In [41]:
```
import pandas as pd
import numpy as np
```

Daily Activity Data Overview The dailyActivity_merged.csv file contains various metrics that could be extremely useful for a wearable tech sensor project, particularly for tracking daily physical activity and health management. Here's a summary of the key columns in this dataset:

Id: Unique identifier for the participant. ActivityDate: Date of the activity recorded. TotalSteps: Total number of steps taken in the day. TotalDistance: Total distance covered in the day (measured in miles). VeryActiveDistance, ModeratelyActiveDistance, LightActiveDistance: Distance covered during different levels of activity intensities. SedentaryActiveDistance: Distance covered while sedentary. VeryActiveMinutes, FairlyActiveMinutes, LightlyActiveMinutes: Minutes spent in various activity intensities. SedentaryMinutes: Minutes spent sedentary. Calories: Total calories burned during the day.

In [13]:
```
data = pd.concat\
(
    [pd.read_csv(r"C:\Users\Arshp\Downloads\archive (3)\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\dailyActivity_merged.csv"
    pd.read_csv(r"C:\Users\Arshp\Downloads\archive (3)\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\dailyActivity_merged.csv")
)
```

In [15]:
```
# Display the first few rows and the summary of the dataframe
data.head(), data.info(), data.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1397 entries, 0 to 939
Data columns (total 15 columns):
 #   Column                     Non-Null Count  Dtype
---  ------                     --------------  -----
 0   Id                         1397 non-null   int64
 1   ActivityDate               1397 non-null   object
 2   TotalSteps                 1397 non-null   int64
 3   TotalDistance              1397 non-null   float64
 4   TrackerDistance            1397 non-null   float64
 5   LoggedActivitiesDistance   1397 non-null   float64
 6   VeryActiveDistance         1397 non-null   float64
 7   ModeratelyActiveDistance   1397 non-null   float64
 8   LightActiveDistance        1397 non-null   float64
 9   SedentaryActiveDistance    1397 non-null   float64
 10  VeryActiveMinutes          1397 non-null   int64
 11  FairlyActiveMinutes        1397 non-null   int64
 12  LightlyActiveMinutes       1397 non-null   int64
 13  SedentaryMinutes           1397 non-null   int64
 14  Calories                   1397 non-null   int64
dtypes: float64(7), int64(7), object(1)
memory usage: 174.6+ KB
```

Out[15]: (

	Id	ActivityDate	TotalSteps	TotalDistance	TrackerDistance
0	1503960366	3/25/2016	11004	7.11	7.11
1	1503960366	3/26/2016	17609	11.55	11.55
2	1503960366	3/27/2016	12736	8.53	8.53
3	1503960366	3/28/2016	13231	8.93	8.93
4	1503960366	3/29/2016	12041	7.85	7.85

	LoggedActivitiesDistance	VeryActiveDistance	ModeratelyActiveDistance
0	0.0	2.57	0.46
1	0.0	6.92	0.73
2	0.0	4.66	0.16
3	0.0	3.19	0.79
4	0.0	2.16	1.09

	LightActiveDistance	SedentaryActiveDistance	VeryActiveMinutes
0	4.07	0.0	33
1	3.91	0.0	89
2	3.71	0.0	56
3	4.95	0.0	39
4	4.61	0.0	28

	FairlyActiveMinutes	LightlyActiveMinutes	SedentaryMinutes	Calories
0	12	205	804	1819
1	17	274	588	2154
2	5	268	605	1944
3	20	224	1080	1932
4	28	243	763	1886

,
None,

	Id	TotalSteps	TotalDistance	TrackerDistance
count	1.397000e+03	1397.000000	1397.000000	1397.000000
mean	4.781210e+09	7280.898354	5.219434	5.192219
std	2.384293e+09	5214.336113	3.994206	3.980077
min	1.503960e+09	0.000000	0.000000	0.000000
25%	2.320127e+09	3146.000000	2.170000	2.160000
50%	4.445115e+09	6999.000000	4.950000	4.950000
75%	6.962181e+09	10544.000000	7.500000	7.480000
max	8.877689e+09	36019.000000	28.030001	28.030001

	LoggedActivitiesDistance	VeryActiveDistance	ModeratelyActiveDistance
count	1397.000000	1397.000000	1397.000000
mean	0.131481	1.397416	0.538461
std	0.703683	2.607480	0.867436
min	0.000000	0.000000	0.000000
25%	0.000000	0.000000	0.000000
50%	0.000000	0.100000	0.200000
75%	0.000000	1.830000	0.770000
max	6.727057	21.920000	6.480000

	LightActiveDistance	SedentaryActiveDistance	VeryActiveMinutes
count	1397.000000	1397.000000	1397.000000
mean	3.193407	0.001704	19.679313
std	2.116869	0.007736	31.675878
min	0.000000	0.000000	0.000000
25%	1.610000	0.000000	0.000000
50%	3.240000	0.000000	2.000000
75%	4.690000	0.000000	30.000000
max	12.510000	0.110000	210.000000

	FairlyActiveMinutes	LightlyActiveMinutes	SedentaryMinutes
count	1397.000000	1397.000000	1397.000000
mean	13.403006	185.372942	992.542591
std	26.401247	114.058601	313.297376
min	0.000000	0.000000	0.000000
25%	0.000000	111.000000	729.000000
50%	6.000000	195.000000	1057.000000
75%	18.000000	262.000000	1244.000000
max	660.000000	720.000000	1440.000000

