TravelTide Exploration

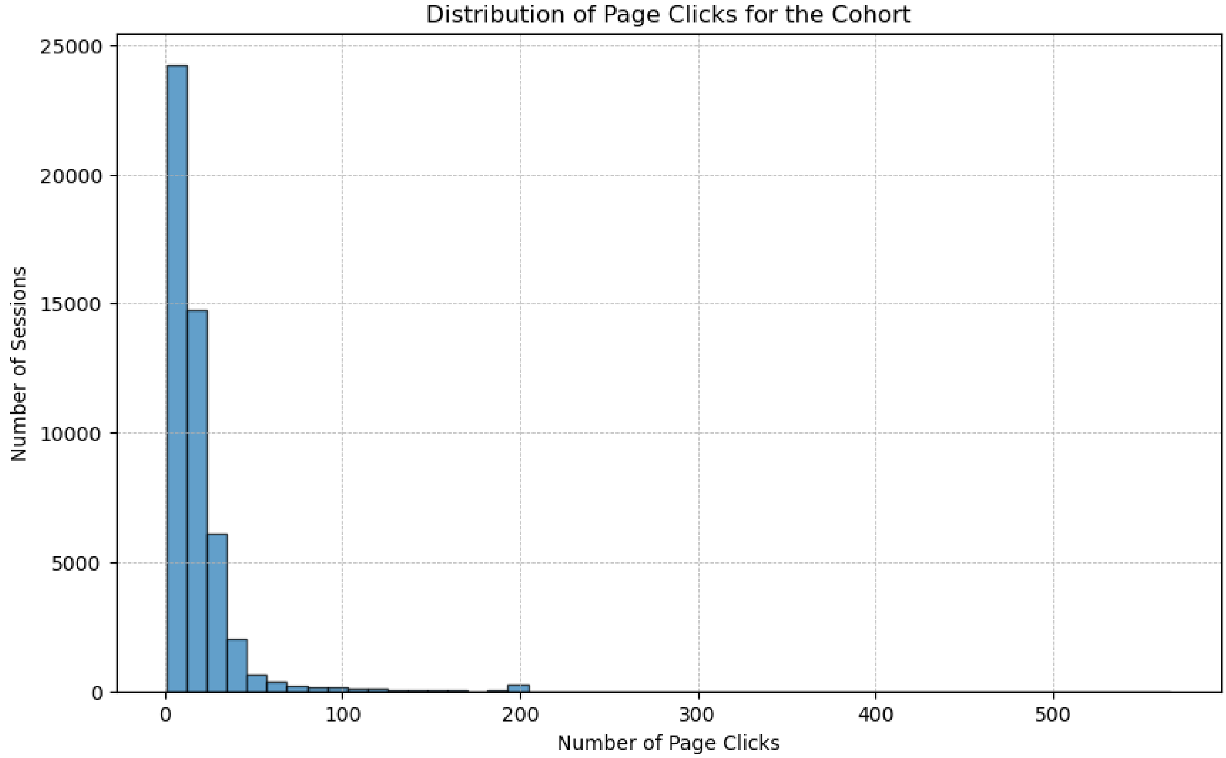
In [1]:
```
from sqlalchemy import create_engine

# Create a connection using SQLAlchemy
DATABASE_URL = "postgresql+psycopg2://Test:bQNxVzJL4g6u@ep-noisy-flower-846766.us-east
engine = create_engine(DATABASE_URL)
```

In [2]:
```
# Cohort filter definition
cohort_filter = """
WITH CohortUsers AS (
    SELECT user_id
    FROM sessions
    WHERE session_start > '2023-01-04'
    GROUP BY user_id
    HAVING COUNT(session_id) > 7
)
"""
```

In [3]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

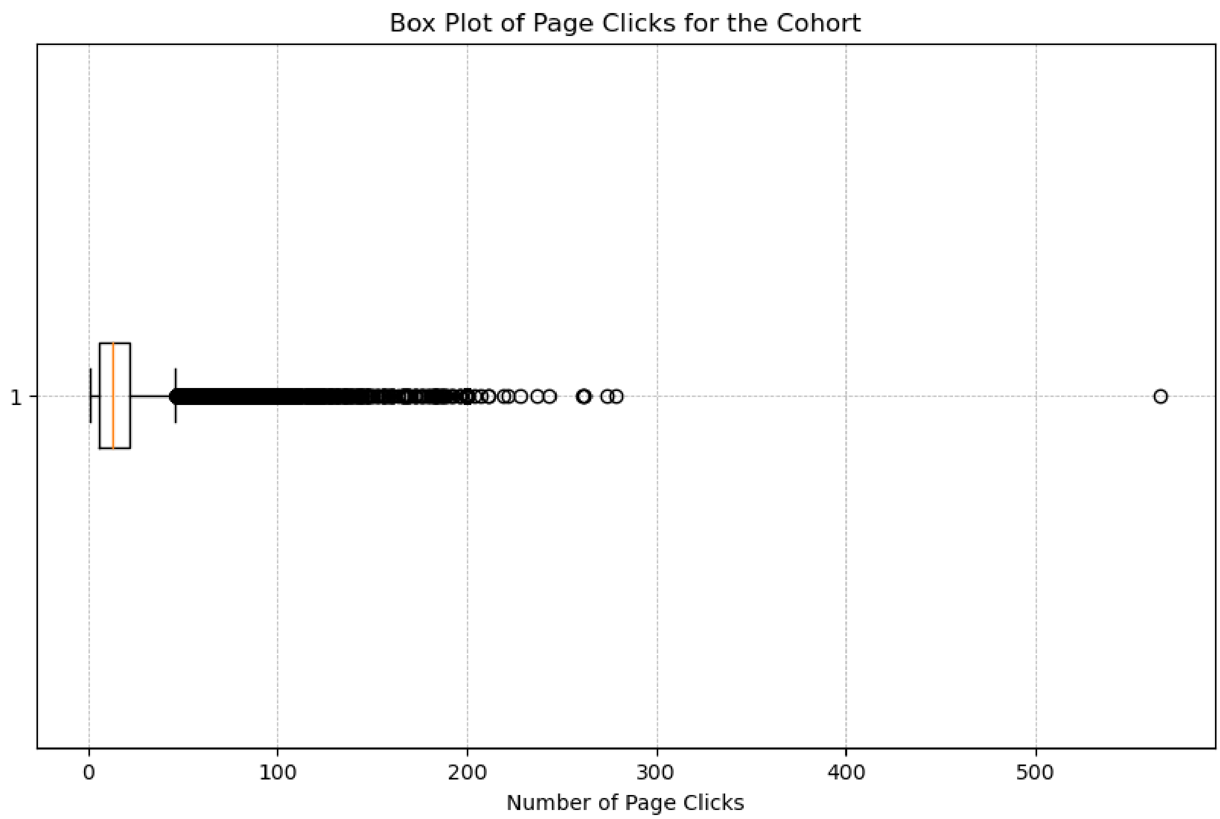
In [4]:
```
# Page Clicks

query = f"""
{cohort_filter}
SELECT page_clicks
FROM sessions
WHERE session_start > '2023-01-04' AND user_id IN (SELECT user_id FROM CohortUsers)
"""

# Fetch the data
df_page_clicks = pd.read_sql(query, engine)
```

In [5]:
```
# Histogram for Page clicks

plt.figure(figsize=(10, 6))
plt.hist(df_page_clicks['page_clicks'], bins=50, edgecolor='k', alpha=0.7)
plt.title('Distribution of Page Clicks for the Cohort')
plt.xlabel('Number of Page Clicks')
plt.ylabel('Number of Sessions')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```



In [6]:

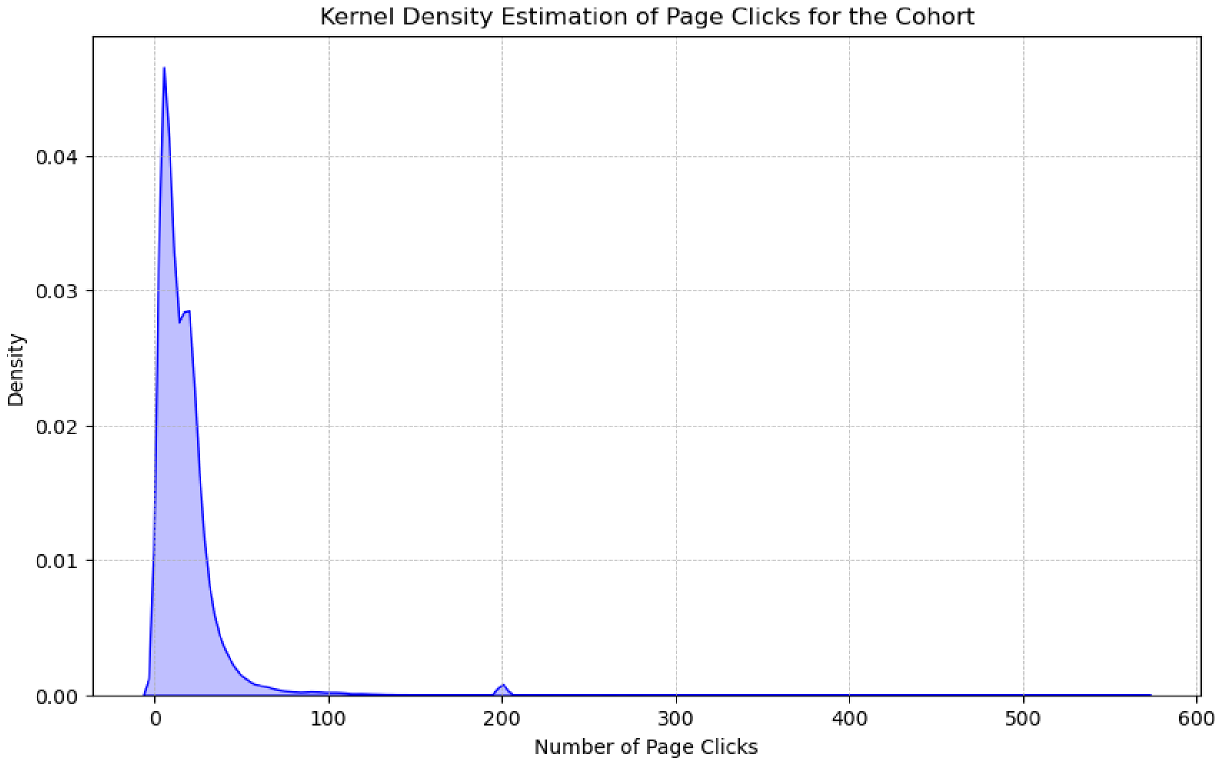
```
# Box Plot for Page Clicks

plt.figure(figsize=(10, 6))
plt.boxplot(df_page_clicks['page_clicks'], vert=False)
plt.title('Box Plot of Page Clicks for the Cohort')
plt.xlabel('Number of Page Clicks')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```



In [7]:
```
# Number of Outliers in Page clicks

# Calculate Q1, Q3 and IQR
Q1 = df_page_clicks['page_clicks'].quantile(0.25)
Q3 = df_page_clicks['page_clicks'].quantile(0.75)
IQR = Q3 - Q1

# Determine boundaries
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Filter the dataframe for outliers and count them
outliers = df_page_clicks[(df_page_clicks['page_clicks'] < lower_bound) | (df_page_cli
num_outliers = outliers.shape[0]

# Total number of data points
total_data_points = len(df_page_clicks)

# Calculate the percentage
percentage_outliers = (num_outliers / total_data_points) * 100

print(f"Number of outliers: {num_outliers}")
print(f"Percentage of outliers: {percentage_outliers}")
```

```
Number of outliers: 2103
Percentage of outliers: 4.273434801162342
```

In [8]:
```
# KDE Plot for Page Clicks
import seaborn as sns

plt.figure(figsize=(10, 6))
sns.kdeplot(df_page_clicks['page_clicks'], fill=True, color='blue')
plt.title('Kernel Density Estimation of Page Clicks for the Cohort')
plt.xlabel('Number of Page Clicks')
plt.ylabel('Density')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```



In [9]:
```
# Skewness and Kurtosis
from scipy.stats import skew, kurtosis

skewness = skew(df_page_clicks['page_clicks'])
kurt = kurtosis(df_page_clicks['page_clicks'])
print(f"Skewness: {skewness}")
print(f"Kurtosis: {kurt}")
```

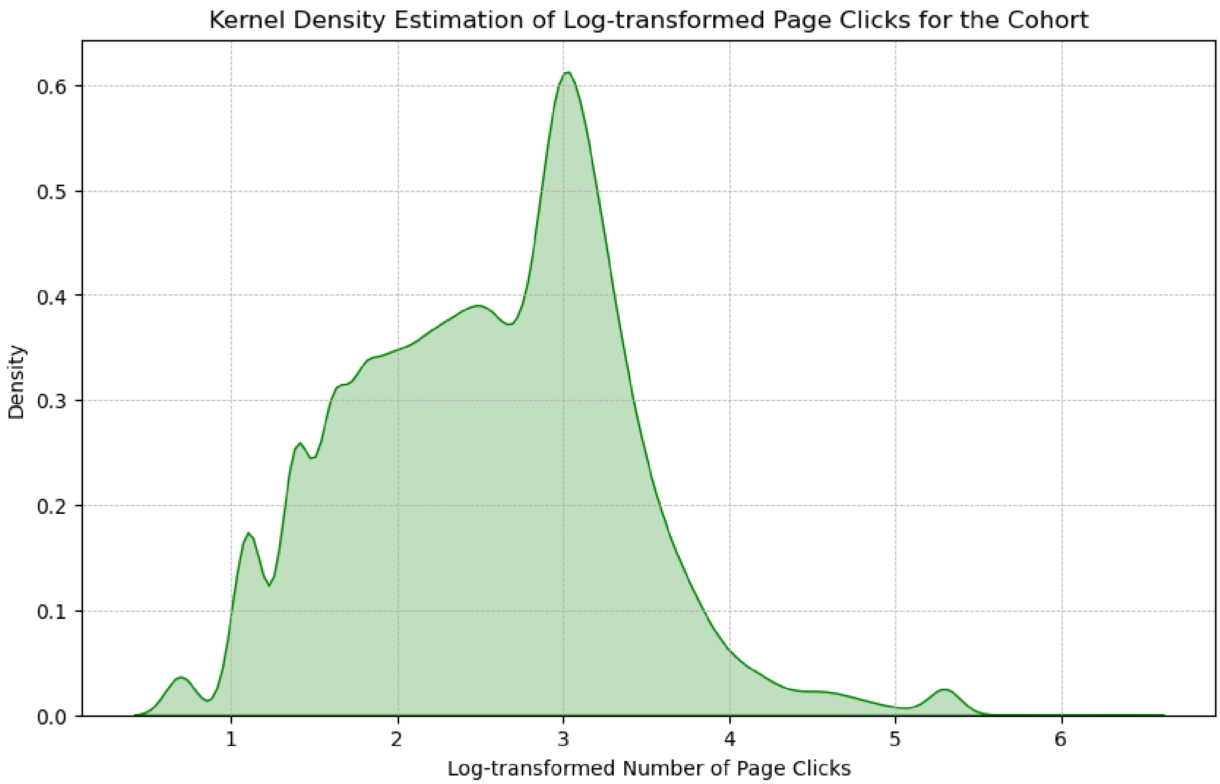
```
Skewness: 5.3902993496060505
Kurtosis: 44.364076329644234
```

In [10]:
```
# Descriptive Statistics
print(df_page_clicks['page_clicks'].describe())
```

```
count    49211.000000
mean        17.588791
std         21.495987
min          1.000000
25%          6.000000
50%         13.000000
75%         22.000000
max        566.000000
Name: page_clicks, dtype: float64
```

In [11]:
```
# If skewness is significantly different from 0, consider transformations

import numpy as np

if abs(skewness) > 1:
    # Log transformation (adding 1 to handle zero values)
    df_page_clicks['log_page_clicks'] = df_page_clicks['page_clicks'].apply(lambda x:
    plt.figure(figsize=(10, 6))
    sns.kdeplot(df_page_clicks['log_page_clicks'], fill=True, color='green')
    plt.title('Kernel Density Estimation of Log-transformed Page Clicks for the Cohort
    plt.xlabel('Log-transformed Number of Page Clicks')
```

