FRANK ASTO
DS 260
DATA MINING

PROJECT: WATER USE IN GROUNDWATER MANAGEMENT DISTRICTS

```
Groundwater Management Districts 1, 3, and 4
```

ABSTRACT

This project looks at water use in Kansas Groundwater Management Districts (GMDs) 1, 3, and 4 (Western KS). Water use is an important consideration to determine the longevity of the aquifer's water supply. The goal of the project is to understand how the quantity of water use is related to aquifer decline.

1 INTRODUCTION

In order to examine patterns of water use in GMDs 1, 3, and 4 this project will use data from the Division of Water Resources that tracks annual water use as well as data on well depth levels from the WIZARD Water Well Levels Database.

2 BASELINE OR INITIAL ANALYSIS

To begin this project, the students and faculty in the Kansas Data Science Consortium will work to integrate water use data and water depth data at different levels of geography. For instance, the data could be integrated at the county level as well as different HUC levels (e.g. HUC 10).

Once the data is integrated, summary statistics should be prepared to show how use and well depth levels have changed over time. These summaries should be made both graphically and with tables.

3 FINAL ANALYSIS

Once the data integration and the summary statistics are complete, the final analysis of this project is to establish the relationship between water use and changes in well depth levels. Doing this may require incorporating additional data about precipitation levels to think about how much water is recharging the aquifer. An element to consider is that based on geological and other conditions that effect recharge rates.

4 FINAL GOALS & EVALUATION

The final goal of this project is to be able to establish the quantity of water use that would lead to sustainable water levels in the aquifer. These quantities will vary across geography in GMDs 1, 3, and 4 and the level of water use reduction that will lead to sustainable aquifer levels (well depth levels) will also vary with geography. A goal of this project is to provide estimates for these water level reductions and to document the process by which those estimates were achieved.

In this project, I used Jupyter Notebook as the development environment. I utilized Python with the Pandas library for data cleaning, Matplotlib for data visualization, and ipywidgets to add interactive elements. Throughout my analysis, I also incorporated other helpful libraries as needed. This marks the beginning of my project, where I focus on exploring and analyzing the data effectively.

```python
# Loading 3 csv files

import pandas as pd

#loading the 3 files
df_main = pd.read_csv("GMD_Water_Use_groundwaterbywell_county_cousub_sec.csv")
df_wizard = pd.read_csv("GMD Water Use_WIZARD Data.csv")
df_wimas = pd.read_csv("GMD Water Use_WIMASusebyHUC10.csv")

#size of rows and columns
print("Total Rows and Columns in GMD Water file:")
print(df_main.shape)

print("\nColumn Names: \n")
# Print all column names
for col in df_main.columns:
    print(col)

#size of rows and columns
print("\nTotal Rows and Columns in Wizard file:")
print(df_wizard.shape)

print("\nColumn Names: \n")
# Print all column names
for col in df_wizard.columns:
    print(col)

#size of rows and columns
print("\nTotal Rows and Columns in Wimas file:")
print(df_wimas.shape)

print("\nColumn Names: \n")
# Print all column names
for col in df_wimas.columns:
    print(col)
```

```
Total Rows and Columns in GMD Water file:
(51051, 245)

Column Names:

OID_
PDIV_ID
long_nad83
lat_nad83
gmd
county_abrev
source
active_20230507
AF_USED_1990
AF_USED_1991
AF_USED_1992
AF_USED_1993
```

```
# Make a copy of the full dataset
df_main_cols = df_main.copy()

# Drop columns I don't need
static_columns_to_drop = [
    'county_abrev', 'aquier_codes', 'COUSUB_NAME', 'COUSUB_GEOID',
    'PLSS_ID', 'PDIV_ID', 'OID_', 'source', 'active_20230507',
]

columns_to_drop = []
for col in static_columns_to_drop:
    if col in df_main_cols.columns:
        columns_to_drop.append(col)

df_main_cols.drop(columns=columns_to_drop, inplace=True)

# Drop ACRES columns (1990-2022)
acres_columns_to_drop = []
for year in range(1990, 2023):
    col_name = f'ACRES_{year}'
    if col_name in df_main_cols.columns:
        acres_columns_to_drop.append(col_name)

df_main_cols.drop(columns=acres_columns_to_drop, inplace=True)

# Preview cleaned data
df_main_cols.head()
```

	long_nad83	lat_nad83	gmd	AF_USED_1990	AF_USED_1991	AF_USED_1992	AF_USED_1993	AF_USED_1994	AF_USED_1995	AF_USED_1996	...	AF_USED_REC_2015
0	-100.44270	37.52048	3.0	203.614535	232.099948	97.22235	89.985914	141.856247	155.997680	178.000000	...	0.0
1	-97.05257	38.18752	NaN	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	...	0.0
2	-100.71220	39.91526	NaN	29.875618	13.183940	0.00000	5.781784	17.750444	24.710681	11.158474	...	0.0
3	-96.62501	38.02865	NaN	0.000000	0.000000	0.00000	0.000000	0.000000	0.000000	0.000000	...	0.0
4	-99.15205	38.48850	NaN	98.728560	99.187052	0.00000	0.000000	82.642066	102.028228	0.000000	...	0.0

5 rows × 203 columns

```
# Print data type of df_main_cols
pd.set_option('display.max_rows', None)
print(df_main_cols.dtypes)
```

```
long_nad83          float64
lat_nad83           float64
gmd                 float64
AF_USED_1990        float64
AF_USED_1991        float64
AF_USED_1992        float64
AF_USED_1993        float64
AF_USED_1994        float64
AF_USED_1995        float64
AF_USED_1996        float64
AF_USED_1997        float64
AF_USED_1998        float64
AF_USED_1999        float64
AF_USED_2000        float64
AF_USED_2001        float64
AF_USED_2002        float64
AF_USED_2003        float64
AF_USED_2004        float64
```

```
#checking for null values in df_main_cols
print(df_main_cols.isnull().sum())
```

```
long_nad83              0
lat_nad83               0
gmd                 17256
AF_USED_1990            0
AF_USED_1991            0
AF_USED_1992            0
AF_USED_1993            0
AF_USED_1994            0
AF_USED_1995            0
AF_USED_1996            0
AF_USED_1997            0
AF_USED_1998            0
AF_USED_1999            0
AF_USED_2000            0
AF_USED_2001            0
AF_USED_2002            0
AF_USED_2003            0
AF_USED_2004            0
```

While checking for null values, I discovered that the GMD column contains 17,256 missing entries, which I need to address. To help resolve this issue, I found a reliable dataset online from the State of Kansas, which includes information on Groundwater Management Districts (GMDs), county names, county codes, and other relevant details.

01/05/2010

County List

State Number	State	API County No.	County Name	Cnty Abbr	District
15	KS	1	ALLEN	AL	03
15	KS	3	ANDERSON	AN	03
15	KS	5	ATCHISON	AT	03
15	KS	7	BARBER	BA	01
15	KS	9	BARTON	BT	04
15	KS	11	BOURBON	BB	03
15	KS	13	BROWN	BR	03
15	KS	15	BUTLER	BU	02
15	KS	17	CHASE	CS	02
15	KS	19	CHAUTAUQUA	CQ	03
15	KS	21	CHEROKEE	CK	03
15	KS	23	CHEYENNE	CN	04
15	KS	25	CLARK	CA	01
15	KS	27	CLAY	CY	02
15	KS	29	CLOUD	CD	02
15	KS	31	COFFEY	CF	03
15	KS	33	COMANCHE	CM	01
15	KS	35	COWLEY	CL	02
15	KS	37	CRAWFORD	CR	03
15	KS	39	DECATUR	DC	04
15	KS	41	DICKINSON	DK	02
15	KS	43	DONIPHAN	DP	03
15	KS	45	DOUGLAS	DG	03
15	KS	47	EDWARDS	ED	01
15	KS	49	ELK	EK	03
15	KS	51	ELLIS	EL	04
15	KS	53	ELLSWORTH	EW	02
15	KS	55	FINNEY	FI	01
15	KS	57	FORD	FO	01
15	KS	59	FRANKLIN	FR	03
15	KS	61	GEARY	GE	02
15	KS	63	GOVE	GO	04
15	KS	65	GRAHAM	GH	04
15	KS	67	GRANT	GT	01
15	KS	69	GRAY	GY	01
15	KS	71	GREELEY	GL	01
15	KS	73	GREENWOOD	GW	03
15	KS	75	HAMILTON	HM	01
15	KS	77	HARPER	HP	02
15	KS	79	HARVEY	HV	02
15	KS	81	HASKELL	HS	01
15	KS	83	HODGEMAN	HG	01
15	KS	85	JACKSON	JA	03

Using this information, I created a dictionary (hash table) to map the missing GMD values, using the county names as reference keys for accurate assignment.

```
# I have 17256 missing values for GMD, but I have the County Names, I'll make a dictionary to map it with the missing district

county_to_gmd = {
    'Allen': 3, 'Anderson': 3, 'Atchison': 3, 'Barber': 1, 'Barton': 4,
    'Bourbon': 3, 'Brown': 3, 'Butler': 2, 'Chase': 2, 'Chautauqua': 3,
    'Cherokee': 3, 'Cheyenne': 4, 'Clark': 1, 'Clay': 2, 'Cloud': 2,
    'Coffey': 3, 'Comanche': 1, 'Cowley': 2, 'Crawford': 3, 'Decatur': 4,
    'Dickinson': 2, 'Doniphan': 3, 'Douglas': 3, 'Edwards': 1, 'Elk': 3,
    'Ellis': 4, 'Ellsworth': 2, 'Finney': 1, 'Ford': 1, 'Franklin': 3,
    'Geary': 2, 'Gove': 4, 'Graham': 4, 'Grant': 1, 'Gray': 1,
    'Greeley': 1, 'Greenwood': 3, 'Hamilton': 1, 'Harper': 2, 'Harvey': 2,
    'Haskell': 1, 'Hodgeman': 1, 'Jackson': 3, 'Jefferson': 3, 'Jewell': 2,
    'Johnson': 3, 'Kearny': 1, 'Kingman': 2, 'Kiowa': 1, 'Labette': 3,
    'Lane': 1, 'Leavenworth': 3, 'Lincoln': 2, 'Linn': 3, 'Logan': 4,
    'Lyon': 3, 'Mcpherson': 2, 'McPherson': 2, 'Marion': 2, 'Marshall': 2, 'Meade': 1,
    'Miami': 3, 'Mitchell': 2, 'Montgomery': 3, 'Morris': 2, 'Morton': 1,
    'Nemaha': 3, 'Neosho': 3, 'Ness': 1, 'Norton': 4, 'Osage': 3,
    'Osborne': 4, 'Ottawa': 2, 'Pawnee': 1, 'Phillips': 4, 'Pottawatomie': 3,
    'Pratt': 1, 'Rawlins': 4, 'Reno': 2, 'Republic': 2, 'Rice': 2,
    'Riley': 2, 'Rooks': 4, 'Rush': 1, 'Russell': 4, 'Saline': 2,
    'Scott': 1, 'Sedgwick': 2, 'Seward': 1, 'Shawnee': 3, 'Sheridan': 4,
    'Sherman': 4, 'Smith': 4, 'Stafford': 1, 'Stanton': 1, 'Stevens': 1,
    'Sumner': 2, 'Thomas': 4, 'Trego': 4, 'Wabaunsee': 3, 'Wallace': 4,
    'Washington': 2, 'Wichita': 1, 'Wilson': 3, 'Woodson': 3, 'Wyandotte': 3
}
```

```
# Split out the two subsets
has_gmd    = df_main_cols[df_main_cols['gmd'].notna()].copy()
missing_gmd = df_main_cols[df_main_cols['gmd'].isna()].copy()

# Fill the missing ones
missing_gmd['gmd'] = missing_gmd.apply(
    lambda row: county_to_gmd.get(row['COUNTY_NAME'], row['gmd']),
    axis=1)

# Re-combine
df_main_cols = pd.concat([has_gmd, missing_gmd], ignore_index=True)
```

```
# this is again to check for missing values, 3 for gmd and 7 for county names, so I'll drop them
pd.set_option('display.max_rows', None)
print(df_main_cols.isnull().sum())
```

```
long_nad83          0
lat_nad83           0
gmd                 3
AF_USED_1990        0
AF_USED_1991        0
AF_USED_1992        0
AF_USED_1993        0
AF_USED_1994        0
AF_USED_1995        0
AF_USED_1996        0
AF_USED_1997        0
AF_USED_1998        0
AF_USED_1999        0
AF_USED_2000        0
AF_USED_2001        0
AF_USED_2002        0
AF_USED_2003        0
AF_USED_2004        0
```

```
# removing the missing data from the columns
df_main_cols = df_main_cols.dropna(subset=['gmd', 'COUNTY_NAME', 'COUNTY_GEOID'])
```

After mapping the missing values in the GMD column, I rechecked the data and found that only 3 null values remained. Additionally, I identified 7 missing entries in the COUNTY_NAME and COUNTY_GEOID columns, which I decided to remove to ensure data integrity.

Next, I reviewed my dataset again for any remaining null values, then examined the unique values in the GMD column. Since my analysis focuses only on districts 1, 3, and 4, I filtered the dataset to include only records where GMD is 1, 3, or 4. After that, I checked the shape of the dataset to see how many records remained. Finally, I removed any duplicate entries and rechecked the shape to confirm the cleanup was successful.

```
# check again
pd.set_option('display.max_rows', None)
print(df_main_cols.isnull().sum())
```

```
long_nad83          0
lat_nad83           0
gmd                 0
AF_USED_1990        0
AF_USED_1991        0
AF_USED_1992        0
AF_USED_1993        0
AF_USED_1994        0
AF_USED_1995        0
AF_USED_1996        0
AF_USED_1997        0
AF_USED_1998        0
AF_USED_1999        0
AF_USED_2000        0
AF_USED_2001        0
AF_USED_2002        0
AF_USED_2003        0
AF_USED_2004        0
```

```
#check unique GMD - districts
df_main_cols["gmd"].unique()
```

```
array([3., 1., 4., 5., 2.])
```

```
#keeping only 1, 3, 4 on GMD
df_main_cols = df_main_cols[df_main_cols['gmd'].isin([1, 3, 4])]
```

```
#check again unique GMD - districts
df_main_cols["gmd"].unique()
```

```
array([3., 1., 4.])
```

```
#checking count of rows in dataframe
print("Total Rows and Columns in GMD Water file:")
print(df_main_cols.shape)
```

```
Total Rows and Columns in GMD Water file:
(32398, 203)
```

```
#remove duplicates on df_main_cols
df_main_cols = df_main_cols.drop_duplicates()
```

```
#checking count of rows in dataframe
print("Total Rows and Columns in GMD Water file:")
print(df_main_cols.shape)
#df_main_cols.to_csv('GMD_Water_Use_groundwaterbywell_county_cousub_sec1.csv')
```

```
Total Rows and Columns in GMD Water file:
(32267, 203)
```

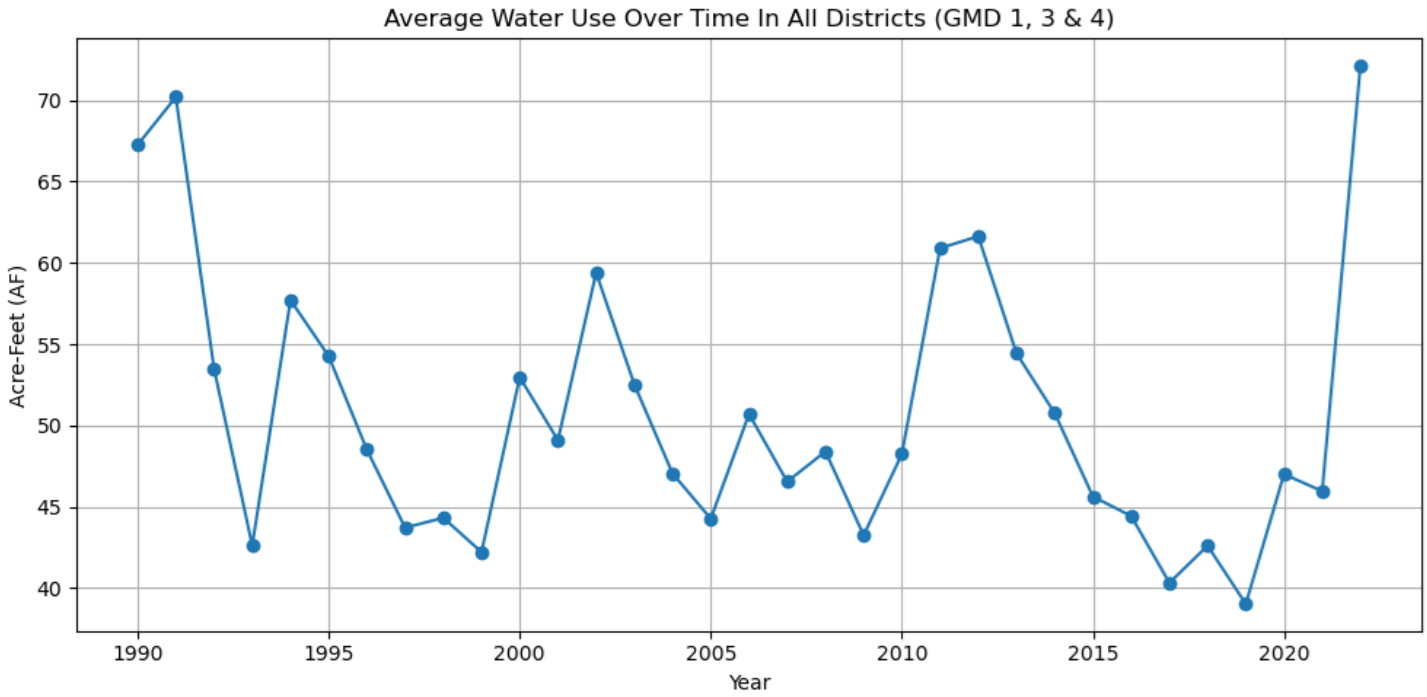
Now, I will group the data by COUNTY_NAME and GMD, calculating the mean water use for each year to summarize the data at the county and district level.

```python
# Load your data
df = pd.read_csv('GMD_Water_Use_groundwaterbywell_county_cousub_sec1.csv')

# Identify the annual-use columns
af_cols = [c for c in df.columns if c.startswith('AF_')]

# Group by county and GMD, taking the mean of each AF_ column
avg_by_county_gmd = (
    df
    .groupby(['COUNTY_GEOID', 'COUNTY_NAME', 'gmd'])[af_cols]
    .mean()
    .reset_index()
)

print(avg_by_county_gmd.head())
avg_by_county_gmd.to_csv('GMD_Water_Use_avg_by_county_gmd.csv', index=False)
```

```
   COUNTY_GEOID COUNTY_NAME  gmd  AF_USED_1990  AF_USED_1991  AF_USED_1992  \
0       20001.0       Allen  3.0      0.071549      0.138758      0.135785
1       20003.0    Anderson  3.0      2.431684      2.124594      2.285964
2       20005.0    Atchison  3.0     10.777812     10.323945      2.745418
3       20007.0      Barber  1.0     22.509554     21.968067     16.985252
4       20009.0      Barton  4.0      2.012269      2.200495      2.134358

   AF_USED_1993  AF_USED_1994  AF_USED_1995  AF_USED_1996  ...  \
0      0.068392      0.068392      0.068392      0.068392  ...
1      2.638711      3.034919      2.228344      2.249087  ...
2      3.284701      4.946841      3.546237      3.307811  ...
3     15.494719     21.522428     16.191125     17.145777  ...
4      1.676959      1.951272      1.489780      1.690621  ...

   AF_USED_REC_2013  AF_USED_REC_2014  AF_USED_REC_2015  AF_USED_REC_2016  \
0          0.000000          0.000000          0.000000          0.000000
1          0.000000          0.000000          0.000000          0.000000
2          6.941531          0.000000          1.880302          0.036096
3          0.375137          2.251872          1.909810          1.886355
4          0.000000          0.000000          0.000000          0.000000

   AF_USED_REC_2017  AF_USED_REC_2018  AF_USED_REC_2019  AF_USED_REC_2020  \
0          0.000000          0.000000          0.000000          0.000000
1          0.000000          0.000000          0.000000          0.000000
2          1.352697          1.290199          0.000000          1.842209
3          1.136121          0.437396          0.072338          0.123259
4          0.000000          0.000000          0.000000          0.000000

   AF_USED_REC_2021  AF_USED_REC_2022
0          0.000000          0.000000
1          0.000000          0.000000
2          3.554307          2.929472
3          0.361107          1.537949
4          0.000000          0.000000

[5 rows x 201 columns]
```

With the data now cleaned and organized, I will begin exploratory data analysis, starting by calculating the total average water use for each year to identify overall trends over time.

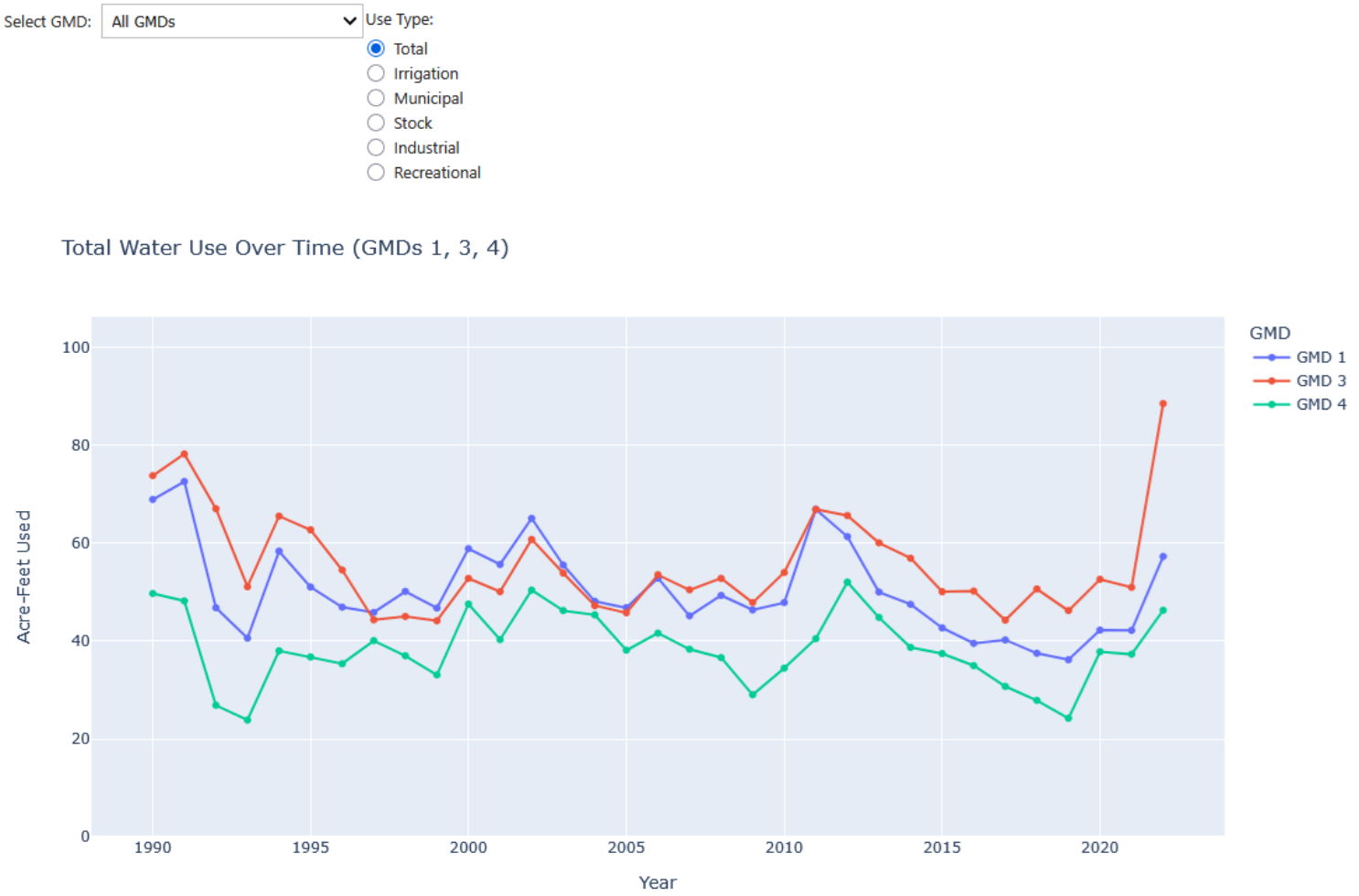
```python
import pandas as pd
import matplotlib.pyplot as plt

# Load the already-saved, smaller dataset
df = pd.read_csv('GMD_Water_Use_avg_by_county_gmd.csv')

# Identify only the AF_USED_ columns with a numeric year suffix
af_columns = [
    col for col in df.columns
    if col.startswith("AF_USED_") and col[8:].isdigit()
]

# Extract the years for the x-axis
years = [int(col.split("_")[-1]) for col in af_columns]

# Compute the mean across all rows (i.e. across all county+GMD groups) for each year
avg_af_per_year = df[af_columns].mean()

# Plot the time series
plt.figure(figsize=(10, 5))
plt.plot(years, avg_af_per_year, marker="o")
plt.title("Average Water Use Over Time In All Districts (GMD 1, 3 & 4)")
plt.xlabel("Year")
plt.ylabel("Acre-Feet (AF)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



Now that I'm beginning to gain some insights and a clearer understanding of the data, I'm moving forward with a more detailed analysis. In this step, I will filter the data by different GMDs (1, 3, 4, and all combined) and compare water use across various categories, including Irrigation, Municipal, Stock, Industrial, Recreational, and Total Use. This comparison will help highlight usage patterns across districts and water use types.

```
import pandas as pd
import plotly.express as px
import ipywidgets as widgets
from IPython.display import display
df_main_cols = pd.read_csv('GMD_Water_Use_avg_by_county_gmd.csv')

# Define a dictionary for use types and their column prefixes
use_types = {
    "Total": "AF_USED_",
    "Irrigation": "AF_USED_IRR_",
    "Municipal": "AF_USED_MUN_",
    "Stock": "AF_USED_STK_",
    "Industrial": "AF_USED_IND_",
    "Recreational": "AF_USED_REC_"
}

# Interactive function
def plot_gmd_use(gmd_number, use_type):
    prefix = use_types[use_type]
    af_columns = [col for col in df_main_cols.columns if col.startswith(prefix) and col[len(prefix):].isdigit()]
    years = [int(col.split('_')[-1]) for col in af_columns]

    if gmd_number == "All":
        # Plot all GMDs together
        df_list = []
        for gmd in [1, 3, 4]:
            gmd_df = df_main_cols[df_main_cols['gmd'] == gmd]
            avg_af = gmd_df[af_columns].mean()
            df_temp = pd.DataFrame({
                'Year': years,
                'Avg_AF_Used': avg_af.values,
                'GMD': f'GMD {gmd}'
            })
            df_list.append(df_temp)

        df_plot = pd.concat(df_list, ignore_index=True)
        fig = px.line(df_plot, x='Year', y='Avg_AF_Used', color='GMD',
                      title=f'{use_type} Water Use Over Time (GMDs 1, 3, 4)', markers=True)
        max_y = df_plot['Avg_AF_Used'].max()

    else:
        # Single GMD
        gmd_df = df_main_cols[df_main_cols['gmd'] == gmd_number]
        avg_af = gmd_df[af_columns].mean()
        df_plot = pd.DataFrame({
            'Year': years,
            'Avg_AF_Used': avg_af.values
        })
        fig = px.line(df_plot, x='Year', y='Avg_AF_Used',
                      title=f'{use_type} Water Use Over Time (GMD {gmd_number})', markers=True)
        max_y = df_plot['Avg_AF_Used'].max()
```

```
    fig.update_layout(
        height=600,
        xaxis_title='Year',
        yaxis_title='Acre-Feet Used',
        yaxis=dict(range=[0, max_y * 1.2])
    )
    fig.show()

# Create widgets
gmd_dropdown = widgets.Dropdown(
    options=[("GMD 1", 1), ("GMD 3", 3), ("GMD 4", 4), ("All GMDs", "All")],
    value=1,
    description='Select GMD:'
)

use_radio = widgets.RadioButtons(
    options=list(use_types.keys()),
    value="Total",
    description='Use Type:',
    layout={'width': 'max-content'}
)

# Display both widgets
ui = widgets.HBox([gmd_dropdown, use_radio])
out = widgets.interactive_output(plot_gmd_use, {'gmd_number': gmd_dropdown, 'use_type': use_radio})

display(ui, out)
```

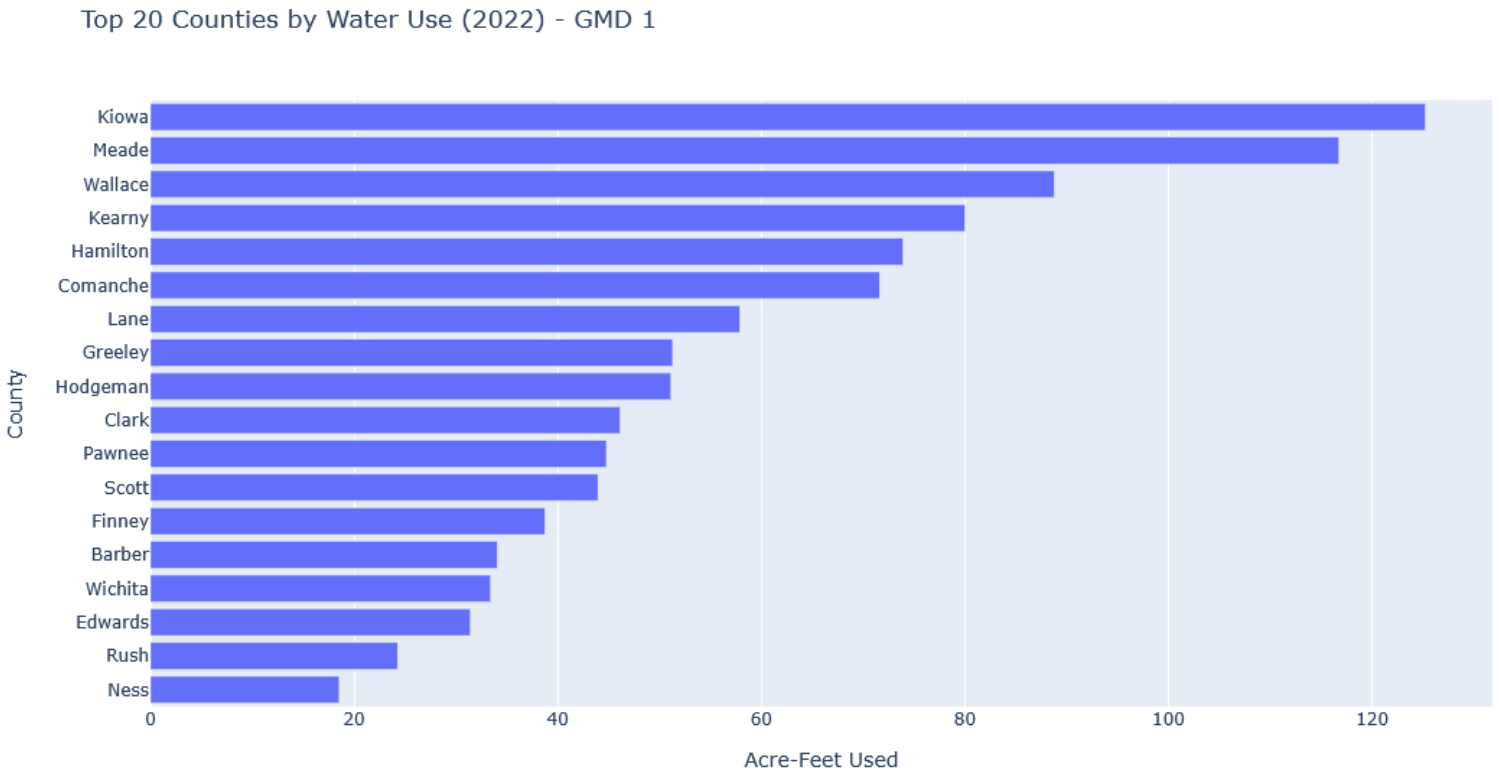


To provide a clearer picture of regional water usage, a bar chart breaks down county-level data across the three GMDs and the total combined results over time, allowing for an easy comparison of usage patterns and highlighting any significant variations between regions and years.

```
import pandas as pd
import plotly.express as px
import ipywidgets as widgets
from IPython.display import display

# Load data
df_main_cols = pd.read_csv("GMD_Water_Use_avg_by_county_gmd.csv")

# Extract all AF_USED_ columns with years
af_year_cols = [col for col in df_main_cols.columns if col.startswith("AF_USED_") and col[8:].isdigit()]
available_years = sorted([int(col.split("_")[-1]) for col in af_year_cols])

# Create widgets
gmd_dropdown = widgets.Dropdown(
    options=[("GMD 1", 1), ("GMD 3", 3), ("GMD 4", 4), ("All GMDs", "All")],
    value=1,
    description="Select GMD:"
)

year_dropdown = widgets.Dropdown(
    options=available_years,
    value=available_years[-1],  # default to most recent year
    description="Select Year:"
)
```

```python
def plot_top_counties(gmd_number, selected_year):
    year_col = f"AF_USED_{selected_year}"

    # Filter by GMD
    if gmd_number == "All":
        df_filtered = df_main_cols.copy()
    else:
        df_filtered = df_main_cols[df_main_cols["gmd"] == gmd_number]

    if year_col not in df_filtered.columns:
        print(f"Year column {year_col} not found.")
        return

    # Group by county and calculate mean water use
    df_grouped = (
        df_filtered
        .groupby("COUNTY_NAME")[year_col]
        .mean()
        .sort_values(ascending=False)
        .head(20)
        .reset_index()
    )

    fig = px.bar(
        df_grouped,
        x=year_col,
        y="COUNTY_NAME",
        orientation="h",
        title=f"Top 20 Counties by Water Use ({selected_year}) - GMD {gmd_number if gmd_number != 'All' else '1, 3, 4'}",
        labels={year_col: "Acre-Feet Used", "COUNTY_NAME": "County"}
    )

    fig.update_layout(yaxis=dict(autorange="reversed"), height=600)
    fig.show()

# Display widgets and chart
ui = widgets.HBox([gmd_dropdown, year_dropdown])
out = widgets.interactive_output(plot_top_counties, {"gmd_number": gmd_dropdown, "selected_year": year_dropdown})

display(ui, out)
```

Select GMD: GMD 1 Select Year: 2022

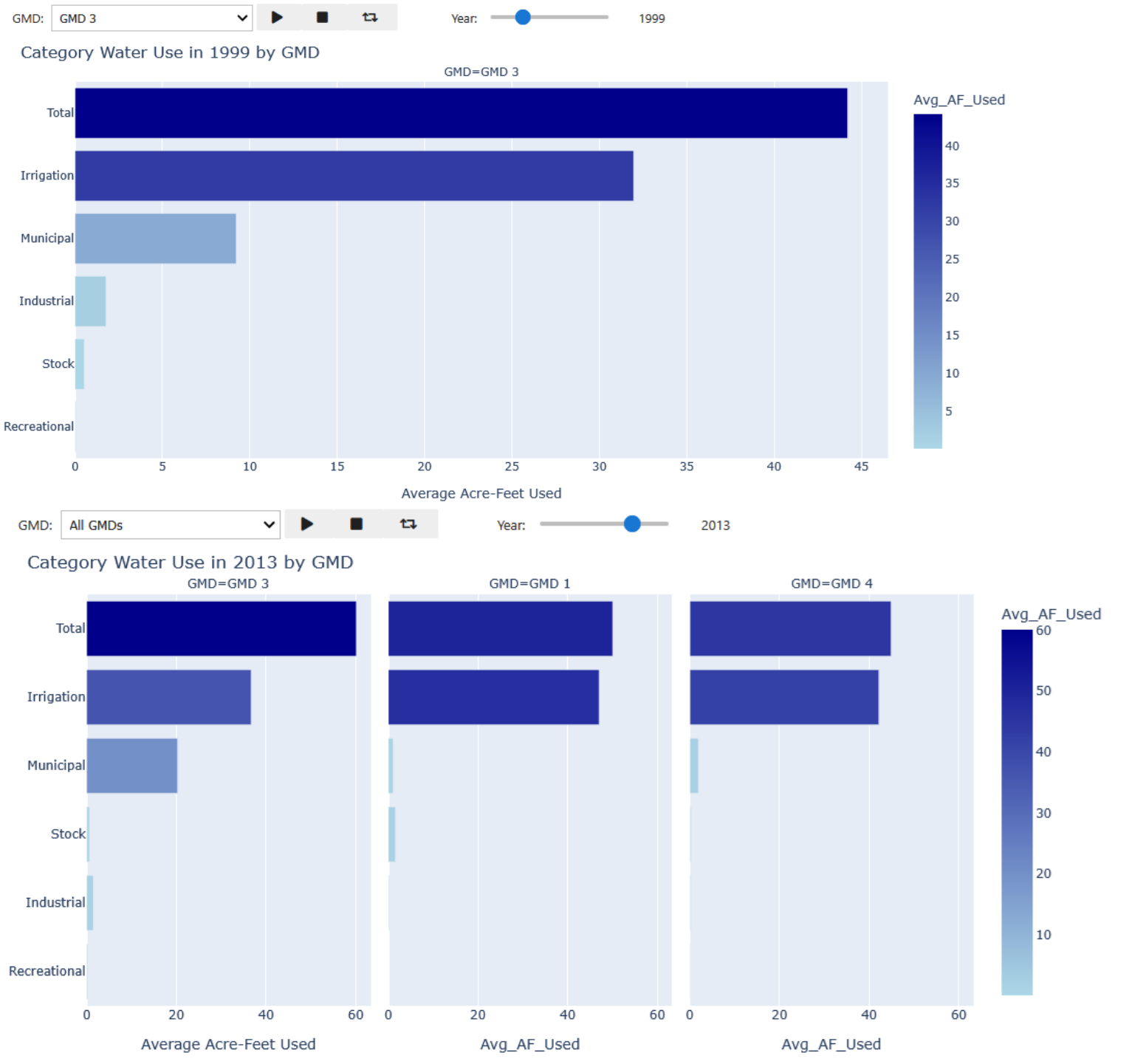


I also created a bar chart to visualize how each water use category compares across GMDs 1, 3, 4, and all combined over the years. To make the analysis more engaging, I implemented a play button animation, allowing users to see how water use in each category evolves over time.

```
# Creating a bar chart for the GMDs with categories and a Play animation for the years

# Imports
import pandas as pd
import plotly.express as px
import ipywidgets as widgets
from IPython.display import display

df_main_cols = pd.read_csv('GMD_Water_Use_avg_by_county_gmd.csv')

# Define use types and their column prefixes
use_types = {
    "Total":       "AF_USED_",
    "Irrigation":  "AF_USED_IRR_",
    "Municipal":   "AF_USED_MUN_",
    "Stock":       "AF_USED_STK_",
    "Industrial":  "AF_USED_IND_",
    "Recreational":"AF_USED_REC_"
}

# Derive years from the Total prefix
total_prefix = use_types['Total']
years = sorted(
    int(c.split('_')[-1])
    for c in df_main_cols.columns
    if c.startswith(total_prefix) and c[len(total_prefix):].isdigit()
)

# GMD selection dropdown
gmd_dropdown = widgets.Dropdown(
    options=[('GMD 1',1), ('GMD 3',3), ('GMD 4',4), ('All GMDs','All')],
    value=1,
    description='GMD:'
)
```

```python
# Year slider and play button for animation
year_slider = widgets.IntSlider(
    value=years[0],
    min=years[0],
    max=years[-1],
    step=1,
    description='Year:'
)
play_button = widgets.Play(
    value=years[0],
    min=years[0],
    max=years[-1],
    step=1,
    interval=400,  # speed of animation
    description='Play'
)
# Link play button and slider
widgets.jslink((play_button, 'value'), (year_slider, 'value'))

# Update function for animated bar chart
def update_bar_chart(gmd_selected, year_selected):
    data = []
    gmd_list = [1, 3, 4] if gmd_selected == 'All' else [gmd_selected]

    for gmd in gmd_list:
        df_g = df_main_cols[df_main_cols['gmd'] == gmd]
        for category, prefix in use_types.items():
            # find relevant columns for this category-year
            col_year = f"{prefix}{year_selected}"
            val = df_g[col_year].mean() if col_year in df_g.columns else 0
            data.append({
                'GMD': f'GMD {gmd}',
                'Category': category,
                'Avg_AF_Used': val
            })

    # Build DataFrame and sort descending by value
    df_bar = pd.DataFrame(data).sort_values('Avg_AF_Used', ascending=False)

    # Create horizontal bar chart
    fig = px.bar(
        df_bar,
        x='Avg_AF_Used',
        y='Category',
        orientation='h',
        color='Avg_AF_Used',
        color_continuous_scale=['lightblue', 'darkblue'],
        facet_col='GMD',
        category_orders={'Category': df_bar['Category'].tolist()},
        title=f"Category Water Use in {year_selected} by GMD"
    )
```

```
    fig.update_layout(
        xaxis_title='Average Acre-Feet Used',
        yaxis_title='',
        height=500,
        margin={'t': 50, 'b': 40}
    )
    fig.show()

# Display interactive UI with animation controls
ui = widgets.HBox([gmd_dropdown, play_button, year_slider])
out = widgets.interactive_output(
    update_bar_chart,
    {'gmd_selected': gmd_dropdown, 'year_selected': year_slider}
)

display(ui, out)
```



Next, I will begin cleaning the WIZARD dataset and performing some initial analysis. After that, I'll follow the same process with the WIMAS dataset to ensure both are ready for further exploration and comparison.

```
df_wizard.head()
```

	Unnamed: 0	X1	X2	X3	X4	X5	X6	X7	X8	X9	X1.1	wellid	date	depth	lat	long	countcode	town	tshp
0	3	FEB-21-1984	-222.69	-	-	Steel Tape	-	-	---	-	NaN	370136101360401	1984-02-21	-222.69	37.025356	-101.600938	129	ROLLA SE	35
1	4	JAN-25-1985	-210.08	-	-	Steel Tape	-	-	---	-	NaN	370136101360401	1985-01-25	-210.08	37.025356	-101.600938	129	ROLLA SE	35
2	5	JAN-07-1986	-211.54	-	-	Steel Tape	-	-	---	-	NaN	370136101360401	1986-01-07	-211.54	37.025356	-101.600938	129	ROLLA SE	35
3	6	JAN-07-1987	-212.80	-	-	Steel Tape	-	-	---	-	NaN	370136101360401	1987-01-07	-212.80	37.025356	-101.600938	129	ROLLA SE	35
4	7	JAN-25-1988	-212.20	-	-	Steel Tape	-	-	---	-	NaN	370136101360401	1988-01-25	-212.20	37.025356	-101.600938	129	ROLLA SE	35

```
# Select only the needed columns from the original df_wizard
df_wizard_cols = df_wizard[['wellid', 'date', 'depth', 'lat', 'long', 'countcode']].copy()
df_wizard_cols['gmd'] = None
df_wizard_cols['COUNTY_NAME'] = None
```

```
# Print data type of df_wizard_cols_cols
pd.set_option('display.max_rows', None)
print(df_wizard_cols.dtypes)
```

```
wellid          int64
date           object
depth         float64
lat           float64
long          float64
countcode       int64
gmd            object
COUNTY_NAME    object
dtype: object
```

```
print("Total Rows and Columns in Wizard file:")
print(df_wizard_cols.shape)
```

```
Total Rows and Columns in Wizard file:
(627522, 8)
```

```
#remove duplicates on df_wizard_cols
df_wizard_cols = df_wizard_cols.drop_duplicates()
```

```
print("Total Rows and Columns in Wizard file:")
print(df_wizard_cols.shape)
```

```
Total Rows and Columns in Wizard file:
(626836, 8)
```

```
# Convert date column to datetime
df_wizard_cols['date'] = pd.to_datetime(df_wizard_cols['date'], errors='coerce')

# Extract year from date
df_wizard_cols['year'] = df_wizard_cols['date'].dt.year

# Map countcode to COUNTY_NAME
countcode_to_county = {
    1: 'Allen', 3: 'Anderson', 5: 'Atchison', 7: 'Barber', 9: 'Barton', 11: 'Bourbon', 13: 'Brown',
    15: 'Butler', 17: 'Chase', 19: 'Chautauqua', 21: 'Cherokee', 23: 'Cheyenne', 25: 'Clark', 27: 'Clay',
    29: 'Cloud', 31: 'Coffey', 33: 'Comanche', 35: 'Cowley', 37: 'Crawford', 39: 'Decatur', 41: 'Dickinson',
    43: 'Doniphan', 45: 'Douglas', 47: 'Edwards', 49: 'Elk', 51: 'Ellis', 53: 'Ellsworth', 55: 'Finney',
    57: 'Ford', 59: 'Franklin', 61: 'Geary', 63: 'Gove', 65: 'Graham', 67: 'Grant', 69: 'Gray', 71: 'Greeley',
    73: 'Greenwood', 75: 'Hamilton', 77: 'Harper', 79: 'Harvey', 81: 'Haskell', 83: 'Hodgeman', 85: 'Jackson',
    87: 'Jefferson', 89: 'Jewell', 91: 'Johnson', 93: 'Kearny', 95: 'Kingman', 97: 'Kiowa', 99: 'Labette',
    101: 'Lane', 103: 'Leavenworth', 105: 'Lincoln', 107: 'Linn', 109: 'Logan', 111: 'Lyon', 113: 'McPherson',
    115: 'Marion', 117: 'Marshall', 119: 'Meade', 121: 'Miami', 123: 'Mitchell', 125: 'Montgomery',
    127: 'Morris', 129: 'Morton', 131: 'Nemaha', 133: 'Neosho', 135: 'Ness', 137: 'Norton', 139: 'Osage',
    141: 'Osborne', 143: 'Ottawa', 145: 'Pawnee', 147: 'Phillips', 149: 'Pottawatomie', 151: 'Pratt',
    153: 'Rawlins', 155: 'Reno', 157: 'Republic', 159: 'Rice', 161: 'Riley', 163: 'Rooks', 165: 'Rush',
    167: 'Russell', 169: 'Saline', 171: 'Scott', 173: 'Sedgwick', 175: 'Seward', 177: 'Shawnee', 179: 'Sheridan',
    181: 'Sherman', 183: 'Smith', 185: 'Stafford', 187: 'Stanton', 189: 'Stevens', 191: 'Sumner', 193: 'Thomas',
    195: 'Trego', 197: 'Wabaunsee', 199: 'Wallace', 201: 'Washington', 203: 'Wichita', 205: 'Wilson',
    207: 'Woodson', 209: 'Wyandotte'
}

df_wizard_cols['COUNTY_NAME'] = df_wizard_cols['countcode'].map(countcode_to_county)
df_wizard_cols.head()
```

	wellid	date	depth	lat	long	countcode	gmd	COUNTY_NAME	year
0	370136101360401	1984-02-21	-222.69	37.025356	-101.600938	129	None	Morton	1984
1	370136101360401	1985-01-25	-210.08	37.025356	-101.600938	129	None	Morton	1985
2	370136101360401	1986-01-07	-211.54	37.025356	-101.600938	129	None	Morton	1986
3	370136101360401	1987-01-07	-212.80	37.025356	-101.600938	129	None	Morton	1987
4	370136101360401	1988-01-25	-212.20	37.025356	-101.600938	129	None	Morton	1988

```
# Adding the values of gmd by mapping the county with the district code
df_wizard_cols['gmd'] = df_wizard_cols['COUNTY_NAME'].map(county_to_gmd)

# View the cleaned result
df_wizard_cols.head()
```

	wellid	date	depth	lat	long	countcode	gmd	COUNTY_NAME	year
0	370136101360401	1984-02-21	-222.69	37.025356	-101.600938	129	1	Morton	1984
1	370136101360401	1985-01-25	-210.08	37.025356	-101.600938	129	1	Morton	1985
2	370136101360401	1986-01-07	-211.54	37.025356	-101.600938	129	1	Morton	1986
3	370136101360401	1987-01-07	-212.80	37.025356	-101.600938	129	1	Morton	1987
4	370136101360401	1988-01-25	-212.20	37.025356	-101.600938	129	1	Morton	1988

```
df_wizard_cols["gmd"].unique()
```

```
array([1, 3, 2, 4], dtype=int64)
```

```
#keeping only 1, 3, 4 on GMD
df_wizard_cols = df_wizard_cols[df_wizard_cols['gmd'].isin([1, 3, 4])]
```

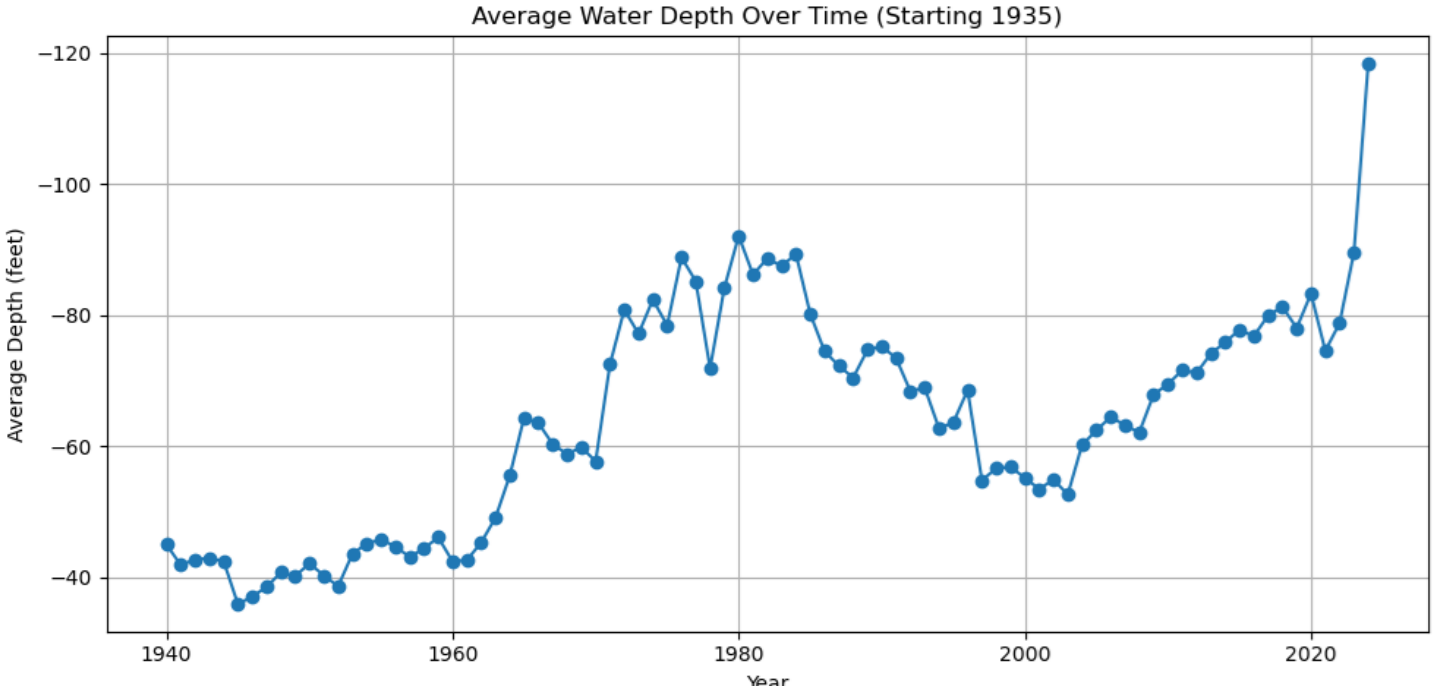
```
df_wizard_cols["gmd"].unique()
```

```
array([1, 3, 4], dtype=int64)
```

```
pd.set_option('display.max_rows', None)
print(df_wizard_cols.isnull().sum())
```

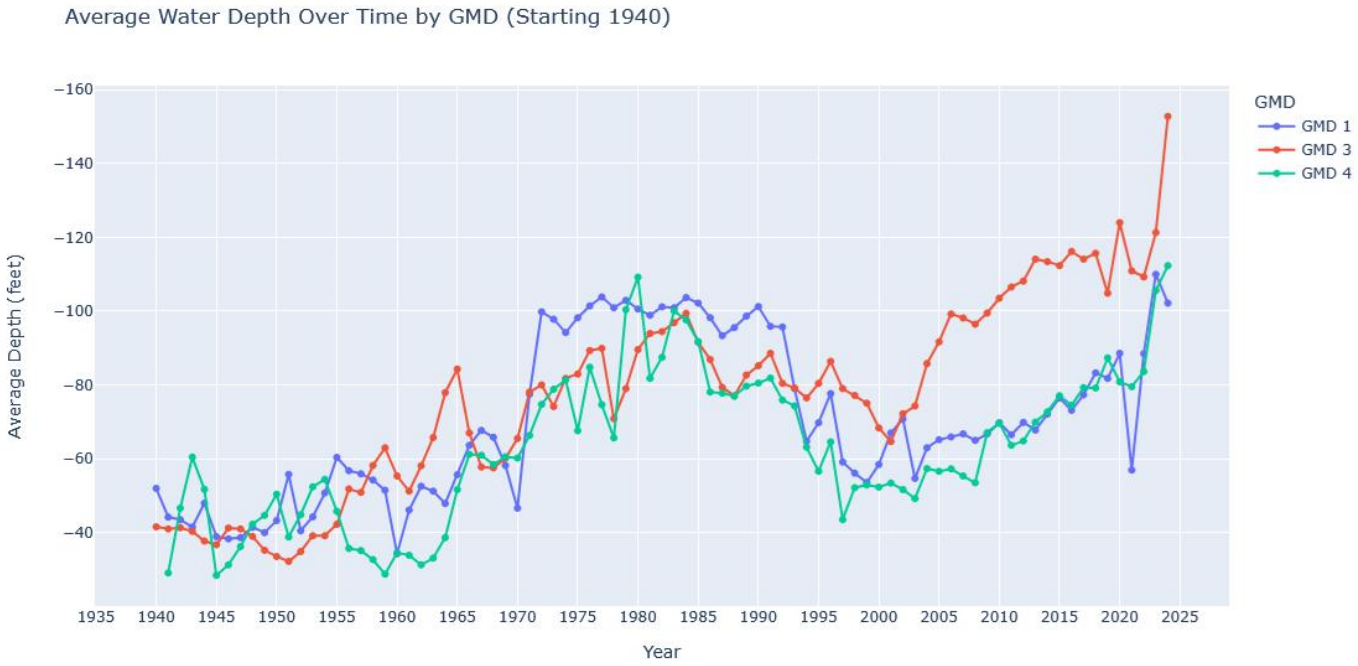
```
wellid         0
date           0
depth          0
lat            0
long           0
countcode      0
gmd            0
COUNTY_NAME    0
year           0
dtype: int64
```

```python
import matplotlib.pyplot as plt

# Filter data to include only years from 1940 and onward
depth_by_year_filtered = df_wizard_cols[df_wizard_cols['year'] >= 1940]

# Group by year and calculate the average depth
depth_by_year_avg = depth_by_year_filtered.groupby('year')['depth'].mean()

# Plot the results with reversed y-axis
plt.figure(figsize=(10, 5))
plt.plot(depth_by_year_avg.index, depth_by_year_avg.values, marker='o', linestyle='-')
plt.title("Average Water Depth Over Time (Starting 1935)")
plt.xlabel("Year")
plt.ylabel("Average Depth (feet)")
plt.gca().invert_yaxis()  # Reverses the y-axis
plt.grid(True)
plt.tight_layout()
plt.show()
```



```
import pandas as pd
import plotly.express as px
import ipywidgets as widgets
from IPython.display import display

# Map COUNTY_NAME to GMD using df_main_cols
county_to_gmd = df_main_cols[['COUNTY_NAME', 'gmd']].drop_duplicates().set_index('COUNTY_NAME')['gmd'].to_dict()
df_wizard_cols['gmd'] = df_wizard_cols['COUNTY_NAME'].map(county_to_gmd)

# Filter WIZARD data: years >= 1940 and GMDs 1, 3, 4
df_wizard_filtered = df_wizard_cols[
    (df_wizard_cols['year'] >= 1940) &
    (df_wizard_cols['gmd'].isin([1, 3, 4]))
]

# Define plot function
def plot_depth_by_gmd(gmd_selection):
    if gmd_selection == "All":
        fig_df = df_wizard_filtered.groupby(['year', 'gmd'])['depth'].mean().reset_index()
        fig_df['GMD'] = fig_df['gmd'].apply(lambda x: f'GMD {int(x)}')
        fig = px.line(
            fig_df, x='year', y='depth', color='GMD',
            title='Average Water Depth Over Time by GMD (Starting 1940)',
            markers=True
        )
    else:
        gmd_num = int(gmd_selection)
        gmd_df = df_wizard_filtered[df_wizard_filtered['gmd'] == gmd_num]
        avg_depth = gmd_df.groupby('year')['depth'].mean().reset_index()
        fig = px.line(
            avg_depth, x='year', y='depth',
            title=f'Average Water Depth Over Time - GMD {gmd_num} (Starting 1935)',
            markers=True
        )

    fig.update_layout(
        height=600,
        xaxis_title='Year',
        yaxis_title='Average Depth (feet)',
        yaxis=dict(autorange='reversed'),  # Shallower appears higher
        xaxis=dict(dtick=5)
    )
    fig.show()

# Dropdown for GMD selection
gmd_dropdown = widgets.Dropdown(
    options=[("GMD 1", "1"), ("GMD 3", "3"), ("GMD 4", "4"), ("All GMDs", "All")],
    value="1",
    description="Select GMD:"
)

# Show dropdown + plot
widgets.interact(plot_depth_by_gmd, gmd_selection=gmd_dropdown)
```

Select GMD: All GMDs



```
# Print wimas columns, for this file I am going to use all the columns
df_wimas_cols = df_wimas.copy()
df_wimas_cols.head()
```

	Unnamed: 0	HUC	AF_2022	AF_2021	AF_2020	AF_2019	AF_2018	AF_2017	AF_2016	AF_2015	...	AF_1998	AF_1997	AF_1996	AF_
0	1	1024000510	34.149111	42.209531	16.065011	17.664822	31.268279	49.350163	41.262111	4.099113	...	8.089894	8.722085	9.772258	5.34
1	2	1024000511	213.024201	247.547477	184.706783	174.046635	267.340981	295.580618	248.677092	146.852611	...	145.760486	199.176618	145.436411	162.92
2	3	1024000512	107.882029	84.079819	16.424734	30.498001	175.255135	181.701262	219.479099	188.557743	...	151.901268	124.718107	151.816812	167.03
3	4	1024000513	428.455085	391.641192	284.778718	108.238152	499.658359	493.424516	412.376819	190.564826	...	134.864463	208.361490	149.204084	185.47
4	5	1024000702	6610.387137	642.489279	660.358375	541.104192	677.537936	654.653912	618.448328	569.268948	...	503.635097	499.654136	470.712688	462.27

5 rows × 36 columns

```
# Print data type of df_wimas_cols
pd.set_option('display.max_rows', None)
print(df_wimas_cols.dtypes)
```

```
Unnamed: 0      int64
HUC             int64
AF_2022       float64
AF_2021       float64
AF_2020       float64
AF_2019       float64
AF_2018       float64
AF_2017       float64
AF_2016       float64
AF_2015       float64
AF_2014       float64
AF_2013       float64
AF_2012       float64
AF_2011       float64
AF_2010       float64
AF_2009       float64
AF_2008       float64
AF_2007       float64
```

```
#checking for null values in df_wimas_cols
print(df_wimas_cols.isnull().sum())
```

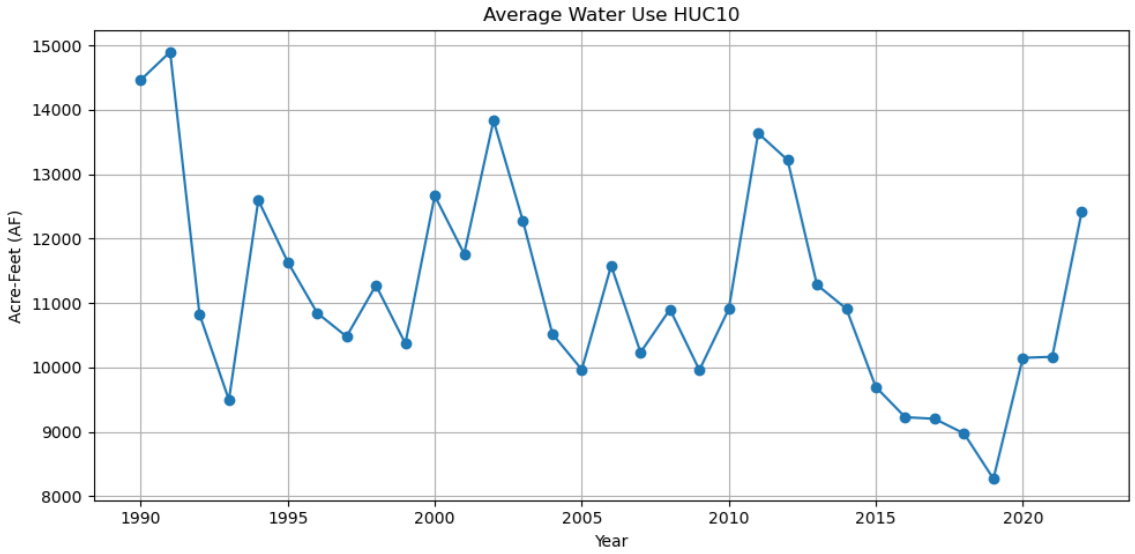
```
Unnamed: 0    0
HUC           0
AF_2022       0
AF_2021       0
AF_2020       0
AF_2019       0
AF_2018       0
AF_2017       0
AF_2016       0
AF_2015       0
AF_2014       0
AF_2013       0
AF_2012       0
AF_2011       0
AF_2010       0
AF_2009       0
AF_2008       0
AF_2007       0
```

```
#remove duplicates on df_wimas_cols
df_wimas_cols = df_wimas_cols.drop_duplicates()
```

```
#checking count of rows in dataframe
print("Total Rows and Columns in Wimas file:")
print(df_wimas_cols.shape)
```

```
Total Rows and Columns in Wimas file:
(332, 36)
```

```python
# Creating a simple line chart with average water use HUC10
import matplotlib.pyplot as plt

# SGet all AF_USED columns
af_columns = [col for col in df_wimas_cols.columns if col.startswith("AF_")]

# Extract years from the column names for x-axis
years = [int(col.split("_")[-1]) for col in af_columns]

# Calculate the mean per year
avg_af_per_year = df_wimas_cols[af_columns].mean()

# Plot with years on x-axis
plt.figure(figsize=(10, 5))
plt.plot(years, avg_af_per_year, marker="o")
plt.title("Average Water Use HUC10")
plt.xlabel("Year")
plt.ylabel("Acre-Feet (AF)")
plt.grid(True)
plt.tight_layout()
plt.show()
```



I'm creating an interactive map to explore water use and groundwater depth across Kansas. Users can select GMDs 1, 3, 4, or all, and switch between viewing water use by HUC10 regions or well depth data. Since the WIMAS dataset didn't include GMDs, I used a spatial join with HUC10 and GMD boundaries to add that information. I did the same for the WIZARD dataset by converting well coordinates into a GeoDataFrame and joining it with GMDs. I also extracted the year from the date column to support time-based analysis.

```python
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point

# Load data
wimas_df = df_wimas_cols.copy()
wizard_df = df_wizard_cols.copy()
huc10_gdf = gpd.read_file("HUC_10_Boundaries.geojson")
gmd_gdf = gpd.read_file("Groundwater_Management_Districts_(GMD).geojson")

# Ensure CRS matches between HUC10 and GMD
huc10_gdf = huc10_gdf.to_crs(gmd_gdf.crs)

# Spatial join - map HUC10s to GMDs
huc10_with_gmd = gpd.sjoin(huc10_gdf, gmd_gdf, how="inner", predicate="intersects")

# Clean HUC codes for merging with WIMAS
wimas_df["HUC"] = wimas_df["HUC"].astype(str)
huc10_with_gmd["HUC_10"] = huc10_with_gmd["HUC_10"].astype(str)

# Merge WIMAS water use data with HUC10 + GMD regions
merged_huc10 = huc10_with_gmd.merge(wimas_df, left_on="HUC_10", right_on="HUC")

# Convert WIZARD wells to GeoDataFrame
wizard_df["geometry"] = wizard_df.apply(lambda row: Point(row["long"], row["lat"]), axis=1)
wizard_gdf = gpd.GeoDataFrame(wizard_df, geometry="geometry", crs="EPSG:4326")
wizard_gdf = wizard_gdf.to_crs(gmd_gdf.crs)

# Spatial join - map wells to GMDs using correct GMD_ID
wizard_with_gmd = gpd.sjoin(wizard_gdf, gmd_gdf, how="inner", predicate="within")

# Extract year for filtering and analysis
wizard_with_gmd["year"] = pd.to_datetime(wizard_with_gmd["date"]).dt.year
```

```
# Preview merged HUC10 with GMD and water use
merged_huc10.head()
```

	OBJECTID_1	OBJECTID_left	HUC_8	HUC_10	HU_10_DS	HU_10_NAME	SHAPE_LENG	ACRES	SQ_MILES_left	Shape__Area_left	...	AF_1998	AF_1997
0	15	15	10250001	1025000108	1025000211	Dry Willow Creek-Arikaree River	0.567389	73869	115.420312	1.515661e+08	...	195.000000	176.767909
1	17	17	10250003	1025000310	1025000311	Bonny Reservoir-South Fork Republican River	0.322000	58455	91.335938	4.524742e+07	...	554.785463	438.234521
2	18	18	10250003	1025000311	1025000312	Cherry Creek-South Fork Republican RIver	0.797186	24436	38.181250	1.670696e+08	...	17056.956302	20777.393990
3	19	19	10250003	1025000311	1025000312	Cherry Creek-South Fork Republican River	2.122425	228334	356.771875	1.477500e+09	...	17056.956302	20777.393990
4	20	20	10250003	1025000312	1025000402	Hackberry Creek-South Fork Republican River	1.857240	184127	287.698438	1.245945e+09	...	6095.769505	6858.280291

5 rows × 62 columns

```
# Preview wells with GMD and depth
wizard_with_gmd.head()
```

	wellid	date	depth	lat	long	countcode	gmd	COUNTY_NAME	year	geometry	...	PERIMETER	GMD_	GMD_ID	NAME	SYM
0	370136101360401	1984-02-21	-222.69	37.025356	-101.600938	129	3.0	Morton	1984	POINT (-101.60094 37.02536)	...	742771	6	3	Southwest Kansas GMD #3	57
1	370136101360401	1985-01-25	-210.08	37.025356	-101.600938	129	3.0	Morton	1985	POINT (-101.60094 37.02536)	...	742771	6	3	Southwest Kansas GMD #3	57
2	370136101360401	1986-01-07	-211.54	37.025356	-101.600938	129	3.0	Morton	1986	POINT (-101.60094 37.02536)	...	742771	6	3	Southwest Kansas GMD #3	57
3	370136101360401	1987-01-07	-212.80	37.025356	-101.600938	129	3.0	Morton	1987	POINT (-101.60094 37.02536)	...	742771	6	3	Southwest Kansas GMD #3	57
4	370136101360401	1988-01-25	-212.20	37.025356	-101.600938	129	3.0	Morton	1988	POINT (-101.60094 37.02536)	...	742771	6	3	Southwest Kansas GMD #3	57

5 rows × 24 columns

```
# See what GMDs are available
merged_huc10["GMD_ID"].unique(), wizard_with_gmd["GMD_ID"].unique()
```

```
(array([4, 1, 2, 3, 5]), array([3, 5, 4, 1]))
```

```python
# Imports
import pandas as pd
import geopandas as gpd
import plotly.express as px
import plotly.graph_objects as go
import ipywidgets as widgets
import json
import warnings
warnings.filterwarnings("ignore")

# Load Groundwater Management District boundaries and prepare GeoJSON
gmd_gdf = gpd.read_file("Groundwater_Management_Districts_(GMD).geojson")
gmd_gdf = gmd_gdf[gmd_gdf["GMD_ID"].isin([1, 3, 4])]
gmd_geojson = json.loads(gmd_gdf.to_json())

# Approximate Kansas boundary via bounding box (lat/lon)
ks_lon_min, ks_lat_min = -102.051, 36.993
ks_lon_max, ks_lat_max = -94.588, 40.003
ks_lons = [ks_lon_min, ks_lon_max, ks_lon_max, ks_lon_min, ks_lon_min]
ks_lats = [ks_lat_min, ks_lat_min, ks_lat_max, ks_lat_max, ks_lat_min]

# === FILTER DATA FOR GMDs 1, 3, 4 ONLY ===
merged_huc10 = merged_huc10[merged_huc10["GMD_ID"].isin([1, 3, 4])]
wizard_with_gmd = wizard_with_gmd[wizard_with_gmd["GMD_ID"].isin([1, 3, 4])]

# Setup year options
years = list(range(1990, 2023))
year_options = ["All Years"] + years

# Widgets
gmd_selector = widgets.Dropdown(options=[1, 3, 4, 'All'], value=1, description="GMD:")
year_selector = widgets.Dropdown(options=year_options, value=2022, description="Year:")
data_selector = widgets.RadioButtons(options=["Water Used", "Water Depth"], value="Water Used", description="Data:")

# Map update function
def update_map(gmd_selected, year_selected, data_type_selected):
    # Select data for HUC10 and wells
    if gmd_selected == 'All':
        df_huc = merged_huc10.copy()
        df_wiz = wizard_with_gmd.copy()
    else:
        df_huc = merged_huc10[merged_huc10["GMD_ID"] == gmd_selected].copy()
        df_wiz = wizard_with_gmd[wizard_with_gmd["GMD_ID"] == gmd_selected].copy()

    # Build figure
    if data_type_selected == "Water Used":
        df = df_huc
        if year_selected != "All Years":
            year_col = f"AF_{year_selected}"
            df = df[df[year_col].notnull()]
            color_col = year_col
        else:
            af_cols = [c for c in df.columns if c.startswith("AF_")]
            df["avg_water_use"] = df[af_cols].mean(axis=1)
            color_col = "avg_water_use"
        fig = px.choropleth_mapbox(
            df,
            geojson=df.geometry.__geo_interface__,
            locations=df.index,
            color=color_col,
```

```
            mapbox_style="open-street-map",
            color_continuous_scale=[[0, "yellow"], [1, "darkblue"]],
            center={"lat": 38.5, "lon": -98.5},
            zoom=6,
            opacity=0.6,
            hover_name="HU_10_NAME",
            hover_data={
                "HUC_10": True,
                color_col: True,
                "GMD_ID": True
            },
            title=f"{('All GMDs' if gmd_selected=='All' else f'GMD {gmd_selected}')} - Water Used ({year_selected})"
        )
    else:
        df = df_wiz
        if year_selected != "All Years":
            df = df[df["year"] == int(year_selected)]
        fig = px.scatter_mapbox(
            df,
            lat="lat",
            lon="long",
            color="depth",
            mapbox_style="open-street-map",
            color_continuous_scale="RdBu",
            center={"lat": 38.5, "lon": -98.5},
            zoom=6,
            size_max=12,
            hover_name="wellid",
            hover_data={
                "year": True,
                "depth": True,
                "GMD_ID": True
            },
            title=f"{('All GMDs' if gmd_selected=='All' else f'GMD {gmd_selected}')} - Water Depth ({year_selected})"
        )

    # Highlight GMD boundaries when 'All' is selected
    if gmd_selected == 'All':
        fig.update_layout(
            mapbox_layers=[
                {"source": gmd_geojson, "type": "line", "color": "darkblue", "line": {"width": 3}}
            ]
        )

    # Overlay Kansas state boundary
    fig.add_trace(go.Scattermapbox(
        lon=ks_lons,
        lat=ks_lats,
        mode='lines',
        line=dict(color='black', width=3),
        showlegend=False
    ))
```

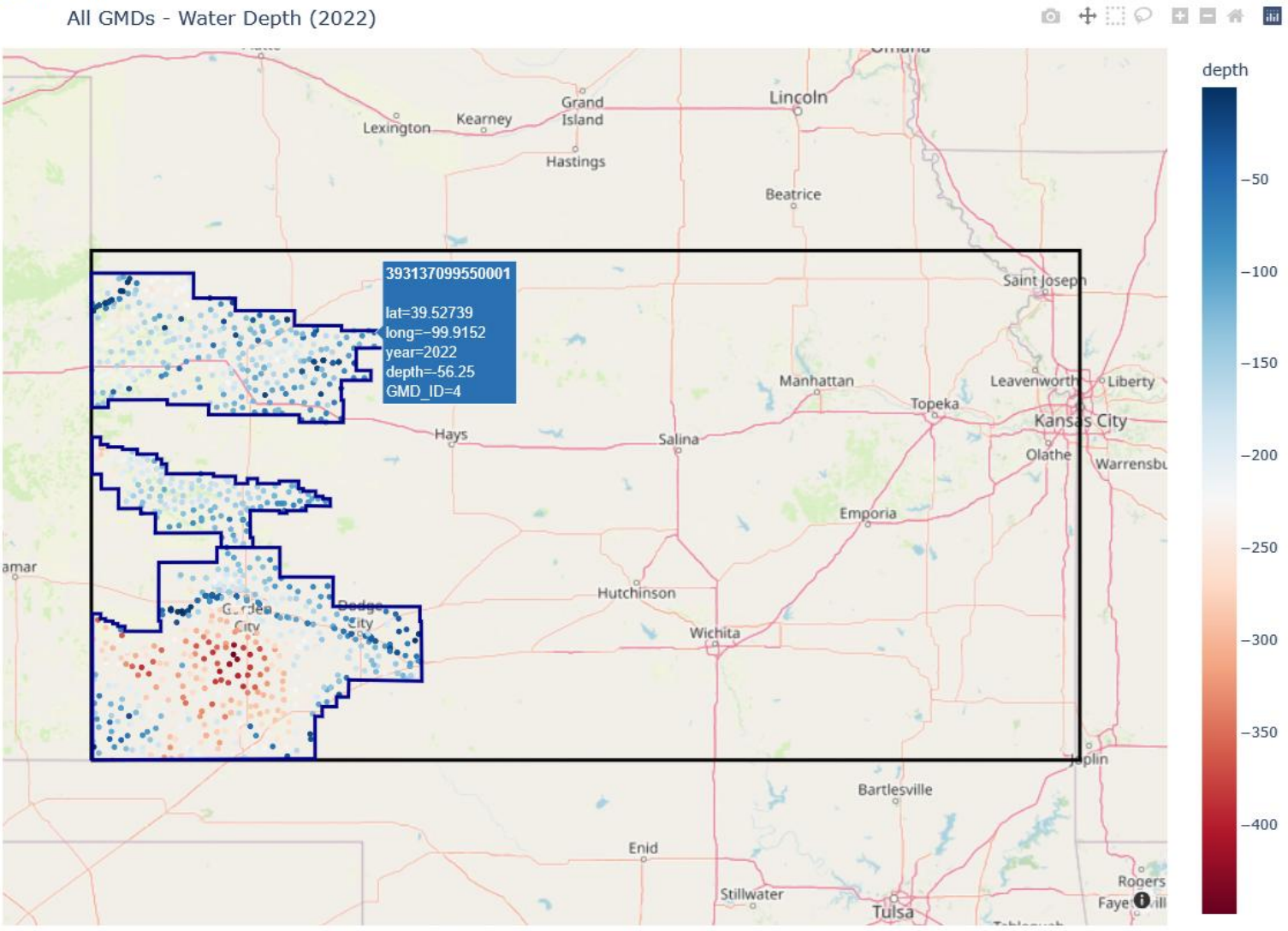
```
    # Final adjustments
    fig.update_layout(
        height=800,
        margin={"r": 0, "t": 40, "l": 0, "b": 0}
    )
    fig.show()

# Launch interactive map
widgets.interact(
    update_map,
    gmd_selected=gmd_selector,
    year_selected=year_selector,
    data_type_selected=data_selector
)
```

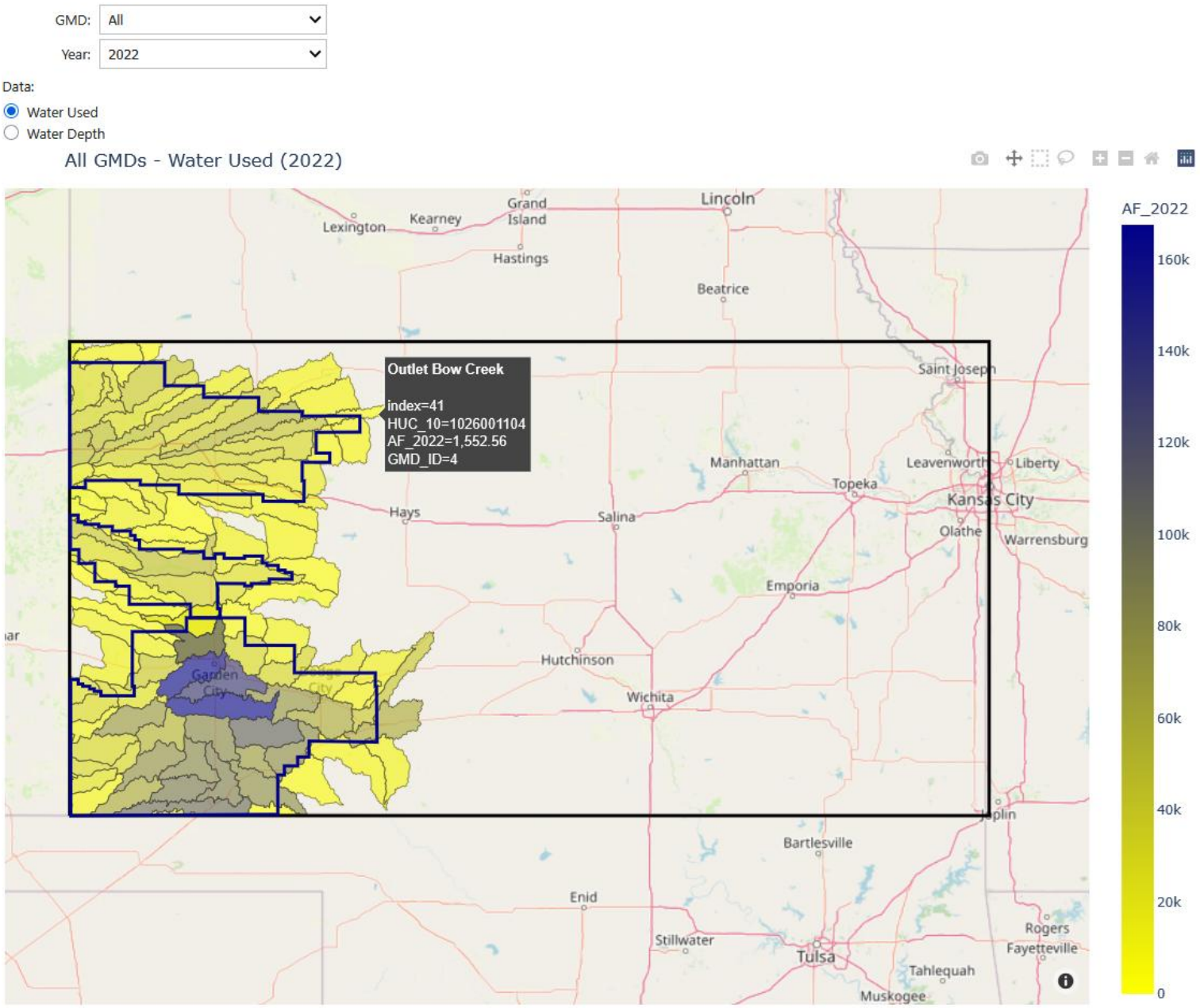
GMD: All

Year: 2022

Data:

- ○ Water Used
- ◉ Water Depth

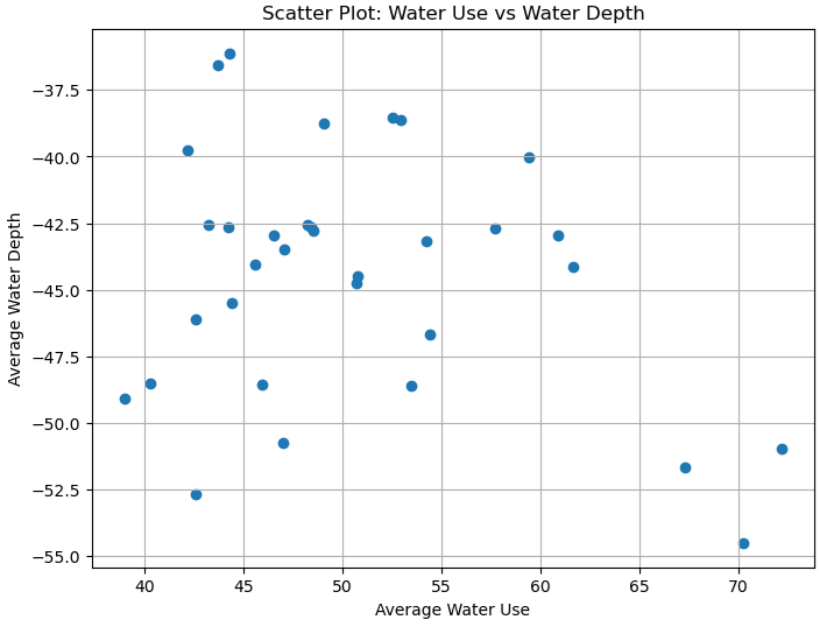




Next, I plan to create scatter plots to explore potential correlations between general water use and well water depth. I will also analyze the relationship between water use in HUC10 regions and well depth, to see if increased usage corresponds with changes in groundwater levels.

```python
import pandas as pd

# Load your county+GMD water-use summary
water_df = pd.read_csv('GMD_Water_Use_avg_by_county_gmd.csv')

# Melt the AF_USED_ columns into long form
years = list(range(1990, 2023))
af_cols = [f"AF_USED_{yr}" for yr in years]

water_long = (
    water_df
    .melt(
        id_vars=['COUNTY_GEOID','COUNTY_NAME','gmd'],
        value_vars=af_cols,
        var_name='year',
        value_name='water_use'
    )
)
water_long['year'] = water_long['year'].str.extract('(\d{4})').astype(int)

# Average water use per year (each county+GMD counts once)
avg_water = water_long.groupby('year')['water_use'].mean()

# Load your well-level depth dataset
wizard = pd.read_csv('GMD Water Use_WIZARD Data.csv')  # replace with actual filename
wizard['year'] = pd.to_datetime(wizard['date'], errors='coerce').dt.year

# Average depth per year (each well counts once)
avg_depth = wizard.groupby('year')['depth'].mean()

# Combine into one DataFrame and drop any years missing one of the metrics
summary = (
    pd.concat([avg_water, avg_depth], axis=1, keys=['avg_water_use','avg_depth'])
    .dropna()
)
```

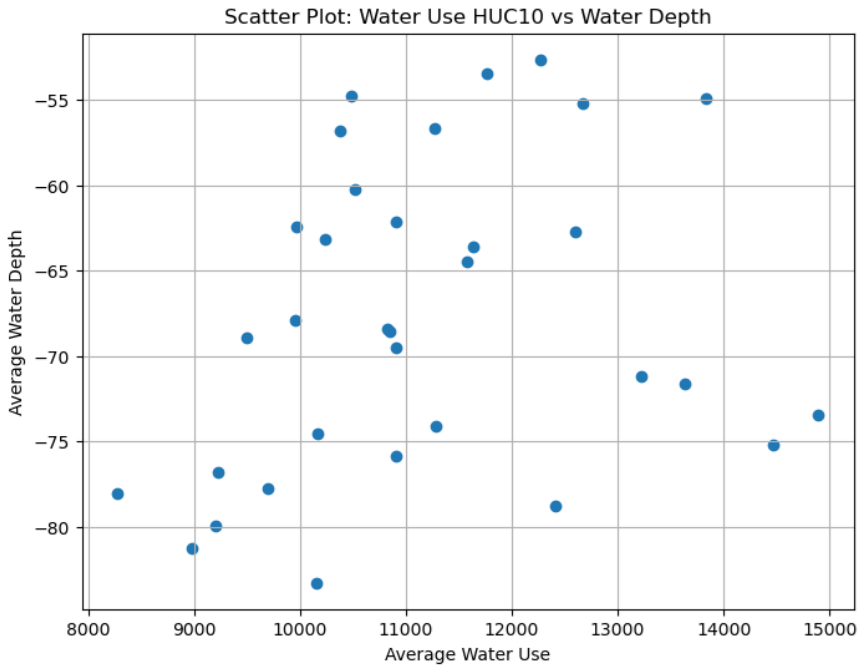
```python
import matplotlib.pyplot as plt

# summary_df should already exist with columns 'avg_water_use' and 'avg_depth'
plt.figure(figsize=(8, 6))
plt.scatter(summary['avg_water_use'], summary['avg_depth'])
plt.xlabel('Average Water Use')
plt.ylabel('Average Water Depth')
plt.title('Scatter Plot: Water Use vs Water Depth')
plt.grid(True)
plt.show()

corr = summary["avg_water_use"].corr(summary["avg_depth"])
print("Correlation:", corr)
```




```
Correlation: -0.2858656407715039
```

```
#comparing Total Average Water use vs Water Depth in Wells
import pandas as pd

# List your years
years = list(range(1990, 2023))

# Compute average water use per year (across all records)
water_use_ave = df_wimas_cols.copy()
water_use_avg = [
    df_wimas_cols[f"AF_{yr}"].mean()
    for yr in years
]

# Compute average depth per year
wizard1 = df_wizard_cols
wizard1["year"] = pd.to_datetime(
    wizard1["date"], errors="coerce"
).dt.year

depth_ave = df_wizard_cols
depth_avg = [
    wizard1.loc[
        wizard1["year"] == yr, "depth"
    ].mean()
    for yr in years
]

# Build your 3-column summary
summary_df2 = pd.DataFrame({
    "year": years,
    "avg_water_use": water_use_avg,
    "avg_depth": depth_avg
})
```

```
import matplotlib.pyplot as plt

# summary_df should already exist with columns 'avg_water_use' and 'avg_depth'
plt.figure(figsize=(8, 6))
plt.scatter(summary_df2['avg_water_use'], summary_df2['avg_depth'])
plt.xlabel('Average Water Use')
plt.ylabel('Average Water Depth')
plt.title('Scatter Plot: Water Use HUC10 vs Water Depth')
plt.grid(True)
plt.show()

corr = summary_df2["avg_water_use"].corr(summary_df2["avg_depth"])
print("Correlation:", corr)
```




```
Correlation: 0.2612600271717973
```

To understand the relationship between water usage and groundwater depth, I implemented a machine learning model using the Random Forest Regressor algorithm. This algorithm is part of supervised learning, which means it learns from labeled data—in this case, it learns how various types of water use relate to recorded well depths. Since we're predicting a continuous value (depth), this is specifically a regression problem.

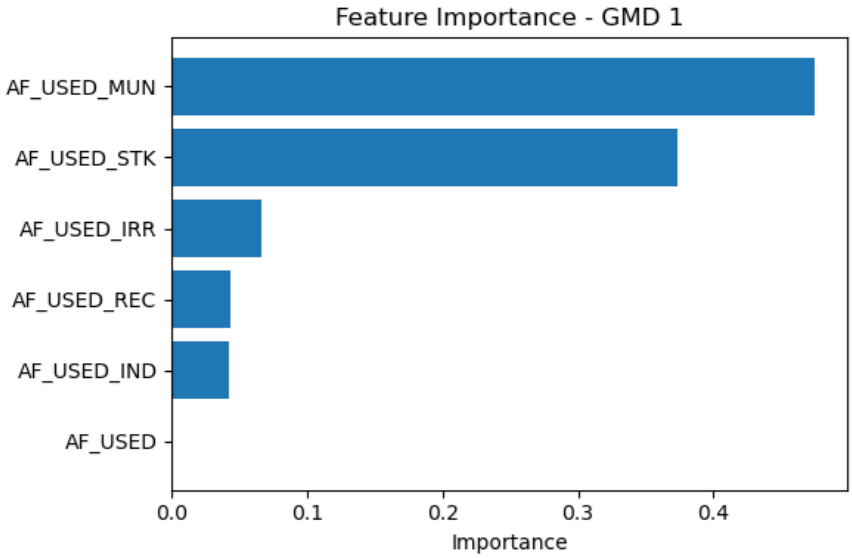
The analysis began by preparing two key datasets. One contained average annual water use by county across six categories; irrigation, municipal, stock, industrial, recreational, and total use. The second dataset (WIZARD) provided well depth readings over time, which I processed to extract the year and then calculated the average depth per county, GMD, and year. To align both datasets, the water use data was reshaped from a wide to a long format, and all categories were merged into a single, clean structure that matched with the well depth data.

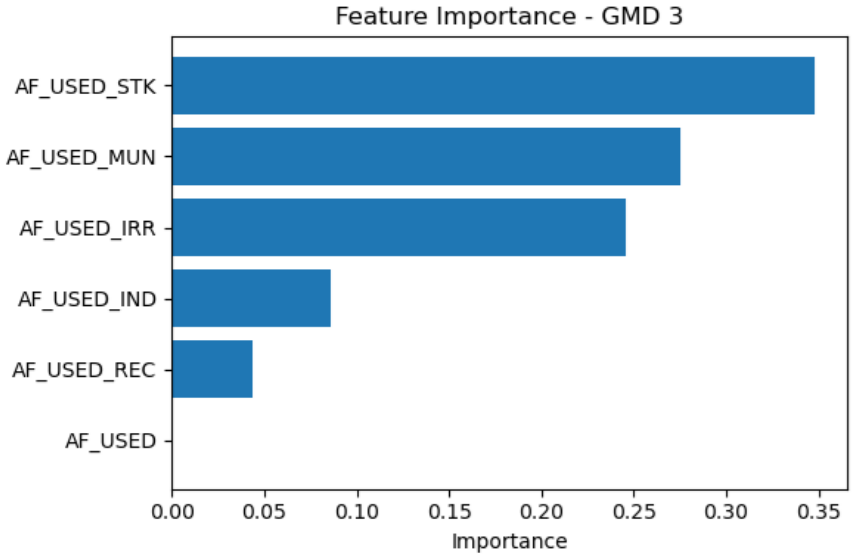
Once the datasets were merged, I trained a separate Random Forest model for each Groundwater Management District (GMD 1, 3, and 4). Each model used the six water use categories to predict groundwater depth. The data was split into training and testing sets to ensure the model could generalize well to new data. After training, I evaluated each model using key metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R-squared (R^2) to measure accuracy.

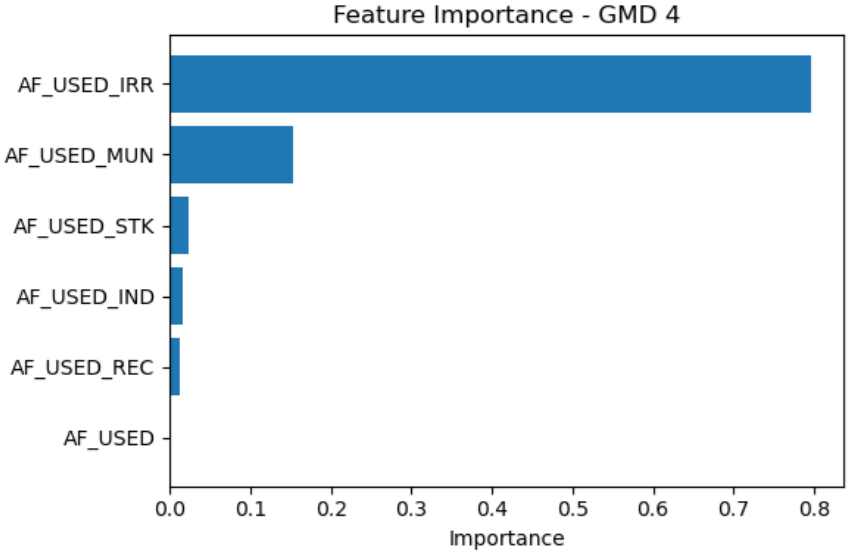
Finally, I visualized the feature importance for each GMD model. These charts show which categories of water use had the most influence on predicting well depth. This step is valuable for identifying which types of usage are most associated with changes in groundwater levels, offering useful insights for water resource management in each region.

```python
import pandas as pd
import matplotlib.pyplot as plt
from functools import reduce
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Load datasets
water_use_df = pd.read_csv('GMD_Water_Use_avg_by_county_gmd.csv')
wizard_df    = df_wizard_cols.copy()

# Preprocess WIZARD: extract year
wizard_df['date'] = pd.to_datetime(wizard_df['date'], errors='coerce')
wizard_df['year'] = wizard_df['date'].dt.year

# Aggregate average depth per county, gmd, year
df_depth = (
    wizard_df
    .groupby(['COUNTY_NAME', 'gmd', 'year'])['depth']
    .mean().reset_index()
)

# Reshape water use (wide → long) for each category
categories = [
    'AF_USED',
    'AF_USED_IRR',
    'AF_USED_MUN',
    'AF_USED_STK',
    'AF_USED_IND',
    'AF_USED_REC'
]
base_cols = ['COUNTY_NAME', 'gmd']
reshaped = []

for cat in categories:
    cols = [c for c in water_use_df.columns if c.startswith(cat)]
    df_cat = water_use_df[base_cols + cols]
    df_long = df_cat.melt(
        id_vars=base_cols,
        value_vars=cols,
        var_name='type_year',
        value_name=cat
    )
    df_long['year'] = df_long['type_year'].str.extract(r'_(\d+)$').astype(int)
    reshaped.append(df_long.drop(columns='type_year'))
```

```
water_long = reduce(
    lambda left, right: pd.merge(left, right, on=base_cols + ['year'], how='inner'),
    reshaped
)

# Merge water use & depth
final_df = pd.merge(
    water_long,
    df_depth,
    on=['COUNTY_NAME', 'gmd', 'year'],
    how='inner'
).dropna()

# Train/Evaluate & Plot for each GMD
results = []
for gmd in [1, 3, 4]:
    df_g = final_df[final_df['gmd'] == gmd]
    X = df_g[categories]
    y = df_g['depth']

    # split
    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )

    # train
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)

    # eval
    y_pred = model.predict(X_test)
    results.append({
        'GMD': f'GMD {gmd}',
        'MAE': mean_absolute_error(y_test, y_pred),
        'RMSE': mean_squared_error(y_test, y_pred, squared=False),
        'R²': r2_score(y_test, y_pred)
    })

    # feature importances
    imp = model.feature_importances_
    idx = imp.argsort()
    plt.figure(figsize=(6,4))
    plt.barh([categories[i] for i in idx], imp[idx])
    plt.title(f'Feature Importance - GMD {gmd}')
    plt.xlabel('Importance')
    plt.tight_layout()
    plt.show()

    # Print data metrics
metrics_df = pd.DataFrame(results)
print(metrics_df)
```





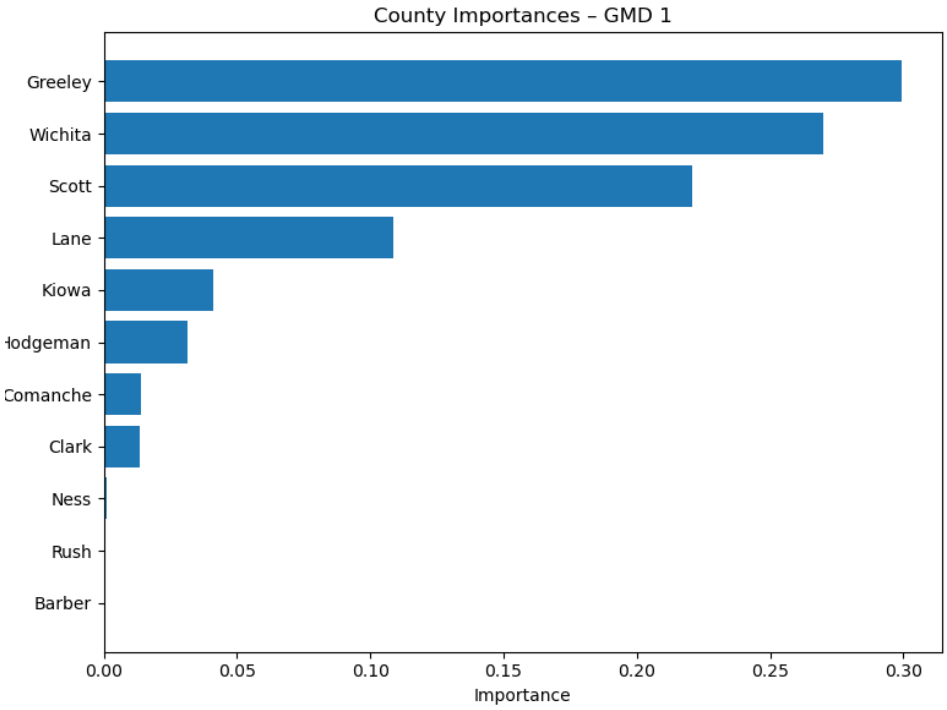


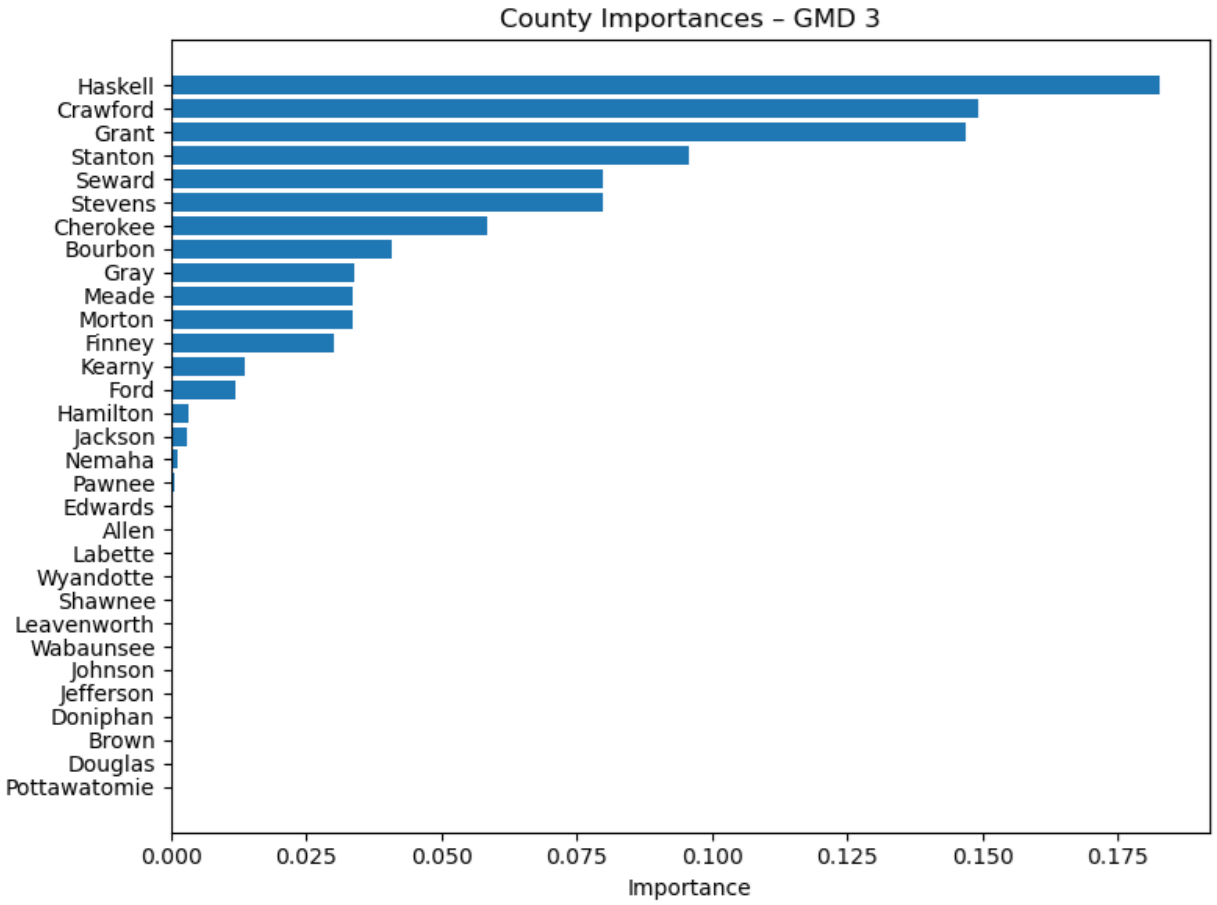
	GMD	MAE	RMSE	R^2
0	GMD 1	0.383647	0.835001	0.999609
1	GMD 3	1.102610	2.999565	0.998970
2	GMD 4	0.370569	0.789675	0.999763

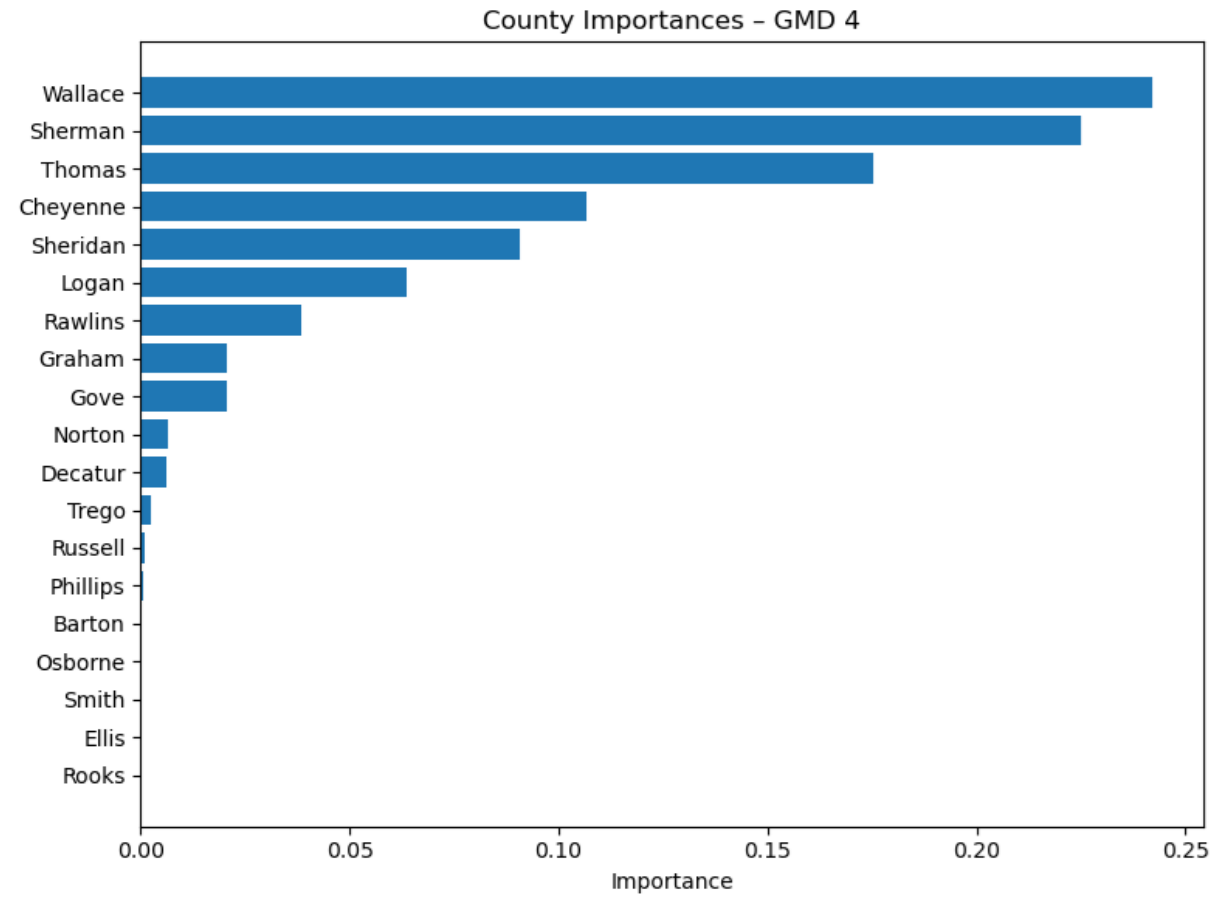
To see how much each county influences groundwater depth, I used the Random Forest regression model with county names as the main input. For each GMD (1, 3, and 4), the model learned from the data and made predictions based only on which county the measurements came from. After testing the model, I measured how well it performed and created charts to show which counties had the biggest impact on predicting water depth over the years.

```
import pandas as pd
import matplotlib.pyplot as plt
from functools import reduce
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Load & preprocess data
water_use_df = pd.read_csv('GMD_Water_Use_avg_by_county_gmd.csv')
wizard_df    = df_wizard_cols.copy()  # assumes df_wizard_cols is already in memory

wizard_df['date'] = pd.to_datetime(wizard_df['date'], errors='coerce')
wizard_df['year'] = wizard_df['date'].dt.year

df_depth = (
    wizard_df
    .groupby(['COUNTY_NAME', 'gmd', 'year'])['depth']
    .mean()
    .reset_index()
)

# Reshape water use from wide→long
categories = [
    'AF_USED', 'AF_USED_IRR', 'AF_USED_MUN',
    'AF_USED_STK', 'AF_USED_IND', 'AF_USED_REC'
]
base_cols = ['COUNTY_NAME', 'gmd']
reshaped = []

for cat in categories:
    cols = [c for c in water_use_df.columns if c.startswith(cat)]
    df_cat = water_use_df[base_cols + cols].copy()
    df_long = df_cat.melt(
        id_vars=base_cols,
        value_vars=cols,
        var_name='type_year',
        value_name=cat
    )
    df_long['year'] = df_long['type_year'].str.extract(r'_(\d+)$').astype(int)
    reshaped.append(df_long.drop(columns='type_year'))

water_long = reduce(
    lambda left, right: pd.merge(left, right, on=base_cols + ['year'], how='inner'),
    reshaped
)
```

```python
# Merge into final_df
final_df = pd.merge(
    water_long,
    df_depth,
    on=['COUNTY_NAME', 'gmd', 'year'],
    how='inner'
).dropna()

# Train & evaluate per GMD with counties as features
results2 = []
for gmd in [1, 3, 4]:
    df_g = final_df[final_df['gmd'] == gmd]
    X = pd.get_dummies(df_g['COUNTY_NAME'])  # one-hot of county names
    y = df_g['depth']

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

    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    results2.append({
        'GMD': f'GMD {gmd}',
        'MAE': mean_absolute_error(y_test, y_pred),
        'RMSE': mean_squared_error(y_test, y_pred, squared=False),
        'R²': r2_score(y_test, y_pred)
    })

    # Plot county importances
    importances = model.feature_importances_
    counties = X.columns
    sorted_idx = importances.argsort()

    plt.figure(figsize=(8, 6))
    plt.barh([counties[i] for i in sorted_idx], importances[sorted_idx])
    plt.title(f'County Importances - GMD {gmd}')
    plt.xlabel('Importance')
    plt.tight_layout()
    plt.show()

# 5) Summary metrics
metrics_df = pd.DataFrame(results2)
print(metrics_df)
```





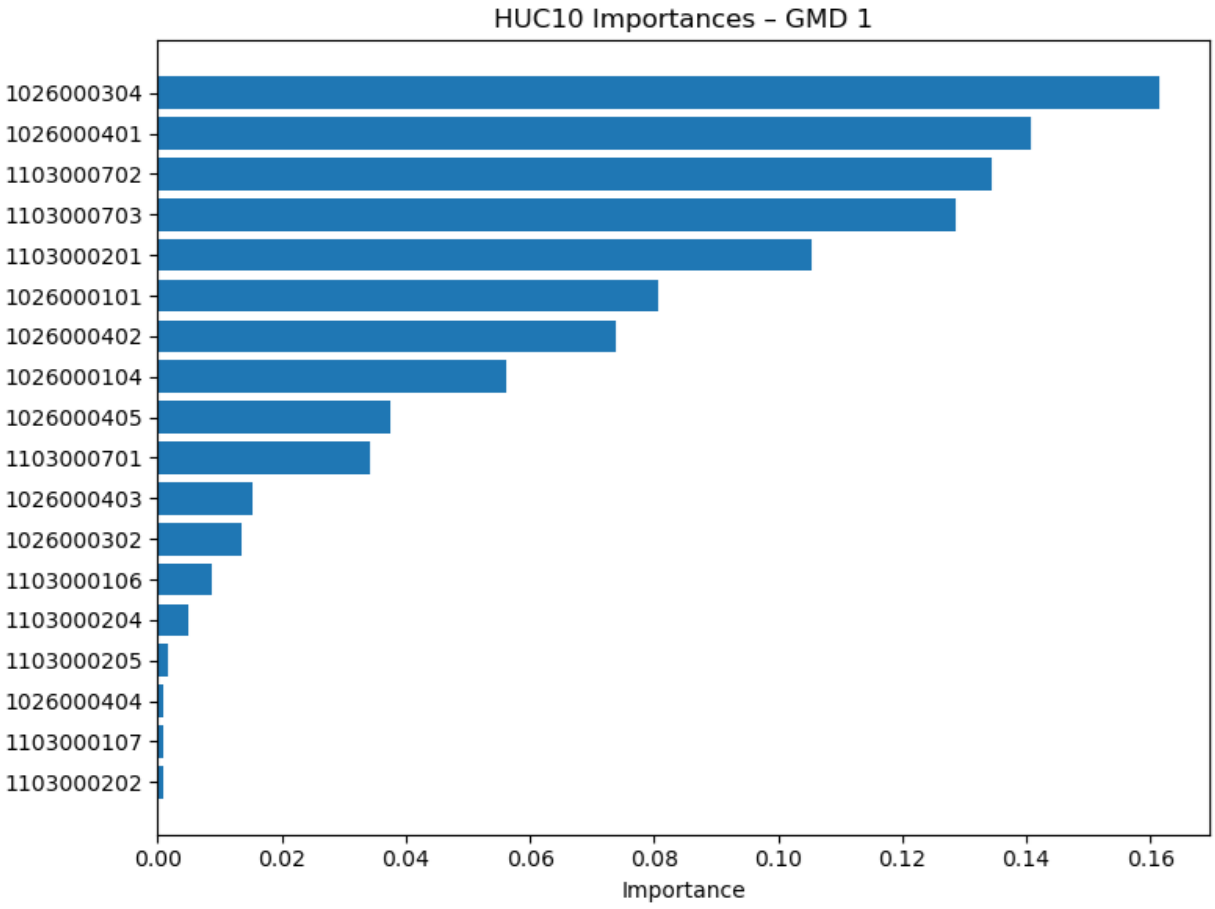


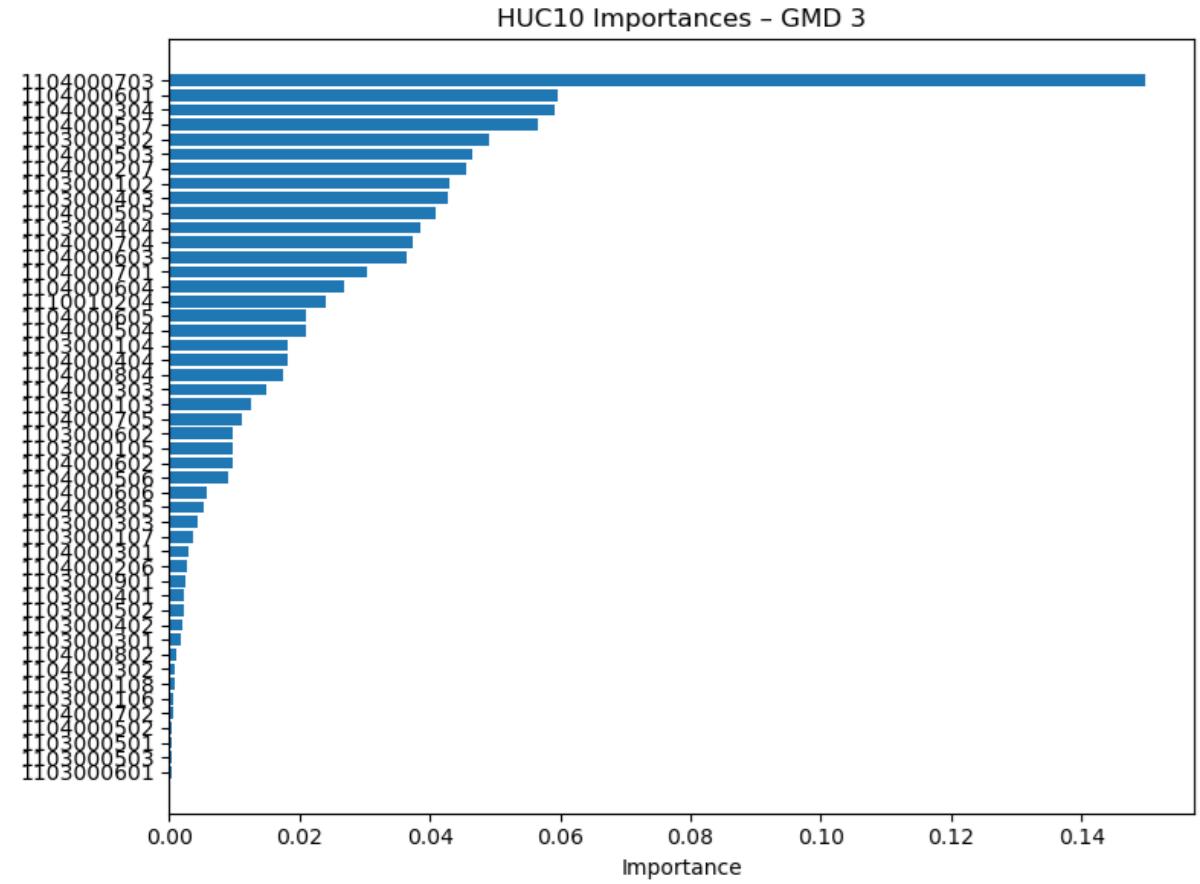
	GMD	MAE	RMSE	R^2
0	GMD 1	5.909887	8.375346	0.960691
1	GMD 3	13.586030	19.977537	0.954329
2	GMD 4	8.244851	12.197399	0.943370

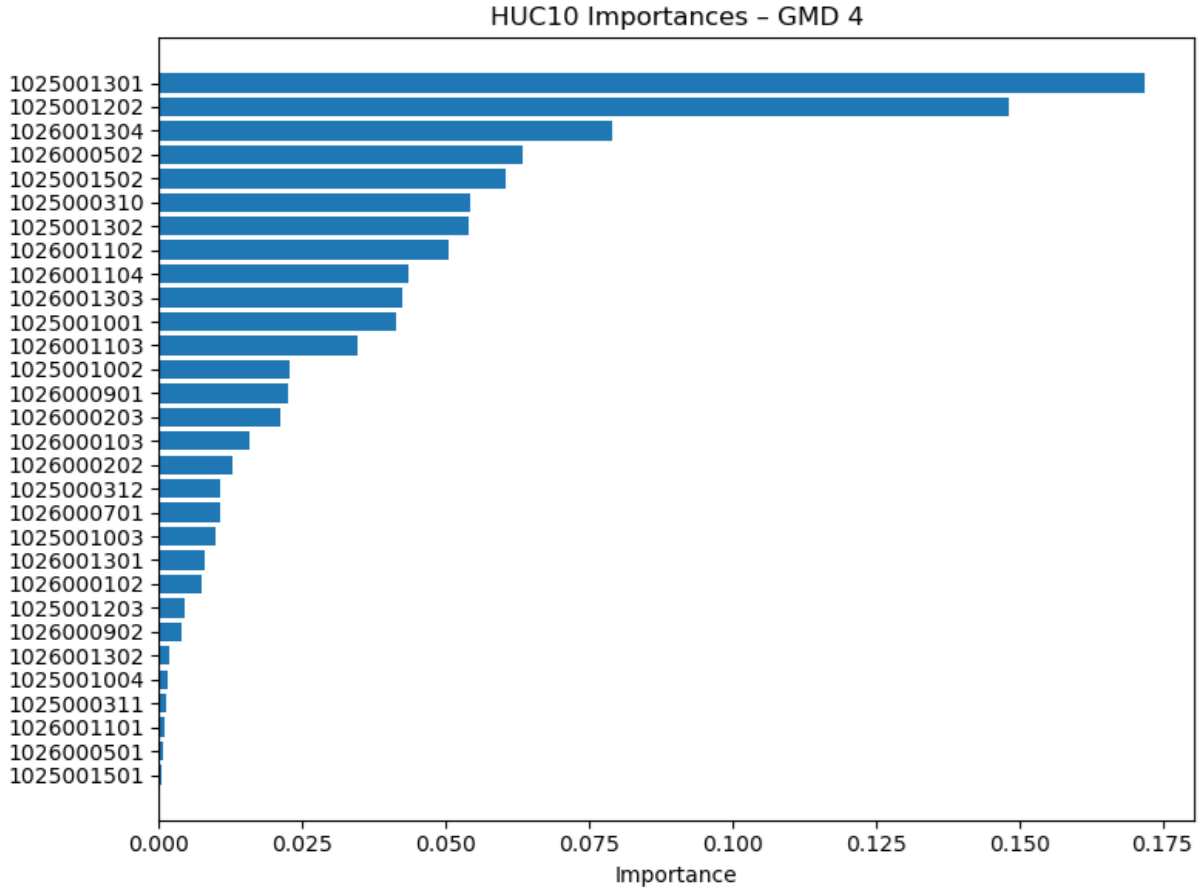
In this part of the analysis, the Random Forest model was used to explore how different HUC10 regions affect groundwater depth across GMDs 1, 3, and 4. Each well location was matched to a HUC10 area, and the average water depth was calculated by region and year. The model then used the HUC10 areas as input to predict water depth for each GMD. After testing the model, performance was measured using MAE, RMSE, and R^2 scores. Finally, charts were created to show which HUC10 areas had the greatest impact on predicting groundwater depth, helping to identify the most influential regions within each district.

```
import pandas as pd
import geopandas as gpd
from shapely.geometry import Point
import matplotlib.pyplot as plt
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

# Assign each well to its HUC10 polygon
wizard_df['geometry'] = wizard_df.apply(lambda r: Point(r['long'], r['lat']), axis=1)
wizard_gdf = gpd.GeoDataFrame(wizard_df, geometry='geometry', crs='EPSG:4326')
wizard_gdf = wizard_gdf.to_crs(merged_huc10.crs)

wizard_with_huc10 = gpd.sjoin(
    wizard_gdf,
    merged_huc10[['HUC_10', 'GMD_ID', 'geometry']],
    how='inner',
    predicate='within'
)
wizard_with_huc10['year'] = pd.to_datetime(wizard_with_huc10['date'], errors='coerce').dt.year

# Compute average depth per HUC_10, GMD_ID, year
df_depth_huc = (
    wizard_with_huc10
    .groupby(['HUC_10', 'GMD_ID', 'year'])['depth']
    .mean()
    .reset_index(name='avg_depth')
)

# Melt your merged_huc10 water-use into long form
af_cols = [c for c in merged_huc10.columns if c.startswith('AF_') and c.split('_')[1].isdigit()]
df_huc_long = merged_huc10.melt(
    id_vars=['HUC_10', 'GMD_ID'],
    value_vars=af_cols,
    var_name='AF_year',
    value_name='water_use'
)
df_huc_long['year'] = df_huc_long['AF_year'].str.extract(r'AF_(\d+)$').astype(int)
df_huc_long = df_huc_long.dropna(subset=['water_use'])

# Merge features & target into modeling DataFrame
df_model = pd.merge(
    df_huc_long[['HUC_10', 'GMD_ID', 'year', 'water_use']],
    df_depth_huc,
    on=['HUC_10', 'GMD_ID', 'year'],
    how='inner'
).dropna()
```

```python
# Train & evaluate one Random Forest per GMD, plotting HUC10 importances
results = []
for gmd in [1, 3, 4]:
    df_g = df_model[df_model['GMD_ID'] == gmd]

    # One-hot encode HUC10 codes as features
    X = pd.get_dummies(df_g['HUC_10'], prefix='', prefix_sep='')
    y = df_g['avg_depth']

    X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.2, random_state=42
    )
    model = RandomForestRegressor(n_estimators=100, random_state=42)
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)

    mae  = mean_absolute_error(y_test, y_pred)
    rmse = mean_squared_error(y_test, y_pred, squared=False)
    r2   = r2_score(y_test, y_pred)
    results.append({'GMD': f'GMD {gmd}', 'MAE': mae, 'RMSE': rmse, 'R²': r2})

    # Plot feature importances (HUC10 blocks)
    importances = model.feature_importances_
    features = X.columns
    idx = importances.argsort()

    plt.figure(figsize=(8, 6))
    plt.barh([features[i] for i in idx], importances[idx])
    plt.title(f'HUC10 Importances - GMD {gmd}')
    plt.xlabel('Importance')
    plt.tight_layout()
    plt.show()

# summary metrics
metrics_df = pd.DataFrame(results)
print(metrics_df)
```







	GMD	MAE	RMSE	R²
0	GMD 1	10.011557	14.051812	0.915419
1	GMD 3	16.561532	22.103960	0.915160
2	GMD 4	10.106197	13.683509	0.910759

Model Summary and Key Insights Across GMDs 1, 3, and 4

Throughout this project, I used the Random Forest Regressor to understand the factors that influence groundwater depth across GMDs 1, 3, and 4. I tested three different approaches to see how well water depth could be predicted using different kinds of input: water use categories, county names, and HUC10 regions. Each model offered unique insights and helped highlight what matters most in understanding aquifer behavior across different areas of Kansas.

In the first model, I used six types of water use (irrigation, municipal, stock, industrial, recreational, and total) to predict water depth. This category-based model revealed different patterns in each GMD. In GMD 1, the most important feature was municipal water use, followed by stock use. In GMD 3, stock water use ranked first, with municipal use again in second place. Meanwhile, in GMD 4, irrigation use had the strongest influence, followed by municipal use. These results show that although irrigation is often seen as the main driver of groundwater use, other types of water use can be more influential in certain districts. The feature importance graphs made it easy to see which categories had the biggest effect in each region, helping guide future policy toward the most impactful usage types.

In the second model, I shifted focus to the counties themselves to see how much location alone could explain changes in groundwater depth. The model learned from county names and revealed that certain counties had a stronger influence on depth trends than others. In GMD 1, Greeley County had the highest impact, followed by Wichita County. In GMD 3, Haskell County was the most important, with Crawford County in second place. For GMD 4, Wallace County had the greatest influence on the model, followed by Sherman County. The bar charts of feature importance clearly highlighted which counties contributed the most to the model's predictions, offering a more localized view of which areas may require closer attention for sustainable water use.

Finally, the third model used HUC10 watershed regions as input features. By assigning each well to a HUC10 area, the model could evaluate water depth patterns based on geographical water flow regions rather than political boundaries. In GMD 1, the most important HUC10 region was 1026000304, followed by 1026000401. For GMD 3, 1104000703 ranked highest, followed by 1104000601. In GMD 4, the most influential regions were 1025001301 and 1025001202. The graphs visualizing the importance of each HUC10 region made it easier to pinpoint the most sensitive or heavily impacted areas, which could be essential for future groundwater planning.

Together, these three models provided a well-rounded understanding of groundwater dynamics in western Kansas. By analyzing the issue from three different angles — water usage behavior, county influence, and hydrological regions — I was able to generate meaningful insights to support decisions about sustainable water management at both local and regional levels.