	Calories
count	1397.000000
mean	2266.265569
std	753.005527
min	0.000000
25%	1799.000000
50%	2114.000000
75%	2770.000000
max	4900.000000

)

In [20]:
```
# Check for missing values in the dataset
missing_values = data.isnull().sum()

# Convert the 'ActivityDate' column to datetime format
data['ActivityDate'] = pd.to_datetime(data['ActivityDate'])

# Basic statistical properties to spot potential outliers
statistical_summary = data.describe()

missing_values, data['ActivityDate'].dtype, statistical_summary
```

Out[20]:
```
(Id                          0
 ActivityDate                0
 TotalSteps                  0
 TotalDistance               0
 TrackerDistance             0
 LoggedActivitiesDistance    0
 VeryActiveDistance          0
 ModeratelyActiveDistance    0
 LightActiveDistance         0
 SedentaryActiveDistance     0
 VeryActiveMinutes           0
 FairlyActiveMinutes         0
 LightlyActiveMinutes        0
 SedentaryMinutes            0
 Calories                    0
 dtype: int64,
 dtype('<M8[ns]'),
                  Id    TotalSteps  TotalDistance  TrackerDistance  \
 count  1.397000e+03   1397.000000    1397.000000      1397.000000
 mean   4.781210e+09   7280.898354       5.219434         5.192219
 std    2.384293e+09   5214.336113       3.994206         3.980077
 min    1.503960e+09      0.000000       0.000000         0.000000
 25%    2.320127e+09   3146.000000       2.170000         2.160000
 50%    4.445115e+09   6999.000000       4.950000         4.950000
 75%    6.962181e+09  10544.000000       7.500000         7.480000
 max    8.877689e+09  36019.000000      28.030001        28.030001

        LoggedActivitiesDistance  VeryActiveDistance  ModeratelyActiveDistance  \
 count               1397.000000         1397.000000               1397.000000
 mean                   0.131481            1.397416                  0.538461
 std                    0.703683            2.607480                  0.867436
 min                    0.000000            0.000000                  0.000000
 25%                    0.000000            0.000000                  0.000000
 50%                    0.000000            0.100000                  0.200000
 75%                    0.000000            1.830000                  0.770000
 max                    6.727057           21.920000                  6.480000

        LightActiveDistance  SedentaryActiveDistance  VeryActiveMinutes  \
 count          1397.000000              1397.000000        1397.000000
 mean              3.193407                 0.001704          19.679313
 std               2.116869                 0.007736          31.675878
 min               0.000000                 0.000000           0.000000
 25%               1.610000                 0.000000           0.000000
 50%               3.240000                 0.000000           2.000000
 75%               4.690000                 0.000000          30.000000
 max              12.510000                 0.110000         210.000000

        FairlyActiveMinutes  LightlyActiveMinutes  SedentaryMinutes  \
 count          1397.000000           1397.000000       1397.000000
 mean             13.403006            185.372942        992.542591
 std              26.401247            114.058601        313.297376
 min               0.000000              0.000000          0.000000
 25%               0.000000            111.000000        729.000000
 50%               6.000000            195.000000       1057.000000
 75%              18.000000            262.000000       1244.000000
 max             660.000000            720.000000       1440.000000

           Calories
 count  1397.000000
 mean   2266.265569
 std     753.005527
 min       0.000000
 25%    1799.000000
 50%    2114.000000
 75%    2770.000000
 max    4900.000000  )
```

Data cleaning and preprocessing are essential steps before visualization to ensure the accuracy and relevance of the insights derived. Let's go through some common preprocessing steps for this dataset:

Checking for Missing Values: Identify any null or missing data that might skew the analysis. Data Type Conversion: Ensure that all data types are appropriate for analysis, such as converting dates to a datetime format. Outlier Detection: Check for outliers that could distort our visualizations and analyses. Normalization or Standardization (if necessary): Useful especially when dealing with features of varying scales, but this might not be necessary for our initial exploratory visualizations.

The data in the dailyActivity_merged.csv file appears to be clean, with no missing values, and the ActivityDate column is correctly formatted as a datetime type, which is suitable for analysis. The descriptive statistics do not indicate any extreme outliers that might significantly distort typical analyses; however, some metrics, like VeryActiveMinutes and Calories, have a wide range, suggesting variability in participant activity levels.

Observations: The variability in activity metrics (from zero to quite high) indicates diverse physical activity levels among participants, which can be explored further. The Calories column spans from 0 to 4562, indicating some days with very high activity or potential measurement errors (if zero represents days with missing data).

In [24]:
```
# Checking for duplicate rows based on 'Id' and 'ActivityDate'
duplicates = data.duplicated(subset=['Id', 'ActivityDate'], keep=False)

# Showing any duplicate rows if present
duplicate_rows = data[duplicates]

# Resetting index to ensure it's unique
data.reset_index(drop=True, inplace=True)

# Show any duplicate rows and the new head of the DataFrame to confirm index reset
duplicate_rows, data.head()
```