```
    plt.ylabel('Density')
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    plt.show()
```

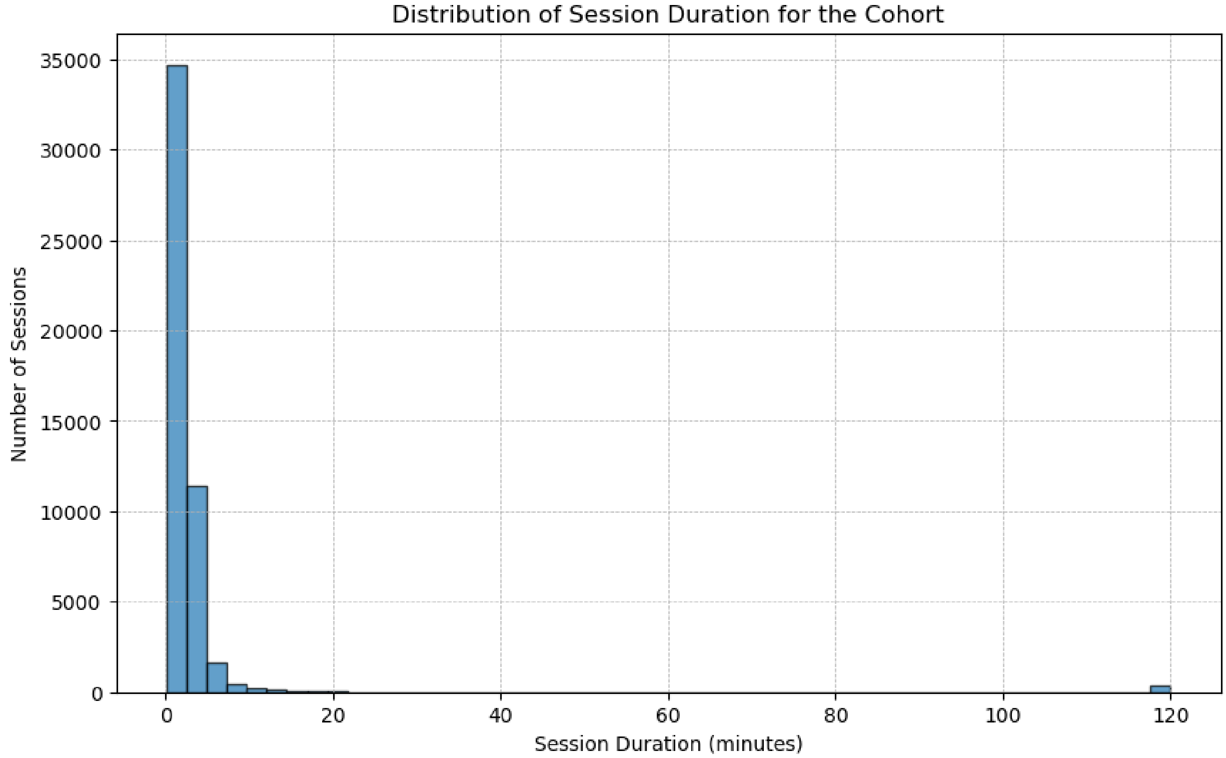


In [12]:

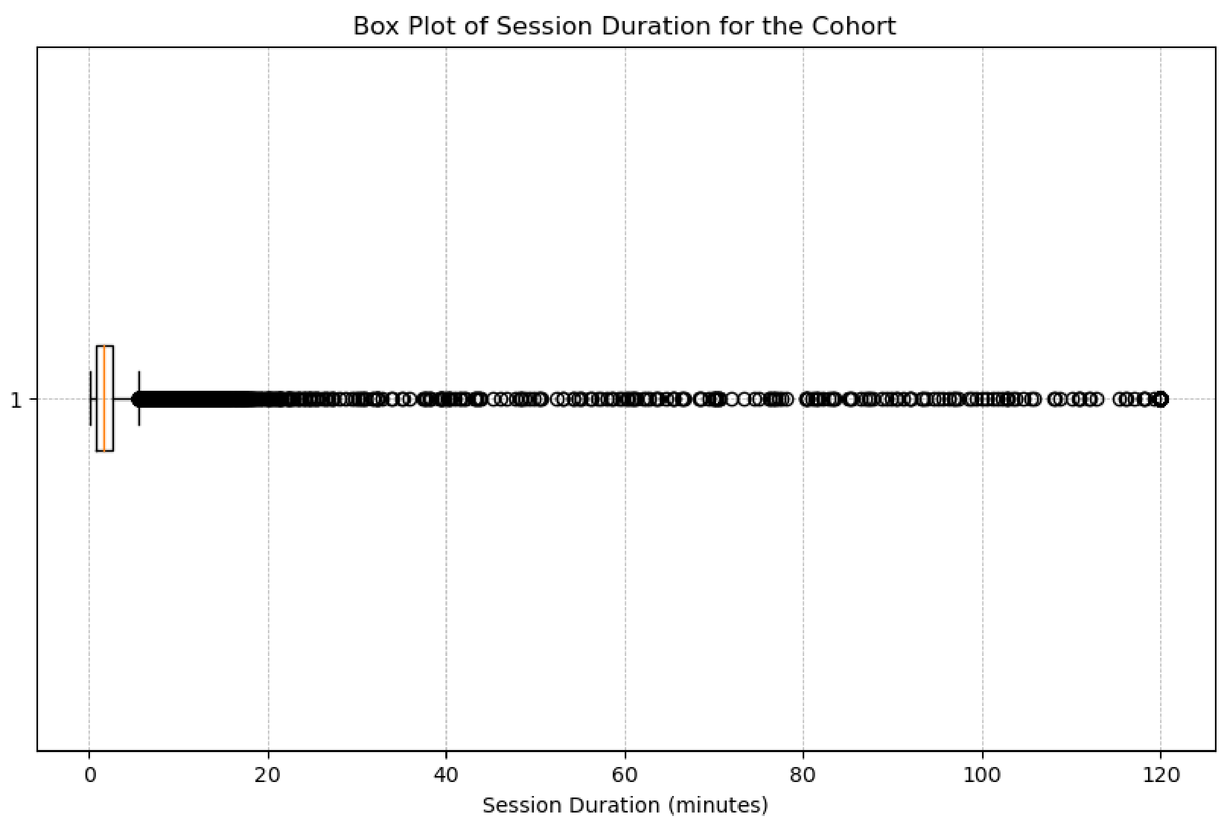
```
# Let's get the Session Duration now

query_duration = f"""
{cohort_filter}
SELECT session_start, session_end
FROM sessions
WHERE session_start > '2023-01-04' AND user_id IN (SELECT user_id FROM CohortUsers)
"""

# Fetch the data
df_session_times = pd.read_sql(query_duration, engine)

# Calculate session duration
df_session_times['session_duration'] = (df_session_times['session_end'] - df_session_t
```

In [13]:

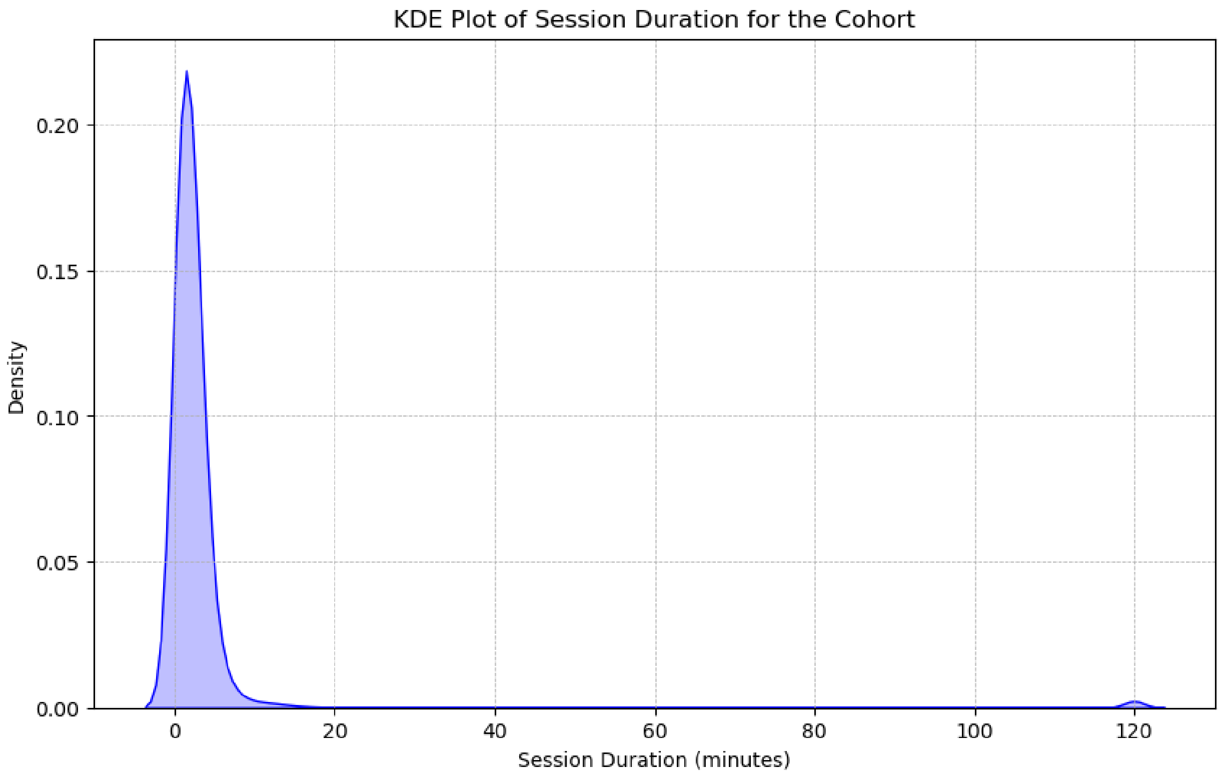
```
# Histogram for duration

plt.figure(figsize=(10, 6))
plt.hist(df_session_times['session_duration'], bins=50, edgecolor='k', alpha=0.7)
plt.title('Distribution of Session Duration for the Cohort')
plt.xlabel('Session Duration (minutes)')
plt.ylabel('Number of Sessions')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```



