

राष्ट्रीय प्रौद्योगिकी संस्थान पुदुच्चेरी

NATIONAL INSTITUTE OF TECHNOLOGY PUDUCHERRY

KARAIKAL – 609 609

Final Year project - First Review Meeting

AI Based Design for Frequency Selective Surface (FSS)

Presented by

**KAPA ADHEESHWAR REDDY**

Reg No: EC21B1023

**SADHU POORNA SAI**

Reg No:EC21B1043

**SANAPALA CHANDRA SEKHAR**

Reg No:EC21B1044

B.TECH Department of ECE,

NIT Puducherry, Karaikal,

India

Project Guide

**Dr. Boopathi Rani . R**

Assistant Professor,

Depatment of ECE,

NIT Puducherry, Karaikal,

India

**Problem Statement:**

The current challenge lies in designing Frequency Selective Surfaces (FSS) to meet specific performance criteria through traditional methods, which require significant computational resources and expertise. Additionally, extracting control parameters for reconfigurable bandpass filters and reducing dataset requirements in AI-driven design processes present further difficulties. Therefore, this project seeks to develop AI models that address these challenges by enhancing design efficiency, accuracy, and data management.

**Objectives:**

* Develop AI Models for FSS Design: To leverage AI, particularly generative models, for the efficient and accurate design of Frequency Selective Surfaces with customized performance characteristics.
* Enhance Prediction Accuracy: To apply optimization techniques to improve the accuracy of FSS design parameters.
* Optimize Data Efficiency: To implement data shrinking techniques that reduce the need for computational time without sacrificing model performance in AI-driven designs.
* Integrate Hybrid Modeling Approaches: To combine traditional polynomial-transfer functions with neural networks to enhance parametric modeling for complex microwave spatial filters.

**Introduction:**

In modern wireless communication systems, Frequency Selective Surfaces (FSS) play a crucial role in filtering, controlling, and modifying electromagnetic waves. Traditional FSS design methodologies often involve iterative processes that are computationally expensive and time-consuming. However, the recent integration of artificial intelligence (AI) techniques, particularly in the form of deep learning and optimization algorithms, offers new avenues to simplify and enhance the design of FSS with desired electromagnetic properties. This project focuses on leveraging AI models for the inverse design of FSS, with a focus on improving accuracy, data efficiency, and overall performance in real-world applications, such as antenna design and electromagnetic shielding.

Methodology:

The project will follow a structured approach to tackle the design challenges associated with FSS using AI:

* **Generative Models for Inverse Design:** A generative model will be developed and trained on existing FSS datasets to automatically generate designs with desired electromagnetic characteristics. The model will be optimized to produce high-quality results while reducing computational time.
* **Dataset Reduction Techniques:** Data reduction strategies, such as principal component analysis (PCA) or clustering, will be employed to minimize the dataset size needed for training while preserving essential data features, thereby reducing computational load without sacrificing performance.
* **Optimization Techniques:** This technique will be applied to fine-tune model parameters, leading to more accurate FSS designs and bandpass filter configurations. Optimization techniques like Bayesian help by intelligently selecting the most promising parameters during training.
* **Hybrid Modeling:** A combination of traditional polynomial transfer functions and deep neural networks will be used to model complex microwave filters. This hybrid approach will balance the strengths of both techniques to produce more accurate and efficient models.

**Model 1: AI Model for Prediction of S21 and S11 Values**

**Code 1: Initial Linear Regression with Separate S11 and S21 Models**

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Specify the file path

# Use the absolute path, e.g., 'C:/path/to/your/data.xlsx'

# Or use the relative path if it's in the same directory, e.g., './data.xlsx'

file\_path = './dataset.xls'  # Change this to your actual file path

# Load the dataset from an Excel file, explicitly specifying the engine

df = pd.read\_csv('./dataset.xls')# Use the 'openpyxl' engine for .xlsx files or 'xlrd' for older .xls files.

# Display the first few rows of the dataset to understand its structure

print("Dataset preview:")

print(df)

# Check if the necessary columns exist

required\_columns = ['Freq [GHz]', 'dB(S(1,1)) []', 'dB(S(2,1)) []']

for column in required\_columns:

    if column not in df.columns:

        raise ValueError(f"Missing required column: {column}")

# Features and Targets

X = df[['Freq [GHz]']]  # Use the correct column name

y\_s11 = df['dB(S(1,1)) []']  # Use the correct column name

y\_s21 = df['dB(S(2,1)) []']  # Use the correct column name

# Split data into training and testing sets

X\_train, X\_test, y\_s11\_train, y\_s11\_test = train\_test\_split(X, y\_s11, test\_size=0.2, random\_state=42)

X\_train, X\_test, y\_s21\_train, y\_s21\_test = train\_test\_split(X, y\_s21, test\_size=0.2, random\_state=42)

# Initialize the Linear Regression models

model\_s11 = LinearRegression()

model\_s21 = LinearRegression()

# Train the models

model\_s11.fit(X\_train, y\_s11\_train)

model\_s21.fit(X\_train, y\_s21\_train)

# Make predictions

y\_s11\_pred = model\_s11.predict(X\_test)

y\_s21\_pred = model\_s21.predict(X\_test)

# Evaluate the models

print("\nS11 Prediction")

print("Mean Squared Error:", mean\_squared\_error(y\_s11\_test, y\_s11\_pred))

print("R^2 Score:", r2\_score(y\_s11\_test, y\_s11\_pred))

print("\nS21 Prediction")

print("Mean Squared Error:", mean\_squared\_error(y\_s21\_test, y\_s21\_pred))

print("R^2 Score:", r2\_score(y\_s21\_test, y\_s21\_pred))

# Plot the results

plt.figure(figsize=(12, 6))

# S11 Plot

plt.subplot(1, 2, 1)

plt.scatter(X\_test, y\_s11\_test, color='blue', label='Actual S11')

plt.plot(X\_test, y\_s11\_pred, color='red', linewidth=2, label='Predicted S11')

plt.title('Frequency vs S11')

plt.xlabel('Frequency (Hz)')

plt.ylabel('S11')

plt.legend()

# S21 Plot

plt.subplot(1, 2, 2)

plt.scatter(X\_test, y\_s21\_test, color='green', label='Actual S21')

plt.plot(X\_test, y\_s21\_pred, color='orange', linewidth=2, label='Predicted S21')

plt.title('Frequency vs S21')

plt.xlabel('Frequency (Hz)')

plt.ylabel('S21')

plt.legend()

plt.show()

plt.tight\_layout

**Purpose**: Trains separate linear regression models for S11​ and S21​ parameters based on the frequency, using simple linear regression.

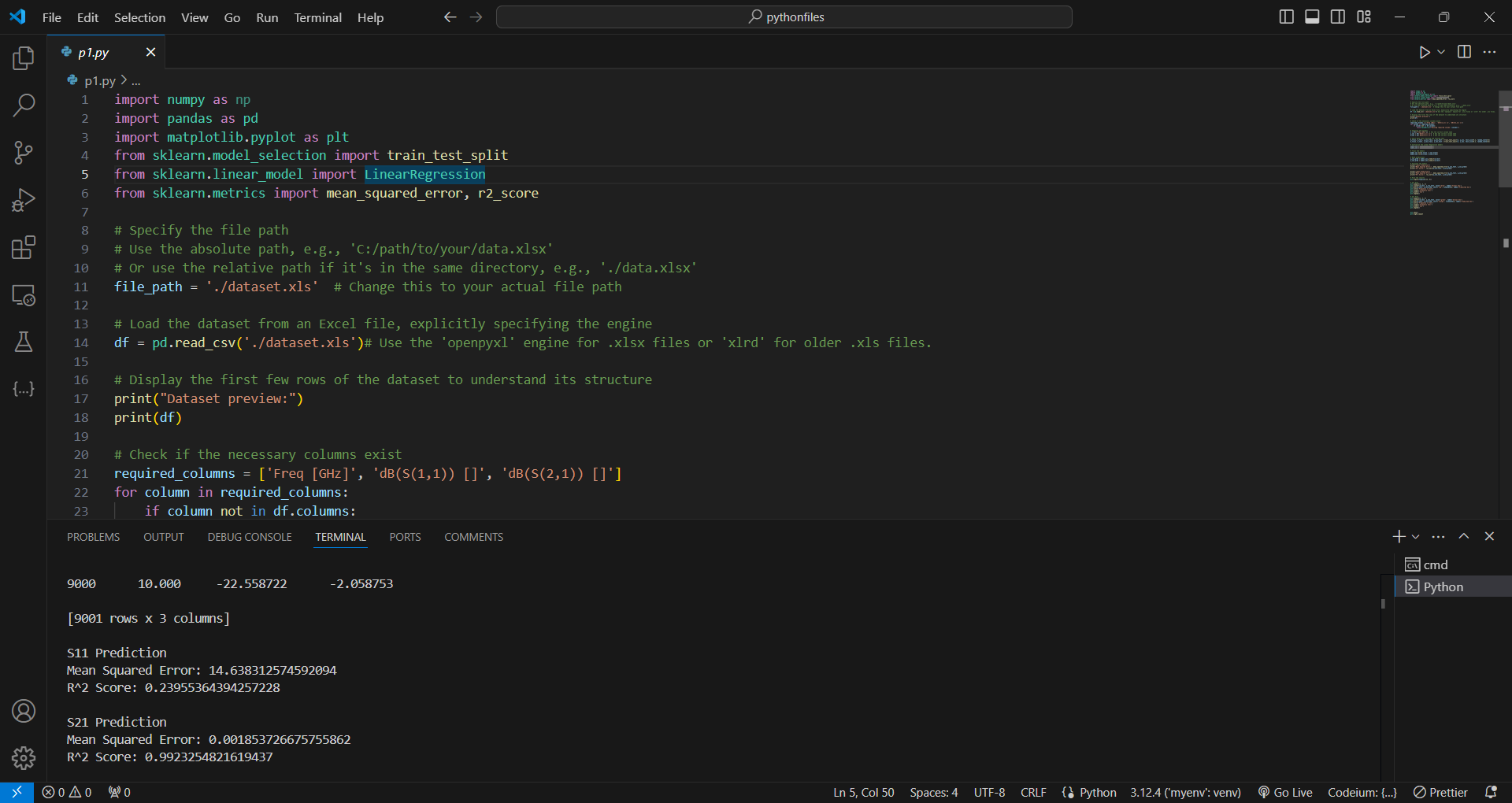
**Data Handling**:

* The dataset is processed to handle S11​ and S21​ values separately.
* The dataset is split into training and testing sets independently for S11​ and S21​ data.

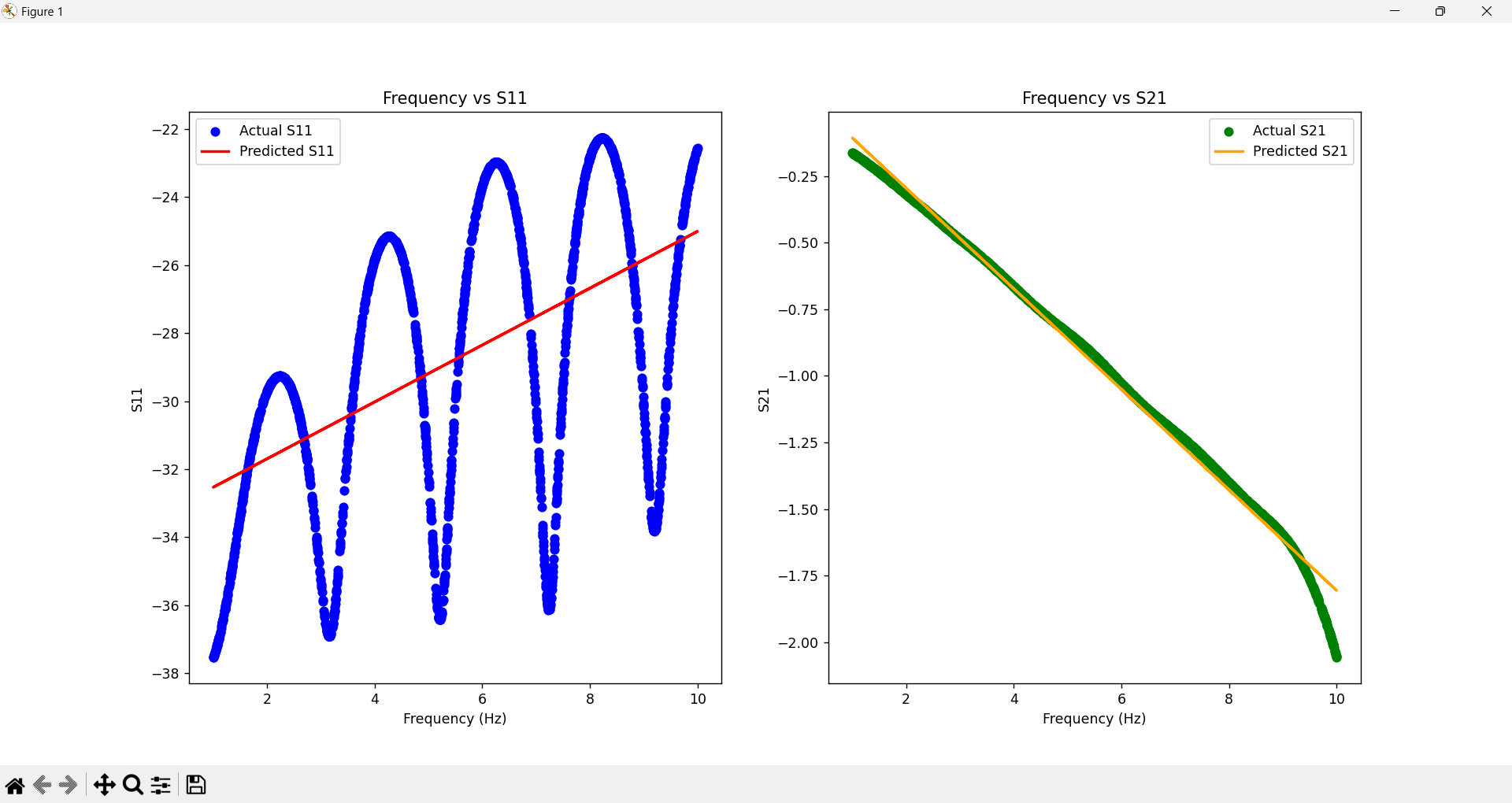
**Modeling**:

* Two separate Linear Regression models are trained: one for S11​ and one for S21​.
* The performance of the models is evaluated using Mean Squared Error (MSE) and R² scores.

**Output**:



**Visualization:**

****

**Observations**:

* This approach is straightforward and easy to implement.
* However, it does not consider potential interactions between S11 ​, S21​, and frequency. Each model is trained independently without leveraging possible correlations or joint effects between the features.

**Code 2: Combined Model Using Both S11 and S21**

**Code:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import Ridge

from sklearn.metrics import mean\_squared\_error, r2\_score

# Specify the file path

file\_path = './dataset.xls'  # Change this to your actual file path

# Load the dataset from an Excel file

df = pd.read\_csv('./dataset.xls')  # Use pd.read\_excel for .xls files

# Display the first few rows of the dataset to understand its structure

print("Dataset preview:")

print(df.head())

# Check if the necessary columns exist

required\_columns = ['Freq [GHz]', 'dB(S(1,1)) []', 'dB(S(2,1)) []']

for column in required\_columns:

    if column not in df.columns:

        raise ValueError(f"Missing required column: {column}")

# Features and Targets

X = df[['Freq [GHz]']]  # Use the correct column name

y\_s11 = df['dB(S(1,1)) []']  # Use the correct column name

y\_s21 = df['dB(S(2,1)) []']  # Use the correct column name

# Split data into training and testing sets

X\_train, X\_test, y\_s11\_train, y\_s11\_test = train\_test\_split(X, y\_s11, test\_size=0.2, random\_state=42)

X\_train, X\_test, y\_s21\_train, y\_s21\_test = train\_test\_split(X, y\_s21, test\_size=0.2, random\_state=42)

# Initialize the Ridge Regression models

model\_s11 = Ridge(alpha=1.0)  # You can adjust the alpha parameter for regularization strength

model\_s21 = Ridge(alpha=1.0)  # You can adjust the alpha parameter for regularization strength

# Train the models

model\_s11.fit(X\_train, y\_s11\_train)

model\_s21.fit(X\_train, y\_s21\_train)

# Make predictions

y\_s11\_pred = model\_s11.predict(X\_test)

y\_s21\_pred = model\_s21.predict(X\_test)

# Evaluate the models

print("\nS11 Prediction")

print("Mean Squared Error:", mean\_squared\_error(y\_s11\_test, y\_s11\_pred))

print("R^2 Score:", r2\_score(y\_s11\_test, y\_s11\_pred))

print("\nS21 Prediction")

print("Mean Squared Error:", mean\_squared\_error(y\_s21\_test, y\_s21\_pred))

print("R^2 Score:", r2\_score(y\_s21\_test, y\_s21\_pred))

# Plot the results

plt.figure(figsize=(12, 6))

# S11 Plot

plt.subplot(1, 2, 1)

plt.scatter(X\_test, y\_s11\_test, color='blue', label='Actual S11')

plt.plot(X\_test, y\_s11\_pred, color='red', linewidth=2, label='Predicted S11')

plt.title('Frequency vs S11')

plt.xlabel('Frequency (GHz)')

plt.ylabel('S11')

plt.legend()

# S21 Plot

plt.subplot(1, 2, 2)

plt.scatter(X\_test, y\_s21\_test, color='green', label='Actual S21')

plt.plot(X\_test, y\_s21\_pred, color='orange', linewidth=2, label='Predicted S21')

plt.title('Frequency vs S21')

plt.xlabel('Frequency (GHz)')

plt.ylabel('S21')

plt.legend()

plt.tight\_layout()  # Adjust layout

plt.show()

**Purpose**: Combines S11​, S21 ​, and frequency as features to predict S11​ and S21​, using Ridge Regression to potentially improve predictive performance and handle overfitting.