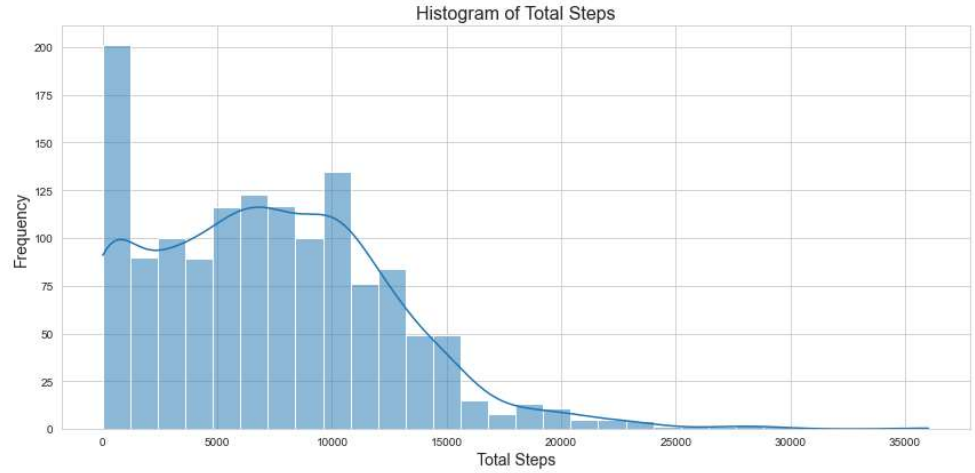
Out[24]:
```
(            Id ActivityDate  TotalSteps  TotalDistance  TrackerDistance  \
 18   1503960366   2016-04-12         224           0.14             0.14
 37   1624580081   2016-04-12        6627           4.31             4.31
 59   1844505072   2016-04-12           0           0.00             0.00
 71   1927972279   2016-04-12          24           0.02             0.02
 83   2022484408   2016-04-12        6717           4.72             4.72
 95   2026352035   2016-04-12        1019           0.63             0.63
 107  2320127002   2016-04-12        2098           1.41             1.41
 122  2347167796   2016-04-12           0           0.00             0.00
 134  2873212765   2016-04-12           0           0.00             0.00
 164  3977333714   2016-04-12         759           0.57             0.57
 196  4020332650   2016-04-12           8           0.01             0.01
 228  4057192912   2016-04-12         187           0.14             0.14
 263  4445114986   2016-04-12         278           0.19             0.19
 275  4558609924   2016-04-12        1260           0.83             0.83
 290  4702921684   2016-04-12           0           0.00             0.00
 302  5553957443   2016-04-12        3436           2.24             2.24
 365  6962181067   2016-04-12        5893           3.90             3.90
 377  7007744171   2016-04-12        7413           5.77             4.96
```

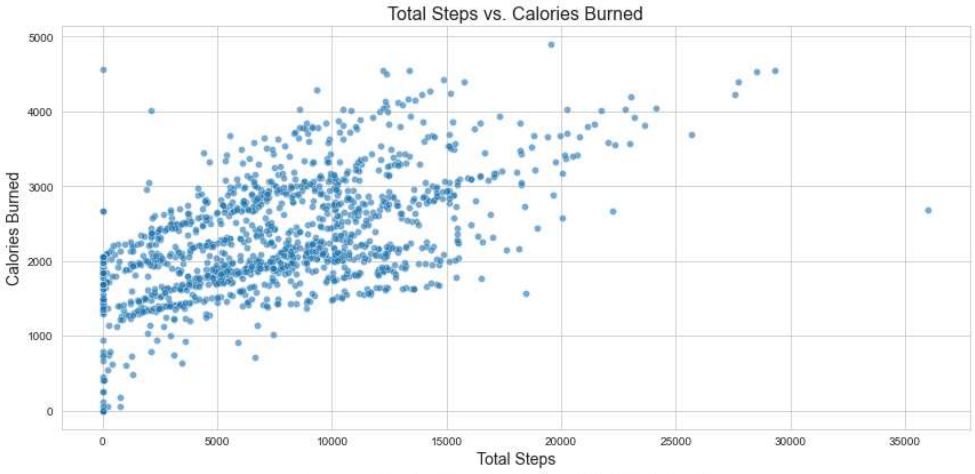
Visualizations To deepen our understanding, I'll create a few visualizations:

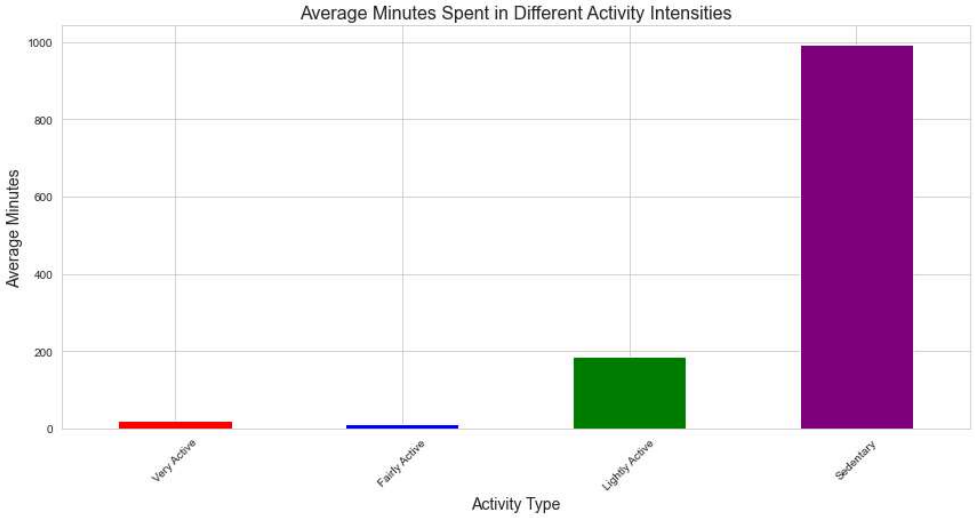
A histogram to see the distribution of total steps taken by participants. A scatter plot to check the relationship between total steps and calories burned. A bar graph to visualize average time spent in different activity intensities.

In [25]:

```
# Create a figure to hold the plots again
fig, axes = plt.subplots(3, 1, figsize=(12, 18))

# Histogram of Total Steps
sns.histplot(data=data, x="TotalSteps", bins=30, kde=True, ax=axes[0])
axes[0].set_title('Histogram of Total Steps', fontsize=16)
axes[0].set_xlabel('Total Steps', fontsize=14)
axes[0].set_ylabel('Frequency', fontsize=14)

# Scatter plot of Total Steps vs Calories
sns.scatterplot(data=data, x="TotalSteps", y="Calories", ax=axes[1], alpha=0.6)
axes[1].set_title('Total Steps vs. Calories Burned', fontsize=16)
axes[1].set_xlabel('Total Steps', fontsize=14)
axes[1].set_ylabel('Calories Burned', fontsize=14)

# Bar plot of average minutes in different activity levels
activity_minutes = data[['VeryActiveMinutes', 'FairlyActiveMinutes', 'LightlyActiveMinutes', 'SedentaryMinutes']].mean()
activity_minutes.plot(kind='bar', ax=axes[2], color=['red', 'blue', 'green', 'purple'])
axes[2].set_title('Average Minutes Spent in Different Activity Intensities', fontsize=16)
axes[2].set_xlabel('Activity Type', fontsize=14)
axes[2].set_ylabel('Average Minutes', fontsize=14)
axes[2].set_xticklabels(['Very Active', 'Fairly Active', 'Lightly Active', 'Sedentary'], rotation=45)

# Tight layout to avoid overlap
plt.tight_layout()

# Show the plots
plt.show()
```







Histogram of Total Steps: This histogram reveals the distribution of total steps taken per day by participants. It highlights that most people are moderately active, with a significant number taking fewer steps, indicating a potential area to encourage more physical activity. Scatter Plot of Total Steps vs. Calories Burned: The scatter plot shows a clear positive correlation between the number of steps taken and calories burned. This can be particularly useful for wearable tech applications that aim to help users increase their physical activity to achieve calorie-burning goals. Bar Graph of Average Minutes Spent in Different Activity Intensities: The bar graph illustrates that most of the time is spent sedentary, with a relatively smaller amount of time spent in light, fairly active, or very active states. This insight could be used to design interventions or features that encourage users to reduce sedentary time and increase time spent in more active states. These visualizations provide a foundational understanding of the activity patterns among the participants, which can inform feature design and health interventions in wearable technology.

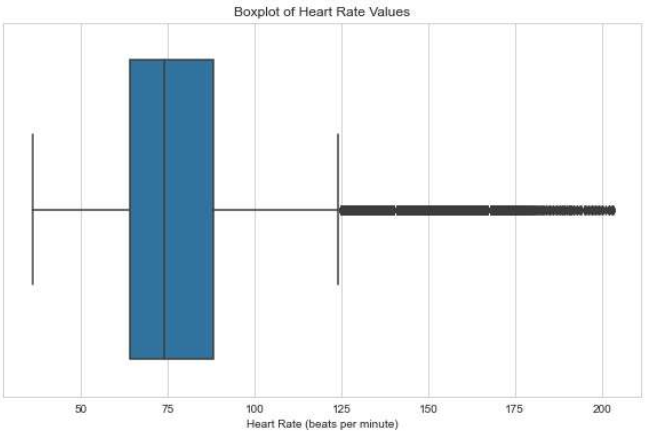
In [26]:
```
heart_rate_df = pd.concat\
(
    [pd.read_csv(r"C:\Users\Arshp\Downloads\archive (3)\mturkfitbit_export_3.12.16-4.11.16\Fitabase Data 3.12.16-4.11.16\heartrate_seconds_merged.
    pd.read_csv(r"C:\Users\Arshp\Downloads\archive (3)\mturkfitbit_export_4.12.16-5.12.16\Fitabase Data 4.12.16-5.12.16\heartrate_seconds_merged.c
)
```

In [27]:
```
# Display the first few rows and the summary of the dataframe
heart_rate_df.head(), heart_rate_df.info(), heart_rate_df.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 3638339 entries, 0 to 2483657
Data columns (total 3 columns):
 #   Column  Dtype
---  ------  -----
 0   Id      int64
 1   Time    object
 2   Value   int64
dtypes: int64(2), object(1)
memory usage: 111.0+ MB
```

Out[27]:
```
(           Id                 Time  Value
 0  2022484408  4/1/2016 7:54:00 AM     93
 1  2022484408  4/1/2016 7:54:05 AM     91
 2  2022484408  4/1/2016 7:54:10 AM     96
 3  2022484408  4/1/2016 7:54:15 AM     98
 4  2022484408  4/1/2016 7:54:20 AM    100,
 None,
                  Id         Value
 count  3.638339e+06  3.638339e+06
 mean   5.462465e+09  7.809913e+01
 std    1.978491e+09  1.922622e+01
 min    2.022484e+09  3.600000e+01
 25%    4.020333e+09  6.400000e+01
 50%    5.553957e+09  7.400000e+01
 75%    6.962181e+09  8.800000e+01
 max    8.877689e+09  2.030000e+02)
```