In [14]:
```
# Box Plot for Session Duration

plt.figure(figsize=(10, 6))
plt.boxplot(df_session_times['session_duration'], vert=False)
plt.title('Box Plot of Session Duration for the Cohort')
plt.xlabel('Session Duration (minutes)')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```



In [15]:
```
# KDE Plot for Session Duration

plt.figure(figsize=(10, 6))
sns.kdeplot(df_session_times['session_duration'], fill=True, color='blue')
plt.title('KDE Plot of Session Duration for the Cohort')
plt.xlabel('Session Duration (minutes)')
plt.ylabel('Density')
plt.grid(True, which='both', linestyle='--', linewidth=0.5)
plt.show()
```



In [16]:
```
# Number of Outliers in Session Duration

# Calculate Q1, Q3 and IQR
Q1_duration = df_session_times['session_duration'].quantile(0.25)
Q3_duration = df_session_times['session_duration'].quantile(0.75)
IQR_duration = Q3_duration - Q1_duration

# Determine boundaries
lower_bound_duration = Q1_duration - 1.5 * IQR_duration
upper_bound_duration = Q3_duration + 1.5 * IQR_duration

# Filter the dataframe for outliers and count them
outliers_duration = df_session_times[(df_session_times['session_duration'] < lower_bou
num_outliers_duration = outliers_duration.shape[0]

# Total number of data points
total_data_points_duration = len(df_session_times)

# Calculate the percentage
percentage_outliers_duration = (num_outliers_duration / total_data_points_duration) *

print(f'Number of outliers in session duration: {num_outliers_duration}')
print(f'Percentage of outliers in session duration: {percentage_outliers_duration}')
```

```
Number of outliers in session duration: 2332
Percentage of outliers in session duration: 4.738777915506696
```

Let's spend some time looking at users

In [17]:

```
# SQL query to find the number of unique users in the cohort
query_unique_users_in_cohort = f"""
{cohort_filter}
SELECT COUNT(DISTINCT user_id) as unique_users
FROM CohortUsers
"""

# Fetch the data
df_unique_users_in_cohort = pd.read_sql(query_unique_users_in_cohort, engine)

# Display the result
print(f"Number of unique users in the cohort: {df_unique_users_in_cohort['unique_users
```

```
Number of unique users in the cohort: 5998
```

In [18]:

```
# SQL query to find the number of unique users in the cohort who made a booking
query_users_with_bookings = f"""
{cohort_filter}
SELECT COUNT(DISTINCT s.user_id) as users_with_bookings
FROM sessions s
INNER JOIN CohortUsers c ON s.user_id = c.user_id
WHERE s.flight_booked = True OR s.hotel_booked = True
"""

# Fetch the data
df_users_with_bookings = pd.read_sql(query_users_with_bookings, engine)

# Display the result
print(f"Number of unique users in the cohort who made a booking: {df_users_with_bookin
```

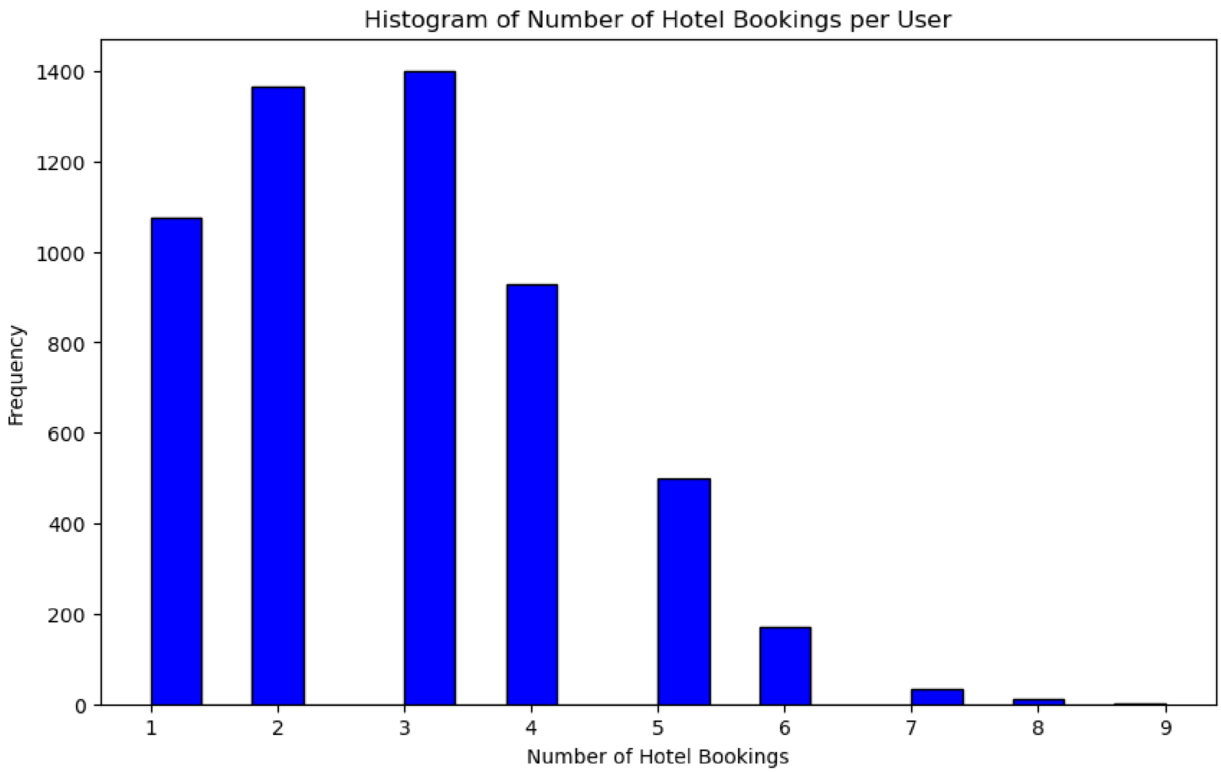
```
Number of unique users in the cohort who made a booking: 5566
```

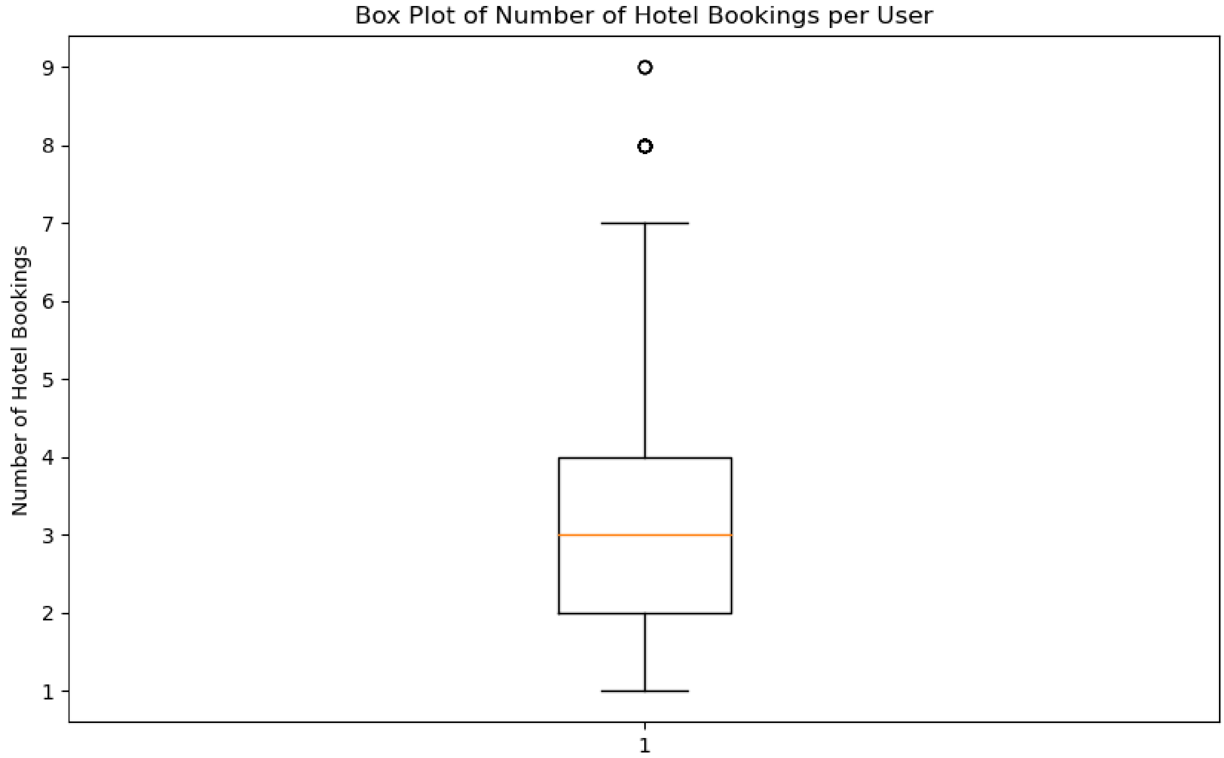
In [19]:

```
# SQL query to find users who frequently book hotels
query_frequent_hotel_bookers = f"""
{cohort_filter}
SELECT s.user_id, COUNT(*) as num_hotel_bookings
FROM sessions s
INNER JOIN CohortUsers c ON s.user_id = c.user_id
WHERE s.hotel_booked = True
GROUP BY s.user_id
ORDER BY num_hotel_bookings DESC
"""
df_frequent_hotel_bookers = pd.read_sql(query_frequent_hotel_bookers, engine)

# Descriptive Statistics
print(df_frequent_hotel_bookers['num_hotel_bookings'].describe())

# Histogram
plt.figure(figsize=(10, 6))
plt.hist(df_frequent_hotel_bookers['num_hotel_bookings'], bins=20, color='blue', edgec
plt.xlabel('Number of Hotel Bookings')
plt.ylabel('Frequency')
plt.title('Histogram of Number of Hotel Bookings per User')
plt.show()
```

```
# Box Plot
plt.figure(figsize=(10, 6))
plt.boxplot(df_frequent_hotel_bookers['num_hotel_bookings'])
plt.ylabel('Number of Hotel Bookings')
plt.title('Box Plot of Number of Hotel Bookings per User')
plt.show()
```

```
count    5486.000000
mean        2.842144
std         1.411458
min         1.000000
25%         2.000000
50%         3.000000
75%         4.000000
max         9.000000
Name: num_hotel_bookings, dtype: float64
```

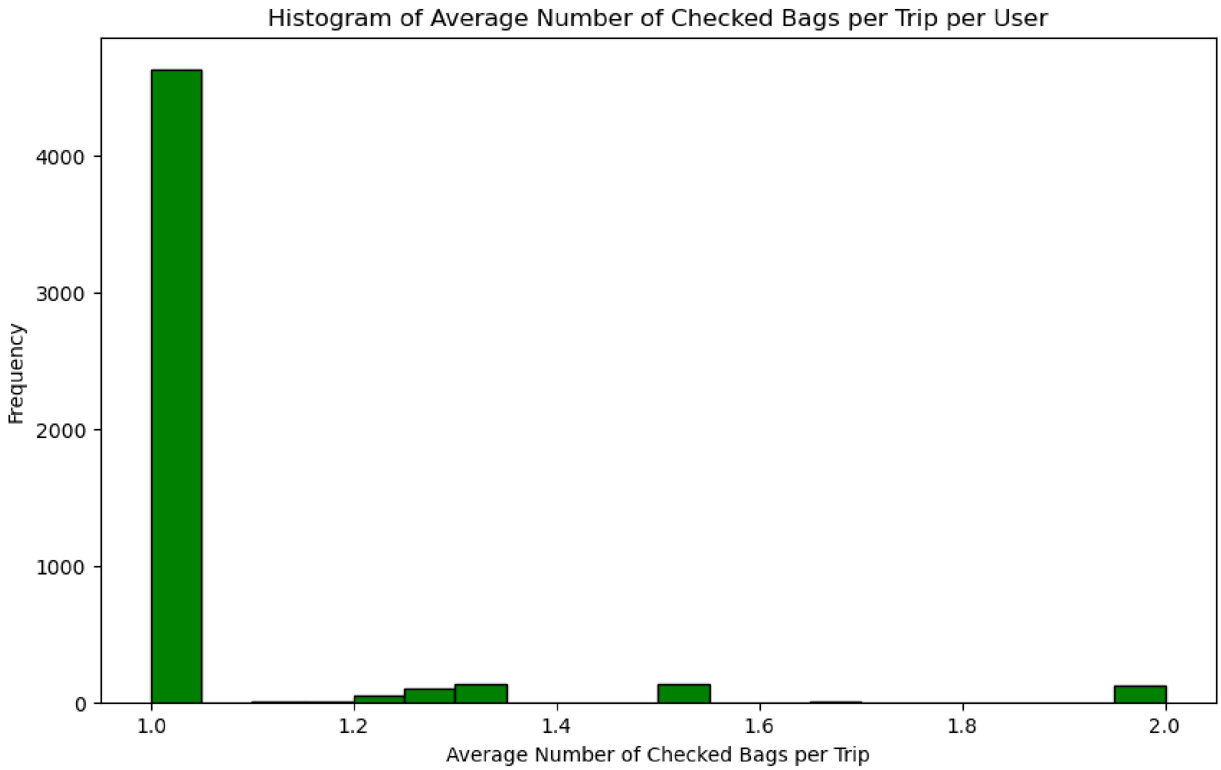


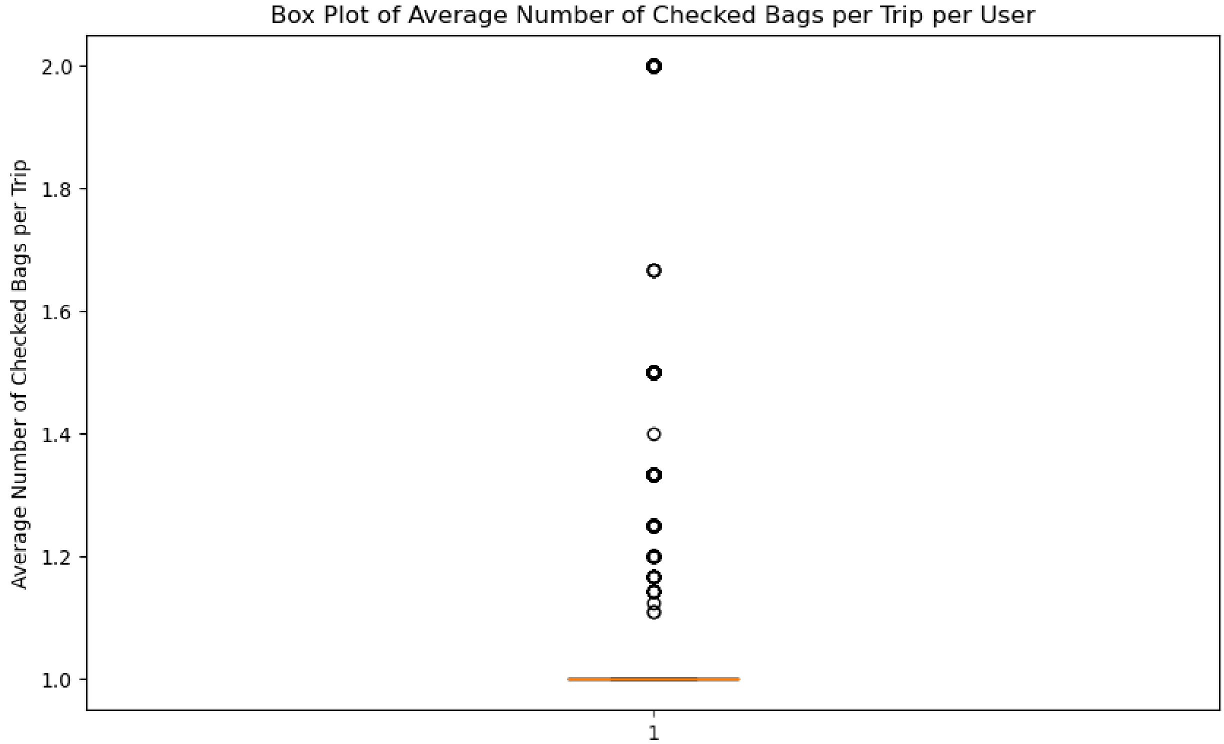


In [20]:
```
# SQL query to find users who frequently check bags, but now with average checked bags
query_frequent_bag_checkers = f"""
{cohort_filter},
FilteredSessions AS (
    SELECT s.user_id, s.trip_id
    FROM sessions s
    INNER JOIN CohortUsers c ON s.user_id = c.user_id
),
BagCounts AS (
    SELECT s.user_id, f.trip_id, COUNT(*) as num_checked_bags
    FROM flights f
    INNER JOIN FilteredSessions s ON f.trip_id = s.trip_id
    GROUP BY s.user_id, f.trip_id
)
SELECT user_id, AVG(num_checked_bags) as avg_checked_bags_per_trip
FROM BagCounts
GROUP BY user_id
ORDER BY avg_checked_bags_per_trip DESC;
"""

df_frequent_bag_checkers = pd.read_sql(query_frequent_bag_checkers, engine)
print("sql done")
# Descriptive Statistics
print(df_frequent_bag_checkers['avg_checked_bags_per_trip'].describe())

# Histogram
plt.figure(figsize=(10, 6))
plt.hist(df_frequent_bag_checkers['avg_checked_bags_per_trip'], bins=20, color='green'
plt.xlabel('Average Number of Checked Bags per Trip')
plt.ylabel('Frequency')
plt.title('Histogram of Average Number of Checked Bags per Trip per User')
plt.show()

# Box Plot
plt.figure(figsize=(10, 6))
```

```python
plt.boxplot(df_frequent_bag_checkers['avg_checked_bags_per_trip'])
plt.ylabel('Average Number of Checked Bags per Trip')
plt.title('Box Plot of Average Number of Checked Bags per Trip per User')
plt.show()
```

```
sql done
count    5206.000000
mean        1.053488
std         0.179770
min         1.000000
25%         1.000000
50%         1.000000
75%         1.000000
max         2.000000
Name: avg_checked_bags_per_trip, dtype: float64
```





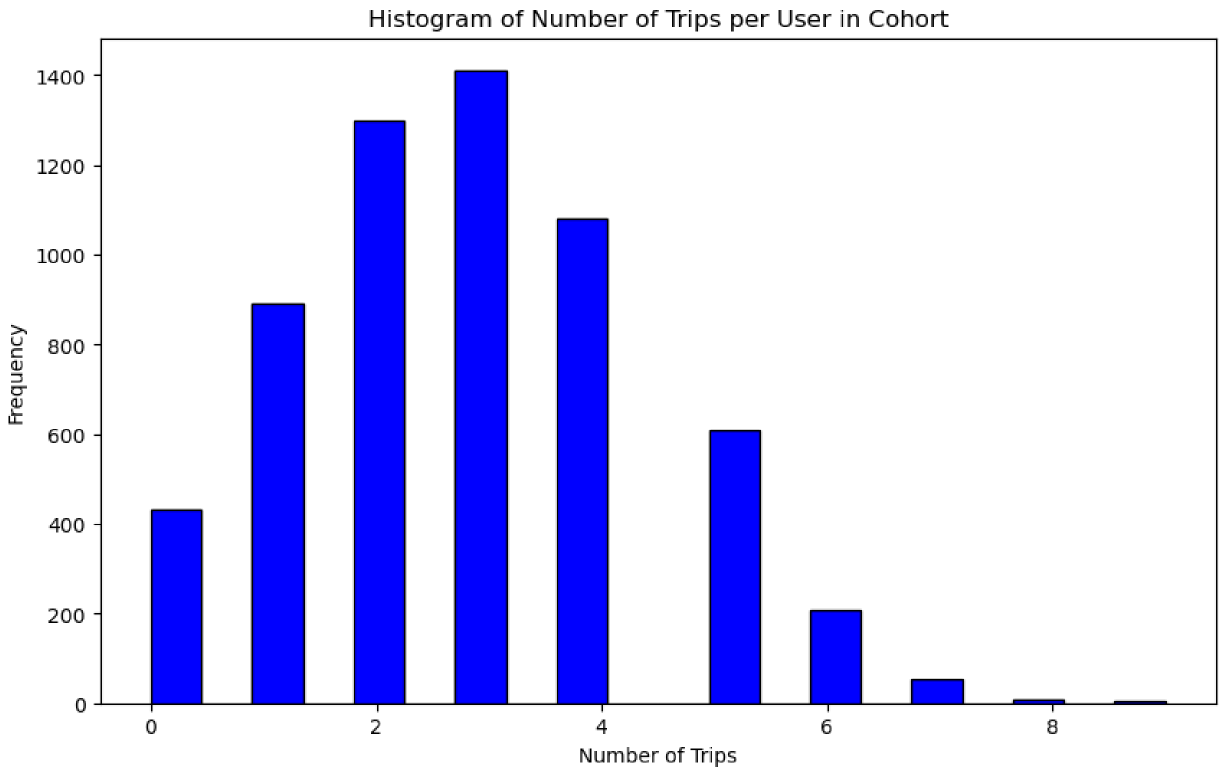
Almost every user checks at least one bag, this is not an insight that will contribte to our segmentation. Let's check other aspects

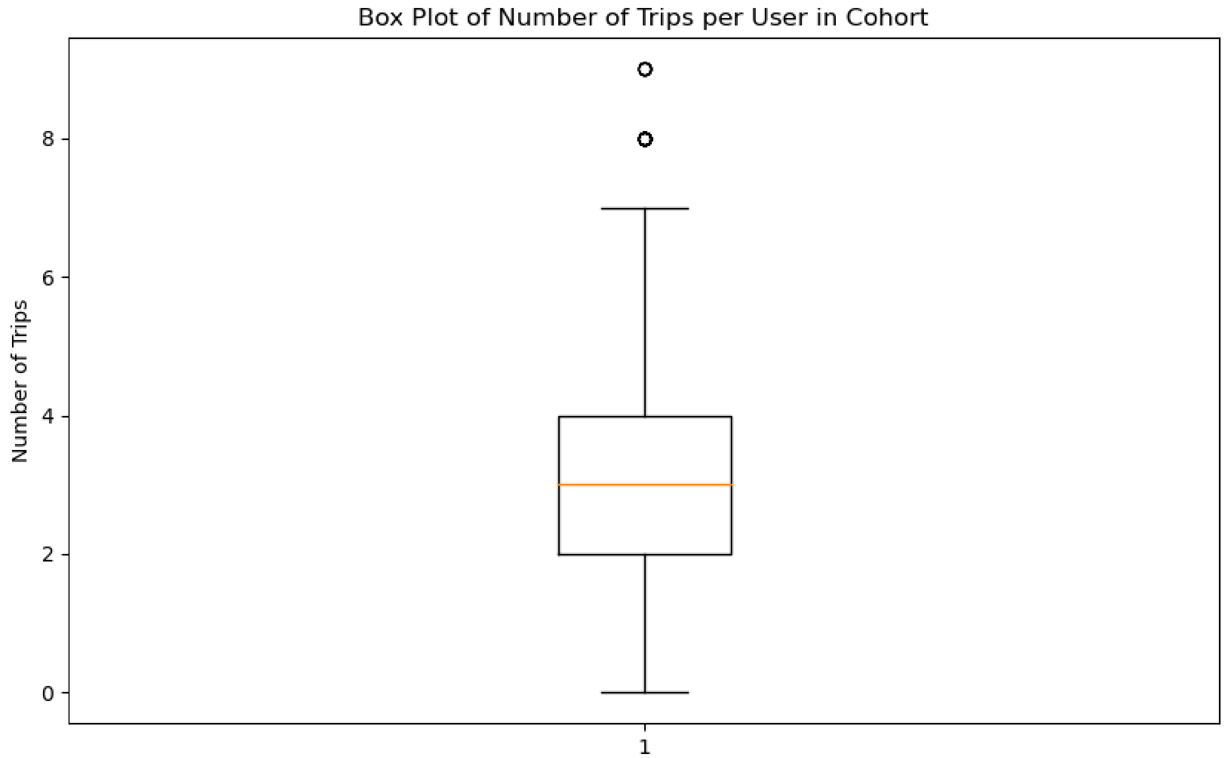
Let's find the high frequency travellers

In [21]:
```
# SQL query to find high frequency travelers in the cohort
query_high_frequency_travelers = f"""
{cohort_filter}
SELECT s.user_id, COUNT(DISTINCT s.trip_id) as num_trips
FROM sessions s
INNER JOIN CohortUsers c ON s.user_id = c.user_id
GROUP BY s.user_id
ORDER BY num_trips DESC
"""

# Execute the query and store the result in a DataFrame
df_high_frequency_travelers = pd.read_sql(query_high_frequency_travelers, engine)

# Display descriptive statistics
print(df_high_frequency_travelers['num_trips'].describe())

# Plotting
# Histogram
plt.figure(figsize=(10, 6))
plt.hist(df_high_frequency_travelers['num_trips'], bins=20, color='blue', edgecolor='b
plt.xlabel('Number of Trips')
plt.ylabel('Frequency')
plt.title('Histogram of Number of Trips per User in Cohort')
plt.show()

# Box Plot
plt.figure(figsize=(10, 6))
plt.boxplot(df_high_frequency_travelers['num_trips'])
```

```
plt.ylabel('Number of Trips')
plt.title('Box Plot of Number of Trips per User in Cohort')
plt.show()
```

```
count    5998.000000
mean        2.805435
std         1.589629
min         0.000000
25%         2.000000
50%         3.000000
75%         4.000000
max         9.000000
Name: num_trips, dtype: float64
```





Long Distance Flights

Vincenty Function to rpeplace the Haversine function - Vincenty is much more accurate

https://en.wikipedia.org/wiki/Vincenty%27s_formulae

In [22]:
```
from math import radians, sin, cos, sqrt, atan2

def vincenty_distance(lat1, lon1, lat2, lon2):
    """
    Calculate the great-circle distance between two points
    on the Earth surface given their latitude and longitude
    in decimal degrees.
    """
    # WGS-84 ellipsiod parameters
    a = 6378137.0  # semi-major axis in meters
    f = 1 / 298.257223563  # flattening
    b = (1 - f) * a  # semi-minor axis

    # convert decimal degrees to radians
    lat1, lon1, lat2, lon2 = map(radians, [lat1, lon1, lat2, lon2])

    # calculations
    U1 = atan((1 - f) * tan(lat1))
    U2 = atan((1 - f) * tan(lat2))
    sinU1 = sin(U1)
    cosU1 = cos(U1)
    sinU2 = sin(U2)
    cosU2 = cos(U2)
```

```
    lon_diff = lon2 - lon1
    Lambda = lon_diff  # initial approximation for Lambda
    sinLambda = sin(Lambda)
    cosLambda = cos(Lambda)

    # iterate until change is insignificant
    for _ in range(1000):
        sinSigma = sqrt((cosU2 * sin(Lambda)) ** 2 + (cosU1 * sinU2 - sinU1 * cosU2 *
        cosSigma = sinU1 * sinU2 + cosU1 * cosU2 * cos(Lambda)
        sigma = atan2(sinSigma, cosSigma)
        sinAlpha = cosU1 * cosU2 * sin(Lambda) / sinSigma
        cos2Alpha = 1 - sinAlpha ** 2
        cos2SigmaM = cosSigma - 2 * sinU1 * sinU2 / cos2Alpha
        C = f / 16 * cos2Alpha * (4 + f * (4 - 3 * cos2Alpha))
        Lambda_prev = Lambda
        Lambda = lon_diff + (1 - C) * f * sinAlpha * (sigma + C * sinSigma * (cos2Sigr
        # break if change in lambda is insignificant
        if abs(Lambda - Lambda_prev) < 1e-12:
            break

    # final calculations
    u2 = cos2Alpha * (a ** 2 - b ** 2) / (b ** 2)
    A = 1 + u2 / 16384 * (4096 + u2 * (-768 + u2 * (320 - 175 * u2)))
    B = u2 / 1024 * (256 + u2 * (-128 + u2 * (74 - 47 * u2)))
    deltaSigma = B * sinSigma * (cos2SigmaM + B / 4 * (cosSigma * (-1 + 2 * cos2SigmaM

    # distance in meters
    s = b * A * (sigma - deltaSigma)

    return s
```

Let's explore Flight Distance

In [23]:
```
# SQL query to get flight data
flight_query = f"""
{cohort_filter}
SELECT
    f.origin_airport,
    f.destination_airport,
    u.home_airport_lat AS origin_airport_lat,
    u.home_airport_lon AS origin_airport_lon,
    f.destination_airport_lat,
    f.destination_airport_lon
FROM flights f
JOIN sessions s ON f.trip_id = s.trip_id
JOIN CohortUsers c ON s.user_id = c.user_id
JOIN users u ON s.user_id = u.user_id
"""

# Execute the query and load the data into a DataFrame
df_flights = pd.read_sql(flight_query, engine)
```

In [24]:
```
!pip install geopy
```

```
Requirement already satisfied: geopy in c:\users\bhaze\anaconda3\lib\site-packages
(2.4.0)
Requirement already satisfied: geographiclib<3,>=1.52 in c:\users\bhaze\anaconda3\lib
\site-packages (from geopy) (2.0)
```

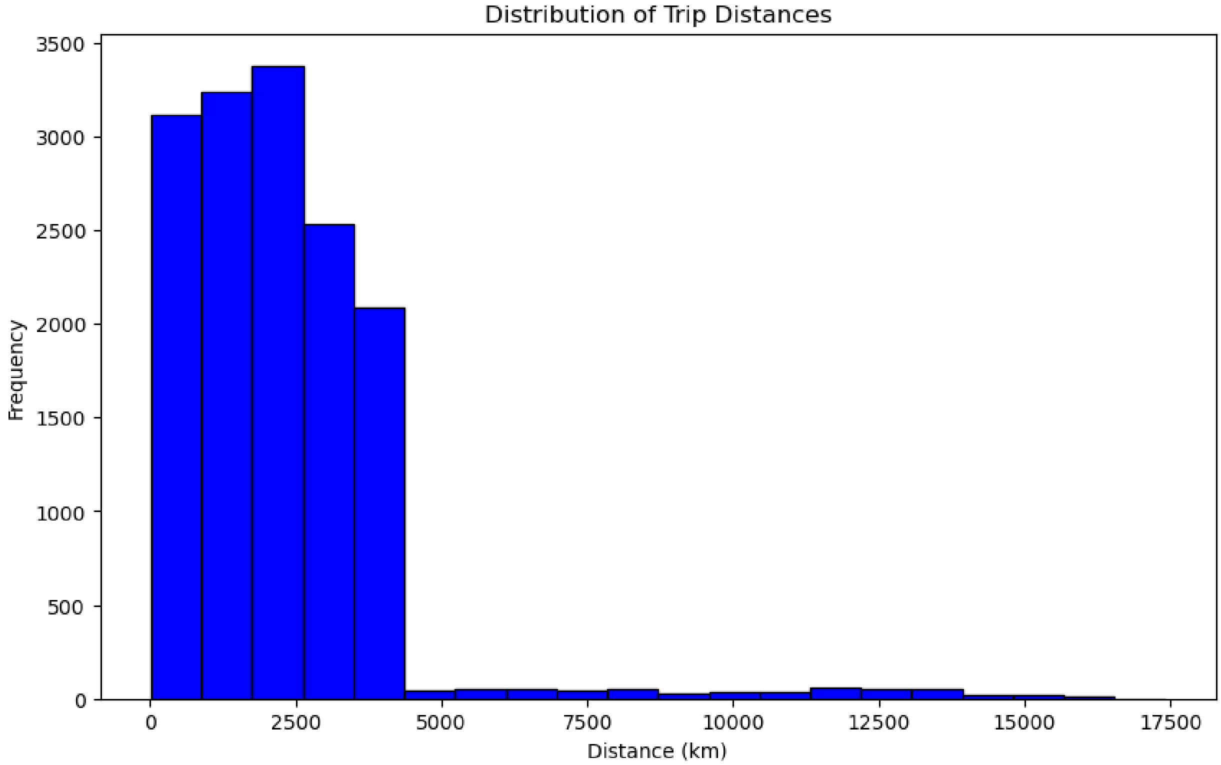
In [25]:
```
from geopy.distance import geodesic

# Initialize an empty list to store the distances
distances = []

# Loop through the DataFrame to calculate distances
for index, row in df_flights.iterrows():
    origin = (row['origin_airport_lat'], row['origin_airport_lon'])
    destination = (row['destination_airport_lat'], row['destination_airport_lon'])
    distance = geodesic(origin, destination).kilometers
    distances.append(distance)

# Add the distances to the DataFrame
df_flights['trip_distance_km'] = distances

# Display the updated DataFrame
print(df_flights.head())
```

```
  origin_airport destination_airport  origin_airport_lat  origin_airport_lon  \
0            IND                 YMX              39.717             -86.294   
1            LAX                 YVR              33.942            -118.408   
2            LGA                 YZD              40.777             -73.872   
3            IAD                 JFK              38.944             -77.456   
4            LUK                 DEN              39.103             -84.419   

   destination_airport_lat  destination_airport_lon  trip_distance_km  
0                   45.517                  -73.417       1235.221296  
1                   49.195                 -123.182       1739.332492  
2                   43.862                  -79.370        567.979189  
3                   40.640                  -73.779        366.896751  
4                   39.858                 -104.667       1740.377783  
```

In [26]:
```
# Display descriptive statistics
print(df_flights['trip_distance_km'].describe())
```

```
count    14919.000000
mean      2331.347257
std       1988.553779
min         17.685030
25%       1069.915667
50%       2064.546063
75%       3132.946871
max      17407.930322
Name: trip_distance_km, dtype: float64
```

In [27]:
```
# Create a histogram for trip distances
plt.figure(figsize=(10, 6))
plt.hist(df_flights['trip_distance_km'], bins=20, color='blue', edgecolor='black')
plt.title('Distribution of Trip Distances')
plt.xlabel('Distance (km)')
plt.ylabel('Frequency')
plt.show()
```



We're going to go off track and start looking at data in a different way. I'm really going to need to clean up this notebook!

Free Hotel Meal segemntation

- Budget-Conscious Travelers
- International Travelers
- Couples on Weekend Getaways

Budget-Conscious Travelers

We are going to look at a few aspects of the data. Some can be reused in other explorations:

1. Average Base Fare for Flights
2. Average Hotel Per Room Cost
3. Usage of Discounts

In [28]:
```
# Get the average base fares
avg_base_fare_query = f"""
{cohort_filter}
SELECT AVG(f.base_fare_usd) AS avg_base_fare, s.user_id
FROM flights f
JOIN sessions s ON f.trip_id = s.trip_id
JOIN CohortUsers c ON s.user_id = c.user_id
GROUP BY s.user_id;
```

```
"""
df_avg_base_fare = pd.read_sql(avg_base_fare_query, engine)

# Get the number of bookings with a discount
discount_booking_query = f"""
{cohort_filter}
SELECT COUNT(*) AS num_discounts, s.user_id
FROM sessions s
JOIN CohortUsers c ON s.user_id = c.user_id
WHERE CAST(s.flight_discount AS INTEGER) = 1 OR CAST(s.hotel_discount AS INTEGER) = 1
GROUP BY s.user_id;
"""

df_discount_booking = pd.read_sql(discount_booking_query, engine)

# Get the Average Price for Hotel Stays
average_price_hotel_stays_query = f"""
{cohort_filter}
SELECT AVG(h.hotel_per_room_usd) AS avg_hotel_price, s.user_id
FROM hotels h
JOIN sessions s ON h.trip_id = s.trip_id
JOIN CohortUsers c ON s.user_id = c.user_id
GROUP BY s.user_id;

"""

df_average_hotel_stays = pd.read_sql(average_price_hotel_stays_query, engine)
```

In [29]:

```
# Descriptive Statistics first

print(df_avg_base_fare['avg_base_fare'].describe())
print(df_discount_booking['num_discounts'].describe())
print(df_average_hotel_stays['avg_hotel_price'].describe())
```

```
count     5206.000000
mean       538.526171
std        685.388137
min          5.350000
25%        284.835000
50%        392.895000
75%        564.665375
max      14280.380000
Name: avg_base_fare, dtype: float64
count    5618.000000
mean        2.533464
std         1.204718
min         1.000000
25%         2.000000
50%         2.000000
75%         3.000000
max         7.000000
Name: num_discounts, dtype: float64
count    5435.000000
mean      178.336311
std        84.305174
min        24.000000
25%       124.500000
50%       163.400000
75%       212.000000
max      1063.000000
Name: avg_hotel_price, dtype: float64
```

What the heck is with a $5.35 flight??? This probably won't add anything to the analysis, but I need to find out

In [30]:
```
# SQL query to investigate the $5.35 flight with latitude and longitude
investigate_cheap_flight_query = """
SELECT f.base_fare_usd, u.home_airport_lat AS origin_lat, u.home_airport_lon AS origir
       f.destination_airport_lat AS dest_lat, f.destination_airport_lon AS dest_lon, f
FROM flights f
JOIN sessions s ON f.trip_id = s.trip_id
JOIN users u ON s.user_id = u.user_id
WHERE f.base_fare_usd = 5.35
ORDER BY s.user_id;
"""

# Execute the query and store the result in a DataFrame
df_cheap_flight = pd.read_sql(investigate_cheap_flight_query, engine)

# Calculate the distance using vincenty_distance
#df_cheap_flight['calculated_distance'] = df_cheap_flight.apply(
#    lambda row: vincenty_distance((row['origin_lat'], row['origin_lon']), (row['dest_
#)

# Display the DataFrame with calculated distance
print(df_cheap_flight)
```

```
   base_fare_usd  origin_lat  origin_lon  dest_lat  dest_lon  \
0           5.35      33.818    -118.151    33.942  -118.408
1           5.35      33.676    -117.868    33.942  -118.408
2           5.35      33.818    -118.151    33.942  -118.408
3           5.35      33.818    -118.151    33.942  -118.408
4           5.35      33.818    -118.151    33.942  -118.408
5           5.35      33.818    -118.151    33.942  -118.408
6           5.35      40.692     -74.169    40.640   -73.779
7           5.35      42.422     -87.868    42.947   -87.896

       departure_time  return_flight_booked         return_time  seats  \
0 2022-03-26 11:00:00                  True 2022-03-30 11:00:00      1
1 2022-09-01 13:00:00                 False                 NaT      1
2 2023-02-22 07:00:00                  True 2023-02-24 07:00:00      1
3 2023-01-14 08:00:00                  True 2023-01-24 08:00:00      1
4 2023-03-25 08:00:00                  True 2023-03-29 08:00:00      1
5 2023-05-26 07:00:00                  True 2023-05-27 07:00:00      1
6 2023-06-22 11:00:00                  True 2023-06-26 11:00:00      1
7 2023-06-20 15:00:00                 False                 NaT      1

   checked_bags        trip_airline  destination  user_id   home_city
0             1     United Airlines  los angeles    49728  long beach
1             1     Delta Air Lines  los angeles   245954   santa ana
2             0     Delta Air Lines  los angeles   507931  long beach
3             2             Volaris  los angeles   510094  long beach
4             0     United Airlines  los angeles   624064  long beach
5             1     Delta Air Lines  los angeles   721200  long beach
6             1   American Airlines     new york   789756      newark
7             1  Southwest Airlines    milwaukee   920115     chicago
```

Interesting. I guess, $5.35 would be cheaper than a taxi in these cases. But really?

Back to the Budget-Conscious Travelers. We need to filter the fares a bit to exclude some outliers. We will use the 1st and 99th percentiles

In [31]:

```
# Calculate the 1st and 99th percentiles
lower_percentile = df_avg_base_fare['avg_base_fare'].quantile(0.01)
upper_percentile = df_avg_base_fare['avg_base_fare'].quantile(0.99)

# Filter out values below the 1st percentile and above the 99th percentile
df_filtered_avg_base_fare = df_avg_base_fare[(df_avg_base_fare['avg_base_fare'] >= low
                                              (df_avg_base_fare['avg_base_fare'] <= upp

# Display the filtered DataFrame statistics
print(df_filtered_avg_base_fare['avg_base_fare'].describe())
```

```
count    5100.000000
mean      491.411119
std       395.947058
min        62.410000
25%       286.955833
50%       392.895000
75%       559.405000
max      3449.390000
Name: avg_base_fare, dtype: float64
```

$62.41 is still low, but within reason for some of the budget airlines.

Now a ratio of fare to distance. This can help determine which fares are lower or higher than average