Homework 1 - Coding

In this problem you will generate plots for the bike sharing demand dataset in this notebook. Please follow the instructions in markdown cells and comments, and add your code after the comment "write your code here." Most plots were already introduced in the first lab. See https://github.com/tsourolampis/cs365-spring23/blob/main/lab/Lab1.ipynb.

Download the dataset from sklearn

In [19]:
```
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_openml
import warnings
warnings.filterwarnings("ignore")

bike_sharing = fetch_openml(
    "Bike_Sharing_Demand", version=2, as_frame=True
)

df = bike_sharing.frame
```

In [8]:
```
display(df)
```

	season	year	month	hour	holiday	weekday	workingday	weather	temp	feel_tem
0	spring	0.0	1.0	0.0	False	6.0	False	clear	9.84	14.39
1	spring	0.0	1.0	1.0	False	6.0	False	clear	9.02	13.63
2	spring	0.0	1.0	2.0	False	6.0	False	clear	9.02	13.63
3	spring	0.0	1.0	3.0	False	6.0	False	clear	9.84	14.39
4	spring	0.0	1.0	4.0	False	6.0	False	clear	9.84	14.39
...	...	...	...	...	...	...	...	...	...	
17374	spring	1.0	12.0	19.0	False	1.0	True	misty	10.66	12.88
17375	spring	1.0	12.0	20.0	False	1.0	True	misty	10.66	12.88
17376	spring	1.0	12.0	21.0	False	1.0	True	clear	10.66	12.88
17377	spring	1.0	12.0	22.0	False	1.0	True	clear	10.66	13.63
17378	spring	1.0	12.0	23.0	False	1.0	True	clear	10.66	13.63

17379 rows × 13 columns

Correlation matrix [5pts]

Correlation is a measure that quantifies the degree to which a pair of variables are linearly related. Intuitively, with a positive correlation between A and B, we will likely see B increase as A increases.

Formally, the correlation is defined as normalized covariance, i.e., $corr(A,B) = \frac{cov(A,B)}{\sigma_A \sigma_B}$, where $\sigma_A = \sqrt{Var(A)}$ is the standard deviation of A.

In Python, the correlation between columns of a dataframe can be calculated using dataframe.corr() function. Generate a correlation matrix of four features "temp", "feel_temp", "humidity" and "windspeed". Which two features are most positively correlated?

In [8]:
```
df.drop(['season','year','month','hour','weekday','count'],axis=1).corr()
```

Out[8]:

	temp	feel_temp	humidity	windspeed
temp	1.000000	0.987672	-0.069881	-0.023125
feel_temp	0.987672	1.000000	-0.051918	-0.062336
humidity	-0.069881	-0.051918	1.000000	-0.290105
windspeed	-0.023125	-0.062336	-0.290105	1.000000

Answer: Temp and feel_temp are the two features that are most positively correlated