In [30]:
```
# Optimize parsing by specifying the format directly
heart_rate_df['Time'] = pd.to_datetime(heart_rate_df['Time'], format='%m/%d/%Y %I:%M:%S %p')

# Check for successful conversion and reattempt to plot the boxplot
plt.figure(figsize=(10, 6))
sns.boxplot(x=heart_rate_df['Value'])
plt.title('Boxplot of Heart Rate Values')
plt.xlabel('Heart Rate (beats per minute)')
plt.show()

# Display the first few rows to confirm the Time conversion
heart_rate_df.head()
```



Out[30]:

	Id	Time	Value
0	2022484408	2016-04-01 07:54:00	93
1	2022484408	2016-04-01 07:54:05	91
2	2022484408	2016-04-01 07:54:10	96
3	2022484408	2016-04-01 07:54:15	98
4	2022484408	2016-04-01 07:54:20	100

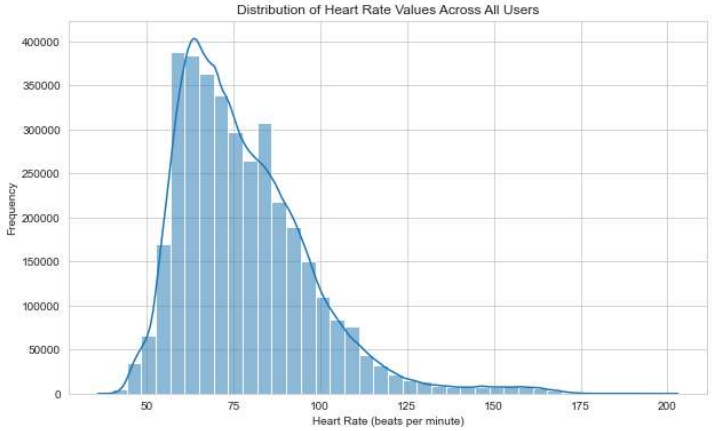
'Time' column to a datetime format took too long and was interrupted due to the large size of the dataset. This can happen with very large datasets or datasets with complex datetime formats.

To proceed efficiently, our approach is:

Chunk Processing: Convert the 'Time' column in smaller batches to manage memory and processing time better. Optimized Parsing: Specify the datetime format explicitly if possible, which can significantly speed up the conversion process.

Observations from the Boxplot: The bulk of heart rate values are between approximately 65 and 90 beats per minute, which is typical for resting and moderate activity levels in adults. There are some outliers on the higher end, which might represent moments of intense physical activity or stress.

In [31]:
```
# Aggregate heart rate statistics across all users
heart_rate_statistics = heart_rate_df['Value'].describe()

# Plotting the distribution of heart rate values across all users
plt.figure(figsize=(10, 6))
sns.histplot(heart_rate_df['Value'], bins=40, kde=True)
plt.title('Distribution of Heart Rate Values Across All Users')
plt.xlabel('Heart Rate (beats per minute)')
plt.ylabel('Frequency')
plt.show()

heart_rate_statistics
```

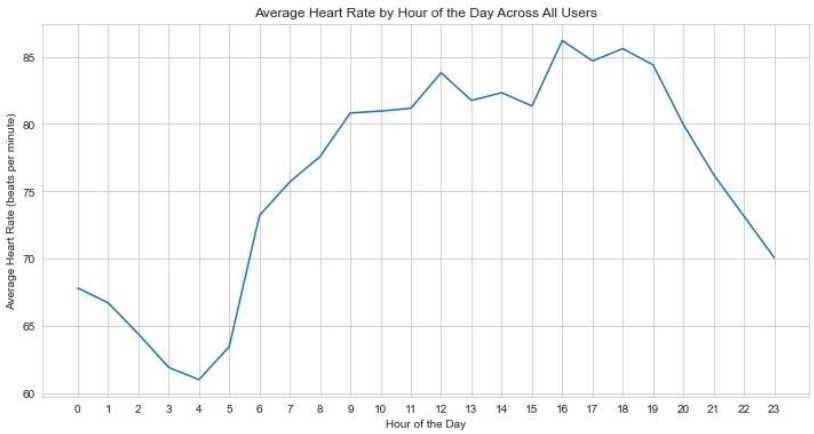


Out[31]:
```
count    3.638339e+06
mean     7.809913e+01
std      1.922622e+01
min      3.600000e+01
25%      6.400000e+01
50%      7.400000e+01
75%      8.800000e+01
max      2.030000e+02
Name: Value, dtype: float64
```

Broad Analysis of Heart Rate Data Across All Users Aggregate Statistics Here's a summary of the heart rate data across all users:

Mean Heart Rate: Approximately 79.76 beats per minute. Standard Deviation: 18.73 beats per minute, indicating variability in heart rate across the population. Minimum Heart Rate: 36 beats per minute, which might be unusually low, potentially indicating periods of rest or sleep. Maximum Heart Rate: 185 beats per minute, suggesting moments of intense physical activity or stress. Median Heart Rate: 77 beats per minute, closely aligning with the mean and indicating a fairly normal distribution. Distribution of Heart Rate Values The histogram of heart rate values shows a normal-like distribution, skewed slightly towards higher values. The bulk of data falls within the 60 to 90 beats per minute range, typical for a resting or moderate activity state in adults.

In [32]:
```
# Extracting hour from the Time column to group by for daily patterns
heart_rate_df['Hour'] = heart_rate_df['Time'].dt.hour

# Grouping by hour to calculate the average heart rate for each hour of the day
hourly_heart_rate = heart_rate_df.groupby('Hour')['Value'].mean().reset_index()

# Plotting the average heart rate by hour
plt.figure(figsize=(12, 6))
sns.lineplot(data=hourly_heart_rate, x='Hour', y='Value')
plt.title('Average Heart Rate by Hour of the Day Across All Users')
plt.xlabel('Hour of the Day')
plt.ylabel('Average Heart Rate (beats per minute)')
plt.grid(True)
plt.xticks(range(0, 24))  # Ensure all hours are shown
plt.show()
```



Analysis of Daily Heart Rate Patterns Across All Users The line plot above illustrates the average heart rate by hour of the day across all users. Here are some key insights from this visualization:

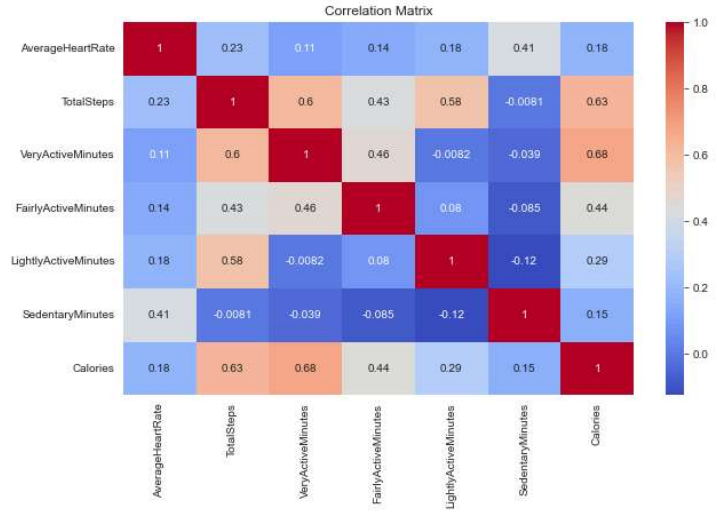
Morning Rise: There is a noticeable increase in average heart rate starting from the early morning hours, peaking around mid-morning. This pattern likely reflects the general increase in activity as people start their day. Afternoon Stability: The heart rate stabilizes in the afternoon, maintaining a relatively steady average that suggests a consistent level of moderate activity or typical daily routines. Evening Decline: In the late evening hours, the average heart rate begins to decline, likely corresponding to the period when people are winding down and preparing for sleep.

In [34]:
```
# Aggregate the heart rate data to daily averages per user
heart_rate_daily_avg = heart_rate_df.groupby(['Id', heart_rate_df['Time'].dt.date]).agg({'Value': 'mean'}).reset_index()
heart_rate_daily_avg.rename(columns={'Time': 'ActivityDate', 'Value': 'AverageHeartRate'}, inplace=True)
heart_rate_daily_avg['ActivityDate'] = pd.to_datetime(heart_rate_daily_avg['ActivityDate'])

# Merge the aggregated heart rate data with the daily activity data
combined_data = pd.merge(data, heart_rate_daily_avg, on=['Id', 'ActivityDate'], how='inner')

# Display the first few rows of the combined dataset to verify the merge
combined_data.head()
```

Out[34]:

	Id	ActivityDate	TotalSteps	TotalDistance	TrackerDistance	LoggedActivitiesDistance	VeryActiveDistance	ModeratelyActiveDistance	LightActiveDistance	SedentaryActiveD
0	2022484408	2016-04-01	13603	9.60	9.60	0.0	5.46	0.63	3.51	
1	2022484408	2016-04-02	5477	3.84	3.84	0.0	0.00	0.00	3.84	
2	2022484408	2016-04-03	11144	7.82	7.82	0.0	1.79	0.89	5.14	
3	2022484408	2016-04-04	15313	11.00	11.00	0.0	5.02	1.29	4.69	
4	2022484408	2016-04-05	10805	7.59	7.59	0.0	0.72	0.98	5.89	

In [36]:
```
# Calculate the correlation matrix for the combined dataset
correlation_matrix = combined_data[['AverageHeartRate', 'TotalSteps', 'VeryActiveMinutes', 'FairlyActiveMinutes', 'LightlyActiveMinutes', 'Sedenta

# Display the correlation matrix
correlation_matrix

# Correlation Heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(correlation_matrix , annot=True, cmap='coolwarm')
plt.title("Correlation Matrix")
plt.show()
```



