Saguache_County_Water_Study

June 20, 2023

1 Saguache County Water Study

Cammie Newmyer

The Saguache County Water Study is a data project that combines well completion reports from the Colorado Division of Water Resources well permit site with input from a master water contractor. The data has been carefully prepared, cleaned, and engineered to create a high-quality dataset suitable for training a deep learning model. This study aims to provide valuable insights into water resources in Saguache County, Colorado, and contribute to informed decision-making and sustainable water management practices. https://dwr.state.co.us/Tools/WellPermits

The dataset used in the Saguache County Water Study contains several key features that have been engineered and prepared for analysis and modeling. These features provide important information related to well completion reports and water resources in Saguache County.

Feature Descriptions: Township_N: This feature represents the township location of the well in Saguache County, providing a geographical reference point. Range_E: The range feature indicates the range location of the well within Saguache County, helping to narrow down the well's specific area. Section: This feature denotes the section number where the well is located within the township and range, providing more precise spatial information. Well_Depth: This feature specifies the depth of the well, indicating how deep the well reaches into the ground to access the water source. Elevation_Of_Clay_Layer: If reaching the smectite clay layer is likely, then the depth to the aquifer is calculated using the difference between the elevation of the location of the well and the depth at which the smectite clay layer is reached. Smectite_Clay_Layer: This is the target feature for the model. A value of 1 indicates that the smectite clay layer is likely at this location. A value of 0 indicates it is not likely. Depth_to_Aquifer: This feature is given by well completion reports identifying blue clay or alternatively when the data is not available expert opinion on the depth of the smectite clay layer in that area. Elevation: The elevation feature represents the elevation of the surface of the well, which can be crucial in understanding the vertical positioning of the water source.

2 Data Preparation

Import packages for analysis and visualization.

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import tensorflow as tf
```

```
import keras
import numpy as np
import pickle

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix
from keras.models import Sequential
from keras.layers import Dense, Flatten
from keras.callbacks import EarlyStopping

import warnings

# Suppress warnings
warnings.filterwarnings("ignore")

#from google.colab import drive
#drive.mount('/drive/')
```

[2]:
```
from PIL import Image, ImageDraw, ImageFont
from IPython.display import Image as DisplayImage, display

# Open the image
image_path = 'C:/Users/newmy/Desktop/WaterProj/
 ↪iNorthern_End_of_San_Luis_Valley,_Colorado_(7235128470).jpg'
image = Image.open(image_path)

# Create an ImageDraw object
draw = ImageDraw.Draw(image)

# Define the font and font size
font = ImageFont.truetype("arial.ttf", size=100)

# Define the text to be annotated
text = "Northern End of the San Luis Valley - Saguache County, Colorado"

# Calculate the position to place the text
text_width, text_height = draw.textsize(text, font=font)
x = (image.width - text_width) // 2
y = image.height - text_height - 50

# Set the color of the text
text_color = (255, 255, 255)  # White color

# Annotate the image with the text
draw.text((x, y), text, fill=text_color, font=font)
```

```
# Save the annotated image
annotated_image_path = 'C:/Users/newmy/Desktop/WaterProj/annotated_image.jpg'
image.save(annotated_image_path)

# Display the annotated image
display(DisplayImage(filename=annotated_image_path))
```

3 Exploratory Data Analysis

[3]:
```
# Read the data and verify input
data = pd.read_csv('C:/Users/newmy/Desktop/WaterProj/dWaterProjDataB.csv')
data
```

[3]:

	Township_N	Range_E	Section	Well_Depth	Elevation_Of_Clay_Layer	\
0	42	7	1	0	7700	
1	42	7	2	0	7700	
2	42	7	3	0	7700	
3	42	7	4	0	7700	
4	42	7	5	0	7700	
..	…	…	…	…	…	

```
981          69        0        0         991                    7270
982          69        0        0         400                    7270
983          69        0        0         620                    7270
984          69        0        0        1000                    7270
985          69        0        0         640                    7270

     Smectite_Clay_Layer  Depth_to_Aquifer  Elevation
0                      0                 0       7700
1                      0                 0       7700
2                      0                 0       7700
3                      0                 0       7700
4                      0                 0       7700
..                   …                 …        …
981                    1               300       7570
982                    1               300       7570
983                    1               300       7570
984                    1               300       7570
985                    1               300       7570

[986 rows x 8 columns]
```

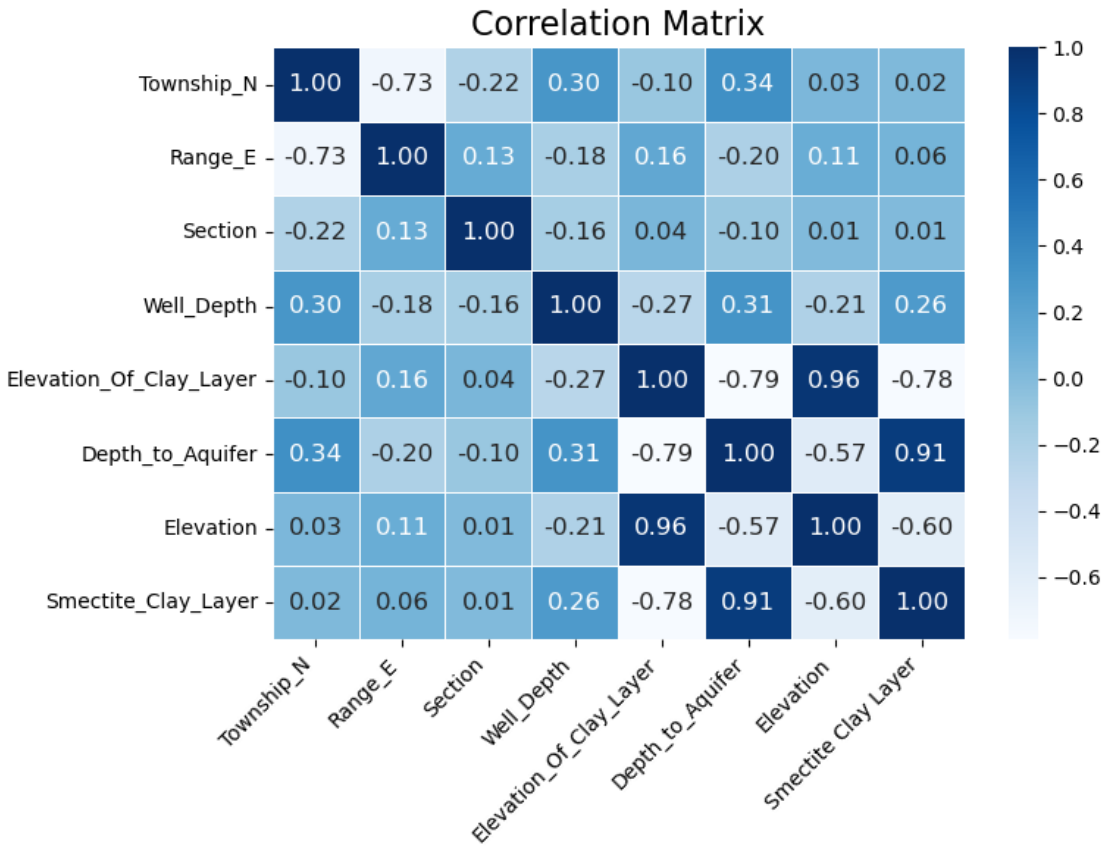
[4]:
```
# Examine the data
data.info()
column_unique_counts = data.nunique()
print(column_unique_counts)
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 986 entries, 0 to 985
Data columns (total 8 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   Township_N               986 non-null    int64
 1   Range_E                  986 non-null    int64
 2   Section                  986 non-null    int64
 3   Well_Depth               986 non-null    int64
 4   Elevation_Of_Clay_Layer  986 non-null    int64
 5   Smectite_Clay_Layer      986 non-null    int64
 6   Depth_to_Aquifer         986 non-null    int64
 7   Elevation                986 non-null    int64
dtypes: int64(8)
memory usage: 61.8 KB
Township_N                   5
Range_E                      6
Section                     37
Well_Depth                 183
Elevation_Of_Clay_Layer    119
Smectite_Clay_Layer          2
Depth_to_Aquifer            54
```

```
Elevation                    46
dtype: int64
```

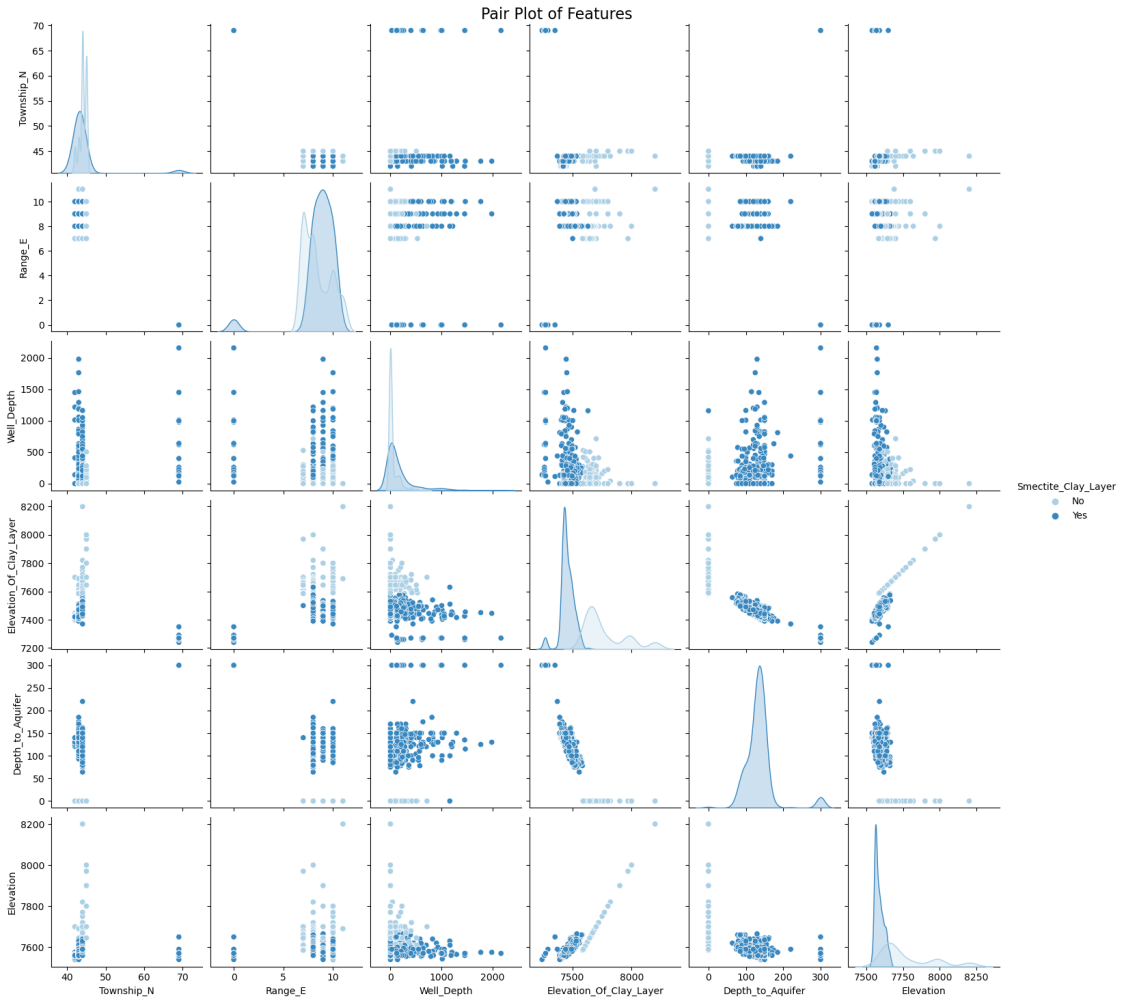
4 Exploratory Data Visualization

[5]:
```
# Define the desired feature order
feature_order = ['Township_N', 'Range_E', 'Section', 'Well_Depth',␣
 ↪'Elevation_Of_Clay_Layer', 'Depth_to_Aquifer', 'Elevation',␣
 ↪'Smectite_Clay_Layer']

# Set up the figure and axes
fig, ax = plt.subplots(figsize=(8, 6))

# Create a heatmap of the correlation matrix
correlation_matrix = data.corr()

# Reorder the rows and columns of the correlation matrix
correlation_matrix = correlation_matrix.reindex(index=feature_order,␣
 ↪columns=feature_order)

# Customize the heatmap
heatmap = sns.heatmap(correlation_matrix, annot=True, cmap='Blues', fmt=".2f",␣
 ↪linewidths=0.5, annot_kws={"fontsize": 12}, ax=ax)

# Rotate and align the horizontal axis labels
heatmap.set_xticklabels(heatmap.get_xticklabels(), rotation=45, ha='right')

# Modify the x-axis tick labels
x_tick_labels = heatmap.get_xticklabels()
x_tick_labels[-1] = 'Smectite Clay Layer'
heatmap.set_xticklabels(x_tick_labels)

# Rotate and align the vertical axis labels
heatmap.set_yticklabels(heatmap.get_yticklabels(), rotation=0, ha='right')

# Set the plot title and labels
ax.set_title('Correlation Matrix', fontsize=16)

# Show the plot
plt.tight_layout()
plt.show()
```

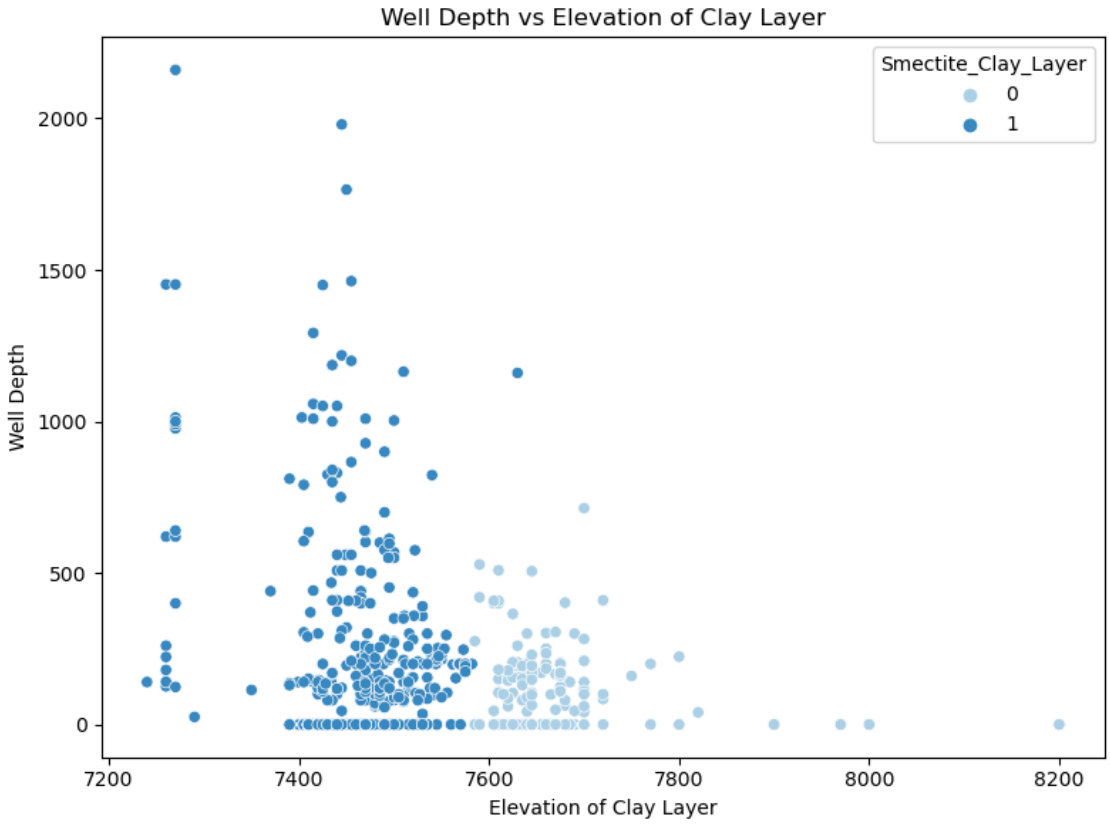


[6]:
```
# Select the columns for the pair plot
pairplot_columns = ['Township_N', 'Range_E', 'Well_Depth',␣
 ↪'Elevation_Of_Clay_Layer', 'Depth_to_Aquifer', 'Elevation',␣
 ↪'Smectite_Clay_Layer']

# Subset the data with the selected columns
pairplot_data = data[pairplot_columns]

# Set the hue to 'Smectite_Clay_Layer' for color differentiation
pairplot_data['Smectite_Clay_Layer'] = pairplot_data['Smectite_Clay_Layer'].
 ↪map({0: 'No', 1: 'Yes'})

# Create the pair plot with labeled axes and blue hue
pairplot = sns.pairplot(pairplot_data, hue='Smectite_Clay_Layer',␣
 ↪palette='Blues')

# Add a title to the pair plot
pairplot.fig.suptitle('Pair Plot of Features', fontsize=16)
```

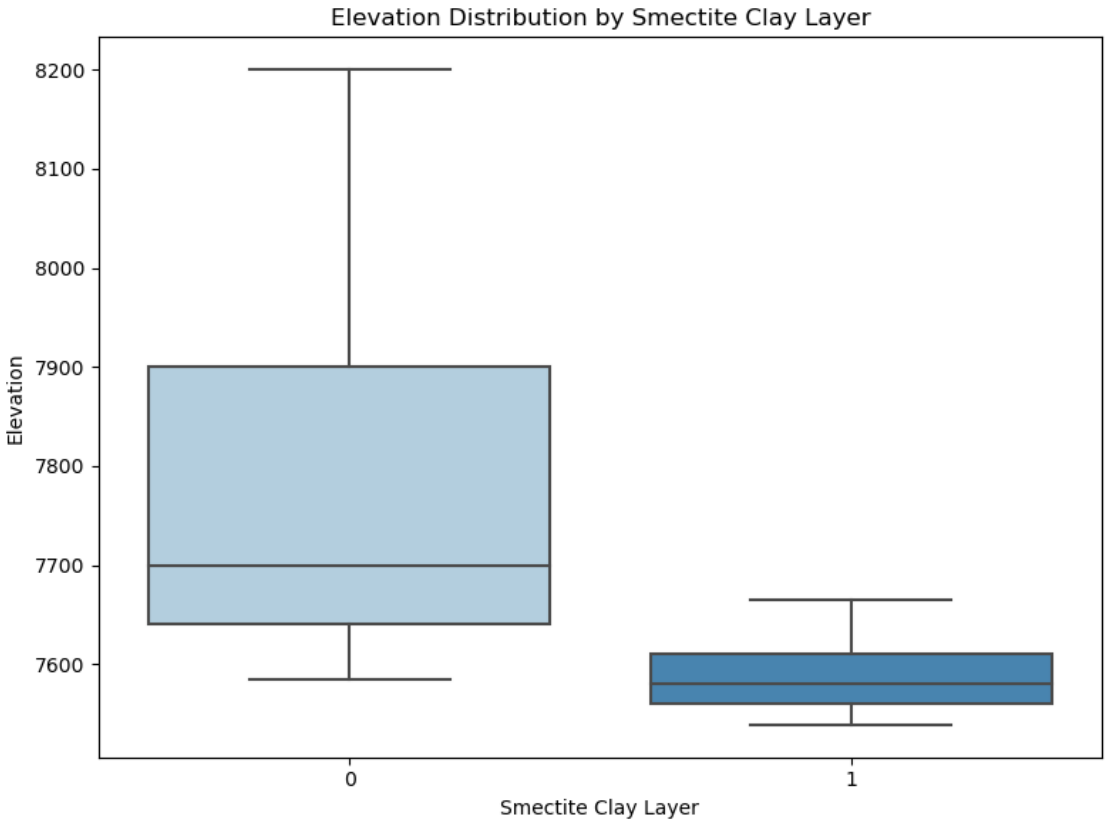
```
# Adjust the spacing between subplots to avoid overlap
pairplot.tight_layout()

# Display the plot
plt.show()
```



[7]:
```
# Scatter Plot
plt.figure(figsize=(8, 6))
sns.scatterplot(x='Elevation_Of_Clay_Layer', y='Well_Depth',␣
 ↪hue='Smectite_Clay_Layer', data=data, palette='Blues')
plt.xlabel('Elevation of Clay Layer')
plt.ylabel('Well Depth')
plt.title('Well Depth vs Elevation of Clay Layer')
plt.tight_layout()
plt.show()
```

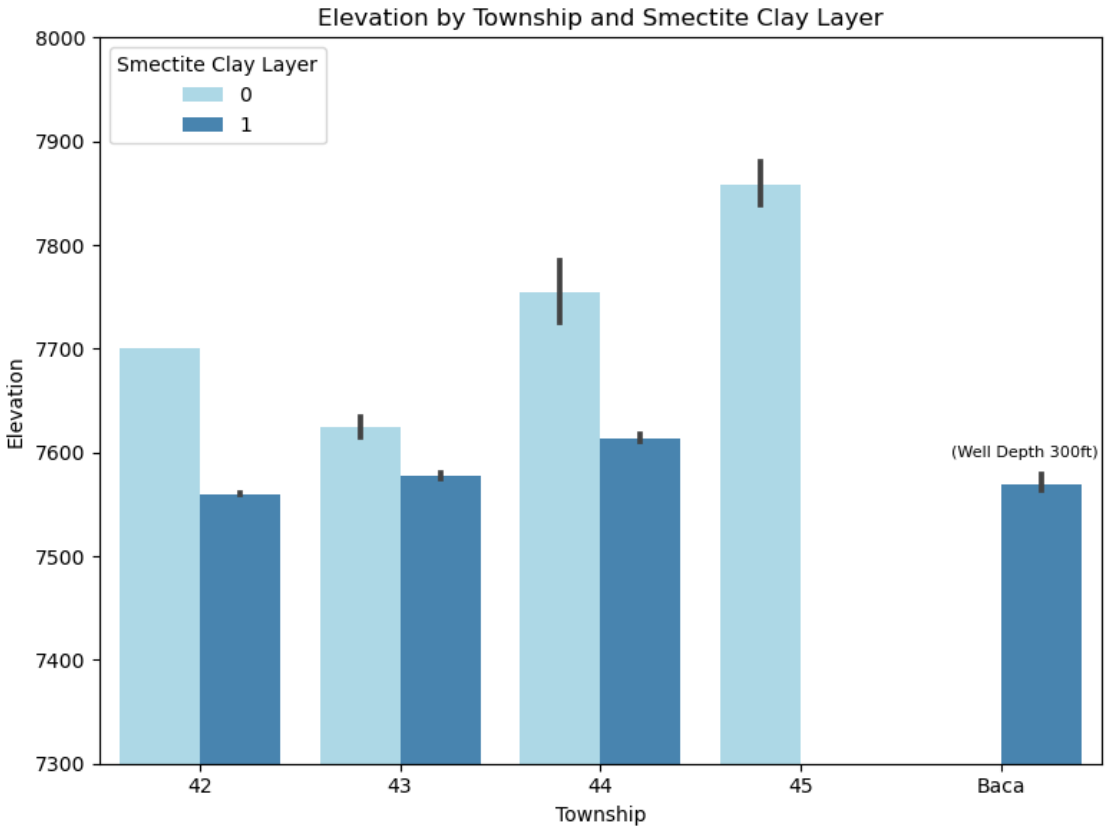


[8]:
```
# Box Plot
plt.figure(figsize=(8, 6))
sns.boxplot(x='Smectite_Clay_Layer', y='Elevation', data=data.drop('Section',␣
 ↪axis=1), palette='Blues')
plt.xlabel('Smectite Clay Layer')
plt.ylabel('Elevation')
plt.title('Elevation Distribution by Smectite Clay Layer')
plt.tight_layout()
plt.show()
```



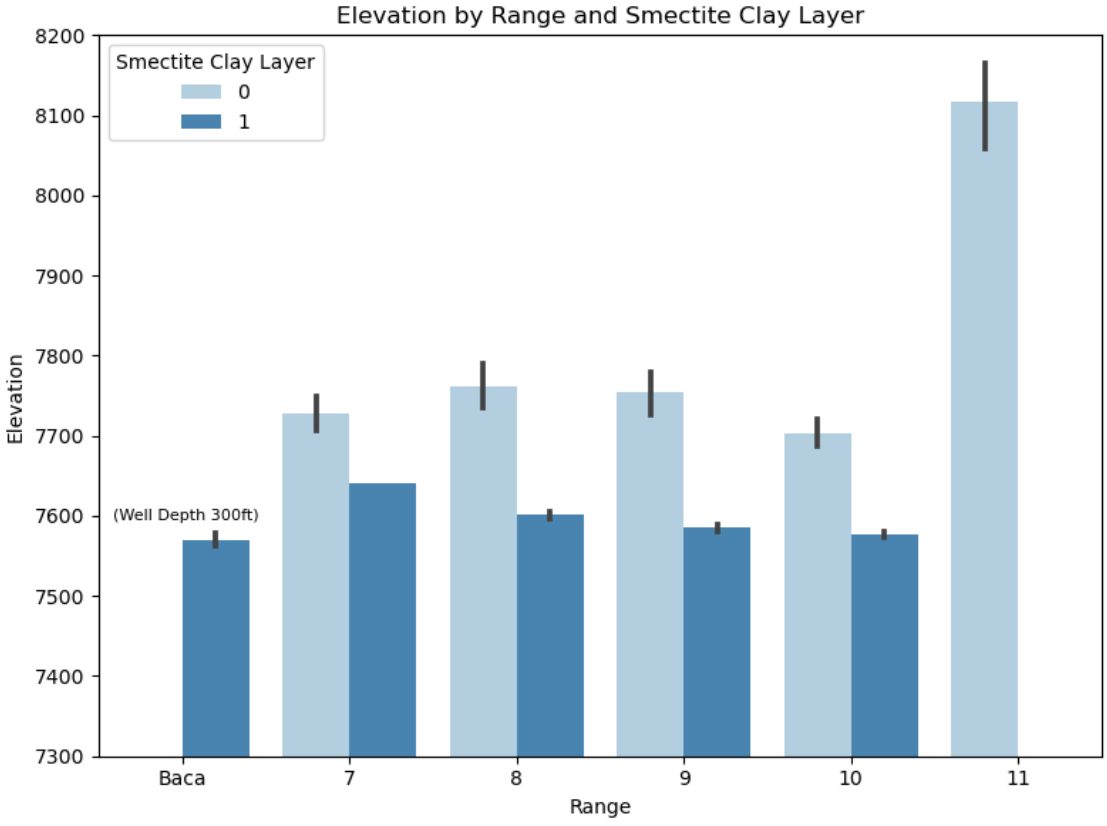
[9]:
```
# Bar Chart
plt.figure(figsize=(8, 6))
ax = sns.barplot(x='Township_N', y='Elevation', hue='Smectite_Clay_Layer',␣
 ↪data=data.drop('Section', axis=1), palette='Blues')
plt.xlabel('Township')
plt.ylabel('Elevation')
plt.title('Elevation by Township and Smectite Clay Layer')
plt.ylim(7300, 8000)  # Set the y-axis limits

# Modify the tick labels on the horizontal axis
xtick_labels = ax.get_xticklabels()
xtick_labels = ['Baca' if label.get_text() == '69' else label.get_text() for␣
 ↪label in xtick_labels]
ax.set_xticklabels(xtick_labels)

# Customize the colors of the bars
bar_colors = ['darkblue' if label == 'Baca' else 'lightblue' for label in␣
 ↪xtick_labels]
for patch, color in zip(ax.patches, bar_colors):
    patch.set_facecolor(color)
```

```
# Annotate the 'Baca' bar with 'Depth 300''
baca_index = xtick_labels.index('Baca')
baca_bar = ax.patches[baca_index]
x = baca_bar.get_x() + baca_bar.get_width()
y = 7600
ax.annotate('(Well Depth 300ft)', xy=(x, y), xytext=(-25, 0),␣
 ↪textcoords='offset points', ha='left', va='center', fontsize=8)

# Move the legend to the upper left corner and add title
plt.legend(title='Smectite Clay Layer', loc='upper left')

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

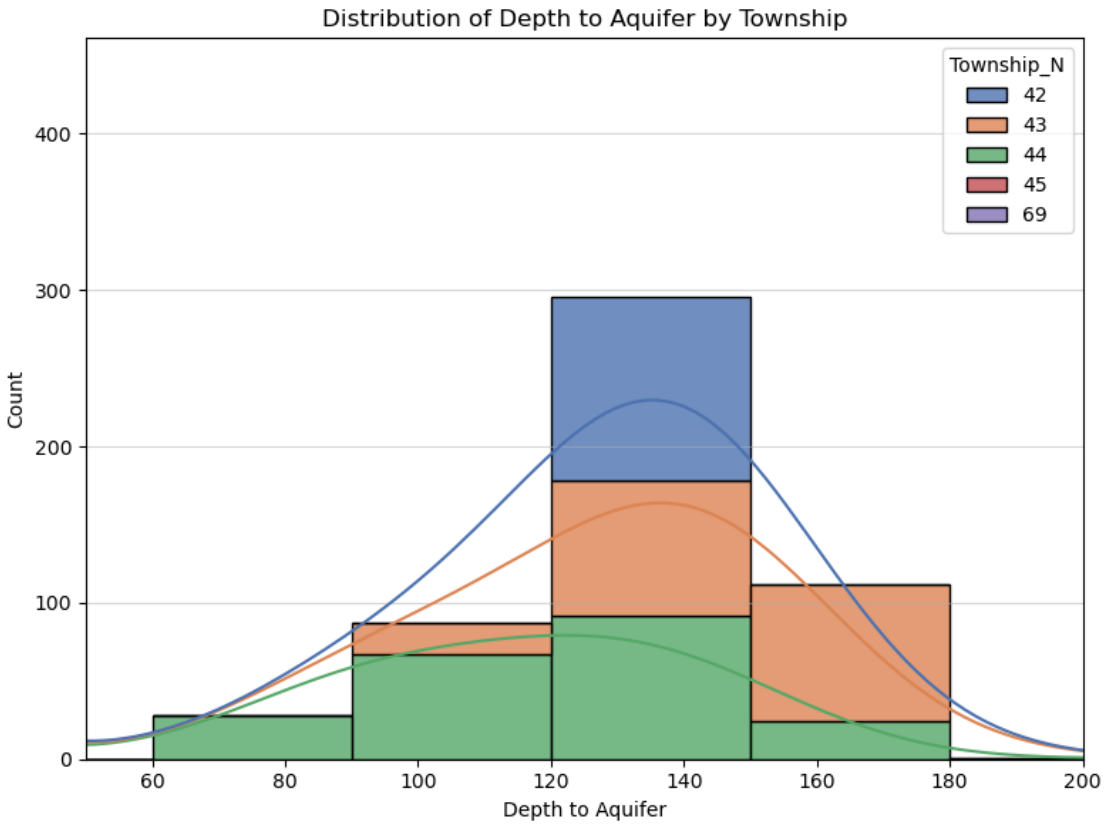


[10]:
```
# Bar Chart
plt.figure(figsize=(8, 6))
ax = sns.barplot(x='Range_E', y='Elevation', hue='Smectite_Clay_Layer',␣
 ↪data=data.drop('Section', axis=1), palette='Blues')
plt.xlabel('Range')
```

```
plt.ylabel('Elevation')
plt.title('Elevation by Range and Smectite Clay Layer')
plt.ylim(7300, 8200)  # Set the y-axis limits

# Modify the tick labels on the horizontal axis
xtick_labels = ax.get_xticklabels()
xtick_labels = ['Baca' if label.get_text() == '0' else label.get_text() for␣
 ↪label in xtick_labels]
ax.set_xticklabels(xtick_labels)

# Annotate the 'Baca' bar with 'Depth 300ft'
baca_index = xtick_labels.index('Baca')
baca_bar = ax.patches[baca_index]
x = baca_bar.get_x() + baca_bar.get_width()
y = 7600
ax.annotate('(Well Depth 300ft)', xy=(x, y), xytext=(-35, 0),␣
 ↪textcoords='offset points', ha='left', va='center', fontsize=8)

# Move the legend to the upper left corner and add title
plt.legend(title='Smectite Clay Layer', loc='upper left')

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

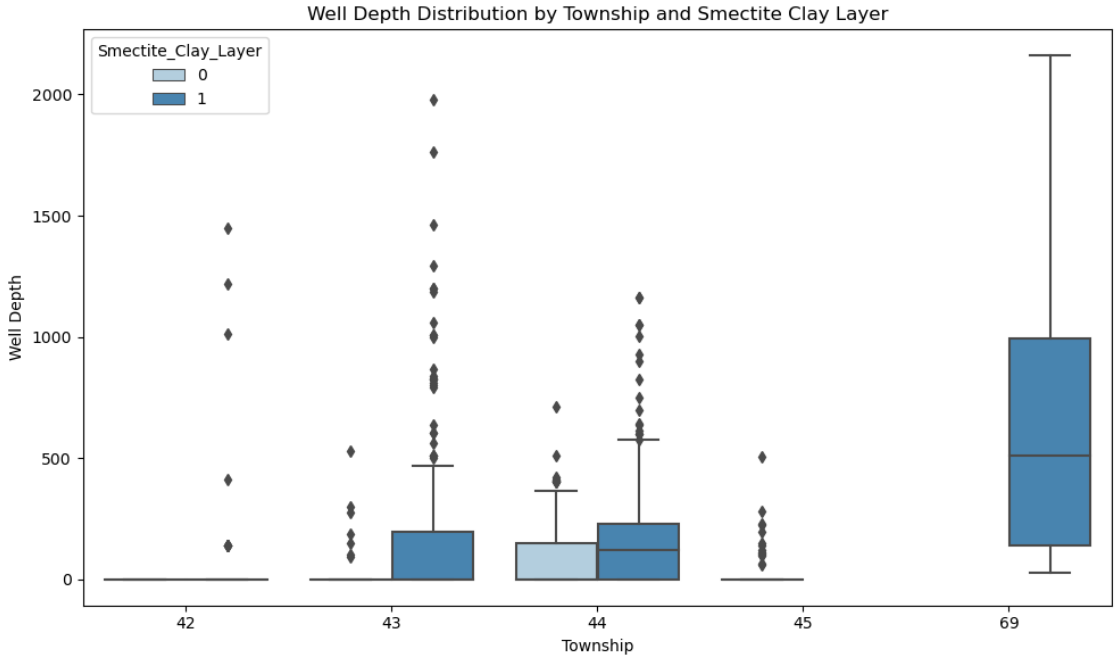


[11]:
```
# Histogram
plt.figure(figsize=(8, 6))
sns.histplot(data, x='Depth_to_Aquifer', bins=10, kde=True, hue='Township_N',␣
 ↪palette='deep', multiple='stack', alpha=0.8, edgecolor='black')
plt.xlabel('Depth to Aquifer')
plt.ylabel('Count')
plt.title('Distribution of Depth to Aquifer by Township')
plt.xlim(50, 200)  # Set the horizontal axis limits
plt.grid(axis='y', alpha=0.5)

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

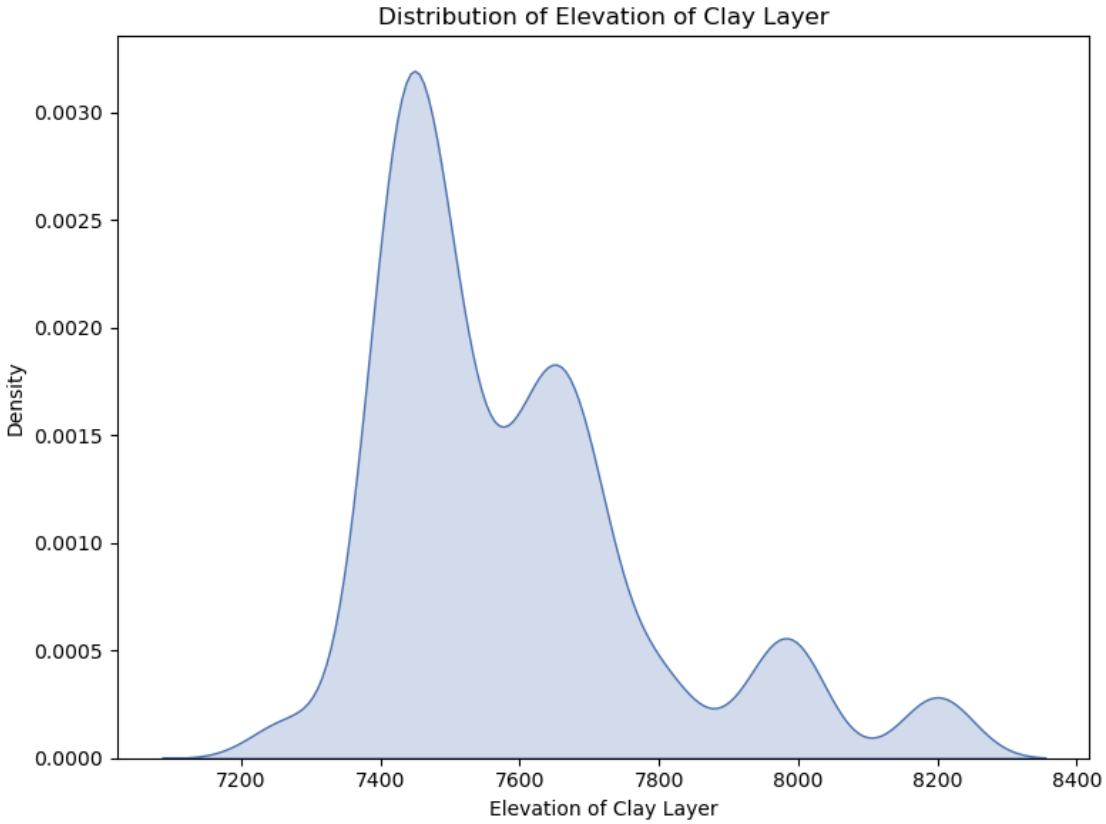


[12]:
```
plt.figure(figsize=(10, 6))
sns.boxplot(x='Township_N', y='Well_Depth', hue='Smectite_Clay_Layer',␣
 ↪data=data, palette='Blues')
plt.xlabel('Township')
plt.ylabel('Well Depth')
```

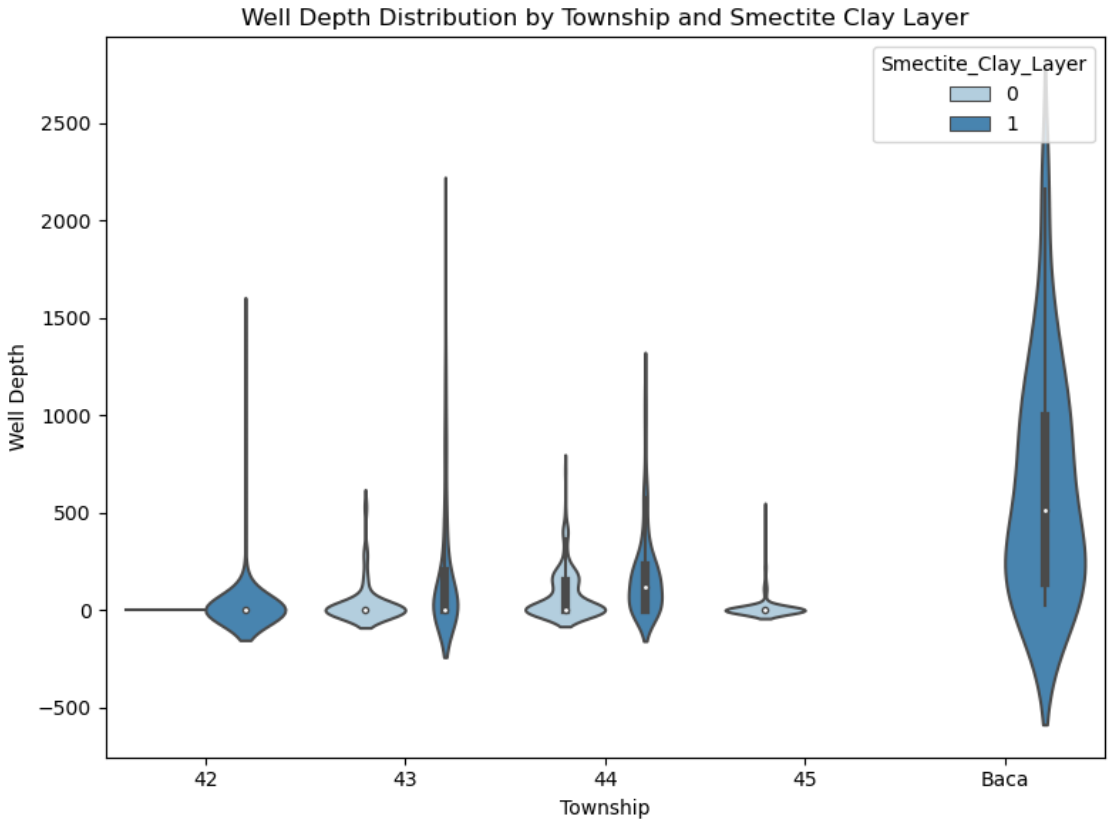
```
plt.title('Well Depth Distribution by Township and Smectite Clay Layer')
plt.tight_layout()
plt.show()
```



[13]:
```
plt.figure(figsize=(8, 6))
sns.kdeplot(data['Elevation_Of_Clay_Layer'], shade=True, color=(0.298, 0.447, 0.
↪690))
plt.xlabel('Elevation of Clay Layer')
plt.ylabel('Density')
plt.title('Distribution of Elevation of Clay Layer')
plt.tight_layout()
plt.show()
```



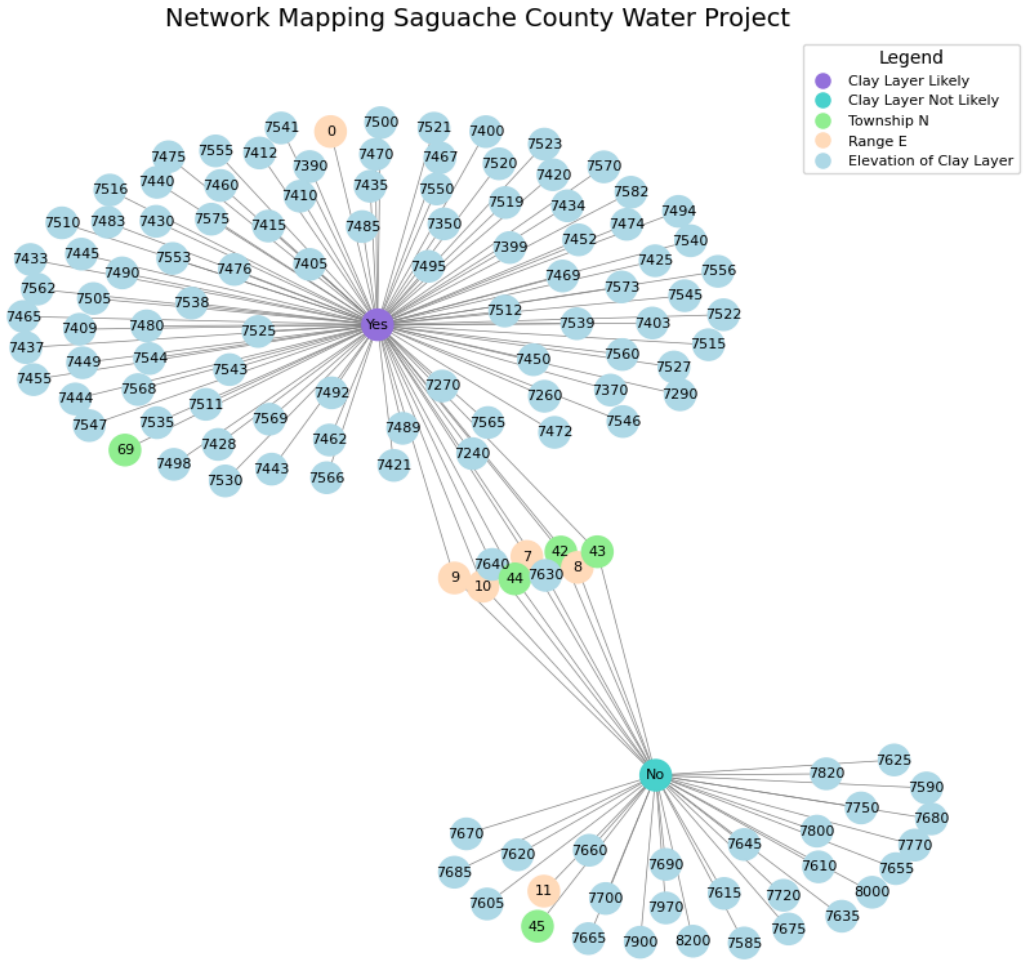
[14]:
```
# Violin Plot
plt.figure(figsize=(8, 6))
violin_plot = sns.violinplot(x='Township_N', y='Well_Depth',␣
 ↪hue='Smectite_Clay_Layer', data=data.drop('Section', axis=1),␣
 ↪palette='Blues')
plt.xlabel('Township')
plt.ylabel('Well Depth')
plt.title('Well Depth Distribution by Township and Smectite Clay Layer')
plt.tight_layout()

# Replace "69" with "Baca" in the x-axis labels
township_labels = violin_plot.get_xticklabels()
township_labels = [label.get_text().replace('69', 'Baca') for label in␣
 ↪township_labels]
violin_plot.set_xticklabels(township_labels)

plt.show()
```



[15]:
```
import pandas as pd
import networkx as nx
import matplotlib.pyplot as plt

# Read the data
idata = pd.read_csv('C:/Users/newmy/Desktop/WaterProj/dWaterProjDataB.csv')

# Subset the relevant columns for the network graph
subset = idata[['Smectite_Clay_Layer', 'Township_N', 'Range_E',␣
 ↪'Elevation_Of_Clay_Layer']]

# Check if 'Smectite_Clay_Layer' column exists in the DataFrame
if 'Smectite_Clay_Layer' in subset.columns:
    # Replace 1 with 'Yes' and 0 with 'No' in Smectite_Clay_Layer column
    subset.loc[:, 'Smectite_Clay_Layer'] = subset['Smectite_Clay_Layer'].map({1:
 ↪ 'Yes', 0: 'No'})

# Create an empty NetworkX graph object
graph = nx.Graph()
```

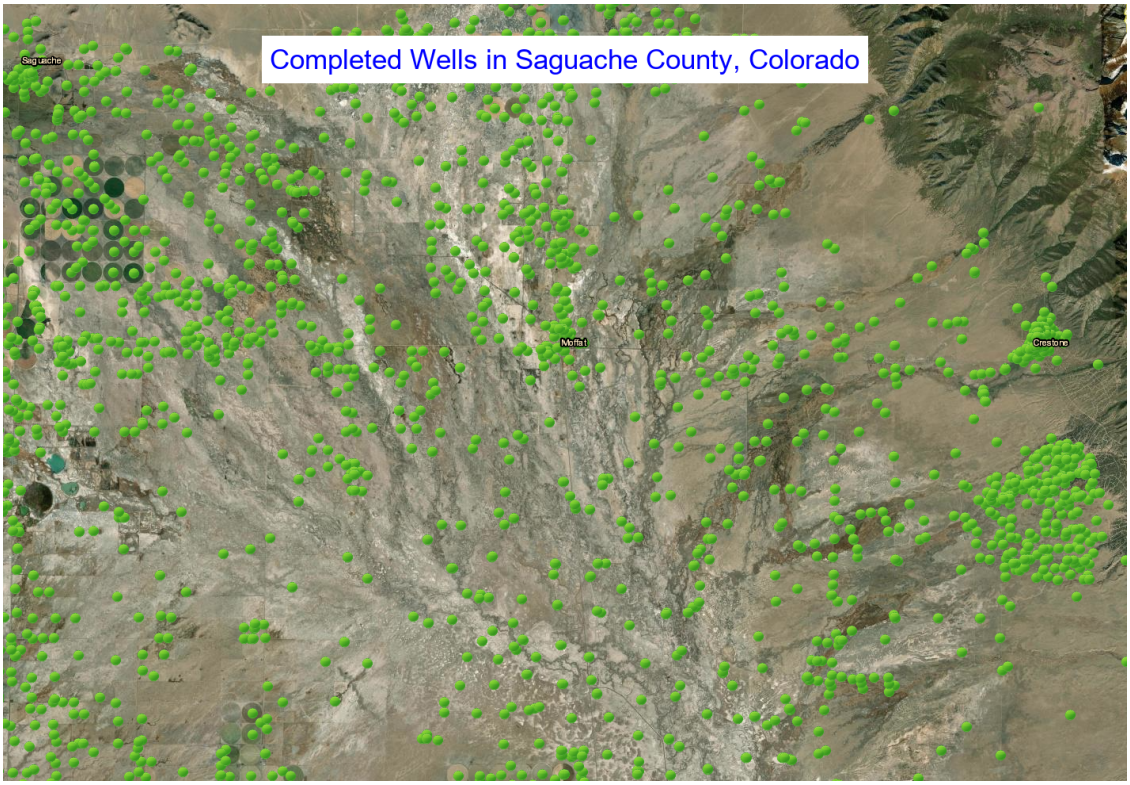
```
# Add the smectite_clay_layer nodes to the graph
graph.add_nodes_from(subset['Smectite_Clay_Layer'])

# Iterate over the data and add edges based on Township_N, Range_E, and␣
 ↪Elevation_to_Clay_Layer
for _, row in subset.iterrows():
    clay_layer = row['Smectite_Clay_Layer']
    township = row['Township_N']
    range_value = row['Range_E']
    elevation = row['Elevation_Of_Clay_Layer']

    # Add edges based on Township_N
    graph.add_edge(clay_layer, township)

    # Add edges based on Range_E
    graph.add_edge(clay_layer, range_value)

    # Add edges based on Elevation_to_Clay_Layer
    graph.add_edge(clay_layer, elevation)

# Define color mappings for nodes
color_map = {
    'Yes': 'mediumpurple',
    'No': 'mediumturquoise',
    **{str(i): 'lightgreen' for i in range(41, 70)},  # Change the color␣
 ↪mapping for range values 41 to 69 to green
    **{str(i): 'peachpuff' for i in range(0, 12)},  # Add the color mapping for␣
 ↪range values 6 to 11 as orange
    **{str(i): 'lightblue' for i in range(7000, 8300)},  # Add the color␣
 ↪mapping for elevation values 7000 to 8200 as lightblue
}

# Get the node colors
node_colors = [color_map.get(str(label), 'gray') for label in graph.nodes()]

# Create the plot
plt.figure(figsize=(11, 10))
pos = nx.spring_layout(graph)
nx.draw_networkx(graph, pos=pos, with_labels=True, node_color=node_colors,␣
 ↪edge_color='gray', font_size=8, width=0.5)

# Create the legend
legend_labels = [('Clay Layer Likely', 'mediumpurple'), ('Clay Layer Not␣
 ↪Likely', 'mediumturquoise'),
                 ('Township N', 'lightgreen'), ('Range E', 'peachpuff'),␣
 ↪('Elevation of Clay Layer', 'lightblue')]
```

```
legend_elements = [plt.Line2D([], [], marker='o', markersize=8, linestyle='',␣
 ↪label=label, color=color)
                   for label, color in legend_labels]
plt.legend(handles=legend_elements, loc='upper right', title='Legend', fontsize␣
 ↪= 8)

plt.title('Network Mapping Saguache County Water Project', fontsize=14)  # Add␣
 ↪a title for the graph
plt.axis('off')
plt.show()
```



[16]:
```
# Open the image
image_path2 = 'C:/Users/newmy/Desktop/WaterProj/iWells.png'
image2 = Image.open(image_path2)
```

```
# Define the font and font size
font = ImageFont.truetype("arial.ttf", size=35)

# Define the text to be annotated
text = "Completed Wells in Saguache County, Colorado"

# Calculate the position to place the text
text_width, text_height = font.getsize(text)
x = (image2.width - text_width) // 2
y = 50

# Set the color of the highlight bar
highlight_color = (255, 255, 255)  #  color

# Calculate the dimensions of the highlight bar
highlight_width = text_width + 20
highlight_height = text_height + 20
highlight_x = (image2.width - highlight_width) // 2
highlight_y = y - 10

# Create an ImageDraw object
draw2 = ImageDraw.Draw(image2)

# Draw the highlight bar
draw2.rectangle((highlight_x, highlight_y, highlight_x + highlight_width,␣
 ↪highlight_y + highlight_height),
                fill=highlight_color)

# Set the color of the text
text_color = (0, 0, 255)  # Blue color

# Annotate the image with the text
draw2.text((x, y), text, fill=text_color, font=font)

# Save the annotated image in PNG format
annotated_image_path2 = 'C:/Users/newmy/Desktop/WaterProj/annotated_image2.png'
image2.save(annotated_image_path2, format='PNG')

# Display the annotated image
display(DisplayImage(filename=annotated_image_path2))
```



5 Build, Test and Visualize the Model Performance

[17]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Conv1D, MaxPooling1D
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import EarlyStopping

# Read the data
data = pd.read_csv('C:/Users/newmy/Desktop/WaterProj/dWaterProjDataB.csv')

# Separate the input features (X) and the target label (y)
X = data.drop(columns=['Smectite_Clay_Layer'])
y = data['Smectite_Clay_Layer']

# Split the data into train, test, and validation sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15,␣
 ↪random_state=42)
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
 ↪15, random_state=42)

# Perform any necessary preprocessing
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)
X_test_scaled = scaler.transform(X_test)

# Build the deep learning model
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=3, activation='relu',␣
 ↪input_shape=(X_train_scaled.shape[1], 1)))
model.add(MaxPooling1D(pool_size=2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(64, activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Define the optimizer with a smaller learning rate
optimizer = Adam(learning_rate = 0.0001)  # Set the learning rate to 0.0001

# Compile the model with the new optimizer
model.compile(optimizer=optimizer, loss='binary_crossentropy',␣
 ↪metrics=['accuracy'])

# Define Early Stopping callback
early_stopping = EarlyStopping(patience=3, monitor='val_loss')

# Train the model with Early Stopping
history = model.fit(X_train_scaled, y_train, epochs=50, batch_size=32,␣
 ↪validation_data=(X_val_scaled, y_val), callbacks=[early_stopping], verbose=1)

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(X_test_scaled, y_test)
print('Test Loss:', test_loss)
print('Test Accuracy:', test_accuracy)

# Calculate test accuracy as a percentage
test_accuracy_percentage = test_accuracy * 100
print('Test Accuracy (Percentage): {:.2f}%'.format(test_accuracy_percentage))

# Make predictions
predictions = model.predict(X_test_scaled)
predicted_labels = [1 if pred > 0.5 else 0 for pred in predictions]

# Plot the training and validation accuracy
```

```
train_accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
train_loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(train_accuracy) + 1)
```

```
Epoch 1/50
23/23 [==============================] - 1s 9ms/step - loss: 0.6918 - accuracy:
0.6222 - val_loss: 0.6850 - val_accuracy: 0.7222
Epoch 2/50
23/23 [==============================] - 0s 2ms/step - loss: 0.6706 - accuracy:
0.7809 - val_loss: 0.6659 - val_accuracy: 0.7778
Epoch 3/50
23/23 [==============================] - 0s 2ms/step - loss: 0.6514 - accuracy:
0.8230 - val_loss: 0.6437 - val_accuracy: 0.8571
Epoch 4/50
23/23 [==============================] - 0s 2ms/step - loss: 0.6301 - accuracy:
0.8581 - val_loss: 0.6185 - val_accuracy: 0.8968
Epoch 5/50
23/23 [==============================] - 0s 2ms/step - loss: 0.6076 - accuracy:
0.8596 - val_loss: 0.5931 - val_accuracy: 0.8889
Epoch 6/50
23/23 [==============================] - 0s 3ms/step - loss: 0.5829 - accuracy:
0.8652 - val_loss: 0.5652 - val_accuracy: 0.8968
Epoch 7/50
23/23 [==============================] - 0s 2ms/step - loss: 0.5560 - accuracy:
0.8680 - val_loss: 0.5355 - val_accuracy: 0.9048
Epoch 8/50
23/23 [==============================] - 0s 2ms/step - loss: 0.5279 - accuracy:
0.8834 - val_loss: 0.5058 - val_accuracy: 0.8810
Epoch 9/50
23/23 [==============================] - 0s 2ms/step - loss: 0.4988 - accuracy:
0.8806 - val_loss: 0.4753 - val_accuracy: 0.9048
Epoch 10/50
23/23 [==============================] - 0s 2ms/step - loss: 0.4695 - accuracy:
0.8862 - val_loss: 0.4436 - val_accuracy: 0.9206
Epoch 11/50
23/23 [==============================] - 0s 2ms/step - loss: 0.4395 - accuracy:
0.8904 - val_loss: 0.4137 - val_accuracy: 0.9048
Epoch 12/50
23/23 [==============================] - 0s 2ms/step - loss: 0.4097 - accuracy:
0.9031 - val_loss: 0.3838 - val_accuracy: 0.9286
Epoch 13/50
23/23 [==============================] - 0s 2ms/step - loss: 0.3803 - accuracy:
0.9129 - val_loss: 0.3563 - val_accuracy: 0.9365
Epoch 14/50
23/23 [==============================] - 0s 2ms/step - loss: 0.3534 - accuracy:
0.9228 - val_loss: 0.3305 - val_accuracy: 0.9365
```

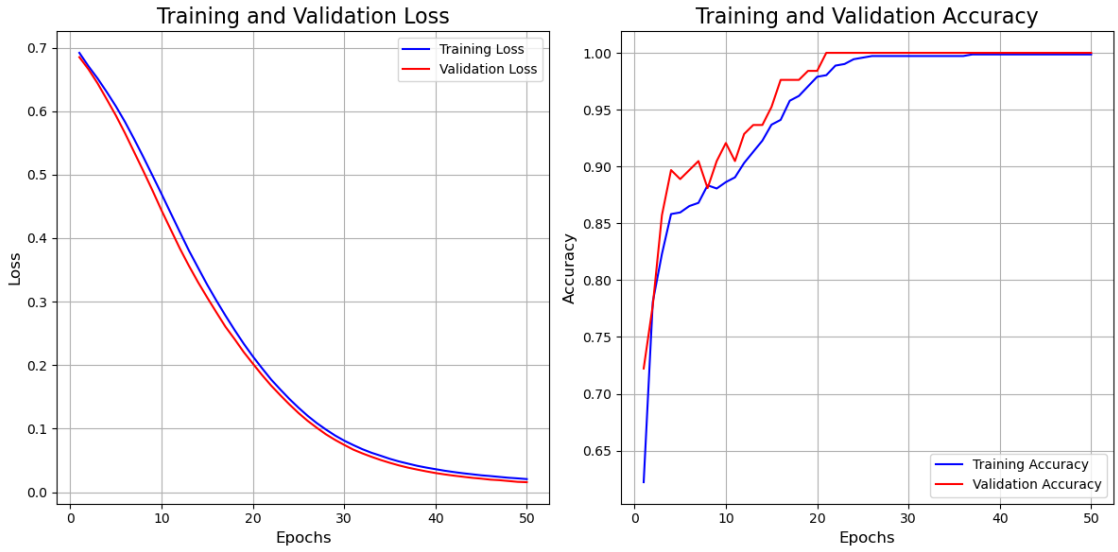
```
Epoch 15/50
23/23 [==============================] - 0s 2ms/step - loss: 0.3266 - accuracy:
0.9368 - val_loss: 0.3065 - val_accuracy: 0.9524
Epoch 16/50
23/23 [==============================] - 0s 2ms/step - loss: 0.3018 - accuracy:
0.9410 - val_loss: 0.2832 - val_accuracy: 0.9762
Epoch 17/50
23/23 [==============================] - 0s 2ms/step - loss: 0.2780 - accuracy:
0.9579 - val_loss: 0.2604 - val_accuracy: 0.9762
Epoch 18/50
23/23 [==============================] - 0s 2ms/step - loss: 0.2553 - accuracy:
0.9621 - val_loss: 0.2410 - val_accuracy: 0.9762
Epoch 19/50
23/23 [==============================] - 0s 2ms/step - loss: 0.2337 - accuracy:
0.9705 - val_loss: 0.2206 - val_accuracy: 0.9841
Epoch 20/50
23/23 [==============================] - 0s 2ms/step - loss: 0.2137 - accuracy:
0.9789 - val_loss: 0.2023 - val_accuracy: 0.9841
Epoch 21/50
23/23 [==============================] - 0s 2ms/step - loss: 0.1952 - accuracy:
0.9803 - val_loss: 0.1845 - val_accuracy: 1.0000
Epoch 22/50
23/23 [==============================] - 0s 2ms/step - loss: 0.1774 - accuracy:
0.9888 - val_loss: 0.1678 - val_accuracy: 1.0000
Epoch 23/50
23/23 [==============================] - 0s 2ms/step - loss: 0.1617 - accuracy:
0.9902 - val_loss: 0.1527 - val_accuracy: 1.0000
Epoch 24/50
23/23 [==============================] - 0s 2ms/step - loss: 0.1468 - accuracy:
0.9944 - val_loss: 0.1382 - val_accuracy: 1.0000
Epoch 25/50
23/23 [==============================] - 0s 2ms/step - loss: 0.1330 - accuracy:
0.9958 - val_loss: 0.1246 - val_accuracy: 1.0000
Epoch 26/50
23/23 [==============================] - 0s 2ms/step - loss: 0.1202 - accuracy:
0.9972 - val_loss: 0.1123 - val_accuracy: 1.0000
Epoch 27/50
23/23 [==============================] - 0s 2ms/step - loss: 0.1087 - accuracy:
0.9972 - val_loss: 0.1014 - val_accuracy: 1.0000
Epoch 28/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0986 - accuracy:
0.9972 - val_loss: 0.0911 - val_accuracy: 1.0000
Epoch 29/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0893 - accuracy:
0.9972 - val_loss: 0.0823 - val_accuracy: 1.0000
Epoch 30/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0811 - accuracy:
0.9972 - val_loss: 0.0744 - val_accuracy: 1.0000
```

```
Epoch 31/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0741 - accuracy:
0.9972 - val_loss: 0.0670 - val_accuracy: 1.0000
Epoch 32/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0676 - accuracy:
0.9972 - val_loss: 0.0610 - val_accuracy: 1.0000
Epoch 33/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0620 - accuracy:
0.9972 - val_loss: 0.0556 - val_accuracy: 1.0000
Epoch 34/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0573 - accuracy:
0.9972 - val_loss: 0.0506 - val_accuracy: 1.0000
Epoch 35/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0525 - accuracy:
0.9972 - val_loss: 0.0462 - val_accuracy: 1.0000
Epoch 36/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0483 - accuracy:
0.9972 - val_loss: 0.0421 - val_accuracy: 1.0000
Epoch 37/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0448 - accuracy:
0.9986 - val_loss: 0.0386 - val_accuracy: 1.0000
Epoch 38/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0415 - accuracy:
0.9986 - val_loss: 0.0356 - val_accuracy: 1.0000
Epoch 39/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0387 - accuracy:
0.9986 - val_loss: 0.0327 - val_accuracy: 1.0000
Epoch 40/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0362 - accuracy:
0.9986 - val_loss: 0.0301 - val_accuracy: 1.0000
Epoch 41/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0337 - accuracy:
0.9986 - val_loss: 0.0279 - val_accuracy: 1.0000
Epoch 42/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0317 - accuracy:
0.9986 - val_loss: 0.0260 - val_accuracy: 1.0000
Epoch 43/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0298 - accuracy:
0.9986 - val_loss: 0.0243 - val_accuracy: 1.0000
Epoch 44/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0282 - accuracy:
0.9986 - val_loss: 0.0226 - val_accuracy: 1.0000
Epoch 45/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0265 - accuracy:
0.9986 - val_loss: 0.0213 - val_accuracy: 1.0000
Epoch 46/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0254 - accuracy:
0.9986 - val_loss: 0.0197 - val_accuracy: 1.0000
```

```
Epoch 47/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0240 - accuracy:
0.9986 - val_loss: 0.0189 - val_accuracy: 1.0000
Epoch 48/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0226 - accuracy:
0.9986 - val_loss: 0.0175 - val_accuracy: 1.0000
Epoch 49/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0217 - accuracy:
0.9986 - val_loss: 0.0162 - val_accuracy: 1.0000
Epoch 50/50
23/23 [==============================] - 0s 2ms/step - loss: 0.0206 - accuracy:
0.9986 - val_loss: 0.0157 - val_accuracy: 1.0000
5/5 [==============================] - 0s 869us/step - loss: 0.0145 - accuracy:
1.0000
Test Loss: 0.014485756866633892
Test Accuracy: 1.0
Test Accuracy (Percentage): 100.00%
5/5 [==============================] - 0s 3ms/step
```

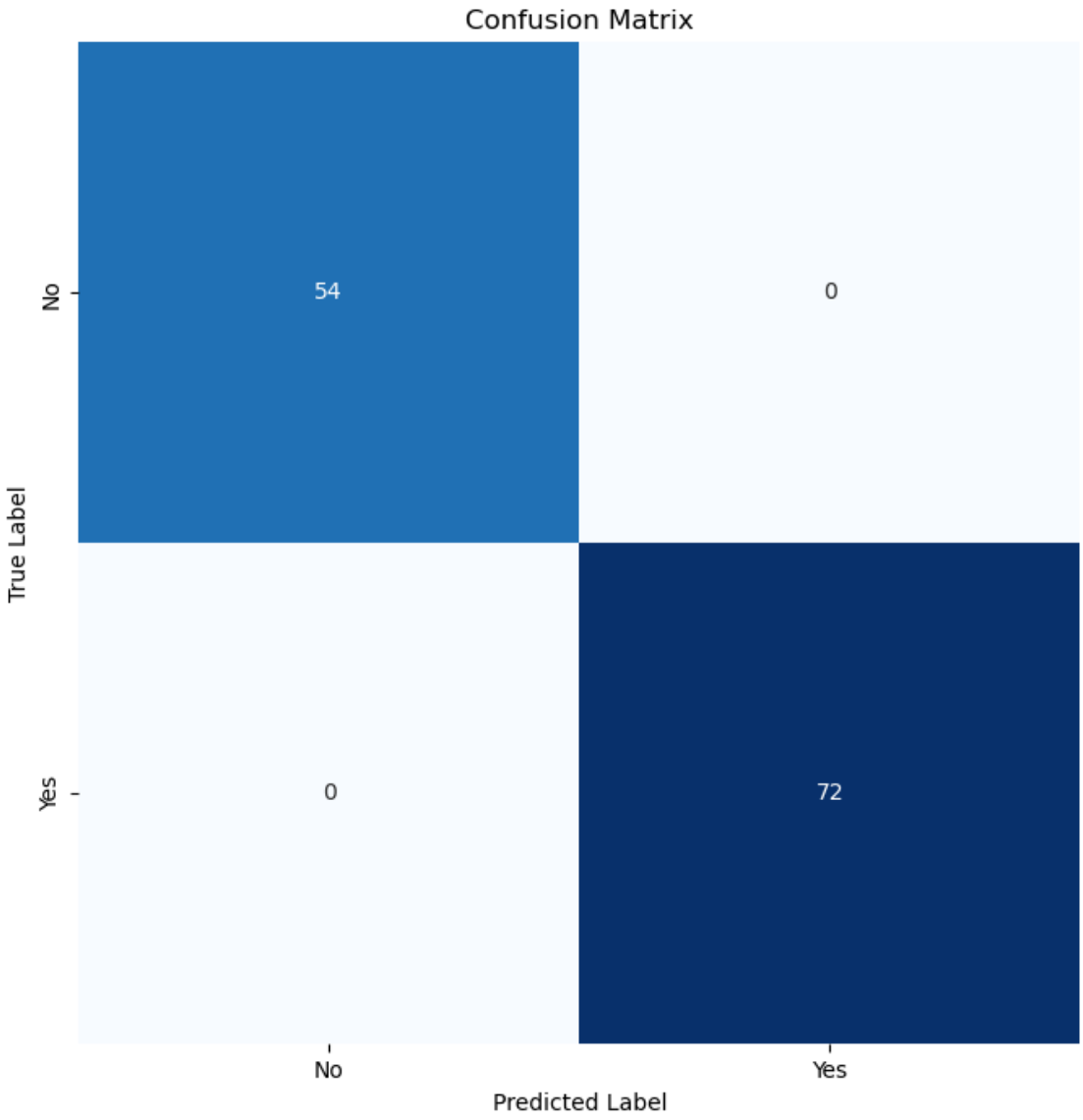
[18]:
```python
import matplotlib.pyplot as plt

# Create the figure and axes
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(12, 6))

# Plot training and validation loss
ax1.plot(epochs, train_loss, 'b', label='Training Loss')  # Truncate the␣
 ↪train_loss array
ax1.plot(epochs, val_loss, 'r', label='Validation Loss')  # Truncate the␣
 ↪val_loss array
ax1.set_title('Training and Validation Loss', fontsize=16)
ax1.set_xlabel('Epochs', fontsize=12)
ax1.set_ylabel('Loss', fontsize=12)
ax1.legend(loc='upper right')

# Plot training and validation accuracy
ax2.plot(epochs, train_accuracy, 'b', label='Training Accuracy')
ax2.plot(epochs, val_accuracy, 'r', label='Validation Accuracy')
ax2.set_title('Training and Validation Accuracy', fontsize=16)
ax2.set_xlabel('Epochs', fontsize=12)
ax2.set_ylabel('Accuracy', fontsize=12)
ax2.legend(loc='lower right')

plt.subplots_adjust(wspace=0.3)
ax1.grid(True)
ax2.grid(True)

plt.tight_layout()
```

```
plt.show()
```



[19]:
```
# Generate predictions on the validation set
validation_predictions = model.predict(X_val_scaled)
predicted_labels = [1 if pred > 0.5 else 0 for pred in validation_predictions]

# Compute the confusion matrix
confusion = confusion_matrix(y_val, predicted_labels)

# Create a figure and axes
fig, ax = plt.subplots(figsize=(10, 8))

# Customize the heatmap
sns.heatmap(confusion, annot=True, cmap='Blues', fmt='d', cbar=False,␣
 ↪square=True,
            xticklabels=['No', 'Yes'], yticklabels=['No', 'Yes'], ax=ax)

# Add labels and title
ax.set_xlabel('Predicted Label')
ax.set_ylabel('True Label')
ax.set_title('Confusion Matrix')

# Show the plot
plt.show()
```

```
4/4 [==============================] - 0s 668us/step
```



[20]:
```
from tabulate import tabulate

# Make predictions
predictions = model.predict(X_test_scaled)
predicted_labels = [1 if pred > 0.5 else 0 for pred in predictions]

# Create a DataFrame with elevation and predictions
elevation_predictions = pd.DataFrame({'Elevation of Clay Layer':␣
 ↪X_test['Elevation'], 'Predicted Label': predicted_labels})

# Filter the DataFrame based on predictions
```

```
predictions_1 = elevation_predictions[elevation_predictions['Predicted Label']␣
 ↪== 1]
predictions_0 = elevation_predictions[elevation_predictions['Predicted Label']␣
 ↪== 0]

# Display the tables
if not predictions_1.empty:
    print("Predictions with Elevation for Predicted Label 1:")
    print(tabulate(predictions_1, headers='keys', tablefmt='psql'))
else:
    print("No Predictions with Elevation for Predicted Label 1")

if not predictions_0.empty:
    print("Elevation for Predicted Label 0:")
    print(tabulate(predictions_0[['Elevation of Clay Layer']], headers='keys',␣
 ↪tablefmt='psql'))
else:
    print("No Elevation for Predicted Label 0")

# Save the model to a file
with open('model.pkl', 'wb') as file:
    pickle.dump(model, file)

    # Calculate the average elevation of the predicted values
average_elevation = np.mean(elevation_predictions['Elevation of Clay Layer'])

# Calculate the average elevation for predicted values of 1
average_elevation_1 = np.mean(predictions_1['Elevation of Clay Layer'])
print(f"Average Elevation for Predicted Label Smectite Clay Layer:␣
 ↪{average_elevation_1:.2f}")

# Calculate the average elevation for predicted values of 0
average_elevation_0 = np.mean(predictions_0['Elevation of Clay Layer'])
print(f"Average Elevation for Predicted Label No Smectite Clay Layer:␣
 ↪{average_elevation_0:.2f}")

# Save the model to a file
with open('model.pkl', 'wb') as file:
    pickle.dump(model, file)
```

```
5/5 [==============================] - 0s 750us/step
Predictions with Elevation for Predicted Label 1:
+-----+---------------------------+-------------------+
|     |   Elevation of Clay Layer |   Predicted Label |
|-----+---------------------------+-------------------|
| 613 |                      7610 |                 1 |
| 731 |                      7605 |                 1 |
```

275	7570	1
582	7620	1
299	7570	1
351	7560	1
594	7635	1
652	7590	1
294	7565	1
66	7565	1
377	7580	1
755	7580	1
107	7560	1
139	7560	1
67	7565	1
213	7610	1
670	7590	1
363	7580	1
76	7565	1
298	7570	1
88	7560	1
59	7565	1
237	7585	1
221	7605	1
362	7580	1
477	7610	1
266	7575	1
589	7635	1
281	7580	1
643	7605	1
669	7595	1
479	7620	1
280	7580	1
535	7640	1
758	7590	1
734	7595	1
588	7635	1
660	7580	1
630	7605	1
983	7570	1
136	7560	1
39	7575	1
359	7580	1
244	7585	1
296	7560	1
355	7565	1
70	7565	1
342	7565	1
312	7560	1
761	7580	1

310	7560	1
63	7565	1
199	7590	1
593	7625	1
96	7560	1
321	7560	1
542	7640	1
971	7560	1
120	7565	1
86	7550	1
570	7615	1
547	7650	1
558	7645	1
533	7635	1
668	7590	1
689	7640	1
615	7605	1
254	7570	1
981	7570	1
55	7565	1
746	7640	1
247	7580	1
260	7590	1
72	7565	1
655	7610	1
44	7565	1
218	7630	1
618	7610	1
286	7565	1
694	7630	1
567	7630	1
314	7555	1
215	7635	1
678	7575	1
635	7610	1
259	7590	1
333	7570	1
60	7565	1
521	7620	1
587	7630	1
730	7605	1
585	7625	1
370	7560	1
290	7565	1
198	7590	1
305	7560	1

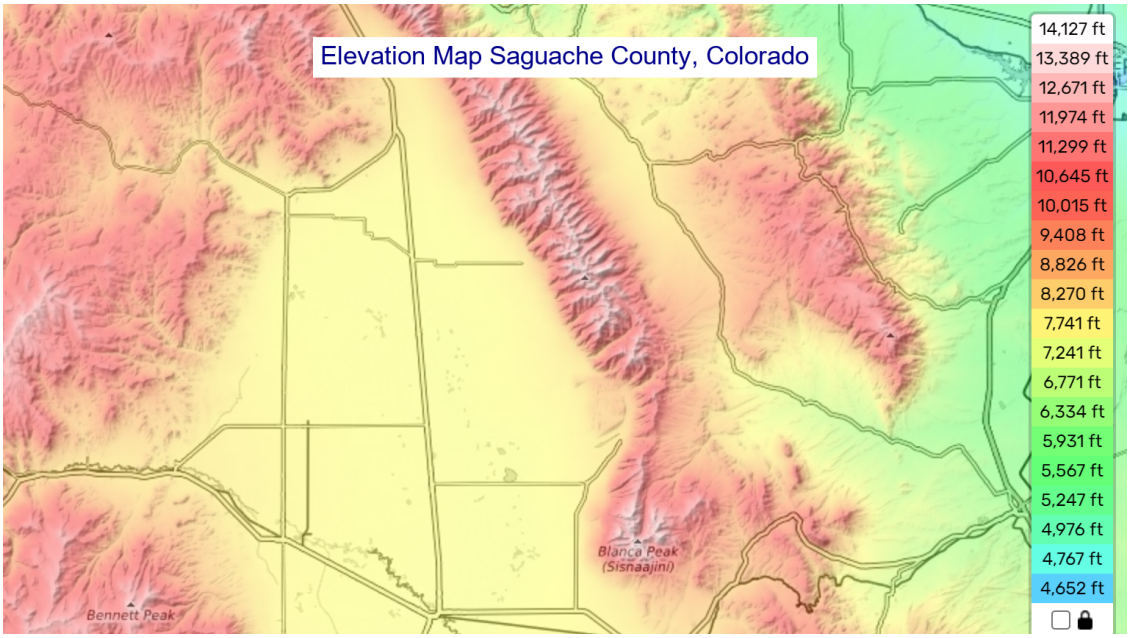
Elevation for Predicted Label 0:

	Elevation of Clay Layer
451	7820
436	7645
707	7610
718	7670
494	7675
866	8000
682	7770
832	7970
486	7665
425	7640
420	7640
849	8000
23	7700
30	7700
168	7645
493	7685
778	8200
490	7685
440	7680
900	7800
938	7800
449	7700
921	7800
158	7640
947	7800
854	8000
813	7970
516	7635
712	7610
929	7800
961	7800
789	8200
879	7800
888	7900
792	8200
910	7645
922	7800
798	8200
184	7590
836	7970
819	7970
522	7610
870	8000
916	7645
823	7970

```
| 807 |                        7970 |
| 519 |                        7620 |
| 394 |                        7690 |
| 927 |                        7800 |
| 465 |                        7630 |
| 853 |                        8000 |
| 500 |                        7675 |
+-----+-----------------------------+
Average Elevation for Predicted Label Smectite Clay Layer: 7589.43
Average Elevation for Predicted Label No Smectite Clay Layer: 7800.77
```

[21]:
```python
# Open the image
image_path3 = 'C:/Users/newmy/Desktop/WaterProj/iElevation.png'
image3 = Image.open(image_path3)

# Create an ImageDraw object
draw3 = ImageDraw.Draw(image3)

# Define the font and font size
font = ImageFont.truetype("arial.ttf", size=35)

# Define the text to be annotated
text = "Elevation Map Saguache County, Colorado"

# Calculate the position to place the text
text_width, text_height = draw3.textsize(text, font=font)
x = (image3.width - text_width) // 2
y = 50  # Place the annotation at the top

# Set the color of the text
text_color = (0, 0, 139)  # Dark blue color

# Set the size of the highlighter bar
highlighter_height = text_height + 10

# Calculate the coordinates for the highlighter bar
bar_x1 = x - 10
bar_y1 = y - 5
bar_x2 = x + text_width + 10
bar_y2 = y + highlighter_height

# Draw the white highlighter bar
highlighter_color = (255, 255, 255)  # White color
draw3.rectangle([(bar_x1, bar_y1), (bar_x2, bar_y2)], fill=highlighter_color)

# Annotate the image with the text
draw3.text((x, y), text, fill=text_color, font=font)
```

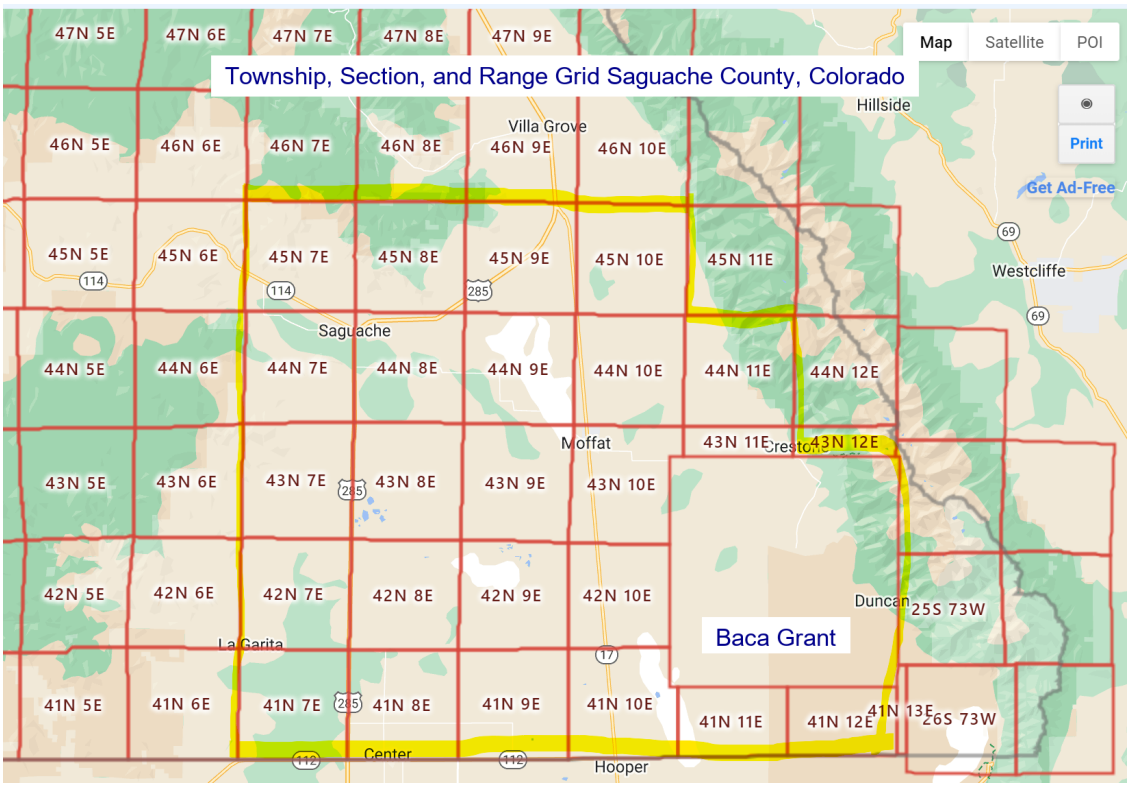
```
# Save the annotated image in PNG format
annotated_image_path3 = 'C:/Users/newmy/Desktop/WaterProj/annotated_image3.png'
image3.save(annotated_image_path3, format='PNG')

# Display the annotated image
display(DisplayImage(filename=annotated_image_path3))
```



6 Present User Interface for Predictions

[22]:
```
# Open the image
image_path4 = 'C:/Users/newmy/Desktop/WaterProj/iSTR.png'
image4 = Image.open(image_path4)

# Create an ImageDraw object
draw4 = ImageDraw.Draw(image4)

# Define the font and font size
font = ImageFont.truetype("arial.ttf", size=35)

# Define the text to be annotated
text_top = "Township, Section, and Range Grid Saguache County, Colorado"
text_bottom = "Baca Grant"

# Calculate the position to place the text
text_top_width, text_top_height = draw4.textsize(text_top, font=font)
```

```
text_bottom_width, text_bottom_height = draw4.textsize(text_bottom, font=font)
x_top = (image4.width - text_top_width) // 2
y_top = 50 + 10 + 25  # Place the top annotation at the desired position
x_bottom = image4.width - text_bottom_width - 425  # Place the bottom␣
 ↪annotation on the right side
y_bottom = image4.height - text_bottom_height - 200  # Place the bottom␣
 ↪annotation on the bottom right corner

# Set the color of the text
text_color = (0, 0, 139)  # Dark blue color

# Calculate the coordinates for the top highlighter bar
padding_x = 20
padding_y = 10
highlighter_top_x = x_top - padding_x
highlighter_top_y = y_top - padding_y
highlighter_top_width = text_top_width + 2 * padding_x
highlighter_top_height = text_top_height + 2 * padding_y

# Calculate the coordinates for the bottom highlighter bar
highlighter_bottom_x = x_bottom - padding_x
highlighter_bottom_y = y_bottom - padding_y
highlighter_bottom_width = text_bottom_width + 2 * padding_x
highlighter_bottom_height = text_bottom_height + 2 * padding_y

# Draw the white highlighter bars
highlighter_color = (255, 255, 255)  # White color
draw4.rectangle((highlighter_top_x, highlighter_top_y, highlighter_top_x +␣
 ↪highlighter_top_width, highlighter_top_y + highlighter_top_height),␣
 ↪fill=highlighter_color)
draw4.rectangle((highlighter_bottom_x, highlighter_bottom_y,␣
 ↪highlighter_bottom_x + highlighter_bottom_width, highlighter_bottom_y +␣
 ↪highlighter_bottom_height), fill=highlighter_color)

# Annotate the image with the top text
draw4.text((x_top, y_top), text_top, fill=text_color, font=font)

# Annotate the image with the bottom text
draw4.text((x_bottom, y_bottom), text_bottom, fill=text_color, font=font)

# Save the annotated image in PNG format
annotated_image_path4 = 'C:/Users/newmy/Desktop/WaterProj/annotated_image4.png'
image4.save(annotated_image_path4, format='PNG')

# Display the annotated image
display(DisplayImage(filename=annotated_image_path4))
```



[23]:
```
while True:
    # Prompt the user to enter section, township, and range values
    township = int(input("Enter the township: "))
    range_value = int(input("Enter the range: "))
    section = int(input("Enter the section: "))

    # Filter the data based on user input
    indices = (data['Township_N'] == township) & (data['Range_E'] ==␣
 ↪range_value) & (data['Section'] == section)
    user_data = data.loc[indices]

    if len(user_data) > 0:
        # Remove the unnecessary column from user data
        user_data = user_data.drop(columns=['Smectite_Clay_Layer'])

        # Scale the user input using the same scaler used for training data
        user_data_scaled = scaler.transform(user_data)

        # Make predictions on the user input
        predictions = model.predict(user_data_scaled)
        predicted_label = [1 if pred > 0.5 else 0 for pred in predictions]
```

```
        if predicted_label[0] == 1:
            # Get the corresponding elevation of the clay layer from the␣
↪original dataset
            elevation_of_clay_layer = data.loc[indices,␣
↪'Elevation_Of_Clay_Layer']

            if len(elevation_of_clay_layer) > 1:
                # Calculate the average elevation
                average_elevation = elevation_of_clay_layer.mean()

                # Display the average elevation of the clay layer
                print("Smectite clay layer likely at elevation:",␣
↪average_elevation, "ft.")
            else:
                # Display the elevation of the clay layer
                print("Smectite clay layer likely at elevation:",␣
↪elevation_of_clay_layer.item(), "ft.")
        else:
            print("No smectite clay layer predicted.")
    else:
        print("No matching data found for the given township, range, and␣
↪section.")

    # Prompt the user for the next action
    choice = input("Enter '1' to re-enter data or any other key to quit: ")
    if choice != '1':
        break
```

```
Enter the township: 43
Enter the range: 9
Enter the section: 13
1/1 [==============================] - 0s 5ms/step
Smectite clay layer likely at elevation: 7415 ft.
Enter '1' to re-enter data or any other key to quit:
```

7 Conclusion

The final model used in the Saguache County Water Study is a deep learning model with a specific architecture. The architecture of the deep learning model consists of multiple layers designed to process the input data and make predictions. Here is a description of each layer in the model:

Conv1D Layer: This layer performs 1D convolution on the input data. It has 32 filters and a kernel size of 3. The activation function used is ReLU, which introduces non-linearity to the model. The input shape is determined based on the number of features in the scaled training data.

MaxPooling1D Layer: This layer performs max pooling operation to downsample the output from the previous convolutional layer. It uses a pool size of 2, which means it takes the maximum value within a sliding window of size 2.

Flatten Layer: This layer flattens the output from the previous layer into a one-dimensional vector. It reshapes the data from a 2D representation to a 1D representation, preparing it for the subsequent fully connected layers.

Dense Layers: Two dense layers follow the flatten layer. The first dense layer has 128 units with a ReLU activation function, which helps to capture complex relationships in the data. The second dense layer has 64 units, also using the ReLU activation function.

Output Layer: The final dense layer consists of a single unit with a sigmoid activation function. It produces the output prediction for the presence of the smectite clay layer. The sigmoid activation function squeezes the output between 0 and 1, representing the probability of the positive class (presence of the clay layer).

The model utilizes the Adam optimizer with a learning rate of 0.0001 and is compiled with the binary cross-entropy loss function, which is suitable for binary classification tasks. Early stopping is employed as a callback, with a patience of 3 epochs and monitoring the validation loss.

The model architecture leverages convolutional layers to capture local patterns and spatial dependencies in the input data, followed by dense layers to learn higher-level representations and make predictions. This combination of convolutional and dense layers enables the model to effectively process the well completion report data and predict the presence of the smectite clay layer.

After training the model, it is evaluated on the test set to assess its performance. The test loss and test accuracy are calculated and displayed. Additionally, the test accuracy is also presented as a percentage for better interpretation. The accuracy of the model is illustrated by the Confusion Matrix where all test data was correctly identified.

To utilize the trained model, users have the option to input Township, Range, and Section values. These values are used as input to the model, which predicts the presence of the smectite clay layer at the specified location. If the model predicts the presence of the clay layer, the average elevation of the layer at that location is returned as an output.

[ ]: