Configuration Manual

MSc Research Project
MSCDAD

Adeola Deborah Adeniji
Student ID: X23104201

School of Computing
National College of Ireland

Supervisor: Mr. David Hamill

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: Adeola Deborah Adeniji

Student ID: X23104201

Programme: MSc. Data Analytics **Year:** 2024

Module: MSc. Research Project

Supervisor: Mr. David Hamill

Submission Due Date: 12/12/2024

Project Title: Groundwater Level Forecasting: USA

Word Count: 446

Page Count: 10

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: Adeola Deborah Adeniji

Date: 11th December 2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Adeola Deborah Adeniji
X23104201

1 Introduction

This configuration manual provides a guide to the environment setup used for this project. It outlines a step-by-step preparation process necessary for similar implementation of the project titled “Groundwater Level Forecasting”.

2 System Configuration Requirements

This project executed on Jupyter notebook, under the Anaconda package management was compatible with Python programming language. The platform provided details about the operating system, such as the operating system name, release number, machine type, python version and the RAM. In Figure 1, the system information is shown.

Figure 1: System Information

```
# Get system information
print("Operating System:", platform.system())
print("OS Version:", platform.version())
print("OS Release:", platform.release())
print("Processor:", platform.processor())
print("Machine:", platform.machine())
print("Python Version:", platform.python_version())

# Check RAM
import psutil
print("Total RAM:", round(psutil.virtual_memory().total / (1024 * 1024 * 1024), 2), "GB")
```

```
Operating System: Windows
OS Version: 10.0.19045
OS Release: 10
Processor: Intel64 Family 6 Model 42 Stepping 7, GenuineIntel
Machine: AMD64
Python Version: 3.12.1
Total RAM: 7.88 GB
```

3 Environment Setup

The process of setting up of the environment involves launching the Anaconda Prompt to open the Jupyter Notebook for the project. Figure 2 shows the dictionary of the server.

Figure 2: Anacona Prompt

```
Anaconda Prompt - jupyter  notebook
Unable to create process using 'C:\ProgramData\anaconda3\python.exe C:\ProgramData\anaconda3\Scripts\conda-script.py shell.cmd.exe activate C:\Program

C:\Users\Dee>jupyter notebook
[W 2024-12-11 01:00:24.974 ServerApp] A `_jupyter_server_extension_points` function was not found in notebook_shim. Instead, a `_jupyter_server_extens
n was found and will be used for now. This function name will be deprecated in future releases of Jupyter Server.
[I 2024-12-11 01:00:25.025 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-12-11 01:00:25.053 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-12-11 01:00:25.104 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-12-11 01:00:25.132 ServerApp] notebook | extension was successfully linked.
[I 2024-12-11 01:00:26.279 ServerApp] notebook_shim | extension was successfully linked.
[I 2024-12-11 01:00:26.393 ServerApp] notebook_shim | extension was successfully loaded.
[I 2024-12-11 01:00:26.400 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2024-12-11 01:00:26.402 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2024-12-11 01:00:26.409 LabApp] JupyterLab extension loaded from C:\Users\Dee\AppData\Local\Programs\Python\Python312\Lib\site-packages\jupyterlab
[I 2024-12-11 01:00:26.410 LabApp] JupyterLab application directory is C:\Users\Dee\AppData\Local\Programs\Python\Python312\share\jupyter\lab
[I 2024-12-11 01:00:26.411 LabApp] Extension Manager is 'pypi'.
[I 2024-12-11 01:00:26.416 ServerApp] jupyterlab | extension was successfully loaded.
[I 2024-12-11 01:00:26.428 ServerApp] notebook | extension was successfully loaded.
[I 2024-12-11 01:00:26.429 ServerApp] Serving notebooks from local directory: C:\Users\Dee
[I 2024-12-11 01:00:26.430 ServerApp] Jupyter Server 2.12.5 is running at:
[I 2024-12-11 01:00:26.430 ServerApp] http://localhost:8888/tree?token=969c19bb57b9d07d4b28fbfccce3c9887e261d78bba3f4d2
[I 2024-12-11 01:00:26.430 ServerApp]     http://127.0.0.1:8888/tree?token=969c19bb57b9d07d4b28fbfccce3c9887e261d78bba3f4d2
[I 2024-12-11 01:00:26.431 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 2024-12-11 01:00:26.556 ServerApp]

    To access the server, open this file in a browser:
        file:///C:/Users/Dee/AppData/Roaming/jupyter/runtime/jpserver-17064-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/tree?token=969c19bb57b9d07d4b28fbfccce3c9887e261d78bba3f4d2
        http://127.0.0.1:8888/tree?token=969c19bb57b9d07d4b28fbfccce3c9887e261d78bba3f4d2
```

4 Installation of Python Packages and Libraries

The installation of necessary packages used for a smooth workflow in the statistical analysis and time series modelling, are listed follows. The important packages installed are shown in Figure 3, while Figure 4 shows a snippet the important libraries used.

- Platform
- OS
- warnings
- Pmdarima
- Statsmodels
- Pandas
- Seaborn
- NumPy
- Matplotlib

Figure 3: Packages Installed

```
# Install Packages
!pip install gputil
!pip install statsmodels
!pip install pmdarima
```

Figure 4: Libraires Installation

```
# Install Liberies

import platform
import os
import GPUtil
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.api as sm
from datetime import datetime
from statsmodels.tsa.seasonal import seasonal_decompose
from pmdarima import auto_arima
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from statsmodels.tsa.holtwinters import SimpleExpSmoothing
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from statsmodels.tsa.arima.model import ARIMA
```

5 Implementation Stage Explained

The code snippet in Figure 4 shows the successful collection of the groundwater level dataset[1] from Kaggle repository, and license for usage by the California Department of Water Resources.

Figure 4: Data collection

```
def collect_data(file_path="gwl-daily.csv"):
    try:
        GWL = pd.read_csv(file_path)
        print("Groundwater level data has been collected successfully.")
        return GWL
    except FileNotFoundError:
        print("File not found. Please check the file path.")
GWL = collect_data()

Groundwater level data has been collected successfully.
```

[1] https://www.kaggle.com/datasets/alifarahmandfar/continuous-groundwater-level-measurements-2023/data

The code snippet in Figure 5 explores the characteristics of the groundwater level dataset.

Figure 5: Data Understanding.

```
# DISPLAY THE FIRST 5 ROWS ON GWL DATA
GWL.head()
```

	STATION	MSMT_DATE	WLM_RPE	WLM_RPE_QC	WLM_GSE	WLM_GSE_QC	RPE_WSE	RPE_WSE_QC	GSE_WSE	GSE_WSE_QC	WSE	WSE_QC
0	14N01E35P001M	12/27/2022	48.74	1	46.88	1	42.442	2	40.582	2	6.298	2
1	14N01E35P004M	12/27/2022	47.62	1	46.88	1	20.786	2	20.046	2	26.834	2
2	16N03W14H004M	12/27/2022	68.21	1	65.70	1	19.735	2	17.225	2	48.475	2
3	14N01E35P001M	12/26/2022	48.74	1	46.88	1	42.552	2	40.692	2	6.188	2
4	14N01E35P004M	12/26/2022	47.62	1	46.88	1	20.829	2	20.089	2	26.791	2

```
# DISPLAY THE LAST 5 ROWS OF GWL DATA
GWL.tail()
```

	STATION	MSMT_DATE	WLM_RPE	WLM_RPE_QC	WLM_GSE	WLM_GSE_QC	RPE_WSE	RPE_WSE_QC	GSE_WSE	GSE_WSE_QC	WSE	WSE_QC
1048570	09N03E08C004M	3/26/1992	32.056	2	30.016	2	22.165	1	20.125	2	9.891	2
1048571	09N03E08C001M	3/25/1992	30.386	2	30.016	2	9.263	1	8.893	2	21.123	2
1048572	09N03E08C002M	3/25/1992	30.226	2	30.016	2	9.438	1	9.228	2	20.788	2
1048573	09N03E08C003M	3/25/1992	30.186	2	30.016	2	16.279	1	16.109	2	13.907	2
1048574	09N03E08C004M	3/25/1992	32.056	2	30.016	2	22.181	1	20.141	2	9.875	2

The code snippet in Figure 6 show part of the pre-processing and transformation steps involved.

Figure 6: Data Pre-Processing and Transformation.

```
# Remove duplicate rows based on all columns
GWL = GWL.drop_duplicates()
GWL.shape
```

```
(954445, 12)
```

Stations Selection

```
def get_individual_station_dfs(df, station_list):
    station_dfs = {}
    for station in station_list:
        station_dfs[station] = df[df['STATION'] == station]
    return station_df
```

```
station_list = ["09N03E08C001M", "09N03E08C004M", "09N03E08C003M", "09N03E08C002M", "11N04E04N004M"]
station_dfs = get_individual_station_dfs(GWL, station_list)
```

```
station_a = station_dfs["09N03E08C001M"]
station_b = station_dfs["09N03E08C004M"]
station_c = station_dfs["09N03E08C003M"]
station_d = station_dfs["09N03E08C002M"]
station_e = station_dfs["11N04E04N004M"]
```

The code snippet in Figure 7 shows an important step, where the column features are normalized, while Figure 8 provides the information about each station.

Figure 7: Data Pre-Processing and Transformation Code.

Standardization

```
def standardize_df(df):
    scaler = StandardScaler()
    df[['WLM_RPE', 'WLM_GSE', 'RPE_WSE', 'GSE_WSE']] = scaler.fit_transform(df[['WLM_RPE', 'WLM_GSE', 'RPE_WSE', 'GSE_WSE']])
    return df

station_a = standardize_df(station_a)
station_b = standardize_df(station_b)
station_c = standardize_df(station_c)
station_d = standardize_df(station_d)
station_e = standardize_df(station_e)
```

Figure 8: Code Information about each Station A – E

```
# Total number of rows and columns in each station
print(station_a.shape)
print(station_b.shape)
print(station_c.shape)
print(station_d.shape)
print(station_e.shape)
```

```
(11019, 6)
(10851, 6)
(10847, 6)
(10571, 6)
(10368, 6)
```

```
print(station_a.columns)
print(station_b.columns)
print(station_c.columns)
print(station_d.columns)
print(station_e.columns)
```

```
Index(['MSMT_DATE', 'WLM_RPE', 'WLM_GSE', 'RPE_WSE', 'GSE_WSE', 'WSE'], dtype='object')
Index(['MSMT_DATE', 'WLM_RPE', 'WLM_GSE', 'RPE_WSE', 'GSE_WSE', 'WSE'], dtype='object')
Index(['MSMT_DATE', 'WLM_RPE', 'WLM_GSE', 'RPE_WSE', 'GSE_WSE', 'WSE'], dtype='object')
Index(['MSMT_DATE', 'WLM_RPE', 'WLM_GSE', 'RPE_WSE', 'GSE_WSE', 'WSE'], dtype='object')
Index(['MSMT_DATE', 'WLM_RPE', 'WLM_GSE', 'RPE_WSE', 'GSE_WSE', 'WSE'], dtype='object')
```

Figure 9: Descriptive Code for Surface Water Above Ground (WSE)

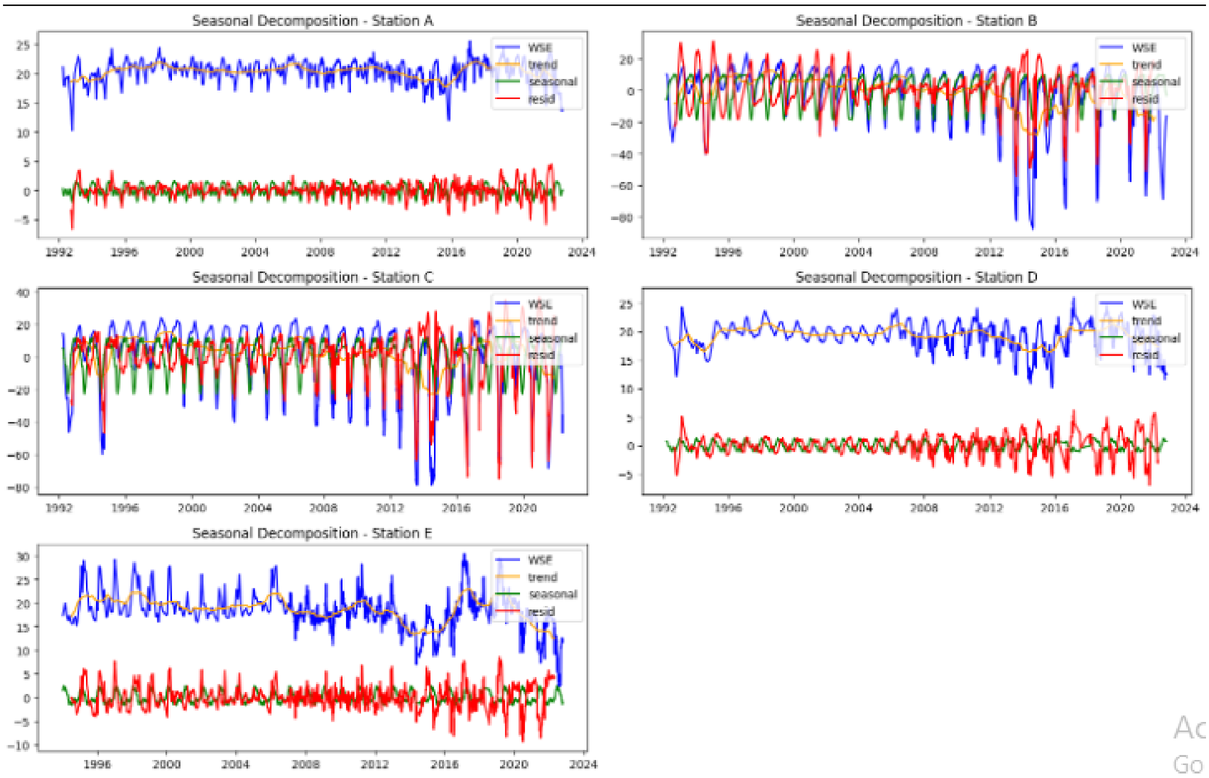
```
stations = [station_a, station_b, station_c, station_d, station_e]
titles = ['Station A', 'Station B', 'Station C', 'Station D', 'Station E']
features = ['WSE', 'WLM_RPE', 'WLM_GSE', 'RPE_WSE', 'GSE_WSE']
```

```
# Plotting Surface Water Elevation (WSE)

plt.figure(figsize=(10, 10))
for i, station in enumerate(stations):
    plt.subplot(3, 2, i + 1)
    plt.plot(station.index, station['WSE'], label='Surface Water Above', color='blue')
    plt.title(f'Surface Water Above Ground\n{titles[i]}')
    plt.xlabel('Index')
    plt.ylabel('Elevation (meters)')
    plt.legend()
    plt.grid(True)

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

Figure 10: Code for Seasonal Decomposition for all Stations

```
plt.figure(figsize=(15, 10))

for i, station in enumerate(stations):
    # Decompose WSE time series
    decomposition = seasonal_decompose(station['WSE'], model='additive', period=365)

    plt.subplot(3, 2, i + 1)
    for component, color in zip([decomposition.observed, decomposition.trend,
                                 decomposition.seasonal, decomposition.resid],
                                ['blue', 'orange', 'green', 'red']):
        plt.plot(component, label=component.name, color=color)

    plt.title(f"Seasonal Decomposition - {titles[i]}")
    plt.legend(loc='upper right')

plt.tight_layout(rect=[0, 0, 1, 0.96])
plt.show()
```

Figure 11: Result for Seasonal Decomposition in all Stations



The code snippet in Figure 12 shows how the data was spitted, while Figure 13 code snippet ensures checking the vital signs of Time Series Data

Figure 12: Data Splitting

```python
for station in stations:
    station.sort_values(by='MSMT_DATE', inplace=True)

def train_test_split_all(stations, train_ratio=0.8):
    return {f"Station_{chr(65 + i)}":
            {
            'train': df.iloc[:int(len(df) * train_ratio)].reset_index(drop=True),
            'test': df.iloc[int(len(df) * train_ratio):].reset_index(drop=True)

            }

            for i, df in enumerate(stations)}

splits = train_test_split_all(stations)

# Display the ranges for verification
for station, split in splits.items():

    split['train'] = split['train'].set_index('MSMT_DATE')
    split['test'] = split['test'].set_index('MSMT_DATE')

    print(f"{station}:")

    print(f"  Training Set: {split['train'].index.min()} to {split['train'].index.max()}")
    print(f"  Testing Set: {split['test'].index.min()} to {split['test'].index.max()}\n")
```

Figure 13: Code used in Checking Vital Signs of Time Series Data

```python
# Ensure 'MSMT_DATE' is a datetime type
station_a['MSMT_DATE'] = pd.to_datetime(station_a['MSMT_DATE'])
station_b['MSMT_DATE'] = pd.to_datetime(station_b['MSMT_DATE'])
station_c['MSMT_DATE'] = pd.to_datetime(station_c['MSMT_DATE'])
station_d['MSMT_DATE'] = pd.to_datetime(station_d['MSMT_DATE'])
station_e['MSMT_DATE'] = pd.to_datetime(station_e['MSMT_DATE'])

# Filter the data to start from March 1992 and resample with monthly frequency
start_date = '1992-03-01'

# For each station, filter the data, set 'MSMT_DATE' as the index, and resample to monthly frequency
station_a_ts = station_a[station_a['MSMT_DATE'] >= start_date].set_index('MSMT_DATE')['WSE'].resample('M').last()
station_b_ts = station_b[station_b['MSMT_DATE'] >= start_date].set_index('MSMT_DATE')['WSE'].resample('M').last()
station_c_ts = station_c[station_c['MSMT_DATE'] >= start_date].set_index('MSMT_DATE')['WSE'].resample('M').last()
station_d_ts = station_d[station_d['MSMT_DATE'] >= start_date].set_index('MSMT_DATE')['WSE'].resample('M').last()
station_e_ts = station_e[station_e['MSMT_DATE'] >= start_date].set_index('MSMT_DATE')['WSE'].resample('M').last()
```

```python
# Checking the type of data for each station (after resampling)
print("Station A Type: ", type(station_a_ts))
print("Station B Type: ", type(station_b_ts))
print("Station C Type: ", type(station_c_ts))
print("Station D Type: ", type(station_d_ts))
print("Station E Type: ", type(station_e_ts))
```

```
Station A Type:  <class 'pandas.core.series.Series'>
Station B Type:  <class 'pandas.core.series.Series'>
Station C Type:  <class 'pandas.core.series.Series'>
Station D Type:  <class 'pandas.core.series.Series'>
Station E Type:  <class 'pandas.core.series.Series'>
```

```python
# Checking the index type for each station to confirm it's datetime
print("Station A Index Type: ", station_a_ts.index.dtype)
print("Station B Index Type: ", station_b_ts.index.dtype)
print("Station C Index Type: ", station_c_ts.index.dtype)
print("Station D Index Type: ", station_d_ts.index.dtype)
print("Station E Index Type: ", station_e_ts.index.dtype)
```

Development of the Time Series Models, consisted of Naïve Base shoown in Figure 14, the Drift Method in Figure 15, the Simple Exponential Smoothing Method in Figure 16, the Holt-Winter Method shown in Figure 17 and the proposed Arima Method shown in Figure 18

Figure 14: Naïve Base Method Code

```
def naive_forecast(train, test, horizon=2920): # 2920 = 8year * 365days
    last_obs = train.iloc[-1]
    forecast = np.repeat(last_obs, len(test))
    future_forecast = np.repeat(last_obs, horizon)
    metrics = {
        "rmse": np.sqrt(mean_squared_error(test, forecast)),
        "mae": mean_absolute_error(test, forecast),
        "r2": r2_score(test, forecast),
        "mape": np.mean(np.abs((test - forecast) / test)) * 100,
        "acf1": test.autocorr(lag=1),
        "aic": len(test) * np.log(mean_squared_error(test, forecast)) + 2
    }
    return forecast, future_forecast, metrics

results = {
    name: naive_forecast(split['train']['WSE'], split['test']['WSE'])
    for name, split in splits.items()
}

for name, (forecast, future_forecast, metrics) in results.items():
    print(f"{name}: RMSE={metrics['rmse']:.2f}, MAE={metrics['mae']:.2f}, R^2={metrics['r2']:.2f}, "
          f"MAPE={metrics['mape']:.2f}%, ACF1={metrics['acf1']:.2f}, AIC={metrics['aic']:.2f}")
    print(f"Future Forecast (8 years): {future_forecast}\n")
```

Figure 15: Drift Method Code

```
def drift_forecast(train, test, horizon=2920): # 2920 = 8year * 365days
    n = len(train)
    drift = (train.iloc[-1] - train.iloc[0]) / (n - 1) if n > 1 else 0
    forecast = train.iloc[-1] + drift * np.arange(1, len(test) + 1)
    future_forecast = train.iloc[-1] + drift * np.arange(1, horizon + 1)
    metrics = {
        "rmse": np.sqrt(mean_squared_error(test, forecast)),
        "mae": mean_absolute_error(test, forecast),
        "r2": r2_score(test, forecast),
        "mape": np.mean(np.abs((test - forecast) / test)) * 100,
        "acf1": test.autocorr(lag=1),
        "aic": len(test) * np.log(mean_squared_error(test, forecast)) + 2
    }
    return forecast, future_forecast, metrics

# Apply drift_forecast for all stations
results_drift = {
    name: drift_forecast(split['train']['WSE'], split['test']['WSE'])
    for name, split in splits.items()
}

# Print results
for name, (forecast, future_forecast, metrics) in results_drift.items():
    print(f"{name}: RMSE={metrics['rmse']:.2f}, MAE={metrics['mae']:.2f}, R^2={metrics['r2']:.2f}, "
          f"MAPE={metrics['mape']:.2f}%, ACF1={metrics['acf1']:.2f}, AIC={metrics['aic']:.2f}")
    print(f"Future Forecast (8 years): {future_forecast}\n")
```

Figure 16: Simple Exponential Smoothing Method Code

```python
def simple_exp_smoothing_forecast(train, test, horizon=2920): # 2920 = 8year * 365days
    model = SimpleExpSmoothing(train).fit()
    forecast = model.forecast(len(test))
    future_forecast = model.forecast(horizon)

    metrics = {
        "rmse": np.sqrt(mean_squared_error(test, forecast)),
        "mae": mean_absolute_error(test, forecast),
        "r2": r2_score(test, forecast),
        "mape": np.mean(np.abs((test - forecast) / test)) * 100,
        "acf1": test.autocorr(lag=1),
        "aic": model.aic
    }

    # Return the forecast and metrics in compact form
    return forecast, future_forecast.values, metrics  # Use .values to get the array

# Apply the function for all stations
results_exp_smoothing = {
    name: simple_exp_smoothing_forecast(split['train']['WSE'], split['test']['WSE'])
    for name, split in splits.items()
}

# Print results in the desired format
for name, (forecast, future_forecast, metrics) in results_exp_smoothing.items():
    print(f"{name}: RMSE={metrics['rmse']:.2f}, MAE={metrics['mae']:.2f}, R^2={metrics['r2']:.2f}, "
          f"MAPE={metrics['mape']:.2f}%, ACF1={metrics['acf1']:.2f}, AIC={metrics['aic']:.2f}")
    print(f"Future Forecast (8 years): {future_forecast}\n")
```

Figure 17: Holt-Winter Method Code

```python
def holt_winters_forecast(train, test, horizon=2920): # 2920 = 8year * 365days
    # Fit the Holt-Winter model (Additive or Multiplicative)
    model = ExponentialSmoothing(train, trend='add', seasonal='add', seasonal_periods=12).fit()
    forecast = model.forecast(len(test))
    future_forecast = model.forecast(horizon)

    # Calculate evaluation metrics
    metrics = {
        "rmse": np.sqrt(mean_squared_error(test, forecast)),
        "mae": mean_absolute_error(test, forecast),
        "r2": r2_score(test, forecast),
        "mape": np.mean(np.abs((test - forecast) / test)) * 100,
        "acf1": test.autocorr(lag=1),
        "aic": model.aic
    }

    # Return the forecast and future forecast values as a compact numpy array
    return forecast, future_forecast.values, metrics

# Apply the function for all stations
results_holt_winters = {
    name: holt_winters_forecast(split['train']['WSE'], split['test']['WSE'])
    for name, split in splits.items()
}

# Print results in the desired format
for name, (forecast, future_forecast, metrics) in results_holt_winters.items():
    print(f"{name}: RMSE={metrics['rmse']:.2f}, MAE={metrics['mae']:.2f}, R^2={metrics['r2']:.2f}, "
          f"MAPE={metrics['mape']:.2f}%, ACF1={metrics['acf1']:.2f}, AIC={metrics['aic']:.2f}")
    print(f"Future Forecast (8 years): {future_forecast}\n")
```

Figure 18: Arima Method Code

```python
def arima_forecast(train, test, horizon=2920, order=(2,1,2)): # 2920 = 8year * 365days
    # Fit the ARIMA model (order: p, d, q)
    model = ARIMA(train, order=order)
    model_fit = model.fit()

    # Forecast on the test data
    forecast = model_fit.forecast(len(test))

    # Forecast for future (horizon years)
    future_forecast = model_fit.forecast(horizon)

    # Calculate evaluation metrics
    metrics = {
        "rmse": np.sqrt(mean_squared_error(test, forecast)),
        "mae": mean_absolute_error(test, forecast),
        "r2": r2_score(test, forecast),
        "mape": np.mean(np.abs((test - forecast) / test)) * 100,
        "acf1": test.autocorr(lag=1),
        "aic": model_fit.aic
    }

    return forecast, future_forecast, metrics

results_arima = {
    name: arima_forecast(split['train']['WSE'], split['test']['WSE'])
    for name, split in splits.items()
}

for name, (forecast, future_forecast, metrics) in results_arima.items():
    print(f"{name}: RMSE={metrics['rmse']:.2f}, MAE={metrics['mae']:.2f}, R^2={metrics['r2']:.2f}, "
          f"MAPE={metrics['mape']:.2f}%, ACF1={metrics['acf1']:.2f}, AIC={metrics['aic']:.2f}")

    future_forecast.index = pd.date_range(
        start = splits[name]['test'].index[-1],
        periods=len(future_forecast), freq='D'
    )

    print(f"Future Forecast (8 years): {future_forecast}\n")
```