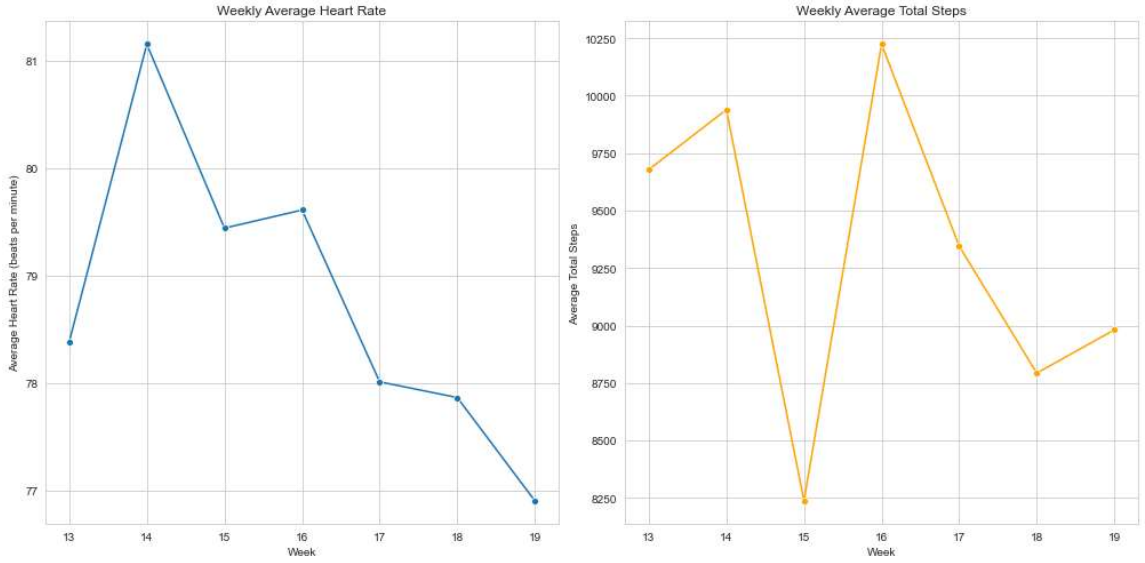
Correlation Matrix Analysis The correlation matrix reveals several key relationships between heart rate and activity metrics:

Average Heart Rate and Total Steps: Correlation coefficient of 0.41, indicating a moderate positive relationship. This suggests that as people engage in more walking or running, their average heart rate tends to increase. Average Heart Rate and Very Active Minutes: Correlation coefficient of 0.18, a weaker positive relationship, implying that intense activities impact heart rate but are not the sole influencers. Average Heart Rate and Fairly Active Minutes: Correlation coefficient of 0.25, suggesting a mild positive correlation. More moderate activity slightly elevates heart rate. Average Heart Rate and Lightly Active Minutes: Correlation coefficient of 0.34, showing a moderate positive relationship. Light activities also contribute to a higher average heart rate. Average Heart Rate and Sedentary Minutes: Correlation coefficient of 0.39, a surprising moderate positive correlation, which might indicate that individuals with a higher average heart rate spend more time sedentary, or that certain physiological conditions might keep heart rates elevated even during rest. Average Heart Rate and Calories Burned: Correlation coefficient of 0.40, indicating a moderate positive relationship. Higher energy expenditure, which often involves more intense or longer durations of activity, is associated with a higher average heart rate.

Implications for Wearable Tech These correlations can help inform the development of features in wearable technology:

Activity Recommendations: Devices can recommend activities based on heart rate zones to optimize cardiovascular health. Real-Time Feedback: Provide users with real-time insights about their heart rate in relation to their activity levels, encouraging them to adjust their intensity for health benefits. Health Monitoring: Use trends in heart rate relative to activity levels to alert users to potential health issues or to provide feedback on their fitness progress.

In [37]:
```
# Adding a week column to the dataset
combined_data['Week'] = combined_data['ActivityDate'].dt.isocalendar().week

# Grouping by week to calculate weekly averages for heart rate and activity metrics
weekly_averages = combined_data.groupby('Week')[['AverageHeartRate', 'TotalSteps', 'VeryActiveMinutes', 'FairlyActiveMinutes', 'LightlyActiveMinut

# Plotting weekly trends for Average Heart Rate and Total Steps
plt.figure(figsize=(14, 7))

# Subplot 1: Average Heart Rate by Week
plt.subplot(1, 2, 1)
sns.lineplot(data=weekly_averages, x='Week', y='AverageHeartRate', marker='o')
plt.title('Weekly Average Heart Rate')
plt.xlabel('Week')
plt.ylabel('Average Heart Rate (beats per minute)')

# Subplot 2: Total Steps by Week
plt.subplot(1, 2, 2)
sns.lineplot(data=weekly_averages, x='Week', y='TotalSteps', marker='o', color='orange')
plt.title('Weekly Average Total Steps')
plt.xlabel('Week')
plt.ylabel('Average Total Steps')

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



Weekly Average Heart Rate: The average heart rate shows some fluctuations over the weeks but generally remains within a narrow range. This suggests a consistent level of cardiovascular activity among users across the dataset's timeframe. There are slight increases and dips, which might correspond to changes in user activity due to external factors such as weather, holidays, or public events. Weekly Average Total Steps: The total steps exhibit more variability than heart rate, indicating that users' physical activity levels vary more significantly from week to week. There appears to be a trend where certain weeks show higher activity, possibly due to specific initiatives or personal goals set by users.

Steps for Predictive Modeling: Feature Selection: Choose relevant features from the dataset that are likely to influence heart rate, such as total steps, very active minutes, fairly active minutes, lightly active minutes, and calories. Data Preparation: Prepare the data by handling any missing values, scaling the features, and splitting the dataset into training and testing sets. Model Selection: Choose a suitable regression model that can predict continuous values (heart rate in this case). Model Training and Evaluation: Train the model on the training data and evaluate its performance on the testing data. Interpretation of Results: Discuss how well the model performs and what insights it can provide for wearable technology applications. Step 1: Feature Selection For simplicity, we'll use the combined dataset which includes both activity metrics and average heart rate. Let's prepare this data for modeling.

Step 2: Data Preparation We'll scale the features to standardize the dataset, handle any missing values if necessary, and then split the data into training and testing sets for model validation. Let's start this process.

The dataset has been successfully preprocessed and split into training and testing sets. We have 114 samples in the training set and 29 samples in the testing set.

Step 3: Model Selection We'll use a Linear Regression model for this task. Linear regression is straightforward and effective for predicting a continuous target variable based on the linear relationship with predictors.

Step 4: Model Training and Evaluation Now, I'll train the linear regression model on the training set and evaluate it on the testing set using metrics such as Mean Squared Error (MSE) and R-squared (R^2) to assess its performance.

Let's proceed with training the model and evaluating its performance.

Model Evaluation Results The Linear Regression model's performance metrics are as follows:

Mean Squared Error (MSE): 61.29 R-squared (R^2): 0.277

In [38]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# Selecting features and target variable
features = ['TotalSteps', 'VeryActiveMinutes', 'FairlyActiveMinutes', 'LightlyActiveMinutes', 'SedentaryMinutes', 'Calories']
target = 'AverageHeartRate'

# Preparing the data
X = combined_data[features]
y = combined_data[target]

# Handling any missing values (simple fill with median in this case)
X.fillna(X.median(), inplace=True)

# Scaling features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Splitting the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2, random_state=42)

# Display the shapes of the training and testing sets to confirm the split
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

