In [1]:
```python
# Import necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_absolute_error, mean_squared_error
from sklearn.cluster import KMeans
import matplotlib.image as mpimg  # For loading map image
```

```
2024-12-04 17:16:04.450479: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU instructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
```

In [2]:
```python
# Load the entire training dataset
train_df = pd.read_csv('new-york-city-taxi-fare-prediction/train.csv', nrows=1000000)

# Display the first few rows of the dataset
train_df.head()

# Check for missing values
train_df.isnull().sum()

# Basic statistics summary
train_df.describe()
```

Out[2]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pas
count	1000000.000000	1000000.000000	1000000.000000	999990.000000	999990.000000	100
mean	11.348079	-72.526640	39.929008	-72.527860	39.919954	
std	9.822090	12.057937	7.626154	11.324494	8.201418	
min	-44.900000	-3377.680935	-3116.285383	-3383.296608	-3114.338567	
25%	6.000000	-73.992060	40.734965	-73.991385	40.734046	
50%	8.500000	-73.981792	40.752695	-73.980135	40.753166	
75%	12.500000	-73.967094	40.767154	-73.963654	40.768129	
max	500.000000	2522.271325	2621.628430	45.581619	1651.553433	

In [3]:
```python
# Define bounds for valid latitude and longitude
min_latitude = 40.30
max_latitude = 45.1
min_longitude = -79.46
max_longitude = -71.51

# Print the size before filtering
print("Size before:", len(train_df))

# Filter the dataset to keep only valid coordinates
train_df = train_df[
    (train_df['pickup_latitude'] >= min_latitude) & (train_df['pickup_latitude'] <= max_
```

```
    (train_df['pickup_longitude'] >= min_longitude) & (train_df['pickup_longitude'] <= m
    (train_df['dropoff_latitude'] >= min_latitude) & (train_df['dropoff_latitude'] <= ma
    (train_df['dropoff_longitude'] >= min_longitude) & (train_df['dropoff_longitude'] <=
]

# Print the size after filtering
print("Size after:", len(train_df))
```

```
Size before: 1000000
Size after: 979251
```

In [4]:

```
# Custom Haversine distance calculation function
def haversine_distance(row):
    R = 6371.0  # Earth radius in kilometers
    lat1, lon1 = row['pickup_latitude'], row['pickup_longitude']
    lat2, lon2 = row['dropoff_latitude'], row['dropoff_longitude']

    # Convert latitude and longitude from degrees to radians
    lat1, lon1, lat2, lon2 = map(np.radians, [lat1, lon1, lat2, lon2])

    # Haversine formula
    dlat = lat2 - lat1
    dlon = lon2 - lon1
    a = np.sin(dlat / 2)**2 + np.cos(lat1) * np.cos(lat2) * np.sin(dlon / 2)**2
    c = 2 * np.arcsin(np.sqrt(a))

    distance = R * c  # Distance in kilometers
    return distance

# Calculate distance and add it to the dataframe
train_df['distance'] = train_df.apply(haversine_distance, axis=1)

# Remove unrealistic fares and distances
train_df = train_df[(train_df['fare_amount'] > 2) & (train_df['distance'] > 0)]
```

In [5]:

```
# Define bounding box for map (using the more refined bounding box you provided)
BB = (-74.5, -72.8, 40.5, 41.8)

# Define the filtering function for data within the bounding box
def select_within_boundingbox(df, BB):
    return (df.pickup_longitude >= BB[0]) & (df.pickup_longitude <= BB[1]) & \
           (df.pickup_latitude >= BB[2]) & (df.pickup_latitude <= BB[3]) & \
           (df.dropoff_longitude >= BB[0]) & (df.dropoff_longitude <= BB[1]) & \
           (df.dropoff_latitude >= BB[2]) & (df.dropoff_latitude <= BB[3])

# Filter data based on the bounding box
train_df = train_df[select_within_boundingbox(train_df, BB)]

# Print the size after bounding box filtering
print("Filtered size:", len(train_df))
```

```
Filtered size: 968509
```

In [6]:

```
import requests
from PIL import Image
import io

# Function to load an image from a URL
def load_image_from_url(url):
    response = requests.get(url, verify=False)  # Disable SSL verification for requests
    img_data = response.content
    img = Image.open(io.BytesIO(img_data))
```

```python
    return np.array(img)

# Load image of NYC map for the broader area from URL
nyc_map = load_image_from_url('https://aiblog.nl/download/nyc_-74.5_-72.8_40.5_41.8.png'

# Optionally, you can also load a zoomed-in map for more detailed plots
nyc_map_zoom = load_image_from_url('https://aiblog.nl/download/nyc_-74.3_-73.7_40.5_40.9
```

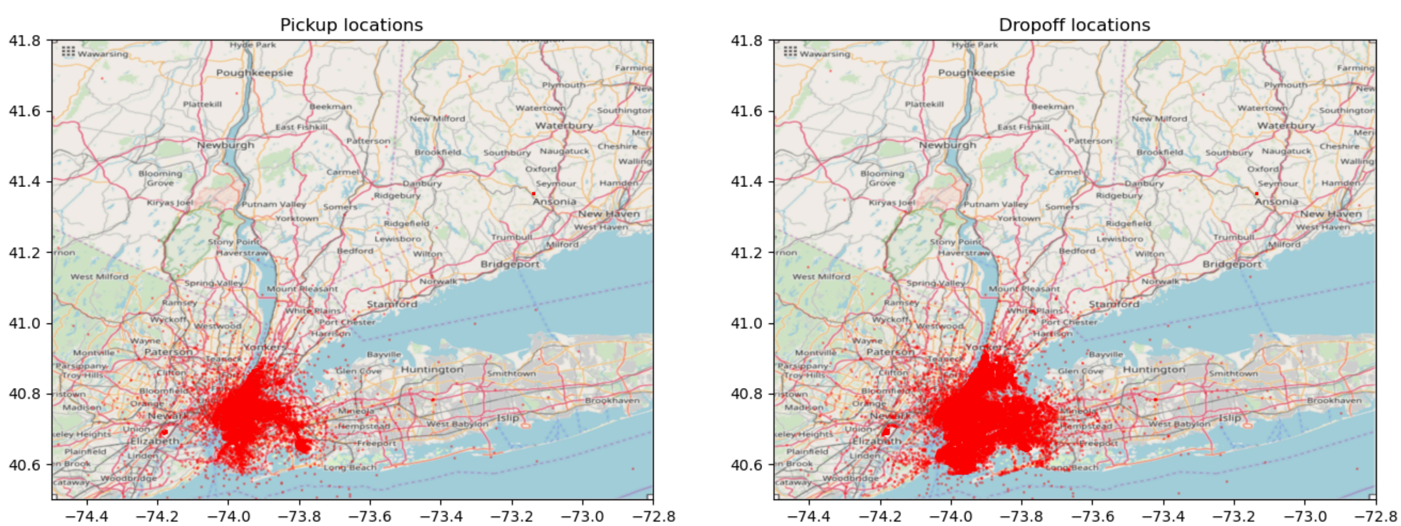
```
/opt/anaconda3/lib/python3.12/site-packages/urllib3/connectionpool.py:1099: InsecureReque
stWarning: Unverified HTTPS request is being made to host 'aiblog.nl'. Adding certificate
verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-
usage.html#tls-warnings
  warnings.warn(
/opt/anaconda3/lib/python3.12/site-packages/urllib3/connectionpool.py:1099: InsecureReque
stWarning: Unverified HTTPS request is being made to host 'aiblog.nl'. Adding certificate
verification is strongly advised. See: https://urllib3.readthedocs.io/en/latest/advanced-
usage.html#tls-warnings
  warnings.warn(
```

