (Conv2D, Conv1D):

tf.keras.layers.Conv2D: Convolutional layer for 2D spatial convolution (commonly used in image processing).

tf.keras.layers.Conv1D: Convolutional layer for 1D temporal convolution (used in sequence data like time series).

1. Pooling Layers (MaxPooling2D, AveragePooling2D):

tf.keras.layers.MaxPooling2D: Max pooling layer for 2D data.

tf.keras.layers.AveragePooling2D: Average pooling layer for 2D data.

1. Recurrent Layers (LSTM, GRU):

tf.keras.layers.LSTM: Long Short-Term Memory layer for sequence data.

tf.keras.layers.GRU: Gated Recurrent Unit layer for sequence data.

1. Dropout Layer (Dropout):

tf.keras.layers.Dropout: Dropout layer to prevent overfitting by randomly setting input units to 0 during training.

1. Flatten Layer (Flatten):

tf.keras.layers.Flatten: Flatten layer to flatten the input into a 1D array.

1. Embedding Layer (Embedding):

tf.keras.layers.Embedding: Embedding layer for mapping discrete values into continuous vectors.

1. Normalization Layers (BatchNormalization):

tf.keras.layers.BatchNormalization: Batch normalization layer to normalize the activations of the previous layer at each batch.

1. Activation Layers (Activation):

tf.keras.layers.Activation: Activation layer to apply an activation function.

When you use a layer, some parameter must be passed, like the below and others:

1. Units:

The number of neurons in the layer

1. Activation:

It is the mathematical function that will decide if the neuron must be activated or not. Basically, using a mathematical function, the neuron decides if the input information is important or not.

1. input\_shape:

Number of inputs (don’t counting with biases)

What is a ***Neural Network Activation Function***? An Activation Function decides whether a neuron should be activated or not. This means that it will decide whether the neuron's input to the network is important or not in the process of prediction using simpler mathematical operations. Basically, using a mathematical function, the neuron decides if the input information is important or not. The activation value is defined when we calculate the sum and multiplications related to the interactions between the weights and input values. To understand it better, watch the lesson 107 “Redes multicamadas - função soma e função de ativação”.

1. Sigmoid Function:

Range: (0, 1)

Used in the output layer for binary classification problems.

1. Hyperbolic Tangent (tanh):

Range: (-1, 1)

Similar to the sigmoid function but centered at 0.

1. Rectified Linear Unit (ReLU):

Range: [0, +∞)

Widely used due to its simplicity and effectiveness in training deep neural networks.

1. Leaky ReLU:

Range: (-∞, +∞)

Variant of ReLU to address the "dying ReLU" problem.

1. Parametric ReLU (PReLU):

Similar to Leaky ReLU but with a learnable parameter for variable instead of a fixed slope.

1. Exponential Linear Unit (ELU):

Range: (-α, +∞)

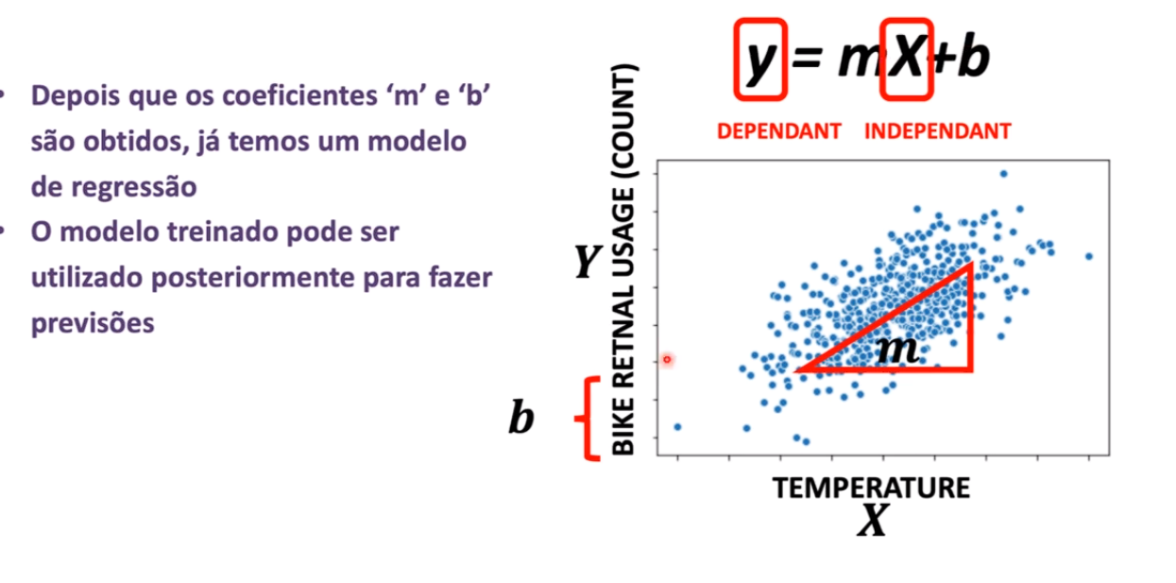
A variant of ReLU that addresses the vanishing gradient problem.

1. Softmax:

Used in the output layer for multi-class classification to obtain class probabilities

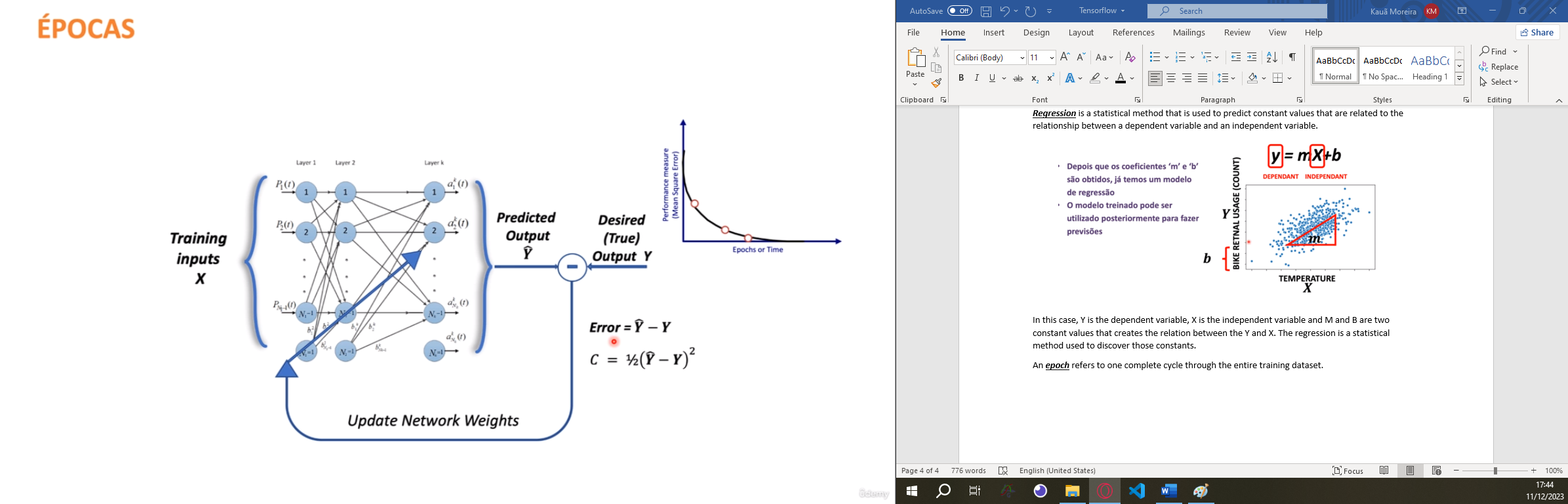
**Lesson 6 - 8:**

***Regression*** is a statistical method that is used to predict constant values that are related to the relationship between a dependent variable and an independent variable.



In this case, Y is the dependent variable, X is the independent variable and M and B are two constant values that creates the relation between the Y and X. The regression is a statistical method used to discover those constants.

An ***epoch*** refers to one complete cycle through the entire training dataset.



O ***gradiente*** é um conjunto de vetores que indicam a direção e a magnitude da mudança em uma função em um espaço multidimensional

|  |  |
| --- | --- |
|  | Dois exemplos de gradiente. Em cada caso o valor da função é indicado pela escala de cinzas. |

***Back propagation*** is an algorithm that calculate the gradient that will be used in the gradient descent.

***Gradient descent*** (Descida do gradiente) refers to the optimization algorithm used in neural networks and machine learning to minimize the loss function and update the model's parameters during training.

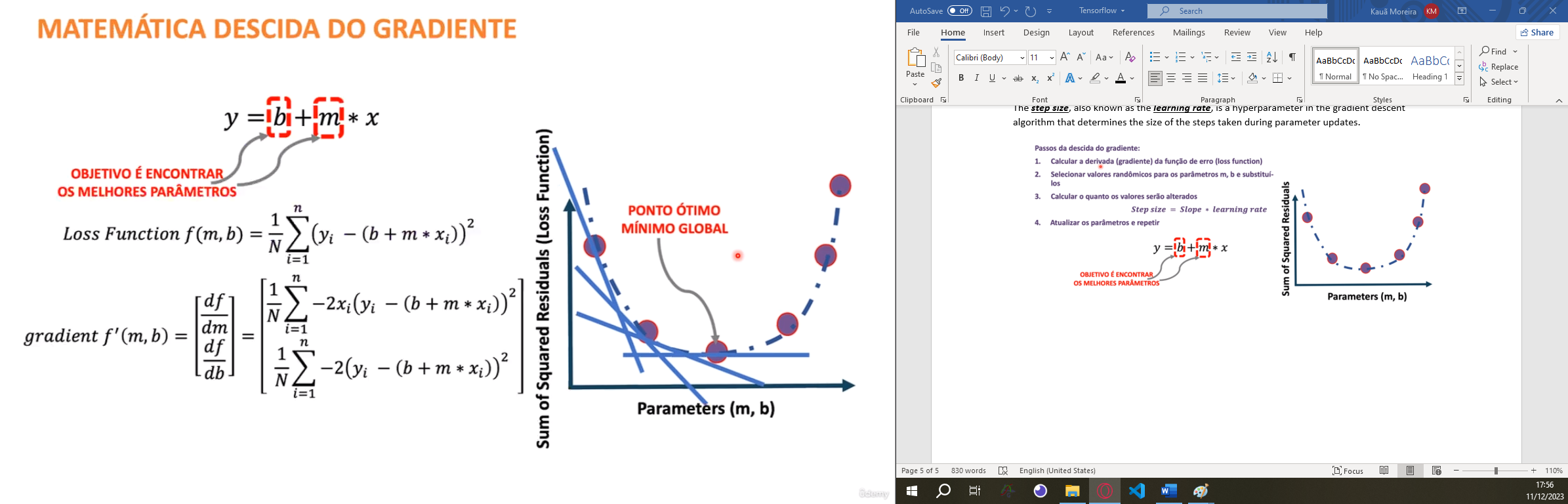
Backpropagation is a step in the training of Neural Networks that occurs at the end of processing each batch. Essentially, when the processing of a batch is completed, the neural network calculates an error and executes the backpropagation function. The backpropagation function operates by passing the error value layer by layer, deriving the error function in each layer, and determining the average rate of change of the error in each of them. In other words, by calculating the average rate of change of the error, we also calculate the gradient of the error, which is the derivative of the error function itself. Using the gradient descent technique, it is possible to move in the opposite direction to that indicated by the error, meaning descending the gradient in an attempt to reduce the error. This involves adjusting the neuron weights and biases. This descent along the gradient is facilitated by an optimization function, essentially an algorithm that defines how the gradient will be utilized to decrease the error. It's important to remember that the gradient's dimensions are determined by the number of free variables. For instance, if there are 5 parameters being passed as input to the neural network, it means the gradient will have 5 dimensions.

The ***step size***, also known as the ***learning rate***, is a hyperparameter in the gradient descent algorithm that determines the size of the steps taken during parameter updates.

The 'step size' (learning rate) is indeed the size of the correction step that a neural network takes in the opposite direction of the gradient calculated by backpropagation, aiming to reduce errors by adjusting the weights and biases. The larger the 'step size,' the bigger the correction step towards the ideal, potentially surpassing it; the smaller the 'step size,' the smaller the step, possibly making the training process slow.

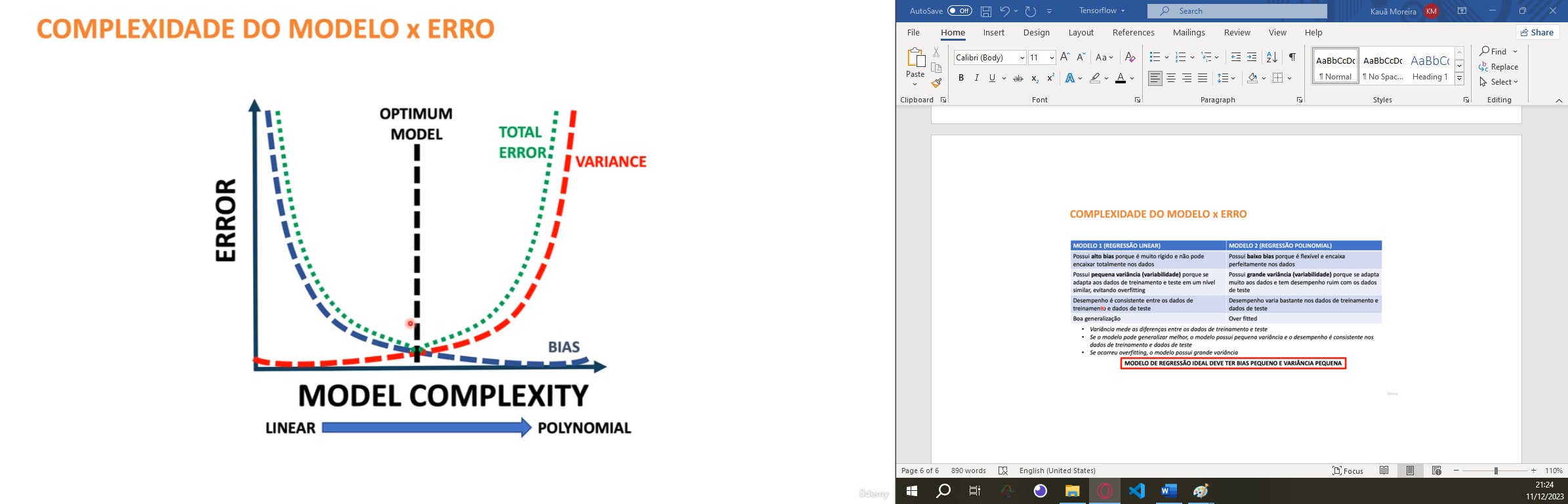
To better understand, consider an analogy: envision a ping pong ball at the top of an immense funnel. The goal is to place the ball at the center of the funnel, achieved by pushing it from the top. If the push is too strong, the ball overshoots the center quickly. If it's gentle, the ball might eventually reach the center, but it will take longer.

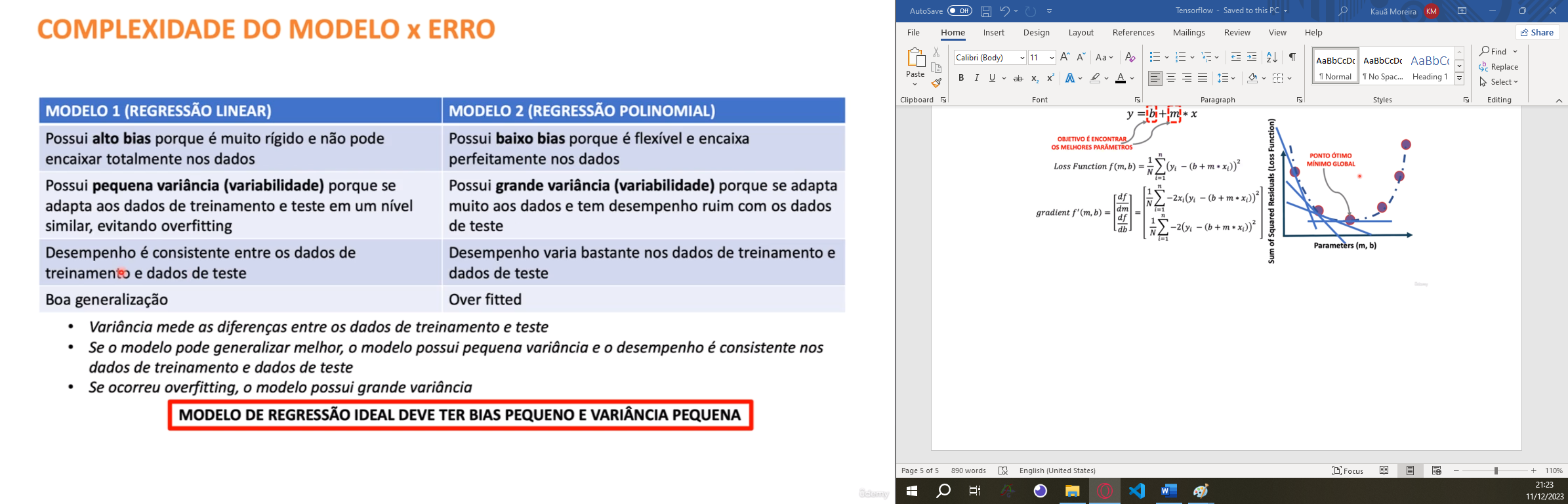
In this analogy, the ball's position represents the error, and the closer it is to the center of the funnel, the smaller the error. The intention is to minimize the error by bringing the ball closer to the center, and the speed at which we push it represents the 'step size.'"



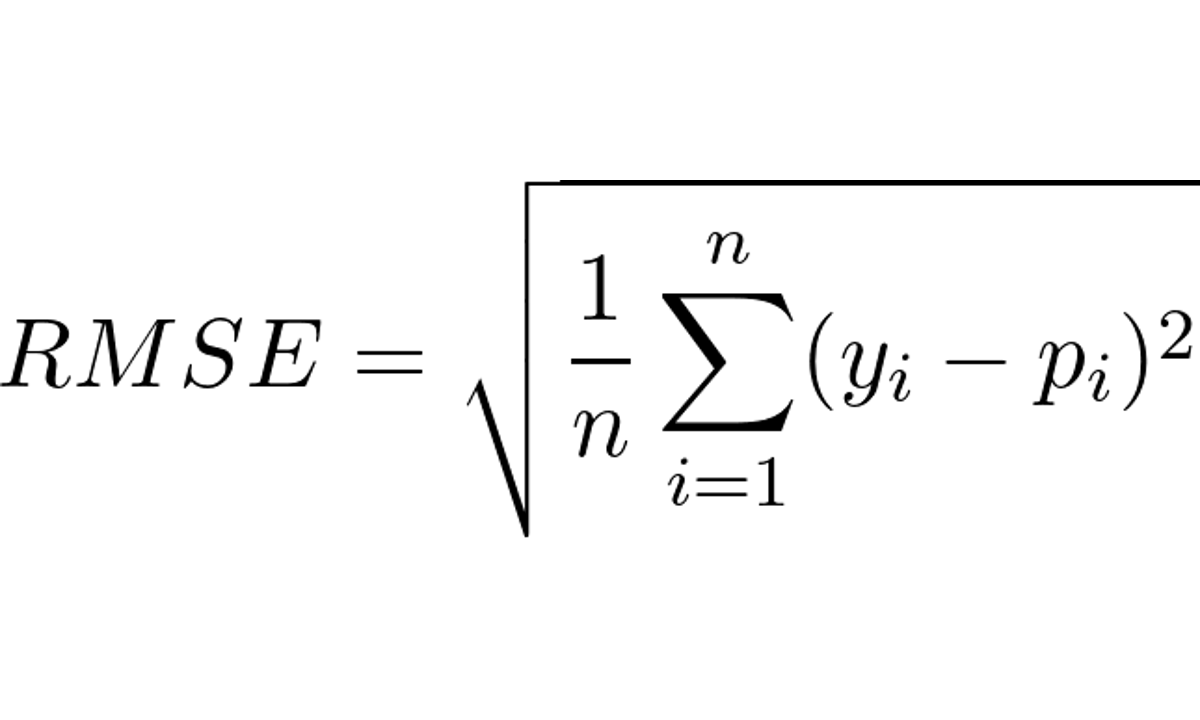
***Variance***, in statistics, measures the extent to which data points in a dataset differ from the mean (average) value. It quantifies the spread or dispersion of a set of values.

***Bias*** (viés) is an additional learnable parameter that is added to each node (neuron) in a layer, along with the weights.

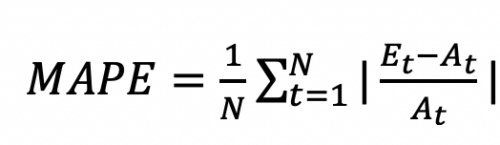




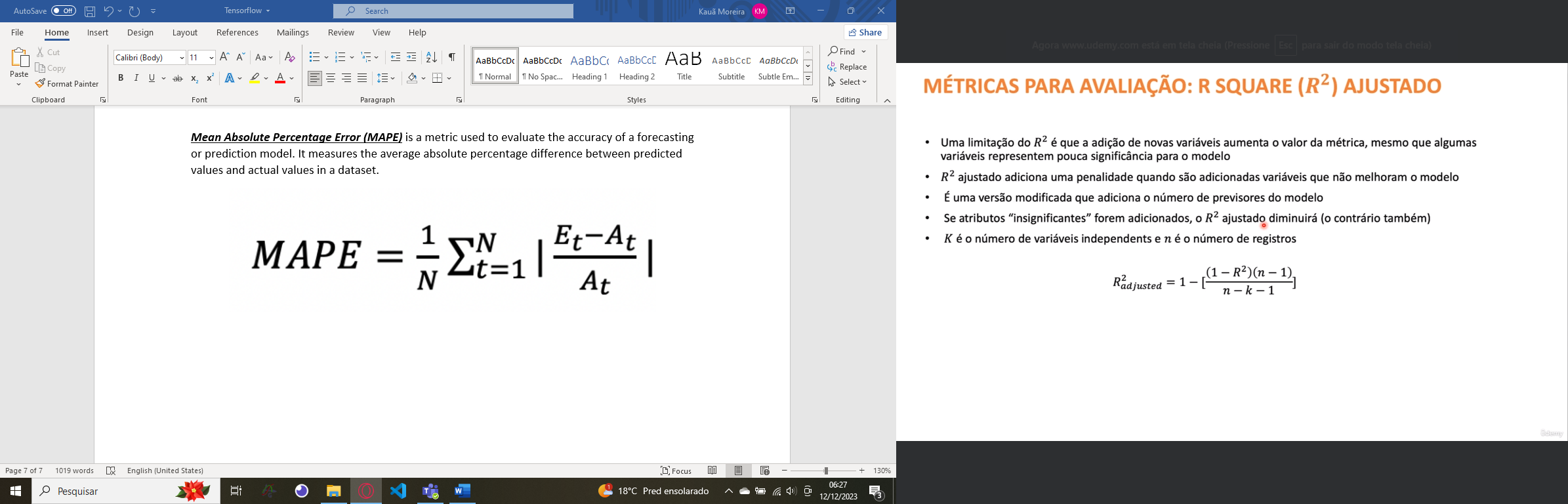
***MSE*** stands for ***Mean Squared Error*** (Erro médio quadrático) and it's a common loss function used in neural networks and regression problems. It measures the average of the squared differences between the predicted values and the actual (true) values in a dataset.



***Mean Absolute Percentage Error (MAPE)*** is a metric used to evaluate the accuracy of a forecasting or prediction model. It measures the average absolute percentage difference between predicted values and actual values in a dataset.



***R-squared (coefficient of determination)*** is a statistical measure used to assess how well a regression model fits the observed data. It provides insights into the proportion of variance in the dependent variable (target) that is explained by the independent variables (features) in the model.



* R-squared = 1: Indicates that the regression model perfectly predicts the dependent variable using the independent variables. All variations in the dependent variable are explained by the independent variables.
* R-squared = 0: Indicates that the regression model does not explain any of the variability of the dependent variable around its mean. It might not be capturing any relationship between the independent and dependent variables.