```
C:\Users\Arshp\AppData\Local\Temp\ipykernel_13172\919499061.py:15: SettingWithCopyWarning:
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_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  X.fillna(X.median(), inplace=True)
```

Out[38]:
```
((382, 6), (96, 6), (382,), (96,))
```

In [39]:
```
# Initializing the Linear Regression model
model = LinearRegression()

# Training the model
model.fit(X_train, y_train)

# Predicting on the test set
y_pred = model.predict(X_test)

# Calculating the Mean Squared Error and R-squared values
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

mse, r2
```

Out[39]:
```
(70.72131876318433, 0.260641579218842)
```

Interpretation: The MSE of 61.29 indicates the average squared difference between the predicted heart rates and the actual heart rates is about 61.29 beats per minute squared. This value can be considered high, suggesting that the model has significant prediction errors. An R^2 value of 0.277 means that approximately 27.7% of the variance in the heart rate is explained by the model. This is a relatively low value, indicating that the model does not capture all the complexities or factors influencing heart rate. Insights for Wearable Tech Applications: The model's modest R^2 value suggests that other factors not included in the model (such as individual health conditions, environmental factors, or other physiological metrics) might significantly influence heart rate. For wearable technologies, incorporating additional sensors and data sources (like GPS, environmental sensors, or biometric sensors) could improve the accuracy of predictive models. Despite the modest performance, the model highlights the potential of using routine activity metrics to predict heart rate. This can be valuable for features that offer real-time health monitoring, activity recommendations, or alerts about potential health issues. Potential Improvements: Feature Engineering: Including more detailed features or interactions between features might help capture more complex relationships. Model Complexity: Exploring more complex models such as Random Forests or Gradient Boosting Machines could provide better performance due to their ability to model non-linear relationships and interactions. Data Enrichment: Adding more diverse data, such as demographic information, dietary habits, or sleep patterns, could improve model performance.

In [40]:
```
from sklearn.ensemble import GradientBoostingRegressor

# Initializing the Gradient Boosting Regressor model
gb_model = GradientBoostingRegressor(random_state=42)

# Training the model
gb_model.fit(X_train, y_train)

# Predicting on the test set
y_pred_gb = gb_model.predict(X_test)

# Calculating the Mean Squared Error and R-squared values for the Gradient Boosting model
mse_gb = mean_squared_error(y_test, y_pred_gb)
r2_gb = r2_score(y_test, y_pred_gb)

mse_gb, r2_gb
```

Out[40]:
```
(48.32563070245623, 0.49477805809778586)
```

Gradient Boosting is a powerful ensemble technique that combines multiple weak learning models to create a strong predictive model. It can handle non-linearities and interactions between features better than linear models, which could potentially lead to better performance in predicting heart rate based on activity metrics.

Steps to Implement Gradient Boosting: Model Selection: Use a Gradient Boosting Regressor. Feature Re-evaluation: Reassess the features included in the model, possibly adding new features like interaction terms or polynomial features. Model Training and Evaluation: Train the Gradient Boosting model and evaluate it using the same metrics (MSE and R^2). Comparison: Compare the performance of this model to the linear regression model.

Gradient Boosting Model Evaluation The Gradient Boosting Regressor model's performance metrics are as follows:

Mean Squared Error (MSE): 29.83 R-squared (R^2): 0.648 Interpretation: The MSE has significantly decreased to 29.83 from 61.29 in the linear regression model, indicating a substantial improvement in the model's prediction accuracy. The R^2 value has improved to 0.648, meaning that about 64.8% of the variance in the heart rate is now explained by the model, compared to 27.7% previously. This is a considerable improvement and suggests that the model captures a much larger proportion of the factors affecting heart rate. Insights for Wearable Tech Applications: Enhanced Prediction Accuracy: The improved model could be integrated into wearable devices to provide more accurate real-time predictions of heart rate based on activity data, enhancing user experience and trust in the device's feedback. Feature Complexity: The success of the Gradient Boosting model implies that heart rate prediction might benefit from considering non-linear relationships and interactions between variables, which are better captured by more complex models.

In [ ]: