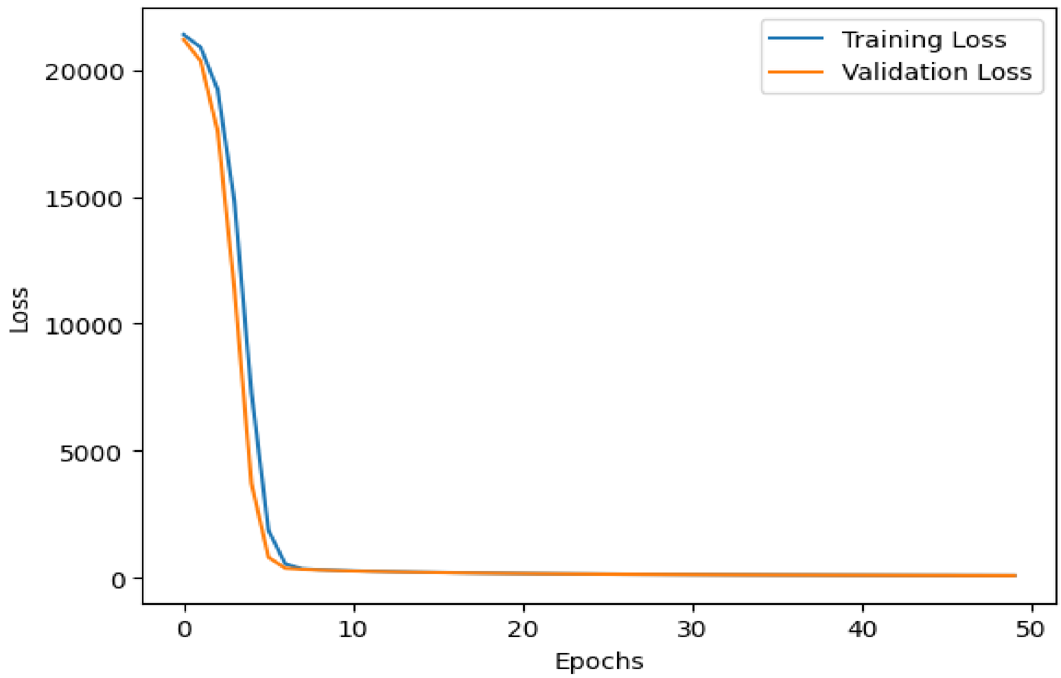
Attempt 1:

Using PCA and ANN

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, r2_score
from keras.models import Sequential
from keras.layers import Dense

# Load dataset
data = pd.read_csv('sample1.csv')

# Define the input features and target variable
X = data[['Longitude', 'Latitude', 'Elevation (m)', 'Altitude (m)',
'Clutter height (m)', 'Distance (m)']]
# Check for typos and ensure the column exists in the DataFrame. You might
need to adjust this based on the actual column name in your CSV file.
y = data['Path Loss (dB)']

# Split the dataset 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)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply PCA for feature selection
pca = PCA(n_components=6)  # Select the number of components based on your
preference
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Build the ANN model
model = Sequential()
model.add(Dense(64, input_dim=X_train_pca.shape[1], activation='relu'))  #
Input layer
model.add(Dense(32, activation='relu'))  # Hidden layer 1
model.add(Dense(16, activation='relu'))  # Hidden layer 2
```

```python
model.add(Dense(1))  # Output layer (regression task for path loss
prediction)

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Train the model
history = model.fit(X_train_pca, y_train, epochs=50, batch_size=32,
validation_split=0.2)

# Evaluate the model on the test set
y_pred = model.predict(X_test_pca)
mse = mean_squared_error(y_test, y_pred)
print(f'Mean Squared Error on test set: {mse}')

r2 = r2_score(y_test, y_pred)
print(f'R-squared on test set: {r2}')

# If you want to visualize the training loss
import matplotlib.pyplot as plt

plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



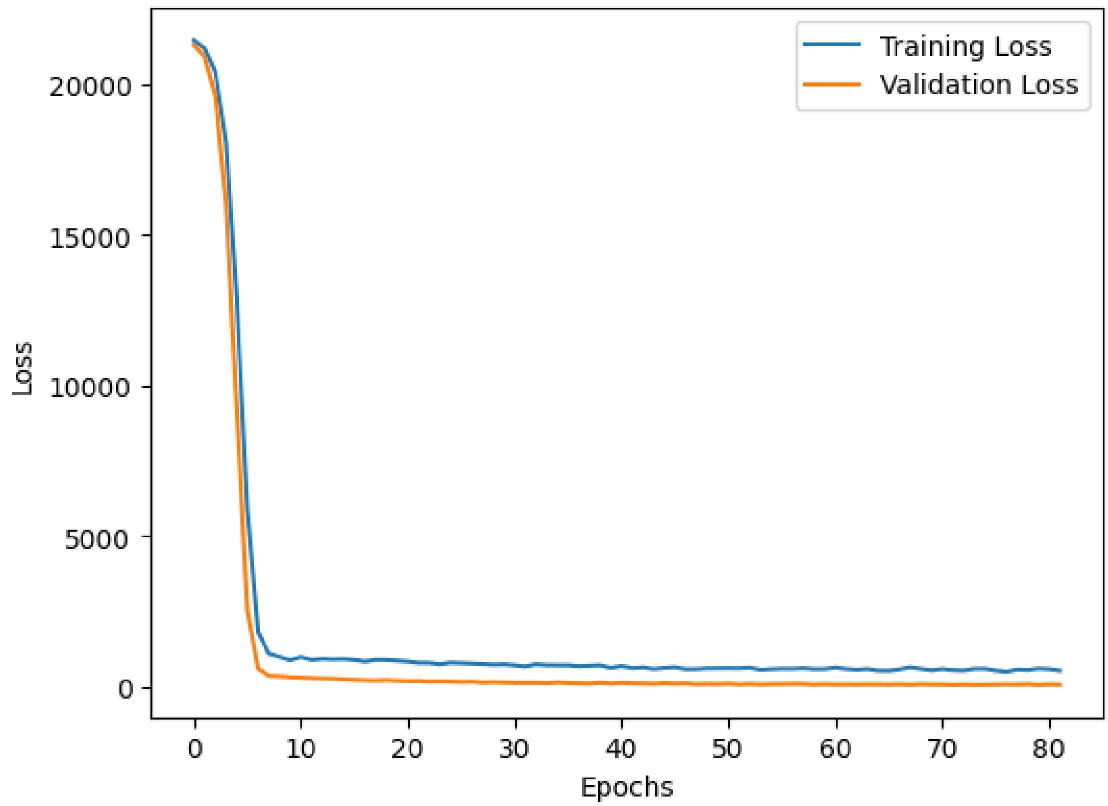

```
Mean Squared Error on test set: 78.35959671996133
R-squared on test set: -0.4744934586045313
```

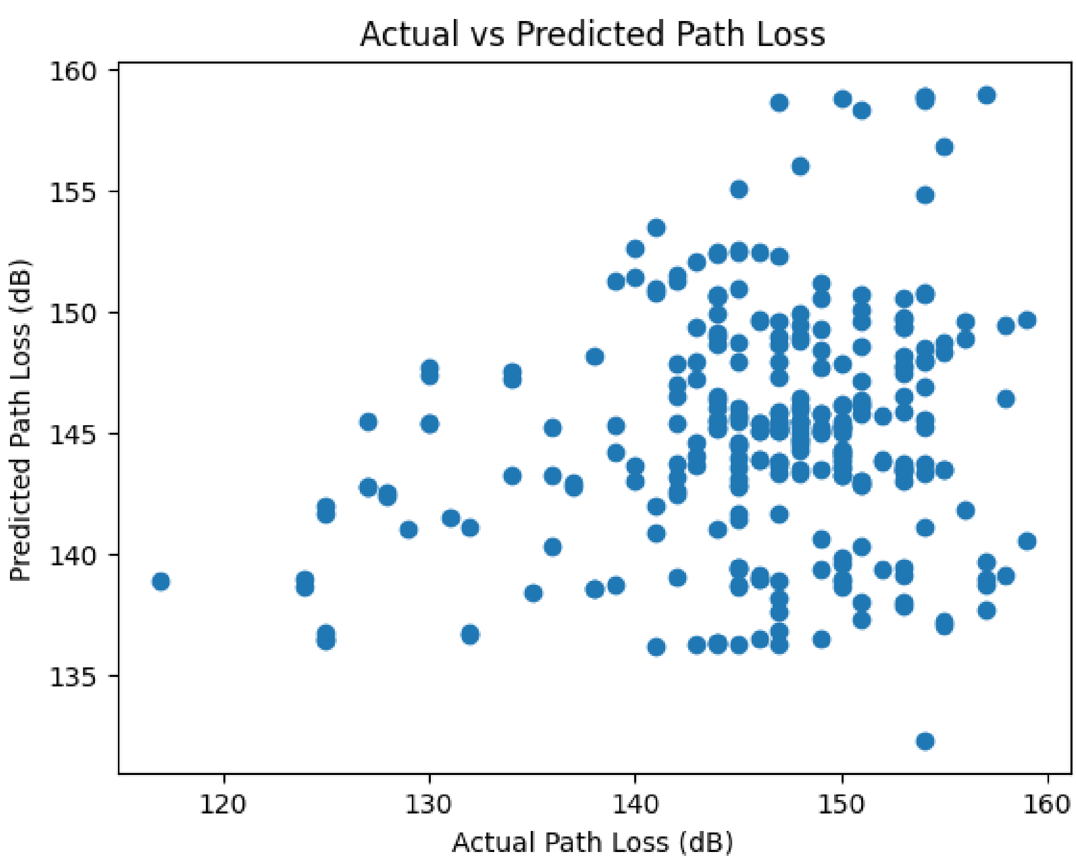
Attempt 2:

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, r2_score
from keras.models import Sequential
from keras.layers import Dense, Dropout
from keras.regularizers import l2
from keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

# Load dataset
data = pd.read_csv('sample1.csv')

# Define the input features and target variable
X = data[['Longitude', 'Latitude', 'Elevation (m)', 'Altitude (m)',
'Clutter height (m)', 'Distance (m)']]
y = data['Path Loss (dB)']

# Split the dataset 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)

# Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply PCA for feature selection
pca = PCA(n_components=6)  # Adjust components based on explained variance
ratio
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Build the improved ANN model with regularization and dropout
model = Sequential()
model.add(Dense(64, input_dim=X_train_pca.shape[1], activation='relu',
kernel_regularizer=l2(0.01)))  # Input layer with L2 regularization
model.add(Dropout(0.3))  # Dropout to prevent overfitting
model.add(Dense(32, activation='relu', kernel_regularizer=l2(0.01)))  #
Hidden layer 1 with L2 regularization
```

```
model.add(Dropout(0.3))  # Dropout layer
model.add(Dense(16, activation='relu'))  # Hidden layer 2
model.add(Dense(1))  # Output layer (for regression task)

# Compile the model
model.compile(optimizer='adam', loss='mean_squared_error')

# Set up early stopping to avoid overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)

# Train the model with early stopping
history = model.fit(X_train_pca, y_train, epochs=100, batch_size=32,
validation_split=0.2, callbacks=[early_stopping])

# Evaluate the model on the test set
y_pred = model.predict(X_test_pca)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
print(f'Mean Squared Error on test set: {mse}')
print(f'R-squared on test set: {r2}')

# Visualize the training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# If you want to visualize the predicted vs actual values
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Path Loss (dB)')
plt.ylabel('Predicted Path Loss (dB)')
plt.title('Actual vs Predicted Path Loss')
plt.show()
```


Training Loss
Validation Loss
Loss
Epochs
20000
15000
10000
5000
0
0
10
20
30
40
50
60
70
80


Actual vs Predicted Path Loss
Predicted Path Loss (dB)
Actual Path Loss (dB)
160
155
150
145
140
135
120
130
140
150
160

```
Mean Squared Error on test set: 64.1284114887103
R-squared on test set: -0.20670507772936975
```

Attempt 03:

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.regularizers import l2
from tensorflow.keras.callbacks import EarlyStopping, ReduceLROnPlateau
from sklearn.ensemble import RandomForestRegressor, VotingRegressor
import matplotlib.pyplot as plt

# Load dataset
data = pd.read_csv('sample1.csv')

# Define the input features and target variable
X = data[['Longitude', 'Latitude', 'Elevation (m)', 'Altitude (m)',
'Clutter height (m)', 'Distance (m)']]
y = data['Path Loss (dB)']

# Split the dataset 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)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply PCA
pca = PCA(n_components=6)  # Adjust n_components based on explained
variance
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Build and train the ANN model
def create_model(neurons=64, dropout_rate=0.3, optimizer='adam',
reg_param=0.01):
    model = Sequential()
    model.add(Dense(neurons, input_dim=X_train_pca.shape[1],
activation='relu', kernel_regularizer=l2(reg_param)))
```

```
    model.add(Dropout(dropout_rate))
    model.add(Dense(int(neurons/2), activation='relu',
kernel_regularizer=l2(reg_param)))
    model.add(Dropout(dropout_rate))
    model.add(Dense(1))  # Output layer for regression
    model.compile(optimizer=optimizer, loss='mean_squared_error')
    return model

# Create and fit the model directly
model = create_model()
early_stopping = EarlyStopping(patience=10, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(factor=0.2, patience=5)
history = model.fit(X_train_pca, y_train, validation_split=0.2,
epochs=100, batch_size=32,
                    callbacks=[early_stopping, reduce_lr], verbose=1)

# Evaluate the model
y_pred_ann = model.predict(X_test_pca)
mse_ann = mean_squared_error(y_test, y_pred_ann)
r2_ann = r2_score(y_test, y_pred_ann)
print(f"Best ANN Model MSE: {mse_ann}, R-squared: {r2_ann}")

# Build Random Forest model for comparison
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train_pca, y_train)
y_pred_rf = rf.predict(X_test_pca)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)
print(f"Random Forest MSE: {mse_rf}, R-squared: {r2_rf}")

# Voting Regressor (Ensemble of ANN and Random Forest)
voting_reg = VotingRegressor(estimators=[('ann', model), ('rf', rf)])
voting_reg.fit(X_train_pca, y_train)
y_pred_voting = voting_reg.predict(X_test_pca)
mse_voting = mean_squared_error(y_test, y_pred_voting)
r2_voting = r2_score(y_test, y_pred_voting)
print(f"Voting Regressor MSE: {mse_voting}, R-squared: {r2_voting}")

# Visualize the predicted vs actual values
plt.scatter(y_test, y_pred_ann, label='ANN Prediction', alpha=0.6)
plt.scatter(y_test, y_pred_rf, label='Random Forest Prediction',
alpha=0.6)
plt.scatter(y_test, y_pred_voting, label='Ensemble Prediction', alpha=0.6)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red')  # Line for perfect predictions
```

```
plt.xlabel('Actual Path Loss (dB)')
plt.ylabel('Predicted Path Loss (dB)')
plt.title('Actual vs Predicted Path Loss')
plt.legend()
plt.show()

# Plot learning curve of ANN model
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Comparison between ANN and Random Forest Model-

```
Best ANN Model MSE: 93.452340169541, R-squared: -0.7584937906676568
Random Forest MSE: 8.636447586206895, R-squared: 0.8374878635867328
```

Therefore, I think I should use Random Forest Model for later use!!!

Attempt 04:

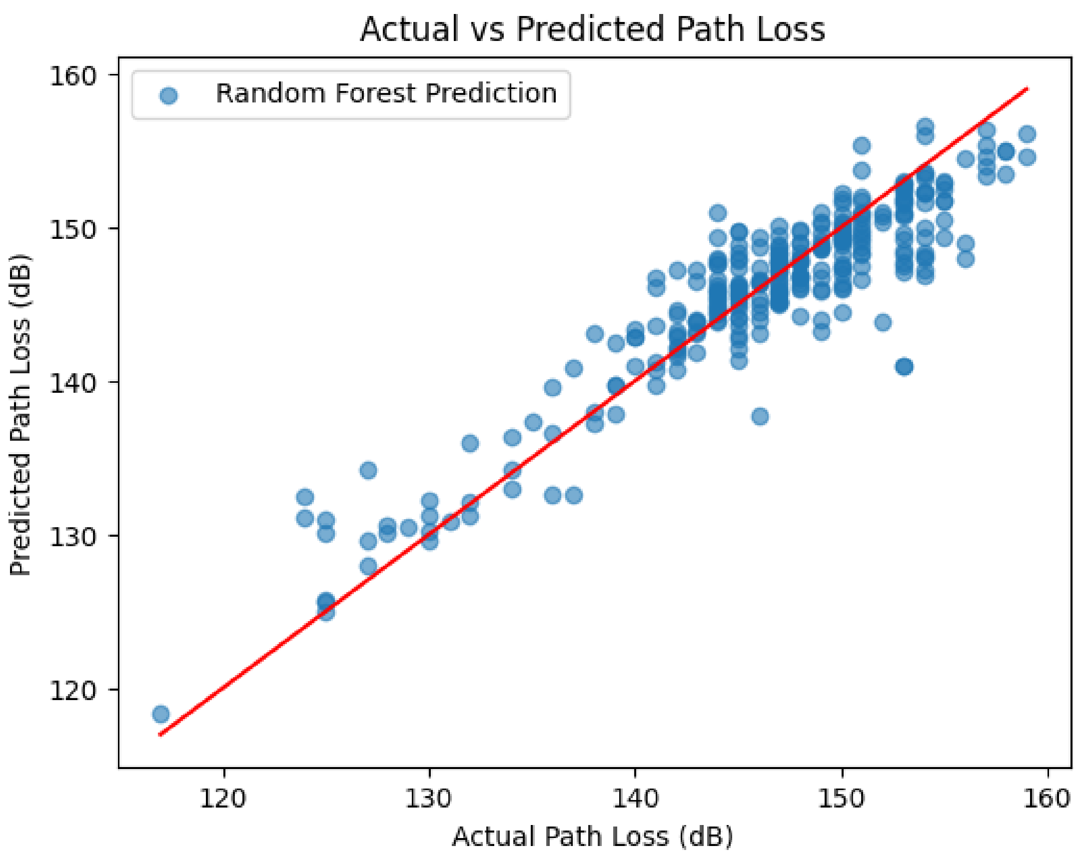
Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt

# Load dataset
data = pd.read_csv('sample1.csv')

# Define the input features and target variable
X = data[['Longitude', 'Latitude', 'Elevation (m)', 'Altitude (m)',
'Clutter height (m)', 'Distance (m)']]
y = data['Path Loss (dB)']

# Split the dataset 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)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply PCA
pca = PCA(n_components=6)  # Adjust n_components based on explained
variance
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Build Random Forest model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train_pca, y_train)

# Make predictions
y_pred_rf = rf.predict(X_test_pca)

# Evaluate the Random Forest model
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest MSE: {mse_rf}, R-squared: {r2_rf}")

# Visualize the predicted vs actual values
plt.scatter(y_test, y_pred_rf, label='Random Forest Prediction',
alpha=0.6)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red')  # Line for perfect predictions
plt.xlabel('Actual Path Loss (dB)')
plt.ylabel('Predicted Path Loss (dB)')
plt.title('Actual vs Predicted Path Loss')
plt.legend()
plt.show()
```

Output:

```
Random Forest MSE: 8.57674793103448, R-squared: 0.8386112327044588
```



Attempt 05: (Using ChatGPT generated Synthetic Data)

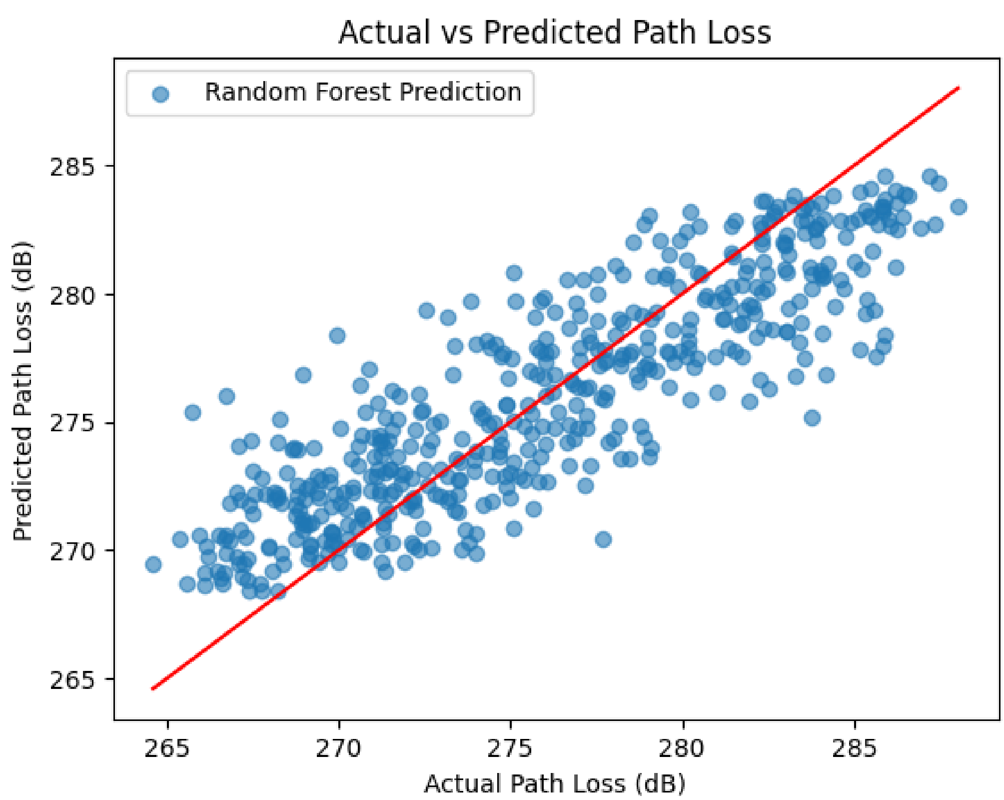
Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
import matplotlib.pyplot as plt

# Load dataset (using read_excel for .xlsx file)
data = pd.read_excel('/content/Processed_Satellite_Data.xlsx')

# Define the input features and target variable
# Ensure these column names align with your actual dataset
# X = data[['Longitude', 'Latitude', 'Elevation', 'Transmitter
# Define the input features with more columns from the dataset
```

```
X = data[['Longitude', 'Latitude', 'Elevation', 'Transmitter Height',
'Antenna Gain', 'Transmitter Power','Frequency Band', 'Polarization Type',
'Environment Type', 'Weather Conditions', 'Humidity Levels','Temperature',
'Atmospheric Pressure', 'Season', 'Clutter Height', 'Line of
Sight','Satellite Altitude', 'Satellite Position Latitude', 'Satellite
Position Longitude','Downlink Frequency', 'Satellite Antenna Gain', 'Beam
Type', 'Polarization Match','Space Weather Conditions', 'Power Settings',
'Transmission Mode', 'Path Length']]

#X = data[['Antenna Gain','Transmitter Power','Frequency
Band','Environment Type','Weather Conditions', 'Humidity
Levels','Temperature', 'Atmospheric Pressure', 'Season', 'Clutter Height',
'Line of Sight','Downlink Frequency', 'Satellite Antenna Gain', 'Beam
Type','Transmission Mode', 'Path Length']]
y = data['Path Loss']  # Update if Path Loss column needs renaming or
calculation


# One-Hot Encoding for categorical columns
X_encoded = pd.get_dummies(X, columns=['Frequency Band', 'Polarization
Type', 'Environment Type','Weather Conditions', 'Season', 'Line of
Sight','Beam Type', 'Polarization Match', 'Space Weather
Conditions','Transmission Mode'])


# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y,
test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply PCA
pca = PCA(n_components=27)  # Adjust n_components based on explained
variance
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Build Random Forest model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train_pca, y_train)

# Make predictions
```

```
y_pred_rf = rf.predict(X_test_pca)

# Evaluate the Random Forest model
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"Random Forest MSE: {mse_rf}, R-squared: {r2_rf}")

# Visualize the predicted vs actual values
plt.scatter(y_test, y_pred_rf, label='Random Forest Prediction',
alpha=0.6)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red')  # Line for perfect predictions
plt.xlabel('Actual Path Loss (dB)')
plt.ylabel('Predicted Path Loss (dB)')
plt.title('Actual vs Predicted Path Loss')
plt.legend()
plt.show()
```

Output:



```
Random Forest MSE: 9.026248311080696, R-squared: 0.7378370870261997
```

Attempt-06: (XGBoost vs Random Forest comparison)

Code:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
import matplotlib.pyplot as plt

# Load dataset (using read_excel for .xlsx file)
data = pd.read_excel('/content/Processed_Satellite_Data.xlsx')

# Define the input features and target variable
X = data[['Longitude', 'Latitude', 'Elevation', 'Transmitter Height',
'Antenna Gain', 'Transmitter Power',
          'Frequency Band', 'Polarization Type', 'Environment Type',
'Weather Conditions', 'Humidity Levels',
          'Temperature', 'Atmospheric Pressure', 'Season', 'Clutter
Height', 'Line of Sight',
          'Satellite Altitude', 'Satellite Position Latitude', 'Satellite
Position Longitude',
          'Downlink Frequency', 'Satellite Antenna Gain', 'Beam Type',
'Polarization Match',
          'Space Weather Conditions', 'Power Settings', 'Transmission
Mode', 'Path Length']]
y = data['Path Loss']  # Update if Path Loss column needs renaming or
calculation

# One-Hot Encoding for categorical columns
X_encoded = pd.get_dummies(X, columns=['Frequency Band', 'Polarization
Type', 'Environment Type',
                                       'Weather Conditions', 'Season',
'Line of Sight', 'Beam Type',
                                       'Polarization Match', 'Space
Weather Conditions', 'Transmission Mode'])

# Split the dataset into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y,
test_size=0.2, random_state=42)

# Feature scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Apply PCA
pca = PCA(n_components=27)  # Adjust n_components based on explained
variance
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Random Forest model
rf = RandomForestRegressor(n_estimators=100, random_state=42)
rf.fit(X_train_pca, y_train)
y_pred_rf = rf.predict(X_test_pca)

# Evaluate Random Forest model
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

# XGBoost model
xgb = XGBRegressor(n_estimators=100, random_state=42)
xgb.fit(X_train_pca, y_train)
y_pred_xgb = xgb.predict(X_test_pca)

# Evaluate XGBoost model
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

# Print evaluation metrics
print(f"Random Forest MSE: {mse_rf}, R-squared: {r2_rf}")
print(f"XGBoost MSE: {mse_xgb}, R-squared: {r2_xgb}")

# Plotting the results
plt.figure(figsize=(14, 6))

# Random Forest plot
plt.subplot(1, 2, 1)
plt.scatter(y_test, y_pred_rf, label='Random Forest Prediction',
alpha=0.6)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red')
```

```python
plt.xlabel('Actual Path Loss (dB)')
plt.ylabel('Predicted Path Loss (dB)')
plt.title('Random Forest: Actual vs Predicted Path Loss')
plt.legend()

# XGBoost plot
plt.subplot(1, 2, 2)
plt.scatter(y_test, y_pred_xgb, label='XGBoost Prediction', alpha=0.6)
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
color='red')
plt.xlabel('Actual Path Loss (dB)')
plt.ylabel('Predicted Path Loss (dB)')
plt.title('XGBoost: Actual vs Predicted Path Loss')
plt.legend()

plt.tight_layout()
plt.show()
```

<u>Output:</u>

```
Random Forest MSE: 9.002724948269005, R-squared: 0.7385203114517987
XGBoost MSE: 5.923123130116552, R-squared: 0.827965821437951
```