CS156 - Pipeline - First Draft

October 21, 2024

1 Data Description

1.1 Data Overview

The dataset used for this project was extracted from my personal **BeReal digital archive**. BeReal is a social media platform that encourages users to share candid, unfiltered moments by taking and posting photos from both their front and back phone cameras at a randomly prompted time each day. The data contains a collection of these moments, referred to as "memories," which include both visual and metadata elements.

1.2 Data Acquisition

The data was obtained through the **BeReal GDPR data request** process, which allows users to download all of their personal data stored by the platform. Upon requesting the data, BeReal provided a ZIP file containing several **JSON files** with information such as: - **Photos** (front and back images) - **Location data** (latitude, longitude) - **Timestamps** (when the photos were taken) - **Other metadata** (comments, reactions, user details)

For the purposes of this project, I focused on the **memories.json** file, which contains metadata related to my BeReal posts. The dataset includes a total of **955 memories**, each consisting of: - Paths to the **front and back images** associated with each memory - The **timestamp** indicating when each memory was captured - **Location data** for posts where the app recorded latitude and longitude - Whether the post was uploaded on time or late - Any reactions or interactions from other users (not used in this analysis)

1.3 Sampling Strategy

The data used in this project covers my BeReal posts from the start of my account usage to the present, providing a complete set of moments captured over time. No additional sampling was necessary as the dataset was already constrained to my personal data archive.

The goal of this project is to analyze and model patterns from these posts, particularly focusing on whether I was with other people or alone at the time of each memory. The dataset was labeled with this information based on a manual inspection of each memory's front and back images, assigning a binary label (with_people or without_people) to each post.

2 Loading Data

In this section, the raw data from my **BeReal archive** is converted into a Python-readable format and loaded into a **pandas DataFrame** for further analysis. The primary dataset, memories.json, contains metadata about each memory, including image paths, timestamps, and location data.

The following steps were taken to load the data: 1. **Reading the JSON File**: The memories.json file was read using Python's json module and then loaded into a pandas DataFrame for ease of manipulation and analysis. 2. **Image Paths**: The frontImage and backImage fields in the dataset contain the paths to the images associated with each memory. These paths were extracted and appended to the DataFrame for further processing. 3. **Timestamps and Location**: The timestamps (date and time of each post) and location data (latitude and longitude) were also extracted and added as new columns in the DataFrame. 4. **Handling Missing Data**: For some memories, location data was not recorded, so these values were set to NaN to handle them appropriately during analysis.

The loaded dataset is now structured in a pandas DataFrame, which will allow us to perform the necessary pre-processing, exploratory data analysis, and modeling in the upcoming sections.

[1]:
```
import pandas as pd
import json

# Load the JSON file into a Python dictionary
with open('/Users/minjaekim/Projects/bereal-ml-project/data/
 ↪iVkH1ljxruQMdiX6bixQvMbcJBu2-14RWaeP4zbC-mXBQf8Up7/memories.json') as f:
    memories_data = json.load(f)

# Create a list to store the relevant data for each memory
memories_list = []

# Loop through each memory and extract the relevant data
for memory in memories_data:
    front_image = memory['frontImage']['path']
    back_image = memory['backImage']['path']
    is_late = memory['isLate']
    date = memory['date']
    taken_time = memory['takenTime']
    bereal_moment = memory['berealMoment']
    # Use .get() to safely access the 'location' field, return None if not␣
 ↪present
    location = memory.get('location', None)

    if location:
        latitude = location.get('latitude', None)
        longitude = location.get('longitude', None)
    else:
        latitude, longitude = None, None  # No location available

    # Add to the list as a dictionary
    memories_list.append({
        'front_image': front_image,
        'back_image': back_image,
        'is_late': is_late,
        'date': date,
        'taken_time': taken_time,
        'bereal_moment': bereal_moment,
        'latitude': latitude,
        'longitude': longitude
    })

# Convert the list of memories to a Pandas DataFrame
memories_df = pd.DataFrame(memories_list)

# Display the first few rows of the DataFrame
memories_df.head()
```

```
[1]:                                          front_image  \
0  /Photos/iVkH1ljxruQMdiX6bixQvMbcJBu2/post/UWat…
1  /Photos/iVkH1ljxruQMdiX6bixQvMbcJBu2/post/g6iG…
2  /Photos/iVkH1ljxruQMdiX6bixQvMbcJBu2/post/FddJ…
3  /Photos/iVkH1ljxruQMdiX6bixQvMbcJBu2/post/aolL…
4  /Photos/iVkH1ljxruQMdiX6bixQvMbcJBu2/post/_F6f…

                                           back_image  is_late  \
0  /Photos/iVkH1ljxruQMdiX6bixQvMbcJBu2/post/2ESG…    False
1  /Photos/iVkH1ljxruQMdiX6bixQvMbcJBu2/post/ITny…     True
2  /Photos/iVkH1ljxruQMdiX6bixQvMbcJBu2/post/H0_D…    False
3  /Photos/iVkH1ljxruQMdiX6bixQvMbcJBu2/post/x6c-…     True
4  /Photos/iVkH1ljxruQMdiX6bixQvMbcJBu2/post/0fxu…    False

                       date                taken_time  \
0  2024-10-08T00:00:00.000Z  2024-10-08T17:48:30.362Z
1  2024-10-07T00:00:00.000Z  2024-10-08T02:08:28.988Z
2  2024-10-07T00:00:00.000Z  2024-10-08T02:18:18.197Z
3  2024-10-06T00:00:00.000Z  2024-10-07T00:43:21.826Z
4  2024-10-06T00:00:00.000Z  2024-10-07T10:26:33.769Z

              bereal_moment   latitude   longitude
0  2024-10-08T17:48:05.346Z  37.790025 -122.401512
1  2024-10-08T00:32:05.356Z  37.783693 -122.409564
2  2024-10-08T00:32:05.356Z  37.784089 -122.409181
3  2024-10-06T20:51:05.354Z  37.784088 -122.409180
4  2024-10-06T20:51:05.354Z  37.784089 -122.409181
```

3 Data Wrangling

This section details the steps taken to clean, preprocess, and engineer features from the raw BeReal dataset. The steps are organized by topic for clarity, showing how each aspect of the data was handled. Additionally, the dataset was explored through visualizations and descriptive statistics to better understand its structure.

3.1 Initial Dataset Overview

Before proceeding with feature engineering and model building, the dataset was explored using: - .info(): To inspect column data types and check for missing values. - .isnull().sum(): To quantify missing values, especially in the location data (latitude and longitude). - Percentage of Missing Data: Calculated the percentage of missing data in each column to better assess its scale and impact on the analysis.

[2]:
```python
print(memories_df.info())

print("\nNumber of values missing for each column:")
print(memories_df.isnull().sum())  # Check for missing values

# Calculate the percentage of missing data in each column
print("\nPercentage of missing data in each column:")
missing_percentage = (memories_df.isnull().sum() / len(memories_df)) * 100
print(missing_percentage[missing_percentage > 0])
```

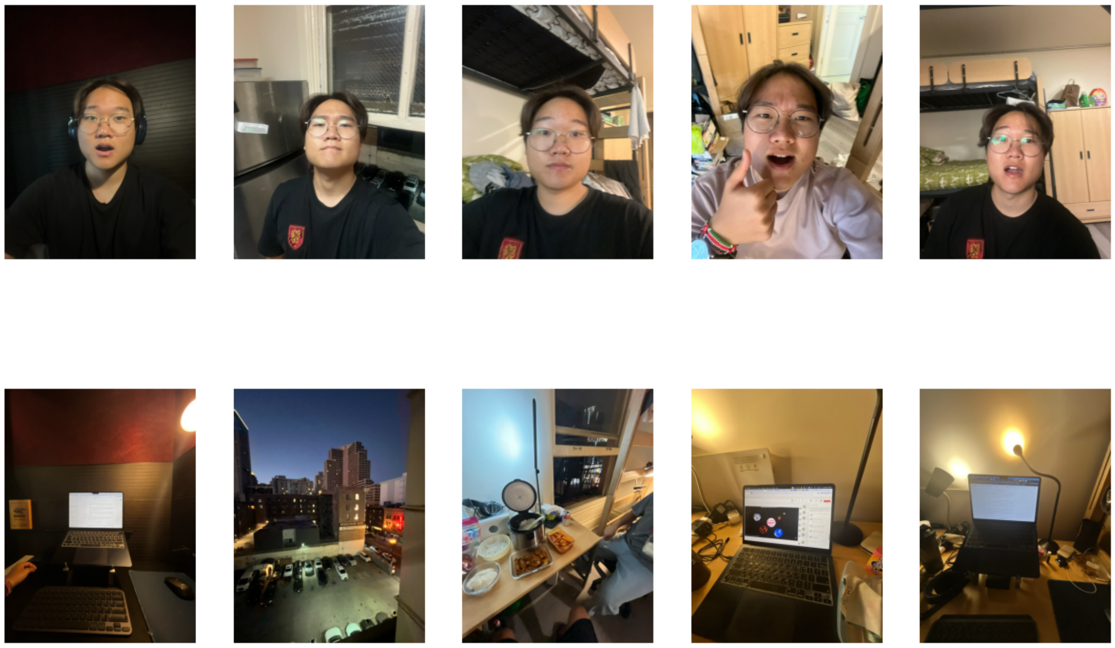
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 955 entries, 0 to 954
Data columns (total 8 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   front_image    955 non-null    object
 1   back_image     955 non-null    object
 2   is_late        955 non-null    bool
 3   date           955 non-null    object
 4   taken_time     955 non-null    object
 5   bereal_moment  955 non-null    object
 6   latitude       809 non-null    float64
 7   longitude      809 non-null    float64
dtypes: bool(1), float64(2), object(5)
memory usage: 53.3+ KB
None

Number of values missing for each column:
front_image        0
back_image         0
is_late            0
date               0
taken_time         0
bereal_moment      0
latitude         146
longitude        146
dtype: int64
```

```
Percentage of missing data in each column:
latitude     15.287958
longitude    15.287958
dtype: float64
```

3.2 Handling Image Data

The raw dataset included image paths that did not match the structure of the provided zip file. To correct this, the image paths were adjusted to match the actual folder structure. Afterward, the first five images were displayed to verify that the paths were correct.

[3]:
```
from PIL import Image
import matplotlib.pyplot as plt

# Function to display images
def display_images(image_paths):
    fig, axes = plt.subplots(1, len(image_paths), figsize=(15, 5))
    for ax, image_path in zip(axes, image_paths):
        img = Image.open(image_path)  # Open the image from the local file
        ax.imshow(img)  # Display the image
        ax.axis('off')  # Remove axes
    plt.show()

# Correct the image paths in the DataFrame by replacing '/Photos/' with 'data/
 ↪Photos/'
memories_df['front_image'] = memories_df['front_image'].apply(lambda x: x.
 ↪replace('/Photos/iVkH1ljxruQMdiX6bixQvMbcJBu2', '/Users/minjaekim/Projects/
 ↪bereal-ml-project/data/iVkH1ljxruQMdiX6bixQvMbcJBu2-14RWaeP4zbC-mXBQf8Up7/
 ↪Photos'))
memories_df['back_image'] = memories_df['back_image'].apply(lambda x: x.
 ↪replace('/Photos/iVkH1ljxruQMdiX6bixQvMbcJBu2', '/Users/minjaekim/Projects/
 ↪bereal-ml-project/data/iVkH1ljxruQMdiX6bixQvMbcJBu2-14RWaeP4zbC-mXBQf8Up7/
 ↪Photos'))

# Get the first 5 front images from the DataFrame
front_image_paths = memories_df['front_image'].head(5)
back_image_paths = memories_df['back_image'].head(5)

# Display the first 5 front images
display_images(front_image_paths)
display_images(back_image_paths)
```



3.3 Location-Based Analysis

Using Folium, the geographical locations of memories were plotted on a map to visualize the spatial distribution. Additionally, missing location data were removed to ensure the visualization works as expected.

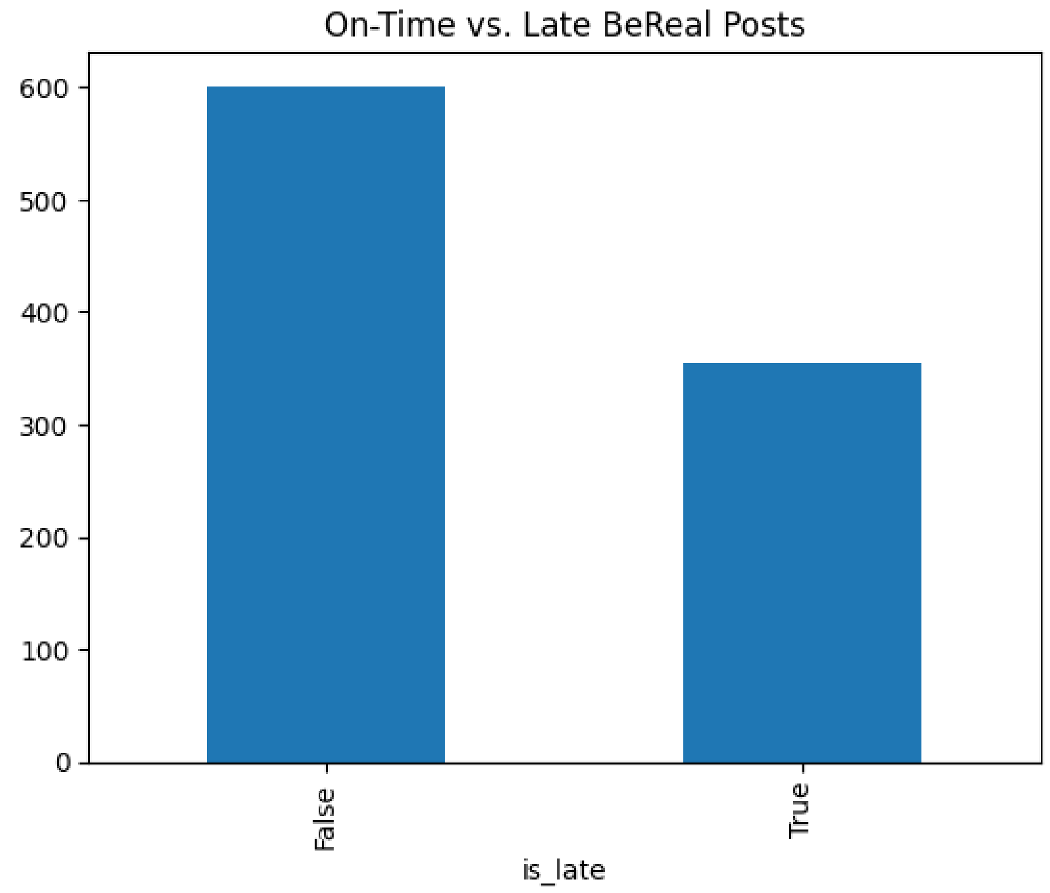
[4]:
```python
import folium

# Initialize a map at a central location
m = folium.Map(location=[memories_df['latitude'].mean(),␣
 ↪memories_df['longitude'].mean()], zoom_start=5)

# Add markers for each BeReal post
for i, row in memories_df.iterrows():
    if not pd.isnull(row['latitude']) and not pd.isnull(row['longitude']):
        folium.Marker([row['latitude'], row['longitude']],␣
 ↪popup=row['bereal_moment']).add_to(m)

# Show the map
m
```

[4]:
```
<folium.folium.Map at 0x7fe751566c70>
```

3.4 On-Time vs. Late Posts

A bar chart was plotted to compare the number of on-time vs. late posts, providing insights into the distribution of posts relative to the platform's daily prompt.

[5]:
```
# Plot the count of 'on-time' vs 'late' posts
memories_df['is_late'].value_counts().plot(kind='bar', title='On-Time vs. Late␣
 ↪BeReal Posts')
plt.show()
```



3.5 Time-Based Feature Engineering

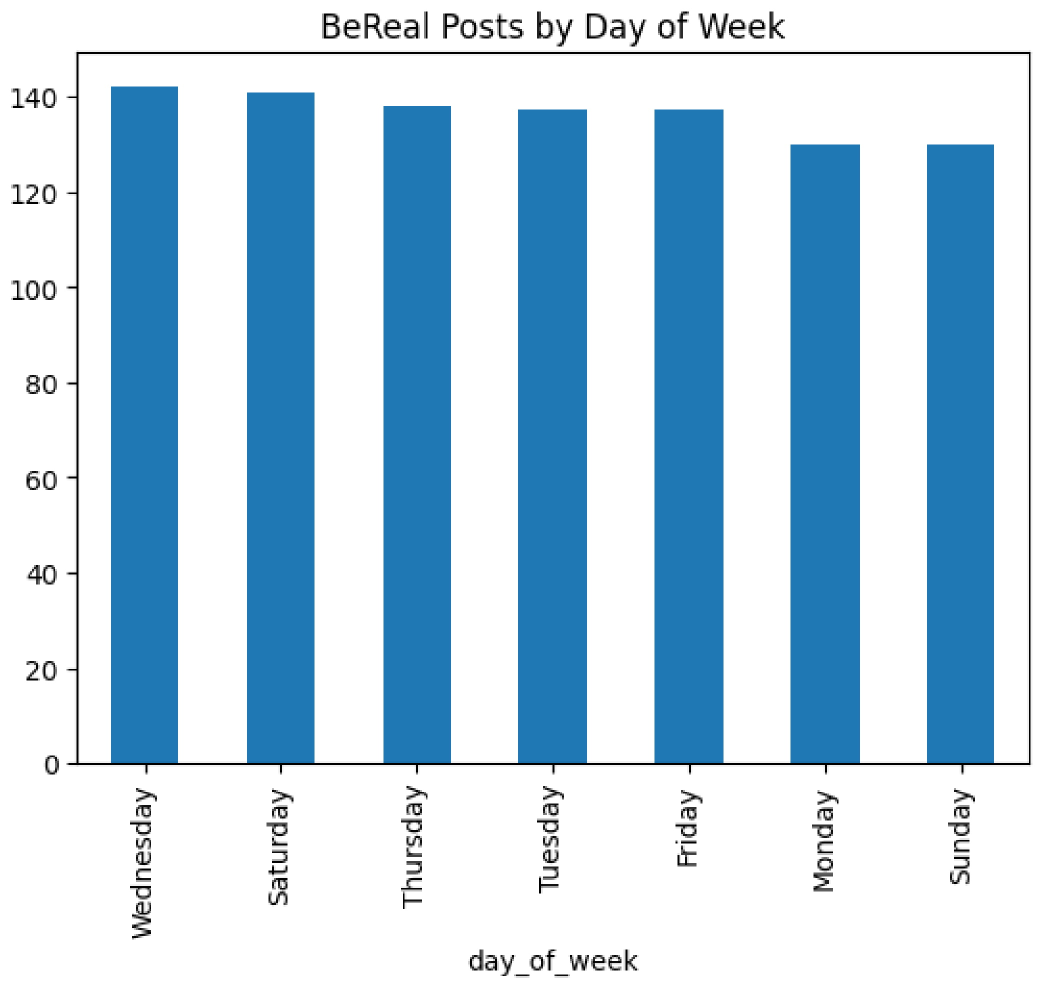
To prepare for further time-based analysis, two new features were engineered: - day_of_week: Extracted from the takenTime column to indicate the day of the week when the memory was captured. - hour_of_day: Created to represent the hour of the day when the memory was posted.

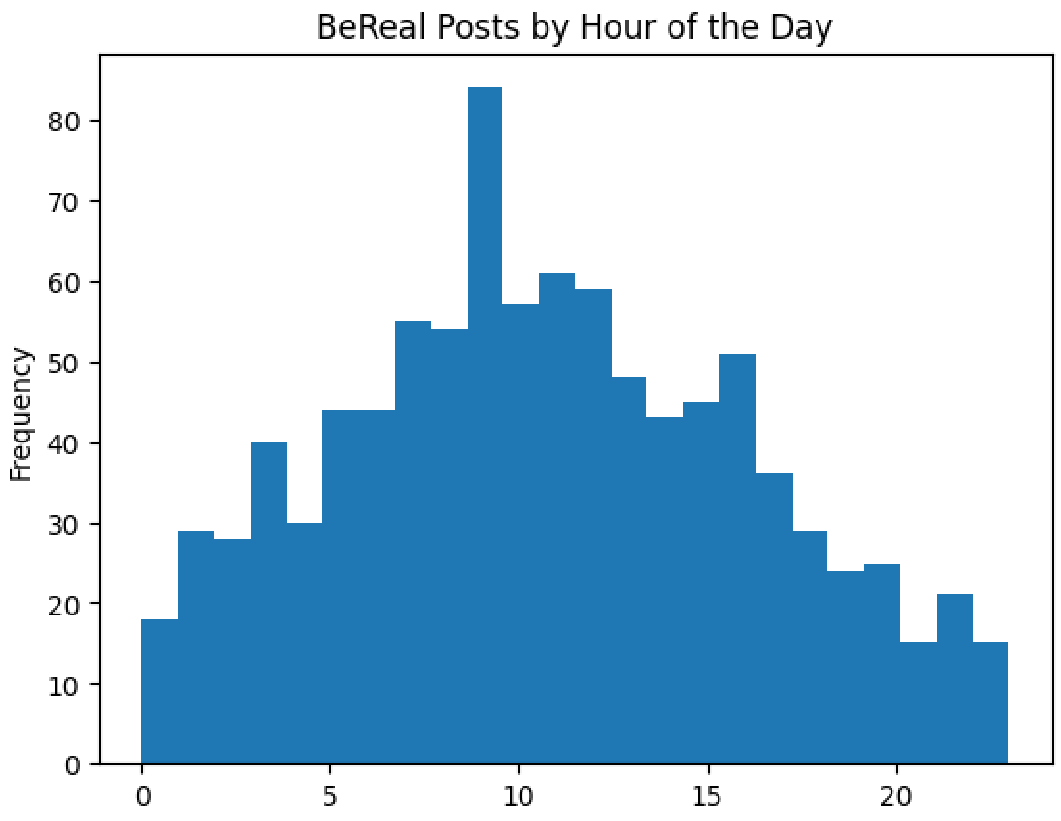
The following visualizations were created: - Day of the Week Distribution: A bar plot showing the number of memories posted on each day of the week. - Hour of the Day Distribution: A histogram showing the distribution of posts across different times of the day.

[6]:
```
# Convert the 'takenTime' column to datetime
memories_df['taken_time'] = pd.to_datetime(memories_df['taken_time'])

# Now you can access .dt attributes for time-based exploration
memories_df['day_of_week'] = memories_df['taken_time'].dt.day_name()
memories_df['hour_of_day'] = memories_df['taken_time'].dt.hour

# Display the new columns to check
print(memories_df[['taken_time', 'day_of_week', 'hour_of_day']].tail())
```

```
                          taken_time day_of_week  hour_of_day
950 2022-08-03 17:48:52.664000+00:00   Wednesday           17
951 2022-08-02 12:33:01.780000+00:00     Tuesday           12
952 2022-08-01 22:03:56.966000+00:00      Monday           22
953 2022-07-31 14:55:15.583000+00:00      Sunday           14
954 2022-07-27 18:43:39.214000+00:00   Wednesday           18
```

[7]:
```
# Plot distribution of BeReal moments by day of the week
memories_df['day_of_week'].value_counts().plot(kind='bar', title='BeReal Posts␣
 ↪by Day of Week')
plt.show()

# Plot distribution by hour of the day
memories_df['hour_of_day'].plot(kind='hist', bins=24, title='BeReal Posts by␣
 ↪Hour of the Day')
plt.show()
```


BeReal Posts by Day of Week
140
120
100
80
60
40
20
0
Wednesday
Saturday
Thursday
Tuesday
Friday
Monday
Sunday
day_of_week



3.6 Labeling the Data

Each memory was manually labeled with a new binary feature, is_with_people, indicating whether other people were present in the post. This labeling was performed through manual inspection of both the front and back images associated with each memory. People that were not associated with me or simply on screens were not counted. The label is_with_people was marked as true only when people were physically present in person and associated with me. People visible on screens or those not personally known were not counted.

[8]:
```
import pandas as pd
import ipywidgets as widgets
from IPython.display import display, clear_output

# Assuming you have a DataFrame named 'memories_df'
# We will add a new column 'is_with_people' that you'll manually label
memories_df['is_with_people'] = None  # Placeholder for manual labeling

# Create widgets for labeling
label_widget = widgets.ToggleButtons(
    options=[True, False],
    description='With People:'
)

save_button = widgets.Button(description="Save Label")
image_index = 0

# Display the first image or relevant metadata
img_display = widgets.Output()

def show_memory(index):
    with img_display:
        clear_output(wait=True)
        print(f"Memory {index+1} out of {len(memories_df)}")
        print(f"Front Image Path: {memories_df.loc[index, 'front_image']}")
        print(f"Back Image Path: {memories_df.loc[index, 'back_image']}")
        print(f"Location: {memories_df.loc[index, 'latitude']}, {memories_df.
 ↪loc[index, 'longitude']}")
        print(f"Date/Time: {memories_df.loc[index, 'taken_time']}")
        # You can also display the images here using libraries like PIL or␣
 ↪OpenCV, if needed.
        # Get the paths for the front and back images
        front_image_path = memories_df.loc[index, 'front_image']
        back_image_path = memories_df.loc[index, 'back_image']

        # Open the images using PIL
        img1 = Image.open(front_image_path)
        img2 = Image.open(back_image_path)

        # Display images side by side
        fig, axs = plt.subplots(1, 2, figsize=(10, 5))  # Create 1 row, 2␣
 ↪columns
        axs[0].imshow(img1)
        axs[0].axis('off')  # Hide axis
        axs[0].set_title('Front Image')

        axs[1].imshow(img2)
        axs[1].axis('off')  # Hide axis
        axs[1].set_title('Back Image')

        plt.show()

# Function to save the label for the current memory
def save_label(b):
    global image_index
    # Save the label to the DataFrame
    memories_df.loc[image_index, 'is_with_people'] = label_widget.value

    # Move to the next image
    image_index += 1
    if image_index < len(memories_df):
        show_memory(image_index)  # Show the next memory
    else:
        with img_display:
            clear_output()
            print("All memories labeled!")

# Display the widgets
save_button.on_click(save_label)
display(img_display)
show_memory(image_index)
display(label_widget, save_button)
```

```
Output()
```

```
ToggleButtons(description='With People:', options=(True, False), value=True)
```

```
Button(description='Save Label', style=ButtonStyle())
```

[9]:
```
# Extracting labeled data to labeled_memories.csv
# memories_df.to_csv('labeled_memories.csv', index=False)
# Commented out because this only needs to be done once right after the first␣
 ↪time the labeling is completed.

# Loading previously labeled memories data
labeled_memories_df = pd.read_csv("/Users/minjaekim/Projects/bereal-ml-project/
 ↪data/iVkH1ljxruQMdiX6bixQvMbcJBu2-14RWaeP4zbC-mXBQf8Up7/labeled_memories.
 ↪csv")
```

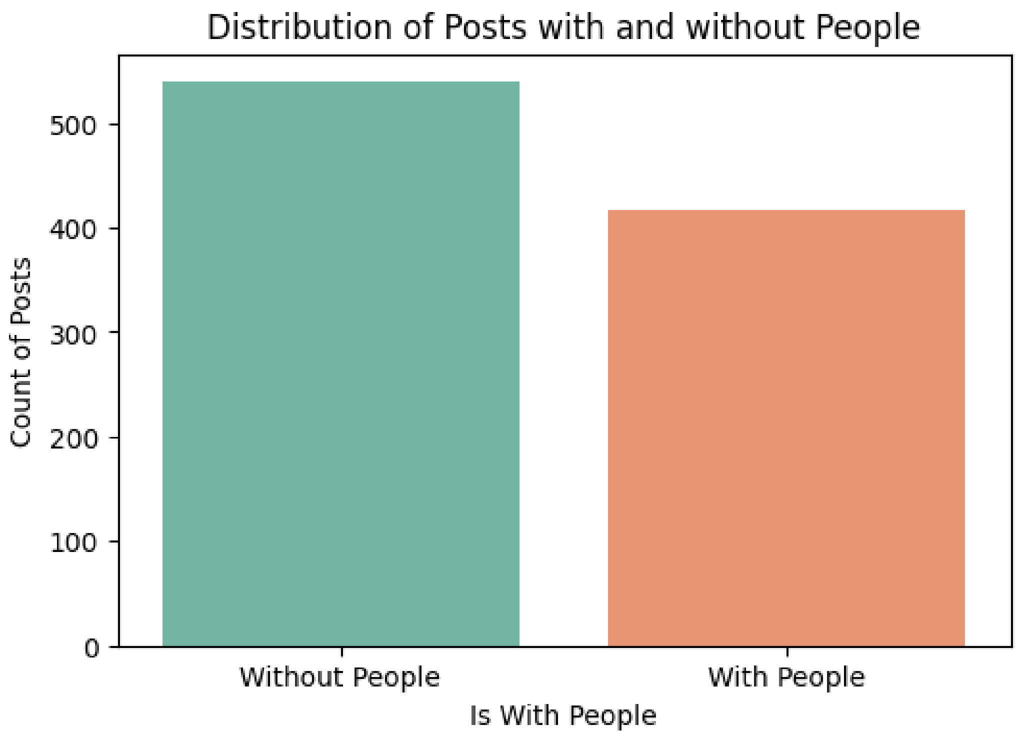
3.7 Analysis of Labeled Data

After labeling the data, additional analysis was conducted to understand the distribution of posts labeled as with_people or without_people and their relationships with other features: - Distribution of Posts: A bar chart was created to show the count of posts labeled as "with people" and "without people." - Cross-Analysis: The relationship between the is_with_people label and features like time of day and location was explored, looking for patterns in social behavior. - Correlation Matrix: A heatmap was plotted to examine the correlations between the features and the is_with_people label to identify potential predictive relationships.

3.7.1 Distribution of Posts

[10]:
```
import seaborn as sns

# Count of posts labeled as "with people" and "without people"
plt.figure(figsize=(6, 4))
sns.countplot(x='is_with_people', data=labeled_memories_df, palette='Set2')
plt.title('Distribution of Posts with and without People')
plt.xlabel('Is With People')
plt.ylabel('Count of Posts')
plt.xticks([0, 1], ['Without People', 'With People'])
plt.show()
```

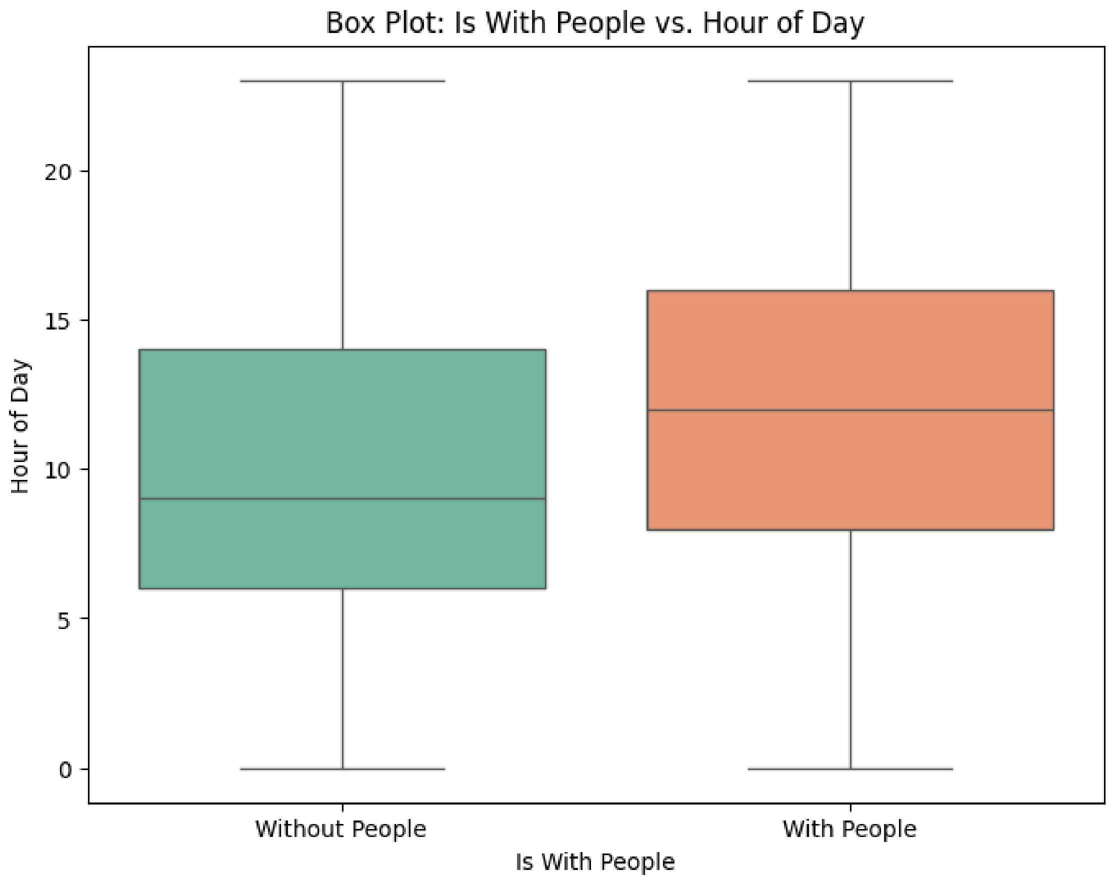
```
/var/folders/r0/68n8gcdd48vgkv5r026jhnlh0000gn/T/ipykernel_17000/2368375072.py:5
: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.countplot(x='is_with_people', data=labeled_memories_df, palette='Set2')
```



3.7.2 Cross-Analysis

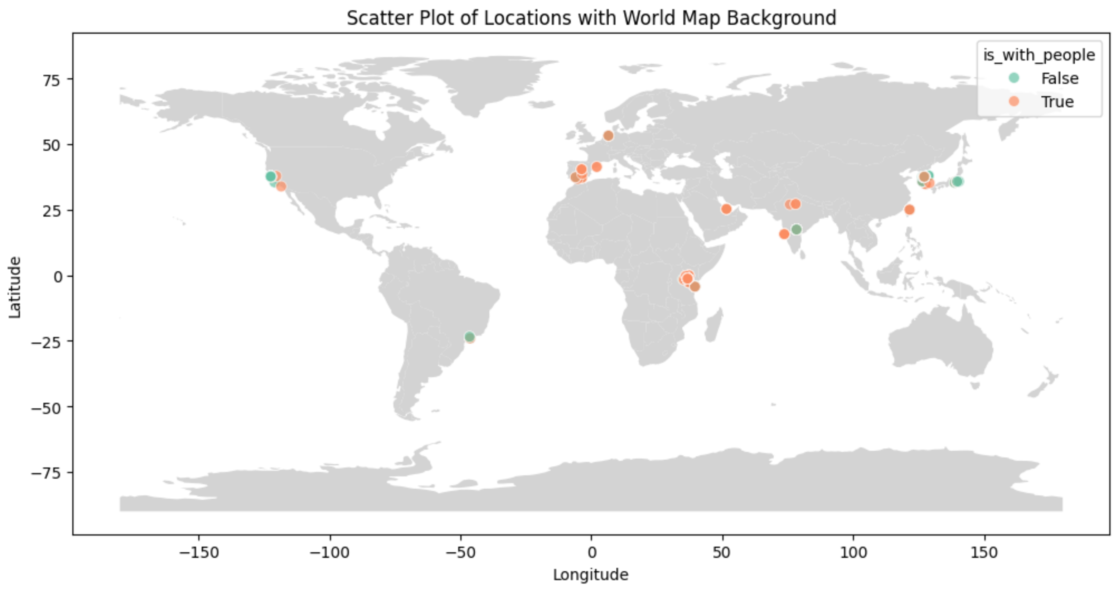
[11]:
```
# Box plot of is_with_people vs hour_of_day
plt.figure(figsize=(8, 6))
sns.boxplot(x='is_with_people', y='hour_of_day', data=labeled_memories_df,␣
 ↪palette='Set2')
plt.title('Box Plot: Is With People vs. Hour of Day')
plt.xlabel('Is With People')
plt.ylabel('Hour of Day')
plt.xticks([0, 1], ['Without People', 'With People'])
plt.show()
```

```
/var/folders/r0/68n8gcdd48vgkv5r026jhnlh0000gn/T/ipykernel_17000/752509882.py:3:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same
effect.

  sns.boxplot(x='is_with_people', y='hour_of_day', data=labeled_memories_df,
palette='Set2')
```



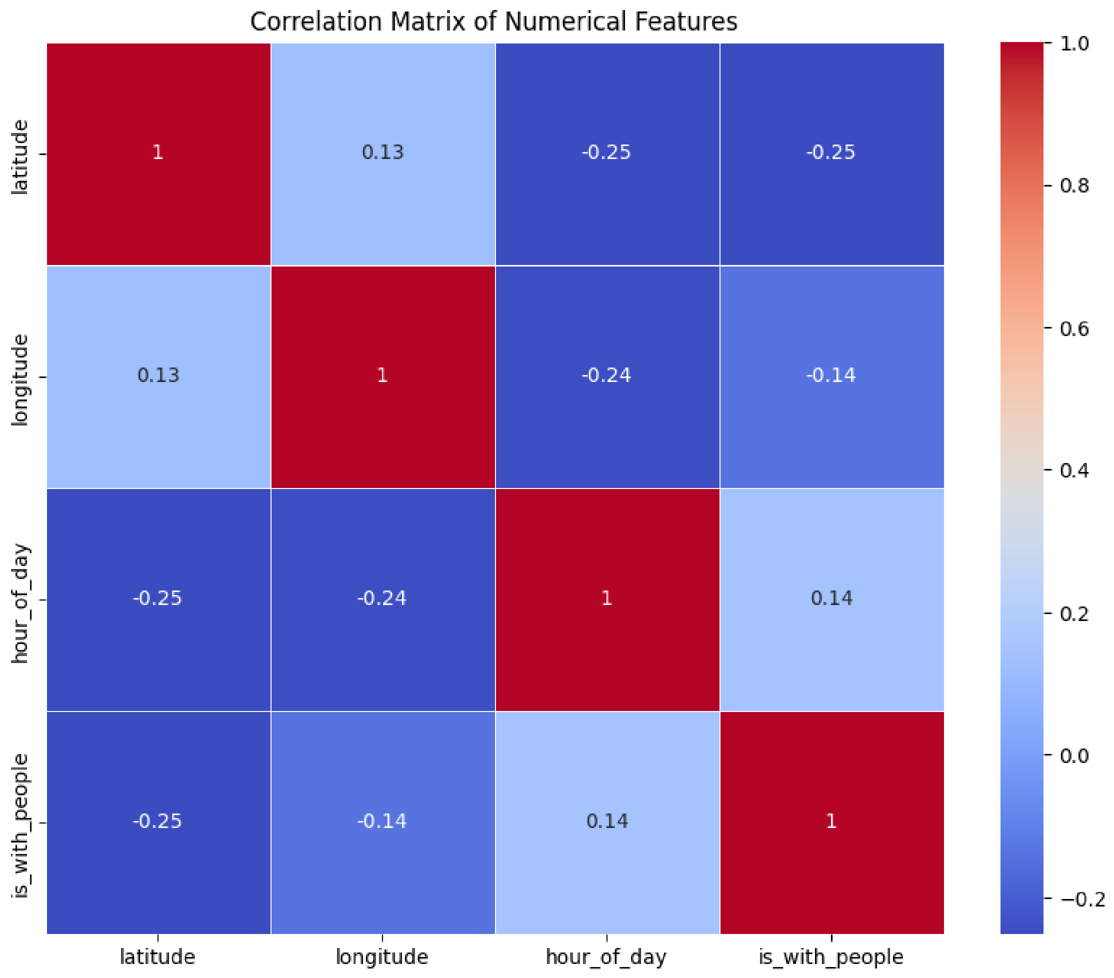
```
[12]: import geopandas as gpd
import matplotlib.pyplot as plt
import seaborn as sns

# Load world map from GeoPandas
```

```
world = gpd.read_file('/Users/minjaekim/Projects/bereal-ml-project/data/
↪ne_110m_admin_0_countries')

# Create a GeoDataFrame for your memories dataset
gdf = gpd.GeoDataFrame(labeled_memories_df, geometry=gpd.
↪points_from_xy(labeled_memories_df['longitude'],␣
↪labeled_memories_df['latitude']))

# Plot the world map
fig, ax = plt.subplots(figsize=(12, 8))
world.plot(ax=ax, color='lightgray')

# Plot the scatter points on the map
sns.scatterplot(x='longitude', y='latitude', hue='is_with_people',␣
↪data=labeled_memories_df, ax=ax, s=50, alpha=0.7, palette='Set2')

# Set titles and labels
plt.title('Scatter Plot of Locations with World Map Background')
plt.xlabel('Longitude')
plt.ylabel('Latitude')

plt.show()
```



3.7.3 Correlation Matrix

```
[13]: # Correlation matrix and heatmap
plt.figure(figsize=(10, 8))
corr_matrix = labeled_memories_df[['latitude', 'longitude', 'hour_of_day',␣
  ↪'is_with_people']].corr()
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5)
plt.title('Correlation Matrix of Numerical Features')
plt.show()
```



4 Analysis Overview and Data splitting

The analysis method chosen for this dataset is classification. The primary goal is to predict whether a memory is labeled as "with people" or "without people" based on the available features, such as the time the memory was captured, its location, and whether the post was late.

Before training the machine learning models, the dataset was split into training and testing sets.

The features selected for prediction included latitude, longitude, hour of day, day of the week, and whether the post was late. The is_with_people label was chosen as the target variable.

The categorical feature, day_of_week, was encoded using one-hot encoding to prepare it for the model. Rows containing any missing values in the features or the target variable were removed to ensure the model received clean data.

The data was split into 80% training and 20% testing using the train_test_split() function from Scikit-learn, with a random seed (random_state=42) to ensure reproducibility. The target variables, y_train and y_test, were converted to integers (0 for False, 1 for True) for compatibility with the machine learning models. This process produced 647 training samples and 162 test samples across 11 feature columns.

[14]:
```
# Reminder of how the df looks now after preprocessing and labeling
labeled_memories_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 955 entries, 0 to 954
Data columns (total 11 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   front_image     955 non-null    object
 1   back_image      955 non-null    object
 2   is_late         955 non-null    bool
 3   date            955 non-null    object
 4   taken_time      955 non-null    object
 5   bereal_moment   955 non-null    object
 6   latitude        809 non-null    float64
 7   longitude       809 non-null    float64
 8   day_of_week     955 non-null    object
 9   hour_of_day     955 non-null    int64
 10  is_with_people  955 non-null    bool
dtypes: bool(2), float64(2), int64(1), object(6)
memory usage: 69.1+ KB
```

[15]:
```
from sklearn.model_selection import train_test_split

# Drop rows with NaN values in the features or target
memories_df_cleaned = labeled_memories_df.dropna()

# Define the features (X) and the target variable (y)
X = memories_df_cleaned[['latitude', 'longitude', 'hour_of_day', 'day_of_week',␣
↪'is_late']]
# For 'day_of_week', encoding it since it's categorical
X = pd.get_dummies(X, columns=['day_of_week'])
y = memories_df_cleaned['is_with_people']  # Target column

# Split the data into training and testing sets (80% train, 20% test)
```

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

# Verify the shapes
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
```

```
(647, 11) (162, 11) (647,) (162,)
```

[16]:
```
# Convert y_train and y_test to integers (0 for False, 1 for True)
y_train = y_train.astype(int)
y_test = y_test.astype(int)
```

5 Model Selection

In this section, various models are selected to predict whether a memory is labeled as with people or without people based on the available features. The goal is to determine which model provides the best predictive accuracy for this classification task. Three models were chosen for this analysis: Logistic Regression, Naive Bayes, and Random Forest. Below is a brief discussion of each model and its mathematical underpinnings.

5.1 Logistic Regression

Logistic Regression is a linear model used for binary classification. It models the probability that a given input belongs to a particular class (0 or 1) using a logistic function, also called a sigmoid function.

The model computes a linear combination of the input features and applies the sigmoid function to predict the probability of a memory being classified as "with people" (1) or "without people" (0). The sigmoid function is defined as follows:

$$Sigmoid(z) = \frac{1}{1 + e^{-z}}$$

where $z = w^T x + b$, and w are the model's learned weights, x is the input vector (features), and b is the bias term.

The predicted class $\hat{y}$ is determined by thresholding the output of the sigmoid function:

$$\hat{y} = \begin{cases} 1, & \text{if } Sigmoid(z) \geq 0.5 \\ 0, & \text{otherwise} \end{cases}$$

5.2 Naive Bayes

Naive Bayes is a probabilistic model that assumes independence between the features given the target class. Despite this assumption, it is often effective for high-dimensional data and can perform well even when the independence assumption is not entirely valid. This makes it appropriate for predicting whether the memories in this dataset are labeled as "with people" or "without people" based on the features provided. The model is based on Bayes' Theorem, which calculates the posterior probability of each class given the input features, and classifies the data point based on the highest posterior probability. The equation for Bayes' Theorem is:

$$P(C \mid X) = \frac{P(X \mid C)P(C)}{P(X)}$$

where:

- $P(C \mid X)$ is the posterior probability of class C given features X,
- $P(X \mid C)$ is the likelihood of features X given class C,
- $P(C)$ is the prior probability of class C, and
- $P(X)$ is the probability of features X (this term is often ignored in practice since it's the same for all classes).

The Naive Bayes algorithm assumes the likelihood $P(X \mid C)$ can be decomposed as the product of the probabilities for each feature independently:

$$P(X \mid C) = \prod_{i=1}^{n} P(x_i \mid C)$$

where x_i are individual features.

5.3 Random Forest

For the final model, a Random Forest classifier was selected, which is a robust ensemble learning method that operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes across the trees. This model is well-suited for handling complex relationships between features and typically avoids overfitting by averaging the results of many trees. Each tree is trained on a random subset of the data and makes a prediction based on the features. The final prediction is made by majority voting from all the individual trees.

The model's underlying algorithm can be summarized in the following pseudocode:

```
1. For each tree in the forest:

    - Select a random subset of the training data.
    - Build a decision tree by splitting the dataset at nodes based on the most informative fe

2. Aggregate the predictions from all trees (for classification, use majority
voting).

3. Output the final class label based on the majority vote.
```

6 Modeling

6.1 Logistic Regression

6.1.1 Training

The logistic regression model is trained using the training dataset, where it optimizes the coefficients for each feature to best fit the relationship between the input features and the target variable, which is the is_with_people label. By minimizing the log-likelihood function during training, the model seeks to maximize the probability of the correct classification for each data point in the training set.

[17]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Initialize the Logistic Regression model
model = LogisticRegression(max_iter=1000)

# Train the model on the training data
model.fit(X_train, y_train)

# Make predictions on the test data
y_pred = model.predict(X_test)
```

6.1.2 Evaluation

The logistic regression model achieved an accuracy of 0.56, which is only slightly better than random guessing for this binary classification task. The confusion matrix reveals that the model correctly predicted 57 out of 85 samples for the '0' class (without people) and 34 out of 77 samples for the '1' class (with people).

When analyzing the precision, recall, and F1 scores, it becomes clear that the model's performance is inconsistent across the two classes. The precision for the '0' class is 0.57, meaning that of all the instances predicted as 'without people,' 57% were correctly identified. In contrast, the precision for the '1' class is 0.55, indicating only a slightly lower ability to correctly predict 'with people' instances.

The recall for the '0' class (0.67) is significantly higher than that of the '1' class (0.44), suggesting that the model struggles more with identifying 'with people' instances correctly. The F1 score, which balances precision and recall, is also more favorable for the '0' class (0.62) compared to the '1' class (0.49).

Given these metrics, it is evident that the classification is close to random, especially for the '1' class, where recall and F1 scores are notably low. This overall performance indicates that the logistic regression model is not capturing meaningful patterns in the data.

[18]:
```python
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Calculate accuracy
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.2f}")

# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
```

```
# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred))
```

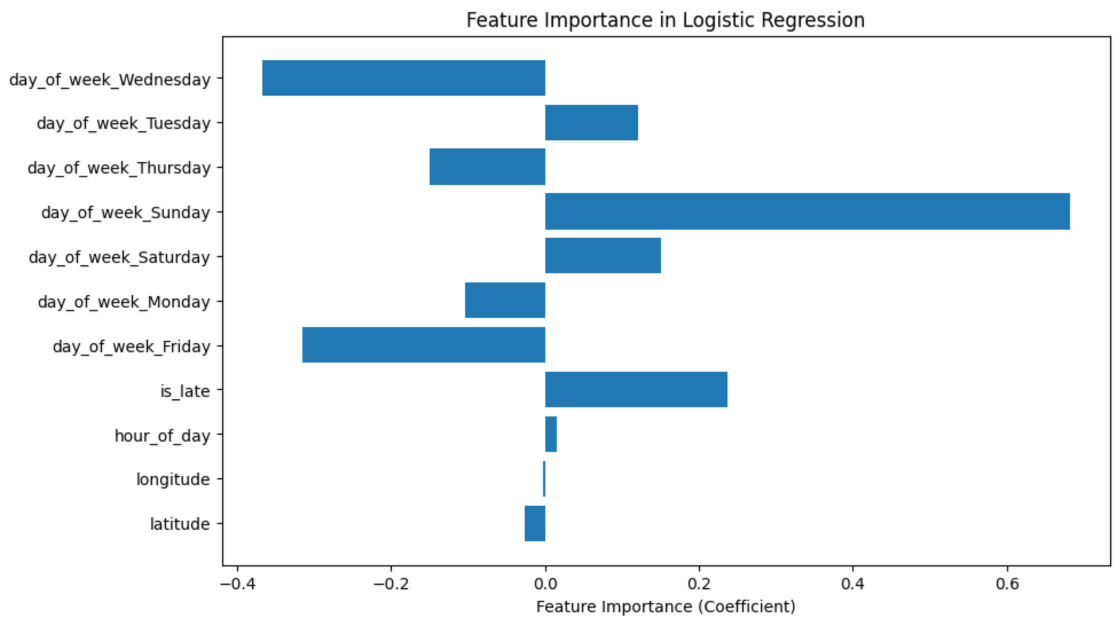
```
Accuracy: 0.56
Confusion Matrix:
[[57 28]
 [43 34]]
Classification Report:
              precision    recall  f1-score   support

           0       0.57      0.67      0.62        85
           1       0.55      0.44      0.49        77

    accuracy                           0.56       162
   macro avg       0.56      0.56      0.55       162
weighted avg       0.56      0.56      0.56       162
```

6.1.3 Feature Importance

A feature importance analysis was conducted by extracting the model's coefficients. The chart below displays the coefficients corresponding to each feature, indicating the relative importance of features in making predictions. However, the interpretation of feature importance in logistic regression can be challenging due to the complex relationships between variables. Therefore, this step is optional but may offer some insight into which features had the largest impact on the predictions.

[19]:
```
# Assuming 'model' is your trained LogisticRegression model
import numpy as np
import matplotlib.pyplot as plt

# Extract feature names (you should have a list of your feature names)
feature_names = X_train.columns  # Or the names of your features if you don't␣
 ↪have a DataFrame

# Get the coefficients from the model
coefficients = model.coef_[0]

# Create a bar plot for feature importance
plt.figure(figsize=(10, 6))
plt.barh(np.arange(len(coefficients)), coefficients, align='center')
plt.yticks(np.arange(len(coefficients)), feature_names)
plt.xlabel('Feature Importance (Coefficient)')
plt.title('Feature Importance in Logistic Regression')
plt.show()

# Print out the coefficients with their corresponding feature names
```

```
for feature, coef in zip(feature_names, coefficients):
    print(f"Feature: {feature}, Coefficient: {coef}")
```



```
Feature: latitude, Coefficient: -0.026358228621871958
Feature: longitude, Coefficient: -0.0030462257698804997
Feature: hour_of_day, Coefficient: 0.014718963707005235
Feature: is_late, Coefficient: 0.2377107547564553
Feature: day_of_week_Friday, Coefficient: -0.3151726268120711
Feature: day_of_week_Monday, Coefficient: -0.10398551313799312
Feature: day_of_week_Saturday, Coefficient: 0.15107404735947844
Feature: day_of_week_Sunday, Coefficient: 0.6817031970689084
Feature: day_of_week_Thursday, Coefficient: -0.14949762166135827
Feature: day_of_week_Tuesday, Coefficient: 0.12087494898567147
Feature: day_of_week_Wednesday, Coefficient: -0.36675404016389845
```

6.2 Gaussian Naive Bayes

6.2.1 Training

The training process involves fitting the Gaussian Naive Bayes model to the training dataset, which includes the latitude, longitude, day of the week, hour of the day, and other features. The model computes the likelihood of the data given each class (with_people or without_people), using the Gaussian probability density function for continuous features like the latitude and longitude.

[20]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix,␣
↪classification_report
```

```python
# Initialize the Gaussian Naive Bayes model
nb_model = GaussianNB()

# Train the model on the training data
nb_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred_nb = nb_model.predict(X_test)
```

```
Naive Bayes Accuracy: 0.55
Confusion Matrix:
[[51 34]
 [39 38]]
Classification Report:
              precision    recall  f1-score   support

           0       0.57      0.60      0.58        85
           1       0.53      0.49      0.51        77

    accuracy                           0.55       162
   macro avg       0.55      0.55      0.55       162
weighted avg       0.55      0.55      0.55       162
```

6.2.2 Evaluation

The Gaussian Naive Bayes model produced an accuracy of 0.55, which is close to random guessing in this binary classification task. The confusion matrix shows that the model correctly predicted 51 out of 85 samples for the '0' class (without people) and 38 out of 77 samples for the '1' class (with people).

The precision for the '0' class is 0.57, while for the '1' class it is slightly lower at 0.53. This indicates that the model is almost equally likely to misclassify both 'with people' and 'without people' posts. The recall for the '0' class is 0.60, meaning that 60% of the 'without people' posts were correctly identified, while the recall for the '1' class is 0.49, showing that the model has difficulty in detecting 'with people' instances.

The F1 score, which balances precision and recall, is similarly weak across both classes: 0.58 for the '0' class and 0.51 for the '1' class. The overall macro and weighted averages for precision, recall, and F1 score remain around 0.55, reflecting a generally poor performance across the board.

Given these metrics, it is clear that the Naive Bayes model is not performing significantly better than random guessing, particularly in its ability to distinguish 'with people' posts, where the recall and F1 scores are notably low.

[ ]:
```python
# Evaluate the model
accuracy_nb = accuracy_score(y_test, y_pred_nb)
print(f"Naive Bayes Accuracy: {accuracy_nb:.2f}")
```

```
# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_nb))

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_nb))
```

6.3 Random Forest Classifier

6.3.1 Training

The Random Forest classifier was initialized with default parameters, and the model was trained on the same feature set used for the previous models, including latitude, longitude, hour_of_day, day_of_week, and is_late. The classifier was trained using 80% of the data as the training set, leaving 20% for evaluation. The number of trees (n_estimators) in the forest was set to 100, with no maximum depth constraint to allow the trees to grow fully.

[21]:
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix,␣
 ↪classification_report

# Initialize the Random Forest Classifier
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Train the model on the training data
rf_model.fit(X_train, y_train)

# Make predictions on the test data
y_pred_rf = rf_model.predict(X_test)
```

6.3.2 Evaluation

The Random Forest classifier achieved an accuracy of 0.72, significantly outperforming the Logistic Regression and Naive Bayes models. The confusion matrix indicates that the model correctly classified 63 out of 85 samples in the '0' class (without people) and 53 out of 77 samples in the '1' class (with people), demonstrating a much better balance in predictive performance between the two classes.

The precision, recall, and F1-score metrics are strong for both classes, with precision values of 0.72 for the '0' class and 0.71 for the '1' class. This reflects that the model is fairly balanced in its ability to correctly predict the 'with people' and 'without people' instances. The recall scores, 0.74 for the '0' class and 0.69 for the '1' class, indicate that the model is able to identify most of the true positives for both classes, with a slight advantage for identifying 'without people' posts.

The overall F1 scores are 0.73 for the '0' class and 0.70 for the '1' class, indicating a strong balance between precision and recall across both categories. The macro average and weighted average values for precision, recall, and F1 score all sit around 0.72, showing that the model performs well across the dataset as a whole.

In summary, the Random Forest classifier provides a noticeable improvement over the Logistic Regression and Naive Bayes models, likely due to its ability to capture more complex interactions between features. However, further tuning or feature engineering could potentially improve the model's performance even more.

[22]:
```
# Calculate accuracy
accuracy_rf = accuracy_score(y_test, y_pred_rf)
print(f"Random Forest Accuracy: {accuracy_rf:.2f}")

# Confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred_rf))

# Classification report
print("Classification Report:")
print(classification_report(y_test, y_pred_rf))
```

```
Random Forest Accuracy: 0.72
Confusion Matrix:
[[63 22]
 [24 53]]
Classification Report:
              precision    recall  f1-score   support

           0       0.72      0.74      0.73        85
           1       0.71      0.69      0.70        77

    accuracy                           0.72       162
   macro avg       0.72      0.71      0.71       162
weighted avg       0.72      0.72      0.72       162
```

6.3.3 Hyperparameter Tuning

Hyperparameter tuning was performed using GridSearchCV to optimize the performance of the Random Forest classifier. The Random Forest algorithm has several hyperparameters that can significantly influence its performance, such as the number of trees (n_estimators), the maximum depth of each tree (max_depth), the minimum number of samples required to split a node (min_samples_split), and the number of features to consider when looking for the best split (max_features).

A grid search was employed to systematically explore a range of values for these hyperparameters and identify the combination that yields the best classification performance. The search was conducted with a 5-fold cross-validation (cv=5), ensuring that the model was evaluated on different subsets of the training data to minimize overfitting and select a robust model.

Once the grid search completed, the best hyperparameter combination was selected based on the cross-validated performance. The model was then retrained using the optimal hyperparameters and tested on the held-out test data to evaluate its performance.

[23]:
```python
from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier

# Define the parameter grid
param_grid = {
    'n_estimators': [100, 200, 300],          # Number of trees in the forest
    'max_depth': [None, 10, 20, 30],          # Maximum depth of the trees
    'min_samples_split': [2, 5, 10],          # Minimum number of samples␣
 ↪required to split a node
    'min_samples_leaf': [1, 2, 4],            # Minimum number of samples␣
 ↪required at a leaf node
    'max_features': ['sqrt', 'log2']           # Number of features to consider␣
 ↪at each split
}

# Initialize GridSearchCV with RandomForestClassifier
grid_search = GridSearchCV(estimator=RandomForestClassifier(),␣
 ↪param_grid=param_grid, cv=5, n_jobs=-1, verbose=1)

# Fit the grid search model to the training data
grid_search.fit(X_train, y_train)

# Get the best parameters and model
print(f"Best Parameters: {grid_search.best_params_}")
best_rf_model = grid_search.best_estimator_

# Make predictions with the best model
y_pred_best_rf = best_rf_model.predict(X_test)
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
Best Parameters: {'max_depth': 20, 'max_features': 'log2', 'min_samples_leaf':
2, 'min_samples_split': 2, 'n_estimators': 100}
Tuned Random Forest Accuracy: 0.74
Confusion Matrix:
[[66 19]
 [23 54]]
Classification Report:
              precision    recall  f1-score   support

           0       0.74      0.78      0.76        85
           1       0.74      0.70      0.72        77

    accuracy                           0.74       162
   macro avg       0.74      0.74      0.74       162
weighted avg       0.74      0.74      0.74       162
```

6.3.4 Tuned Model Evaluation

After tuning the hyperparameters using GridSearchCV, the optimized Random Forest model was evaluated on the test dataset. The hyperparameters chosen by the grid search include a maximum depth of 20 (max_depth), log-based feature selection (max_features: 'log2'), a minimum of two samples required to split an internal node (min_samples_split), and two samples as the minimum leaf size (min_samples_leaf). These settings were used to fit the model, resulting in a test accuracy of 0.74, which is an improvement compared to the baseline Random Forest model.

The confusion matrix shows that the classifier correctly predicted 66 true positives (correctly identified posts with no people) and 54 true negatives (correctly identified posts with people). There were 19 false negatives (incorrectly classified posts as containing people when they did not) and 23 false positives (posts incorrectly classified as not containing people).

The classification report further illustrates the model's performance, with precision, recall, and F1-scores all around 0.74. Both classes—posts with and without people—are classified with relatively balanced precision and recall, indicating that the tuned Random Forest model generalizes well across the two classes. Overall, the tuned model demonstrates a strong ability to correctly classify social presence in the dataset, with only minor misclassification issues.

```
[ ]: # Evaluate the tuned model
     from sklearn.metrics import accuracy_score, confusion_matrix,␣
      ↪classification_report
     accuracy_best_rf = accuracy_score(y_test, y_pred_best_rf)
     print(f"Tuned Random Forest Accuracy: {accuracy_best_rf:.2f}")

     # Confusion matrix
     print("Confusion Matrix:")
     print(confusion_matrix(y_test, y_pred_best_rf))

     # Classification report
     print("Classification Report:")
     print(classification_report(y_test, y_pred_best_rf))
```

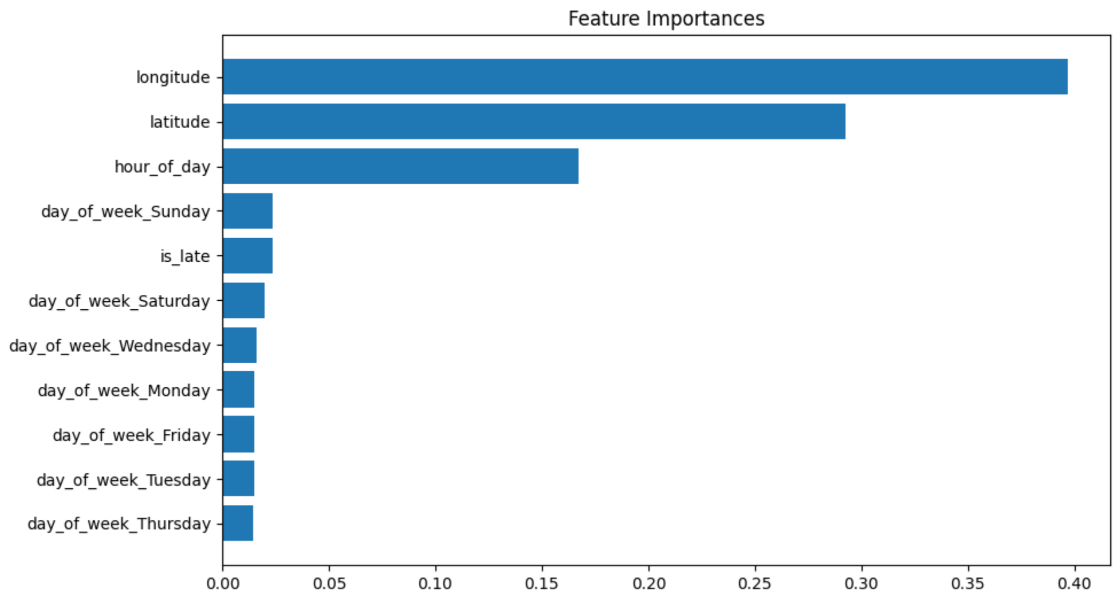
6.3.5 Feature Importance

To gain insights into how the Random Forest model made its predictions, feature importance was computed for each of the variables. Feature importance helps identify which features had the most influence in predicting whether a memory was labeled as "with people" or "without people."

The bar plot above shows the relative importance of each feature. Longitude and Latitude were by far the most significant features, suggesting that the geographical location where the BeReal posts were taken played a key role in the model's classification decisions. The Hour of Day also had a notable influence, indicating that the time when the memory was posted is a relevant predictor of whether it involved social interactions.

Other features like Day of the Week and Is Late had smaller but still measurable impacts on the model's predictions, with posts on specific days (e.g., Sunday) and the lateness of the post contributing to a lesser extent. These insights highlight the importance of both spatial and temporal factors in predicting whether a user was with other people when making a post.

```
[24]: import matplotlib.pyplot as plt
      import numpy as np

      # Get the feature importance from the best Random Forest model
      feature_importances = best_rf_model.feature_importances_

      # Plot feature importance
      features = X_train.columns  # Original features
      indices = np.argsort(feature_importances)[::-1]

      plt.figure(figsize=(10, 6))
      plt.title("Feature Importances")
      plt.barh(range(len(indices)), feature_importances[indices], align="center")
      plt.yticks(range(len(indices)), [features[i] for i in indices])
      plt.gca().invert_yaxis()  # To display the most important feature on top
      plt.show()
```



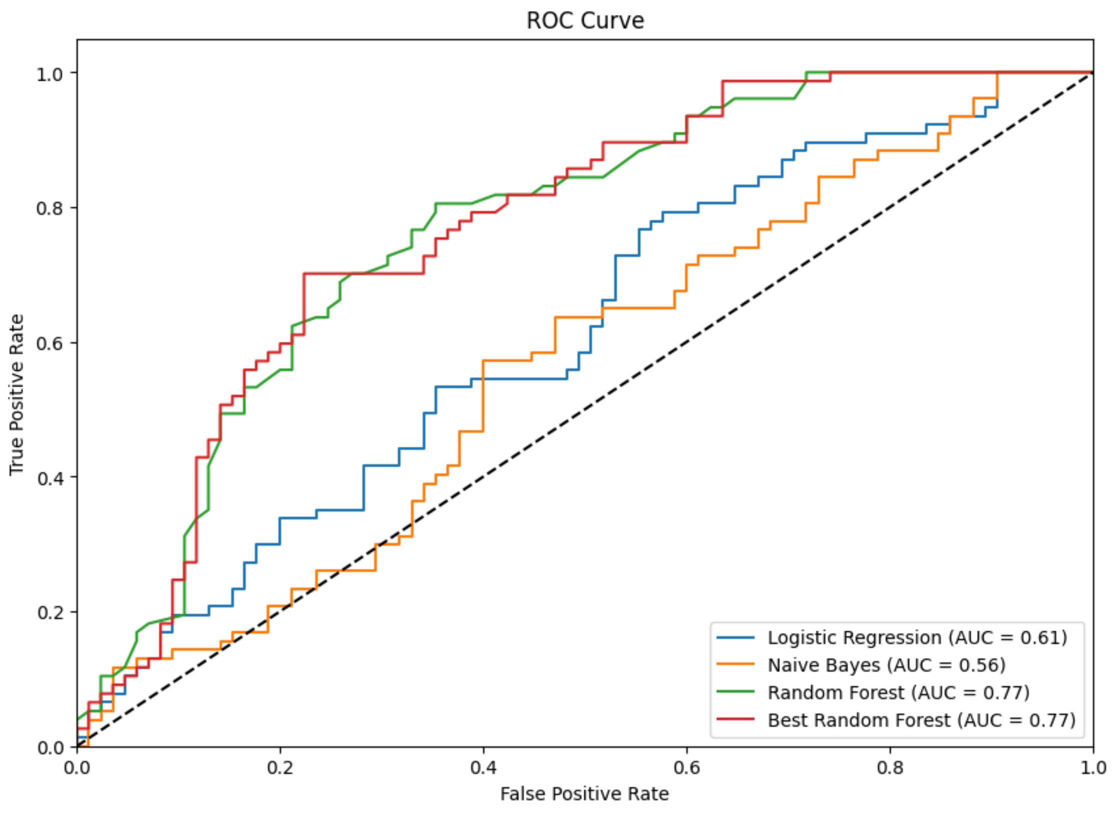
6.4 ROC Curves Analysis

To further evaluate the models, a Receiver Operating Characteristic (ROC) curve was plotted for each. The ROC curve provides insight into the trade-offs between the true positive rate (recall) and false positive rate at various classification thresholds. Additionally, the Area Under the Curve (AUC) value is calculated to represent each model's overall ability to distinguish between the two classes.

From the ROC curves: - Logistic Regression shows an AUC of 0.61, which indicates that it performs only slightly better than random classification (AUC of 0.5). This confirms the prior analysis of its poor performance across other metrics such as precision and recall. - Naive Bayes shows the weakest performance with an AUC of 0.56, making it the least effective of the models evaluated, which is consistent with its lower accuracy and f1-scores. The curve is relatively close to the random guess diagonal, suggesting little separation between the classes. - Random Forest performs significantly better, with both its untuned and tuned versions achieving an AUC of 0.77. The curves demonstrate a higher true positive rate across a range of false positive rates compared to the other models, indicating the Random Forest model's superior ability to classify the data correctly. This further validates the findings from the accuracy, precision, and recall metrics, where the Random Forest outperformed the other models.

In conclusion, the ROC curve analysis reinforces the observations from the confusion matrix and classification report, confirming that the Random Forest model (both tuned and untuned) provides the best performance, followed by Logistic Regression and Naive Bayes.

[27]:
```python
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

# Define a function to plot ROC curve
def plot_roc_curve(model, X_test, y_test, model_name):
    # Predict the probabilities for the positive class (is_with_people = 1)
    y_prob = model.predict_proba(X_test)[:, 1]

    # Compute ROC curve and AUC (Area Under the Curve) score
    fpr, tpr, thresholds = roc_curve(y_test, y_prob)
    roc_auc = auc(fpr, tpr)

    # Plot ROC curve
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {roc_auc:.2f})')

# Plot ROC curves for Logistic Regression, Naive Bayes, and Random Forest
plt.figure(figsize=(10, 7))

# Logistic Regression ROC
plot_roc_curve(model, X_test, y_test, 'Logistic Regression')

# Naive Bayes ROC
plot_roc_curve(nb_model, X_test, y_test, 'Naive Bayes')

# Random Forest ROC
plot_roc_curve(rf_model, X_test, y_test, 'Random Forest')

# Best Random Forest ROC
plot_roc_curve(best_rf_model, X_test, y_test, 'Best Random Forest')

# Plot settings
plt.plot([0, 1], [0, 1], 'k--')  # Dashed diagonal line (random classifier)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```



7 Results and Conclusion

The results of the models trained on the dataset, which aimed to predict whether a memory was labeled as “with people” or “without people,” were mixed, with some models performing better than others in terms of accuracy, precision, recall, and F1 score. Below is a summary of the findings for each model:

7.1 Logistic Regression

The logistic regression model achieved an accuracy of 0.56, which is close to random guessing. While the precision for the “with people” (label 1) class was 0.55, the recall for this class was low at 0.44, indicating that the model often failed to correctly identify instances where people were present. The precision and recall for the “without people” (label 0) class were higher at 0.57 and 0.67, respectively, but overall, the model’s predictions were only marginally better than chance. This suggests that logistic regression may not be well-suited for this problem given the complexity of the features and their relationships.

7.2 Gaussian Naive Bayes

The Naive Bayes model also exhibited relatively poor performance, with an overall accuracy of 0.55. The precision and recall for both classes were nearly identical, with precision scores around 0.53–0.57 and recall values around 0.49–0.60. Like the logistic regression model, Naive Bayes struggled to reliably classify posts, particularly for the "with people" class. Given the assumption of independence between features inherent in Naive Bayes, it is likely that this model's performance was hindered by the correlations between certain features (e.g., latitude and longitude), which are not independent in reality.

7.3 Random Forest Classifier

The Random Forest model significantly outperformed both logistic regression and Naive Bayes, achieving an accuracy of 0.72. The model showed balanced performance across both classes, with precision and recall values around 0.71–0.74 for both "with people" and "without people" labels. The tuned version of the Random Forest model, using GridSearchCV, further improved the performance slightly to an accuracy of 0.74. This suggests that the Random Forest was better able to capture the complexity of the dataset, leveraging non-linear relationships between features such as location (latitude and longitude), time of day, and day of the week to make more accurate predictions.

7.4 Feature Importance

The feature importance analysis revealed that longitude and latitude were the most influential features in the Random Forest model's decision-making process. Hour of Day also played a significant role, indicating that both where and when a post was made are strong predictors of whether the user was with other people. The importance of temporal features like the Day of the Week and the "lateness" of the post had less but still notable contributions.

7.5 Visualization of Results

Several visualizations were generated to provide a clearer understanding of the model's performance: - Confusion Matrices were plotted for each model, showing how the models classified posts across the "with people" and "without people" categories. This revealed common misclassification patterns, particularly for the "with people" class, where most models struggled to achieve a high recall. - Feature Importance Plot (for Random Forest) provided insight into which features were driving the model's predictions. Geographic (longitude and latitude) and temporal data (hour of day) were the most significant predictors of whether people were present in the images. - ROC Curves were generated for all models to visualize the trade-off between true positive and false positive rates at various thresholds. The Random Forest model demonstrated the best performance, with an Area Under the Curve (AUC) of 0.77, significantly outperforming Logistic Regression (AUC of 0.61) and Naive Bayes (AUC of 0.56). This visualization highlights Random Forest's superior ability to distinguish between the two classes compared to the other models.

7.6 Conclusion

In conclusion, the Random Forest model was the most successful at predicting whether a post involved other people, with an accuracy of 0.74. The strong influence of spatial features like longitude and latitude suggests that where a user is posting from is highly indicative of their social behavior. Temporal features like the Hour of Day also proved to be important, possibly capturing patterns in when users are more likely to be with others.

In contrast, both the logistic regression and Naive Bayes models struggled to accurately classify posts, indicating that they may not be suitable for this task due to the complex relationships between the features.

8 Executive Summary

This project aimed to predict whether a memory was labeled as "with people" or "without people" based on data from the BeReal dataset, including both images and metadata such as location, time, and other contextual features. The analysis followed a structured pipeline that included data loading, preprocessing, exploratory analysis, model selection, training, and evaluation. Below is an overview of the process, key results, and insights gained throughout the analysis.

8.1 Pipeline Overview

The pipeline for this project can be broken down into the following steps:

8.1.1 Data Loading and Preprocessing:

- The BeReal dataset, containing metadata and image data, was loaded into a Pandas DataFrame. The metadata included features like latitude, longitude, hour of day, day of week, and lateness.
- Images were preprocessed by flattening their pixel data for inclusion in the models. Additionally, rows with missing latitude or longitude values were dropped to ensure data consistency.
- A new binary feature, is_with_people, was manually labeled based on whether other people were physically present in the image and known to the user.

8.1.2 Exploratory Data Analysis (EDA):

- EDA was conducted to explore the distribution of key features, such as day of week, hour of day, and whether posts were "with people" or "without people."
- Key insights showed the importance of location and time-based features, particularly latitude, longitude, and hour of day, in determining whether people were present in a post.

8.1.3 Data Splitting:

- The dataset was split into training (80%) and test (20%) sets using train_test_split to ensure the models could be properly evaluated.

8.1.4 Model Selection and Training:

- Three models were selected for classification: Logistic Regression, Gaussian Naive Bayes, and Random Forest.
- Each model was trained on the training set and evaluated based on accuracy, precision, recall, and F1 score.

8.1.5 Model Evaluation:

- Logistic Regression achieved an accuracy of 0.56, indicating poor classification performance. Precision and recall scores were also suboptimal, particularly for the "with people" class.
- Naive Bayes achieved an accuracy of 0.55, similarly underperforming and showing issues with recall for correctly classifying posts "with people."
- Random Forest significantly outperformed the other models with an accuracy of 0.72, and after hyperparameter tuning using GridSearchCV, the accuracy improved to 0.74. The Random Forest model showed balanced performance across both classes and revealed that longitude, latitude, and hour of day were the most important features in predicting whether posts involved people.

8.1.6 Feature Importance:

- A feature importance analysis of the Random Forest model highlighted longitude, latitude, and hour of day as the most influential features, emphasizing the significance of spatial and temporal factors in predicting social interaction.

8.1.7 ROC Curves:

- ROC (Receiver Operating Characteristic) curves were plotted for all three models to visualize their ability to distinguish between the "with people" and "without people" categories.
- The Random Forest model, both in its original form and after hyperparameter tuning, achieved the highest Area Under the Curve (AUC) score of 0.77, indicating a stronger ability to distinguish between the two classes. Logistic Regression followed with an AUC of 0.61, while Naive Bayes performed the worst with an AUC of 0.56.
- The ROC curves showed that while the Random Forest model provided the best trade-off between true positives and false positives, both Logistic Regression and Naive Bayes struggled with this balance, reinforcing the results from the confusion matrix and classification report.

8.2 Key Results and Insights

The Random Forest model was the most successful at predicting social behavior, with a final accuracy of 0.74. The model leveraged the strong predictive power of location and time-based features, providing valuable insights into how and where users are likely to be with others. Both logistic regression and Naive Bayes struggled to classify posts accurately, largely due to the complexity of the feature relationships and the assumptions underlying these models (e.g., independence of features in Naive Bayes). The feature importance analysis confirmed that spatial and temporal data, especially longitude and hour of day, are critical for predicting social interactions, which aligns with intuitive expectations about when and where people are more likely to be with others.

8.3 Shortcomings and Areas for Improvement

- Model Limitations: Both logistic regression and Naive Bayes performed poorly, indicating they are not well-suited for this problem. Their inability to handle non-linear relationships between the features likely contributed to this underperformance.
- Feature Engineering: The image data was included in a basic form (flattened pixels), which may not have been the most effective way to incorporate this information. Advanced image processing techniques, such as convolutional neural networks (CNNs), could be applied in future work to better leverage the rich information contained in the images.

  - *As a sidenote, image preprocessing by flattening the pixels of the images was done and can be found on the "image-preprocessing" branch on the GitHub repository. It was excluded here because the image preprocessed training data did not work with the GridSearchCV. This task is also being saved for the next assignment.*
- Temporal Features: While the hour of day was a significant predictor, the analysis could be improved by incorporating more granular temporal features (e.g., time of year, social events) to capture seasonal trends in social interaction.

8.4 Conclusion and Future Work

In conclusion, the Random Forest model demonstrated the best ability to classify posts based on social interaction, driven primarily by location and time-based features. Future work could explore the use of neural networks for handling image data more effectively and enhance the temporal analysis by incorporating more complex time-based features.

Additionally, more visualizations, such as heatmaps of post locations and time-based patterns, could help further illustrate key insights and improve the overall interpretability of the model's predictions.

9 References

- Minjae Kim. (2024). *BeReal Personal Archive Dataset* [Data set]. Collected from personal digital archive.
- BeReal ML Project GitHub Repository