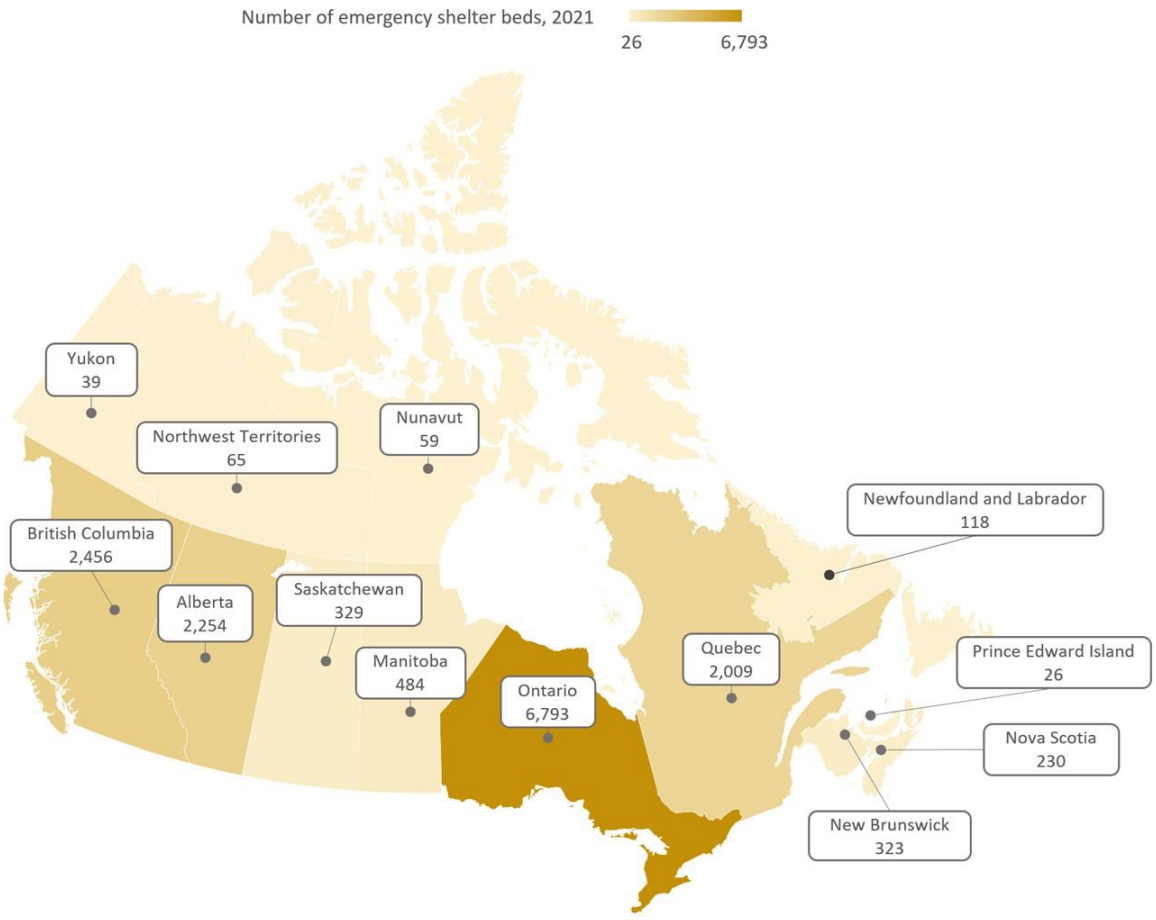
Data 608: Final Report

Alberta Shelter Capacity Website



Section 1: Problem Statement & Summary

Emergency shelters provide 24/7 accommodation for those experiencing homelessness. Understanding and analyzing emergency shelter capacity is crucial, particularly amidst the housing and economic challenges faced in Alberta.

According to the Government of Alberta, “Homelessness is often accompanied by other challenges, such as mental illness, addiction, a lack of resources or difficulties with life skills. Once a person has fallen into homelessness, these challenges can make it difficult to regain and maintain housing.” This further emphasizes the need to be able to better manage emergency shelters capacity within the province with the attempt on improving the lives of those affected by this crisis.

Our project aims to address this need by offering an online dashboard that includes the following components:

1. **Emergency Shelter Map:**

   We have developed an interactive map which visually represents the locations of emergency shelters across Alberta.

2. **Emergency Shelter Dashboard**

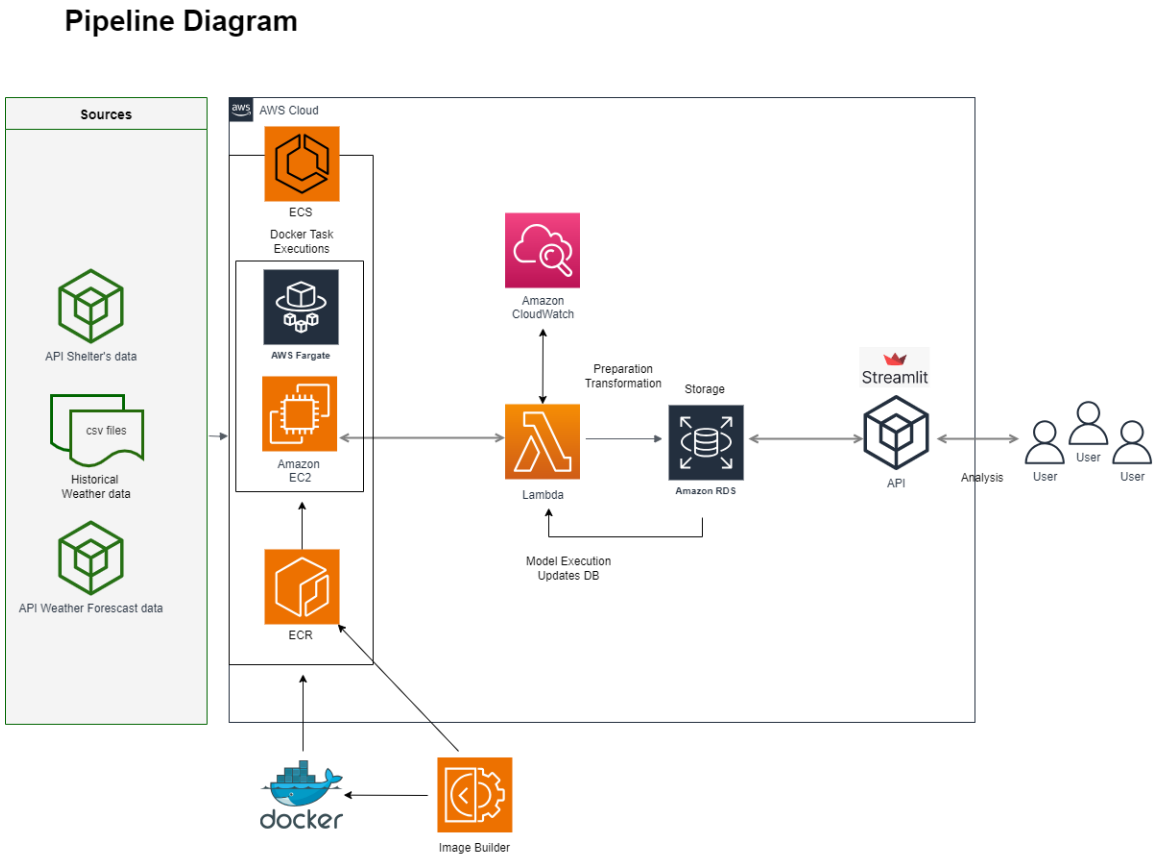
   Our dashboard provides detailed information on emergency shelters in Alberta. Users can access details such as occupancy rates by city and organization and average occupancy changes over time.

3. **Prediction for Daily Shelter Capacity**

   Using a logistic regression model, we developed a predictive model to estimate daily shelter capacity based on weather conditions. By analyzing historical data on shelter capacity and historical weather data, we were able to produce a basic model which takes the five-day forecast for different cities in Alberta and generates a prediction for shelter capacity for that specific region.

Section 2: Data Engineering Lifecycle

2.0: Pipeline Diagram



2.1: Data Generation

The emergency shelter data retrieved from OpenAlberta is provided by Seniors, Community and Social Services (SCSS) in a csv format. The dataset contains Date, Shelter Name, Total Approved Beds, Number of Beds, Emergency Response, and Daily Occupancy from 2013-2024. The data is updated monthly.

We retrieved weather data from two sources: the Natural Resource Canada website and the OpenWeather API. The Natural Resource Canada data provides daily minimum temperature and mean temperature for cities in Alberta. We used this as historical data to train the model. The OpenWeather API provides the five-day forecasts which contain temperature for every three hours daily from which we calculated as a daily average. These averages were used to populate our prediction model.

Weather Station IDs, crucial for historical data retrieval and model training, are acquired from Natural Resource Canada. This comprehensive approach ensures that we have access to up-to-date weather information necessary for accurate predictions while also leveraging historical data for model training and validation.

2.3: Data Ingestion

1. **Emergency Shelter Capacity Data:**

This data is ingested using the OpenAlberta API to obtain daily shelter occupancy rates along with organization names, shelter type, and city name.

2. **Weather Data**

This data was initially obtained through weather station IDs for all regions with emergency shelters in Alberta. Using these IDs, we obtain historical weather data from 2013 to 2023.

The Python scripts to execute the tasks above are scheduled through EC2 lambda functions.

2.3: Data storage

For data storage, we utilize Amazon Relational Database Service (RDS), a managed relational database service provided by AWS. RDS offers a scalable and reliable solution for storing structured data while providing features such as automated backups, high availability, and security enhancements.

