Course M-GFP3: Imaging and non-imaging spectroscopy:
Term Paper

Strategies to enhance predictive modeling of soil organic carbon (SOC) using the LUCAS topsoil spectral library.

Deepak, Khuzaima, Luis

2025-02-20

Table of contents

1 Packages

```
# Use autoreload to automatically reload modules
%load_ext autoreload
%autoreload 2
```

```
import own_functions
```

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
from scipy.signal import savgol_filter
from scipy.stats import pearsonr
from sklearn.cross_decomposition import PLSRegression
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, pairwise_distances
from sklearn.model_selection import cross_val_score, train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
import geopandas as gpd
```

```
import autogluon
```

2 Data

Data splitting (5 P):

- Split your data into a calibration data set (~70%) and an independent test data set (~30%).
- Show that both are representative of the full data set.
- For procedures with randomized approaches, please define and note the seed (in R: set.seed( )) to make the split reproducible for the instructors.
- From this point onward, the composition of the test data set must remain constant and unchanged for all subsequent tasks

2.1 Load and Clean

```
# Load data
data = pd.read_csv('France_spc.csv')

# Remove unnecessary column
data = data.drop(columns=['Unnamed: 0'])

print(f"Data rows: {data.shape[0]}, columns: {data.shape[1]}")
display(data.head())
```

```
Data rows: 2807, columns: 1000
```

	500	502	504	506	508	510	512	514	516	518
0	0.137399	0.139045	0.140758	0.142544	0.144388	0.146281	0.148221	0.150205	0.152239	0.154322
1	0.141740	0.142851	0.144007	0.145208	0.146450	0.147726	0.149025	0.150348	0.151702	0.153081
2	0.140713	0.142216	0.143778	0.145392	0.147053	0.148756	0.150488	0.152257	0.154059	0.155892
3	0.128922	0.129908	0.130919	0.131959	0.133019	0.134102	0.135196	0.136307	0.137433	0.138571
4	0.161760	0.163229	0.164741	0.166298	0.167895	0.169530	0.171194	0.172890	0.174611	0.176356

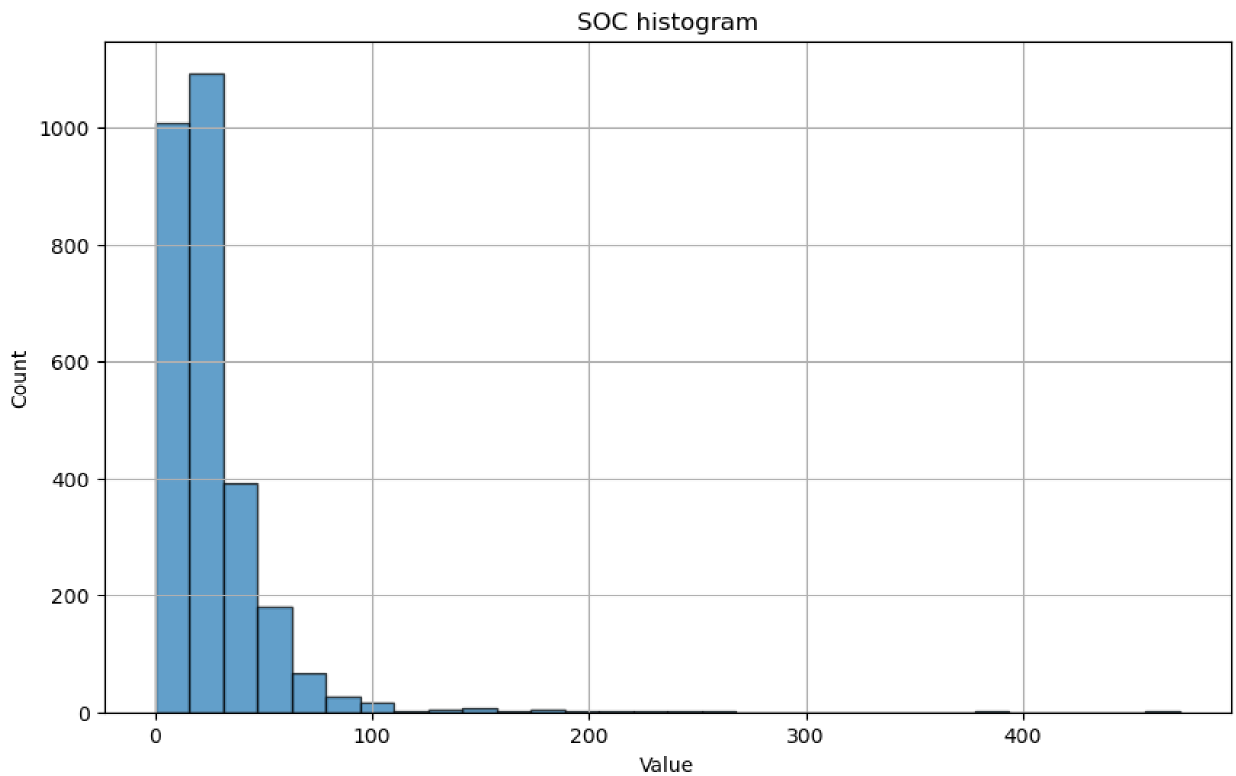
```
target_raw = pd.read_csv('France_lab.csv')
lat_lon = target_raw[['GPS_LAT', 'GPS_LONG']]
target = target_raw['SOC']
print(f"Target rows: {target.shape[0]}")
```

```
Target rows: 2807
```

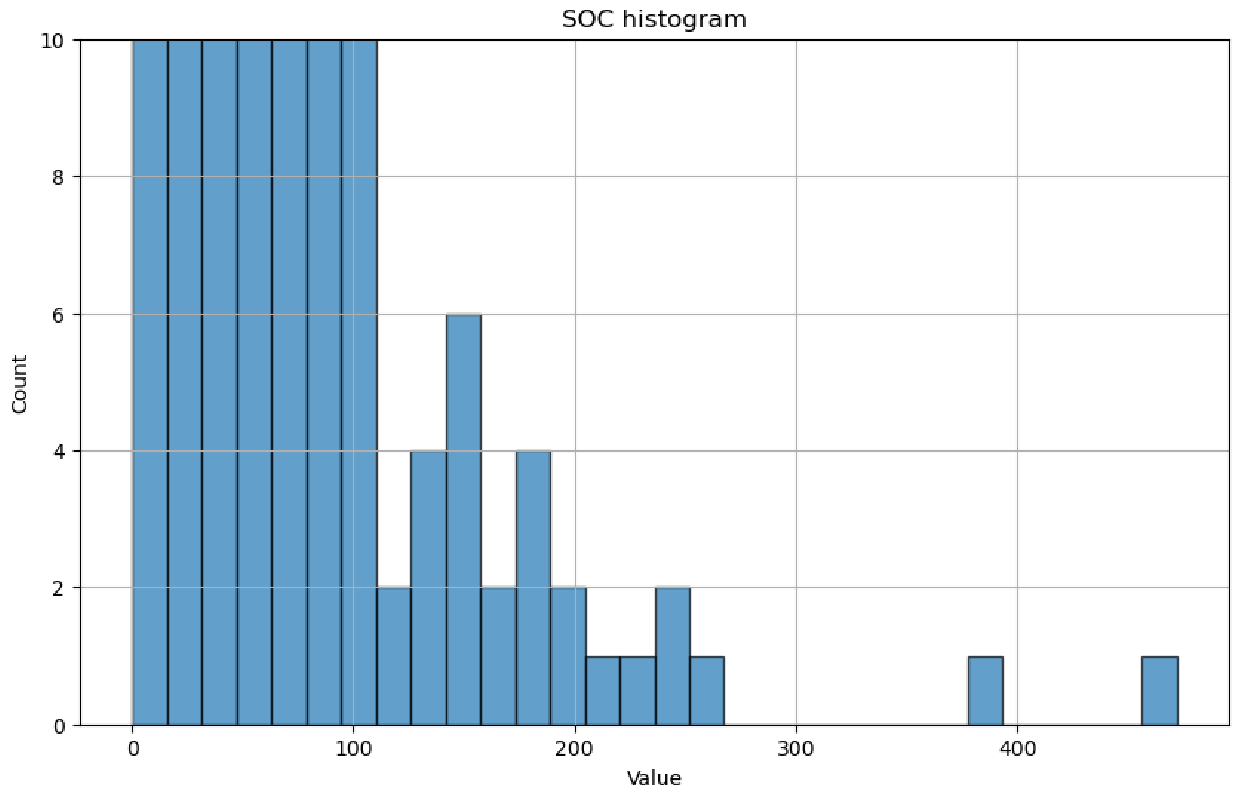
```
target_raw
```

	Unnamed: 0	SAMPLE_ID	CLAY	SILT	SAND	SOC	CaCO3	N	P	K	CEC	GP
0	1	10000	40.0	52.0	8.0	15.6	1	1.4	42.6	491.1	24.6	WG
1	2	10001	26.0	18.0	56.0	19.8	1	1.6	19.5	279.1	20.6	WG
2	3	10002	22.0	41.0	37.0	33.5	1	2.6	37.8	399.1	15.0	WG
3	4	10004	27.0	47.0	26.0	66.1	21	6.6	147.7	1080.6	30.5	WG
4	5	10005	16.0	32.0	52.0	38.1	0	2.6	49.6	293.9	7.8	WG
...	...	...	...	...	...	...	...	...	...	...	...	...
2802	2803	9994	19.0	62.0	19.0	9.1	0	1.2	44.5	131.8	9.7	WG
2803	2804	9995	16.0	41.0	42.0	13.4	0	1.4	33.0	184.4	7.2	WG
2804	2805	9996	13.0	29.0	58.0	8.7	3	1.3	104.9	425.4	7.7	WG
2805	2806	9997	20.0	38.0	42.0	30.6	0	3.0	56.1	107.8	12.6	WG
2806	2807	9998	11.0	56.0	34.0	5.9	0	0.7	39.7	172.1	3.8	WG

```
# plot soc histogram
plt.figure(figsize=(10, 6))
plt.hist(target, bins=30, edgecolor='k', alpha=0.7)
plt.title('SOC histogram')
plt.xlabel('Value')
plt.ylabel('Count')
plt.grid(True)
plt.show()
```




```
# plot soc histogram
plt.figure(figsize=(10, 6))
plt.hist(target, bins=30, edgecolor='k', alpha=0.7)
plt.title('SOC histogram')
plt.xlabel('Value')
plt.ylabel('Count')
plt.ylim((0,10))
plt.grid(True)
plt.show()
```



2.2 Sampling and Splitting

```
# Extract features and target as numpy array
X = data.values
y = target.values
```

```
### Sampling strategies
# Step 1: Generate or Load Data
np.random.seed(100)  # Set seed for reproducibility

# Step 2: Random Split (70% Calibration, 30% Test)
X_train_random, X_test_random, y_train_random, y_test_random = train_test_split(X, y, test_s

print(f"Random Split: {X_train_random.shape[0]} training samples, {X_test_random.shape[0]} t

# Step 3: Apply Kennard-Stone to select 70% of the data
n_train = int(0.7 * X.shape[0])

# Get indices
ks_indices = own_functions.kennard_stone(X, n_train)

# Select Training data
```

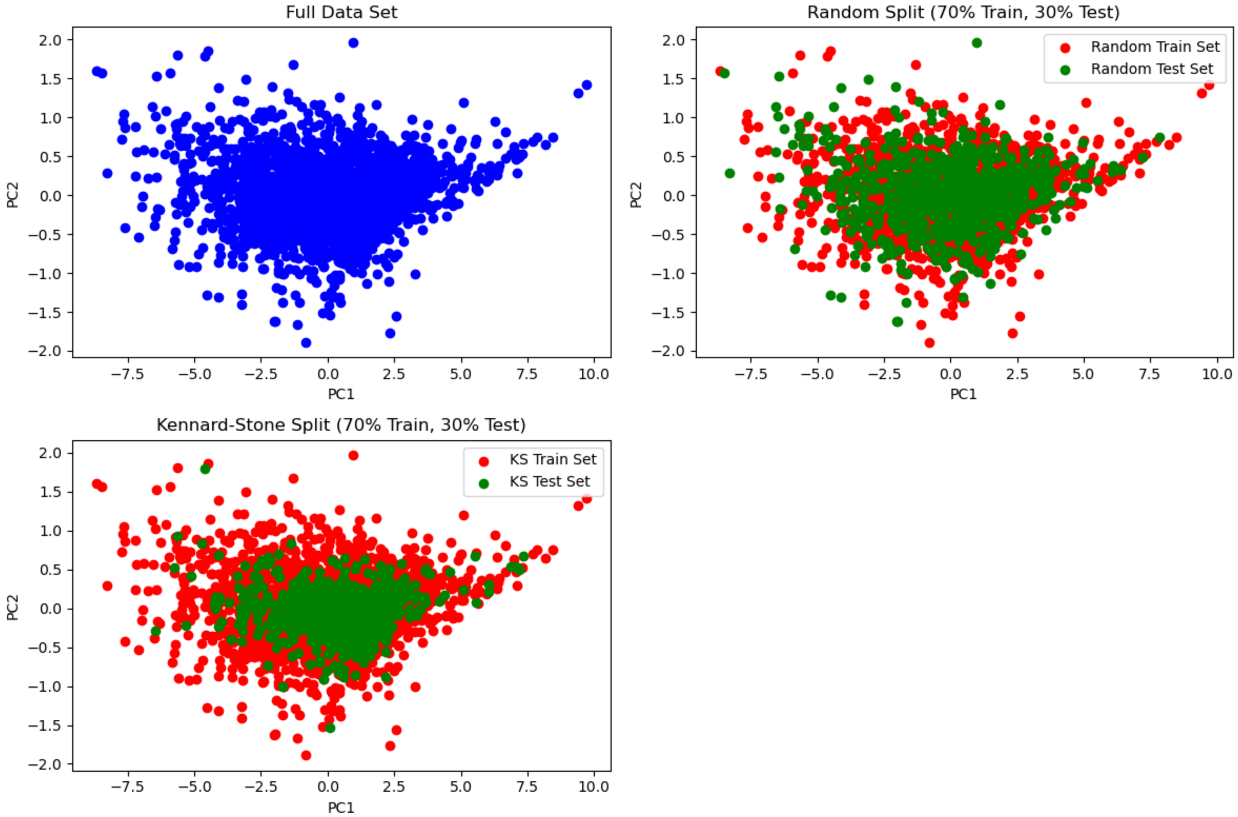
```python
X_train_ks = X[ks_indices,:]
y_train_ks = y[ks_indices]

# Select Test
test_indices = np.setdiff1d(np.arange(X.shape[0]), ks_indices)
X_test_ks = X[test_indices]
y_test_ks = y[test_indices]

print(f"Kennard-Stone: {X_train_ks.shape[0]} training samples, {X_test_ks.shape[0]} test sam

# Step 4: PCA for Visualization
pca = PCA(n_components=2)

# Fit PCA on full data
X_pca = pca.fit_transform(X)

# Transform data
X_train_random_pca = pca.transform(X_train_random)  # PCA on random calibration set
X_test_random_pca = pca.transform(X_test_random)  # PCA on random test set
X_cal_ks_pca = pca.transform(X_train_ks)  # PCA on Kennard-Stone calibration set
X_test_ks_pca = pca.transform(X_test_ks)  # PCA on Kennard-Stone test set
```

```
Random Split: 1964 training samples, 843 test samples
Kennard-Stone: 1964 training samples, 843 test samples
```

```python
#TODO: Show that both test and train are representative of the full dataset
```

```python
# Step 5: Plot Results
own_functions.plot_pca_comparison(X_full=X,
                                  X_train_random=X_train_random,
                                  X_test_random=X_test_random,
                                  X_train_ks=X_train_ks,
                                  X_test_ks=X_test_ks)
```

```
Random Split:
Train set shape: (1964, 1000)
Test set shape: (843, 1000)

Kennard-Stone Split:
Train set shape: (1964, 1000)
Test set shape: (843, 1000)
```




```
X_train = X_train_ks
y_train = y_train_ks

X_test = X_test_ks
y_test = y_test_ks
```

3 Basemodel

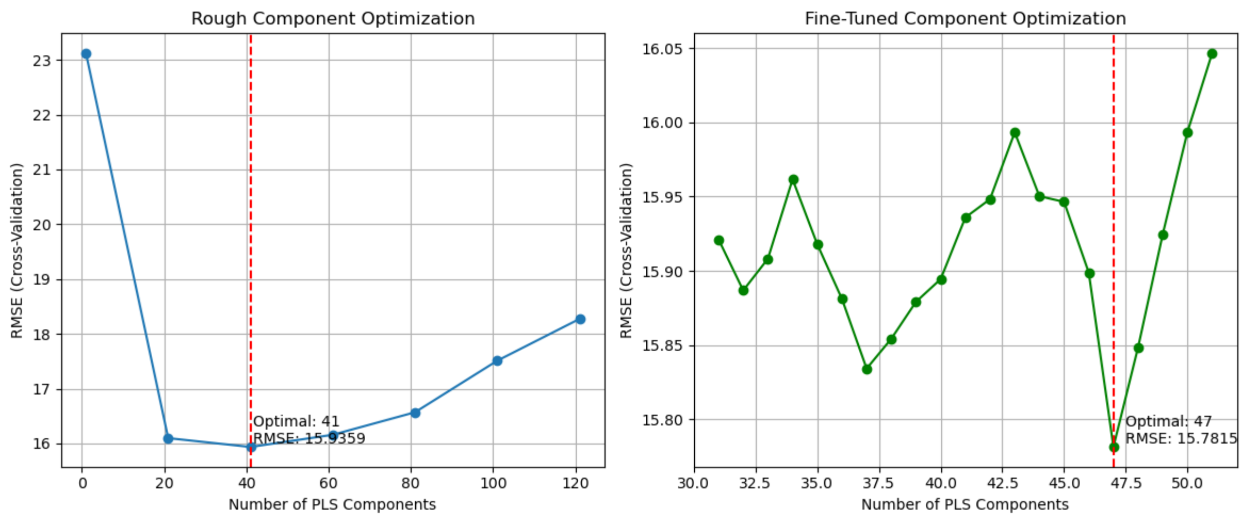
Baseline model (5 P): - Develop a global baseline PLSR model using the *calibration dataset* - (entire VNIR range from 500 nm to 2499 nm in steps of 2 nm) - *without* applying any *spectral preprocessing.* - The target variable is soil organic carbon (SOC). - Perform *internal optimization* to *determine* the *optimal number* of *latent PLS variables* - *report your selected value.* - Apply the optimized model* to the independent test set. - *Compute the validation metrics* (R^2, RMSE, bias, and RPD) - visualize* the results in a *scatter plot* (observed vs. predicted values) - and assess the model's performance.

3.1 Finding optimal number of commponents

```python
import own_functions
```

```python
plsr_base_components = own_functions.optimize_pls_components(X_train=X_train,
                                      y_train=y_train,
                                      max_components=140,
                                      step=20,
                                      fine_tune=True,
                                      show_progress=True,
                                      plot_results=True
                                      )
```

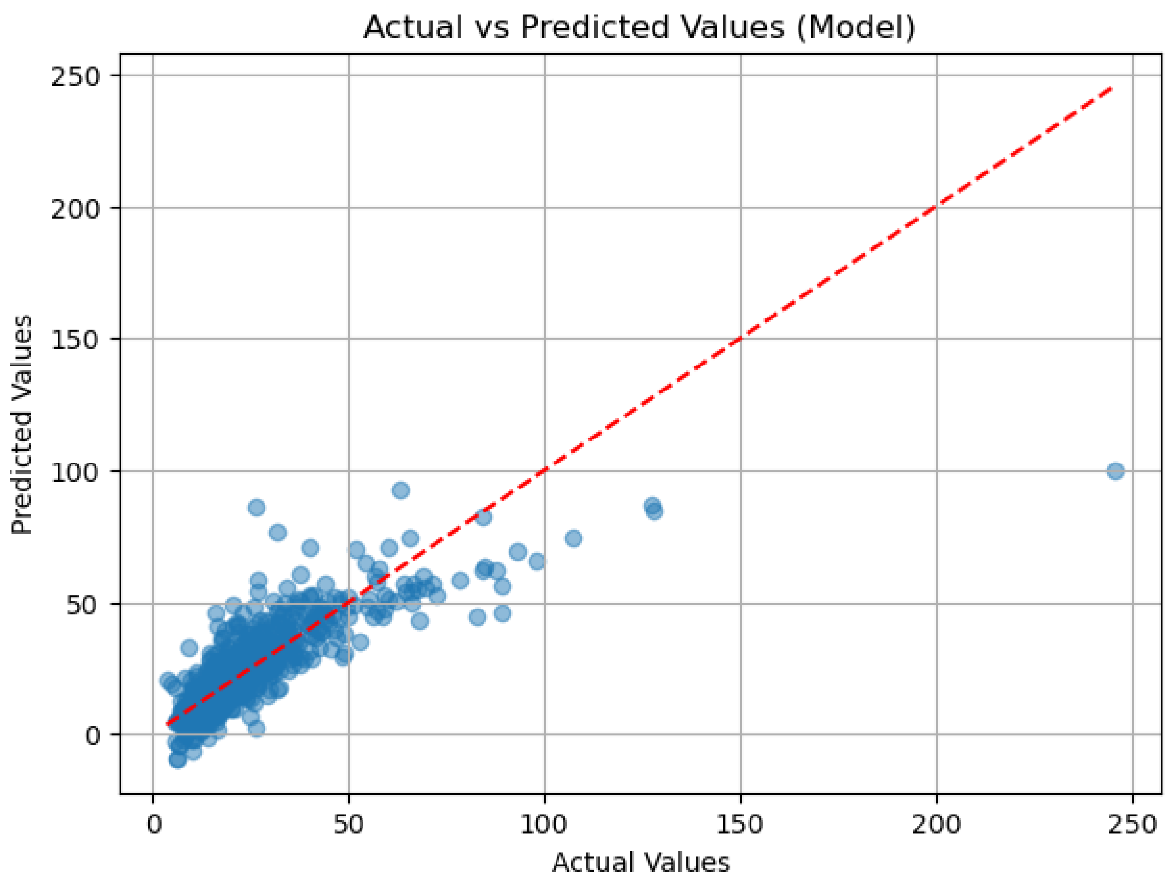
```
Rough Optimization:   0%|          | 0/7 [00:00<?, ?it/s]

Fine Tuning:   0%|          | 0/21 [00:00<?, ?it/s]
```



3.2 Evaluating Base Model

```python
plsr_base_model = PLSRegression(n_components=plsr_base_components["optimal_n"])
plsr_base_model.fit(X_train, y_train)

plsr_base_eval = own_functions.evaluate_model(plsr_base_model,
                             X_test=X_test,
                             y_test=y_test,
                             print_metrics=True,
                             show_plot=True
                             )
```

```
Root Mean Squared Error (RMSE): 10.0547
R²: 0.6552
Bias: -0.1772
RPD: 1.7030
```



4 Model Improvement Strategies (5 P per strategy):

- Develop and evaluate three distinct strategies to improve the baseline model,
  - using the **same independent test set for validation**.
- For each strategy, report the validation metrics
  - (R^2, RMSE, bias, and RPD),
  - visualize the best result in a scatter plot (observed vs.predicted values)
  - assess the performance of these alternative models.
  - Use the same independent test set for all strategies to ensure that validation metrics are directly comparable.

IMPORTANT: Testing two or more spectral preprocessing methods is considered one strategy, not multiple strategies. Similarly, testing one or more alternative regression algorithms counts as one strategy, not multiple.

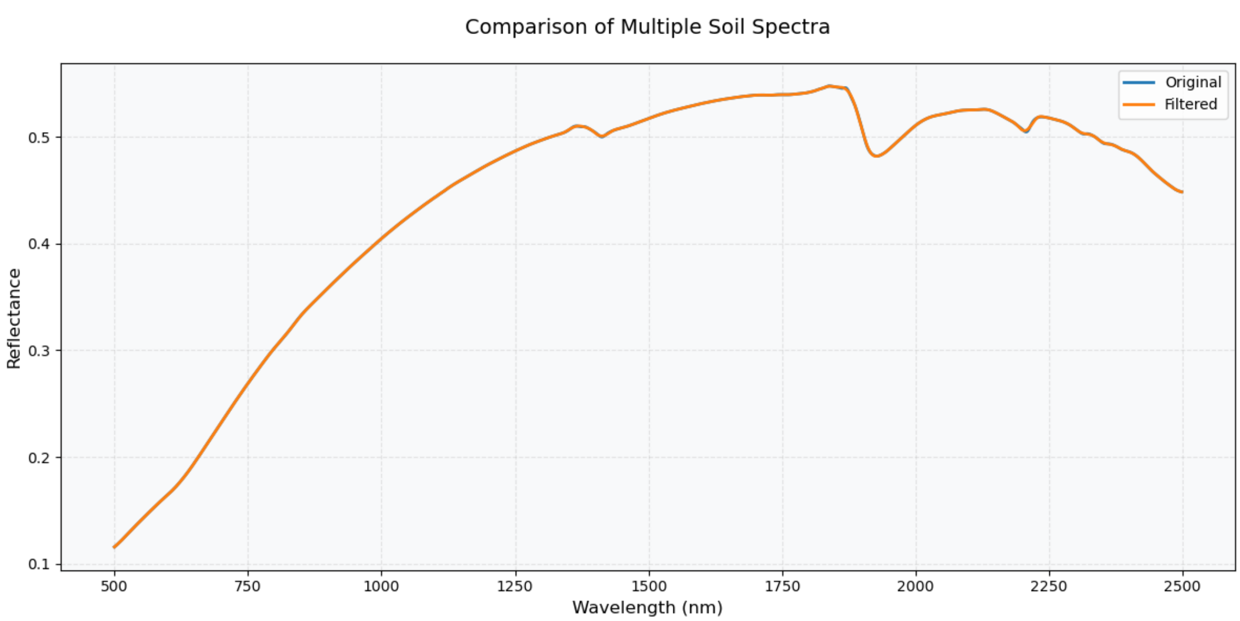
4.1 Varying Preprocessing Strategy

4.1.1 Savitzgy-Golay

```
#TODO: Is scaling necessary? - Does not seem to make a difference -> removed

# Applying Savitzky-Golay filter to calibration and test data
X_train_sg = own_functions.apply_savitzky_golay(X_train, window_length=31, polyorder=4, deri
X_test_sg = own_functions.apply_savitzky_golay(X_test, window_length=31, polyorder=4, deriv=
```

```
def plot_spectra_comparison(*spectra, wavelengths=None, labels=None, title="Spectral Compari
    """
    Create a comparison plot of multiple spectra.

    Args:
        *spectra: Variable number of spectrum data arrays
        wavelengths: X-axis values (optional)
        labels: List/tuple of labels for legend matching number of spectra
        title: Plot title
    """
    import matplotlib.pyplot as plt

    # Create figure and axis
    plt.figure(figsize=(12, 6))

    # Generate x-axis values if not provided
    if wavelengths is None:
        wavelengths = range(len(spectra[0]))

    # Set default labels if not provided
    if labels is None:
        labels = [f'Spectrum {i+1}' for i in range(len(spectra))]

    # Ensure number of labels matches number of spectra
    if len(labels) != len(spectra):
        raise ValueError(f"Number of labels ({len(labels)}) must match number of spectra ({l

    # Plot each spectrum
    for spectrum, label in zip(spectra, labels):
        plt.plot(wavelengths, spectrum, label=label, linewidth=2)

    # Customize plot
    plt.title(title, fontsize=14, pad=20)
    plt.xlabel('Wavelength (nm)', fontsize=12)
    plt.ylabel('Reflectance', fontsize=12)
    plt.grid(True, linestyle='--', alpha=0.7)
```

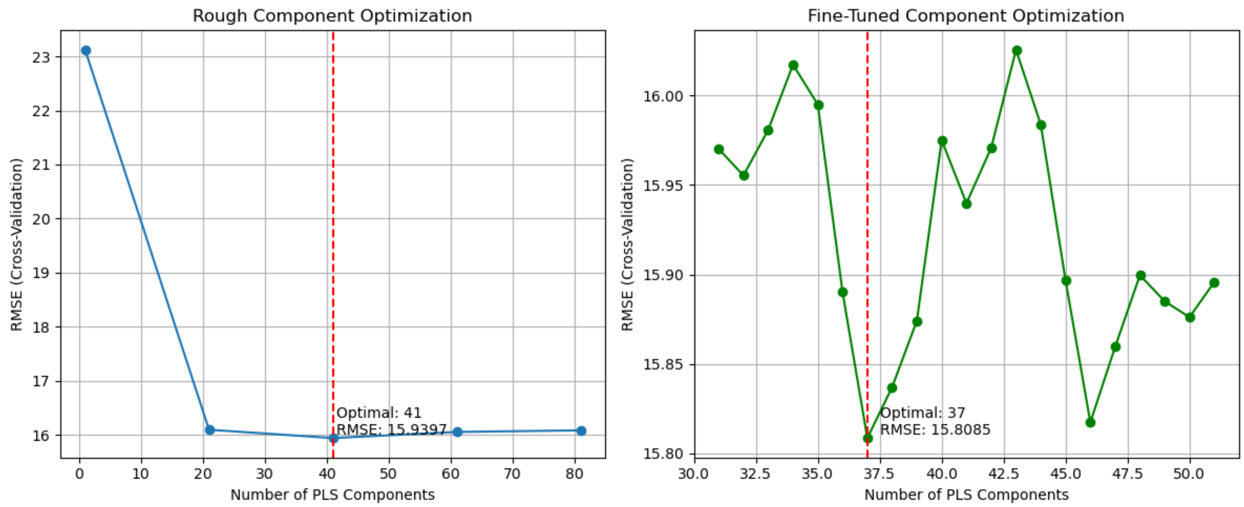
```python
    plt.legend(fontsize=10)

    # Add a subtle background color
    plt.gca().set_facecolor('#f8f9fa')
    plt.grid(True, linestyle='--', alpha=0.3)

    # Adjust layout
    plt.tight_layout()
    plt.show()

# Example usage with multiple spectra:
plot_spectra_comparison(
    X_train[2],
    X_train_sg[2],
    wavelengths=range(500, 2500, 2),
    labels=['Original', 'Filtered'],
    title='Comparison of Multiple Soil Spectra'
)
```




```python
plsr_sgolay_components = own_functions.optimize_pls_components(X_train=X_train_sg,
                                            y_train=y_train,
                                            max_components=100,
                                            step=20,
                                            fine_tune=True,
                                            show_progress=True,
                                            plot_results=True
                                            )
```

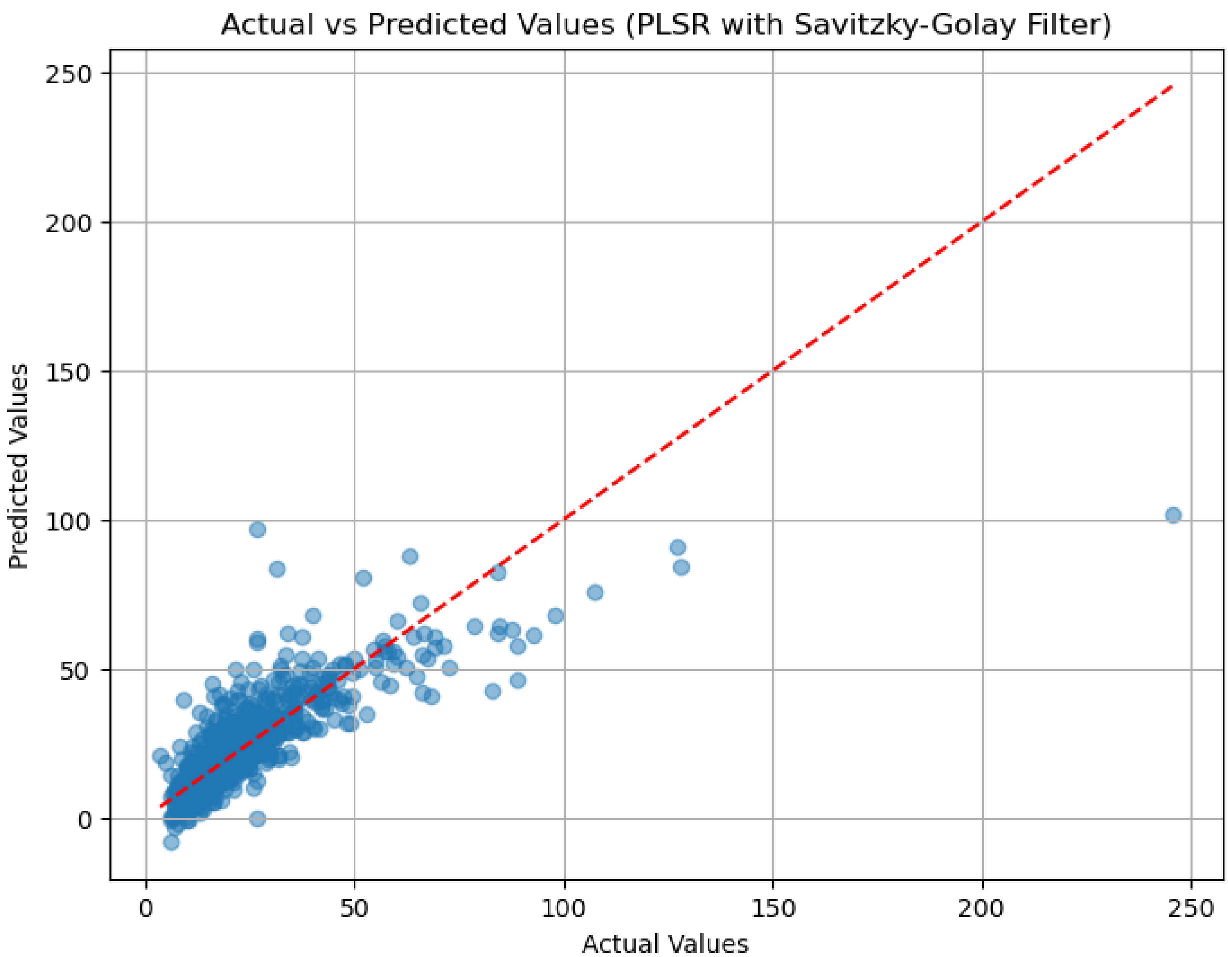
```
Rough Optimization:   0%|          | 0/5 [00:00<?, ?it/s]
```

```
Fine Tuning:   0%|          | 0/21 [00:00<?, ?it/s]
```




```python
plsr_sg_model = PLSRegression(n_components=plsr_sgolay_components["optimal_n"])
plsr_sg_model.fit(X_train_sg, y_train)

plsr_sg_eval = own_functions.evaluate_model(plsr_sg_model,
                             X_test=X_test_sg,
                             y_test=y_test,
                             print_metrics=True,
                             show_plot=True,
                             plot_kwargs={'model_name': 'PLSR with Savitzky-Golay Filter',
                                          'figsize': (8, 6)}
                             )
```

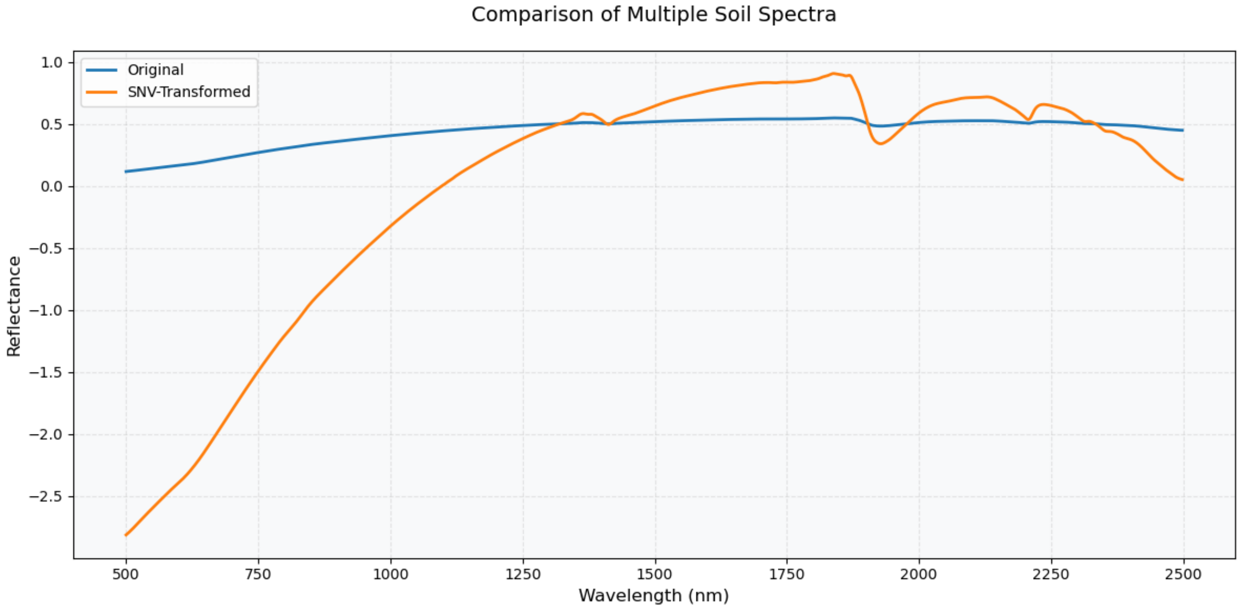
```
Root Mean Squared Error (RMSE): 10.1694
R²: 0.6473
Bias: -0.1207
RPD: 1.6838
```



4.1.2 Standard Normal Variate

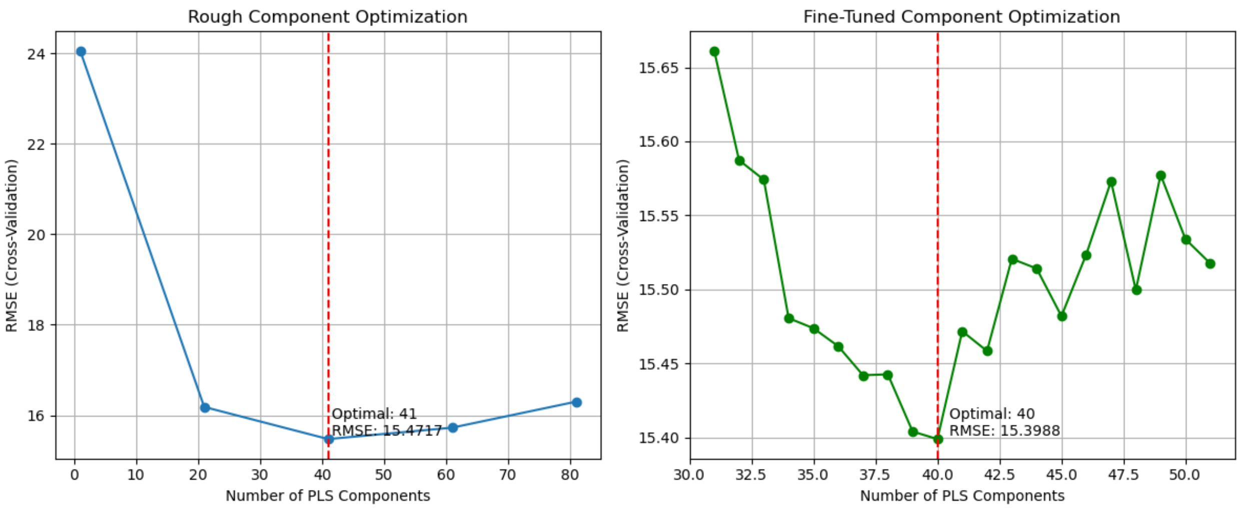
```
import own_functions
```

```
# Applying Savitzky-Golay filter to calibration and test data
X_train_snv = own_functions.standard_normal_variate(X_train)
X_test_snv = own_functions.standard_normal_variate(X_test)

# Example usage with multiple spectra:
plot_spectra_comparison(
    X_train[2],
    X_train_snv[2],
    wavelengths=range(500, 2500, 2),
    labels=['Original', 'SNV-Transformed'],
    title='Comparison of Multiple Soil Spectra'
)
```



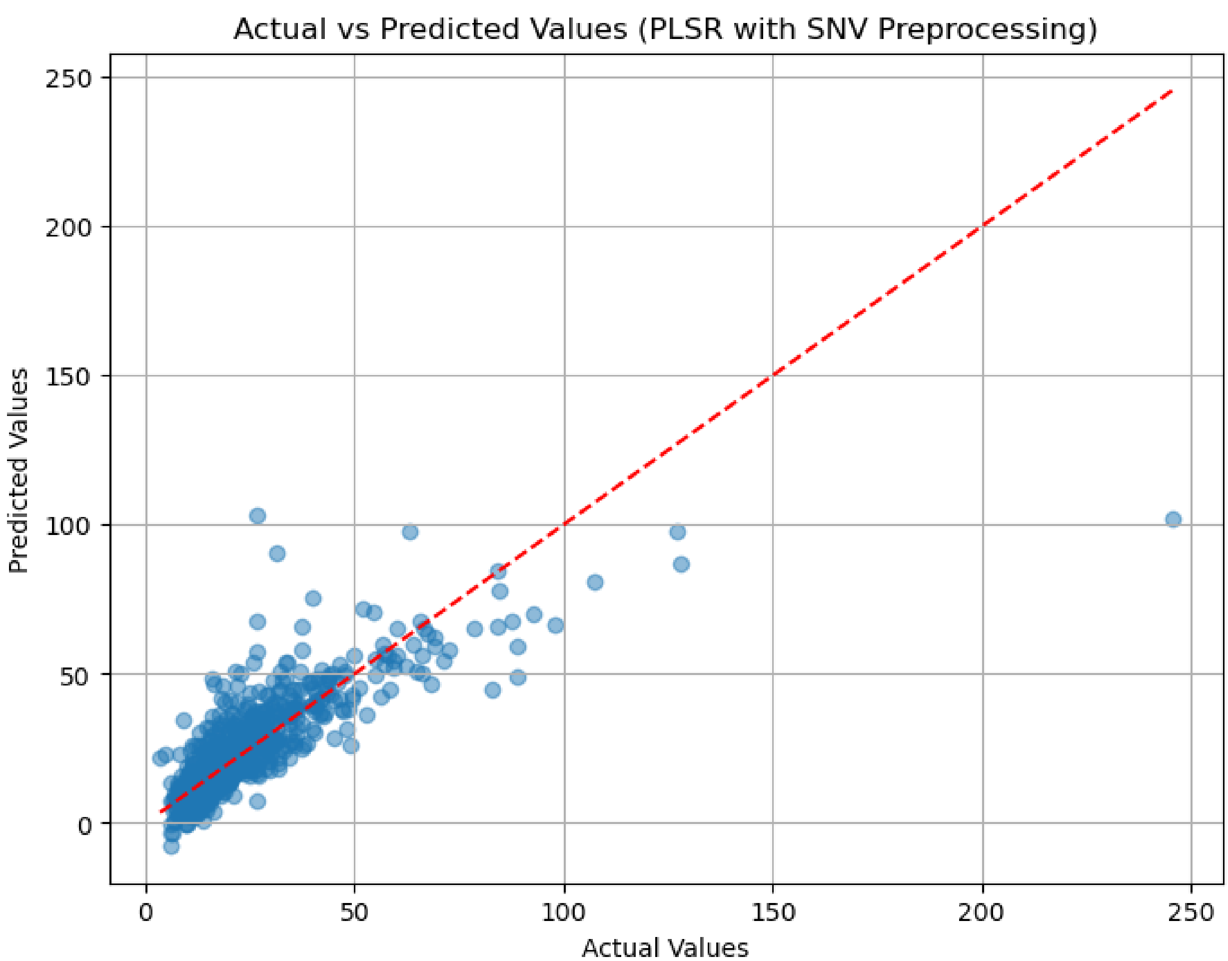

```
plsr_snv_components = own_functions.optimize_pls_components(X_train=X_train_snv,
                                                            y_train=y_train,
                                                            max_components=100,
                                                            step=20,
                                                            fine_tune=True,
                                                            show_progress=True,
                                                            plot_results=True
                                                            )
```

```
Rough Optimization:   0%|          | 0/5 [00:00<?, ?it/s]

Fine Tuning:   0%|          | 0/21 [00:00<?, ?it/s]
```



```python
plsr_snv_model = PLSRegression(n_components=plsr_snv_components["optimal_n"])
plsr_snv_model.fit(X_train_snv, y_train)

plsr_snv_eval = own_functions.evaluate_model(plsr_snv_model,
                             X_test=X_test_snv,
                             y_test=y_test,
                             print_metrics=True,
                             show_plot=True,
                             plot_kwargs={'model_name': 'PLSR with SNV Preprocessing',
                             'figsize': (8, 6)}
                             )
```

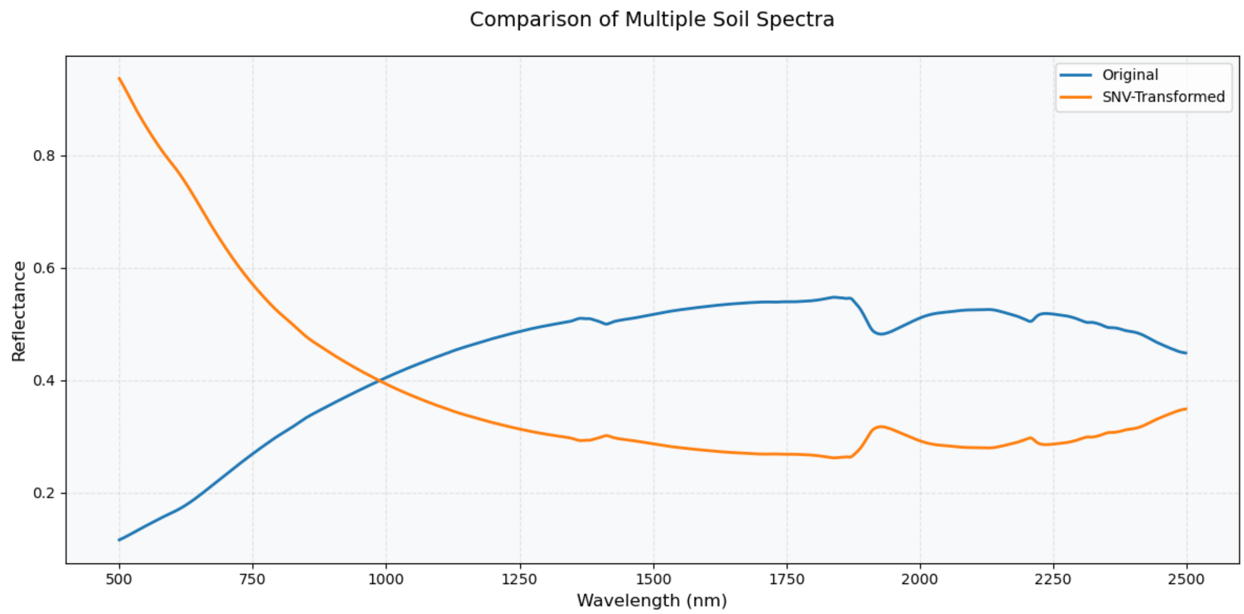
```
Root Mean Squared Error (RMSE): 10.1517
R²: 0.6485
Bias: 0.3009
RPD: 1.6867
```



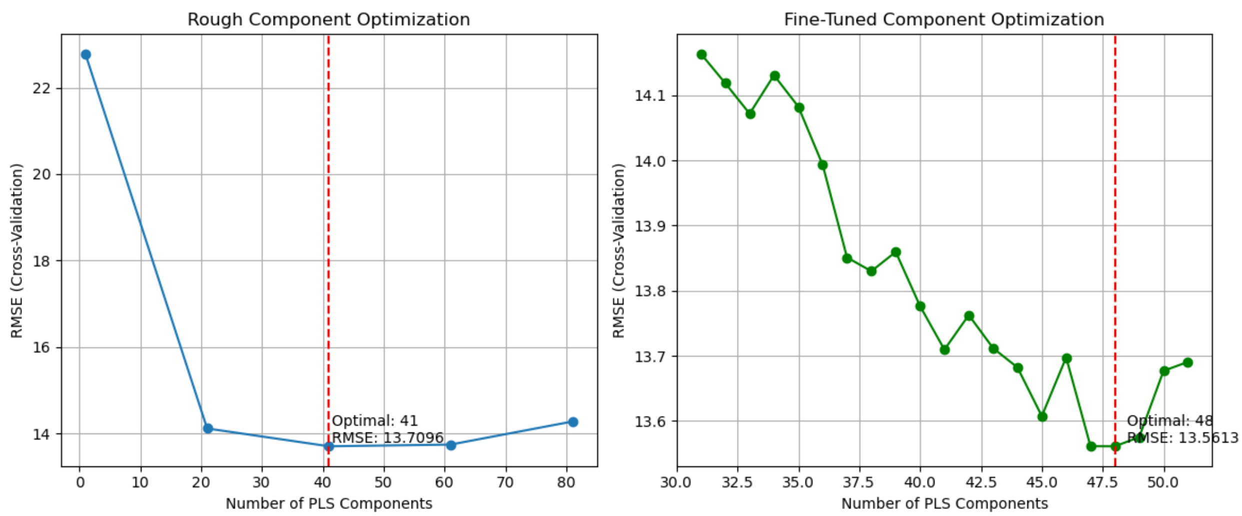
4.1.3 Absorbance

```
# Calculate pseudo abosrbance
X_train_absorb = np.log10(1/X_train)
X_test_absorb = np.log10(1/X_test)
```

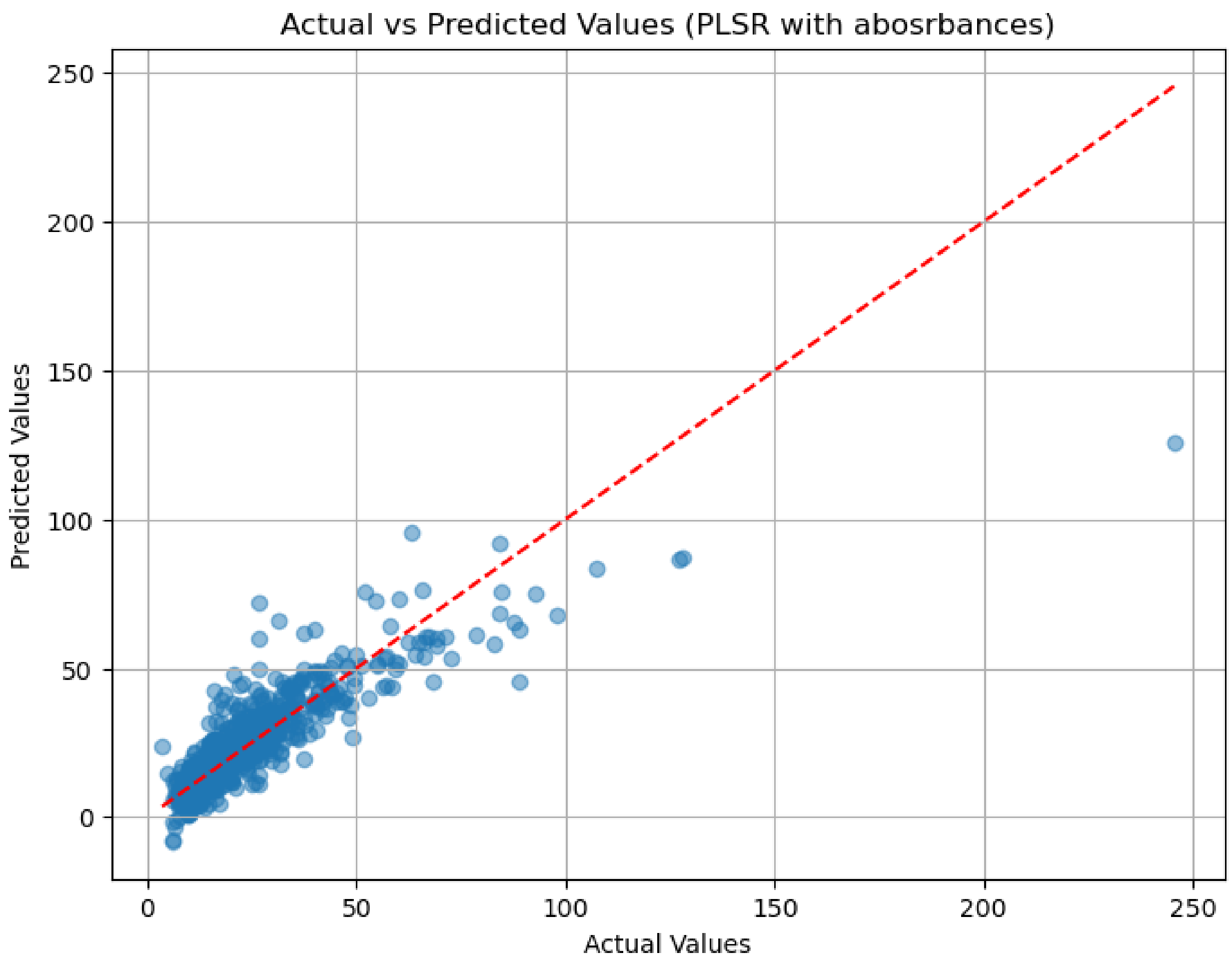
```
# Plot Spectra
plot_spectra_comparison(
    X_train[2],
    X_train_absorb[2],
    wavelengths=range(500, 2500, 2),
    labels=['Original', 'SNV-Transformed'],
    title='Comparison of Multiple Soil Spectra'
)

plsr_absorb_components = own_functions.optimize_pls_components(X_train=X_train_absorb,
                                      y_train=y_train,
                                      max_components=100,
                                      step=20,
                                      fine_tune=True,
                                      show_progress=True,
                                      plot_results=True
                                      )



plsr_absorb_model = PLSRegression(n_components=plsr_absorb_components["optimal_n"])
plsr_absorb_model.fit(X_train_absorb, y_train)

plsr_absorb_eval = own_functions.evaluate_model(plsr_absorb_model,
                             X_test=X_test_absorb,
                             y_test=y_test,
                             print_metrics=True,
                             show_plot=True,
                             plot_kwargs={'model_name': 'PLSR with abosrbances',
                             'figsize': (8, 6)}
                             )
```




```
Rough Optimization:   0%|          | 0/5 [00:00<?, ?it/s]

Fine Tuning:   0%|          | 0/21 [00:00<?, ?it/s]
```




```
Root Mean Squared Error (RMSE): 8.6069
R²: 0.7474
Bias: 0.1506
RPD: 1.9895
```



4.2 Testing Different Models

Our next strategy is to test different models. We will test the following models: - LSTM -

4.2.1 Pytorch LSTM

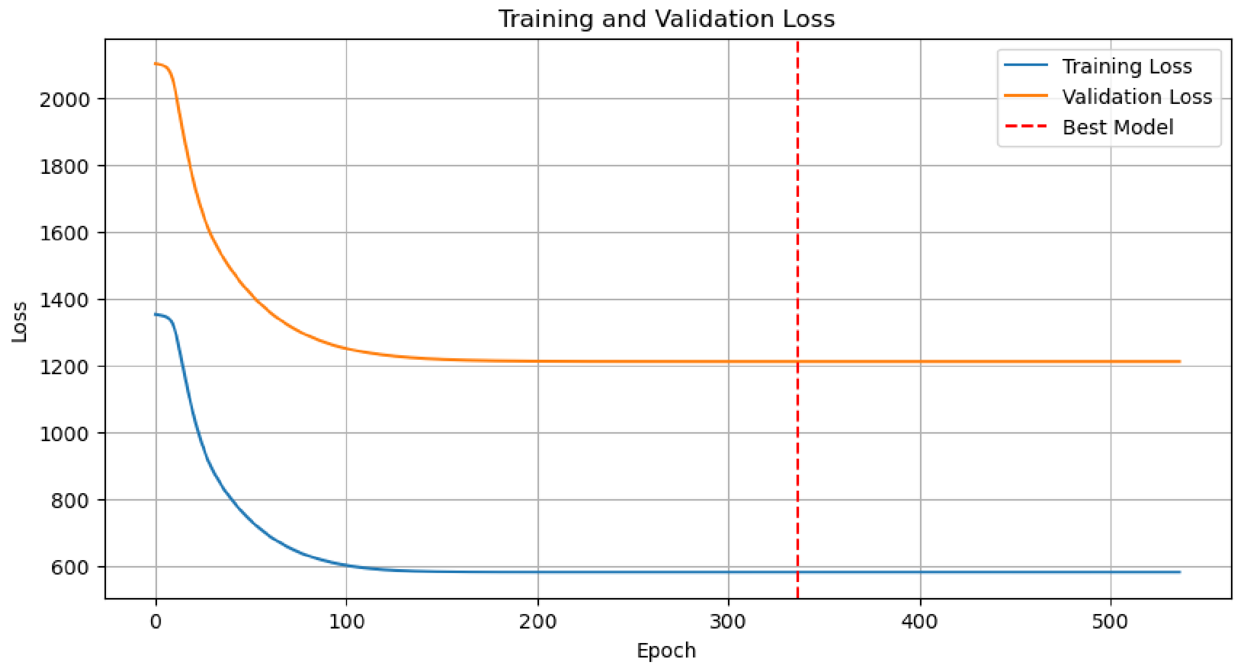
```
import own_functions
```

```
# drop rate 0.2 best
# First split training data into train and validation sets
X_train_final, X_val, y_train_final, y_val = train_test_split(X_train_absorb, y_train,
                                                              test_size=0.2,
                                                              random_state=110)

# Training enhanced LSTM model
LSTM_base_model, history, metrics = own_functions.train_and_evaluate_lstm(
```

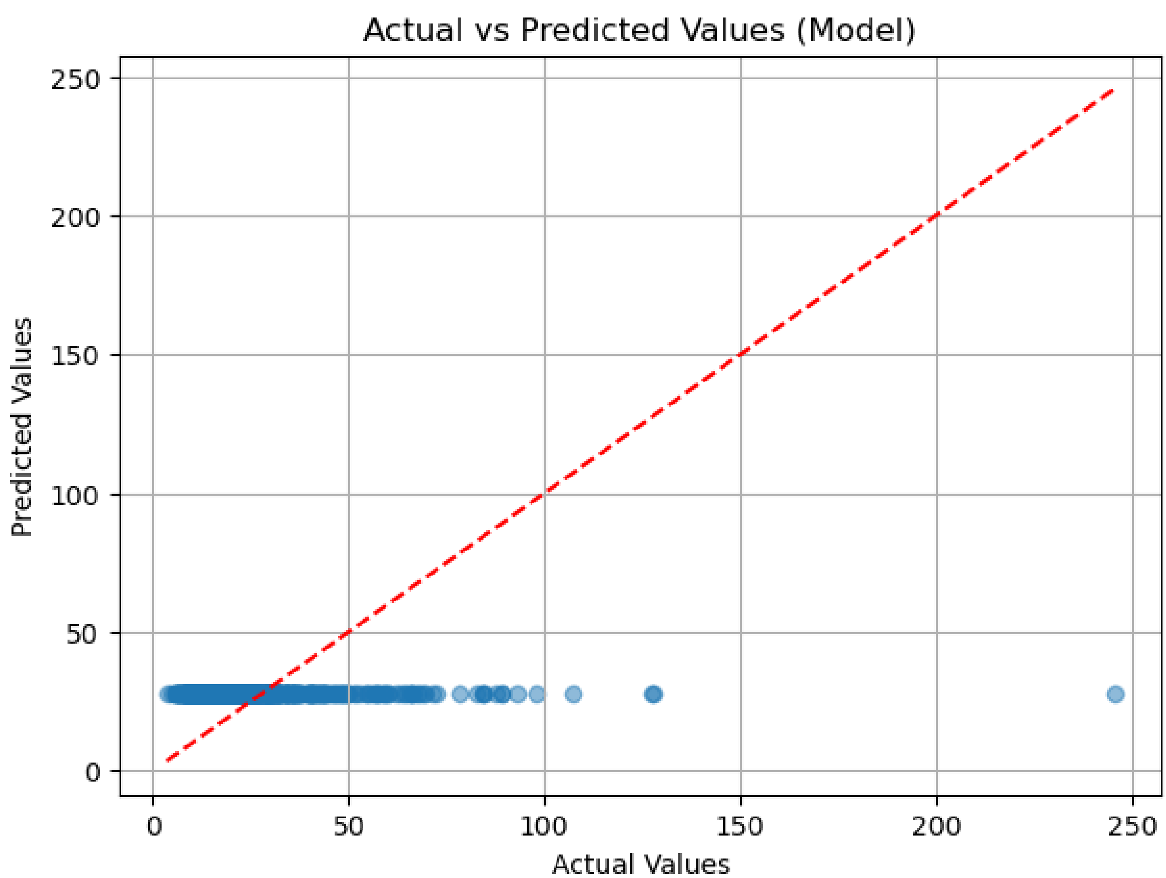
```python
    X_train=X_train_final,
    X_val=X_val,
    X_test=X_test,
    y_train=y_train_final,
    y_val=y_val,
    y_test=y_test,
    hidden_size=256,
    num_layers=5,
    num_epochs=3000,
    learning_rate=0.001,
    patience=200,  # Early stopping patience
    dropout=0.2,
)



# Evaluate LSTM model
_ = own_functions.evaluate_model(LSTM_base_model,
                                 X_test=X_test_absorb, y_test=y_test,
                                 print_metrics=True, show_plot=True)
```

```
Epoch [10/3000], Train Loss: 1323.3580, Val Loss: 2055.6138
Epoch [20/3000], Train Loss: 1073.8004, Val Loss: 1774.0663
Epoch [30/3000], Train Loss: 897.4761, Val Loss: 1591.5127
Epoch [40/3000], Train Loss: 804.3825, Val Loss: 1491.1821
Epoch [50/3000], Train Loss: 739.6238, Val Loss: 1418.3021
Epoch [60/3000], Train Loss: 691.6126, Val Loss: 1362.8917
Epoch [70/3000], Train Loss: 656.5456, Val Loss: 1321.1735
Epoch [80/3000], Train Loss: 631.4288, Val Loss: 1290.1431
Epoch [90/3000], Train Loss: 613.8032, Val Loss: 1267.3346
Epoch [100/3000], Train Loss: 601.7249, Val Loss: 1250.7620
Epoch [110/3000], Train Loss: 593.6379, Val Loss: 1238.8523
Epoch [120/3000], Train Loss: 588.3791, Val Loss: 1230.3783
Epoch [130/3000], Train Loss: 585.0749, Val Loss: 1224.3984
Epoch [140/3000], Train Loss: 583.0453, Val Loss: 1220.2046
Epoch [150/3000], Train Loss: 581.8547, Val Loss: 1217.2766
Epoch [160/3000], Train Loss: 581.1397, Val Loss: 1215.2374
Epoch [170/3000], Train Loss: 580.7720, Val Loss: 1213.8202
Epoch [180/3000], Train Loss: 580.5948, Val Loss: 1212.8375
Epoch [190/3000], Train Loss: 580.4708, Val Loss: 1212.1587
Epoch [200/3000], Train Loss: 580.4212, Val Loss: 1211.6929
Epoch [210/3000], Train Loss: 580.3871, Val Loss: 1211.3760
Epoch [220/3000], Train Loss: 580.3820, Val Loss: 1211.1633
Epoch [230/3000], Train Loss: 580.3714, Val Loss: 1211.0229
Epoch [240/3000], Train Loss: 580.3652, Val Loss: 1210.9321
Epoch [250/3000], Train Loss: 580.3819, Val Loss: 1210.8749
Epoch [260/3000], Train Loss: 580.3779, Val Loss: 1210.8401
```

```
Epoch [270/3000], Train Loss: 580.3584, Val Loss: 1210.8195
Epoch [280/3000], Train Loss: 580.3649, Val Loss: 1210.8080
Epoch [290/3000], Train Loss: 580.3868, Val Loss: 1210.8020
Epoch [300/3000], Train Loss: 580.3781, Val Loss: 1210.7992
Epoch [310/3000], Train Loss: 580.3660, Val Loss: 1210.7980
Epoch [320/3000], Train Loss: 580.3627, Val Loss: 1210.7976
Epoch [330/3000], Train Loss: 580.3510, Val Loss: 1210.7976
Epoch [340/3000], Train Loss: 580.3632, Val Loss: 1210.7977
Epoch [350/3000], Train Loss: 580.3787, Val Loss: 1210.7981
Epoch [360/3000], Train Loss: 580.3858, Val Loss: 1210.7982
Epoch [370/3000], Train Loss: 580.3724, Val Loss: 1210.7983
Epoch [380/3000], Train Loss: 580.3879, Val Loss: 1210.7983
Epoch [390/3000], Train Loss: 580.3510, Val Loss: 1210.7983
Epoch [400/3000], Train Loss: 580.3687, Val Loss: 1210.7983
Epoch [410/3000], Train Loss: 580.3755, Val Loss: 1210.7983
Epoch [420/3000], Train Loss: 580.3708, Val Loss: 1210.7982
Epoch [430/3000], Train Loss: 580.3740, Val Loss: 1210.7983
Epoch [440/3000], Train Loss: 580.3839, Val Loss: 1210.7985
Epoch [450/3000], Train Loss: 580.3721, Val Loss: 1210.7986
Epoch [460/3000], Train Loss: 580.3799, Val Loss: 1210.7988
Epoch [470/3000], Train Loss: 580.3810, Val Loss: 1210.7986
Epoch [480/3000], Train Loss: 580.3684, Val Loss: 1210.7986
Epoch [490/3000], Train Loss: 580.3710, Val Loss: 1210.7985
Epoch [500/3000], Train Loss: 580.3718, Val Loss: 1210.7985
Epoch [510/3000], Train Loss: 580.3718, Val Loss: 1210.7985
Epoch [520/3000], Train Loss: 580.3864, Val Loss: 1210.7986
Epoch [530/3000], Train Loss: 580.3820, Val Loss: 1210.7986
Early stopping triggered at epoch 537

Final Test Metrics:
test_loss: 325.0659
rmse: 18.0296
r2: -0.1087
bias: 5.6444
rpd: 0.9497
Root Mean Squared Error (RMSE): 18.0296
R²: -0.1087
Bias: 5.6444
RPD: 0.9497
```



Training and Validation Loss
Training Loss
Validation Loss
Best Model
Loss
Epoch



Actual vs Predicted Values (Model)
Predicted Values
Actual Values

4.2.2 LSTM with PLSR components

```
# Fit PLSR on the training data and transform the training set
X_train_pls, _ = plsr_absorb_model.transform(X_train_absorb, y_train)

# Transform the test set using the fitted model (trained on the training set)
X_test_pls = plsr_absorb_model.transform(X_test_absorb)
```

```
# First split training data into train and validation sets
X_train_final, X_val, y_train_final, y_val = train_test_split(X_train_pls, y_train,
                                                              test_size=0.2,
                                                              random_state=42)

# Training enhanced LSTM model
LSTM_plsr_model, history, metrics = own_functions.train_and_evaluate_lstm(
    X_train=X_train_final,
    X_val=X_val,
    X_test=X_test_pls,
    y_train=y_train_final,
    y_val=y_val,
    y_test=y_test,
    hidden_size=256,
    num_layers=5,
    num_epochs=3000,
    learning_rate=0.001,
    patience=200,  # Early stopping patience
    dropout=0.2
)


lstm_plsr_eval = own_functions.evaluate_model(LSTM_plsr_model,
                             X_test=X_test_pls, y_test=y_test,
                             print_metrics=True, show_plot=True)
```

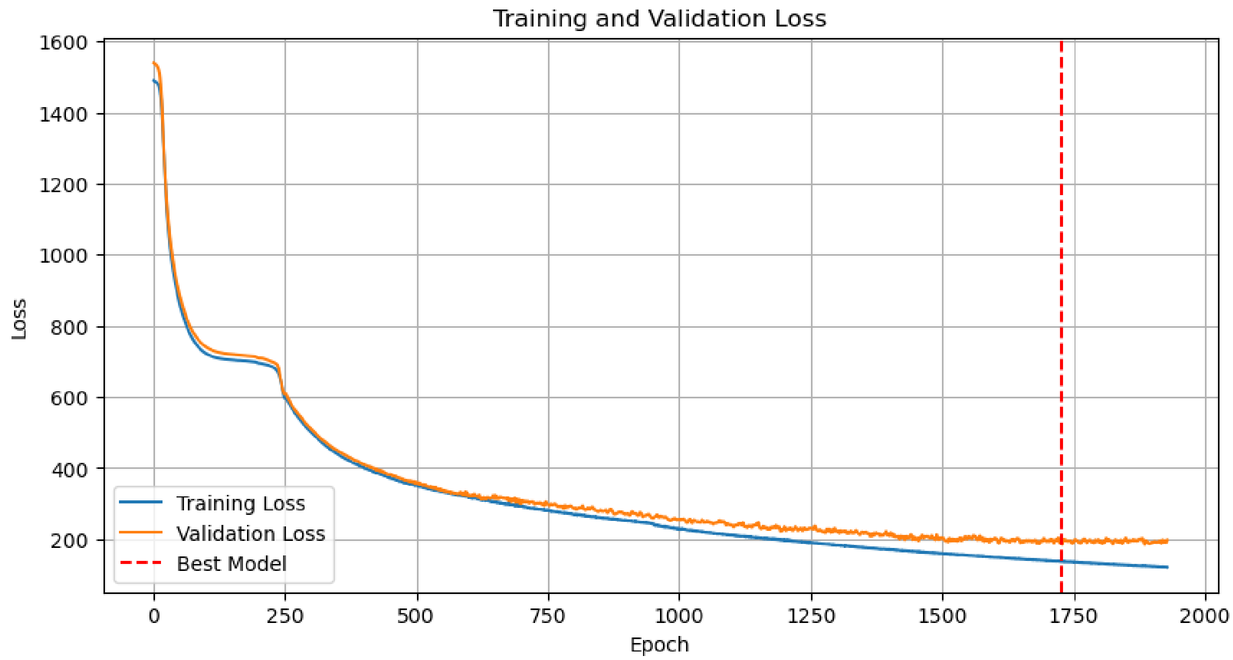
```
Epoch [10/3000], Train Loss: 1476.9949, Val Loss: 1522.3971
Epoch [20/3000], Train Loss: 1298.9865, Val Loss: 1303.0087
Epoch [30/3000], Train Loss: 1055.8999, Val Loss: 1075.3479
Epoch [40/3000], Train Loss: 936.0597, Val Loss: 960.6909
Epoch [50/3000], Train Loss: 864.6968, Val Loss: 888.7760
Epoch [60/3000], Train Loss: 813.6223, Val Loss: 836.5993
Epoch [70/3000], Train Loss: 777.1158, Val Loss: 798.9126
Epoch [80/3000], Train Loss: 751.5482, Val Loss: 772.2037
Epoch [90/3000], Train Loss: 734.0804, Val Loss: 753.6362
Epoch [100/3000], Train Loss: 722.4227, Val Loss: 740.9803
Epoch [110/3000], Train Loss: 714.8896, Val Loss: 732.5262
Epoch [120/3000], Train Loss: 710.1119, Val Loss: 726.9906
```

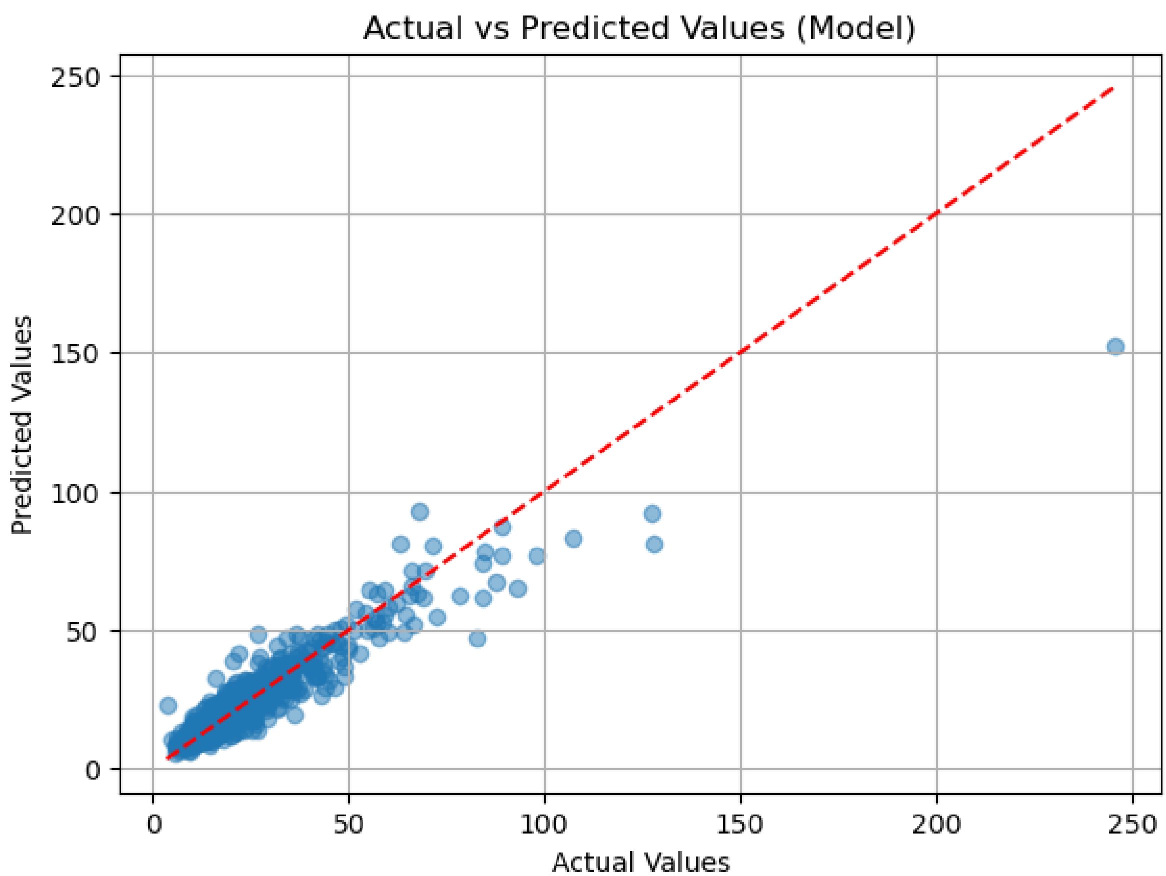
```
Epoch [130/3000], Train Loss: 707.2158, Val Loss: 723.4321
Epoch [140/3000], Train Loss: 705.5001, Val Loss: 721.1772
Epoch [150/3000], Train Loss: 704.2849, Val Loss: 719.7388
Epoch [160/3000], Train Loss: 702.4864, Val Loss: 717.9755
Epoch [170/3000], Train Loss: 701.9086, Val Loss: 716.7893
Epoch [180/3000], Train Loss: 700.6622, Val Loss: 715.4919
Epoch [190/3000], Train Loss: 698.9515, Val Loss: 714.5274
Epoch [200/3000], Train Loss: 695.2794, Val Loss: 709.4814
Epoch [210/3000], Train Loss: 691.9188, Val Loss: 707.7747
Epoch [220/3000], Train Loss: 688.4955, Val Loss: 702.2612
Epoch [230/3000], Train Loss: 681.7617, Val Loss: 696.5450
Epoch [240/3000], Train Loss: 660.8052, Val Loss: 670.4714
Epoch [250/3000], Train Loss: 598.0337, Val Loss: 608.7620
Epoch [260/3000], Train Loss: 578.2463, Val Loss: 587.5004
Epoch [270/3000], Train Loss: 554.2938, Val Loss: 563.7514
Epoch [280/3000], Train Loss: 534.7710, Val Loss: 544.4777
Epoch [290/3000], Train Loss: 516.8218, Val Loss: 526.2027
Epoch [300/3000], Train Loss: 501.3236, Val Loss: 509.9671
Epoch [310/3000], Train Loss: 486.6131, Val Loss: 495.2682
Epoch [320/3000], Train Loss: 473.3205, Val Loss: 481.4204
Epoch [330/3000], Train Loss: 460.4038, Val Loss: 469.4686
Epoch [340/3000], Train Loss: 449.8602, Val Loss: 459.4424
Epoch [350/3000], Train Loss: 439.8952, Val Loss: 449.1520
Epoch [360/3000], Train Loss: 430.5488, Val Loss: 440.9105
Epoch [370/3000], Train Loss: 424.1036, Val Loss: 432.1590
Epoch [380/3000], Train Loss: 414.9257, Val Loss: 424.5315
Epoch [390/3000], Train Loss: 407.8018, Val Loss: 418.7832
Epoch [400/3000], Train Loss: 402.1411, Val Loss: 410.3160
Epoch [410/3000], Train Loss: 394.8529, Val Loss: 405.0167
Epoch [420/3000], Train Loss: 388.8846, Val Loss: 398.1512
Epoch [430/3000], Train Loss: 384.2500, Val Loss: 394.7275
Epoch [440/3000], Train Loss: 378.3047, Val Loss: 386.6680
Epoch [450/3000], Train Loss: 371.8717, Val Loss: 381.3099
Epoch [460/3000], Train Loss: 367.6703, Val Loss: 377.8934
Epoch [470/3000], Train Loss: 363.1318, Val Loss: 371.2940
Epoch [480/3000], Train Loss: 358.1399, Val Loss: 365.4178
Epoch [490/3000], Train Loss: 355.3999, Val Loss: 362.7246
Epoch [500/3000], Train Loss: 351.4532, Val Loss: 356.0226
Epoch [510/3000], Train Loss: 347.9709, Val Loss: 354.5867
Epoch [520/3000], Train Loss: 342.6750, Val Loss: 349.2525
Epoch [530/3000], Train Loss: 339.2817, Val Loss: 344.0117
Epoch [540/3000], Train Loss: 336.5583, Val Loss: 341.4745
Epoch [550/3000], Train Loss: 334.2585, Val Loss: 336.6165
Epoch [560/3000], Train Loss: 330.9122, Val Loss: 333.5845
Epoch [570/3000], Train Loss: 327.8176, Val Loss: 330.5784
Epoch [580/3000], Train Loss: 323.5317, Val Loss: 326.8005
Epoch [590/3000], Train Loss: 321.0453, Val Loss: 329.3196
Epoch [600/3000], Train Loss: 316.9848, Val Loss: 321.2742
```

```
Epoch [610/3000], Train Loss: 315.4804, Val Loss: 325.0527
Epoch [620/3000], Train Loss: 311.8969, Val Loss: 324.5777
Epoch [630/3000], Train Loss: 310.9660, Val Loss: 315.0254
Epoch [640/3000], Train Loss: 306.8080, Val Loss: 314.6121
Epoch [650/3000], Train Loss: 305.7915, Val Loss: 317.6821
Epoch [660/3000], Train Loss: 300.9412, Val Loss: 318.9722
Epoch [670/3000], Train Loss: 299.0072, Val Loss: 315.4145
Epoch [680/3000], Train Loss: 297.4985, Val Loss: 309.6865
Epoch [690/3000], Train Loss: 294.1353, Val Loss: 303.2952
Epoch [700/3000], Train Loss: 293.6654, Val Loss: 299.8304
Epoch [710/3000], Train Loss: 289.3485, Val Loss: 303.7017
Epoch [720/3000], Train Loss: 286.6858, Val Loss: 304.7412
Epoch [730/3000], Train Loss: 285.0848, Val Loss: 300.4618
Epoch [740/3000], Train Loss: 282.4811, Val Loss: 297.2863
Epoch [750/3000], Train Loss: 281.5024, Val Loss: 298.2256
Epoch [760/3000], Train Loss: 278.1667, Val Loss: 296.1386
Epoch [770/3000], Train Loss: 276.5258, Val Loss: 290.8620
Epoch [780/3000], Train Loss: 273.9065, Val Loss: 288.4438
Epoch [790/3000], Train Loss: 272.6160, Val Loss: 290.3289
Epoch [800/3000], Train Loss: 269.4107, Val Loss: 286.7017
Epoch [810/3000], Train Loss: 268.6029, Val Loss: 288.5892
Epoch [820/3000], Train Loss: 266.9337, Val Loss: 283.9859
Epoch [830/3000], Train Loss: 264.2774, Val Loss: 284.5284
Epoch [840/3000], Train Loss: 262.4578, Val Loss: 281.5837
Epoch [850/3000], Train Loss: 260.8549, Val Loss: 281.2022
Epoch [860/3000], Train Loss: 258.2273, Val Loss: 281.7904
Epoch [870/3000], Train Loss: 256.9647, Val Loss: 284.5422
Epoch [880/3000], Train Loss: 256.4256, Val Loss: 270.8945
Epoch [890/3000], Train Loss: 254.4118, Val Loss: 275.8855
Epoch [900/3000], Train Loss: 252.9502, Val Loss: 270.3586
Epoch [910/3000], Train Loss: 251.3546, Val Loss: 277.0125
Epoch [920/3000], Train Loss: 249.6179, Val Loss: 269.4876
Epoch [930/3000], Train Loss: 247.2080, Val Loss: 265.9085
Epoch [940/3000], Train Loss: 245.9761, Val Loss: 268.9632
Epoch [950/3000], Train Loss: 242.8265, Val Loss: 259.3888
Epoch [960/3000], Train Loss: 237.5843, Val Loss: 266.6066
Epoch [970/3000], Train Loss: 236.2847, Val Loss: 262.0168
Epoch [980/3000], Train Loss: 234.7090, Val Loss: 255.5173
Epoch [990/3000], Train Loss: 232.8326, Val Loss: 257.2687
Epoch [1000/3000], Train Loss: 228.2887, Val Loss: 255.8069
Epoch [1010/3000], Train Loss: 227.7953, Val Loss: 254.3607
Epoch [1020/3000], Train Loss: 225.2851, Val Loss: 253.6589
Epoch [1030/3000], Train Loss: 223.1078, Val Loss: 253.0728
Epoch [1040/3000], Train Loss: 221.7785, Val Loss: 247.4736
Epoch [1050/3000], Train Loss: 218.9722, Val Loss: 247.3413
Epoch [1060/3000], Train Loss: 217.5639, Val Loss: 246.0600
Epoch [1070/3000], Train Loss: 216.1585, Val Loss: 246.0295
Epoch [1080/3000], Train Loss: 214.4270, Val Loss: 247.4672
```

```
Epoch [1090/3000], Train Loss: 212.3486, Val Loss: 239.4285
Epoch [1100/3000], Train Loss: 212.0689, Val Loss: 239.6877
Epoch [1110/3000], Train Loss: 210.2218, Val Loss: 239.6194
Epoch [1120/3000], Train Loss: 208.2352, Val Loss: 239.9968
Epoch [1130/3000], Train Loss: 206.7302, Val Loss: 232.5133
Epoch [1140/3000], Train Loss: 205.3598, Val Loss: 234.7560
Epoch [1150/3000], Train Loss: 204.2509, Val Loss: 243.5592
Epoch [1160/3000], Train Loss: 202.5614, Val Loss: 236.1106
Epoch [1170/3000], Train Loss: 201.1573, Val Loss: 230.2865
Epoch [1180/3000], Train Loss: 199.5571, Val Loss: 234.5081
Epoch [1190/3000], Train Loss: 198.1268, Val Loss: 234.0108
Epoch [1200/3000], Train Loss: 198.4915, Val Loss: 231.3544
Epoch [1210/3000], Train Loss: 197.1207, Val Loss: 227.4995
Epoch [1220/3000], Train Loss: 194.4886, Val Loss: 231.3234
Epoch [1230/3000], Train Loss: 192.6752, Val Loss: 227.8806
Epoch [1240/3000], Train Loss: 191.1943, Val Loss: 229.5043
Epoch [1250/3000], Train Loss: 191.0037, Val Loss: 230.5132
Epoch [1260/3000], Train Loss: 188.4146, Val Loss: 234.8319
Epoch [1270/3000], Train Loss: 187.5409, Val Loss: 225.0884
Epoch [1280/3000], Train Loss: 186.0923, Val Loss: 222.8781
Epoch [1290/3000], Train Loss: 185.1309, Val Loss: 219.6052
Epoch [1300/3000], Train Loss: 183.4380, Val Loss: 222.1329
Epoch [1310/3000], Train Loss: 182.3828, Val Loss: 219.8340
Epoch [1320/3000], Train Loss: 180.5748, Val Loss: 216.0920
Epoch [1330/3000], Train Loss: 179.1602, Val Loss: 216.9297
Epoch [1340/3000], Train Loss: 178.0300, Val Loss: 218.6974
Epoch [1350/3000], Train Loss: 176.5683, Val Loss: 218.4111
Epoch [1360/3000], Train Loss: 175.3150, Val Loss: 215.6333
Epoch [1370/3000], Train Loss: 174.4945, Val Loss: 216.0143
Epoch [1380/3000], Train Loss: 172.7871, Val Loss: 215.2256
Epoch [1390/3000], Train Loss: 172.0466, Val Loss: 211.7317
Epoch [1400/3000], Train Loss: 172.1469, Val Loss: 213.9451
Epoch [1410/3000], Train Loss: 170.6136, Val Loss: 210.3991
Epoch [1420/3000], Train Loss: 167.8427, Val Loss: 199.5307
Epoch [1430/3000], Train Loss: 167.3057, Val Loss: 197.9110
Epoch [1440/3000], Train Loss: 165.9662, Val Loss: 210.5001
Epoch [1450/3000], Train Loss: 165.1458, Val Loss: 198.7785
Epoch [1460/3000], Train Loss: 163.8191, Val Loss: 203.0324
Epoch [1470/3000], Train Loss: 162.2490, Val Loss: 206.8297
Epoch [1480/3000], Train Loss: 161.2955, Val Loss: 207.0345
Epoch [1490/3000], Train Loss: 160.9986, Val Loss: 212.2282
Epoch [1500/3000], Train Loss: 159.8334, Val Loss: 202.5370
Epoch [1510/3000], Train Loss: 158.5091, Val Loss: 194.3930
Epoch [1520/3000], Train Loss: 157.5268, Val Loss: 199.4212
Epoch [1530/3000], Train Loss: 156.2196, Val Loss: 198.2691
Epoch [1540/3000], Train Loss: 155.6915, Val Loss: 200.3884
Epoch [1550/3000], Train Loss: 154.6369, Val Loss: 205.7376
Epoch [1560/3000], Train Loss: 153.5350, Val Loss: 209.0687
```

```
Epoch [1570/3000], Train Loss: 153.0084, Val Loss: 200.0821
Epoch [1580/3000], Train Loss: 151.8959, Val Loss: 196.4576
Epoch [1590/3000], Train Loss: 150.2973, Val Loss: 191.6272
Epoch [1600/3000], Train Loss: 148.9880, Val Loss: 200.3698
Epoch [1610/3000], Train Loss: 148.7325, Val Loss: 191.7266
Epoch [1620/3000], Train Loss: 147.2845, Val Loss: 198.2524
Epoch [1630/3000], Train Loss: 146.3434, Val Loss: 191.2391
Epoch [1640/3000], Train Loss: 146.0799, Val Loss: 194.3738
Epoch [1650/3000], Train Loss: 143.7209, Val Loss: 189.7874
Epoch [1660/3000], Train Loss: 143.1698, Val Loss: 195.7730
Epoch [1670/3000], Train Loss: 142.2986, Val Loss: 193.4411
Epoch [1680/3000], Train Loss: 141.5167, Val Loss: 190.4524
Epoch [1690/3000], Train Loss: 140.6909, Val Loss: 194.6500
Epoch [1700/3000], Train Loss: 140.5246, Val Loss: 192.9967
Epoch [1710/3000], Train Loss: 139.0218, Val Loss: 196.3853
Epoch [1720/3000], Train Loss: 138.3493, Val Loss: 196.6891
Epoch [1730/3000], Train Loss: 137.0433, Val Loss: 186.0666
Epoch [1740/3000], Train Loss: 136.0545, Val Loss: 193.9672
Epoch [1750/3000], Train Loss: 134.9852, Val Loss: 189.6192
Epoch [1760/3000], Train Loss: 134.6837, Val Loss: 192.6896
Epoch [1770/3000], Train Loss: 134.0189, Val Loss: 197.8960
Epoch [1780/3000], Train Loss: 132.7158, Val Loss: 194.3234
Epoch [1790/3000], Train Loss: 132.5679, Val Loss: 189.8598
Epoch [1800/3000], Train Loss: 131.0430, Val Loss: 191.5202
Epoch [1810/3000], Train Loss: 130.7209, Val Loss: 190.3528
Epoch [1820/3000], Train Loss: 129.6214, Val Loss: 195.6850
Epoch [1830/3000], Train Loss: 129.0399, Val Loss: 197.5882
Epoch [1840/3000], Train Loss: 128.0919, Val Loss: 195.5797
Epoch [1850/3000], Train Loss: 127.4352, Val Loss: 203.0794
Epoch [1860/3000], Train Loss: 126.4132, Val Loss: 194.0215
Epoch [1870/3000], Train Loss: 125.5426, Val Loss: 197.1165
Epoch [1880/3000], Train Loss: 124.8665, Val Loss: 197.0638
Epoch [1890/3000], Train Loss: 125.1311, Val Loss: 194.8841
Epoch [1900/3000], Train Loss: 123.3983, Val Loss: 191.0178
Epoch [1910/3000], Train Loss: 122.7461, Val Loss: 190.8943
Epoch [1920/3000], Train Loss: 121.6788, Val Loss: 197.3927
Early stopping triggered at epoch 1929

Final Test Metrics:
test_loss: 39.1753
rmse: 6.2590
r2: 0.8664
bias: -0.4263
rpd: 2.7358
Root Mean Squared Error (RMSE): 6.2590
R²: 0.8664
Bias: -0.4263
RPD: 2.7358
```






```
{'y_pred': array([ 17.554638 ,  29.943878 ,  71.74279  ,  23.583864 ,  24.661398 ,
```

13.375087 ,	22.957867 ,	25.22774 ,	36.47243 ,	28.22427 ,
11.682562 ,	21.11962 ,	39.35656 ,	16.566448 ,	20.577812 ,
9.545267 ,	19.650919 ,	10.926707 ,	27.827528 ,	58.33704 ,
16.395775 ,	19.8949 ,	10.420736 ,	53.958225 ,	17.665817 ,
40.262455 ,	52.366566 ,	36.291523 ,	20.532324 ,	14.822706 ,
30.73347 ,	25.535364 ,	20.031012 ,	39.080708 ,	30.449284 ,
15.038757 ,	17.568533 ,	23.669764 ,	18.42095 ,	37.460674 ,
10.315511 ,	15.06505 ,	12.580948 ,	32.837208 ,	31.371553 ,
28.902525 ,	37.408573 ,	45.96419 ,	44.325974 ,	26.090849 ,
18.375883 ,	9.875319 ,	12.033817 ,	10.879965 ,	10.189907 ,
11.400271 ,	11.134239 ,	19.783934 ,	8.710592 ,	11.687713 ,
8.604458 ,	10.284488 ,	11.112972 ,	14.005805 ,	15.137327 ,
18.503912 ,	13.429158 ,	12.053981 ,	12.552679 ,	10.397431 ,
16.508902 ,	17.195803 ,	9.216241 ,	10.1185055,	12.83384 ,
14.335113 ,	10.824083 ,	10.93681 ,	15.736313 ,	7.608075 ,
10.981428 ,	9.5582075,	10.292574 ,	7.6440597,	10.698574 ,
10.222986 ,	11.320516 ,	12.572524 ,	8.9324875,	12.4757 ,
10.229223 ,	16.543604 ,	14.136604 ,	11.071324 ,	11.906612 ,
9.173683 ,	10.6545315,	10.506189 ,	12.5704975,	12.399248 ,
11.487261 ,	33.5427 ,	11.026167 ,	10.569982 ,	48.299812 ,
15.381518 ,	12.13521 ,	18.857172 ,	21.880472 ,	31.987614 ,
11.742801 ,	12.709553 ,	8.544644 ,	10.442242 ,	11.579955 ,
10.8245325,	14.190771 ,	15.264399 ,	15.534687 ,	16.378632 ,
10.803865 ,	9.583427 ,	12.458605 ,	21.386885 ,	11.18239 ,
12.707585 ,	18.693401 ,	11.9905405,	11.635759 ,	15.645243 ,
11.056013 ,	47.013706 ,	15.150961 ,	17.052282 ,	21.95562 ,
15.104683 ,	12.289349 ,	30.102922 ,	30.553982 ,	24.209044 ,
30.715551 ,	36.496754 ,	16.370739 ,	18.293 ,	16.95048 ,
12.857683 ,	14.953086 ,	37.493343 ,	64.37718 ,	17.215588 ,
12.640993 ,	11.893375 ,	13.27306 ,	23.252092 ,	19.315153 ,
14.639355 ,	18.309126 ,	14.710827 ,	13.168448 ,	7.846001 ,
19.062853 ,	15.009278 ,	26.142706 ,	10.631445 ,	14.892431 ,
16.734228 ,	15.113861 ,	10.347481 ,	19.891462 ,	23.364042 ,
22.008745 ,	30.212616 ,	12.188459 ,	15.71101 ,	14.587362 ,
10.9464245,	20.029984 ,	13.889877 ,	12.794987 ,	15.16483 ,
21.759317 ,	13.347634 ,	20.051626 ,	14.428764 ,	25.545334 ,
14.530683 ,	52.60438 ,	20.40062 ,	30.760454 ,	26.297249 ,
23.192492 ,	11.562854 ,	17.473284 ,	14.097467 ,	12.57994 ,
41.60732 ,	13.437208 ,	11.995081 ,	18.177986 ,	40.939686 ,
17.423307 ,	24.534693 ,	50.31759 ,	22.829836 ,	8.195394 ,
59.37018 ,	26.609896 ,	20.05577 ,	21.875074 ,	28.734324 ,
9.545101 ,	17.12656 ,	23.867458 ,	16.652176 ,	17.691694 ,
12.332241 ,	15.178169 ,	16.907394 ,	9.076436 ,	9.877533 ,
7.9122987,	17.489838 ,	48.518326 ,	12.094136 ,	13.5869255,
13.763831 ,	21.200626 ,	17.836382 ,	87.35691 ,	5.90522 ,
17.23174 ,	22.441496 ,	16.638132 ,	38.159298 ,	22.931143 ,
18.656843 ,	15.071286 ,	22.180447 ,	16.71974 ,	22.554155 ,
61.946995 ,	37.05669 ,	29.069979 ,	13.935871 ,	14.201675 ,

13.892534 ,	12.850227 ,	11.956507 ,	12.276686 ,	18.718948 ,
42.557728 ,	71.08807 ,	11.561662 ,	14.070908 ,	11.60595 ,
25.278719 ,	34.58436 ,	55.678577 ,	18.055357 ,	29.36125 ,
15.464841 ,	21.306545 ,	32.019 ,	37.875145 ,	26.188683 ,
16.820736 ,	19.33336 ,	15.115097 ,	22.25376 ,	15.745544 ,
23.427172 ,	19.380209 ,	14.033683 ,	22.273777 ,	34.195057 ,
18.268261 ,	23.87647 ,	66.10261 ,	8.18068 ,	22.135567 ,
35.70045 ,	34.375103 ,	22.009825 ,	10.3081255,	64.56007 ,
11.582939 ,	33.773964 ,	27.376707 ,	22.735174 ,	32.311417 ,
18.406734 ,	11.532261 ,	9.946317 ,	21.073862 ,	7.352037 ,
9.92745 ,	11.493267 ,	12.870199 ,	14.933127 ,	32.01177 ,
45.283333 ,	73.98525 ,	38.42441 ,	12.110559 ,	49.059174 ,
81.09719 ,	83.09725 ,	11.515052 ,	10.613716 ,	9.425875 ,
15.053785 ,	11.198204 ,	14.109471 ,	16.43013 ,	13.268848 ,
14.880234 ,	11.496066 ,	12.539433 ,	10.828272 ,	11.364031 ,
22.299456 ,	57.5303 ,	24.294947 ,	30.799166 ,	24.453077 ,
36.401577 ,	33.175556 ,	17.038424 ,	54.710865 ,	13.620868 ,
13.344478 ,	10.450514 ,	16.484327 ,	12.338276 ,	12.501667 ,
11.411373 ,	10.65053 ,	10.23892 ,	13.390879 ,	13.287432 ,
78.31412 ,	52.026382 ,	25.679058 ,	20.801281 ,	24.685266 ,
19.60126 ,	15.513273 ,	10.469099 ,	29.268675 ,	24.840525 ,
12.52013 ,	13.321831 ,	15.013981 ,	12.054942 ,	15.535204 ,
67.592415 ,	30.433876 ,	9.97851 ,	10.85454 ,	12.518485 ,
29.264061 ,	18.526619 ,	10.472612 ,	21.11899 ,	9.604242 ,
12.317045 ,	13.3410425,	12.705067 ,	23.907108 ,	12.348957 ,
21.561155 ,	28.850647 ,	20.65027 ,	13.382404 ,	34.131584 ,
32.803577 ,	13.780179 ,	21.297504 ,	29.361448 ,	10.924696 ,
24.954939 ,	13.089909 ,	11.966734 ,	21.530884 ,	18.162638 ,
17.628157 ,	14.240722 ,	13.82554 ,	76.70651 ,	20.598488 ,
21.106672 ,	23.789173 ,	14.234191 ,	16.17023 ,	13.023797 ,
21.559175 ,	23.04782 ,	25.382774 ,	23.426039 ,	26.01671 ,
25.898333 ,	19.936064 ,	42.268925 ,	16.215971 ,	22.49438 ,
28.955221 ,	15.256432 ,	11.647049 ,	33.23688 ,	30.278831 ,
19.921213 ,	25.314432 ,	23.099176 ,	14.569479 ,	20.45313 ,
23.968582 ,	30.724693 ,	37.12682 ,	17.888708 ,	27.445473 ,
31.41481 ,	28.32174 ,	20.045897 ,	21.893892 ,	31.875418 ,
30.181309 ,	9.173586 ,	14.383961 ,	12.933123 ,	12.062146 ,
8.49432 ,	13.265076 ,	15.042267 ,	14.023979 ,	12.988524 ,
11.633797 ,	14.353547 ,	13.184746 ,	12.244453 ,	11.846791 ,
8.773142 ,	10.889131 ,	15.910397 ,	12.316376 ,	11.074291 ,
14.244022 ,	10.340889 ,	10.828012 ,	11.9963455,	11.042196 ,
12.527097 ,	11.771417 ,	13.962059 ,	12.521383 ,	10.128203 ,
14.295046 ,	16.339293 ,	15.980588 ,	14.465033 ,	14.183508 ,
10.731536 ,	13.665385 ,	10.060436 ,	15.951105 ,	11.781576 ,
14.601776 ,	15.322576 ,	12.151108 ,	14.645617 ,	17.314589 ,
16.120483 ,	12.621321 ,	13.469178 ,	12.316679 ,	13.308516 ,
11.669124 ,	20.7637 ,	19.29775 ,	44.364613 ,	17.892675 ,
31.329645 ,	14.989101 ,	27.39985 ,	36.21284 ,	50.515804 ,

48.512833 , 80.331604 , 31.363312 , 23.124382 , 39.800762 ,
46.979366 , 40.606724 , 11.189964 , 46.88944 , 13.47703 ,
13.881943 , 27.77301 , 62.65966 , 31.963083 , 24.469461 ,
47.455463 , 25.022343 , 18.459959 , 25.76804 , 20.784414 ,
21.212088 , 10.94429 , 13.6669855, 13.091532 , 13.261982 ,
15.874415 , 19.665571 , 26.65234 , 23.44771 , 16.414022 ,
14.419901 , 22.863268 , 13.897646 , 41.792107 , 38.90065 ,
45.915737 , 49.371555 , 31.315296 , 12.263399 , 30.963991 ,
62.731266 , 91.94937 , 28.759546 , 49.877666 , 12.539829 ,
37.474133 , 31.584576 , 38.961147 , 43.7233 , 29.491695 ,
11.212091 , 10.949663 , 16.365812 , 33.53551 , 11.266697 ,
14.327033 , 13.410329 , 16.326382 , 28.509527 , 16.585867 ,
20.788076 , 10.168418 , 24.822367 , 24.971884 , 22.246784 ,
28.665592 , 30.169128 , 28.447351 , 20.21415 , 26.229086 ,
14.376749 , 15.402897 , 16.336533 , 23.407915 , 22.655249 ,
63.2682 , 17.057951 , 27.14274 , 22.407038 , 21.540663 ,
17.228092 , 31.131224 , 10.427744 , 23.655306 , 30.84339 ,
29.72337 , 11.57591 , 12.629789 , 23.162363 , 152.12224 ,
17.37777 , 49.686085 , 38.217213 , 6.200782 , 17.491488 ,
16.955462 , 12.221439 , 76.69378 , 48.732212 , 21.174788 ,
16.974009 , 21.722109 , 13.397806 , 30.00752 , 21.883944 ,
23.188337 , 21.771322 , 26.45743 , 11.8614 , 13.528744 ,
12.325241 , 10.60121 , 17.709867 , 12.492927 , 56.40955 ,
18.475695 , 24.155245 , 15.563114 , 45.0291 , 30.906923 ,
50.523193 , 32.456066 , 18.669321 , 17.98613 , 18.244167 ,
40.597675 , 25.390787 , 43.51433 , 8.770008 , 15.551195 ,
26.492525 , 12.731497 , 19.182997 , 19.398142 , 12.086107 ,
13.844336 , 19.543814 , 19.663733 , 15.902086 , 32.211285 ,
17.055061 , 16.609568 , 12.5422 , 11.19671 , 22.307482 ,
29.885979 , 17.127344 , 18.54088 , 18.22986 , 14.332409 ,
18.534245 , 8.854955 , 13.379894 , 11.005058 , 9.455654 ,
12.690422 , 12.499697 , 12.940079 , 8.934736 , 25.006073 ,
13.402452 , 20.573765 , 10.877329 , 8.093711 , 46.59967 ,
31.864634 , 40.02611 , 92.66779 , 55.76471 , 29.547998 ,
28.353556 , 13.127747 , 12.657144 , 13.203468 , 11.9830475,
13.63901 , 18.361464 , 10.321588 , 21.434431 , 11.251289 ,
11.940774 , 7.772327 , 15.981235 , 52.98866 , 28.60234 ,
31.327269 , 31.533604 , 21.875006 , 33.89951 , 28.973486 ,
39.061123 , 38.50964 , 28.725042 , 17.768194 , 23.66232 ,
19.722464 , 41.921757 , 13.648109 , 14.257262 , 21.89126 ,
28.267046 , 19.048292 , 62.88085 , 26.106714 , 19.2394 ,
43.560993 , 33.579353 , 29.373468 , 38.420246 , 32.79838 ,
44.19013 , 26.640135 , 22.966896 , 9.720984 , 29.09915 ,
17.466688 , 23.847116 , 10.286739 , 20.243505 , 20.899103 ,
17.164846 , 16.400167 , 16.076075 , 43.09666 , 18.647457 ,
24.760546 , 16.414967 , 10.573597 , 64.85003 , 32.896515 ,
36.177402 , 26.318235 , 36.60629 , 23.404049 , 20.402998 ,
47.446392 , 17.466173 , 23.769876 , 27.580992 , 21.75085 ,

```
        19.234144 ,   8.928478 ,  17.079042 ,  12.365701 ,  15.504739 ,
        13.988066 ,  10.963501 ,  13.514893 ,  13.434107 ,  15.218806 ,
        24.524042 ,  11.149149 ,  19.497814 ,   9.400767 ,  13.035424 ,
        12.362514 ,  10.787676 ,  11.530916 ,  13.42346  ,  15.611746 ,
        16.465837 ,  12.452387 ,  12.5047035,   9.955307 ,  10.32907  ,
        11.770672 ,   9.714632 ,   8.639699 ,  11.256895 ,  12.542776 ,
        29.973228 ,  10.900404 ,  10.462853 ,  15.404116 ,  11.440291 ,
        30.112501 ,  33.4202   ,  32.812374 ,  14.718645 ,  18.706034 ,
        37.01601  ,  24.778025 ,  15.86528  ,  10.593363 ,   9.654915 ,
        10.702266 ,  13.184469 ,  11.963707 ,   8.914192 ,  10.309573 ,
        13.888376 ,  11.595388 ,  13.254608 ,  10.734537 ,  12.481808 ,
        10.862255 ,  13.951749 ,   9.38986  ,  14.535303 ,  22.987494 ,
        13.209288 ,  11.008216 ,  17.189978 ,  13.156677 ,   9.673297 ,
         9.575037 ,  11.499191 ,  20.667397 ,  10.748065 ,  11.618143 ,
        11.314845 ,   9.421642 ,  11.200173 ,  15.538531 ,  13.054743 ,
        11.124045 ,  27.843306 ,  19.899363 ,  13.265014 ,  13.903096 ,
        22.426582 ,  12.584957 ,  31.101923 ,  15.389742 ,  17.344656 ,
        11.272923 ,   6.3225145,  21.726652 ,  26.79259  ,  43.22578  ,
        13.940535 ,  35.361294 ,  20.470942 ,  37.4109   ,  13.69034  ,
        62.067455 ,  10.295662 ,  14.320028 ,   6.2013874,   9.615353 ,
        10.193489 ,  12.932112 ,   9.54437  ,  13.768511 ,  31.721369 ,
        13.790487 ,  30.009695 ,  38.537945 ,  81.00288  ,   9.594304 ,
        14.1083355,  24.776472 ,  17.041985 ,  14.905752 ,  16.416456 ,
         9.9295635,  25.22701  ,   8.273629 ], dtype=float32),
'rmse': 6.259019974514894,
'r2': 0.8663901085255814,
'bias': -0.4263318489454818,
'rpd': 2.735777001992952}
```

4.2.3 Test autoglone

```
# AutoGluon Testing
from autogluon.tabular import TabularDataset, TabularPredictor
import pandas as pd
```

```
# Convert numpy arrays to DataFrame with wavelength columns
wavelengths = range(500, 2500, 2)  # Your wavelength range
train_df = pd.DataFrame(X_train_pls, columns=range(1, X_train_pls.shape[1] + 1))
test_df = pd.DataFrame(X_test_pls, columns=range(1, X_train_pls.shape[1] + 1))
```

```
# Add target variable
train_df['SOC'] = y_train
test_df['SOC'] = y_test
```

```python
# More advanced configuration with safety measures
predictor = TabularPredictor(
    label='SOC',
    problem_type="regression",
    eval_metric='root_mean_squared_error',
).fit(
    train_df,
    presets='best_quality',
    num_gpus=0,
    num_cpus=1,
    memory_limit='auto',
    auto_stack=False,
    verbosity=2
)
```

```python
print(f"predictor path is {predictor.path}")
# Show leaderboard
print("\nModel Leaderboard:")
print(predictor.leaderboard(test_df))
```

```
predictor path is c:\Users\luis_\Desktop\Alles\Uni\Leipzig\WS_24_25\spectroscopy\final_project\

Model Leaderboard:
                  model  score_test  score_val             eval_metric  \
0   NeuralNetFastAI_r191   -6.465653  -7.858265  root_mean_squared_error
1     WeightedEnsemble_L2   -6.565789  -7.667403  root_mean_squared_error
2          CatBoost_r137   -7.100844 -10.370113  root_mean_squared_error
3               CatBoost   -7.319287  -9.634744  root_mean_squared_error
4          CatBoost_r177   -7.464913  -9.398295  root_mean_squared_error
5        NeuralNetFastAI   -7.833119  -8.503627  root_mean_squared_error
6           CatBoost_r13   -8.104972  -8.996269  root_mean_squared_error
7            CatBoost_r9   -8.638408  -9.252907  root_mean_squared_error
8                XGBoost   -8.678744 -11.231818  root_mean_squared_error
9            XGBoost_r33   -9.264493 -10.793384  root_mean_squared_error
10       RandomForestMSE  -10.015084 -12.752302  root_mean_squared_error
11        ExtraTrees_r42  -10.321236 -12.743379  root_mean_squared_error
12         ExtraTreesMSE  -10.338205 -12.501554  root_mean_squared_error
13        KNeighborsUnif  -11.313581 -14.232448  root_mean_squared_error
14  NeuralNetFastAI_r102  -11.314371 -13.623935  root_mean_squared_error
15        KNeighborsDist  -11.326214 -14.656925  root_mean_squared_error

    pred_time_test  pred_time_val     fit_time  pred_time_test_marginal  \
0         0.062636       0.025176     9.271056                 0.062636
1         0.193922       0.033624  1533.309282                 0.021792
2         0.080354       0.006184   367.305470                 0.080354
3         0.063215       0.005027   482.195087                 0.063215
```

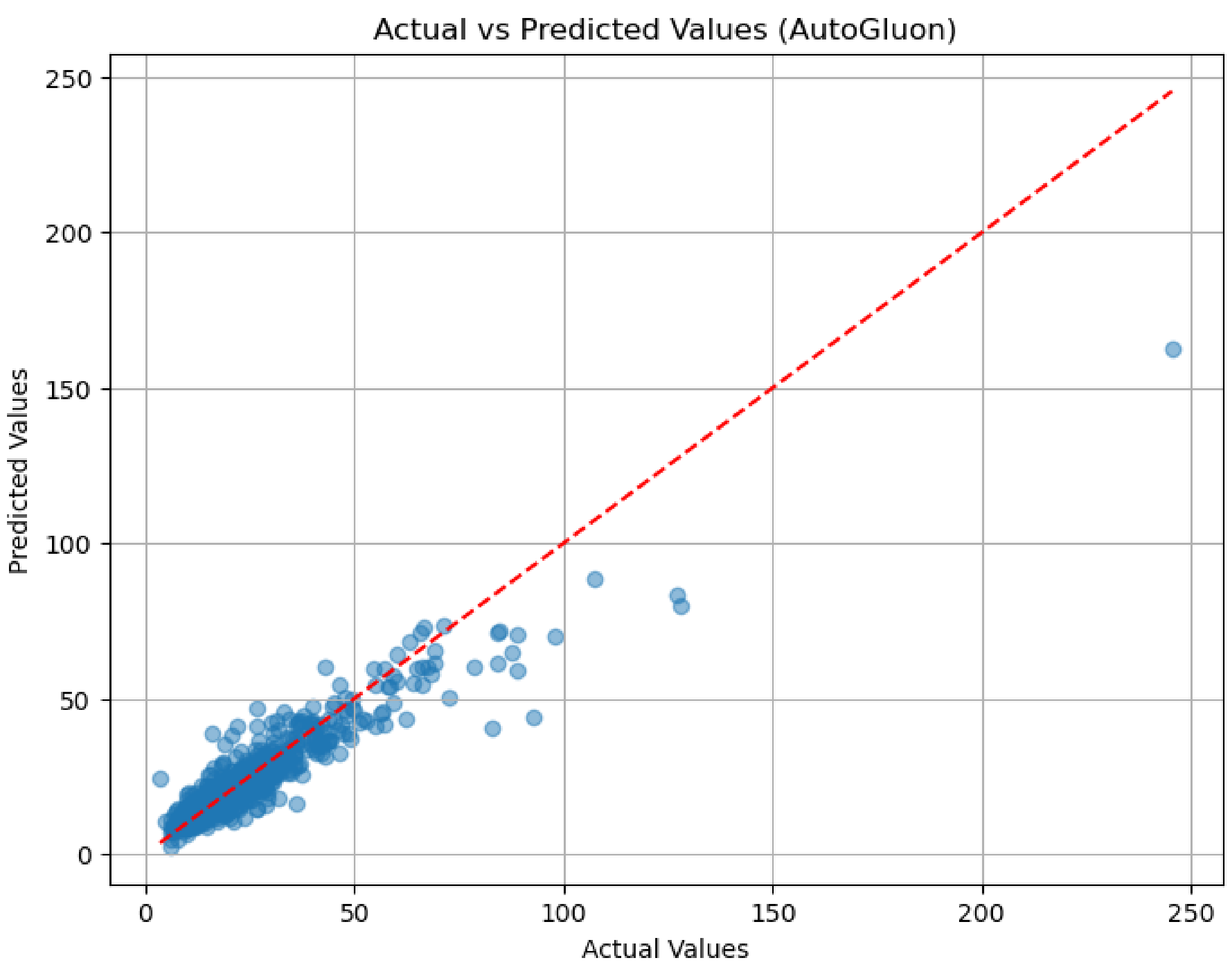
4	0.027581	0.006836	86.732648	0.027581
5	0.040360	0.002593	3.638255	0.040360
6	0.109494	0.008448	1524.012893	0.109494
7	0.057354	0.025461	1030.988951	0.057354
8	0.031350	0.008112	11.257859	0.031350
9	0.109050	0.012634	55.511277	0.109050
10	0.215906	0.480414	4.535789	0.215906
11	0.216460	0.122088	1.322315	0.216460
12	0.212351	0.099897	1.110185	0.212351
13	0.036509	0.023081	0.022188	0.036509
14	0.130343	0.071021	15.547421	0.130343
15	0.029869	0.024546	0.022627	0.029869

	pred_time_val_marginal	fit_time_marginal	stack_level	can_infer \
0	0.025176	9.271056	1	True
1	0.000000	0.025332	2	True
2	0.006184	367.305470	1	True
3	0.005027	482.195087	1	True
4	0.006836	86.732648	1	True
5	0.002593	3.638255	1	True
6	0.008448	1524.012893	1	True
7	0.025461	1030.988951	1	True
8	0.008112	11.257859	1	True
9	0.012634	55.511277	1	True
10	0.480414	4.535789	1	True
11	0.122088	1.322315	1	True
12	0.099897	1.110185	1	True
13	0.023081	0.022188	1	True
14	0.071021	15.547421	1	True
15	0.024546	0.022627	1	True

	fit_order
0	9
1	16
2	13
3	4
4	8
5	6
6	15
7	10
8	7
9	11
10	3
11	12
12	5
13	1
14	14
15	2

```python
# Get predictions and evaluate
y_pred_auto2 = predictor.predict(test_df.drop(columns=['SOC']))

# Evaluate using your existing function
autogluon_eval2 = own_functions.evaluate_model(
    predictor,
    X_test=test_df.drop(columns=['SOC']),
    y_test=test_df['SOC'],
    print_metrics=True,
    show_plot=True,
    plot_kwargs={'model_name': 'AutoGluon', 'figsize': (8, 6)}
)
```

```
Root Mean Squared Error (RMSE): 6.5658
R²: 0.8530
Bias: -0.5860
RPD: 2.6080
```



4.3 Stragtegy 3: Testing auxiallary spectral data

4.3.1 DLR Spectral Data

Load the auxiliary data geopandas dataframe.

Remove Points without results. Check for MREF vs SRC.

Add parameters to training and test data of the plsr absorbance latent variables.

```
dlr_aux_data = gpd.read_parquet("data/auxiliary_data_results.parquet")
dlr_aux_data.head()
```

	y	x	spatial_ref	time	MREF_B02	MREF_B03	MREF_B04	MREF_B08
0	2545990.0	3908010.0	3035	2018-03-01	399	688	706	2692
1	2537990.0	3899990.0	3035	2018-03-01	232	466	446	2672
2	2555990.0	3907990.0	3035	2018-03-01	556	936	937	3173
3	2548010.0	3902010.0	3035	2018-03-01	567	927	946	3366
4	2568010.0	3894010.0	3035	2018-03-01	302	572	557	2808

```
# Assuming dlr_aux_data contains a 'point_index' column as implemented in the improved code

# Get the original indices
original_ks_indices = ks_indices
original_test_indices = test_indices

# Create a mapping from original indices to rows in the auxiliary data
point_to_row = dict(zip(dlr_aux_data['point_index'], dlr_aux_data.index))

# Filter indices to only include those with data in dlr_aux_data
valid_ks_indices = [idx for idx in original_ks_indices if idx in point_to_row]
valid_test_indices = [idx for idx in original_test_indices if idx in point_to_row]

# Map original indices to row positions in dlr_aux_data
mapped_ks_indices = [point_to_row[idx] for idx in valid_ks_indices]
mapped_test_indices = [point_to_row[idx] for idx in valid_test_indices]

# Now use these mapped indices to select data
aux_train_ks = dlr_aux_data.iloc[mapped_ks_indices]
aux_test_ks = dlr_aux_data.iloc[mapped_test_indices]

# Print shapes
print(f"Original indices: {len(original_ks_indices)} training, {len(original_test_indices)}
print(f"Valid indices with aux data: {len(valid_ks_indices)} training, {len(valid_test_indic
```

```
Original indices: 1964 training, 843 test
Valid indices with aux data: 1962 training, 843 test
```

```
# select only relevant columns
aux_train_ks = aux_train_ks[["MREF_B02","MREF_B03", "MREF_B04", "MREF_B08", "MREF_B11", "MRE
aux_test_ks = aux_test_ks[["MREF_B02","MREF_B03",   "MREF_B04", "MREF_B08", "MREF_B11", "MRE

print(f"Training data shape: {aux_train_ks.shape}")
print(f"Test data shape: {aux_test_ks.shape}")
```

```
Training data shape: (1962, 6)
Test data shape: (843, 6)
```

```
original_train_indices = ks_indices  # These are the indices used to create X_train_pls

# Map from original index to position in X_train_pls
original_to_train_pos = {orig_idx: train_pos for train_pos, orig_idx in enumerate(original_t

# Find the positions in X_train_pls that correspond to valid_ks_indices
train_pls_positions = []
for idx in valid_ks_indices:
    if idx in original_to_train_pos:
        train_pls_positions.append(original_to_train_pos[idx])

# Now use these positions to select from X_train_pls
X_train_pls_aux = X_train_pls[train_pls_positions, :]
y_train_aux = y_train[train_pls_positions]

# Do the same for test data
original_test_indices = test_indices  # These are the indices used to create X_test_pls
original_to_test_pos = {orig_idx: test_pos for test_pos, orig_idx in enumerate(original_test

test_pls_positions = []
for idx in valid_test_indices:
    if idx in original_to_test_pos:
        test_pls_positions.append(original_to_test_pos[idx])

X_test_pls_aux = X_test_pls[test_pls_positions, :]
y_test_aux = y_test[test_pls_positions]

# Print shapes
print(f"X_train_pls_aux shape: {X_train_pls_aux.shape}")
print(f"X_test_pls_aux shape: {X_test_pls_aux.shape}")
print(f"y_train_aux shape: {y_train_aux.shape}")
print(f"y_test_aux shape: {y_test_aux.shape}")
```

```
X_train_pls_aux shape: (1962, 48)
X_test_pls_aux shape: (843, 48)
y_train_aux shape: (1962,)
y_test_aux shape: (843,)
```

```
# Add the auxiliary data to the PLSR transformed data
X_train_combined = np.hstack((X_train_pls_aux, aux_train_ks))
X_test_combined = np.hstack((X_test_pls_aux, aux_test_ks))

# Print shapes
print(f"X_train_combined shape: {X_train_combined.shape}")
print(f"X_test_combined shape: {X_test_combined.shape}")
```

```
X_train_combined shape: (1962, 54)
X_test_combined shape: (843, 54)
```

```
# First split training data into train and validation sets
X_train_final, X_val, y_train_final, y_val = train_test_split(X_train_pls_aux, y_train_aux,
                                                              test_size=0.2,
                                                              random_state=42)

# Training enhanced LSTM model
LSTM_plsr_aux_model, history, metrics = own_functions.train_and_evaluate_lstm(
    X_train=X_train_final,
    X_val=X_val,
    X_test=X_test_pls_aux,
    y_train=y_train_final,
    y_val=y_val,
    y_test=y_test_aux,
    hidden_size=256,
    num_layers=5,
    num_epochs=3000,
    learning_rate=0.001,
    patience=200,  # Early stopping patience
    dropout=0.2
)


lstm_plsr_eval = own_functions.evaluate_model(LSTM_plsr_aux_model,
                             X_test=X_test_pls_aux, y_test=y_test_aux,
                             print_metrics=True, show_plot=True)
```

```
Epoch [10/3000], Train Loss: 1544.1721, Val Loss: 1261.2821
Epoch [20/3000], Train Loss: 1360.5717, Val Loss: 1052.5266
```

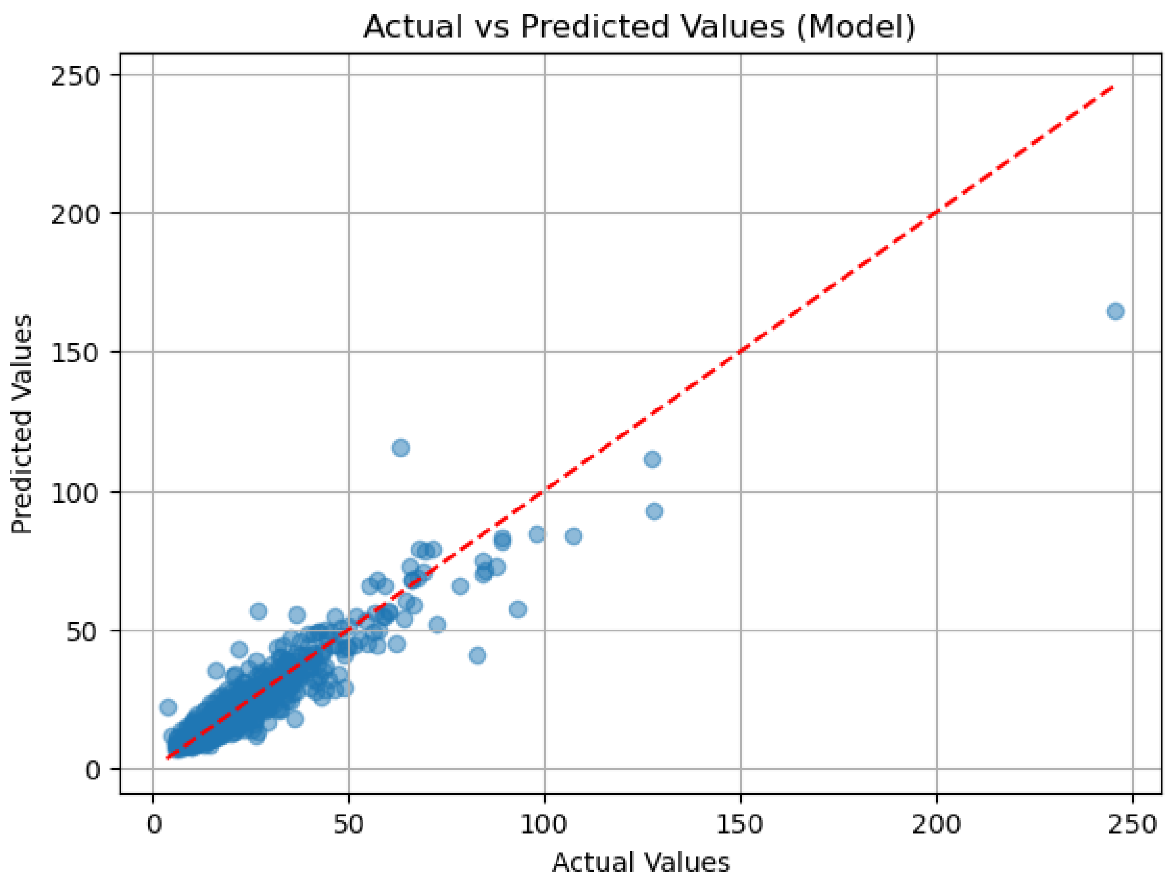
```
Epoch [30/3000], Train Loss: 1113.0549, Val Loss: 825.0857
Epoch [40/3000], Train Loss: 994.2122, Val Loss: 718.9977
Epoch [50/3000], Train Loss: 922.8384, Val Loss: 652.9730
Epoch [60/3000], Train Loss: 871.2383, Val Loss: 605.0820
Epoch [70/3000], Train Loss: 834.0046, Val Loss: 570.8928
Epoch [80/3000], Train Loss: 807.6899, Val Loss: 547.0749
Epoch [90/3000], Train Loss: 789.5063, Val Loss: 530.9082
Epoch [100/3000], Train Loss: 777.2578, Val Loss: 520.2526
Epoch [110/3000], Train Loss: 769.1790, Val Loss: 513.4652
Epoch [120/3000], Train Loss: 764.0148, Val Loss: 509.3117
Epoch [130/3000], Train Loss: 760.8263, Val Loss: 506.8878
Epoch [140/3000], Train Loss: 758.8932, Val Loss: 505.5359
Epoch [150/3000], Train Loss: 757.7497, Val Loss: 504.8125
Epoch [160/3000], Train Loss: 757.1155, Val Loss: 504.3330
Epoch [170/3000], Train Loss: 756.4865, Val Loss: 503.7638
Epoch [180/3000], Train Loss: 754.8932, Val Loss: 501.6933
Epoch [190/3000], Train Loss: 753.5971, Val Loss: 500.1130
Epoch [200/3000], Train Loss: 750.9866, Val Loss: 498.7947
Epoch [210/3000], Train Loss: 748.6475, Val Loss: 496.2296
Epoch [220/3000], Train Loss: 744.1456, Val Loss: 492.8725
Epoch [230/3000], Train Loss: 735.7213, Val Loss: 484.6309
Epoch [240/3000], Train Loss: 724.7082, Val Loss: 472.3252
Epoch [250/3000], Train Loss: 693.4603, Val Loss: 436.7836
Epoch [260/3000], Train Loss: 644.9241, Val Loss: 391.5297
Epoch [270/3000], Train Loss: 619.3101, Val Loss: 366.9921
Epoch [280/3000], Train Loss: 596.6960, Val Loss: 347.6390
Epoch [290/3000], Train Loss: 575.8994, Val Loss: 328.6137
Epoch [300/3000], Train Loss: 557.8859, Val Loss: 313.7814
Epoch [310/3000], Train Loss: 541.5688, Val Loss: 298.5481
Epoch [320/3000], Train Loss: 526.2053, Val Loss: 286.2219
Epoch [330/3000], Train Loss: 513.4929, Val Loss: 275.0917
Epoch [340/3000], Train Loss: 502.1014, Val Loss: 265.7979
Epoch [350/3000], Train Loss: 490.2899, Val Loss: 257.1266
Epoch [360/3000], Train Loss: 480.3848, Val Loss: 250.1747
Epoch [370/3000], Train Loss: 469.9088, Val Loss: 243.2771
Epoch [380/3000], Train Loss: 461.2568, Val Loss: 237.5328
Epoch [390/3000], Train Loss: 454.0612, Val Loss: 232.2784
Epoch [400/3000], Train Loss: 446.9914, Val Loss: 228.2343
Epoch [410/3000], Train Loss: 441.0858, Val Loss: 221.7805
Epoch [420/3000], Train Loss: 433.3401, Val Loss: 217.6800
Epoch [430/3000], Train Loss: 427.5756, Val Loss: 214.0151
Epoch [440/3000], Train Loss: 421.1319, Val Loss: 208.8647
Epoch [450/3000], Train Loss: 414.6551, Val Loss: 205.8686
Epoch [460/3000], Train Loss: 410.0074, Val Loss: 201.4649
Epoch [470/3000], Train Loss: 403.7576, Val Loss: 197.5853
Epoch [480/3000], Train Loss: 400.4884, Val Loss: 195.0382
Epoch [490/3000], Train Loss: 393.4198, Val Loss: 191.2844
Epoch [500/3000], Train Loss: 389.1021, Val Loss: 188.7811
```

```
Epoch [510/3000], Train Loss: 386.2618, Val Loss: 184.5472
Epoch [520/3000], Train Loss: 379.0336, Val Loss: 182.4513
Epoch [530/3000], Train Loss: 375.5767, Val Loss: 179.0549
Epoch [540/3000], Train Loss: 371.2313, Val Loss: 177.9756
Epoch [550/3000], Train Loss: 368.8823, Val Loss: 174.3152
Epoch [560/3000], Train Loss: 363.7503, Val Loss: 171.4698
Epoch [570/3000], Train Loss: 361.1311, Val Loss: 169.5117
Epoch [580/3000], Train Loss: 356.1508, Val Loss: 166.4503
Epoch [590/3000], Train Loss: 353.5980, Val Loss: 164.4879
Epoch [600/3000], Train Loss: 348.8084, Val Loss: 161.0004
Epoch [610/3000], Train Loss: 346.1577, Val Loss: 160.1836
Epoch [620/3000], Train Loss: 343.2888, Val Loss: 159.6847
Epoch [630/3000], Train Loss: 340.6157, Val Loss: 158.9820
Epoch [640/3000], Train Loss: 336.2419, Val Loss: 162.6344
Epoch [650/3000], Train Loss: 335.8001, Val Loss: 166.2111
Epoch [660/3000], Train Loss: 328.8096, Val Loss: 158.2751
Epoch [670/3000], Train Loss: 326.4559, Val Loss: 168.2849
Epoch [680/3000], Train Loss: 325.1979, Val Loss: 173.9978
Epoch [690/3000], Train Loss: 322.4661, Val Loss: 166.0625
Epoch [700/3000], Train Loss: 317.8439, Val Loss: 168.9417
Epoch [710/3000], Train Loss: 313.0585, Val Loss: 162.8007
Epoch [720/3000], Train Loss: 311.0229, Val Loss: 169.7988
Epoch [730/3000], Train Loss: 308.4498, Val Loss: 165.8283
Epoch [740/3000], Train Loss: 304.4712, Val Loss: 158.4907
Epoch [750/3000], Train Loss: 303.9171, Val Loss: 161.2780
Epoch [760/3000], Train Loss: 301.3704, Val Loss: 161.1650
Epoch [770/3000], Train Loss: 296.2749, Val Loss: 157.2540
Epoch [780/3000], Train Loss: 293.5692, Val Loss: 167.0943
Epoch [790/3000], Train Loss: 291.8961, Val Loss: 163.8841
Epoch [800/3000], Train Loss: 288.4707, Val Loss: 158.2478
Epoch [810/3000], Train Loss: 286.9824, Val Loss: 155.0916
Epoch [820/3000], Train Loss: 282.7285, Val Loss: 158.9160
Epoch [830/3000], Train Loss: 281.2551, Val Loss: 157.5095
Epoch [840/3000], Train Loss: 278.1798, Val Loss: 157.1717
Epoch [850/3000], Train Loss: 275.6295, Val Loss: 150.9236
Epoch [860/3000], Train Loss: 274.0131, Val Loss: 147.4631
Epoch [870/3000], Train Loss: 271.8761, Val Loss: 150.4485
Epoch [880/3000], Train Loss: 270.5312, Val Loss: 146.5031
Epoch [890/3000], Train Loss: 268.1956, Val Loss: 156.4339
Epoch [900/3000], Train Loss: 264.7632, Val Loss: 147.6711
Epoch [910/3000], Train Loss: 263.0856, Val Loss: 148.7383
Epoch [920/3000], Train Loss: 263.1754, Val Loss: 148.1244
Epoch [930/3000], Train Loss: 259.8183, Val Loss: 144.9549
Epoch [940/3000], Train Loss: 257.3921, Val Loss: 146.1749
Epoch [950/3000], Train Loss: 255.6140, Val Loss: 142.7679
Epoch [960/3000], Train Loss: 254.4053, Val Loss: 143.2168
Epoch [970/3000], Train Loss: 251.5756, Val Loss: 133.4638
Epoch [980/3000], Train Loss: 250.3265, Val Loss: 136.4730
```

```
Epoch [990/3000], Train Loss: 247.1057, Val Loss: 142.5507
Epoch [1000/3000], Train Loss: 246.5377, Val Loss: 136.9362
Epoch [1010/3000], Train Loss: 244.2855, Val Loss: 140.0231
Epoch [1020/3000], Train Loss: 242.1759, Val Loss: 136.4840
Epoch [1030/3000], Train Loss: 242.0377, Val Loss: 138.2665
Epoch [1040/3000], Train Loss: 238.9992, Val Loss: 136.8106
Epoch [1050/3000], Train Loss: 237.6101, Val Loss: 129.6251
Epoch [1060/3000], Train Loss: 235.9276, Val Loss: 127.9842
Epoch [1070/3000], Train Loss: 233.6136, Val Loss: 134.8420
Epoch [1080/3000], Train Loss: 232.3378, Val Loss: 131.5125
Epoch [1090/3000], Train Loss: 230.6135, Val Loss: 134.8204
Epoch [1100/3000], Train Loss: 227.8888, Val Loss: 127.6874
Epoch [1110/3000], Train Loss: 227.4961, Val Loss: 132.9175
Epoch [1120/3000], Train Loss: 224.5971, Val Loss: 134.1447
Epoch [1130/3000], Train Loss: 223.4994, Val Loss: 136.3112
Epoch [1140/3000], Train Loss: 218.5809, Val Loss: 142.7510
Epoch [1150/3000], Train Loss: 216.9254, Val Loss: 134.8037
Epoch [1160/3000], Train Loss: 213.7336, Val Loss: 128.9296
Epoch [1170/3000], Train Loss: 212.2646, Val Loss: 132.9827
Epoch [1180/3000], Train Loss: 209.0687, Val Loss: 131.2313
Epoch [1190/3000], Train Loss: 208.9754, Val Loss: 135.0260
Epoch [1200/3000], Train Loss: 206.8528, Val Loss: 136.4576
Epoch [1210/3000], Train Loss: 204.8011, Val Loss: 130.4109
Epoch [1220/3000], Train Loss: 202.4032, Val Loss: 130.7426
Epoch [1230/3000], Train Loss: 200.4675, Val Loss: 137.5190
Epoch [1240/3000], Train Loss: 199.7327, Val Loss: 131.7733
Epoch [1250/3000], Train Loss: 197.9577, Val Loss: 134.7573
Epoch [1260/3000], Train Loss: 197.5121, Val Loss: 135.4478
Epoch [1270/3000], Train Loss: 196.8811, Val Loss: 129.4920
Epoch [1280/3000], Train Loss: 193.5388, Val Loss: 126.1479
Epoch [1290/3000], Train Loss: 191.2031, Val Loss: 124.5572
Epoch [1300/3000], Train Loss: 190.5298, Val Loss: 127.4417
Epoch [1310/3000], Train Loss: 188.7287, Val Loss: 127.9528
Epoch [1320/3000], Train Loss: 186.7330, Val Loss: 127.6173
Epoch [1330/3000], Train Loss: 185.4254, Val Loss: 121.7067
Epoch [1340/3000], Train Loss: 183.7905, Val Loss: 120.0501
Epoch [1350/3000], Train Loss: 182.7077, Val Loss: 125.7767
Epoch [1360/3000], Train Loss: 181.0904, Val Loss: 125.3952
Epoch [1370/3000], Train Loss: 179.2245, Val Loss: 119.8019
Epoch [1380/3000], Train Loss: 179.9510, Val Loss: 124.5295
Epoch [1390/3000], Train Loss: 176.2825, Val Loss: 121.1589
Epoch [1400/3000], Train Loss: 175.1880, Val Loss: 132.3049
Epoch [1410/3000], Train Loss: 175.2047, Val Loss: 127.0799
Epoch [1420/3000], Train Loss: 172.2049, Val Loss: 122.1558
Epoch [1430/3000], Train Loss: 171.4966, Val Loss: 123.3746
Epoch [1440/3000], Train Loss: 169.4692, Val Loss: 122.7484
Epoch [1450/3000], Train Loss: 168.3537, Val Loss: 123.1459
Epoch [1460/3000], Train Loss: 166.7366, Val Loss: 123.6101
```

```
Epoch [1470/3000], Train Loss: 165.9348, Val Loss: 115.5010
Epoch [1480/3000], Train Loss: 164.2000, Val Loss: 126.7141
Epoch [1490/3000], Train Loss: 164.3276, Val Loss: 119.9286
Epoch [1500/3000], Train Loss: 161.9965, Val Loss: 124.0390
Epoch [1510/3000], Train Loss: 161.0406, Val Loss: 124.3344
Epoch [1520/3000], Train Loss: 161.0278, Val Loss: 122.6168
Epoch [1530/3000], Train Loss: 157.4306, Val Loss: 124.8954
Epoch [1540/3000], Train Loss: 156.8403, Val Loss: 119.3694
Epoch [1550/3000], Train Loss: 155.8226, Val Loss: 126.4181
Epoch [1560/3000], Train Loss: 154.0519, Val Loss: 123.8667
Epoch [1570/3000], Train Loss: 153.6245, Val Loss: 120.3965
Epoch [1580/3000], Train Loss: 153.2056, Val Loss: 118.1762
Epoch [1590/3000], Train Loss: 150.5750, Val Loss: 123.9308
Epoch [1600/3000], Train Loss: 149.7247, Val Loss: 121.8038
Epoch [1610/3000], Train Loss: 148.0048, Val Loss: 117.8583
Epoch [1620/3000], Train Loss: 147.2606, Val Loss: 115.5548
Epoch [1630/3000], Train Loss: 147.4237, Val Loss: 120.2231
Epoch [1640/3000], Train Loss: 147.1860, Val Loss: 127.8164
Epoch [1650/3000], Train Loss: 144.5185, Val Loss: 129.7091
Epoch [1660/3000], Train Loss: 143.4535, Val Loss: 120.1680
Epoch [1670/3000], Train Loss: 143.3921, Val Loss: 119.5819
Epoch [1680/3000], Train Loss: 142.7135, Val Loss: 123.1209
Epoch [1690/3000], Train Loss: 139.9218, Val Loss: 122.1314
Epoch [1700/3000], Train Loss: 139.0835, Val Loss: 116.7714
Epoch [1710/3000], Train Loss: 137.9424, Val Loss: 115.6254
Epoch [1720/3000], Train Loss: 137.2544, Val Loss: 121.4715
Epoch [1730/3000], Train Loss: 137.0690, Val Loss: 122.0736
Epoch [1740/3000], Train Loss: 135.7626, Val Loss: 126.1657
Epoch [1750/3000], Train Loss: 134.0135, Val Loss: 124.1627
Epoch [1760/3000], Train Loss: 133.6209, Val Loss: 122.3784
Epoch [1770/3000], Train Loss: 132.2018, Val Loss: 116.5889
Epoch [1780/3000], Train Loss: 132.4073, Val Loss: 111.3170
Epoch [1790/3000], Train Loss: 130.4919, Val Loss: 117.3146
Epoch [1800/3000], Train Loss: 129.5024, Val Loss: 123.8470
Epoch [1810/3000], Train Loss: 128.7990, Val Loss: 116.3894
Epoch [1820/3000], Train Loss: 128.5053, Val Loss: 115.7731
Epoch [1830/3000], Train Loss: 126.4875, Val Loss: 115.0462
Epoch [1840/3000], Train Loss: 125.0893, Val Loss: 115.4055
Epoch [1850/3000], Train Loss: 124.6842, Val Loss: 115.2740
Epoch [1860/3000], Train Loss: 124.3054, Val Loss: 115.4073
Epoch [1870/3000], Train Loss: 123.2910, Val Loss: 117.8070
Epoch [1880/3000], Train Loss: 122.6722, Val Loss: 115.7522
Epoch [1890/3000], Train Loss: 122.1090, Val Loss: 120.3975
Epoch [1900/3000], Train Loss: 121.2142, Val Loss: 125.2751
Epoch [1910/3000], Train Loss: 120.3529, Val Loss: 116.4668
Epoch [1920/3000], Train Loss: 119.6043, Val Loss: 117.9794
Epoch [1930/3000], Train Loss: 118.4358, Val Loss: 112.7287
Epoch [1940/3000], Train Loss: 118.0952, Val Loss: 112.3173
```

```
Epoch [1950/3000], Train Loss: 117.3524, Val Loss: 117.1025
Epoch [1960/3000], Train Loss: 116.6451, Val Loss: 114.3603
Epoch [1970/3000], Train Loss: 115.3970, Val Loss: 115.4723
Epoch [1980/3000], Train Loss: 114.8408, Val Loss: 113.7999
Epoch [1990/3000], Train Loss: 113.7793, Val Loss: 114.5056
Epoch [2000/3000], Train Loss: 113.5716, Val Loss: 116.6024
Early stopping triggered at epoch 2006

Final Test Metrics:
test_loss: 38.2456
rmse: 6.1843
r2: 0.8696
bias: -0.4046
rpd: 2.7688
Root Mean Squared Error (RMSE): 6.1843
R²: 0.8696
Bias: -0.4046
RPD: 2.7688
```



4.3.2 ISRIC Soil Data

```
# read pd from csv
isric_soil_data = pd.read_csv('data\soilgrids_parallel.csv')
isric_soil_data.head()
```

	point_index	lon	lat	bdod_0_to_5cm_mean	bdod_5_to_15cm_mean	bdod_15_to_30
0	1	4.584692	45.816720	125.0	134.0	141.0
1	0	4.680379	45.893933	128.0	138.0	141.0
2	3	4.601575	45.908022	133.0	140.0	144.0
3	2	4.671533	45.983716	129.0	139.0	143.0
4	6	4.439863	46.224665	102.0	116.0	121.0

5 Discussion of Results (5 P):

- Briefly discuss your results and interpret them based on the validation metrics for the test set.
- Compare your findings with those of published studies in a similar context.

- Evaluate whether soil VNIR reflectance spectroscopy could serve as a complementary approach for large-scale soil organic carbon assessment in Earth (system) science.

Additional Information:

The length of the discussion section really depends on your results, but as a general guideline, I would expect it to be around one page.

- **Focus on**:
  - directly comparing your different modeling approaches
  - interpreting which performed best based on the validation metrics
- If the results are not as good as expected:
  - consider discussing possible reasons and suggesting ways to improve them
  - (you might find 1-2 examples from the literature helpful here).
- Additionally, you could compare your findings with similar studies that have attempted to model SOC (or related properties) at national or continental scales using spectroscopy—ideally referencing 2-3 relevant publications.
- Finally, reflect on whether and how soil VNIR spectroscopy could contribute to large-scale soil information systems.
  - This is a more theoretical aspect, and you are free in how you approach this point.
  - Important aspects to consider might include:
    * a) Model accuracy (What would be considered a good accuracy in this context?)
    * b) Data harmonization (Challenges when combining datasets from different providers)
    * c) Practical usability (Would end users require programming skills, etc.?)

A recent publication that could provide a useful overview is: Peng et al. (2025): Spectroscopic solutions for generating new global soil information (Link: https://www.sciencedirect.com/science/article/pii/S266667