Final Project: Groundwater Level Projection in Penobscot County, Maine

---

Abstract

This project analyzes historical and projected groundwater levels in Penobscot County, Maine, under climate change. Daily data from 2005 to 2024 were explored using unsupervised learning (PCA, K-Means) and modeled with supervised methods (Linear Regression, Neural Network, Random Forest, Support Vector Machine). The Neural Network performed best and was used to project groundwater levels through 2099 using NASA's NEX-GDDP RCP8.5 data. Results suggest a gradual deepening of groundwater levels, indicating reduced recharge during wet periods and limited recovery during dry years under a high-emissions scenario.

Contents

---

1 Introduction

Climate change is reshaping hydrological systems worldwide, with significant implications for groundwater resources. Rising temperatures and shifting precipitation patterns can alter recharge rates, lower water tables, and increase the risk of water scarcity in vulnerable regions [1]. Among the tools used to anticipate such changes are the Representative Concentration Pathways (RCPs) developed by the Intergovernmental Panel on Climate Change (IPCC), which model future climate conditions based on varying greenhouse gas emission scenarios [2].

One of the most widely used scenarios, RCP8.5, represents a high-emissions trajectory that is commonly applied in groundwater impact studies across different regions, such as Germany and Iran [3,4]. These studies often pair CMIP (Coupled Model Intercomparison Project) outputs with local climate and hydrologic data to assess potential future risks under severe climate conditions.

Building on this framework, my project focuses on Penobscot County, Maine, a region in the northeastern United States with rich hydroclimatic records and a growing need to understand local climate vulnerabilities. By integrating daily historical measurements of air temperature, precipitation, and groundwater levels, and applying climate projections from NASA's NEX-GDDP RCP8.5 dataset, this project aims to estimate future trends in groundwater levels within the region. To support this objective, the project applies unsupervised learning techniques to explore the data, followed by supervised learning models to project groundwater levels through the end of the 21st century.

---

2 Methodology

2.1 Data

This project incorporates four types of data: **air temperature**, **precipitation**, **groundwater levels**, and **RCP climate scenarios**. The first three data are used for model training and testing, while the RCP data is used exclusively for future projections of groundwater levels.

2.1.1 Air Temperature & Precipitation

Daily weather data were obtained from the **NOAA** Global Historical Climatology Network – Daily (GHCND) dataset for the station **BANGOR INTERNATIONAL AIRPORT, ME US**, located at 44.80°N, 68.82°W and 147.3 feet in elevation, with data from `2005-01-01` to `2024-12-31` used in this project [5].

2.1.2 Groundwater Levels

Groundwater levels were retrieved via the **USGS** National Water Information System (NWIS) API. The well site (ID: *445319068560101*) is located in **Kenduskeag, Maine** at 44.89°N, 68.93°W, with a well depth of 101 feet and an elevation of 194 feet [6]. For this project, data from `2005-01-01` to `2024-12-31` were selected to align with the time range used for air temperature and precipitation data. It is important to note that the measurements represent the depth to groundwater below the land surface, where higher values indicate deeper (i.e., lower) groundwater levels.

This groundwater well site is relatively close to the Bangor weather station (only 8.5 miles apart) and both located in **Penobscot County**. Additionally, this well has fewer missing data compared to other groundwater monitoring sites. For these reasons, the site was chosen for groundwater level analysis in this project.

2.1.3 RCP Climate Scenarios

Future projections were based on the **NASA** Earth Exchange Global Daily Downscaled Projections (NEX-GDDP) dataset, which provides CMIP5-derived, downscaled daily climate data under RCP 4.5 and RCP 8.5 scenarios. This dataset includes daily maximum temperature, minimum temperature, and precipitation from 1950 to 2100, with a spatial resolution

of 0.25° (~25 km) [7].

This project uses the **CP8.5** scenario from the GFDL-CM3 model, with data covering the period from `2025-01-01` to `2099-12-31`, extracted for the location 44.92°N, 68.97°W via OpenDAP and processed using Python.

2.2 Modeling

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
import warnings
warnings.filterwarnings("ignore")
```

2.2.1 Data Collection

- **Air Temperature & Precipitation:** Downloaded from the **NOAA** GHCND dataset as a CSV file (pre-downloaded).
- **Groundwater Levels:** Retrieved via the **USGS** NWIS API using the `dataretrieval` Python package.
- **RCP8.5 Climate Data:** Extracted from **NASA** NEX-GDDP using OpenDAP and processed with `xarray`.

In [2]:
```
# Load daily weather data from NOAA (GHCND dataset)
df_weather = pd.read_csv("4015627.csv")

df_weather
```

Out[2]:

	STATION	NAME	DATE	PRCP	SNOW	SNWD	TAVG	TMAX	TMIN
0	USW00014606	BANGOR INTERNATIONAL AIRPORT, ME US	2005-01-01	0.15	0.0	0.0	34.0	46.0	22.0
1	USW00014606	BANGOR INTERNATIONAL AIRPORT, ME US	2005-01-02	0.24	1.0	0.0	23.0	33.0	12.0
2	USW00014606	BANGOR INTERNATIONAL AIRPORT, ME US	2005-01-03	0.05	0.0	1.0	37.0	42.0	32.0
3	USW00014606	BANGOR INTERNATIONAL AIRPORT, ME US	2005-01-04	0.00	0.0	0.0	30.0	34.0	25.0
4	USW00014606	BANGOR INTERNATIONAL AIRPORT, ME US	2005-01-05	0.00	0.0	0.0	20.0	30.0	10.0
...	...	...	...	...	...	...	...	...	...
20332	USW00014610	MILLINOCKET MUNICIPAL AIRPORT, ME US	2025-05-01	0.00	NaN	NaN	NaN	64.0	28.0
20333	USW00014610	MILLINOCKET MUNICIPAL AIRPORT, ME US	2025-05-02	0.09	NaN	NaN	NaN	49.0	43.0
20334	USW00014610	MILLINOCKET MUNICIPAL AIRPORT, ME US	2025-05-03	0.17	NaN	NaN	NaN	68.0	49.0
20335	USW00014610	MILLINOCKET MUNICIPAL AIRPORT, ME US	2025-05-04	0.22	NaN	NaN	NaN	61.0	49.0
20336	USW00014610	MILLINOCKET MUNICIPAL AIRPORT, ME US	2025-05-05	0.00	NaN	NaN	NaN	69.0	44.0

20337 rows × 9 columns

In [3]:
```
from dataretrieval import nwis
```

In [4]:
```
# Define USGS site ID (Kenduskeag, Maine)
site = "445319068560101"

# Download daily groundwater level data from USGS NWIS
df_gwl = nwis.get_record(sites=site, service='dv', start='2005-01-01', end='2024-12-31').reset_index()

df_gwl
```

Out[4]:

	datetime	site_no	72019_Mean	72019_Mean_cd
0	2005-01-01 00:00:00+00:00	445319068560101	20.54	A
1	2005-01-02 00:00:00+00:00	445319068560101	20.62	A
2	2005-01-03 00:00:00+00:00	445319068560101	20.59	A
3	2005-01-04 00:00:00+00:00	445319068560101	20.65	A
4	2005-01-05 00:00:00+00:00	445319068560101	20.71	A
...	...	...	...	...
7295	2024-12-27 00:00:00+00:00	445319068560101	22.40	A
7296	2024-12-28 00:00:00+00:00	445319068560101	22.37	A
7297	2024-12-29 00:00:00+00:00	445319068560101	22.35	A
7298	2024-12-30 00:00:00+00:00	445319068560101	22.30	A
7299	2024-12-31 00:00:00+00:00	445319068560101	22.15	A

7300 rows × 4 columns

In [5]:
```
import xarray as xr
```

In [6]:
```
# URLs for NASA RCP8.5 dataset
url_pr = "https://ds.nccs.nasa.gov/thredds/dodsC/bypass/NEX-GDDP/bcsd/rcp85/r1i1p1/pr/GFDL-CM3.ncml"
url_tasmax = "https://ds.nccs.nasa.gov/thredds/dodsC/bypass/NEX-GDDP/bcsd/rcp85/r1i1p1/tasmax/GFDL-CM3.ncml"
url_tasmin = "https://ds.nccs.nasa.gov/thredds/dodsC/bypass/NEX-GDDP/bcsd/rcp85/r1i1p1/tasmin/GFDL-CM3.ncml"
```

In [7]:
```
# ------------------------------------------------------------
# Download and processing of NASA RCP8.5 climate data
# Location: 44.92°N, 68.97°W
# Time range: 2025-01-01 to 2099-12-31
# Source: NASA NEX-GDDP via OpenDAP
# Output: Local CSV file "rcp85_kenduskeag_2025_2099.csv"
# This block takes a long time to run — recommended to run only once
# ------------------------------------------------------------

"""
# Target location
target_lat = 44.92
```

```
target_lon = 360 - 68.97  # = 291.03

# Function to load and extract daily time series for a variable
def extract_variable(url, var_name, lat, lon, start_year=2025, end_year=2099):
    ds = xr.open_dataset(url)

    # Select nearest grid point
    ds_sel = ds.sel(lat=lat, lon=lon, method="nearest")

    # Filter time
    ds_sel = ds_sel.sel(time=slice(f"{start_year}-01-01", f"{end_year}-12-31"))

    # Convert to DataFrame
    df = ds_sel[var_name].to_dataframe().reset_index()
    df = df[['time', var_name]]
    return df

# Extract each variable
df_pr = extract_variable(url_pr, 'pr', target_lat, target_lon)
df_tasmax = extract_variable(url_tasmax, 'tasmax', target_lat, target_lon)
df_tasmin = extract_variable(url_tasmin, 'tasmin', target_lat, target_lon)

# Merge all variables into a single DataFrame
df = df_pr.merge(df_tasmax, on='time').merge(df_tasmin, on='time')

# Convert units
df['precip_mm_day'] = df['pr'] * 86400  # kg/m²/s → mm/day
df['temp_max_C'] = df['tasmax'] - 273.15
df['temp_min_C'] = df['tasmin'] - 273.15

# Final DataFrame
df_rcp = df[['time', 'precip_mm_day', 'temp_max_C', 'temp_min_C']]
df_rcp.columns = ['DATE', 'PRCP', 'TMAX', 'TMIN']