In [7]:

```python
# This function will plot pickup and dropoff locations on the NYC map
def plot_on_map(df, BB, nyc_map, s=10, alpha=0.2):
    fig, axs = plt.subplots(1, 2, figsize=(16,10))

    # Plot Pickup locations
    axs[0].scatter(df.pickup_longitude, df.pickup_latitude, zorder=1, alpha=alpha, c='r'
    axs[0].set_xlim((BB[0], BB[1]))
    axs[0].set_ylim((BB[2], BB[3]))
    axs[0].set_title('Pickup locations')
    axs[0].imshow(nyc_map, zorder=0, extent=BB)

    # Plot Dropoff locations
    axs[1].scatter(df.dropoff_longitude, df.dropoff_latitude, zorder=1, alpha=alpha, c='
    axs[1].set_xlim((BB[0], BB[1]))
    axs[1].set_ylim((BB[2], BB[3]))
    axs[1].set_title('Dropoff locations')
    axs[1].imshow(nyc_map, zorder=0, extent=BB)
```

In [8]:

```python
# Plot the filtered training data (pickup and dropoff locations) on the map
plot_on_map(train_df, BB, nyc_map, s=1, alpha=0.3)
```

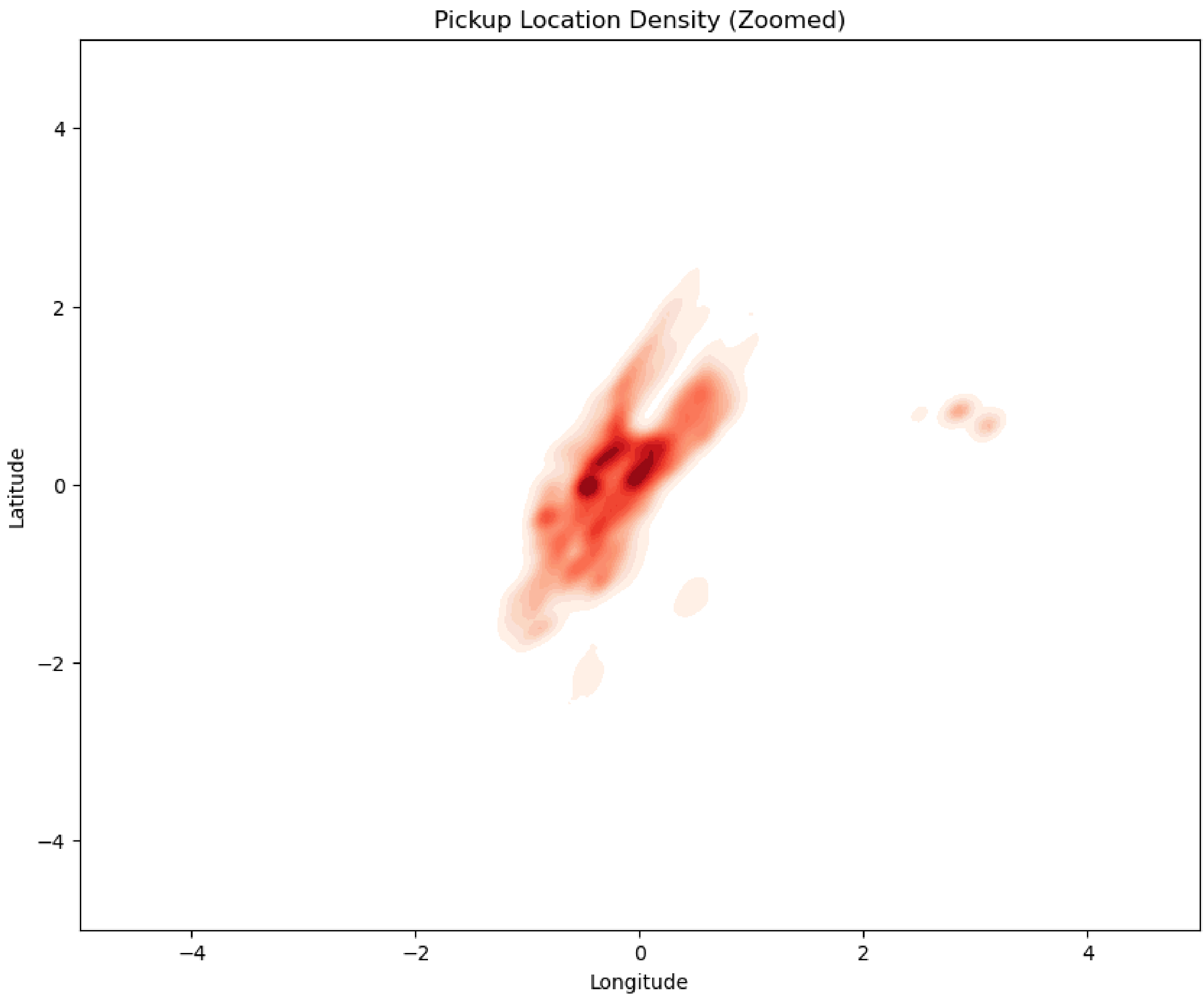


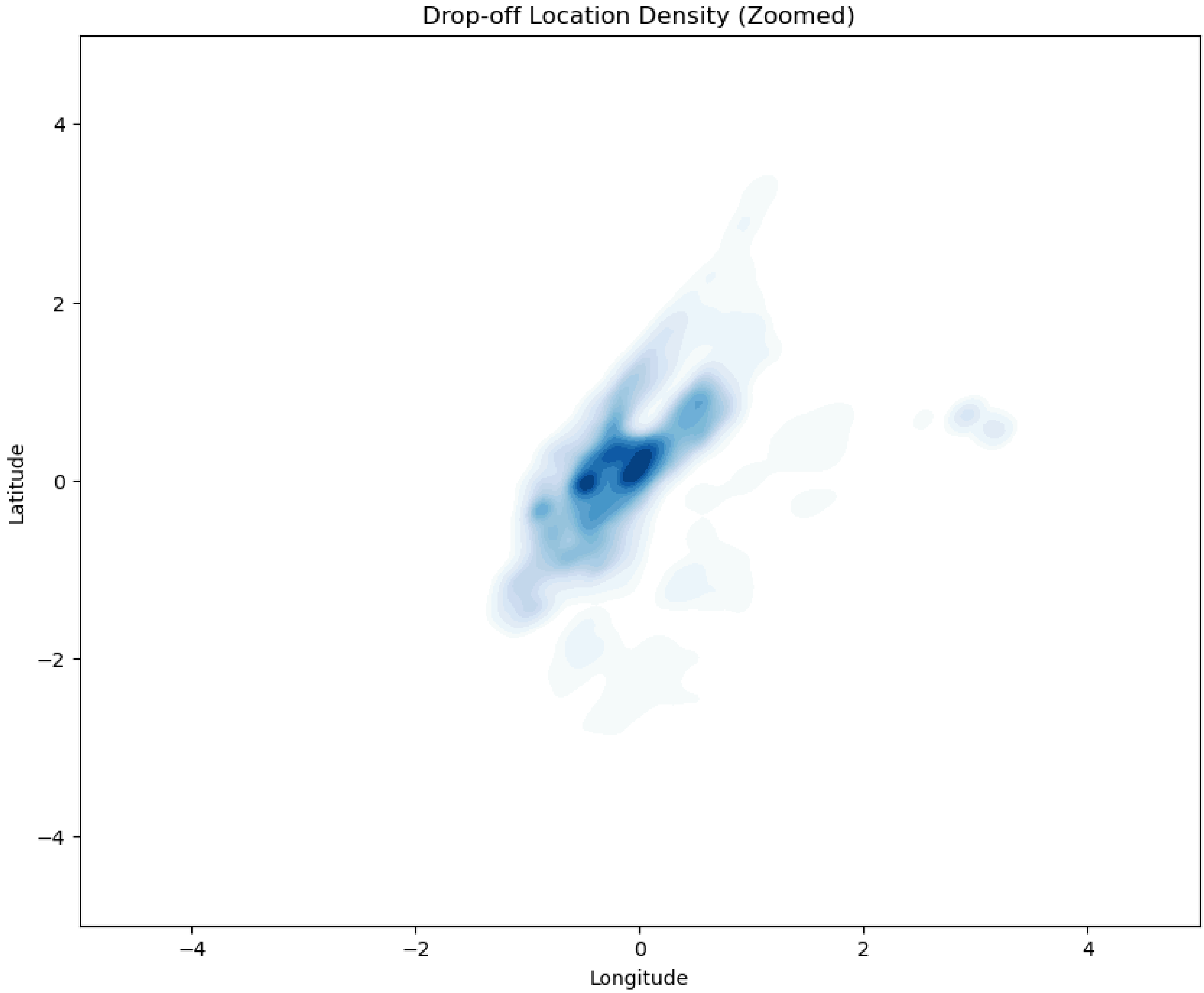
In [9]:

```python
# Geographical Clustering: Use K-means clustering for pickup and drop-off locations
kmeans = KMeans(n_clusters=20, random_state=42)
train_df['pickup_cluster'] = kmeans.fit_predict(train_df[['pickup_latitude', 'pickup_lon
train_df['dropoff_cluster'] = kmeans.fit_predict(train_df[['dropoff_latitude', 'dropoff_
```

In [10]:
```
# Normalize numerical features
scaler = StandardScaler()
train_df[['pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude'
    train_df[['pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latit
```

In [11]:
```
# Filter data based on fare amount (<= 100)
filtered_train_df = train_df[train_df['fare_amount'] <= 100]

# Load your dataset
# Replace 'your_dataset.csv' with the path to your dataset
df = filtered_train_df

# Clean and filter data
df = df.dropna(subset=['pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'drop
df = df[(df['pickup_latitude'].between(-90, 90)) &
        (df['pickup_longitude'].between(-180, 180)) &
        (df['dropoff_latitude'].between(-90, 90)) &
        (df['dropoff_longitude'].between(-180, 180))]

# Further filter data to zoom into the desired range
df = df[(df['pickup_latitude'].between(-5, 5)) &
        (df['pickup_longitude'].between(-5, 5)) &
        (df['dropoff_latitude'].between(-5, 5)) &
        (df['dropoff_longitude'].between(-5, 5))]

# Create a heatmap for pickup locations
plt.figure(figsize=(10, 8))
sns.kdeplot(
    x=df['pickup_longitude'],
    y=df['pickup_latitude'],
    fill=True,
    cmap='Reds',
    levels=20
)
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.title('Pickup Location Density (Zoomed)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()

# Create a heatmap for drop-off locations
plt.figure(figsize=(10, 8))
sns.kdeplot(
    x=df['dropoff_longitude'],
    y=df['dropoff_latitude'],
    fill=True,
    cmap='Blues',
    levels=20
)
plt.xlim(-5, 5)
plt.ylim(-5, 5)
plt.title('Drop-off Location Density (Zoomed)')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.show()
```

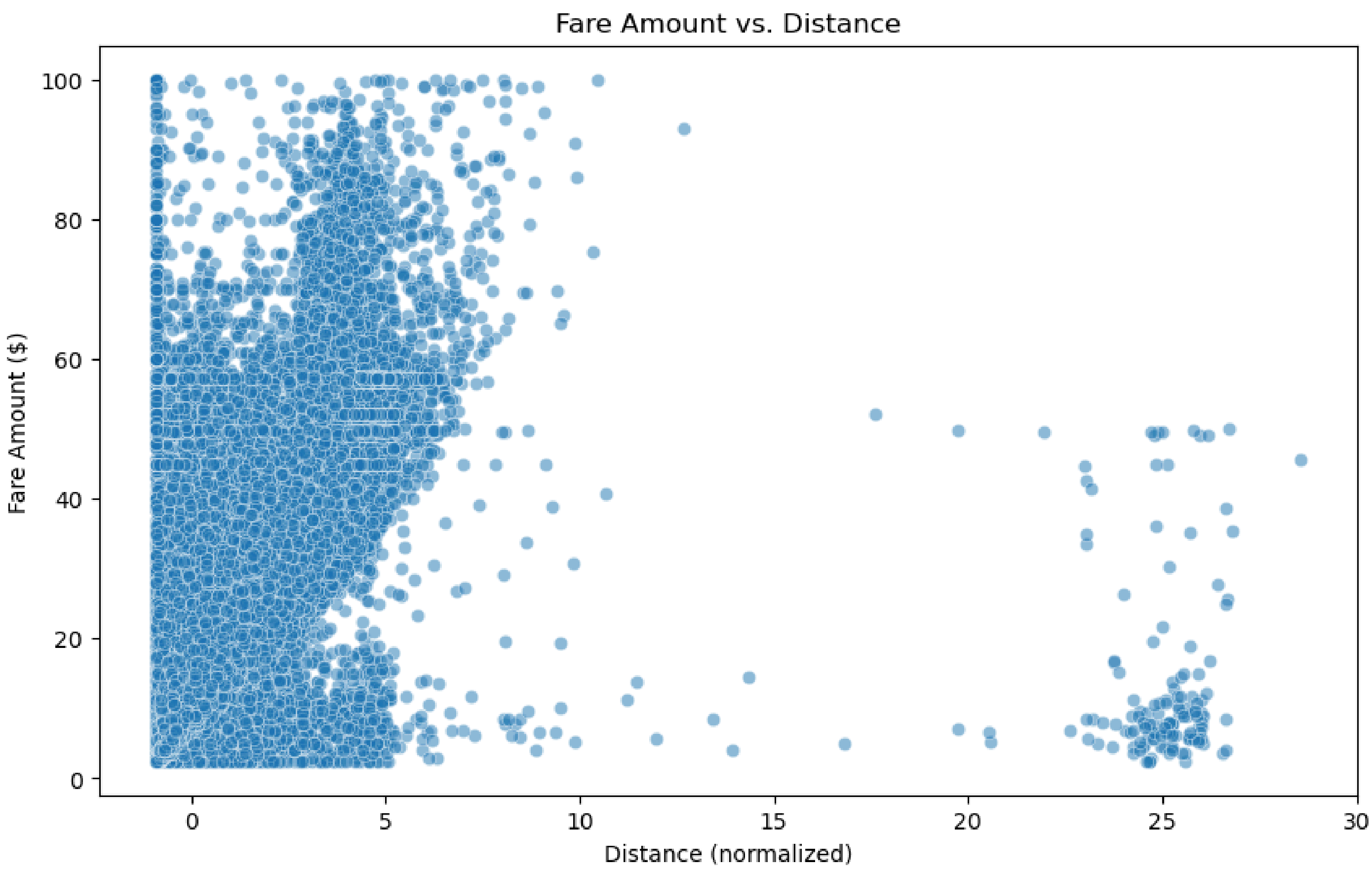

Pickup Location Density (Zoomed)
Latitude
Longitude
4
2
0
−2
−4
−4
−2
0
2
4

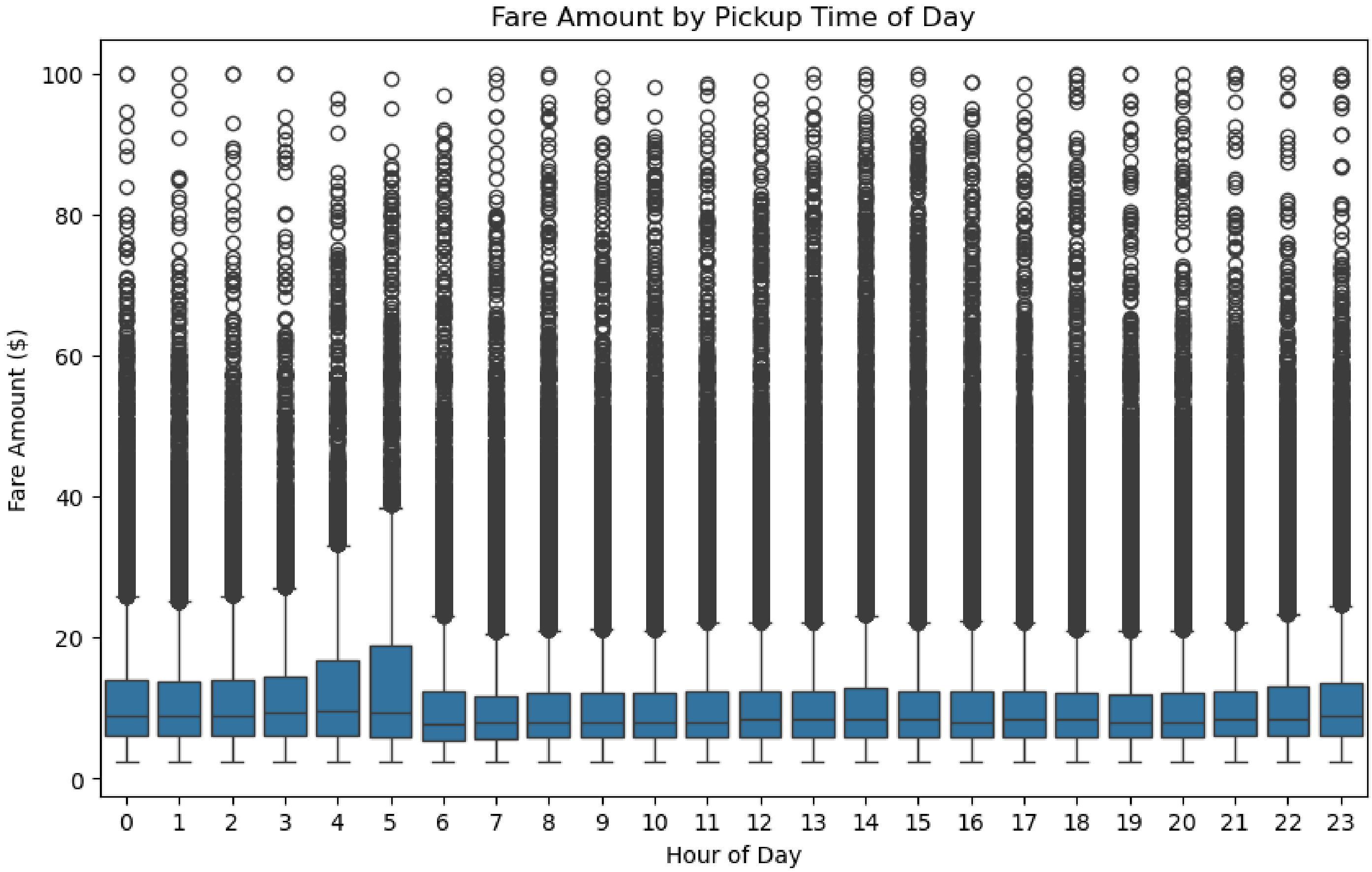


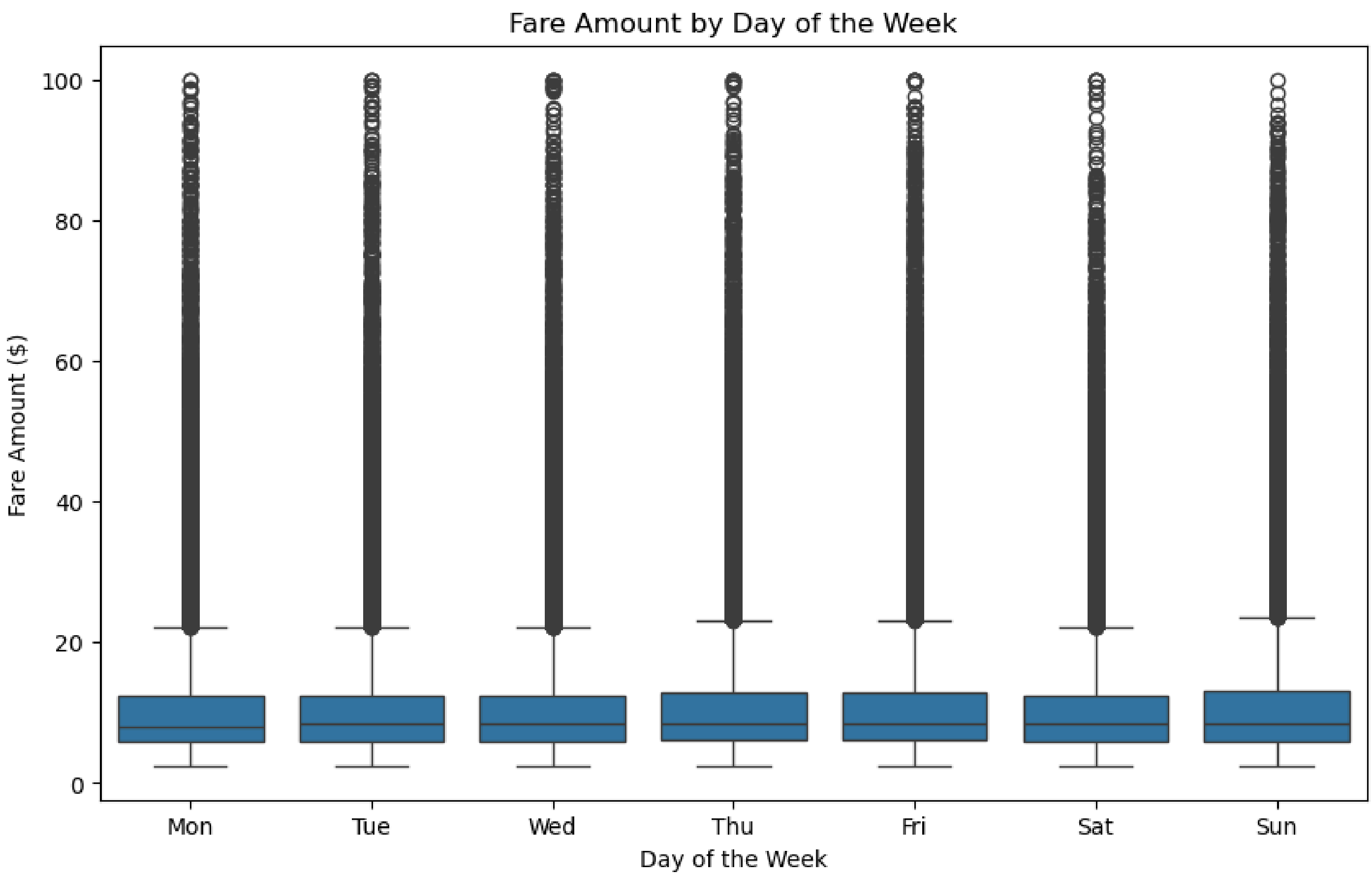
In [12]:
```
# Ensure 'pickup_datetime' is in datetime format in filtered_train_df
filtered_train_df['pickup_datetime'] = pd.to_datetime(filtered_train_df['pickup_datetime

# Extract the hour of day from 'pickup_datetime'
filtered_train_df['hour'] = filtered_train_df['pickup_datetime'].dt.hour

# Extract the day of the week from 'pickup_datetime'
filtered_train_df['day_of_week'] = filtered_train_df['pickup_datetime'].dt.dayofweek  #

# Extract the month from 'pickup_datetime'
filtered_train_df['month'] = filtered_train_df['pickup_datetime'].dt.month  # 1 = Januar

# A. Plot Relationships with Fare Amount

# Fare amount vs. Distance
plt.figure(figsize=(10, 6))
sns.scatterplot(x='distance', y='fare_amount', data=filtered_train_df, alpha=0.5)
plt.title("Fare Amount vs. Distance")
plt.xlabel("Distance (normalized)")
plt.ylabel("Fare Amount ($)")
plt.show()

# Fare amount vs. Pickup Time of Day
plt.figure(figsize=(10, 6))
sns.boxplot(x='hour', y='fare_amount', data=filtered_train_df)
plt.title("Fare Amount by Pickup Time of Day")
```

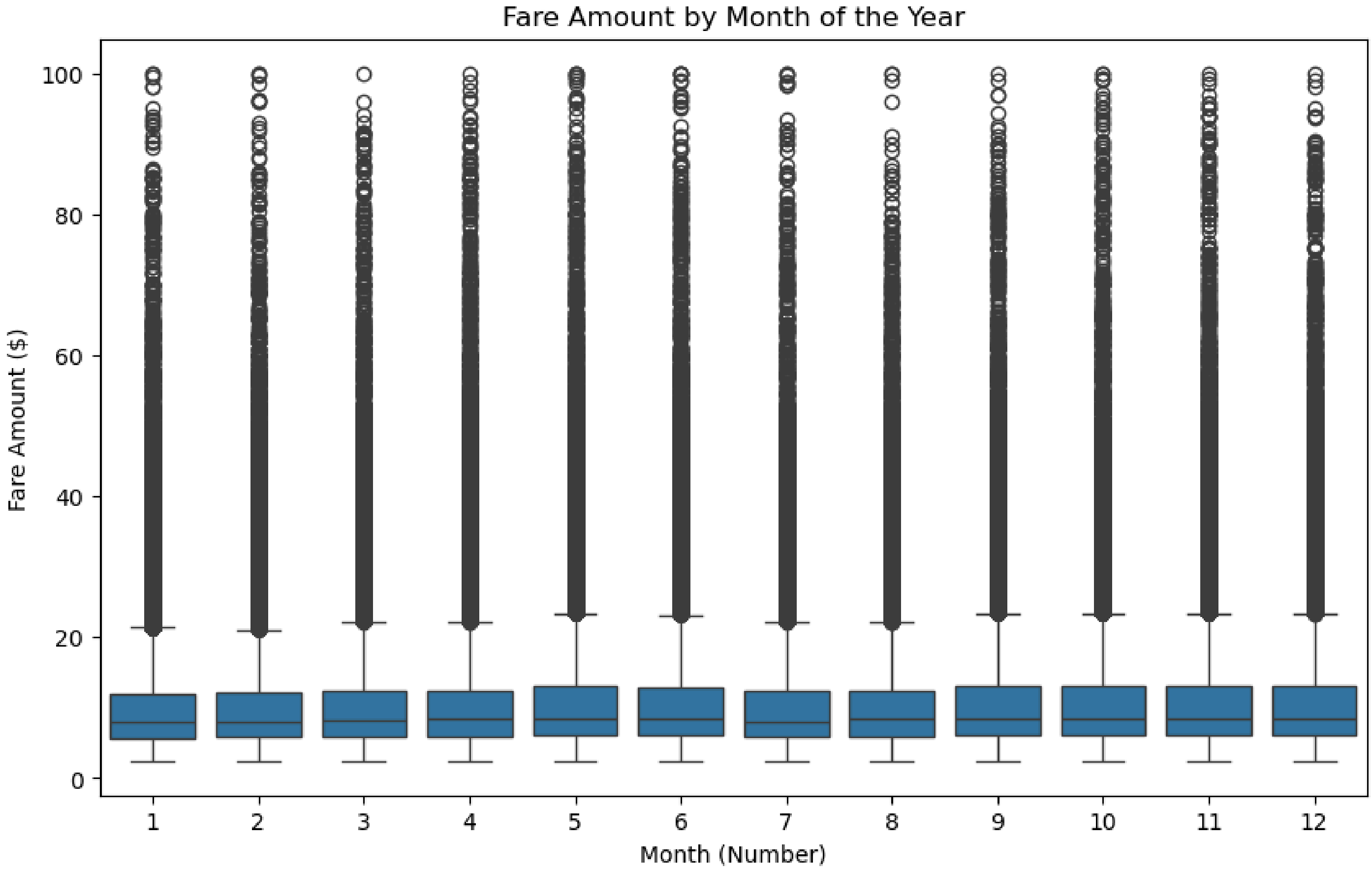
```
plt.xlabel("Hour of Day")
plt.ylabel("Fare Amount ($)")
plt.show()

# Fare amount vs. Day of the Week
plt.figure(figsize=(10, 6))
sns.boxplot(x='day_of_week', y='fare_amount', data=filtered_train_df)
plt.title("Fare Amount by Day of the Week")
plt.xlabel("Day of the Week")
plt.ylabel("Fare Amount ($)")
plt.xticks(ticks=range(7), labels=['Mon', 'Tue', 'Wed', 'Thu', 'Fri', 'Sat', 'Sun'])
plt.show()

# Fare amount vs. Month of the Year (numbered)
plt.figure(figsize=(10, 6))
sns.boxplot(x='month', y='fare_amount', data=filtered_train_df)
plt.title("Fare Amount by Month of the Year")
plt.xlabel("Month (Number)")
plt.ylabel("Fare Amount ($)")
plt.show()

# Fare amount vs. Pickup Cluster
plt.figure(figsize=(10, 6))
sns.boxplot(x='pickup_cluster', y='fare_amount', data=filtered_train_df)
plt.title("Fare Amount by Pickup Cluster")
plt.xlabel("Pickup Cluster")
plt.ylabel("Fare Amount ($)")
plt.show()

# Fare amount vs. Dropoff Cluster
plt.figure(figsize=(10, 6))
sns.boxplot(x='dropoff_cluster', y='fare_amount', data=filtered_train_df)
plt.title("Fare Amount by Dropoff Cluster")
plt.xlabel("Dropoff Cluster")
plt.ylabel("Fare Amount ($)")
plt.show()
```

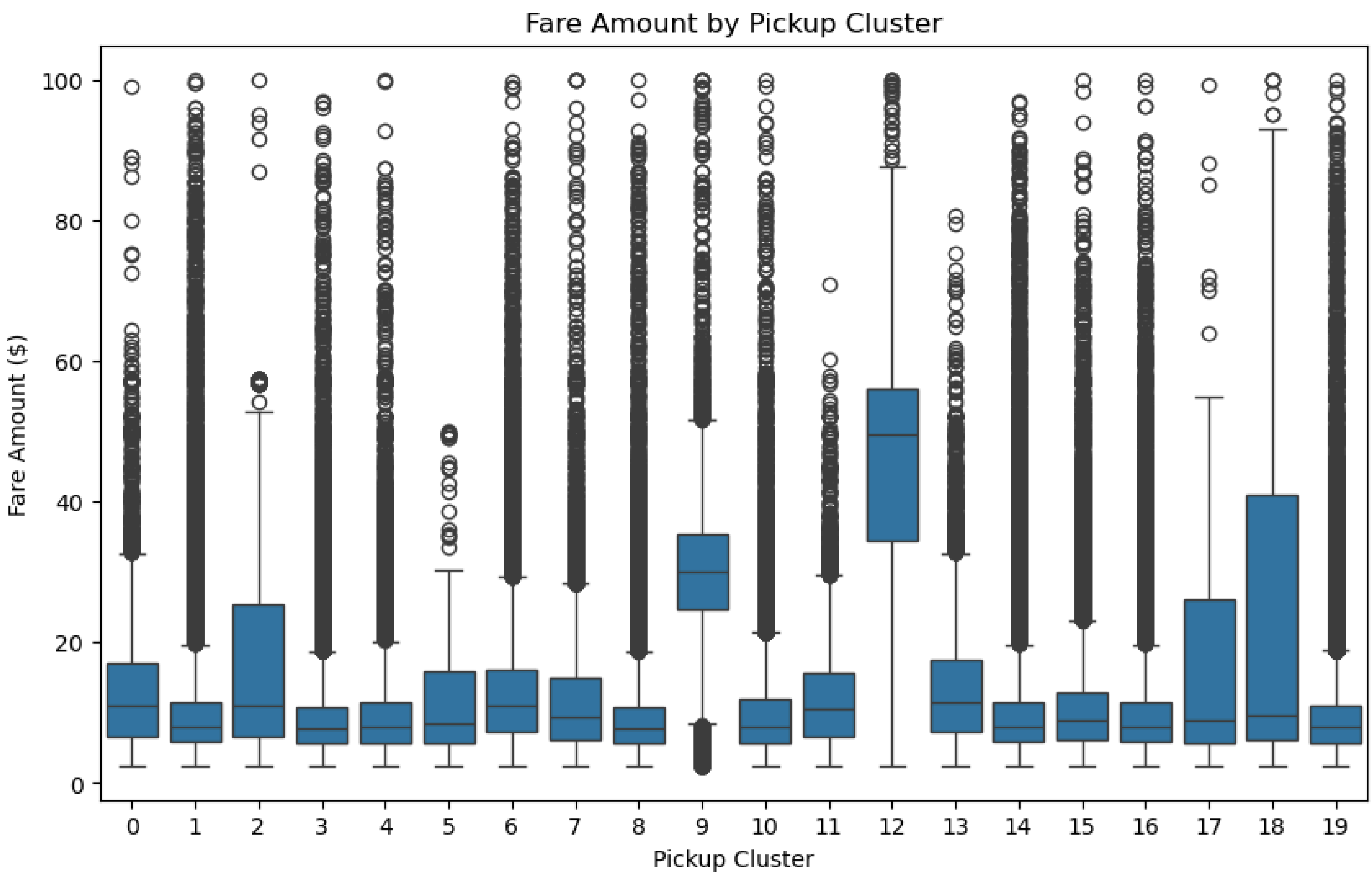
```
/var/folders/w0/rfr3qgwd3y7_5shnvp41s4400000gn/T/ipykernel_4237/787997382.py:2: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  filtered_train_df['pickup_datetime'] = pd.to_datetime(filtered_train_df['pickup_datetim
e'])
/var/folders/w0/rfr3qgwd3y7_5shnvp41s4400000gn/T/ipykernel_4237/787997382.py:5: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  filtered_train_df['hour'] = filtered_train_df['pickup_datetime'].dt.hour
/var/folders/w0/rfr3qgwd3y7_5shnvp41s4400000gn/T/ipykernel_4237/787997382.py:8: SettingWi
thCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  filtered_train_df['day_of_week'] = filtered_train_df['pickup_datetime'].dt.dayofweek  #
Monday=0, Sunday=6
/var/folders/w0/rfr3qgwd3y7_5shnvp41s4400000gn/T/ipykernel_4237/787997382.py:11: SettingW
ithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_g
uide/indexing.html#returning-a-view-versus-a-copy
  filtered_train_df['month'] = filtered_train_df['pickup_datetime'].dt.month  # 1 = Janua
ry, 12 = December
```

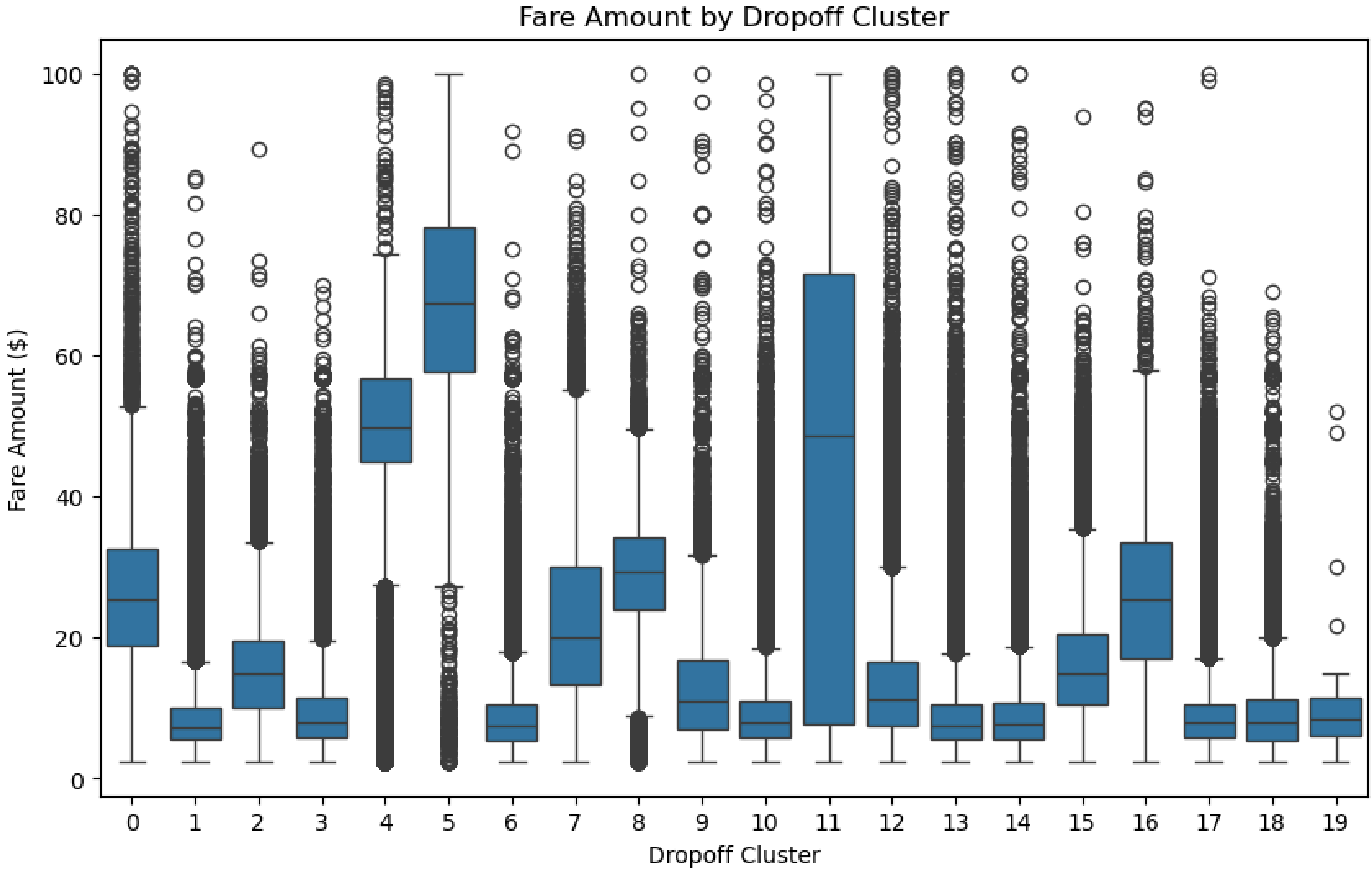



Fare Amount by Pickup Time of Day
Fare Amount ($)
Hour of Day
0
20
40
60
80
100
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23


Fare Amount by Day of the Week
Fare Amount ($)
Day of the Week
0
20
40
60
80
100
Mon
Tue
Wed
Thu
Fri
Sat
Sun


Fare Amount by Month of the Year
Fare Amount ($)
Month (Number)
0
20
40
60
80
100
1
2
3
4
5
6
7
8
9
10
11
12


Fare Amount by Pickup Cluster
Fare Amount ($)
Pickup Cluster
0
20
40
60
80
100
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19



In [13]:
```
# Create a table with average fare for each hour of the day
hourly_avg_fare = filtered_train_df.groupby('hour')['fare_amount'].mean().reset_index()
hourly_avg_fare.columns = ['Hour', 'Average Fare ($)']
print("Average Fare by Hour of Day:")
print(hourly_avg_fare)

# Create a table with average fare for each month
monthly_avg_fare = filtered_train_df.groupby('month')['fare_amount'].mean().reset_index(
monthly_avg_fare.columns = ['Month', 'Average Fare ($)']
print("\nAverage Fare by Month:")
print(monthly_avg_fare)

# Create a table with average fare for each week of the year
weekly_avg_fare = filtered_train_df.groupby('day_of_week')['fare_amount'].mean().reset_i
weekly_avg_fare.columns = ['Week', 'Average Fare ($)']
print("\nAverage Fare by Week:")
print(weekly_avg_fare)
```

```
Average Fare by Hour of Day:
    Hour  Average Fare ($)
0      0         11.650438
1      1         11.386619
2      2         11.356424
3      3         11.865588
4      4         13.477872
5      5         15.263310
6      6         12.154306
7      7         10.969852
8      8         10.873093
9      9         10.809943
10    10         10.897088
11    11         11.072096
12    12         11.106443
13    13         11.556973
14    14         11.820748
15    15         11.954274
16    16         11.770322
17    17         11.359739
18    18         10.911689
19    19         10.528989
20    20         10.742268
21    21         10.946755
22    22         11.256692
23    23         11.526577

Average Fare by Month:
    Month  Average Fare ($)
0       1         10.698151
1       2         10.841487
2       3         11.080444
3       4         11.240290
4       5         11.568795
5       6         11.490351
6       7         11.085911
7       8         11.181410
8       9         11.691943
9      10         11.608222
10     11         11.526990
11     12         11.583679

Average Fare by Week:
   Week  Average Fare ($)
0     0         11.332052
1     1         11.173271
2     2         11.227556
3     3         11.448782
4     4         11.344287
5     5         10.975522
6     6         11.581182
```

In [14]:

```
# Ensure 'pickup_datetime' is in datetime format in train_df
train_df['pickup_datetime'] = pd.to_datetime(train_df['pickup_datetime'])

# Extract the hour of day from 'pickup_datetime'
train_df['hour'] = train_df['pickup_datetime'].dt.hour

# Extract the day of the week from 'pickup_datetime'
train_df['day_of_week'] = train_df['pickup_datetime'].dt.dayofweek  # Monday=0, Sunday=6

# Extract the month from 'pickup_datetime'
```

```
train_df['month'] = train_df['pickup_datetime'].dt.month  # 1 = January, 12 = December

# Split the data into train and test
train_df, test_df = train_test_split(train_df, test_size=0.2, random_state=42)

# Prepare features and target for the train data
features = ['pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitud
X_train = train_df[features]
y_train = train_df['fare_amount']

# Prepare features for the test data (no target in test)
X_test = test_df[features]
y_test = test_df['fare_amount']

# Define the neural network model
model = tf.keras.models.Sequential([
    tf.keras.layers.InputLayer(input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dense(1)  # Output layer for regression
])

# Compile the model
model.compile(optimizer='adam', loss='mean_absolute_error', metrics=['mae'])

# Define callbacks for early stopping and learning rate scheduling
early_stopping = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=5, restor
lr_scheduler = tf.keras.callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.5, pati

# Train the model
history = model.fit(
    X_train, y_train,
    validation_data=(X_test, y_test),
    epochs=10,
    batch_size=256,
    callbacks=[early_stopping, lr_scheduler]
)
```

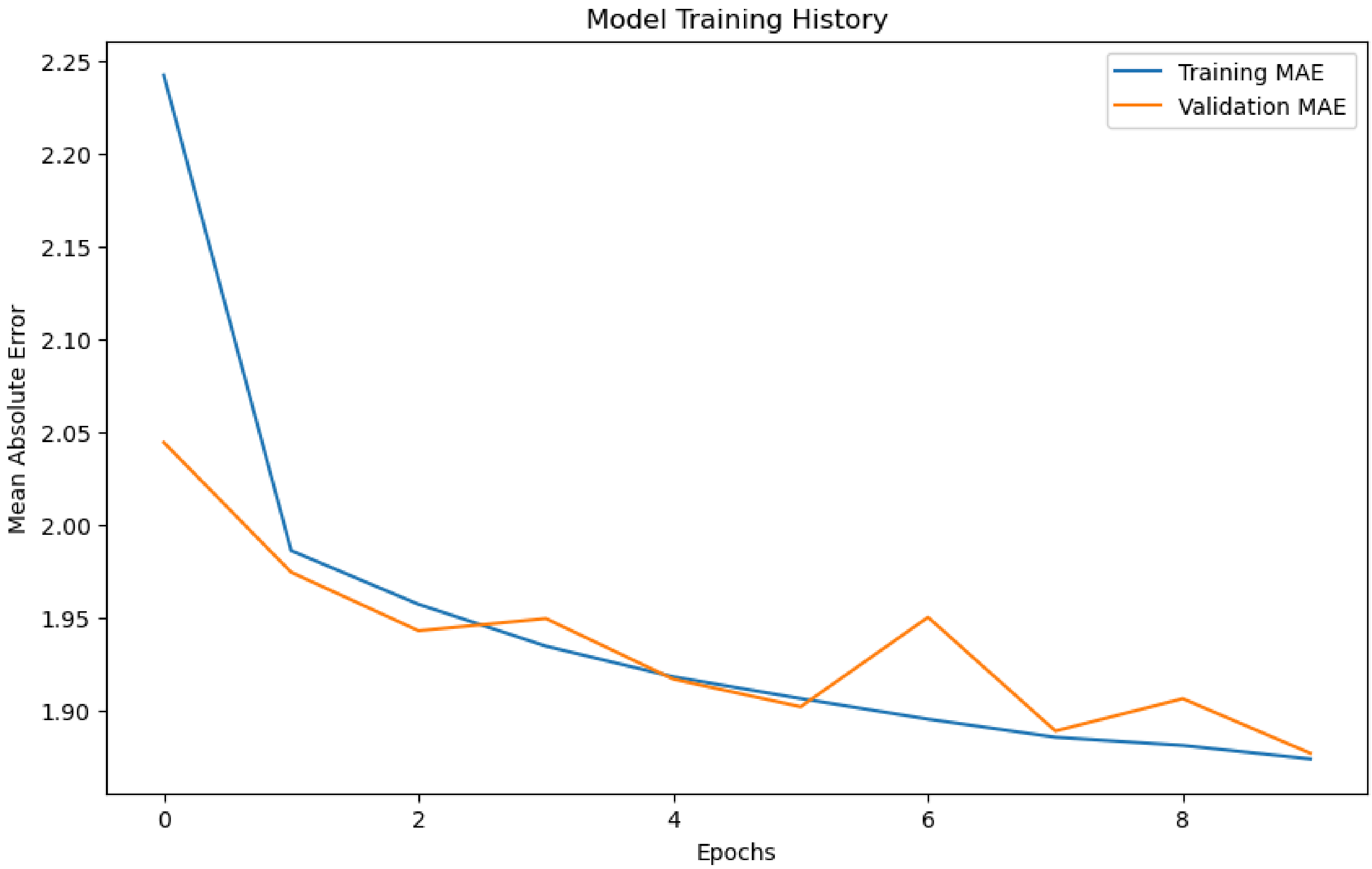
```
/opt/anaconda3/lib/python3.12/site-packages/keras/src/layers/core/input_layer.py:26: User
Warning: Argument `input_shape` is deprecated. Use `shape` instead.
  warnings.warn(
```

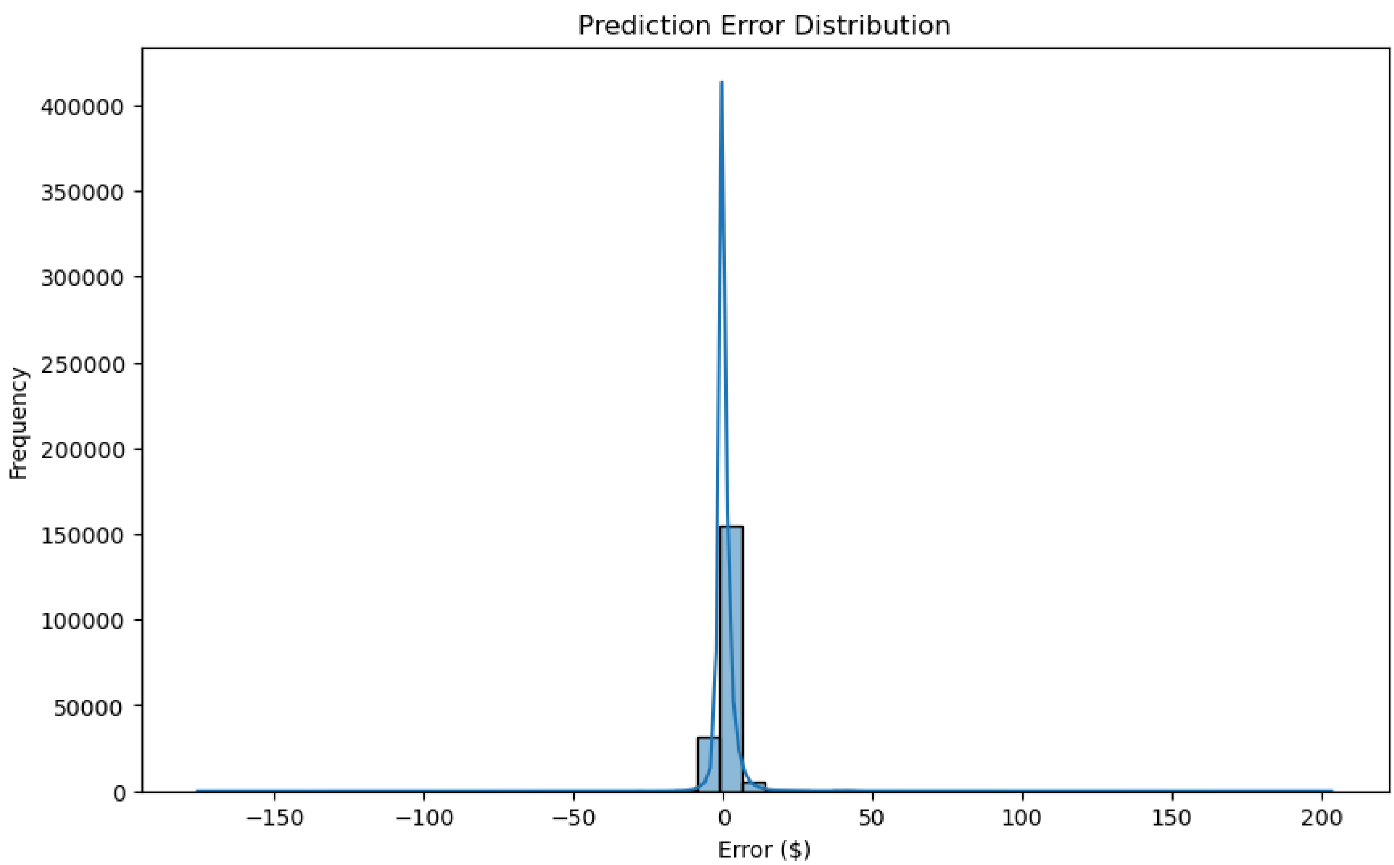
```
Epoch 1/10
3027/3027 ━━━━━━━━━━━━━━━━━━━━ 5s 1ms/step - loss: 2.7798 - mae: 2.7798 - val_loss: 2.044
1 - val_mae: 2.0441 - learning_rate: 0.0010
Epoch 2/10
3027/3027 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 2.0010 - mae: 2.0010 - val_loss: 1.974
2 - val_mae: 1.9742 - learning_rate: 0.0010
Epoch 3/10
3027/3027 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 1.9609 - mae: 1.9609 - val_loss: 1.942
6 - val_mae: 1.9426 - learning_rate: 0.0010
Epoch 4/10
3027/3027 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 1.9354 - mae: 1.9354 - val_loss: 1.949
2 - val_mae: 1.9492 - learning_rate: 0.0010
Epoch 5/10
3027/3027 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 1.9248 - mae: 1.9248 - val_loss: 1.916
5 - val_mae: 1.9165 - learning_rate: 0.0010
Epoch 6/10
3027/3027 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 1.9142 - mae: 1.9142 - val_loss: 1.901
7 - val_mae: 1.9017 - learning_rate: 0.0010
Epoch 7/10
3027/3027 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 1.8856 - mae: 1.8856 - val_loss: 1.949
9 - val_mae: 1.9499 - learning_rate: 0.0010
Epoch 8/10
3027/3027 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 1.8829 - mae: 1.8829 - val_loss: 1.888
6 - val_mae: 1.8886 - learning_rate: 0.0010
Epoch 9/10
3027/3027 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 1.8851 - mae: 1.8851 - val_loss: 1.906
0 - val_mae: 1.9060 - learning_rate: 0.0010
Epoch 10/10
3027/3027 ━━━━━━━━━━━━━━━━━━━━ 4s 1ms/step - loss: 1.8703 - mae: 1.8703 - val_loss: 1.876
6 - val_mae: 1.8766 - learning_rate: 0.0010
```

In [15]:

```python
# Evaluate on the test data
test_predictions = model.predict(X_test)

# Calculate the evaluation metrics on test data
mae_test = mean_absolute_error(y_test, test_predictions)
mse_test = mean_squared_error(y_test, test_predictions)
rmse_test = np.sqrt(mse_test)

print(f"Test Mean Absolute Error: {mae_test}")
print(f"Test Mean Squared Error: {mse_test}")
print(f"Test Root Mean Squared Error: {rmse_test}")

# Plot training history
plt.figure(figsize=(10, 6))
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.title("Model Training History")
plt.xlabel("Epochs")
plt.ylabel("Mean Absolute Error")
plt.legend()
plt.show()

# Visualize the prediction errors
plt.figure(figsize=(10, 6))
sns.histplot(y_test - test_predictions.flatten(), bins=50, kde=True)
plt.title("Prediction Error Distribution")
plt.xlabel("Error ($)")
plt.ylabel("Frequency")
plt.show()
```

```
6054/6054 ━━━━━━━━━━━━━━━━━━━━ 4s 577us/step
Test Mean Absolute Error: 1.8766108148870917
Test Mean Squared Error: 18.022198463288888
Test Root Mean Squared Error: 4.2452559950241975
```





In [ ]: