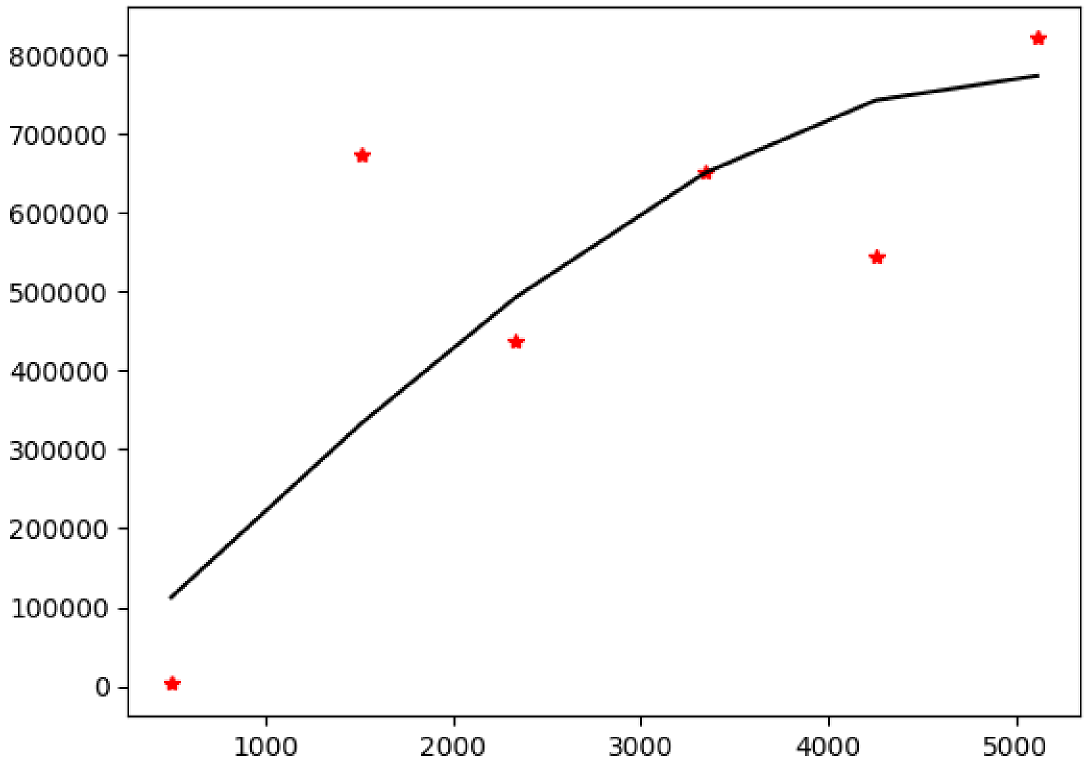
new-wells

May 13, 2024

[1]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
from loadAndPreprocess import load_and_preprocess

well_info, _ = load_and_preprocess()
```

[3]:
```
X = well_info['X']
Y = well_info['Y']

# Define grid points for interpolation
grid_x = np.arange(min(X)-150, max(X)+150, 1.0)
grid_y = np.arange(min(Y)-150, max(Y)+150, 1.0)

from pykrige.ok import OrdinaryKriging

values = (well_info['Total Resources'])

ok_res = OrdinaryKriging(
    X,
    Y,
    values,
    variogram_model='spherical',
    enable_plotting=True
)

# Perform the interpolation
z_res, ss_res = ok_res.execute('grid', grid_x, grid_y)
```



[4]:
```python
from scipy.spatial import distance_matrix
def objective_function(x, existing_wells, z_res):
    # Reshape x to (n_particles, n_wells, 2)
    n_particles = x.shape[0]
    n_wells = x.shape[1] // 2
    new_wells = x.reshape(n_particles, n_wells, 2)

    # Initialize return array
    j = np.zeros(n_particles)

    for i in range(n_particles):
        all_wells = np.vstack((existing_wells, new_wells[i]))
        dist_mat = distance_matrix(all_wells, all_wells)
        np.fill_diagonal(dist_mat, np.inf)
        min_distance = np.min(dist_mat)

        total_resources = 0
        # Calculate the total resources on the new well
        for k in range(n_wells):
            # Round coordinates to nearest integer
            ix = int(round(new_wells[i][k][0]))
            iy = int(round(new_wells[i][k][1]))
            # Ensure indices are within the bounds of the array
            ix = max(0, min(ix, z_res.shape[0] - 1))
            iy = max(0, min(iy, z_res.shape[1] - 1))
            total_resources += z_res[ix, iy]

        # Objective is to maximize min_distance and total_resources
        j[i] = -min_distance - total_resources

    return j
```

[5]:
```
import pyswarms as ps

# Extract the existing well coordinates from the DataFrame
existing_wells = well_info[['X', 'Y']].values
num_new_wells = 5
# Define bounds for the problem based on the existing well locations
x_min, y_min = np.min(existing_wells, axis=0)
x_max, y_max = np.max(existing_wells, axis=0)
bounds = (np.array([x_min, y_min] * num_new_wells),
          np.array([x_max, y_max] * num_new_wells))

# Create an instance of the GlobalBestPSO
options = {'c1': 1.5, 'c2': 1.5, 'w': 0.7}
np.random.seed(123) # Set the random seed for numpy to ensure reproducibility
optimizer = ps.single.GlobalBestPSO(n_particles=50, dimensions=num_new_wells *␣
  ↪2, options=options, bounds=bounds)

# Run the optimizer
cost, pos = optimizer.optimize(objective_function, iters=1000,␣
  ↪existing_wells=existing_wells, z_res=z_res)
```

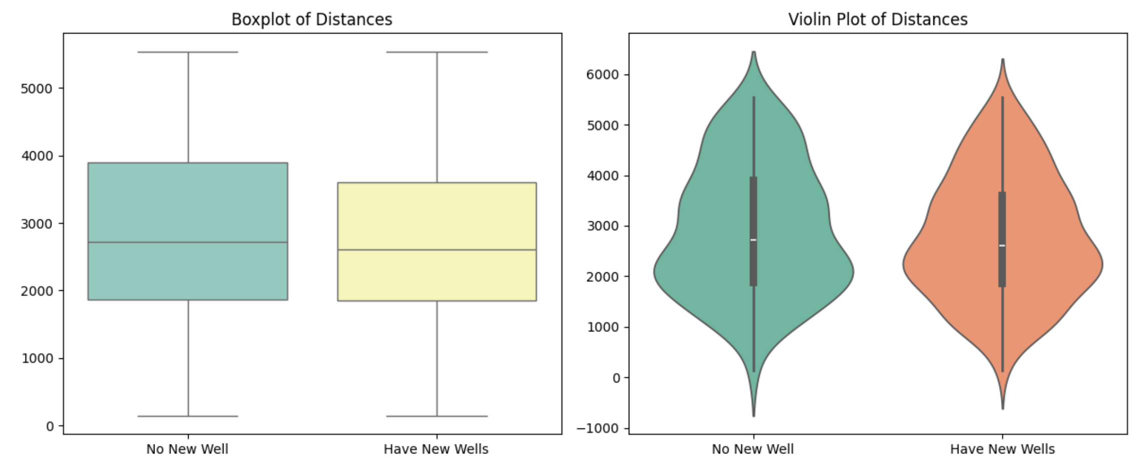
```
2024-05-12 20:23:15,490 - pyswarms.single.global_best - INFO - Optimize for 100
iters with {'c1': 1.5, 'c2': 1.5, 'w': 0.7}
pyswarms.single.global_best: 100%|      |100/100, best_cost=-479
2024-05-12 20:23:15,801 - pyswarms.single.global_best - INFO - Optimization
finished | best cost: -478.97077679169024, best pos: [34591.73390382
46587.43810279 35118.79143585 49144.40706739
 37814.44158456 48195.17018519 36564.43039165 47360.84080491
 37021.83093653 45562.80422665]
```

[6]:
```
existing_wells_df = pd.DataFrame(existing_wells, columns=['X', 'Y'])
existing_wells_df['Type'] = 'Existing'
new_wells_df = pd.DataFrame(pos.reshape(-1, 2), columns=['X', 'Y'])
new_wells_df['Type'] = 'New'

# Not consider the new wells that are too close to existing wells
```

```
dist_mat_existing = distance_matrix(existing_wells_df[['X', 'Y']].values,␣
↪existing_wells_df[['X', 'Y']].values)
np.fill_diagonal(dist_mat_existing, np.inf)

# Consider all wells
all_wells = pd.concat([existing_wells_df, new_wells_df])
dist_mat_all = distance_matrix(all_wells[['X', 'Y']].values, all_wells[['X',␣
↪'Y']].values)
np.fill_diagonal(dist_mat_all, np.inf)


plt.figure(figsize=(16, 6))

plt.figure(figsize=(12, 5))

plt.subplot(1, 2, 1)
sns.boxplot(data=[dist_mat_existing.flatten(), dist_mat_all.flatten()],␣
↪palette="Set3")
plt.xticks([0, 1], ['No New Well', 'Have New Wells'])
plt.title('Boxplot of Distances')

plt.subplot(1, 2, 2)
sns.violinplot(data=[dist_mat_existing.flatten(), dist_mat_all.flatten()],␣
↪palette="Set2")
plt.xticks([0, 1], ['No New Well', 'Have New Wells'])
plt.title('Violin Plot of Distances')

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

```
2024-05-12 20:23:15,834 - matplotlib.category - INFO - Using categorical units
to plot a list of strings that are all parsable as floats or dates. If these
strings should be plotted as numbers, cast to the appropriate data type before
plotting.
2024-05-12 20:23:15,841 - matplotlib.category - INFO - Using categorical units
to plot a list of strings that are all parsable as floats or dates. If these
strings should be plotted as numbers, cast to the appropriate data type before
plotting.
2024-05-12 20:23:15,878 - matplotlib.category - INFO - Using categorical units
to plot a list of strings that are all parsable as floats or dates. If these
strings should be plotted as numbers, cast to the appropriate data type before
plotting.
2024-05-12 20:23:15,883 - matplotlib.category - INFO - Using categorical units
to plot a list of strings that are all parsable as floats or dates. If these
strings should be plotted as numbers, cast to the appropriate data type before
plotting.
```

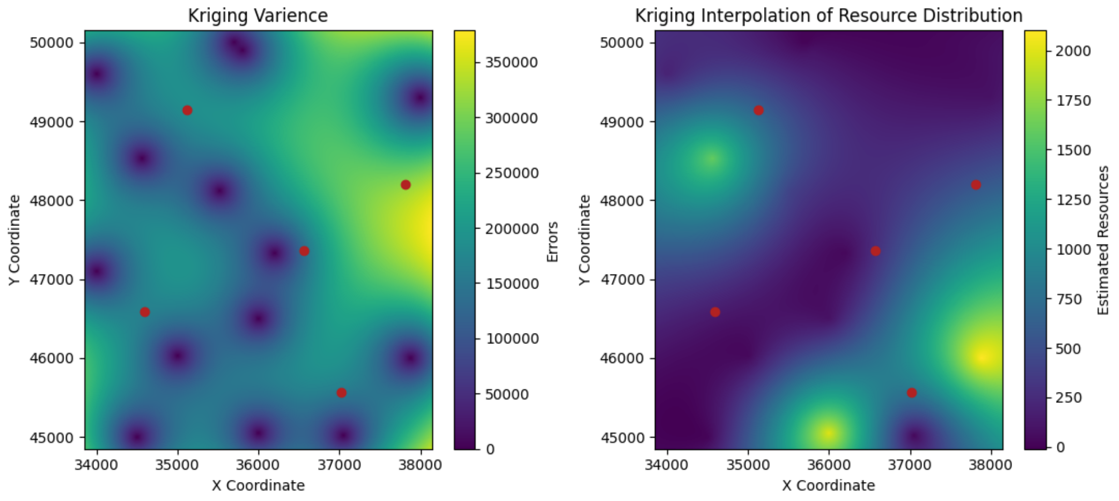
```
<Figure size 1600x600 with 0 Axes>
```



[7]:
```
# Plot the existing and new wells using plotly
import plotly.express as px


wells_df = pd.concat([existing_wells_df, new_wells_df])

fig = px.scatter(wells_df, x='X', y='Y', color='Type')
fig.update_layout(
    height=600,
    width=700,
)
fig.show()
```

[8]:
```
from matplotlib import pyplot as plt

# Create a figure with two subplots
fig, axs = plt.subplots(1, 2, figsize=(11, 5))  # 1 row, 2 columns

# Plot the first kriging result on the first subplot
c1 = axs[0].pcolormesh(grid_x, grid_y, ss_res, shading='auto', cmap='viridis')
fig.colorbar(c1, ax=axs[0], label='Errors')
axs[0].set_xlabel('X Coordinate')
axs[0].set_ylabel('Y Coordinate')
axs[0].set_title('Kriging Varience')
# Add the scatter plot of new wells
axs[0].scatter(pos[::2], pos[1::2], color='firebrick', label='New Wells')


# Plot the second kriging result on the second subplot
c2 = axs[1].pcolormesh(grid_x, grid_y, z_res, shading='auto', cmap='viridis')
fig.colorbar(c2, ax=axs[1], label='Estimated Resources')
```

```python
axs[1].set_xlabel('X Coordinate')
axs[1].set_ylabel('Y Coordinate')
axs[1].set_title('Kriging Interpolation of Resource Distribution')
# Add the scatter plot of new wells
axs[1].scatter(pos[::2], pos[1::2], color='firebrick', label='New Wells')

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


Kriging Varience
Y Coordinate
X Coordinate
Errors
Kriging Interpolation of Resource Distribution
Y Coordinate
X Coordinate
Estimated Resources