# Save future RCP data to a local CSV file
df_rcp.to_csv("rcp85_kenduskeag_2025_2099.csv", index=False)
"""

# Load pre-downloaded RCP data
df_rcp = pd.read_csv("rcp85_kenduskeag_2025_2099.csv")

df_rcp
```

Out[7]:

	DATE	PRCP	TMAX	TMIN
0	2025-01-01 12:00:00	1.894070	1.061157	-13.577332
1	2025-01-02 12:00:00	0.019432	-7.747620	-14.271210
2	2025-01-03 12:00:00	0.000000	-9.183258	-16.994354
3	2025-01-04 12:00:00	0.079074	1.262299	-17.583237
4	2025-01-05 12:00:00	1.745083	2.063904	-2.397827
...	...	...	...	...
27370	2099-12-27 12:00:00	15.121511	13.240601	-2.440888
27371	2099-12-28 12:00:00	3.131023	15.766174	-1.153870
27372	2099-12-29 12:00:00	0.386043	5.985992	-2.690765
27373	2099-12-30 12:00:00	0.284129	3.409027	-2.954010
27374	2099-12-31 12:00:00	0.289650	2.735748	-5.951752

27375 rows × 4 columns

2.2.2 Data Preparation

NOAA

- **Data Cleaning:**
  - **Check for Missing Data:** All 7305 entries are complete with no missing values in the required four columns.
- **Data Formatting:**
  - **Data Type Check:**
    - Converted `DATE` from object to datetime using `pd.to_datetime()`.
    - Convert inches to **mm/day** for `PRCP`, and °F to **°C** for `TMAX` and `TMIN`.
  - **Column Extraction:** Extracted four columns: `DATE`, `PRCP`, `TMAX`, and `TMIN`.
  - **Index Setting:** Set `DATE` as index to enable time-series analysis.

In [8]:

```
# Set target station and date range
target_station = 'BANGOR INTERNATIONAL AIRPORT, ME US'
start_date = '2005-01-01'
end_date = '2024-12-31'

# Filter by station name and date range
df_weather = df_weather[(df_weather['NAME'] == target_station) &
                        (df_weather['DATE'] >= start_date) &
                        (df_weather['DATE'] <= end_date)]

# Check non-null counts and data types
df_weather.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 7305 entries, 0 to 7304
Data columns (total 9 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   STATION  7305 non-null   object 
 1   NAME     7305 non-null   object 
 2   DATE     7305 non-null   object 
 3   PRCP     7305 non-null   float64
 4   SNOW     7305 non-null   float64
 5   SNWD     6568 non-null   float64
 6   TAVG     4505 non-null   float64
 7   TMAX     7305 non-null   float64
 8   TMIN     7305 non-null   float64
dtypes: float64(6), object(3)
memory usage: 570.7+ KB
```

In [9]:
```python
# Convert DATE column to datetime format
df_weather['DATE'] = pd.to_datetime(df_weather['DATE'])

# Convert units to match RCP (mm/day for PRCP, °C for temperature)
df_weather['PRCP'] = df_weather['PRCP'] * 25.4
df_weather['TMAX'] = (df_weather['TMAX'] - 32) * 5 / 9
df_weather['TMIN'] = (df_weather['TMIN'] - 32) * 5 / 9
```

In [10]:
```python
# Keep only required columns
df_weather = df_weather[['DATE', 'PRCP', 'TMAX', 'TMIN']]

df_weather.head()
```

Out[10]:

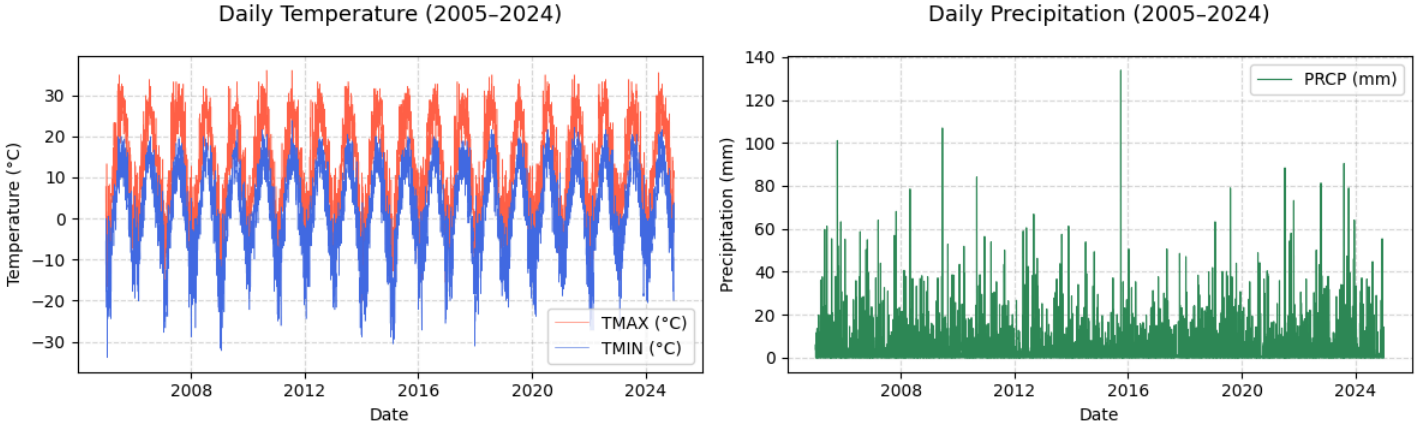
	DATE	PRCP	TMAX	TMIN
0	2005-01-01	3.810	7.777778	-5.555556
1	2005-01-02	6.096	0.555556	-11.111111
2	2005-01-03	1.270	5.555556	0.000000
3	2005-01-04	0.000	1.111111	-3.888889
4	2005-01-05	0.000	-1.111111	-12.222222

In [11]:
```python
# Set index and sort
df_weather = df_weather.sort_values("DATE").set_index("DATE")

# Last check
df_weather.head()
```

Out[11]:

DATE	PRCP	TMAX	TMIN
2005-01-01	3.810	7.777778	-5.555556
2005-01-02	6.096	0.555556	-11.111111
2005-01-03	1.270	5.555556	0.000000
2005-01-04	0.000	1.111111	-3.888889
2005-01-05	0.000	-1.111111	-12.222222

In [12]:
```python
# Plot
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Temperature plot
axes[0].plot(df_weather.index, df_weather['TMAX'], label='TMAX (°C)', color='tomato', linewidth=0.4)
axes[0].plot(df_weather.index, df_weather['TMIN'], label='TMIN (°C)', color='royalblue', linewidth=0.4)
axes[0].set_title("\nDaily Temperature (2005-2024)\n", fontsize=13)
axes[0].set_xlabel("Date")
axes[0].set_ylabel("Temperature (°C)")
axes[0].legend()
axes[0].grid(True, linestyle='--', alpha=0.5)

# Precipitation plot
axes[1].plot(df_weather.index, df_weather['PRCP'], label='PRCP (mm)', color='seagreen', linewidth=0.8)
axes[1].set_title("\nDaily Precipitation (2005-2024)\n", fontsize=13)
axes[1].set_xlabel("Date")
axes[1].set_ylabel("Precipitation (mm)")
axes[1].legend()
axes[1].grid(True, linestyle='--', alpha=0.5)

plt.tight_layout()
# plt.savefig('weather.png', dpi=300)
plt.show()
```



USGS

- Data Cleaning:
  - **Check for Missing Data:** Found `5 missing dates` and filled them using `linear interpolation` after reindexing.
- Data Formatting:
  - **Date Format Adjustment:** Removed the timezone information from `DATE` column using `.dt.tz_localize(None)`.
  - **Column Extraction and Renaming:** Selected `datetime` and `72019_Mean`, then renamed them to `DATE` and `GWL` for consistency.
  - **Index Setting:** Set `DATE` as index to enable time-series analysis.

In [13]:
```
# Check non-null counts and data types
df_gwl.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7300 entries, 0 to 7299
Data columns (total 4 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   datetime       7300 non-null   datetime64[ns, UTC]
 1   site_no        7300 non-null   object
 2   72019_Mean     7300 non-null   float64
 3   72019_Mean_cd  7300 non-null   object
dtypes: datetime64[ns, UTC](1), float64(1), object(2)
memory usage: 228.3+ KB
```

In [14]:
```
# Calculate expected number of daily records
start = pd.to_datetime('2005-01-01')
end = pd.to_datetime('2024-12-31')
num_days = (end - start).days + 1
print(f"Expected number of days: {num_days}\n")

# Identify missing dates in the df_gwl dataset
# Create a complete date range
full_range = pd.date_range(start=start, end=end, freq='D', tz='UTC')

# Convert the DATE column to a set
actual_dates = set(df_gwl["datetime"])

# Identify missing dates
missing_dates = sorted(set(full_range) - actual_dates)

# Print missing dates and total count
print("Missing dates:")
for date in missing_dates:
    print(f" - {date.date()}")
print(f"\nTotal missing dates: {len(missing_dates)}")
```

```
Expected number of days: 7305

Missing dates:
 - 2008-06-08
 - 2018-02-28
 - 2018-03-01
 - 2018-03-05
 - 2018-03-06

Total missing dates: 5
```

In [15]:
```
# Remove timezone information
df_gwl["datetime"] = df_gwl["datetime"].dt.tz_localize(None)
```

In [16]:
```
# Extract and rename columns
df_gwl = df_gwl[["datetime", "72019_Mean"]]
df_gwl.rename(columns={"datetime": "DATE", "72019_Mean": "GWL"}, inplace=True)

df_gwl.head()
```

Out[16]:

	DATE	GWL
0	2005-01-01	20.54
1	2005-01-02	20.62
2	2005-01-03	20.59
3	2005-01-04	20.65
4	2005-01-05	20.71

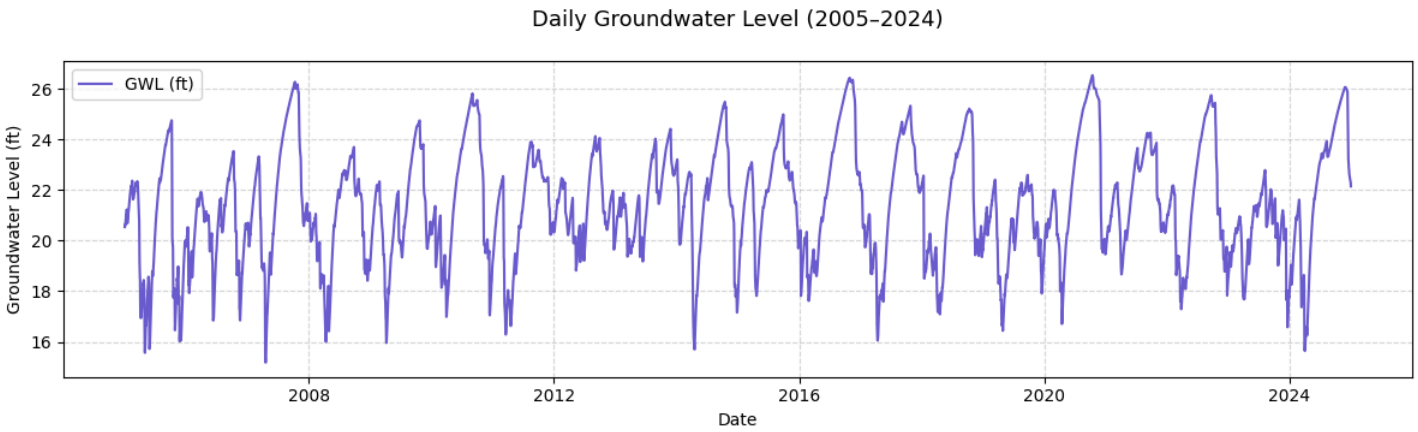
In [17]:
```
# Set index and sort
df_gwl = df_gwl.sort_values("DATE").set_index("DATE")

# Reindex and interpolate
full_range = pd.date_range(start=start, end=end, freq='D')
df_gwl = df_gwl.reindex(full_range)
df_gwl.index.name = 'DATE'
df_gwl['GWL'] = df_gwl['GWL'].interpolate(method='linear')

# Check result
print(f"\n(After reindexing and interpolation)")
print(f"\n - Missing values: {df_gwl['GWL'].isna().sum()}")
print(f" - Total days: {len(df_gwl)}\n")

# Last check
df_gwl.head()
```

```
(After reindexing and interpolation)

 - Missing values: 0
 - Total days: 7305
```

Out[17]:

DATE	GWL
2005-01-01	20.54
2005-01-02	20.62
2005-01-03	20.59
2005-01-04	20.65
2005-01-05	20.71

In [18]:
```
# Plot
plt.figure(figsize=(12, 4))
plt.plot(df_gwl.index, df_gwl['GWL'], label='GWL (ft)', color='slateblue')

plt.title("\nDaily Groundwater Level (2005–2024)\n", fontsize=13)
plt.xlabel("Date")
plt.ylabel("Groundwater Level (ft)")
plt.grid(True, linestyle='--', alpha=0.5)
plt.legend()
plt.tight_layout()
plt.show()
```



NASA

- Data Cleaning:
  - **Check for Missing Data:** There are no missing values across all columns.
- Data Formatting:
  - **Data Type Check:** Converted `DATE` from object to `datetime.date` format.
  - **Index Setting:** Set `DATE` as index to enable time-series analysis.

In [19]:
```
# Check non-null counts and data types
df_rcp.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27375 entries, 0 to 27374
Data columns (total 4 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   DATE    27375 non-null  object
 1   PRCP    27375 non-null  float64
 2   TMAX    27375 non-null  float64
 3   TMIN    27375 non-null  float64
dtypes: float64(3), object(1)
memory usage: 855.6+ KB
```

In [20]:
```
# Convert DATE column to datetime.date and set as index
df_rcp['DATE'] = pd.to_datetime(df_rcp['DATE'].astype(str)).dt.date
df_rcp.set_index('DATE', inplace=True)

# Last check
df_rcp.head()
```

Out[20]:

DATE	PRCP	TMAX	TMIN
2025-01-01	1.894070	1.061157	-13.577332
2025-01-02	0.019432	-7.747620	-14.271210
2025-01-03	0.000000	-9.183258	-16.994354
2025-01-04	0.079074	1.262299	-17.583237
2025-01-05	1.745083	2.063904	-2.397827

- **Data Integration:** Merged the `df_weather` and `df_gwl` datasets into a single table using `DATE` as the index.

In [21]:
```
# Merge on DATE index
df_merged = df_weather.merge(df_gwl, left_index=True, right_index=True)

df_merged.head()
```

Out[21]:

DATE	PRCP	TMAX	TMIN	GWL
2005-01-01	3.810	7.777778	-5.555556	20.54
2005-01-02	6.096	0.555556	-11.111111	20.62
2005-01-03	1.270	5.555556	0.000000	20.59
2005-01-04	0.000	1.111111	-3.888889	20.65
2005-01-05	0.000	-1.111111	-12.222222	20.71

- **Feature Engineering:**
  - **1-day Lag Features:** for air temperature and precipitation to capture short-term persistence.
  - **7-day Rolling Averages:** for air temperature and precipitation to reflect short-term climate trends.
  - **14-day Rolling Averages:** for air temperature and precipitation to incorporate longer-term seasonal patterns.

    Note: The first 13 days (`2005-01-01` to `2005-01-13` and `2025-01-01` to `2025-01-13`) were dropped due to missing values resulting from lag and rolling window calculations.

In [22]:
```
# Feature Engineering for training data

# 1-day lag features
df_merged['TMAX_t-1'] = df_merged['TMAX'].shift(1)
df_merged['TMIN_t-1'] = df_merged['TMIN'].shift(1)
df_merged['PRCP_t-1'] = df_merged['PRCP'].shift(1)

# 7-day rolling mean features
df_merged['TMAX_7day_mean'] = df_merged['TMAX'].rolling(window=7).mean()
df_merged['TMIN_7day_mean'] = df_merged['TMIN'].rolling(window=7).mean()
df_merged['PRCP_7day_mean'] = df_merged['PRCP'].rolling(window=7).mean()

# 14-day rolling mean features
df_merged['TMAX_14day_mean'] = df_merged['TMAX'].rolling(window=14).mean()
df_merged['TMIN_14day_mean'] = df_merged['TMIN'].rolling(window=14).mean()
df_merged['PRCP_14day_mean'] = df_merged['PRCP'].rolling(window=14).mean()

# Drop rows with missing values due to rolling & lag operations
df_model = df_merged.dropna()

df_model.head()
```

Out[22]:

DATE	PRCP	TMAX	TMIN	GWL	TMAX_t-1	TMIN_t-1	PRCP_t-1	TMAX_7day_mean	TMIN_7day_mean	PRCP_7day_mean	TMAX_14day_mean	TMIN_14day_mean	PR
2005-01-14	11.684	13.333333	-3.888889	21.18	6.111111	-7.222222	0.000	1.269841	-13.015873	3.229429	1.071429	-10.833333	
2005-01-15	0.000	-3.888889	-9.444444	20.93	13.333333	-3.888889	11.684	0.873016	-12.380952	2.612571	0.238095	-11.111111	
2005-01-16	1.524	-5.000000	-11.666667	20.75	-3.888889	-9.444444	0.000	0.634921	-11.507937	2.830286	-0.158730	-11.150794	
2005-01-17	3.048	-7.777778	-15.555556	20.69	-5.000000	-11.666667	1.524	-0.714286	-12.222222	3.011714	-1.111111	-12.261905	
2005-01-18	0.000	-14.444444	-22.222222	20.79	-7.777778	-15.555556	3.048	-2.539683	-13.095238	3.011714	-2.222222	-13.571429	

In [23]:
```
# Feature Engineering for RCP climate data

# Lag features
df_rcp['TMAX_t-1'] = df_rcp['TMAX'].shift(1)
df_rcp['TMIN_t-1'] = df_rcp['TMIN'].shift(1)
df_rcp['PRCP_t-1'] = df_rcp['PRCP'].shift(1)

# Rolling features
df_rcp['TMAX_7day_mean'] = df_rcp['TMAX'].rolling(window=7).mean()
df_rcp['TMIN_7day_mean'] = df_rcp['TMIN'].rolling(window=7).mean()
df_rcp['PRCP_7day_mean'] = df_rcp['PRCP'].rolling(window=7).mean()

df_rcp['TMAX_14day_mean'] = df_rcp['TMAX'].rolling(window=14).mean()
df_rcp['TMIN_14day_mean'] = df_rcp['TMIN'].rolling(window=14).mean()
df_rcp['PRCP_14day_mean'] = df_rcp['PRCP'].rolling(window=14).mean()

# Drop rows with missing values due to rolling & lag operations
df_rcp = df_rcp.dropna()

df_rcp.head()
```

Out[23]:

DATE	PRCP	TMAX	TMIN	TMAX_t-1	TMIN_t-1	PRCP_t-1	TMAX_7day_mean	TMIN_7day_mean	PRCP_7day_mean	TMAX_14day_mean	TMIN_14day_mean	PRCP_1
2025-01-14	0.024324	-2.025055	-10.571167	1.881531	-6.317078	0.334157	0.274676	-8.395229	0.748054	-0.492852	-9.912538	
2025-01-15	0.020704	-0.381653	-15.352997	-2.025055	-10.571167	0.024324	-0.343000	-9.848842	0.426606	-0.595910	-10.039371	
2025-01-16	0.389848	2.543762	-13.638550	-0.381653	-15.352997	0.020704	-0.053109	-10.708243	0.482299	0.139188	-9.994181	
2025-01-17	0.023067	-7.071472	-13.106445	2.543762	-13.638550	0.389848	-0.567221	-11.131335	0.485594	0.290030	-9.716473	
2025-01-18	0.000000	-2.236054	-16.466522	-7.071472	-13.106445	0.023067	-0.776476	-11.751940	0.408188	0.040148	-9.636708	

- **Data Transformation:** Selected features were scaled to [0, 1] using `MinMaxScaler` to ensure all input features are on a comparable scale.

In [24]:
```
from sklearn.preprocessing import MinMaxScaler
```

In [25]:
```
# Separate features and target
X = df_model.drop(columns=['GWL'])
y = df_model['GWL']

# Fit scaler on X
scaler = MinMaxScaler()
X_scaled = pd.DataFrame(scaler.fit_transform(X), columns=X.columns, index=X.index)
```

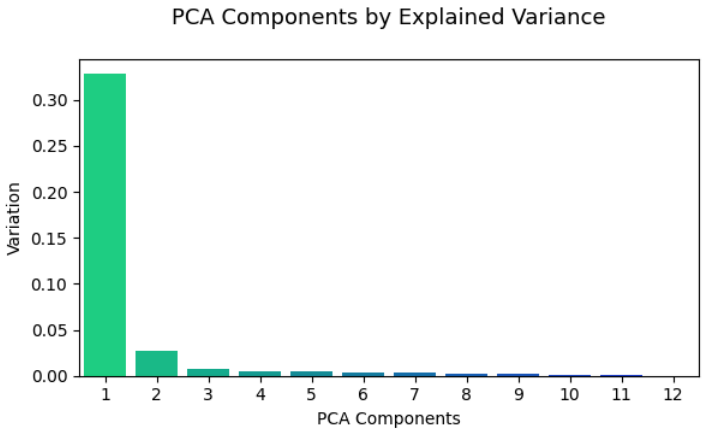
In [26]:
```
# Scale future RCP features using the previously fitted scaler
X_future_scaled = pd.DataFrame(scaler.transform(df_rcp), columns=df_rcp.columns, index=df_rcp.index)
```

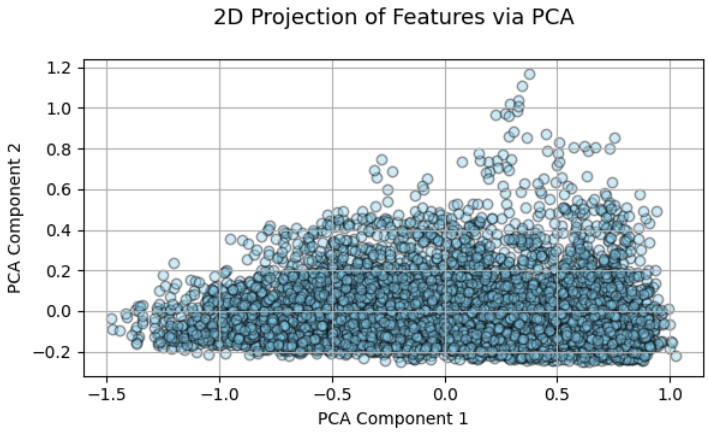
2.2.3 Model Building & Testing

- Unsupervised Learning:
  - Dimensional Reduction:
    - **PCA Component Analysis:** The first component explains the majority of the variation, justifying dimensionality reduction.
    - **2D PCA Projection:** The data points are densely distributed without clear separation.
  - K-Means Clustering:
    - **Elbow Method:** `k=2` is optimal, indicating that two clusters sufficiently capture underlying structure.
    - **Clustering on PCA data:** K-Means divides the data into two clusters, but the data distribution is continuous without natural boundaries.
    - **GWL Comparison:** Both clustering methods show different medians, but overlapping ranges and outliers suggest the separation isn't perfectly distinct.

In [27]:
```
from sklearn.decomposition import PCA
```

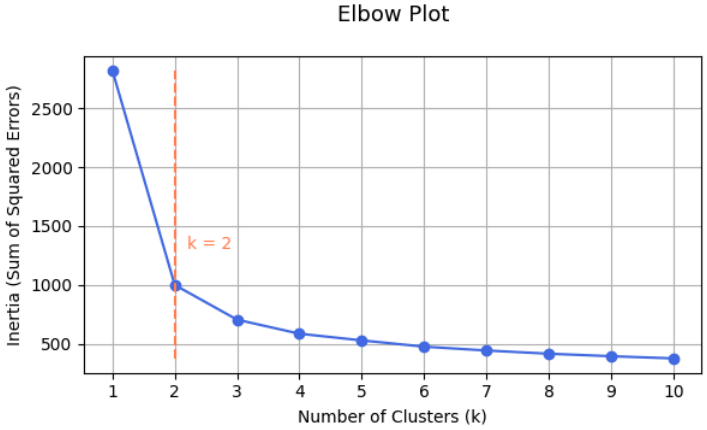
In [28]:
```
# Fit full PCA
pca_full = PCA()
pca_full.fit(X_scaled)
pca_data = pca_full.fit_transform(X_scaled)
pca_features = list(range(1, len(pca_full.explained_variance_) + 1))

# 2D PCA
pca_2d = PCA(n_components=2)
X_pca = pca_2d.fit_transform(X_scaled)

# Plot
fig, axes = plt.subplots(1, 2, figsize=(12, 4), facecolor='white')

# Left: PCA components by variation
axes[0].set_facecolor('white')
sns.barplot(x=pca_features, y=pca_full.explained_variance_, palette="winter_r", ax=axes[0])
axes[0].set_ylabel('Variation', fontsize=10)
axes[0].set_xlabel('PCA Components', fontsize=10)
axes[0].set_title('\nPCA Components by Explained Variance\n', fontsize=13)

# Right: 2D PCA scatter plot
axes[1].scatter(X_pca[:, 0], X_pca[:, 1], color='skyblue', edgecolor='k', alpha=0.4)
axes[1].set_title('\n2D Projection of Features via PCA\n', fontsize=13)
axes[1].set_xlabel('PCA Component 1')
axes[1].set_ylabel('PCA Component 2')
axes[1].grid(True)

plt.tight_layout()
# plt.savefig('PCA.png', dpi=300)
plt.show()
```

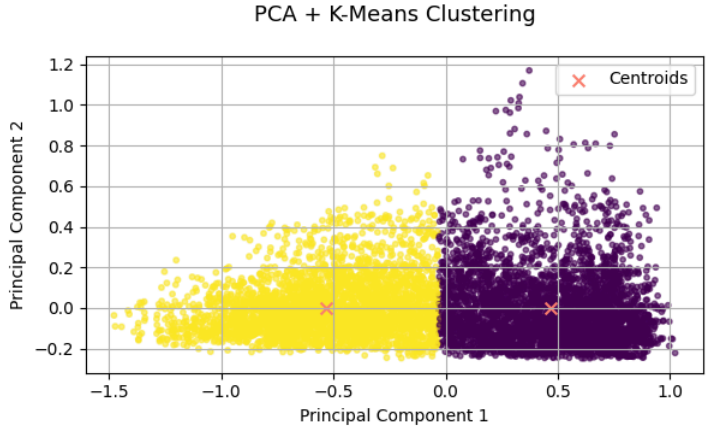




In [29]:
```
from sklearn.cluster import KMeans
```

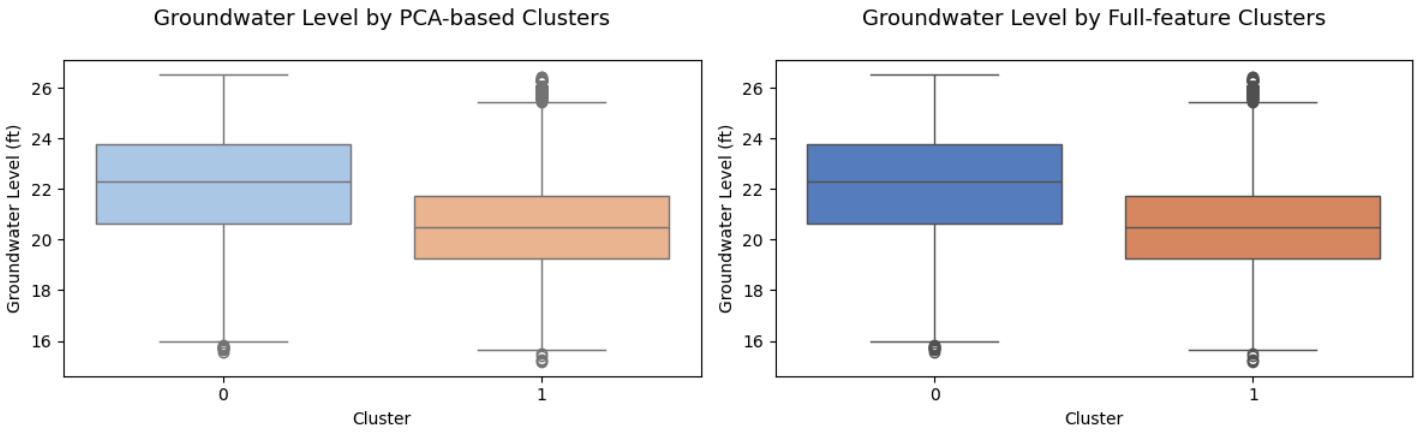
In [30]:
```
# Elbow plot preparation
k_range = range(1, 11)
inertia = []
for k in k_range:
    kmeans = KMeans(n_clusters=k, random_state=42)
    kmeans.fit(X_scaled)
    inertia.append(kmeans.inertia_)

inertia_diffs = np.diff(inertia)
elbow_point = np.argmin(inertia_diffs[1:]) + 2

# KMeans clustering on PCA-reduced data (X_pca)
kmeans_pca = KMeans(n_clusters=2, random_state=42)
clusters_pca = kmeans_pca.fit_predict(X_pca)
centers_pca = kmeans_pca.cluster_centers_

# Plot
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Left: Elbow plot
axes[0].plot(k_range, inertia, marker='o', linestyle='-', color='royalblue')
axes[0].vlines(elbow_point, ymin=min(inertia), ymax=max(inertia), linestyles='dashed', colors='coral')
axes[0].text(elbow_point + 0.2, inertia[elbow_point] + 600, f'k = {elbow_point}', color='coral')
axes[0].set_title('\nElbow Plot\n', fontsize=13)
axes[0].set_xlabel('Number of Clusters (k)')
axes[0].set_ylabel('Inertia (Sum of Squared Errors)')
axes[0].set_xticks(list(k_range))
axes[0].grid(True)

# Right: PCA + KMeans clustering plot
axes[1].scatter(X_pca[:, 0], X_pca[:, 1], c=clusters_pca, cmap='viridis', s=10, alpha=0.6)
axes[1].scatter(centers_pca[:, 0], centers_pca[:, 1], c='salmon', marker='x', s=50, label='Centroids')
axes[1].set_title('\nPCA + K-Means Clustering\n', fontsize=13)
axes[1].set_xlabel('Principal Component 1')
axes[1].set_ylabel('Principal Component 2')
axes[1].legend()
axes[1].grid(True)

plt.tight_layout()
# plt.savefig('clustering.png', dpi=300)
plt.show()
```





In [31]:
```
# KMeans clustering on full-feature data (X_scaled)
kmeans_full = KMeans(n_clusters=2, random_state=42)
clusters_full = kmeans_full.fit_predict(X_scaled)

# Plot
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Left: PCA clusters
df_pca_clustered = df_model.copy()
df_pca_clustered['Cluster'] = clusters_pca
sns.boxplot(x='Cluster', y='GWL', data=df_pca_clustered, palette='pastel', ax=axes[0])
axes[0].set_title('\nGroundwater Level by PCA-based Clusters\n', fontsize=13)
axes[0].set_xlabel('Cluster')
axes[0].set_ylabel('Groundwater Level (ft)')

# Right: Full-feature clusters
df_clustered = df_model.copy()
```

```
df_clustered['Cluster'] = clusters_full
sns.boxplot(x='Cluster', y='GWL', data=df_clustered, palette='muted', ax=axes[1])
axes[1].set_title('\nGroundwater Level by Full-feature Clusters\n', fontsize=13)
axes[1].set_xlabel('Cluster')
axes[1].set_ylabel('Groundwater Level (ft)')

plt.tight_layout()
# plt.savefig('clustering_boxplot.png', dpi=300)
plt.show()

# Average GWL values table
avg_pca = df_pca_clustered.groupby('Cluster')['GWL'].mean().reset_index().round(4)
avg_pca.columns = ['Cluster', 'Avg GWL (PCA)']
avg_full = df_clustered.groupby('Cluster')['GWL'].mean().reset_index().round(4)
avg_full.columns = ['Cluster', 'Avg GWL (Full)']

avg_combined = pd.merge(avg_pca, avg_full, on='Cluster')
avg_combined['Difference'] = (avg_combined['Avg GWL (PCA)'] - avg_combined['Avg GWL (Full)']).round(4)

threshold = 21.0
avg_combined['Cluster'] = avg_combined[['Avg GWL (PCA)', 'Avg GWL (Full)']].mean(axis=1).apply(
    lambda x: 'Drought' if x > threshold else 'No Drought'
)

avg_combined
```



Out[31]:

	Cluster	Avg GWL (PCA)	Avg GWL (Full)	Difference
0	Drought	22.1584	22.1571	0.0013
1	No Drought	20.5169	20.5179	-0.0010

- **Supervised Learning:** To evaluate different regression models, the dataset was split by date. **2005–2023** data was used for **training and validation** (randomly split 80/20), while **2024** data was held out as an unseen **test set** to evaluate the model's generalization to future observations.

  - **Linear Regression (LR):**
    - `LinearRegression()` : A simple yet powerful supervised learning method for predicting quantitative outcomes. Despite its simplicity, it remains widely used and serves as the foundation for many modern statistical learning techniques. [8].
    - **Finding:** Despite its simplicity, LR served as a reliable baseline (Test RMSE: **2.392**, R^2: **0.28**, MAPE: **8.79%**), but was clearly outperformed by more flexible models.
  - **Neural Network (NN):**
    - `MLPRegressor(hidden_layer_sizes=(64, 32), solver='lbfgs', max_iter=2000)` : A type of feedforward neural network with hidden layers. It is trained using backpropagation to reduce prediction error through gradient descent. [9].
    - **Finding:** NN achieved the best generalization performance (Test RMSE: **2.209**, R^2: **0.38**, MAPE: **7.42%**), suggesting it effectively captured underlying patterns in the data.
  - **Random Forest (RF):**
    - `RandomForestRegressor(n_estimators=300, max_depth=15, min_samples_leaf=1)` : An ensemble method based on decision trees. It uses bagging and randomization to build multiple trees and predicts by majority vote to improve accuracy and reduce overfitting [10].
    - **Finding:** RF delivered the best validation performance (Val RMSE: **1.449**, R^2: **0.58**, MAPE: **5.15%**), though its generalization to the test set was slightly weaker than that of NN.
  - **Support Vector Machine (SVM):**
    - `SVR(C=100, epsilon=0.01)` : Support vector regression estimates a function within a specified ε-insensitive margin, balancing prediction accuracy and model simplicity. It uses kernel methods to handle nonlinearity and produces sparse solutions based only on support vectors [11].
    - **Finding:** SVM achieved balanced performance across both validation and testing datasets (Test RMSE: **2.341**, R^2: **0.31**, MAPE: **7.28%**), with the lowest test MAPE, indicating strong and stable generalization.

Model	Val RMSE	Val R^2	Val MAPE	Test RMSE	Test R^2	Test MAPE
Linear Regression (LR)	1.937	0.24	7.34%	2.392	0.28	8.79%
Neural Network (NN)	1.711	0.41	6.14%	2.209	0.38	7.42%
Random Forest (RF)	1.449	0.58	5.15%	2.369	0.29	7.92%
Support Vector Machine (SVM)	1.751	0.38	6.09%	2.341	0.31	7.28%

Summary:

Random Forest achieved the best performance on the validation set, suggesting strong learning from historical data. Although **Support Vector Machine** achieved the lowest test MAPE, the **Neural Network** model demonstrated the best overall performance in both RMSE and R^2. Given the long-term forecasting nature of the RCP scenario, I selected the Neural Network as the final projection model due to its stronger generalization and better fit to observed variability.

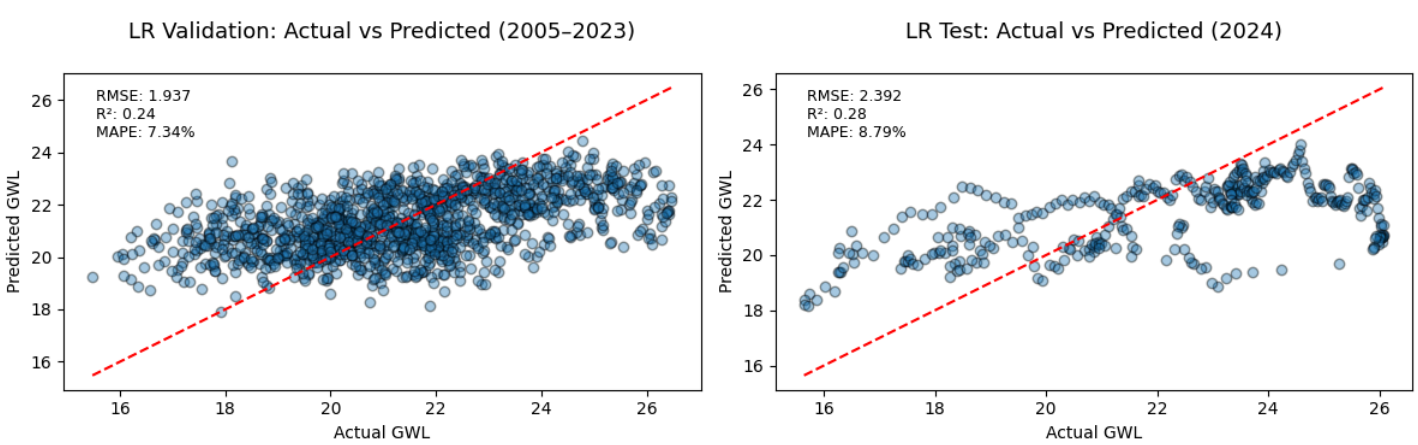
In [32]:
```
from sklearn.model_selection import train_test_split
```

In [33]:
```
# Select data from 2005 to 2023 for training and validation
X_trainval = X_scaled.loc[X_scaled.index.year < 2024]
y_trainval = y.loc[y.index.year < 2024]

# Select 2024 data for testing
X_test = X_scaled.loc[X_scaled.index.year == 2024]
y_test = y.loc[y.index.year == 2024]

# Split train+val into 80/20
X_train, X_val, y_train, y_val = train_test_split(X_trainval, y_trainval, test_size=0.2, random_state=42)

print(f"Training data: {len(X_train)}")
print(f"Validation data: {len(X_val)}")
print(f"Test data (2024): {len(X_test)}")
```

```
Training data: 5540
Validation data: 1386
Test data (2024): 366
```

In [34]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_percentage_error
```

In [35]:
```
# Train Linear Regression model
lr_model = LinearRegression()
lr_model.fit(X_train, y_train)

# Predict on validation and test sets
y_val_pred = lr_model.predict(X_val)
y_test_pred = lr_model.predict(X_test)

# Evaluate performance
val_rmse = mean_squared_error(y_val, y_val_pred, squared=False)
val_r2 = r2_score(y_val, y_val_pred)
val_mape = mean_absolute_percentage_error(y_val, y_val_pred)

test_rmse = mean_squared_error(y_test, y_test_pred, squared=False)
test_r2 = r2_score(y_test, y_test_pred)
test_mape = mean_absolute_percentage_error(y_test, y_test_pred)

# Plot actual vs predicted
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Validation plot
axes[0].scatter(y_val, y_val_pred, alpha=0.4, edgecolor='k')
axes[0].plot([y_val.min(), y_val.max()], [y_val.min(), y_val.max()], 'r--')
axes[0].set_title("\nLR Validation: Actual vs Predicted (2005–2023)\n", fontsize=13)
axes[0].text(0.05, 0.95, f"RMSE: {val_rmse:.3f}\nR²: {val_r2:.2f}\nMAPE: {val_mape*100:.2f}%",
             transform=axes[0].transAxes, fontsize=9, verticalalignment='top')
axes[0].set_xlabel("Actual GWL")
axes[0].set_ylabel("Predicted GWL")

# Test plot
axes[1].scatter(y_test, y_test_pred, alpha=0.4, edgecolor='k')
axes[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
axes[1].set_title("\nLR Test: Actual vs Predicted (2024)\n", fontsize=13)
axes[1].text(0.05, 0.95, f"RMSE: {test_rmse:.3f}\nR²: {test_r2:.2f}\nMAPE: {test_mape*100:.2f}%",
             transform=axes[1].transAxes, fontsize=9, verticalalignment='top')
axes[1].set_xlabel("Actual GWL")
axes[1].set_ylabel("Predicted GWL")

plt.tight_layout()
# plt.savefig('lr_scatterplot.png', dpi=300)
plt.show()

# Print final metrics
print(f"Validation RMSE: {val_rmse:.3f}, R²: {val_r2:.2f}, MAPE: {val_mape*100:.2f}%")
print(f"Test RMSE: {test_rmse:.3f}, R²: {test_r2:.2f}, MAPE: {test_mape*100:.2f}%")
```

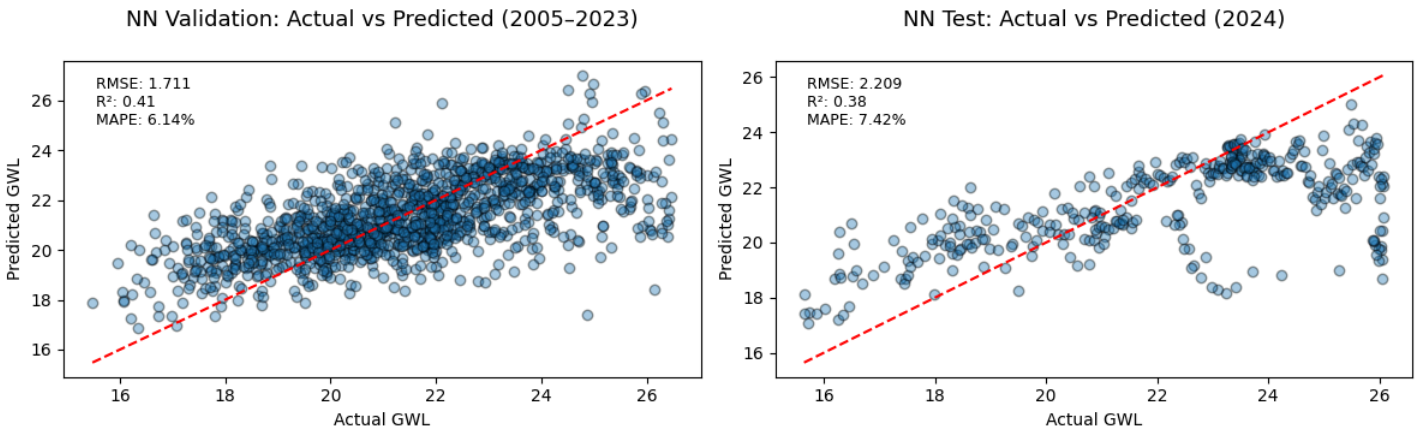



```
Validation RMSE: 1.937, R²: 0.24, MAPE: 7.34%
Test RMSE: 2.392, R²: 0.28, MAPE: 8.79%
```

In [36]:
```
from sklearn.neural_network import MLPRegressor
```

In [37]:
```
# Train Neural Network model (ReLU activation by default)
nn_model = MLPRegressor(hidden_layer_sizes=(64, 32), solver='lbfgs', max_iter=2000, random_state=42)
nn_model.fit(X_train, y_train)
```

```
# Predict on validation and test sets
y_val_pred = nn_model.predict(X_val)
y_test_pred = nn_model.predict(X_test)

# Evaluate performance
val_rmse = mean_squared_error(y_val, y_val_pred, squared=False)
val_r2 = r2_score(y_val, y_val_pred)
val_mape = mean_absolute_percentage_error(y_val, y_val_pred)

test_rmse = mean_squared_error(y_test, y_test_pred, squared=False)
test_r2 = r2_score(y_test, y_test_pred)
test_mape = mean_absolute_percentage_error(y_test, y_test_pred)

# Plot actual vs predicted
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Validation plot
axes[0].scatter(y_val, y_val_pred, alpha=0.4, edgecolor='k')
axes[0].plot([y_val.min(), y_val.max()], [y_val.min(), y_val.max()], 'r--')
axes[0].set_title("\nNN Validation: Actual vs Predicted (2005–2023)\n", fontsize=13)
axes[0].text(0.05, 0.95, f"RMSE: {val_rmse:.3f}\nR²: {val_r2:.2f}\nMAPE: {val_mape*100:.2f}%",
             transform=axes[0].transAxes, fontsize=9, verticalalignment='top')
axes[0].set_xlabel("Actual GWL")
axes[0].set_ylabel("Predicted GWL")

# Test plot
axes[1].scatter(y_test, y_test_pred, alpha=0.4, edgecolor='k')
axes[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
axes[1].set_title("\nNN Test: Actual vs Predicted (2024)\n", fontsize=13)
axes[1].text(0.05, 0.95, f"RMSE: {test_rmse:.3f}\nR²: {test_r2:.2f}\nMAPE: {test_mape*100:.2f}%",
             transform=axes[1].transAxes, fontsize=9, verticalalignment='top')
axes[1].set_xlabel("Actual GWL")
axes[1].set_ylabel("Predicted GWL")

plt.tight_layout()
# plt.savefig('nn_scatterplot.png', dpi=300)
plt.show()

# Print final metrics
print(f"Validation RMSE: {val_rmse:.3f}, R²: {val_r2:.2f}, MAPE: {val_mape*100:.2f}%")
print(f"Test RMSE: {test_rmse:.3f}, R²: {test_r2:.2f}, MAPE: {test_mape*100:.2f}%")
```

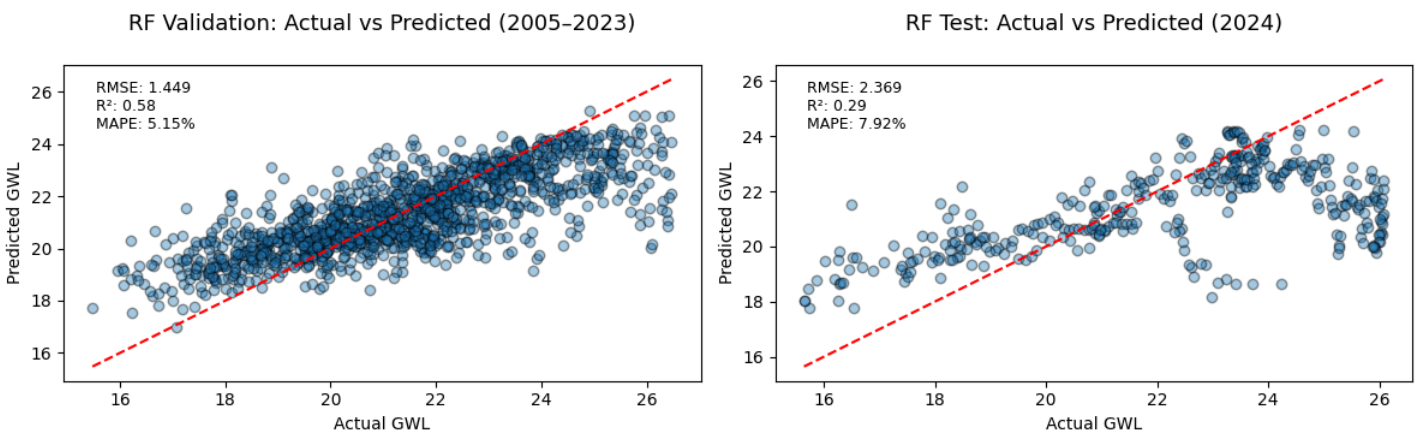



```
Validation RMSE: 1.711, R²: 0.41, MAPE: 6.14%
Test RMSE: 2.209, R²: 0.38, MAPE: 7.42%
```

In [38]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [39]:
```
# Train Random Forest model
rf_model = RandomForestRegressor(n_estimators=300, max_depth=15, min_samples_leaf=1, random_state=42, n_jobs=-1)
rf_model.fit(X_train, y_train)

# Predict on validation and test sets
y_val_pred = rf_model.predict(X_val)
y_test_pred = rf_model.predict(X_test)

# Evaluate performance
val_rmse = mean_squared_error(y_val, y_val_pred, squared=False)
val_r2 = r2_score(y_val, y_val_pred)
val_mape = mean_absolute_percentage_error(y_val, y_val_pred)

test_rmse = mean_squared_error(y_test, y_test_pred, squared=False)
test_r2 = r2_score(y_test, y_test_pred)
test_mape = mean_absolute_percentage_error(y_test, y_test_pred)

# Plot actual vs predicted
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Validation plot
axes[0].scatter(y_val, y_val_pred, alpha=0.4, edgecolor='k')
axes[0].plot([y_val.min(), y_val.max()], [y_val.min(), y_val.max()], 'r--')
axes[0].set_title("\nRF Validation: Actual vs Predicted (2005–2023)\n", fontsize=13)
axes[0].text(0.05, 0.95, f"RMSE: {val_rmse:.3f}\nR²: {val_r2:.2f}\nMAPE: {val_mape*100:.2f}%",
             transform=axes[0].transAxes, fontsize=9, verticalalignment='top')
axes[0].set_xlabel("Actual GWL")
axes[0].set_ylabel("Predicted GWL")

# Test plot
axes[1].scatter(y_test, y_test_pred, alpha=0.4, edgecolor='k')
axes[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
axes[1].set_title("\nRF Test: Actual vs Predicted (2024)\n", fontsize=13)
axes[1].text(0.05, 0.95, f"RMSE: {test_rmse:.3f}\nR²: {test_r2:.2f}\nMAPE: {test_mape*100:.2f}%",
             transform=axes[1].transAxes, fontsize=9, verticalalignment='top')
axes[1].set_xlabel("Actual GWL")
axes[1].set_ylabel("Predicted GWL")

plt.tight_layout()
```

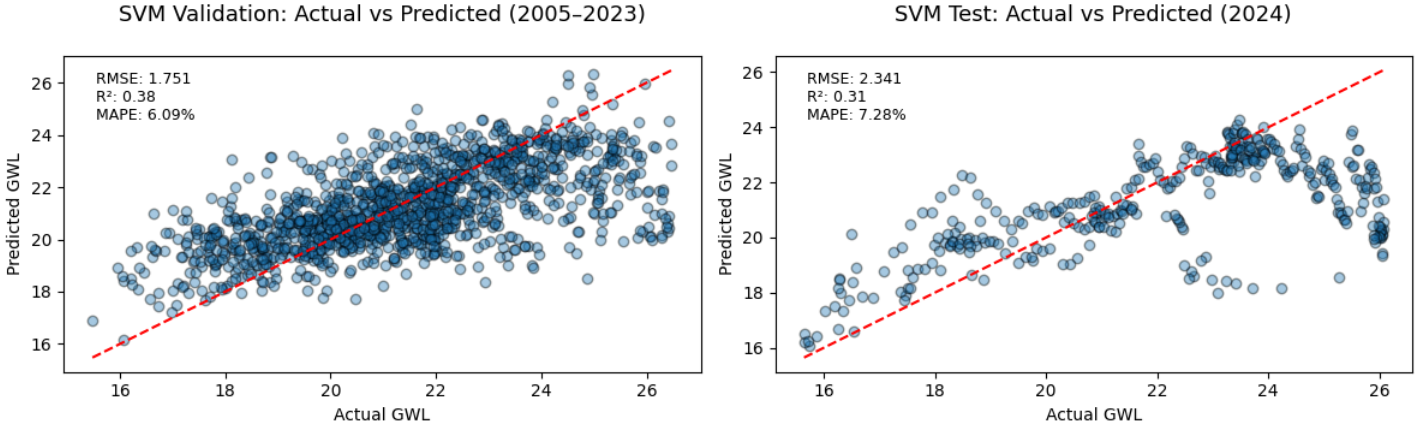
```
# plt.savefig('rf_scatterplot.png', dpi=300)
plt.show()

# Print result
print(f"Validation RMSE: {val_rmse:.3f}, R²: {val_r2:.2f}, MAPE: {val_mape*100:.2f}%")
print(f"Test RMSE: {test_rmse:.3f}, R²: {test_r2:.2f}, MAPE: {test_mape*100:.2f}%")
```




```
Validation RMSE: 1.449, R²: 0.58, MAPE: 5.15%
Test RMSE: 2.369, R²: 0.29, MAPE: 7.92%
```

In [40]:
```
from sklearn.svm import SVR
```

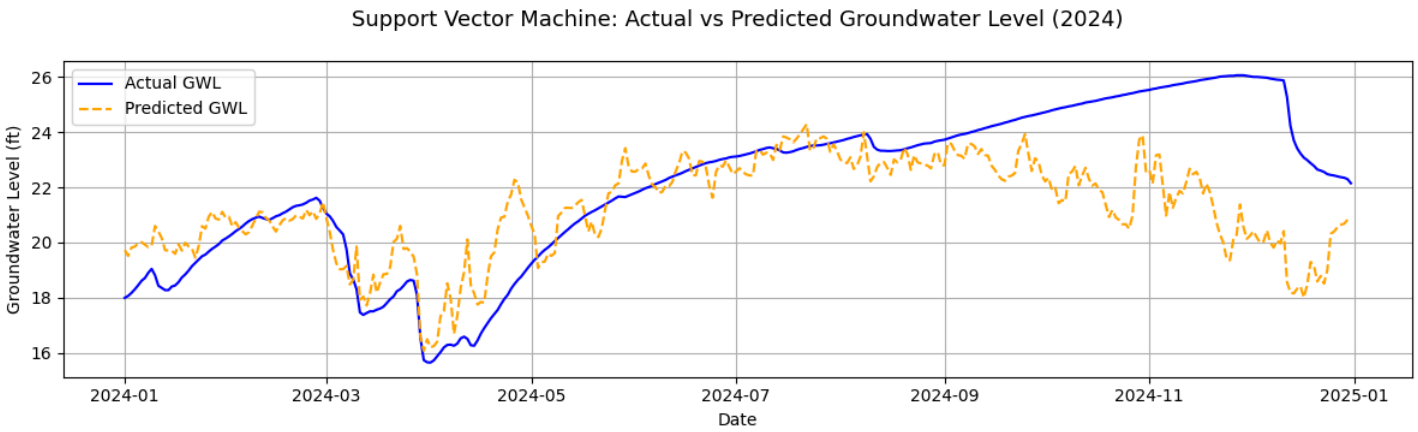
In [41]:
```
# Train SVR model (RBF kernel by default)
svr_model = SVR(C=100, epsilon=0.01)
svr_model.fit(X_train, y_train)

# Predict on validation and test sets
y_val_pred = svr_model.predict(X_val)
y_test_pred = svr_model.predict(X_test)

# Evaluate performance
val_rmse = mean_squared_error(y_val, y_val_pred, squared=False)
val_r2 = r2_score(y_val, y_val_pred)
val_mape = mean_absolute_percentage_error(y_val, y_val_pred)

test_rmse = mean_squared_error(y_test, y_test_pred, squared=False)
test_r2 = r2_score(y_test, y_test_pred)
test_mape = mean_absolute_percentage_error(y_test, y_test_pred)

# Plot actual vs predicted
fig, axes = plt.subplots(1, 2, figsize=(12, 4))

# Validation plot
axes[0].scatter(y_val, y_val_pred, alpha=0.4, edgecolor='k')
axes[0].plot([y_val.min(), y_val.max()], [y_val.min(), y_val.max()], 'r--')
axes[0].set_title("\nSVM Validation: Actual vs Predicted (2005–2023)\n", fontsize=13)
axes[0].text(0.05, 0.95, f"RMSE: {val_rmse:.3f}\nR²: {val_r2:.2f}\nMAPE: {val_mape*100:.2f}%",
             transform=axes[0].transAxes, fontsize=9, verticalalignment='top')
axes[0].set_xlabel("Actual GWL")
axes[0].set_ylabel("Predicted GWL")

# Test plot
axes[1].scatter(y_test, y_test_pred, alpha=0.4, edgecolor='k')
axes[1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
axes[1].set_title("\nSVM Test: Actual vs Predicted (2024)\n", fontsize=13)
axes[1].text(0.05, 0.95, f"RMSE: {test_rmse:.3f}\nR²: {test_r2:.2f}\nMAPE: {test_mape*100:.2f}%",
             transform=axes[1].transAxes, fontsize=9, verticalalignment='top')
axes[1].set_xlabel("Actual GWL")
axes[1].set_ylabel("Predicted GWL")

plt.tight_layout()
# plt.savefig('svm_scatterplot.png', dpi=300)
plt.show()

# Print final metrics
print(f"Validation RMSE: {val_rmse:.3f}, R²: {val_r2:.2f}, MAPE: {val_mape*100:.2f}%")
print(f"Test RMSE: {test_rmse:.3f}, R²: {test_r2:.2f}, MAPE: {test_mape*100:.2f}%")
```




```
Validation RMSE: 1.751, R²: 0.38, MAPE: 6.09%
Test RMSE: 2.341, R²: 0.31, MAPE: 7.28%
```

In [42]:
```
# Create DataFrame for test set comparison
test_plot = pd.DataFrame({
    'Actual GWL': y_test,
```

```
    'Predicted GWL': y_test_pred
}, index=y_test.index)

# Sort by date
test_plot = test_plot.sort_index()

# Plot actual vs predicted GWL over time (2024 Test data)
plt.figure(figsize=(12, 4))
plt.plot(test_plot.index, test_plot['Actual GWL'], label='Actual GWL', color='blue', linewidth=1.5)
plt.plot(test_plot.index, test_plot['Predicted GWL'], label='Predicted GWL', color='orange', linestyle='--', linewidth=1.5)
plt.title('\nSupport Vector Machine: Actual vs Predicted Groundwater Level (2024)\n', fontsize=13)
plt.xlabel('Date')
plt.ylabel('Groundwater Level (ft)')
plt.legend()
plt.grid(True)
plt.tight_layout()
# plt.savefig('svm_timeseriesplot.png', dpi=300)
plt.show()
```



3 Results

3.1 Groundwater Level Projection under RCP 8.5

To evaluate long-term groundwater trends under the RCP 8.5 climate scenario, I applied the trained neural network model to future climate projections from the NASA NEX-GDDP dataset (2025–2099). The figure visualizes the **observed groundwater levels (2005–2024)** in black and the **projected levels (2025–2099)** in blue. A red dashed line represents the fitted **linear trend** across the entire time range.

Despite year-to-year fluctuations, the linear trend line indicates a **slight upward trend** in groundwater levels over the coming decades.

3.2 Groundwater Level Statistics

Observed and projected groundwater levels were compared using annual and overall averages to assess potential changes:

- The **average groundwater level** is projected to **increase** from 21.39 ft to 21.81 ft, which may signal a decline in overall water availability.
- The **lowest annual average GWL** also **increases**, meaning that even during the wettest years, groundwater levels are not as shallow as before.
- The **highest annual average GWL** remains **relatively stable**, indicating that the driest-year conditions (deepest water levels) are not getting significantly worse.

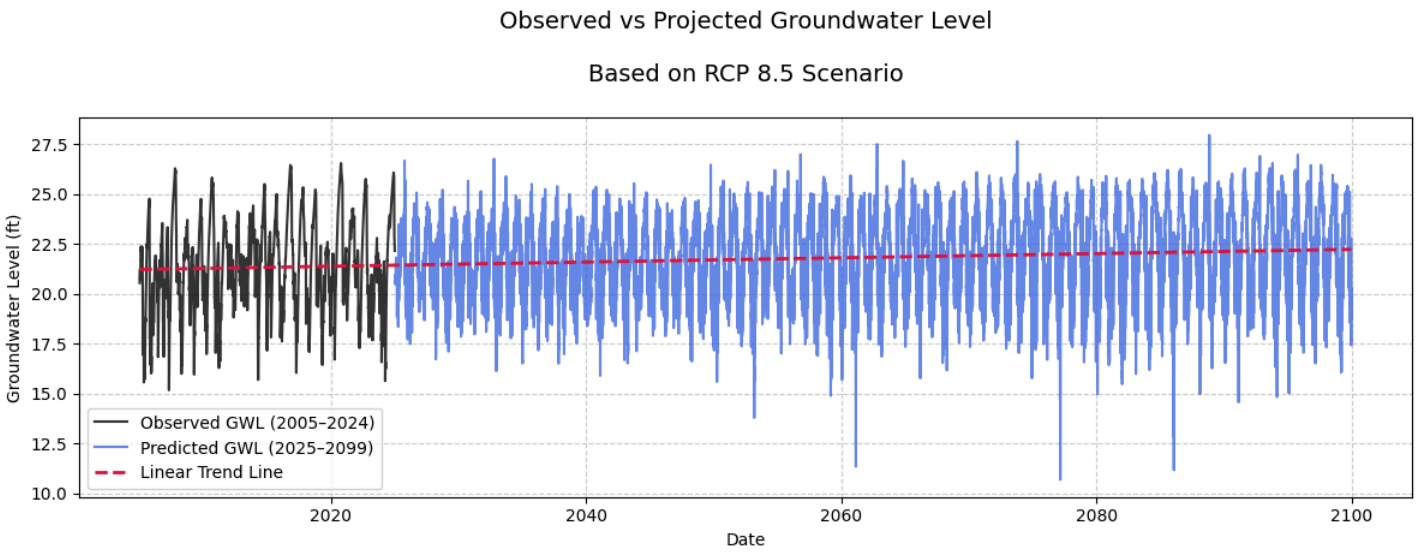
In summary, groundwater levels are projected to deepen overall, with reduced recovery during wet periods and persistent dryness during dry years. This indicates a gradual decline in groundwater accessibility under the RCP 8.5 scenario.

In [43]:
```
# Predict GWL
predicted_gwl = nn_model.predict(X_future_scaled)
```

In [44]:
```
from sklearn.linear_model import LinearRegression
```

In [45]:
```
# Create a DataFrame to store predicted GWL and corresponding dates
df_prediction = pd.DataFrame({
    'DATE': X_future_scaled.index,
    'Predicted_GWL': predicted_gwl
})
df_prediction.set_index('DATE', inplace=True)

# Combine historical and future predicted GWL into one DataFrame
df_combined = pd.DataFrame({
    'GWL': pd.concat([df_gwl['GWL'], df_prediction['Predicted_GWL']])
})

# Drop NA and ensure datetime index
df_trend = df_combined.dropna(subset=['GWL'])
df_trend.index = pd.to_datetime(df_trend.index)

# Convert index to number of days since the first date
X = (df_trend.index - df_trend.index[0]).days.values.reshape(-1, 1)
y = df_trend['GWL'].values

# Fit linear regression model
model = LinearRegression()
model.fit(X, y)
trend_line = model.predict(X)

# Plot
plt.figure(figsize=(12, 5))
plt.plot(df_gwl.index, df_gwl['GWL'], label='Observed GWL (2005-2024)', color='black', linewidth=1.5, alpha=0.8)
plt.plot(df_prediction.index, df_prediction['Predicted_GWL'], label='Predicted GWL (2025-2099)', color='royalblue', linewidth=1.5, alpha=0.8)
plt.plot(df_trend.index, trend_line, label='Linear Trend Line', color='crimson', linewidth=2, linestyle='--')
```

```
plt.title("\nObserved vs Projected Groundwater Level\n\nBased on RCP 8.5 Scenario\n", fontsize=14)
plt.xlabel("Date")
plt.ylabel("Groundwater Level (ft)")
plt.grid(True, linestyle='--', alpha=0.6)
plt.legend()
plt.tight_layout()
# plt.savefig('rcp.png', dpi=300)
plt.show()
```



In [46]:
```
# Ensure both DataFrames have DatetimeIndex
if not isinstance(df_gwl.index, pd.DatetimeIndex):
    df_gwl.index = pd.to_datetime(df_gwl.index)

if not isinstance(df_prediction.index, pd.DatetimeIndex):
    df_prediction.index = pd.to_datetime(df_prediction.index)

# Calculate average GWL
mean_gwl_past = df_gwl['GWL'].mean()
mean_gwl_future = df_prediction['Predicted_GWL'].mean()

# Calculate the lowest annual average GWL
min_yearly_past = df_gwl['GWL'].resample('Y').mean().min()
min_yearly_future = df_prediction['Predicted_GWL'].resample('Y').mean().min()

# Calculate the highest annual average GWL
max_yearly_past = df_gwl['GWL'].resample('Y').mean().max()
max_yearly_future = df_prediction['Predicted_GWL'].resample('Y').mean().max()

# Determine trend description
def trend_text(past, future, threshold=0.05):
    if future > past + threshold:
        return "Increase"
    elif future < past - threshold:
        return "Decrease"
    else:
        return "Stable"

# Display results
print("\nGroundwater Level Comparison:\n")
print(f"{' ':<40}{'2005-2024':>12}{'2025-2099':>12}{'Trend':>12}")
print("-" * 76)
print(f"{'Average GWL':<40}{mean_gwl_past:12.2f}{mean_gwl_future:12.2f}{trend_text(mean_gwl_past, mean_gwl_future):>12}")
print(f"{'Lowest Annual Avg GWL':<40}{min_yearly_past:12.2f}{min_yearly_future:12.2f}{trend_text(min_yearly_past, min_yearly_future):>12}")
print(f"{'Highest Annual Avg GWL':<40}{max_yearly_past:12.2f}{max_yearly_future:12.2f}{trend_text(max_yearly_past, max_yearly_future):>12}")
```

```
Groundwater Level Comparison:

                                        2005-2024   2025-2099       Trend
----------------------------------------------------------------------------
Average GWL                                 21.39       21.81    Increase
Lowest Annual Avg GWL                       20.23       21.28    Increase
Highest Annual Avg GWL                      22.47       22.46      Stable
```

---

4 Limitations

While this project aims to project groundwater levels in Penobscot County under the RCP8.5 scenario, several limitations should be acknowledged.

First, there is a spatial mismatch between the meteorological data and groundwater measurements. Although both the Bangor weather station and the Kenduskeag well are located within Penobscot County, they are approximately 8.5 miles apart, with an elevation difference of about 46.7 feet. This geographic separation may introduce discrepancies between local climatic conditions and actual groundwater responses.

Second, as Maine is a northern state where snow accumulation and melt significantly influence groundwater recharge, the exclusion of snow-related variables poses a limitation. The climate projections used in this study (NASA NEX-GDDP) include only temperature and precipitation data, without explicit snow data. This omission may reduce the model's ability to fully capture seasonal recharge dynamics.

Lastly, data completeness varies across monitoring sites. While the Kenduskeag well had minimal missing data (only 5 out of 7305 days), many other locations in the region suffer from more substantial gaps. Incomplete time series can hinder model training and reduce predictive reliability, especially when missing values are not randomly distributed.

Future work could address these issues by incorporating additional climatic variables (e.g., snow depth or snow water equivalent), improving spatial alignment between datasets, and exploring imputation or data augmentation techniques for regions with sparse records.

---

5 Conclusion

The analysis reveals that under the RCP8.5 scenario, groundwater levels in Penobscot County are projected to become deeper overall by the end of the 21st century. Although year-to-year variability remains, the long-term trend suggests a modest but consistent decline in groundwater accessibility. The average groundwater level is expected to rise from 21.39 ft to 21.81 ft, with the lowest annual averages increasing as well, indicating weaker recharge even during the wettest years. These findings underscore the importance of incorporating climate projections into long-term water resource planning and highlight the potential vulnerability of northern regions like Maine to climate-driven hydrologic changes.

---

6 References

1. IPCC. *Climate Change 2021: The Physical Science Basis*. Contribution of Working Group I to the Sixth Assessment Report of the Intergovernmental Panel on Climate Change. Cambridge University Press, 2021. https://doi.org/10.1017/9781009157896.

2. Moss, Richard H., et al. "The next generation of scenarios for climate change research and assessment." *Nature*, vol. 463, no. 7282, 2010, pp. 747–756. https://doi.org/10.1038/nature08823.

3. Jing, Ming, et al. "Assessing the Response of Groundwater Quantity and Travel Time Distribution to 1.5, 2, and 3 °C Global Warming in a Mesoscale Central German Basin." *Hydrology and Earth System Sciences*, vol. 24, no. 5, 2020, pp. 1511–1526. https://doi.org/10.5194/hess-24-1511-2020.

4. Danesh, Azin Shahni, et al. "Climate Change Impact Assessment on Water Resources in Iran: Applying Dynamic and Statistical Downscaling Methods." *Journal of Water and Climate Change*, vol. 7, no. 3, 2016, pp. 551–567. https://doi.org/10.2166/wcc.2016.045.

5. "Daily Summaries Station Details – GHCND:USW00014606." *NOAA Climate Data Online*, National Centers for Environmental Information, www.ncdc.noaa.gov/cdo-web/datasets/GHCND/stations/GHCND:USW00014606/detail. Accessed 13 May 2025.

6. "USGS 445319068560101 ME-PEW456 Kenduskeag, ME." *National Water Information System*, U.S. Geological Survey, https://waterdata.usgs.gov/nwis/inventory?site_no=445319068560101. Accessed 13 May 2025.

7. "NEX-GDDP: NASA Earth Exchange Global Daily Downscaled Projections." *NASA Goddard Space Flight Center*, NASA Earth Exchange (NEX), https://www.nccs.nasa.gov/services/data-collections/land-based-products/nex-gddp. Accessed 13 May 2025.

8. James, Gareth, et al. "Linear Regression." *An Introduction to Statistical Learning with Applications in R*. Springer, 2021, pp. 59–128. https://doi.org/10.1007/978-1-0716-1418-1_3.

9. Géron, Aurélien. "Introduction to Artificial Neural Networks with Keras." *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. 2nd ed., O'Reilly Media, 2019, pp. 277–324.

10. Breiman, Leo. "Random forests." *Machine Learning*, vol. 45, no. 1, 2001, pp. 5–32. https://doi.org/10.1023/A:1010933404324.

11. Smola, Alex J., and Schölkopf, Bernhard. "A tutorial on support vector regression." *Statistics and Computing*, vol. 14, no. 3, 2004, pp. 199–222. https://doi.org/10.1023/B:STCO.0000035301.49549.88.
