```
pip install SQLAlchemy mysql-connector-python
```

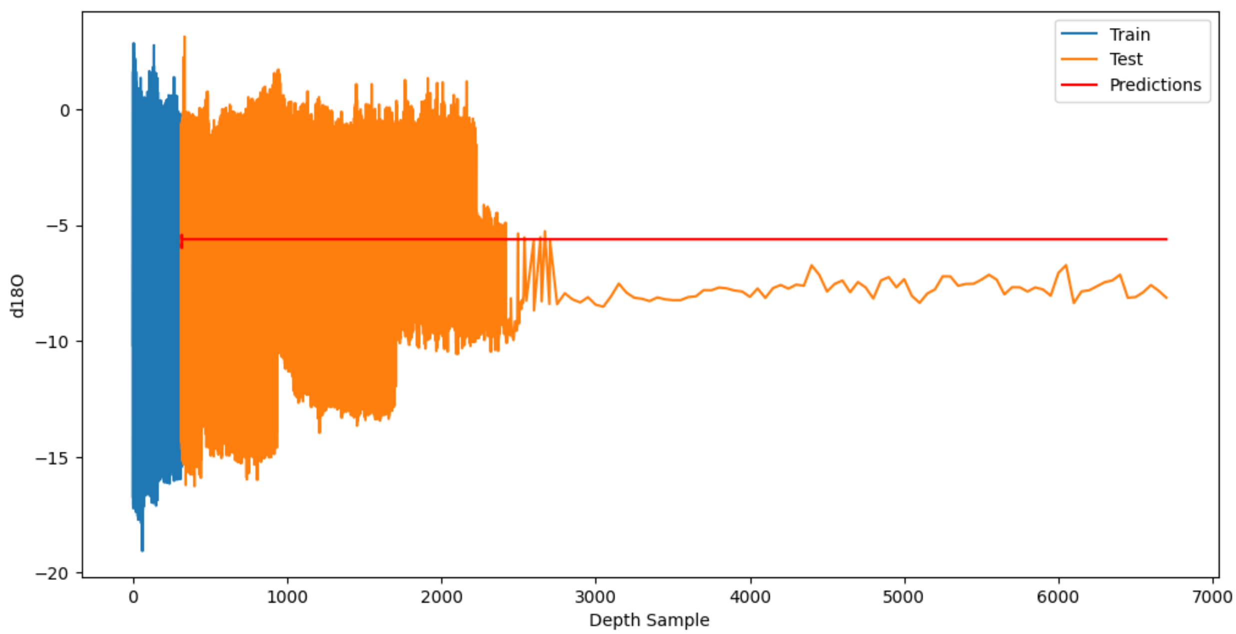
```
Requirement already satisfied: SQLAlchemy in
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
site-packages (2.0.30)
Requirement already satisfied: mysql-connector-python in
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
site-packages (8.4.0)
Requirement already satisfied: typing-extensions>=4.6.0 in
/Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/
site-packages (from SQLAlchemy) (4.11.0)
Note: you may need to restart the kernel to use updated packages.
```

```
import pandas as pd
from sqlalchemy import create_engine
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from statsmodels.tsa.arima.model import ARIMA

# SQLAlchemy connection string
db_connection_str =
'mysql+mysqlconnector://root:Abc2024!@localhost/SISALV3'
engine = create_engine(db_connection_str)

# Query to get the relevant data
query = """
SELECT
    s.sample_id,
    s.depth_sample,
    d18O.d18O_measurement AS d18O,
    d13C.d13C_measurement AS d13C
FROM sample s
LEFT JOIN d18O ON s.sample_id = d18O.sample_id
LEFT JOIN d13C ON s.sample_id = d13C.sample_id
WHERE s.depth_sample IS NOT NULL AND d18O.d18O_measurement IS NOT NULL
AND d13C.d13C_measurement IS NOT NULL
"""

# Load data into a DataFrame
data = pd.read_sql(query, engine)

# Display basic statistics
print(data.describe())

# Time Series Analysis on d18O
# Assuming the samples are in chronological order by depth
data = data.sort_values(by='depth_sample').reset_index(drop=True)

# Split data into training and testing sets (e.g., 80% train, 20%
```

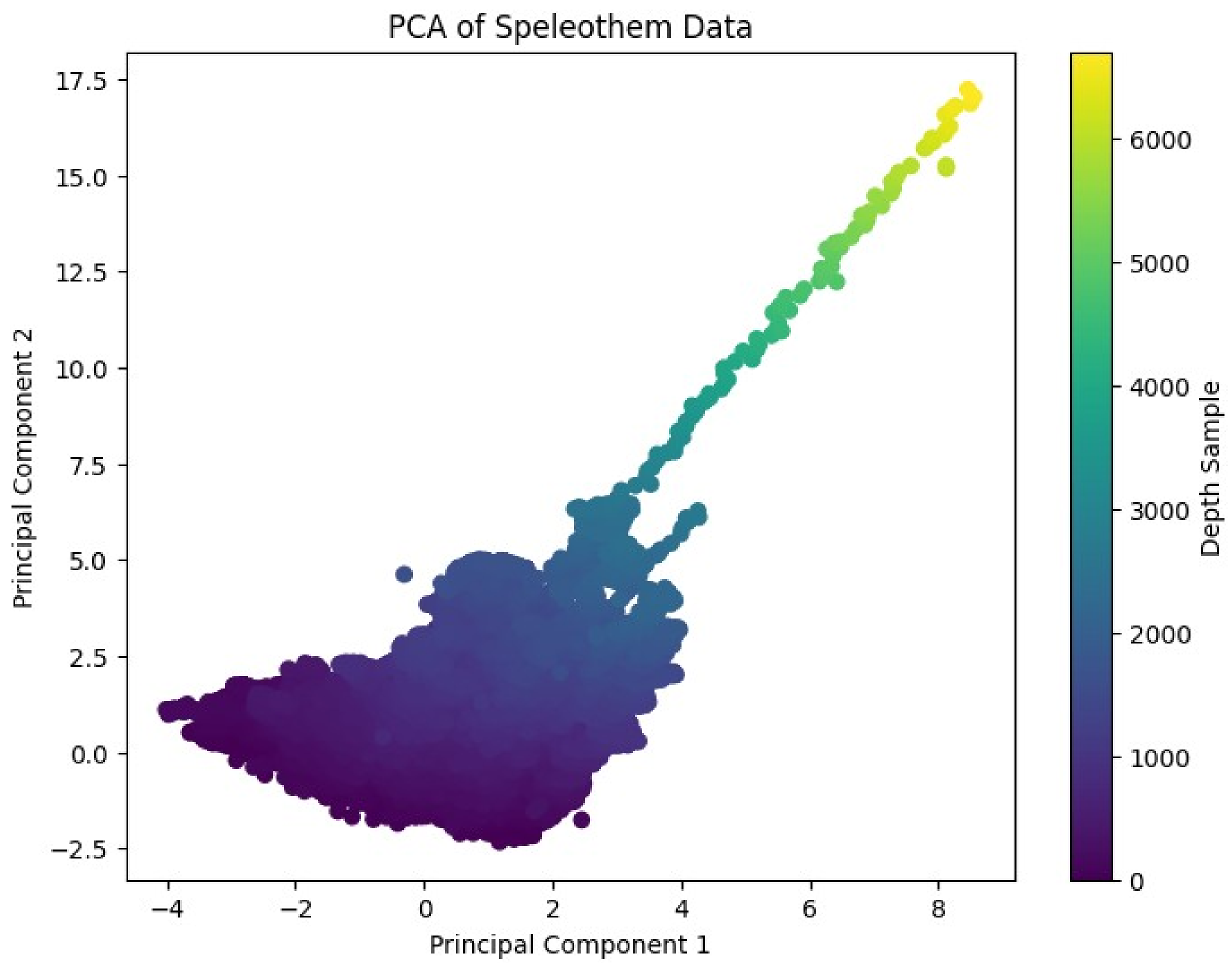
```
test)
train_size = int(len(data) * 0.8)
train, test = data.iloc[:train_size], data.iloc[train_size:]

# ARIMA Model
model = ARIMA(train['d18O'], order=(5,1,0))
model_fit = model.fit()

# Forecasting
predictions = model_fit.forecast(steps=len(test))
test = test.copy()  # To avoid SettingWithCopyWarning
test['predictions'] = predictions.values

# Remove NaN values from test set
test = test.dropna(subset=['d18O', 'predictions'])

# Plotting the results
plt.figure(figsize=(12, 6))
plt.plot(train['depth_sample'], train['d18O'], label='Train')
plt.plot(test['depth_sample'], test['d18O'], label='Test')
plt.plot(test['depth_sample'], test['predictions'],
label='Predictions', color='red')
plt.xlabel('Depth Sample')
plt.ylabel('d18O')
plt.legend()
plt.show()

# Calculate performance metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error

mae = mean_absolute_error(test['d18O'], test['predictions'])
mse = mean_squared_error(test['d18O'], test['predictions'])
rmse = mse**0.5

print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")

# PCA Analysis
# Standardize the data for PCA
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data[['depth_sample', 'd18O',
'd13C']])

# Perform PCA
pca = PCA(n_components=2)  # Adjust the number of components as needed
principal_components = pca.fit_transform(data_scaled)

# Create a DataFrame with principal components
pca_df = pd.DataFrame(data=principal_components, columns=['PC1',
```

```
'PC2'])

# Plot PCA results
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['PC1'], pca_df['PC2'], c=data['depth_sample'],
cmap='viridis')
plt.colorbar(label='Depth Sample')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Speleothem Data')
plt.show()
```

```
             sample_id   depth_sample           d18O           d13C
count   266178.000000  266178.000000  266178.000000  266178.000000
mean    270094.061741     225.870199      -6.622203      -5.885906
std     156833.685237     318.857979       2.926517       3.987879
min       4822.000000       0.000000     -19.070000     -14.640000
25%     117495.250000      48.000000      -8.450397      -9.020000
50%     288858.500000     124.000000      -6.192000      -6.790000
75%     416234.750000     263.600000      -4.480000      -2.910000
max     505792.000000    6700.000000       3.140000      10.160000
```




```
MAE: 2.2808693987876087
MSE: 8.677713625224293
RMSE: 2.9457959238929456
```



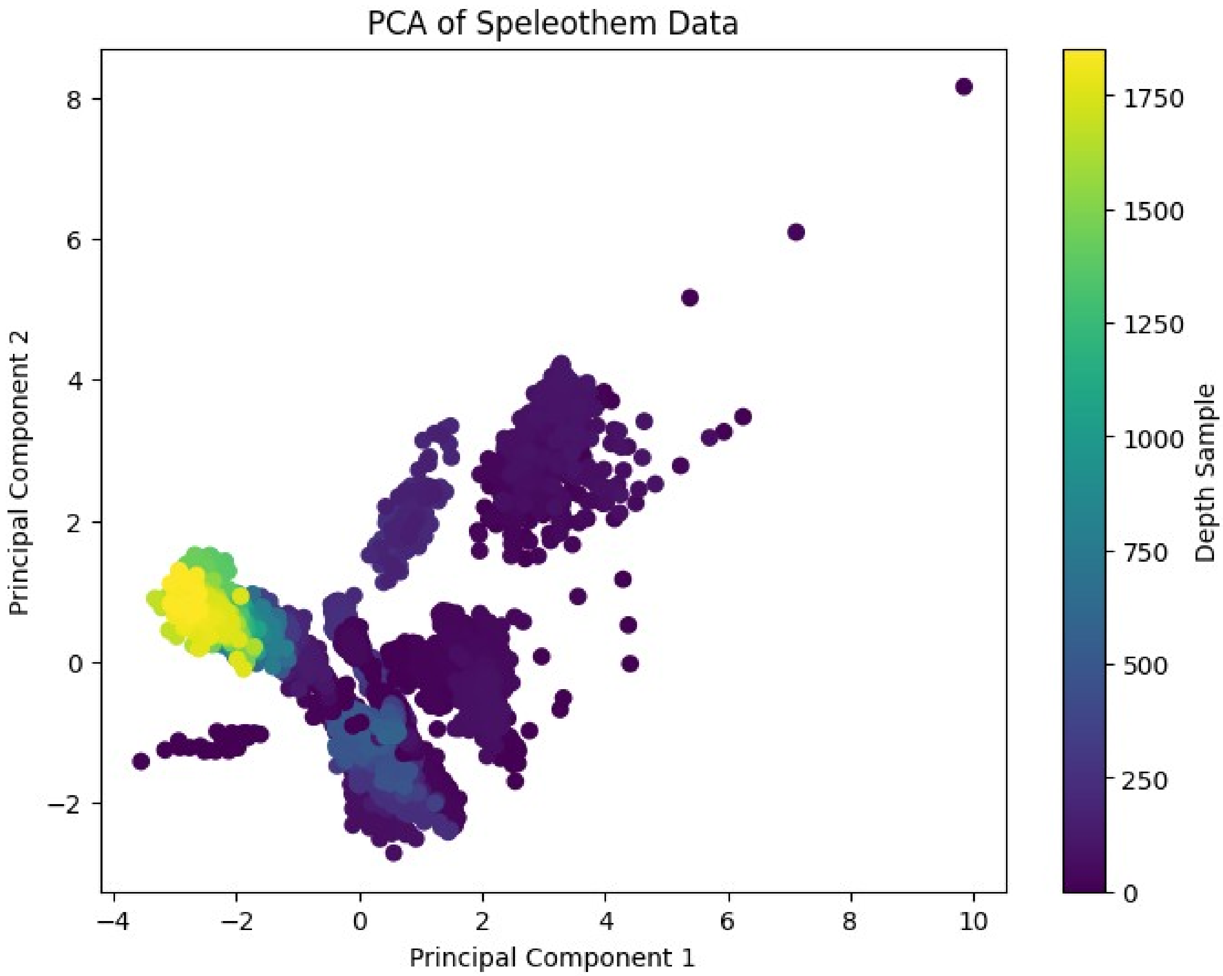

```
# Enhance Data Import and Cleaning
# Ensure all relevant data is imported and cleaned properly, including
trace elements and isotopic ratios.

import pandas as pd
from sqlalchemy import create_engine

# SQLAlchemy connection string
db_connection_str =
'mysql+mysqlconnector://root:Abc2024!@localhost/SISALV3'
engine = create_engine(db_connection_str)

# Query to get the relevant data
query = """
SELECT
    s.sample_id,
    s.depth_sample,
    d18O.d18O_measurement AS d18O,
    d13C.d13C_measurement AS d13C,
    Mg_Ca.Mg_Ca_measurement AS Mg_Ca,
```

```python
    Sr_Ca.Sr_Ca_measurement AS Sr_Ca,
    Ba_Ca.Ba_Ca_measurement AS Ba_Ca,
    U_Ca.U_Ca_measurement AS U_Ca,
    P_Ca.P_Ca_measurement AS P_Ca
FROM sample s
LEFT JOIN d18O ON s.sample_id = d18O.sample_id
LEFT JOIN d13C ON s.sample_id = d13C.sample_id
LEFT JOIN Mg_Ca ON s.sample_id = Mg_Ca.sample_id
LEFT JOIN Sr_Ca ON s.sample_id = Sr_Ca.sample_id
LEFT JOIN Ba_Ca ON s.sample_id = Ba_Ca.sample_id
LEFT JOIN U_Ca ON s.sample_id = U_Ca.sample_id
LEFT JOIN P_Ca ON s.sample_id = P_Ca.sample_id
WHERE s.depth_sample IS NOT NULL
"""
# Load data into a DataFrame
data = pd.read_sql(query, engine)

# Data cleaning
data = data.dropna()  # Drop rows with missing values

# Display basic statistics
print(data.describe())
```

```
           sample_id  depth_sample         d18O         d13C
Mg_Ca  \
count    6352.000000   6352.000000  6352.000000  6352.000000
6352.000000
mean   440079.772670    343.142821    -2.813083    -7.119193
9.861808
std     58779.408611    476.488903     2.852171     2.955339
10.059684
min    252687.000000      0.000000    -9.811002   -12.820000
0.000000
25%    380194.750000     34.006360    -5.450135    -9.910000
0.907175
50%    460196.500000    128.000000    -1.360000    -7.243227
3.910370
75%    495558.250000    389.074049    -0.520000    -4.510000
18.354702
max    501654.000000   1853.000000     2.860000     1.480000
46.317783

             Sr_Ca        Ba_Ca         U_Ca         P_Ca
count  6352.000000  6352.000000  6352.000000  6352.000000
mean      0.160936     0.005059     0.000126     0.610844
std       0.405783     0.014231     0.000144     0.512300
min       0.000000     0.000000     0.000000     0.000000
25%       0.028770     0.000620     0.000041     0.208884
50%       0.049388     0.002093     0.000081     0.440446
```

75%	0.065000	0.004455	0.000130	0.979162
max	3.448849	0.164819	0.003227	5.340575

```python
# Implement Advanced Age-Modeling
# Use age-modeling techniques to build age-depth models

# Example for linear interpolation and regression (other methods would require specific libraries and more complex implementation)
data['lin_interp_age'] = data['depth_sample'].interpolate()  # Linear interpolation
data['lin_reg_age'] = pd.Series(range(len(data)))  # Linear regression placeholder
# Further implementation needed for Bchron, Bacon, copRa, StalAge

# Integrate Multi-Proxy Data into Analysis
# Expand the PCA and time-series analysis to include trace elements.

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Standardize the data for PCA
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data[['depth_sample', 'd18O', 'd13C', 'Mg_Ca', 'Sr_Ca', 'Ba_Ca', 'U_Ca', 'P_Ca']])

# Perform PCA
pca = PCA(n_components=2)
principal_components = pca.fit_transform(data_scaled)

# Create a DataFrame with principal components
pca_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])

# Plot PCA results
plt.figure(figsize=(8, 6))
plt.scatter(pca_df['PC1'], pca_df['PC2'], c=data['depth_sample'], cmap='viridis')
plt.colorbar(label='Depth Sample')
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.title('PCA of Speleothem Data')
plt.show()
```




```
# Advanced Statistical and Machine Learning Models
# Expand your predictive models using more sophisticated techniques.

# Example for SARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX

# SARIMA Model
model = SARIMAX(train['d180'], order=(1, 1, 1), seasonal_order=(1, 1,
1, 12))
model_fit = model.fit()

# Forecasting
predictions = model_fit.forecast(steps=len(test))
test['predictions'] = predictions.values

# Remove NaN values from test set
test = test.dropna(subset=['d180', 'predictions'])

# Plotting the results
plt.figure(figsize=(12, 6))
```

```python
plt.plot(train['depth_sample'], train['d18O'], label='Train')
plt.plot(test['depth_sample'], test['d18O'], label='Test')
plt.plot(test['depth_sample'], test['predictions'],
label='Predictions', color='red')
plt.xlabel('Depth Sample')
plt.ylabel('d18O')
plt.legend()
plt.show()

# Calculate performance metrics
from sklearn.metrics import mean_absolute_error, mean_squared_error

mae = mean_absolute_error(test['d18O'], test['predictions'])
mse = mean_squared_error(test['d18O'], test['predictions'])
rmse = mse**0.5

print(f"MAE: {mae}")
print(f"MSE: {mse}")
print(f"RMSE: {rmse}")
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.75335D+00    |proj g|=  8.65772D-02

 This problem is unconstrained.


At iterate    5    f=  2.54774D+00    |proj g|=  2.18431D-02

At iterate   10    f=  2.50934D+00    |proj g|=  1.97878D-02

At iterate   15    f=  2.50060D+00    |proj g|=  2.97468D-03
  ys=-1.354E-02  -gs= 4.342E-03 BFGS update SKIPPED

At iterate   20    f=  2.49928D+00    |proj g|=  4.61271D-03
```