We organized our data storage strategy based on the frequency of data updates and the specific requirements of our application. Below is a breakdown of our data storage tables and their respective update frequencies.

Frequency	**Description**	**Table**
Once	Initial Load	shelter_weather_hst city_weather_hst weather_plot_data_hst
Monthly	Monthly load of available current year data	shelter_weather_year city_weather_year weather_plot_data_year
Daily	Get weather forecast, train and execute model and load table with predictions	city_weather_prediction weather_plot_data

2.4: Data transformation

Our data transformation process involves several key steps to prepare the raw data for analysis and prediction. Below is an overview of the steps implemented for transformation in our data pipeline:

1. Identification of Weather Stations:

- We begin by identifying unique cities from our dataset
- For each city, we execute a code to retrieve all available weather station IDs associated with that city
- Among the obtained IDs, we isolate for the oldest station, which serves as the primary source for historical weather data retrieval within that specific region

2. **Temperature Data Retrieval:**

- Once weather station IDs are determined, we constructed URLs to access weather data for each station
- Using these URLs, we looped through the weather stations to extract daily minimum and average temperatures using python

3. Integration with Shelter Capacity Data:

- We executed a code to determine the capacity status of each shelter for a given day based on the overnight stay within a shelter and their respective capacity
- Based on the maximum capacity data, we classified the shelter capacity as either met or not met.

Throughout the transformation process, data IDs, weather stations, and relevant information are stored in an Amazon RDS. This data is accessed through SQL queries, facilitating efficient retrieval.

For prediction purposes, we aggregate shelter capacity status at the city level. It should be noted that if any shelter within a city reports capacity not met for a particular day, the entire city is classified as not meeting capacity. While effective for simplifying predictions, this approach may introduce limitations in capturing nuances at the individual shelter level.

2.5: Data serving

In the data serving phase of our project, we focus on providing users with seamless access to valuable insights and capacity predictions through a user-friendly web interface. Here is how we achieve this:

1. Containerization with Docker and Deployment:

- To streamline deployment and management, we utilized Docker to create a container to run our application. Docker Engine provided the runtime environment, while the Docker Image served as the template containing everything needed to run the application.
- This image was stored in Amazon Elastic Container Registry (ECR), enabling version control and easy distribution.
- We used Fargate for orchestrating and deploying the container in AWS. This allowed for simplified scaling and resource allocation based on demand.
- We configured the security groups associated with our Docker containers to allow outbound traffic on the appropriate ports for connecting to the RDS instance, and we configured the security group associated with the RDS instance to allow inbound traffic from the Docker containers' security group on the ports used by the database engine.

2. Streamlit Web Application:

- We leverage Streamlit, a Python library for creating interactive web applications, to develop our user interface.
- The website offers users easy access to valuable insights regarding shelter capacity in Alberta. Users can explore and visualize shelter occupancy trends and access predictive analytics on shelter capacity.

2.6: Data outputs: ML, Analytics

Our website opens with a map of emergency shelters in Alberta. The map is interactive and allows users to view all shelters across the province as pictured in Figure 1.

Figure 1: Map of Emergency Shelters in Alberta

Entering our second page, the dashboard, we can see a line plot of average occupancy rate over the years according to the city. The city can be changed in a dropdown menu. The graph for Calgary is pictured in Figure 2.

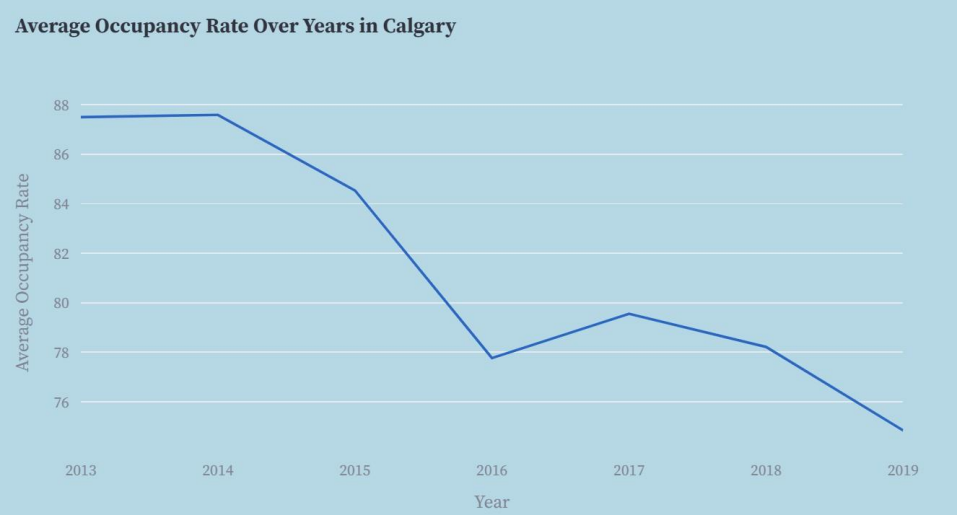


Figure 2: Average Occupancy Rate Over the Years in Calgary

Figure 3 displays the top 5 shelter organizations by number of shelters. This shows the top 5 across all shelters in Alberta and is not filtered by city.

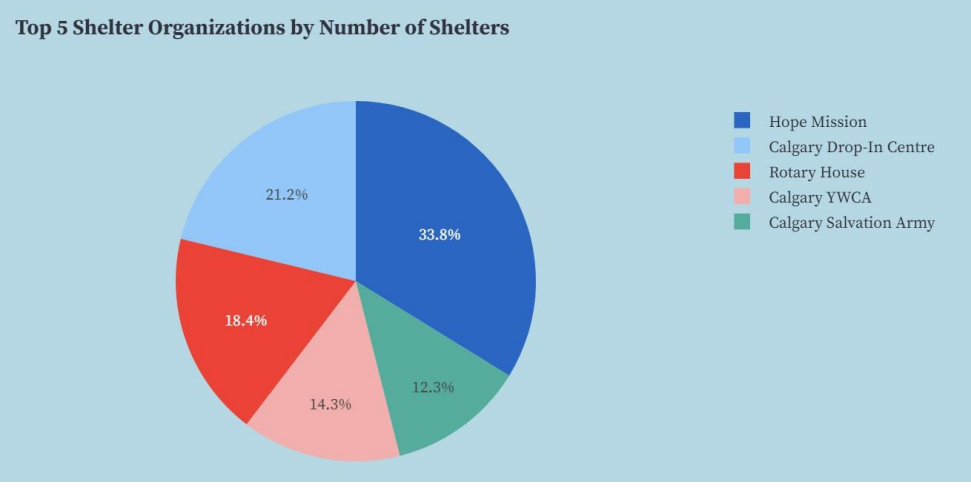


Figure 3: Top 5 Shelter Organizations by Number of Shelters

Figure 4 displays the frequency of shelter types across Alberta. We can see Short Term Supportive and Adult Emergency shelters are the most common.

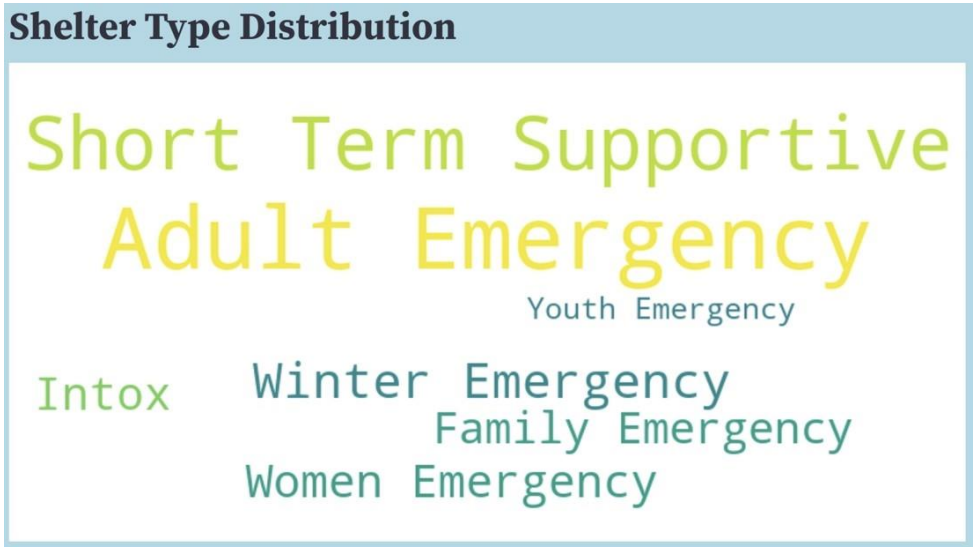


Figure 4: Distribution of Shelter Types

Our third page, which displays the predictions for shelter capacity, opens with Figure 5 which shows the proportion of shelters with capacity met to shelters with capacity not met for the data.

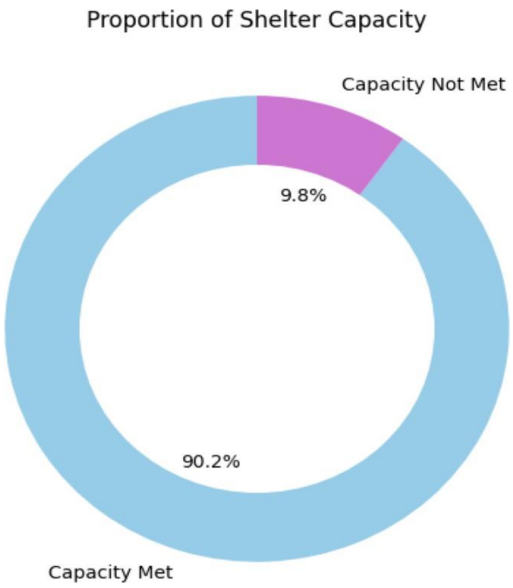


Figure 5: Proportion of Shelter Capacity

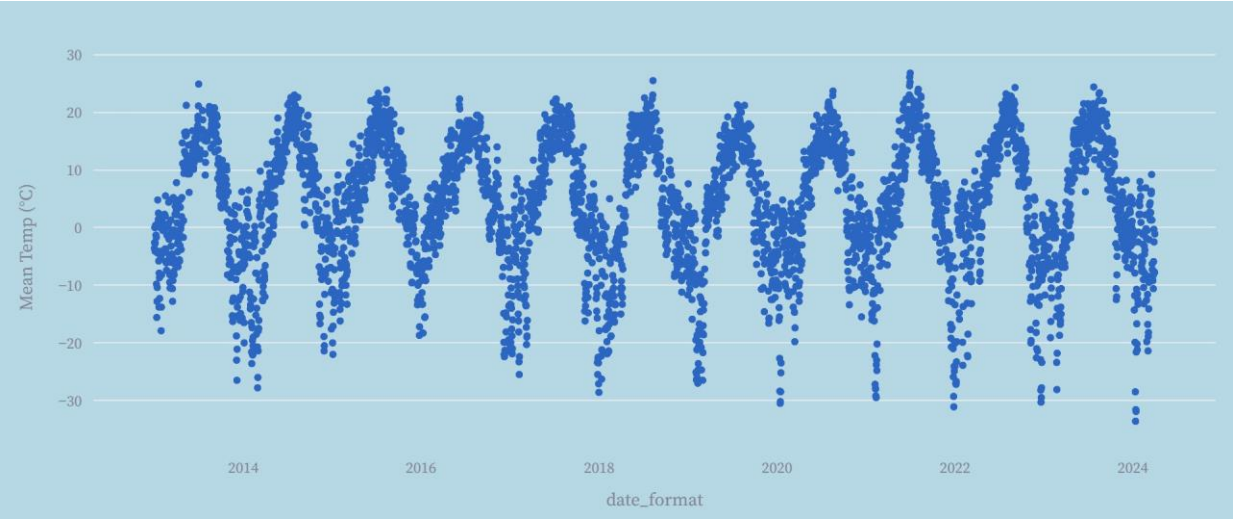


Figure 6: Mean Temperature over Time in Calgary

City Weather Prediction for shelters

	date	Temperature	City	Predicted_Status
6	2024-04-12	7.915	Edmonton	Capacity Met
7	2024-04-13	7.6713	Edmonton	Capacity Met
8	2024-04-14	8.5863	Edmonton	Capacity Met
9	2024-04-15	7.0025	Edmonton	Capacity Met
10	2024-04-16	0.8288	Edmonton	Capacity Not Met
11	2024-04-17	0.2775	Edmonton	Capacity Not Met

Figure 7: Shelter Capacity Predictions for Edmonton

The table above shows the capacity predictions for shelters in Edmonton, Predictions for Calgary, Fort McMurray, and Medicine Hat are also available on the application.

Section 3: Limitations and possible next steps

Our logistic regression model faces significant limitations. There are undoubtedly many underlying factors affecting shelter capacity which we cannot capture with the data available to us. Consequently, our classification error is quite high. We prioritized deploying the model regardless of performance to create a more interesting project we could learn more from.

In the future, it would be ideal to incorporate more data and experiment with more robust models, potentially exploring time series and forecasting models to produce more reliable predictions of shelter capacity.

Moreover, the weather data we utilized relies on a single weather station per region rather than a comprehensive collective average, potentially impacting the precision of our estimations. Additionally, it's noteworthy that Calgary consistently experiences at least one shelter operating beyond capacity.

Lastly, we would like to improve the speed of our application. Some ways we consider doing this is to experiment with applying Cython and parallel processing to allow for greater scalability.

Section 4: Conclusions

Throughout this project, our team gained invaluable insights and experiences in various aspects of cloud computing and data engineering. Here are some key learnings and reflections:

1. **Complexity of Cloud Computing:**

- We encountered the complexity of cloud computing while building and maintaining a stable data engineering pipeline. For example, we dedicated a significant amount of time verifying security configurations to enable communication between Docker and RDS.
- Organizing architecture and ensuring the stability and speed of the application presented challenges that required careful consideration. For example, we had to consider multiple storage techniques to run the application efficiently and different ways to host our online dashboard.

2. **Tools and Technologies:**

- We explored many tools and technologies available in the cloud computing ecosystem to facilitate our processes.
- From data ingestion to transformation and serving, we experimented with tools like Amazon RDS, Docker, and AWS Fargate, discovering their functionalities and advantages.

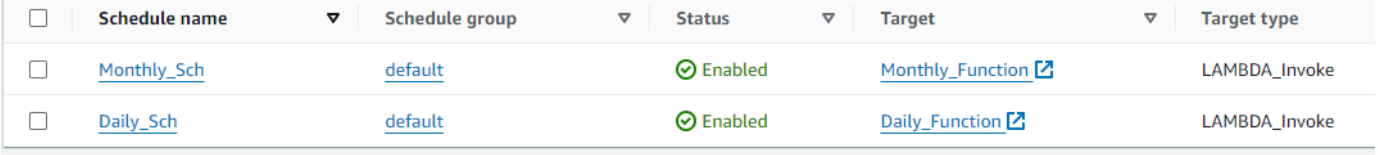
3. Linux Proficiency:

- Working within cloud environments necessitated a solid understanding of Linux systems and command-line operations.
- We enhanced our Linux proficiency through tasks such as managing EC2 instances, navigating file systems, and executing shell commands.

Overall, this project provided us with a deeper understanding of cloud computing principles, practical experience with AWS services, and useful lessons in teamwork and problem-solving.

Section 5: Code

Lambda functions (Schedule)



Schedule name	Schedule group	Status	Target	Target type
Monthly_Sch	default	Enabled	Monthly_Function	LAMBDA_Invoke
Daily_Sch	default	Enabled	Daily_Function	LAMBDA_Invoke

Connecting to RDS:

```
# Connect to RDS
conf ={
    'host':"databasebms.crukm2qo2466.us-east-1.rds.amazonaws.com",
    'port':5432,
    'database':"databasebms",
    'user':"postgres",
    'password':"d7t7b7s5"
}

engine = create_engine("postgresql://{user}:{password}@{host}:{port}/{user}".format(**conf))
```

Retrieving data:

```
sql="""
SELECT * FROM "shelter_weather_hst"
UNION ALL
SELECT * FROM "shelter_weather_year"
"""
complete_shelter_df = pd.read_sql_query(sql, engine)
```

Building the map page:

```
def map():

    st.caption("Emergency shelters provide 24/7 accommodation for those experiencing homelessne
                )
    st.divider()
    # hex_color = "#28558f"
    # st.markdown(
    #     "<h3 style='color: {hex_color};'>Emergency shelters provide 24/7 accommodation for th
    #     unsafe_allow_html=True)

    st.write("")
    st.write("")
    st.write("")
    st.write("")
    st.write("")
    st.write("")

    st.title("Map of Emergency Shelters in Alberta")

    embed_url = "https://www.google.com/maps/embed/v1/search?key=AIzaSyCcpO4gsqsJwtwGHytw-6wSiY

    st.markdown(f'<iframe src="{embed_url}" width="100%" height="600" frameborder="0" allowfull
                unsafe_allow_html=True)
```

Building the dashboard page:

```
def intro():
    df = pd.DataFrame(data)  # Assuming data is defined elsewhere

    st.title(':bar_chart: Emergency Shelters In Alberta')

    # Add dropdown filter for City
    cities = df['City'].unique()
    selected_city = st.selectbox('Select City for Shelter Capacities', cities)

    df['Capacity'] = pd.to_numeric(df['Capacity'], errors='coerce')

    capacity_average = df[df['City'] == selected_city].groupby(['ShelterType'])['Capacity'].mean().reset_index()
    fig = px.bar(capacity_average, x='ShelterType', y='Capacity',
                 title=f'Shelter Capacities for Different Shelter Types in {selected_city}',
                 labels={'Capacity': 'Average Capacity', 'ShelterType': 'ShelterType'})
    st.plotly_chart(fig)

    st.title(':chart_with_upwards_trend: Shelter Occupancy Rates Over Time')

    df['Date'] = pd.to_datetime(df['Date'])
    df['YEAR'] = df['Date'].dt.year
    df['MONTH'] = df['Date'].dt.month_name()
```

```
df['Capacity'] = pd.to_numeric(df['Capacity'], errors='coerce')
df['Overnight'] = pd.to_numeric(df['Overnight'], errors='coerce')
df['Occupancy Rate'] = df['Overnight'] / df['Capacity'] * 100

years = df['YEAR'].unique()
selected_year = st.selectbox('Select Year', years)

organizations = df['Organization'].unique()
selected_org = st.selectbox('Select Organization', organizations)

shelters = df[df['Organization'] == selected_org]['ShelterName'].unique()
selected_shelter = st.selectbox('Select Shelter', shelters)

df_filtered = df[(df['YEAR'] == selected_year) & (df['Organization'] == selected_org) & (
        df['ShelterName'] == selected_shelter)]

occupancy_rates_by_month = df_filtered.groupby('MONTH')['Occupancy Rate'].mean().reset_index()

fig = px.bar(occupancy_rates_by_month, x='MONTH', y='Occupancy Rate',
             title=f'Average Occupancy Rate of {selected_shelter} in {selected_org} for {selected_year}',
             labels={'Occupancy Rate': 'Average Occupancy Rate (%)', 'MONTH': 'Month'})
st.plotly_chart(fig)
```

```
city_data1 = df[df['City'] == selected_city1]

city_occupancy_over_time1 = city_data1.groupby('YEAR')['Occupancy Rate'].mean().reset_index()

st.title(f'Average Occupancy Rate Over Years in {selected_city1}')
fig = px.line(city_occupancy_over_time1, x='YEAR', y='Occupancy Rate',
              title=f'Average Occupancy Rate Over Years in {selected_city1}',
              labels={'Occupancy Rate': 'Average Occupancy Rate', 'YEAR': 'Year'})
st.plotly_chart(fig)

organization_distribution = df['Organization'].value_counts().reset_index()
organization_distribution.columns = ['Organization', 'Number of Shelters']

# Sort the DataFrame by 'Number of Shelters' in descending order
organization_distribution = organization_distribution.sort_values(by='Number of Shelters', ascending=False)

# Select the top 5 organizations
top_5_organizations = organization_distribution.head()

fig = px.pie(top_5_organizations, values='Number of Shelters', names='Organization',
             title='Top 5 Shelter Organizations by Number of Shelters')
st.plotly_chart(fig)

shelter_type_counts = df['ShelterType'].value_counts()

wordcloud = WordCloud(width=800, height=400, background_color='white').generate_from_frequencies(
    shelter_type_counts)

st.title('Shelter Type Distribution')
st.image(wordcloud.to_array(), use_column_width=True)
```

Building the weather prediction page:

```
def weather():
    st.title("Predicting Capacity")

    status_counts = complete_shelter_df['Status'].value_counts(normalize=True) * 100

    fig, ax = plt.subplots(figsize=(4, 3))
    ax.pie(status_counts, labels=status_counts.index, autopct='%1.1f%%', startangle=90,
           colors=['skyblue', 'orchid'])
    ax.add_artist(plt.Circle( xy: (0, 0), radius: 0.7, color='white'))  # Add white circle in the middle

    # Display plot in Streamlit
    st.pyplot(fig)

    sql="""
    SELECT * FROM "city_weather_prediction"
    """
    cw_df = pd.read_sql_query(sql, engine)

    sql="""
    SELECT * FROM "weather_plot_data_hst"
    UNION ALL
    SELECT * FROM "weather_plot_data_year"
    """
```

```
wplot_df = pd.read_sql_query(sql, engine)

result = cw_df['City'].drop_duplicates().tolist()

state = st.sidebar.radio("Pick The Region", options=result)



wplot_df_state = wplot_df[wplot_df['city'] == state]
wplot_df_state_filtered = wplot_df_state[['date_format', 'Mean Temp (°C)']]
#st.scatter_chart(data=wplot_df_state_filtered, x='date_format', y='Mean Temp (°C)')

# Create scatter plot
scatter_plot = px.scatter(wplot_df_state_filtered, x='date_format', y='Mean Temp (°C)')

# Display plot in Streamlit
st.plotly_chart(scatter_plot, use_container_width=True)


cw_df_state = cw_df[cw_df['City'] == state]
st.write("City Weather Prediction for shelters")
st.dataframe(cw_df_state, height=300)
```

Code for building logistic regression model for prediction:

```
sql="""
SELECT * FROM "shelter_weather_hst"
UNION ALL
SELECT * FROM "shelter_weather_year"
"""
complete_shelter_df = pd.read_sql_query(sql, engine)



s3 = boto3.client('s3')
bucket_name = 'bucketbms1'
csv_file_name = 'complete_city_df.csv'

complete_city_df.to_csv( path_or_buf: 'complete_city_df.csv', index=False)

s3.upload_file(Filename=csv_file_name, Bucket=bucket_name, Key=csv_file_name)




data = complete_city_df

x = data['Temperature']
y = data['Status']
city = data['city']
```

```
# Connect to RDS
conf ={
    'host':"databasebms.crukm2qo2466.us-east-1.rds.amazonaws.com",
    'port':5432,
    'database':"databasebms",
    'user':"postgres",
    'password':"d7t7b7s5"
}

engine = create_engine("postgresql://{user}:{password}@{host}:{port}/{user}".format(**conf))



sql="""
SELECT * FROM "city_weather_hst"
UNION ALL
SELECT * FROM "city_weather_year"
"""
complete_city_df = pd.read_sql_query(sql, engine)
```

```
accuracy = accuracy_score(y_te, y_te_pred)

print("Accuracy:", accuracy)

conf_matrix = confusion_matrix(y_te, y_te_pred, labels=['Capacity Met', 'Capacity Not Met'])
print("Confusion Matrix")
print(conf_matrix)

prediction_df = pd.DataFrame({
    'Temperature': data_df.loc[y_te.index, 'Temperature'],
    'City': data_df.loc[y_te.index, 'City'],
    'Predicted_Status': y_te_pred
})


current_year = complete_shelter_df['Year'].max()
last_years_data = complete_shelter_df[complete_shelter_df['Year'].isin(range(current_year - 2, current_year + 1))]
unique_data_df = last_years_data[['city']].drop_duplicates()
print("Cities with shelters")
print(unique_data_df)
```

```python
def get_weather_forecast(latitude, longitude, api_key):
    url = f"http://api.openweathermap.org/data/2.5/forecast?lat={latitude}&lon={longitude}&appid={api_key}"
    response = requests.get(url)
    data = response.json()
    return data

def parse_weather_data(weather_data, city):
    parsed_data = []
    for forecast in weather_data.get('list', []):
        date = forecast.get('dt_txt', '')
        temperature = forecast.get('main', {}).get('temp', '')
        unit = "K"  # Temperature unit is Kelvin by default
        if 'units' in forecast.get('main', {}):
            unit = forecast['main']['units']
        parsed_data.append({'City': city, 'Date': date, 'Temperature': temperature, 'Unit': unit})
    return parsed_data
```

```python
def main():
    api_key = "dec12609e3de3aaabb2d8fdb1714d4c3"

    cities = {
    'Calgary': {'latitude': 51.0447, 'longitude': -114.0719},
    'Edmonton': {'latitude': 53.5444, 'longitude': -113.4909},
    'Lethbridge': {'latitude': 49.6950, 'longitude': -112.8430},
    'Fort McMurray': {'latitude': 56.7268, 'longitude': -111.3843},
    'Medicine Hat': {'latitude': 50.0400, 'longitude': -110.6768},
    'Lloydminster': {'latitude': 53.2761, 'longitude': -110.0030}
}

    all_weather_data = []
    for city, coords in cities.items():
        weather_data = get_weather_forecast(coords['latitude'], coords['longitude'], api_key)
        parsed_data = parse_weather_data(weather_data, city)
        all_weather_data.extend(parsed_data)

    df = pd.DataFrame(all_weather_data)
    return df
```

```python
if __name__ == "__main__":
    df_wforecast = main()

df_wforecast['Date'] = pd.to_datetime(df_wforecast['Date'])
df_wforecast['Date'] = df_wforecast['Date'].dt.date
df_wforecast['Temperature_Celsius'] = df_wforecast['Temperature'] - 273.15
average_temperatures = df_wforecast.groupby(['City', 'Date']).agg({'Temperature_Celsius': 'mean'}).reset_index()

average_temperatures.rename(columns={'City': 'city'}, inplace=True)
average_temperatures.rename(columns={'Date': 'date'}, inplace=True)
average_temperatures.rename(columns={'Temperature_Celsius': 'Temperature'}, inplace=True)

weather_forecast = pd.merge(unique_data_df,average_temperatures, left_on='city', right_on='city')

print("Weather Forecast by Date and City")
print(weather_forecast.head(5))
```

```python
#Production dataframe
production_x = weather_forecast['Temperature']
production_city = weather_forecast['city']

production_data_df = pd.DataFrame({
    'Temperature': production_x,
    'City': production_city
})

encoded_features_production = onehot_encoder.transform(production_data_df[['City']])

X_production = np.hstack((production_data_df[['Temperature']], encoded_features_production.toarray()))

X_scaled_production = scaler.transform(X_production)

production_y_pred = model.predict(X_scaled_production)

production_prediction_df = pd.DataFrame({
    'Temperature': production_data_df['Temperature'],
    'City': production_data_df['City'],
    'Predicted_Status': production_y_pred
})
```

```python
print("Production Prediction:")
print(production_prediction_df.head(5))

df_city_weather_prediction = pd.merge(weather_forecast, production_prediction_df, left_on=['city', 'Temperature'], right_on=['City', 'Temperature'])

df_city_weather_prediction.drop(columns=['city'], inplace=True)

df_city_weather_prediction.to_sql( name: 'city_weather_prediction', engine, index=False, if_exists='replace')



sql="""
SELECT * FROM "city_weather_prediction"
"""
cw_df = pd.read_sql_query(sql, engine)

print("Table created: city_weather_prediction")

print(cw_df.head(10))
```

We relied on changing elements of our architecture to speed up our application. We opted to use RDS for storing our data instead of DynamoDB or S3 bucket storage as originally planned. RDS provides a structured and relational database environment, allowing us to efficiently query different tables for analysis.

By leveraging RDS, we improved the efficiency of data retrieval and processing, resulting in faster response times and smoother user experiences.

Project link: *http://54.87.203.4C:8501/*

References:

- Canada, I. (2023, March 10). *Infrastructure Canada - Shelter Capacity Report 2021*. Secure.infc.gc.ca.https://secure.infc.gc.ca/homelessness-sans-abri/reports-rapports/shelter-cap-hebergement-2021-eng.html#h2.09
- *Alberta's homelessness response | Alberta.ca*. (n.d.). www.alberta.ca. https://www.alberta.ca/homelessness-response
- *Index of /climate/observations/daily/csv*. (n.d.). Dd.weather.gc.ca. Retrieved April 11, 2024, from https://dd.weather.gc.ca/climate/observations/daily/csv/