**What is Silhouette Score?**

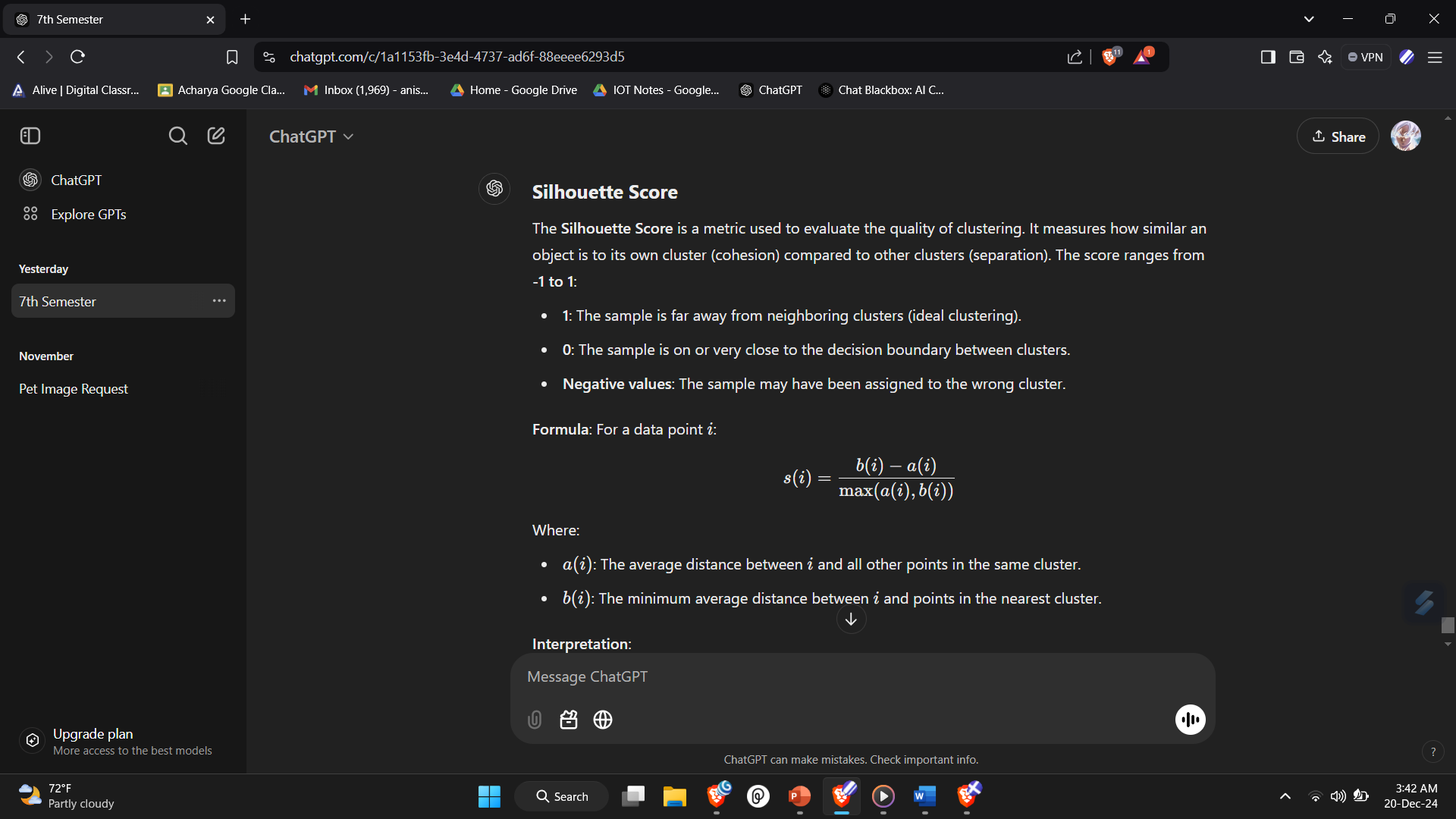
The **Silhouette Score** is a metric used to evaluate the quality of clustering. It measures how similar an object is to its own cluster (cohesion) compared to other clusters (separation). The score ranges from **-1 to 1**:

* **1**: The sample is far away from neighboring clusters (ideal clustering).
* **0**: The sample is on or very close to the decision boundary between clusters.
* **Negative values**: The sample may have been assigned to the wrong cluster.

**Interpretation**:

* A high Silhouette Score indicates well-separated and cohesive clusters.
* A score close to 0 indicates overlapping clusters.
* A negative score suggests incorrect clustering.

A score of **0.3460** suggests that the clustering is **moderately effective**, but there is room for improvement in cluster separation.



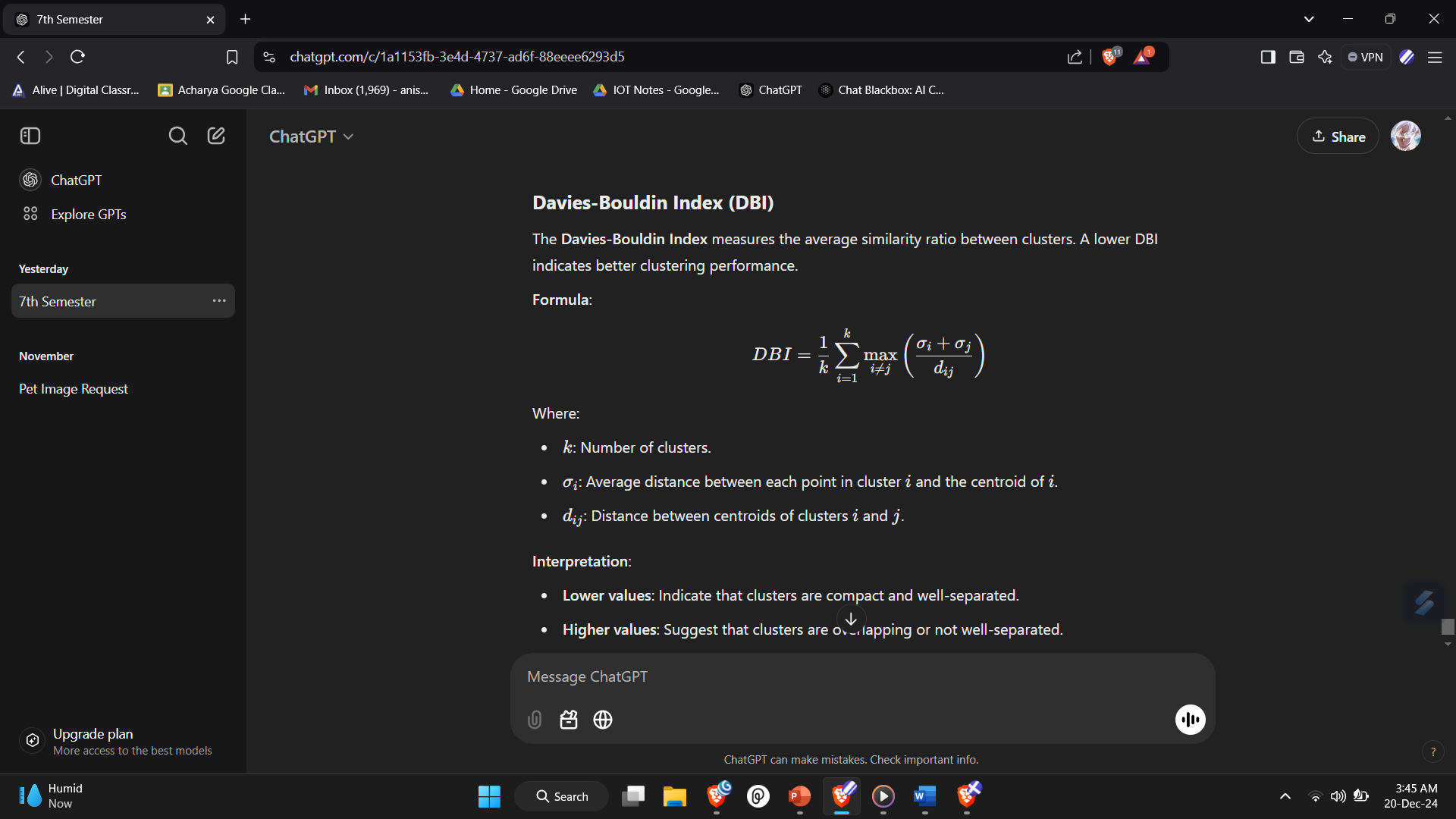
**What is Davies-Bouldin Index?**

The **Davies-Bouldin Index** measures the average similarity ratio between clusters. A lower DBI indicates better clustering performance.

**Interpretation**:

* **Lower values**: Indicate that clusters are compact and well-separated.
* **Higher values**: Suggest that clusters are overlapping or not well-separated.

A DBI of **0.9572** indicates **excellent clustering**, where the clusters are compact and well-separated.



**Using Silhouette Score and DBI Together**

* **Silhouette Score** helps assess how well individual points fit within their clusters.
* **Davies-Bouldin Index** gives an overall view of the cluster compactness and separation.
* **Complementary Metrics**: While Silhouette Score is intuitive and visualizable, DBI provides additional insights into cluster structures.

**Explanation of Code (in Bullets)**

**Markdown Cells**

* **Introduction to LSTM Model**: Provides context for using LSTM for feature extraction.
* **Feature Extraction**: Discusses using LSTM and Fourier Transform for extracting features.
* **K-Means Clustering**: Mentions clustering the extracted features into 10 groups.
* **Accuracy Evaluation**: Introduces metrics like Silhouette Score and Davies-Bouldin Score for evaluating clustering.
* **Clustering Results**: Interprets clustering performance based on Silhouette Score.
* **Visualization**: Details the generation of graphs from CSV data.

**Code Cells**

1. **Imports and Preprocessing**:
   * Loads necessary libraries like Pandas, Numpy, and TensorFlow/Keras for building LSTM models.
   * Defines a preprocessing function for CSV files.
2. **Feature Extraction**:
   * Implements LSTM feature prediction.
   * Combines LSTM features with Fourier-transformed features for clustering.
3. **K-Means Clustering**:
   * Performs clustering on the combined features.
   * Creates directories to organize data based on cluster labels.
4. **Clustering Accuracy**:
   * Evaluates clustering using Silhouette Score and Davies-Bouldin Score.
5. **Data Visualization**:
   * Converts CSV data into visual graphs using Matplotlib and Seaborn.
   * Saves graphs in predefined folders.

**Explanation of Code (in Details)**

**1. Markdown: LSTM Model Creation for Feature Extraction**

* This section introduces the primary goal: using an LSTM model to extract features from data, possibly time-series data.

**2. Code: Import Libraries and Define Preprocessing Function**

import pandas as pd

import numpy as np

import os

from sklearn.cluster import KMeans

from sklearn.preprocessing import MinMaxScaler

from keras.\_tf\_keras.keras.models import Sequential

from keras.\_tf\_keras.keras.layers import Dense, LSTM, Dropout

* **Purpose**:
  + Imports libraries for data manipulation (pandas, numpy).
  + Uses scikit-learn for clustering (KMeans) and data scaling (MinMaxScaler).
  + Leverages Keras for defining and training the LSTM model.
* **Preprocessing Function**:

def preprocess\_csv(file\_path):

try:

df = pd.read\_csv(file\_path)

# Scaling or other processing could be done here

return df

except Exception as e:

print(f"Error processing {file\_path}: {e}")

return None

* + Reads a CSV file and handles any errors during loading.

**3. Markdown: Feature Extraction Using LSTM and Fourier Transform**

* Highlights that two types of features are extracted:
  + **LSTM Features**: Leveraging a trained LSTM model to predict features from input data.
  + **Fourier Transform Features**: Applying Fourier Transform to analyze frequency-domain information.

**4. Code: Feature Extraction**

lstm\_features = model.predict(X)

lstm\_features = lstm\_features.reshape(lstm\_features.shape[0], -1)

fourier\_features = [fourier\_transform(data.flatten()) for data in processed\_data]

combined\_features = []

for lstm\_f, fourier\_f in zip(lstm\_features, fourier\_features):

combined\_features.append(np.concatenate([lstm\_f, fourier\_f]))

* **LSTM Features**:
  + Uses a trained LSTM model to predict and reshape features from input data X.
* **Fourier Transform Features**:
  + A placeholder function (fourier\_transform) applies Fourier analysis to the data.
* **Combining Features**:
  + Concatenates LSTM and Fourier features into a single feature set for each data point.

**5. Markdown: K-Means Clustering**

* Explains that the combined features are clustered into **10 clusters** using the K-Means algorithm.

**6. Code: K-Means Clustering**

kmeans = KMeans(n\_clusters=10, random\_state=0)

kmeans.fit(combined\_features)

labels = kmeans.labels\_

os.makedirs('Signature Fault Clusters Version 16 Final/VRM', exist\_ok=True)

for i in range(10):

os.makedirs(os.path.join('Signature Fault Clusters Version 16 Final/VRM', f'Cluster {i}'), exist\_ok=True)

* **Purpose**:
  + Clusters the combined features into 10 groups using K-Means.
  + Saves the results into directories corresponding to each cluster.

**7. Markdown: Accuracy Evaluation**

* Introduces metrics to evaluate the quality of clustering:
  + **Silhouette Score**: Measures the cohesion of clusters (ranges from -1 to 1).
  + **Davies-Bouldin Score**: Evaluates the separation between clusters (lower is better).

**8. Code: Accuracy Metrics**

from sklearn.metrics import silhouette\_score, davies\_bouldin\_score

silhouette\_avg = silhouette\_score(combined\_features, labels)

print(f"Silhouette Score: {silhouette\_avg}")

db\_score = davies\_bouldin\_score(combined\_features, labels)

print(f"Davies-Bouldin Score: {db\_score}")

* Computes the scores and prints the results.
* Provides an understanding of clustering performance:
  + High Silhouette Score indicates well-separated and compact clusters.
  + Low Davies-Bouldin Score signifies better cluster quality.

**9. Markdown: CSV Files to Graphs**

* Explains the visualization of data points in clusters through graphs.

**10. Code: Data Visualization**

import matplotlib.pyplot as plt

import seaborn as sns

csv\_folder = 'Signature Fault Clusters Version 16 Final/VRM/VRM Cluster 9'

output\_folder = 'Signature Fault Clusters Version 16 Final/VRM Graph/VRM Cluster 9'

os.makedirs(output\_folder, exist\_ok=True)

for file in os.listdir(csv\_folder):

if file.endswith('.csv'):

data = pd.read\_csv(os.path.join(csv\_folder, file))

plt.figure(figsize=(10, 6))

sns.lineplot(data=data)

plt.savefig(os.path.join(output\_folder, f"{file}.png"))

plt.close()

* **Purpose**:
  + Reads CSV files from a directory corresponding to a cluster.
  + Plots line graphs using Matplotlib and Seaborn.
  + Saves graphs in a specified folder.

**LSTM Model Creation: Explanation with Code**

The LSTM model is built using Keras to extract features from time-series data. Below is a detailed explanation in bullet points:

**1. Import Required Libraries**

from keras.\_tf\_keras.keras.models import Sequential

from keras.\_tf\_keras.keras.layers import Dense, LSTM, Dropout

* **Sequential**: Allows creating a stack of layers linearly.
* **LSTM**: Implements the Long Short-Term Memory layer, ideal for sequence modeling and time-series data.
* **Dropout**: Regularization technique to reduce overfitting.
* **Dense**: Fully connected layer used for the output.

**2. Initialize the Model**

model = Sequential()

* Creates an empty model to which layers can be added sequentially.

**3. Add LSTM Layers**

model.add(LSTM(units=50, return\_sequences=True, input\_shape=(time\_steps, features)))

model.add(Dropout(0.2))

* **LSTM Layer**:
  + units=50: Specifies the number of LSTM units (neurons in the layer).
  + return\_sequences=True: Ensures that the output at each time step is returned (required when stacking LSTMs).
  + input\_shape=(time\_steps, features): Defines the shape of the input data.
    - time\_steps: Number of time steps in the sequence.
    - features: Number of features at each time step.
* **Dropout**:
  + Adds regularization to **prevent overfitting** by randomly setting 20% of the weights to zero during training.

**4. Add Additional LSTM Layers**

model.add(LSTM(units=50, return\_sequences=True))

model.add(Dropout(0.2))

model.add(LSTM(units=50, return\_sequences=False))

model.add(Dropout(0.2))

* Stacks additional LSTM layers for deeper learning.
  + **Second LSTM**: Processes outputs from the first LSTM layer.
    - return\_sequences=True ensures compatibility with subsequent layers.
  + **Third LSTM**: Final LSTM layer in the stack.
    - return\_sequences=False because no further recurrent layers follow.

**5. Add a Dense Output Layer**

model.add(Dense(units=1))

* Fully connected layer:
  + units=1: Output layer with one neuron to predict a single continuous value.

**6. Compile the Model**

model.compile(optimizer='adam', loss='mean\_squared\_error')

* **Optimizer**:
  + adam: Adaptive Moment Estimation, a widely used optimizer for training deep learning models.
* **Loss Function**:
  + mean\_squared\_error: Common loss function for regression tasks.

**7. Model Training**

model.fit(X\_train, y\_train, epochs=50, batch\_size=32)

* **Training Parameters**:
  + X\_train: Input features for training.
  + y\_train: Target values for training.
  + epochs=50: Number of iterations over the entire dataset.
  + batch\_size=32: Number of samples processed before updating the model weights.