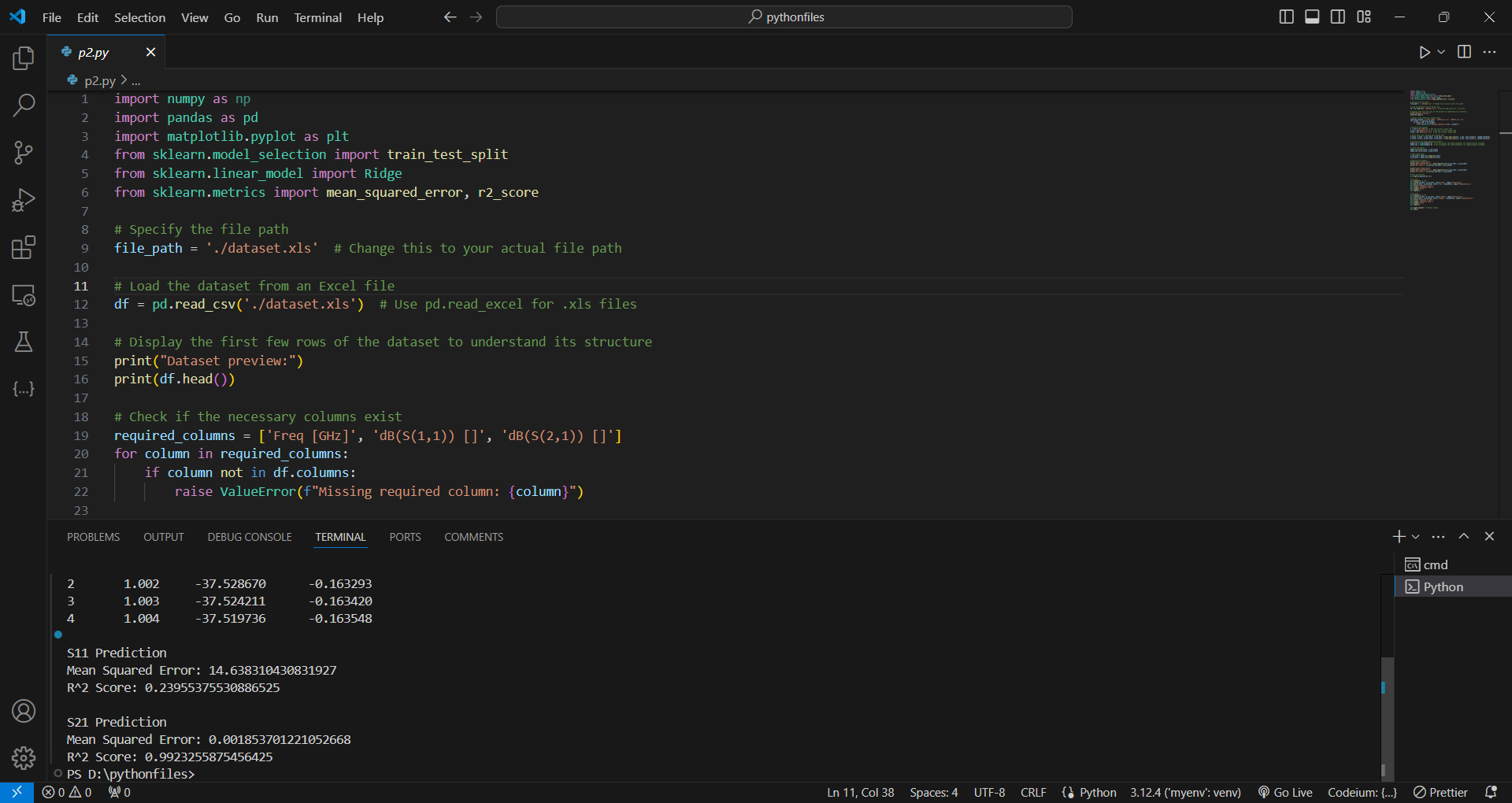
**Data Handling**:

* The dataset is appropriately reshaped to include frequency, S11​ and S21​ as features for prediction.
* The data is processed to train a combined model using these features.

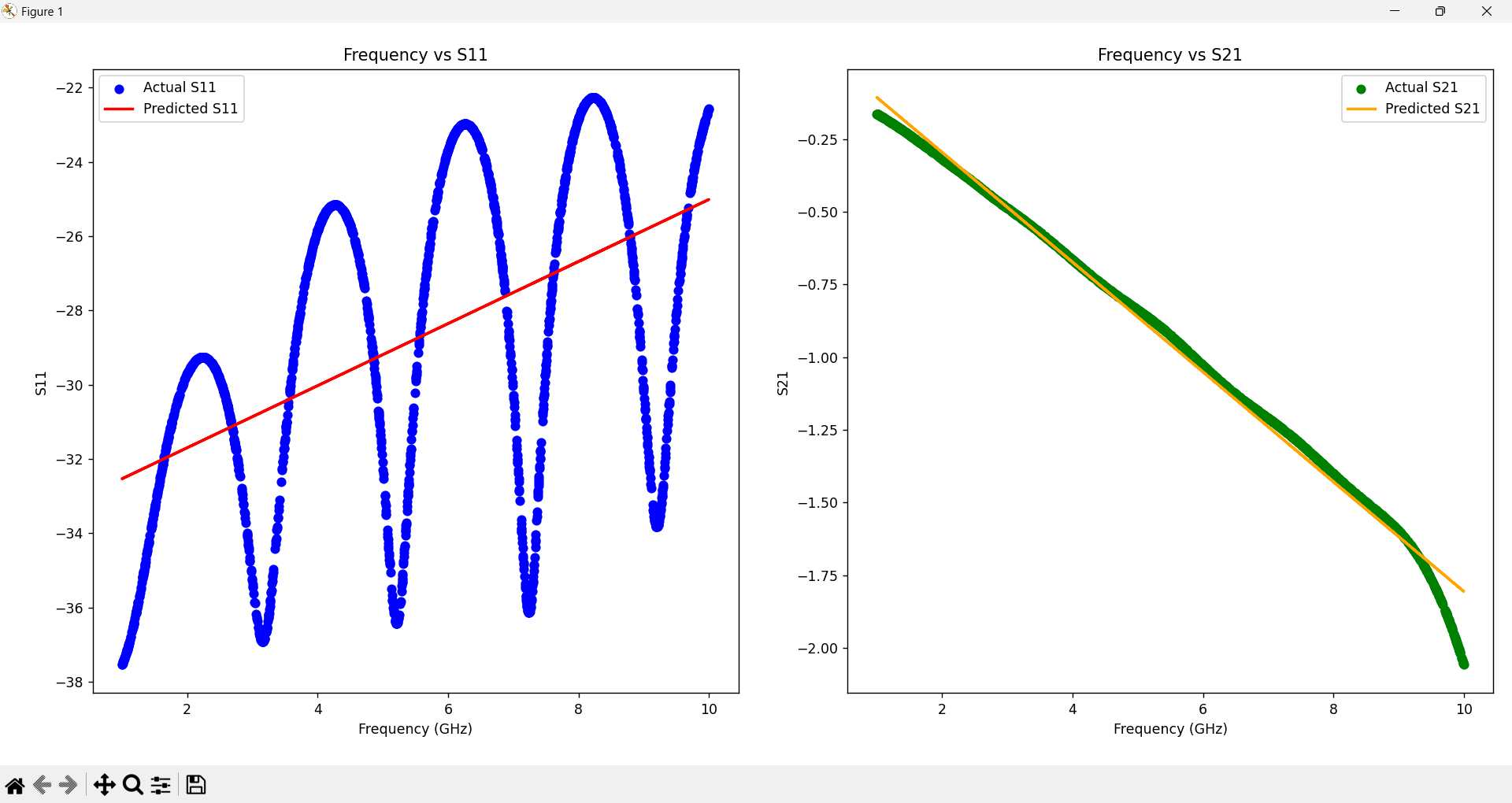
**Modeling**:

* A single Ridge Regression model is trained using frequency, S11​, and S21​ as predictors.
* The model is evaluated using MSE and R² scores.

**Output**:



**Visualization:**



**Improvements**:

* **Feature Integration**: By incorporating multiple features (frequency, S11​, and S21​), the combined model captures more complexity and interactions between variables, potentially enhancing predictive power.
* **Regularization**: The use of Ridge Regression introduces regularization, which can help mitigate overfitting and improve generalization performance.
* **Model Robustness**: The combined model leverages the relationships between features, providing a more comprehensive approach compared to training separate models.

**AI Model for Prediction of Length Parameter of FSS**

**Code 1: Initial Linear Regression with Separate S11 and S21 Models**

**Code:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

# Load the dataset

file\_path = './Length\_variation\_Analyze\_S21.csv.xls'

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

# Reshape the dataset

reshaped\_data = pd.melt(

    data,

    id\_vars=['Freq [GHz]'],

    value\_vars=[col for col in data.columns if "S(1,1)" in col or "S(2,1)" in col],

    var\_name='Parameter',

    value\_name='Value'

)

# Extract length from the parameter names

reshaped\_data['Length'] = reshaped\_data['Parameter'].apply(lambda x: int(x.split("'")[1].replace('mm', '')))

reshaped\_data['Parameter'] = reshaped\_data['Parameter'].apply(lambda x: 'S11' if 'S(1,1)' in x else 'S21')

# Separate S11 and S21 data

s11\_data = reshaped\_data[reshaped\_data['Parameter'] == 'S11']

s21\_data = reshaped\_data[reshaped\_data['Parameter'] == 'S21']

# Split data into features (X) and target (y)

X\_s11 = s11\_data[['Length']]

y\_s11 = s11\_data['Value']

X\_s21 = s21\_data[['Length']]

y\_s21 = s21\_data['Value']

# Split the data into training and testing sets

X\_s11\_train, X\_s11\_test, y\_s11\_train, y\_s11\_test = train\_test\_split(X\_s11, y\_s11, test\_size=0.4, random\_state=42)

X\_s21\_train, X\_s21\_test, y\_s21\_train, y\_s21\_test = train\_test\_split(X\_s21, y\_s21, test\_size=0.4, random\_state=42)

# Train linear regression models

model\_s11 = LinearRegression()

model\_s11.fit(X\_s11\_train, y\_s11\_train)

model\_s21 = LinearRegression()

model\_s21.fit(X\_s21\_train, y\_s21\_train)

# Predict on the test set

y\_s11\_pred = model\_s11.predict(X\_s11\_test)

y\_s21\_pred = model\_s21.predict(X\_s21\_test)

# Evaluate the models

mse\_s11 = mean\_squared\_error(y\_s11\_test, y\_s11\_pred)

mse\_s21 = mean\_squared\_error(y\_s21\_test, y\_s21\_pred)

r2\_s11 = r2\_score(y\_s11\_test, y\_s11\_pred)

r2\_s21 = r2\_score(y\_s21\_test, y\_s21\_pred)

print(f'S11 Model - MSE: {mse\_s11}, R^2: {r2\_s11}')

print(f'S21 Model - MSE: {mse\_s21}, R^2: {r2\_s21}')

- Purpose: Trains separate linear regression models for S11 and S21 parameters based on the length, using simple linear regression.

- Data Handling:

- The data is reshaped to separate S11 and S21 values.

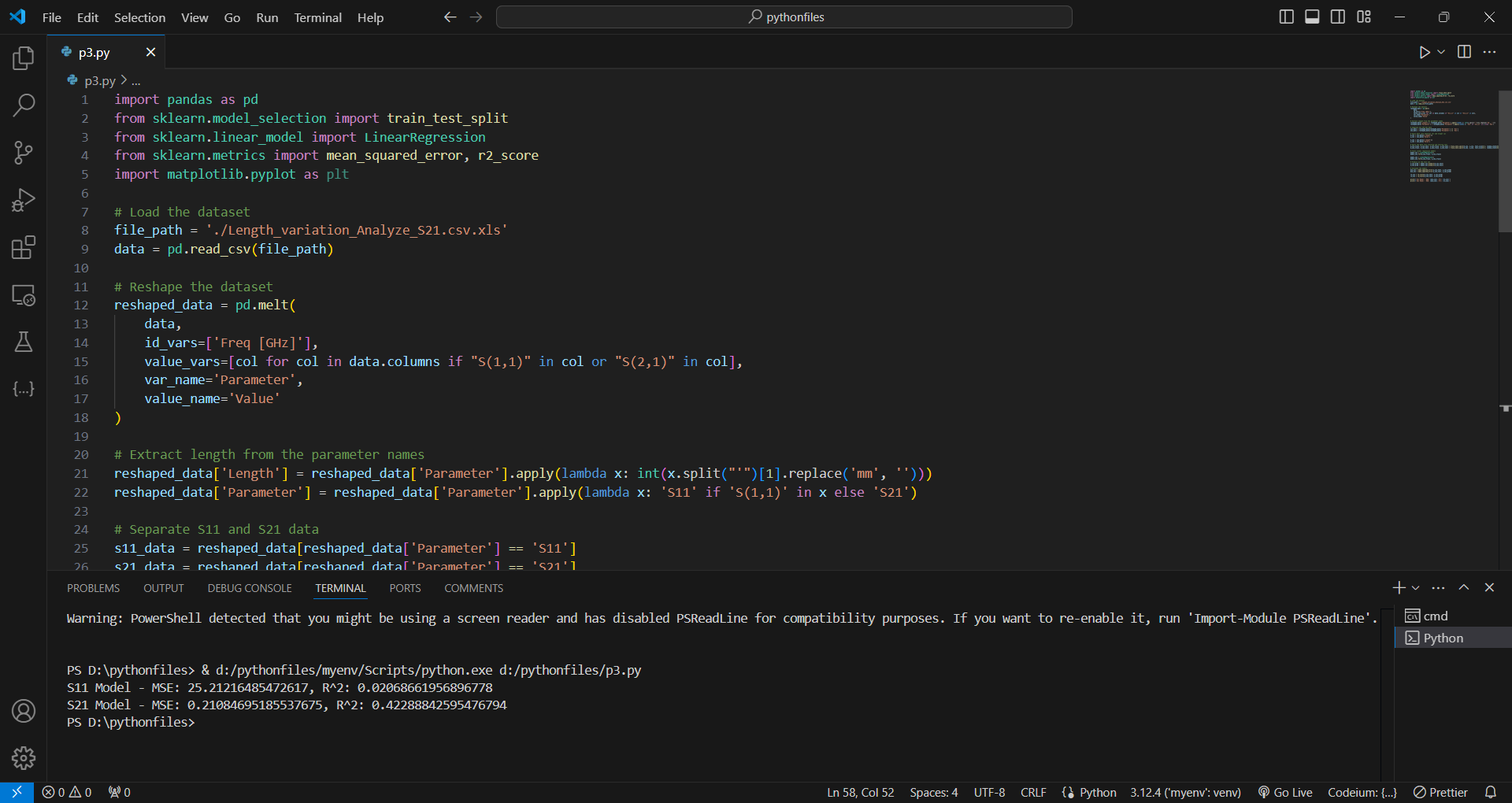
- The dataset is split into training and testing sets for both S11 and S21 data separately.

- Modeling:

- Two separate models are trained, one for S11 and one for S21.

- The models are evaluated using Mean Squared Error (MSE) and R² scores.

- Output:



- Observations:

- This approach is straightforward but does not leverage potential interactions between S11, S21, and frequency.

**Code 2: Combined Model Using Both S11 and S21**

**Code:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

# Load the dataset

file\_path = './Length\_variation\_Analyze\_S21.csv.xls'

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

# Reshape the dataset

reshaped\_data = pd.melt(

    data,

    id\_vars=['Freq [GHz]'],

    value\_vars=[col for col in data.columns if "S(1,1)" in col or "S(2,1)" in col],

    var\_name='Parameter',

    value\_name='Value'

)

# Extract length from the parameter names

reshaped\_data['Length'] = reshaped\_data['Parameter'].apply(lambda x: int(x.split("'")[1].replace('mm', '')))

reshaped\_data['Parameter'] = reshaped\_data['Parameter'].apply(lambda x: 'S11' if 'S(1,1)' in x else 'S21')

# Pivot to create a DataFrame with columns for Freq, S11, S21, and Length

pivoted\_data = reshaped\_data.pivot\_table(index=['Freq [GHz]', 'Length'], columns='Parameter', values='Value').reset\_index()

# Split data into features (X) and target (y)

X = pivoted\_data[['Freq [GHz]', 'S11', 'S21']]

y = pivoted\_data['Length']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.4, random\_state=42)

# Train the linear regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Model - MSE: {mse}, R^2: {r2}')

- Purpose: Combines S11, S21, and frequency as features to predict length, using a single linear regression model.

- Data Handling:

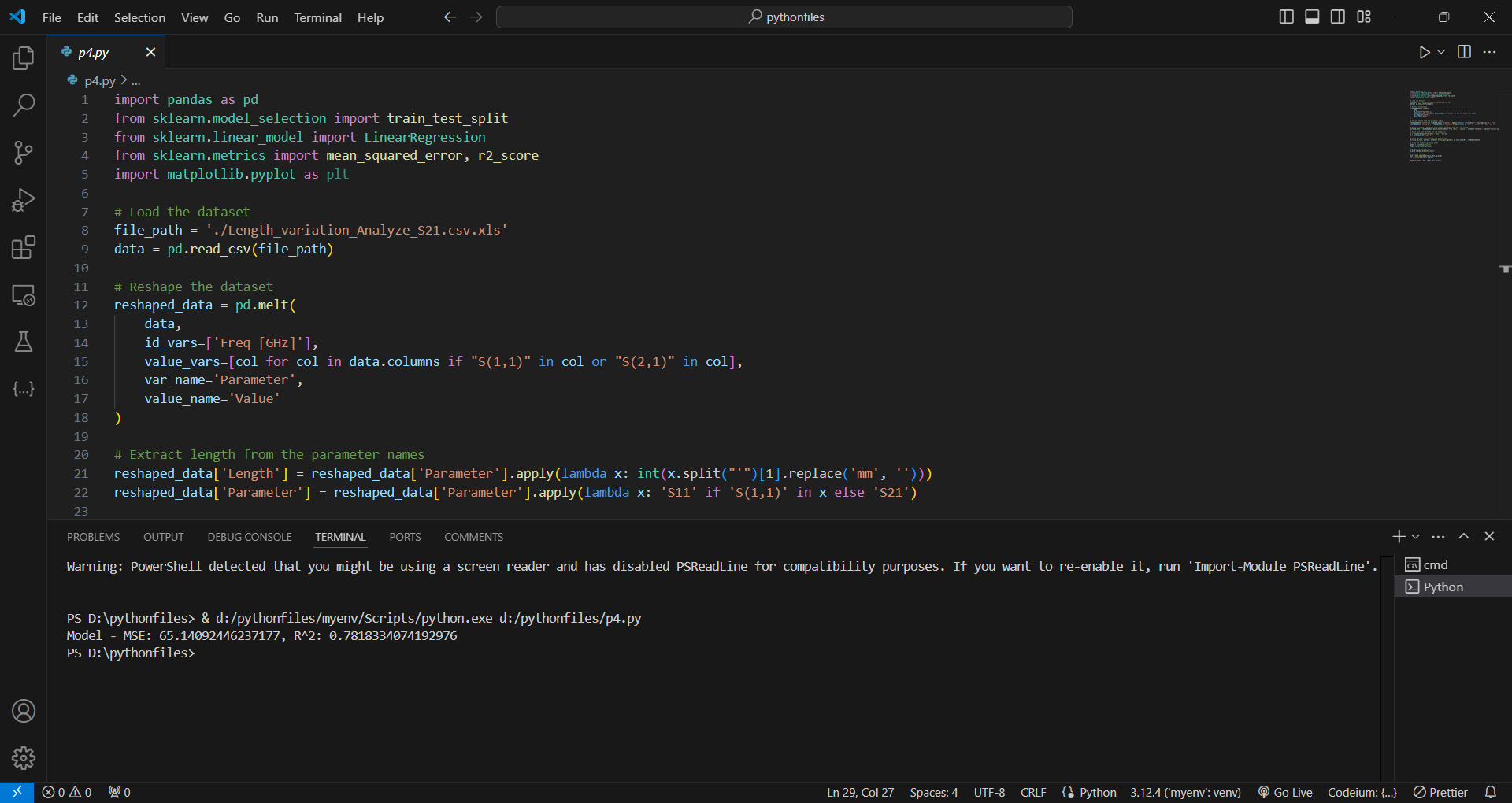
- The dataset is pivoted to create a single DataFrame containing columns for frequency, S11, S21, and length.

- Modeling:

- A single linear regression model is trained using frequency, S11, and S21 as features.

- The model is evaluated with MSE and R².

- Output:



- Improvements:

- By considering multiple features, the model potentially captures more complexity and interactions between variables, improving predictive power.

**Code 3: 3D Linear Regression Model for Frequency and S21**

**Code:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

# Load the dataset

file\_path = './Length\_variation\_Analyze\_S21.csv.xls'  # Ensure this path is correct

data = pd.read\_csv(file\_path)  # Change to pd.read\_excel if the file is an Excel file

# Reshape the dataset

reshaped\_data = pd.melt(

    data,

    id\_vars=['Freq [GHz]'],

    value\_vars=[col for col in data.columns if "dB(S(2,1))" in col],  # Only include S21 values

    var\_name='Parameter',

    value\_name='Value'

)

# Extract length from the parameter names

reshaped\_data['Length'] = reshaped\_data['Parameter'].apply(lambda x: int(x.split("'")[1].replace('mm', '')))

reshaped\_data['Parameter'] = 'S21'  # Since we are only using S21 values

# Pivot to create a DataFrame with columns for Freq, S21, and Length

pivoted\_data = reshaped\_data.pivot\_table(index=['Freq [GHz]', 'Length'], columns='Parameter', values='Value').reset\_index()

# Split data into features (X) and target (y)

X = pivoted\_data[['Freq [GHz]', 'S21']]  # Only use Freq and S21 as features

y = pivoted\_data['Length']

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the linear regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Model - MSE: {mse}, R^2: {r2}')

# Function to predict length

def predict\_length(frequency, s21\_value):

    input\_data = pd.DataFrame([[frequency, s21\_value]], columns=['Freq [GHz]', 'S21'])

    predicted\_length = model.predict(input\_data)[0]

    return predicted\_length

# Input functionality for prediction

input\_freq = float(input("Enter the frequency (GHz): "))

input\_s21 = float(input("Enter the S21 value: "))

predicted\_length = predict\_length(input\_freq, input\_s21)

print(f'Predicted Length: {predicted\_length:.2f} mm')

- Purpose: Focuses solely on S21 and frequency as predictors for length, with an additional 3D visualization.

- Data Handling:

- S11 data is excluded, and only S21 is considered.

- A 3D plot functionality is introduced to visualize relationships.

- Modeling:

- A linear regression model is trained using only frequency and S21.

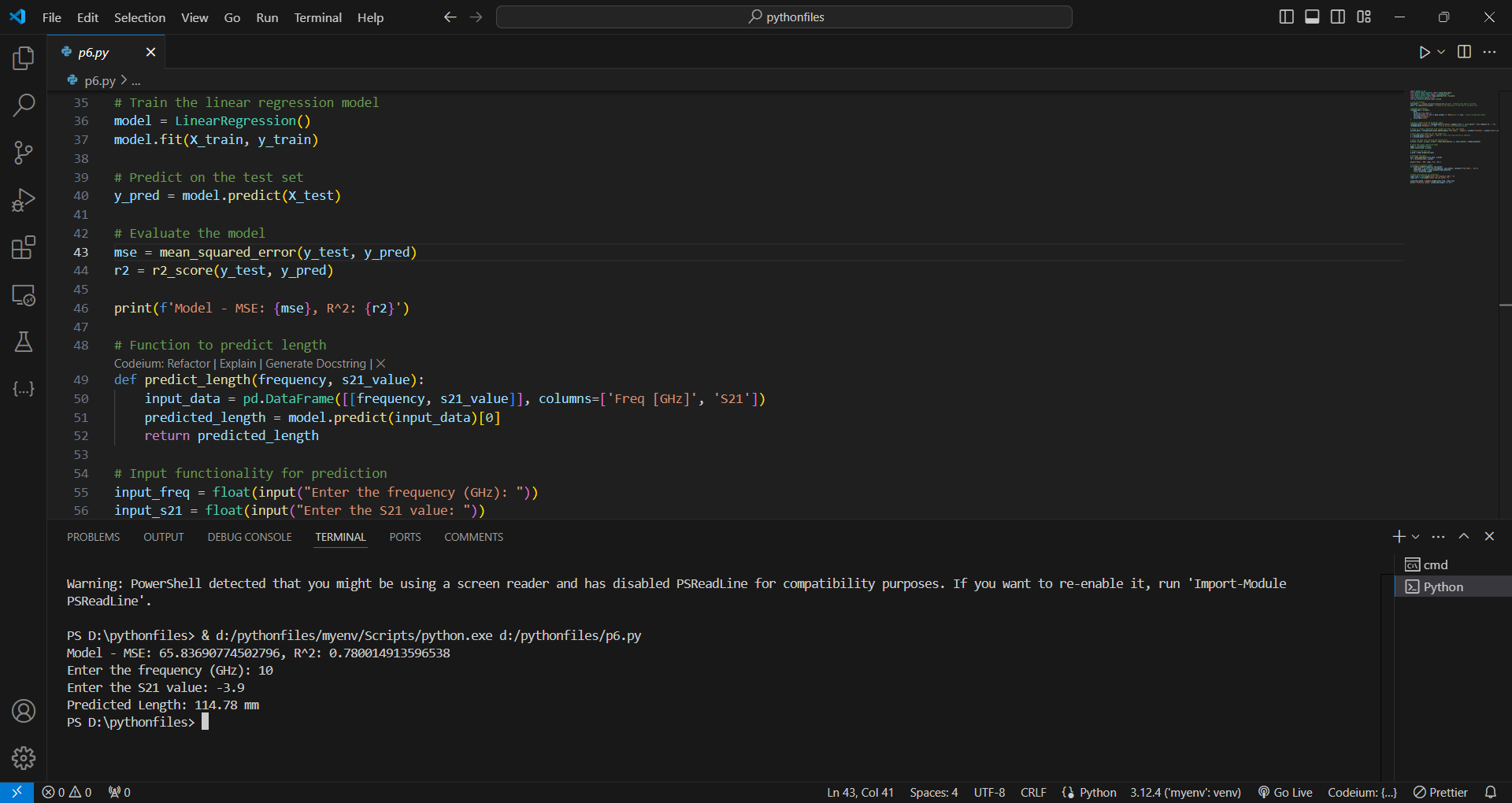
- Model performance is evaluated using MSE and R² scores.

- Input Functionality

- The model is designed to predict the Length of a structure based on two key input variables:

1. **Frequency (in GHz):** The operational frequency of the system.
2. **S21 (Scattering Parameter):** This parameter represents the transmission coefficient, a critical factor in determining the behavior of the structure.

- Output:



- Improvements:

- The model is simplified to focus on S21, and the 3D visualization offers insights into how frequency and S21 jointly influence length.

**Code 4: Polynomial Features and Ridge Regression**

**Code:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import Ridge

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import PolynomialFeatures

import matplotlib.pyplot as plt

from mpl\_toolkits.mplot3d import Axes3D

import numpy as np

# Load the dataset

file\_path = './Length\_variation\_Analyze\_S21.csv.xls'  # Ensure this path is correct

data = pd.read\_csv(file\_path)  # Change to pd.read\_excel if the file is an Excel file

# Reshape the dataset

reshaped\_data = pd.melt(

    data,

    id\_vars=['Freq [GHz]'],

    value\_vars=[col for col in data.columns if "dB(S(2,1))" in col],  # Only include S21 values

    var\_name='Parameter',

    value\_name='Value'

)

# Extract length from the parameter names

reshaped\_data['Length'] = reshaped\_data['Parameter'].apply(lambda x: int(x.split("'")[1].replace('mm', '')))

reshaped\_data['Parameter'] = 'S21'  # Since we are only using S21 values

# Pivot to create a DataFrame with columns for Freq, S21, and Length

pivoted\_data = reshaped\_data.pivot\_table(index=['Freq [GHz]', 'Length'], columns='Parameter', values='Value').reset\_index()

# Split data into features (X) and target (y)

X = pivoted\_data[['Freq [GHz]', 'S21']]  # Only use Freq and S21 as features

y = pivoted\_data['Length']

# Create polynomial features

poly = PolynomialFeatures(degree=2, include\_bias=False)

X\_poly = poly.fit\_transform(X)

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_poly, y, test\_size=0.2, random\_state=42)

# Train a Ridge regression model

model = Ridge(alpha=1.0)

model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Model - MSE: {mse}, R^2: {r2}')

- Purpose: Introduces non-linearity using polynomial features and adds regularization with Ridge regression.

- Data Handling:

- Polynomial features of degree 2 are created to capture non-linear relationships.

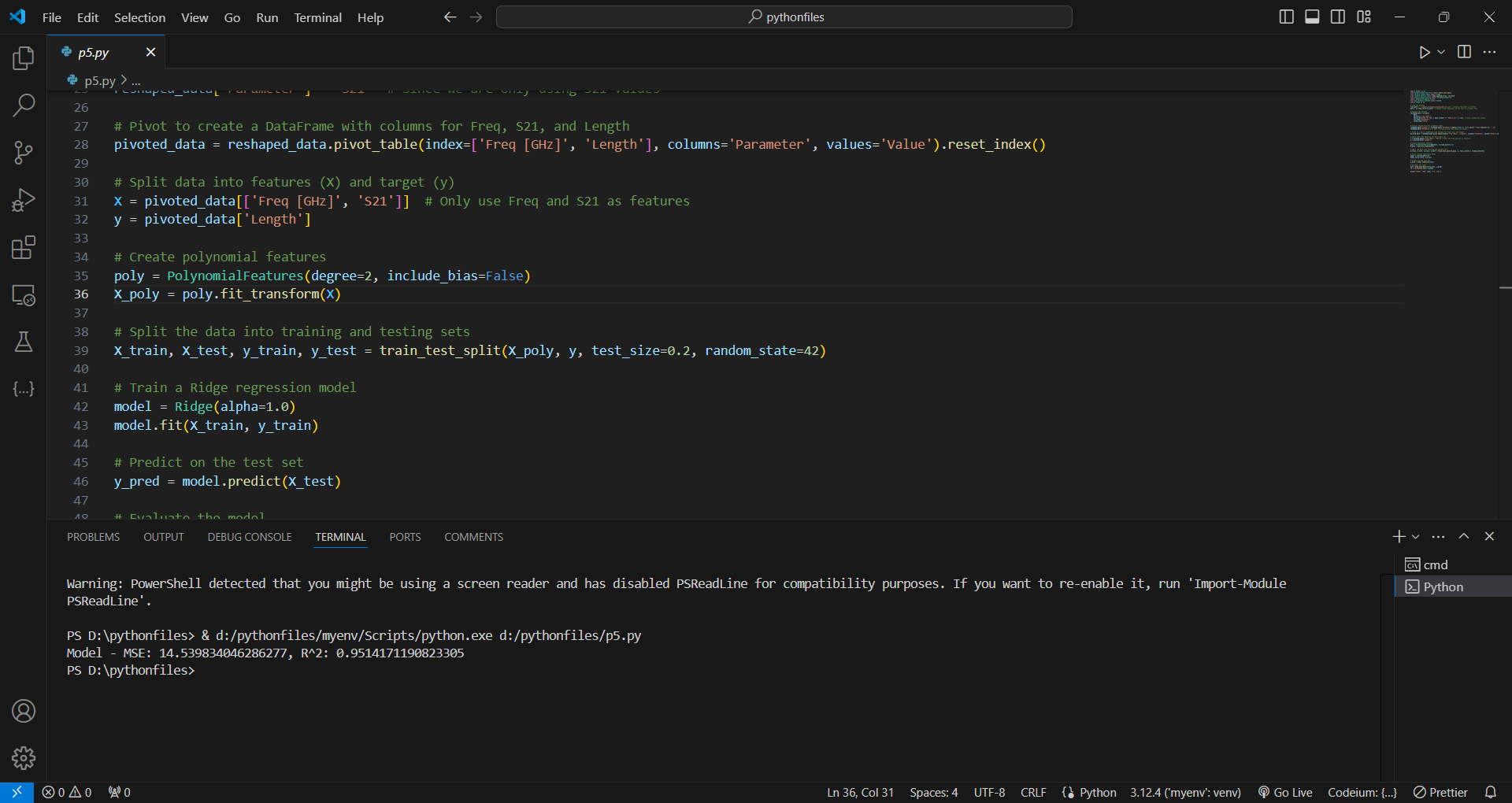
- Modeling:

- A Ridge regression model is trained using the polynomial features.

- Regularization helps prevent overfitting.

- The model is evaluated using MSE and R² scores.

- Output:



- Improvements:

- The use of polynomial features allows the model to capture more complex patterns.

- Ridge regression adds robustness by controlling the model’s complexity.

**Code 5: Grid Search for Hyperparameter Tuning**

**Code:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.linear\_model import Ridge

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import PolynomialFeatures

from sklearn.pipeline import Pipeline

import matplotlib.pyplot as plt

import numpy as np

from scipy.interpolate import griddata

# Load the dataset (adjust for Excel if needed)

file\_path = './Length\_variation\_Analyze\_S21.csv.xls'  # Ensure this path is correct

try:

    data = pd.read\_csv(file\_path)

except FileNotFoundError:

    print(f"File not found: {file\_path}")

    exit()

# Reshape the dataset

reshaped\_data = pd.melt(

    data,

    id\_vars=['Freq [GHz]'],

    value\_vars=[col for col in data.columns if "dB(S(2,1))" in col],

    var\_name='Parameter',

    value\_name='Value'

)

# Extract length from the parameter names

reshaped\_data['Length'] = reshaped\_data['Parameter'].apply(lambda x: int(x.split("'")[1].replace('mm', '')))

reshaped\_data['Parameter'] = 'S21'

# Pivot to create a DataFrame with columns for Freq, S21, and Length

pivoted\_data = reshaped\_data.pivot\_table(index=['Freq [GHz]', 'Length'], columns='Parameter', values='Value').reset\_index()

# Split data into features (X) and target (y)

X = pivoted\_data[['Freq [GHz]', 'S21']]

y = pivoted\_data['Length']

# Create a pipeline that includes PolynomialFeatures and Ridge regression

pipeline = Pipeline([

    ('poly', PolynomialFeatures(degree=3, include\_bias=False)),  # Increase degree for more complexity

    ('ridge', Ridge())

])

# Set up GridSearchCV to tune alpha and degree parameters

param\_grid = {

    'poly\_\_degree': [2, 3, 4],  # Try different degrees

    'ridge\_\_alpha': [0.1, 1.0, 10.0, 100.0]  # Try different regularization strengths

}

# Use cross-validation to find the best parameters

grid\_search = GridSearchCV(pipeline, param\_grid, cv=5, scoring='neg\_mean\_squared\_error')

grid\_search.fit(X, y)

# Use the best model found

best\_model = grid\_search.best\_estimator\_

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train the model on the training set

best\_model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = best\_model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Best Model - MSE: {mse}, R^2: {r2}')

# Function to predict length

def predict\_length(frequency, s21\_value):

    input\_data = pd.DataFrame([[frequency, s21\_value]], columns=['Freq [GHz]', 'S21'])

    predicted\_length = best\_model.predict(input\_data)[0]

    return predicted\_length

# Input functionality for prediction

try:

    input\_freq = float(input("Enter the frequency (GHz): "))

    input\_s21 = float(input("Enter the S21 value: "))

    predicted\_length = predict\_length(input\_freq, input\_s21)

    print(f'Predicted Length: {predicted\_length:.2f} mm')

except ValueError:

    print("Invalid input. Please enter numeric values.")

- Purpose: Enhances the Ridge regression model with hyperparameter tuning using GridSearchCV.

- Data Handling:

- Polynomial features of varying degrees are considered.

- Grid search is used to optimize the degree of polynomial features and the regularization parameter (alpha) in Ridge regression.

- Modeling:

- The best model is selected based on cross-validation performance.

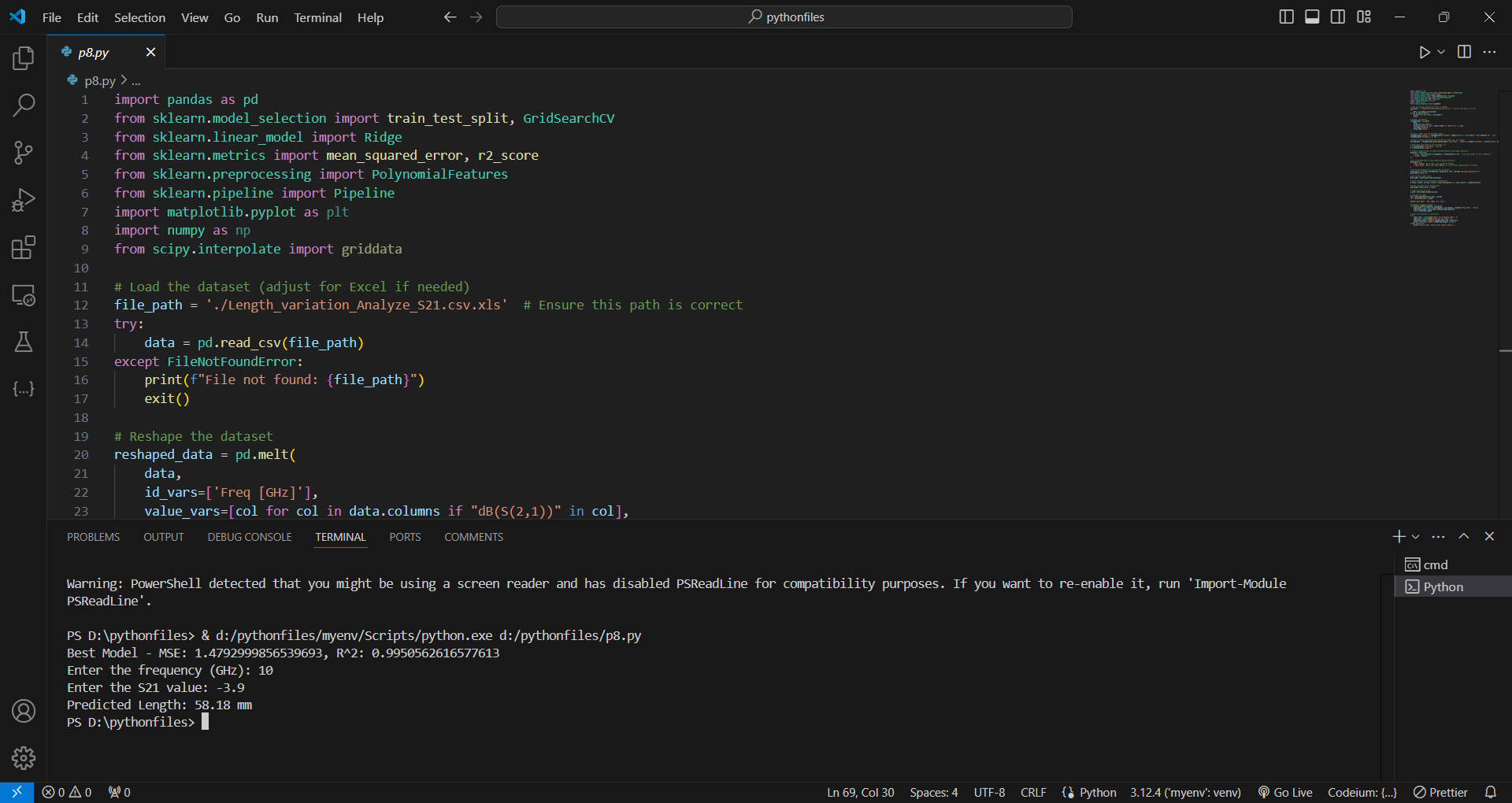
- The model is then retrained and evaluated.

- Input Functionality

- The model is designed to predict the Length of a structure based on two key input variables:

1. **Frequency (in GHz):** The operational frequency of the system.
2. **S21 (Scattering Parameter):** This parameter represents the transmission coefficient, a critical factor in determining the behavior of the structure.

- Output:



- Improvements:

- The model is optimized through hyperparameter tuning, improving accuracy.

**Code 6: Enhanced Visualization with Heatmaps**

**Code:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split, GridSearchCV

from sklearn.linear\_model import Ridge

from sklearn.metrics import mean\_squared\_error, r2\_score

from sklearn.preprocessing import PolynomialFeatures

from sklearn.pipeline import Pipeline

import matplotlib.pyplot as plt

import numpy as np

from scipy.interpolate import griddata

# Load the dataset (adjust for Excel if needed)

file\_path = './Length\_variation\_Analyze\_S21.csv.xls'  # Ensure this path is correct

try:

    data = pd.read\_csv(file\_path)

except FileNotFoundError:

    print(f"File not found: {file\_path}")

    exit()

# Reshape the dataset

reshaped\_data = pd.melt(

    data,

    id\_vars=['Freq [GHz]'],

    value\_vars=[col for col in data.columns if "dB(S(2,1))" in col],

    var\_name='Parameter',

    value\_name='Value'

)

# Extract length from the parameter names

reshaped\_data['Length'] = reshaped\_data['Parameter'].apply(lambda x: int(x.split("'")[1].replace('mm', '')))

reshaped\_data['Parameter'] = 'S21'

# Pivot to create a DataFrame with columns for Freq, S21, and Length

pivoted\_data = reshaped\_data.pivot\_table(index=['Freq [GHz]', 'Length'], columns='Parameter', values='Value').reset\_index()

# Split data into features (X) and target (y)

X = pivoted\_data[['Freq [GHz]', 'S21']]

y = pivoted\_data['Length']

# Create a pipeline that includes PolynomialFeatures and Ridge regression

pipeline = Pipeline([

    ('poly', PolynomialFeatures(degree=3, include\_bias=False)),  # Increase degree for more complexity

    ('ridge', Ridge())

])

# Set up GridSearchCV to tune alpha and degree parameters

param\_grid = {

    'poly\_\_degree': [2, 3, 4],  # Try different degrees

    'ridge\_\_alpha': [0.1, 1.0, 10.0, 100.0]  # Try different regularization strengths

}

# Use cross-validation to find the best parameters

grid\_search = GridSearchCV(pipeline, param\_grid, cv=5, scoring='neg\_mean\_squared\_error')

grid\_search.fit(X, y)

# Use the best model found

best\_model = grid\_search.best\_estimator\_

# Split the data into training and testing sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

# Train the model on the training set

best\_model.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = best\_model.predict(X\_test)

# Evaluate the model

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f'Best Model - MSE: {mse}, R^2: {r2}')

# Function to predict length

def predict\_length(frequency, s21\_value):

    input\_data = pd.DataFrame([[frequency, s21\_value]], columns=['Freq [GHz]', 'S21'])

    predicted\_length = best\_model.predict(input\_data)[0]

    return predicted\_length

# Input functionality for prediction

try:

    input\_freq = float(input("Enter the frequency (GHz): "))

    input\_s21 = float(input("Enter the S21 value: "))

    predicted\_length = predict\_length(input\_freq, input\_s21)

    print(f'Predicted Length: {predicted\_length:.2f} mm')

except ValueError:

    print("Invalid input. Please enter numeric values.")

# Generate a grid of frequency and S21 values for the heatmap

freq\_values = np.linspace(pivoted\_data['Freq [GHz]'].min(), pivoted\_data['Freq [GHz]'].max(), 100)

s21\_values = np.linspace(pivoted\_data['S21'].min(), pivoted\_data['S21'].max(), 100)

freq\_grid, s21\_grid = np.meshgrid(freq\_values, s21\_values)

# Flatten the grids to pass them into the prediction function

freq\_flat = freq\_grid.flatten()

s21\_flat = s21\_grid.flatten()

# Predict lengths for each pair of frequency and S21 in the grid

predicted\_length\_grid = np.array([predict\_length(f, s21) for f, s21 in zip(freq\_flat, s21\_flat)])

predicted\_length\_grid = predicted\_length\_grid.reshape(freq\_grid.shape)

# Interpolate actual lengths onto the same grid for comparison

actual\_length\_grid = griddata(

    (pivoted\_data['Freq [GHz]'], pivoted\_data['S21']),

    pivoted\_data['Length'],

    (freq\_grid, s21\_grid),

)

# Plot the heatmap for actual lengths

plt.figure(figsize=(12, 10))

plt.subplot(2, 1, 1)

plt.contourf(freq\_grid, s21\_grid, actual\_length\_grid, levels=100, cmap='viridis')

plt.colorbar(label='Actual Length (mm)')

plt.xlabel('Frequency (GHz)')

plt.ylabel('S21 (dB)')

plt.title('Heatmap of Actual Length')

# Plot the heatmap for predicted lengths, limiting to 100mm

plt.subplot(2, 1, 2)

plt.contourf(freq\_grid, s21\_grid, predicted\_length\_grid, levels=np.linspace(0, 100, 100), cmap='viridis')

plt.colorbar(label='Predicted Length (mm)')

plt.xlabel('Frequency (GHz)')

plt.ylabel('S21 (dB)')

plt.title('Heatmap of Predicted Length (Limited to 100mm)')

plt.tight\_layout()

plt.show()

- Purpose: Builds on the previous model with improved heatmap visualizations to better compare predictions with actual lengths.

- Data Handling:

- Similar to Code 5, with additional focus on heatmap visualization.

- Modeling:

- The best model is selected based on cross-validation performance.

- The model is then retrained and evaluated.

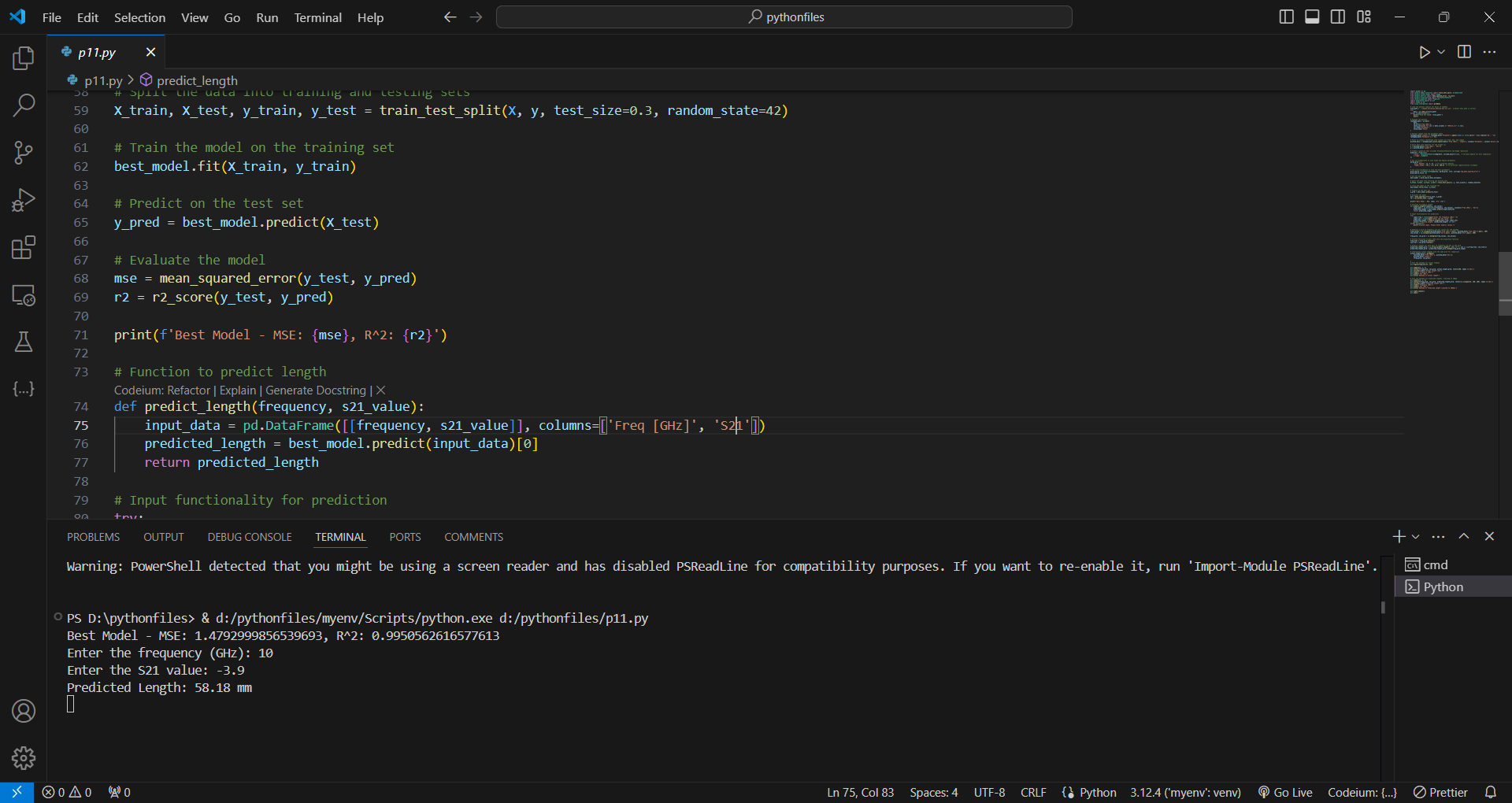
- **Heatmap Generation:**

* The code generates a grid of frequency and S21 values.
* It predicts the length for each combination of these values using the trained model.
* It then creates two heatmaps: one for the actual lengths and one for the predicted lengths. The predicted heatmap is limited to a range of 0 to 100 mm for better visualization.

**- Input Functionality:**

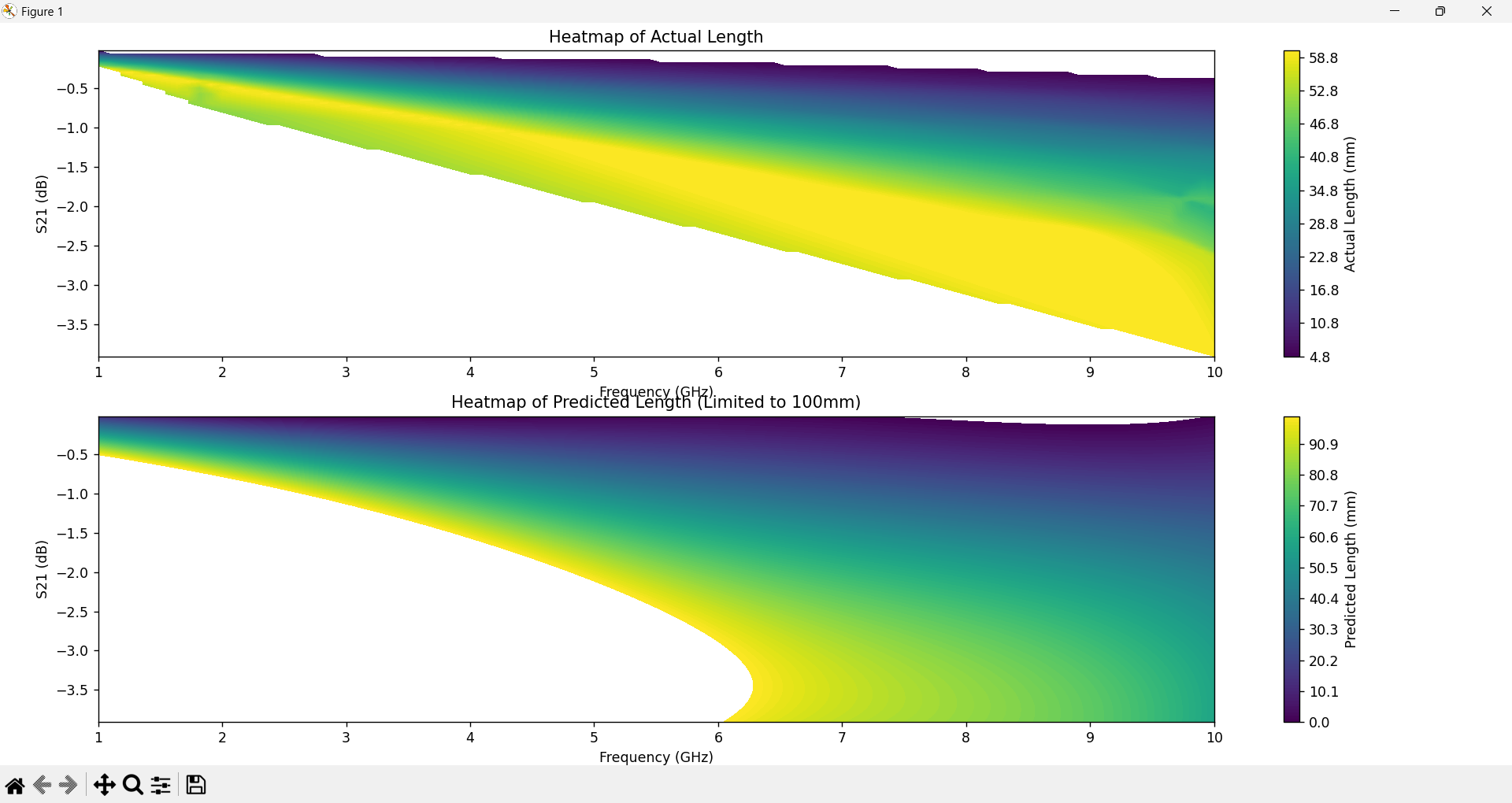
* Users can input a specific frequency and S21 value to get a predicted length.
* Error handling ensures that only numeric input is accepted.

- Output:



- Visualization:

- Detailed heatmaps comparing predicted lengths and actual lengths are created, with a focus on visual clarity.



- Improvements:

- Provides a comprehensive visual representation of model performance, making it easier to identify areas of improvement.

Summary of Improvements

1. Complexity and Feature Interactions:

- Initial models were simple but progressively added complexity by including more features, polynomial transformations, and regularization.

- Improvements in R² scores and MSE across codes indicate better model performance as complexity and tuning were introduced.

2. Visualization:

- Early codes lacked visualization, while later iterations included 3D plots and heatmaps.

- Enhanced visualization helped in better understanding the model’s behavior and accuracy.

3. Hyperparameter Tuning:

- The introduction of GridSearchCV allowed for a more optimized model, which improved overall performance metrics.

4. Focus on S21:

- Later codes focused exclusively on S21 values, aligning with your preference to use only S21 for model training.

Decrements

- Exclusion of S11:

- While the later models improved performance, they neglected S11 data entirely, which might have provided useful additional information if combined effectively with S21.