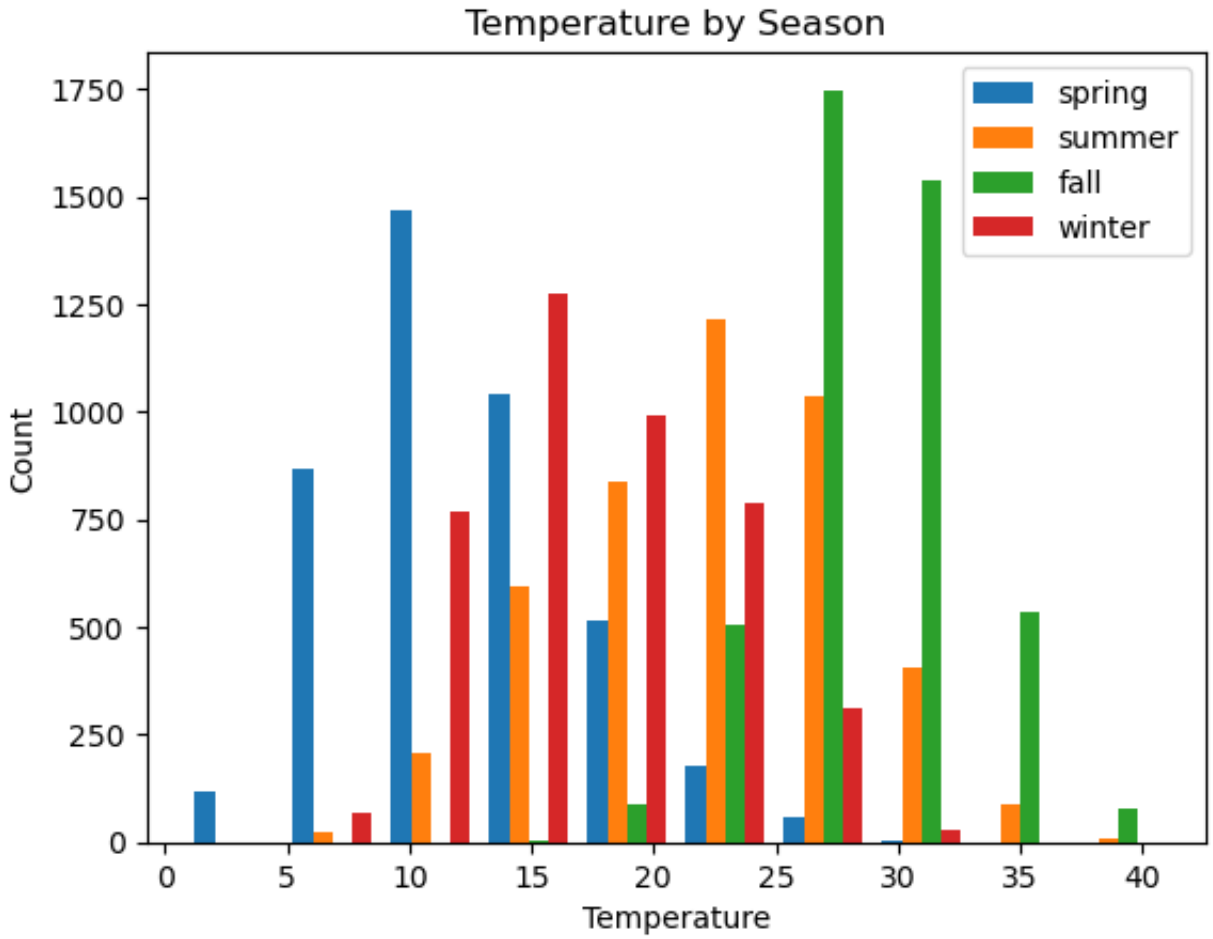
Histogram of temperatures [5pts]

How does temperature change by season? You are asked to generate histograms of temperature values across seasons using different legends. Use histograms with shifts to avoid overlapping.

Use the label parameter in plt.hist to assign labels to different histograms, and use plt.legend() to show the labels in the figure.

In [9]:
```
type0 = df[df['season'] == 'spring']
type1 = df[df['season'] == 'summer']
type2 = df[df['season'] == 'fall']
type3 = df[df['season'] == 'winter']

plt.figure(figsize=(8,10))

common_params = dict(bins=30,
                     range=(0, 60))

a = type0['temp'].values
b = type1['temp'].values
c = type2['temp'].values
d = type3['temp'].values

plt.subplots_adjust(hspace = .4)


fig, ax = plt.subplots(1, 1)
ax.set_xlabel('Temperature')
ax.set_ylabel('Count')

plt.title('Temperature by Season')
plt.hist((a,b,c,d))

ax.legend(('spring', 'summer','fall','winter'), loc='upper right');


plt.show()
```

```
<Figure size 800x1000 with 0 Axes>
```



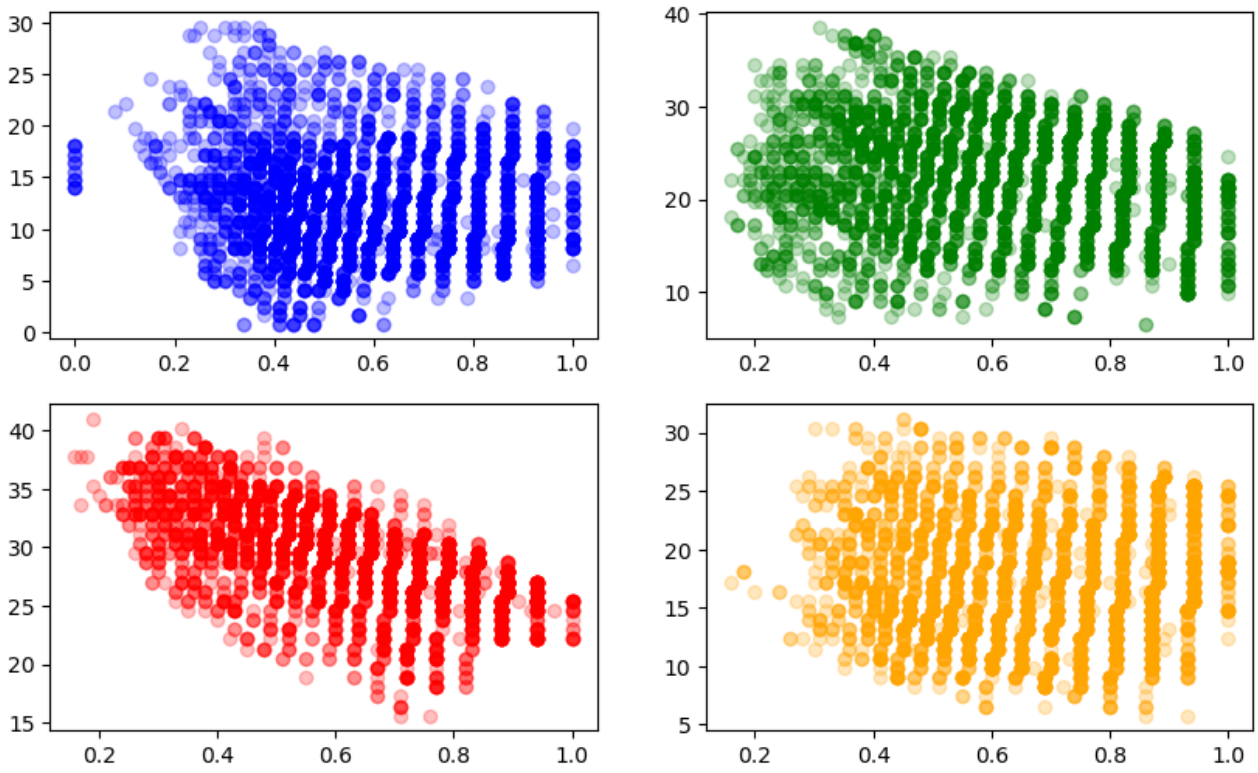
Answer: Generally, spring has the lowest temperature. On summer, the temperature rises compared to spring and remains between 15 degrees and 30 degrees(about median). On fall, the temperature rises again and marks the highest temperature of the data. Finally, when winter approaches, the temperature drops back to about median temperature (between 12 and 24) and is similar to the temperature of spring (slightly colder than spring).

Scatter plot [5pts]

You are asked to generate a figure with four axes. Each axis should show a scatter plot of humidity v.s. temp within one season. You can use the alpha parameter in plt.scatter to adjust the transparency of points.

Which season is unique concerning humidity and temp, and why?

In [32]:
```
fig, ax = plt.subplots(2,2)
fig.set_size_inches(10,6)

ax[0,0].scatter(type0['humidity'], type0['temp'], label = "spring", facecolc

ax[0,1].scatter(type1['humidity'], type1['temp'], label = "summer", facecolc

ax[1,0].scatter(type2['humidity'], type2['temp'], label = "fall", facecolor

ax[1,1].scatter(type3['humidity'], type3['temp'], label = "winter", facecolc



plt.show()
```



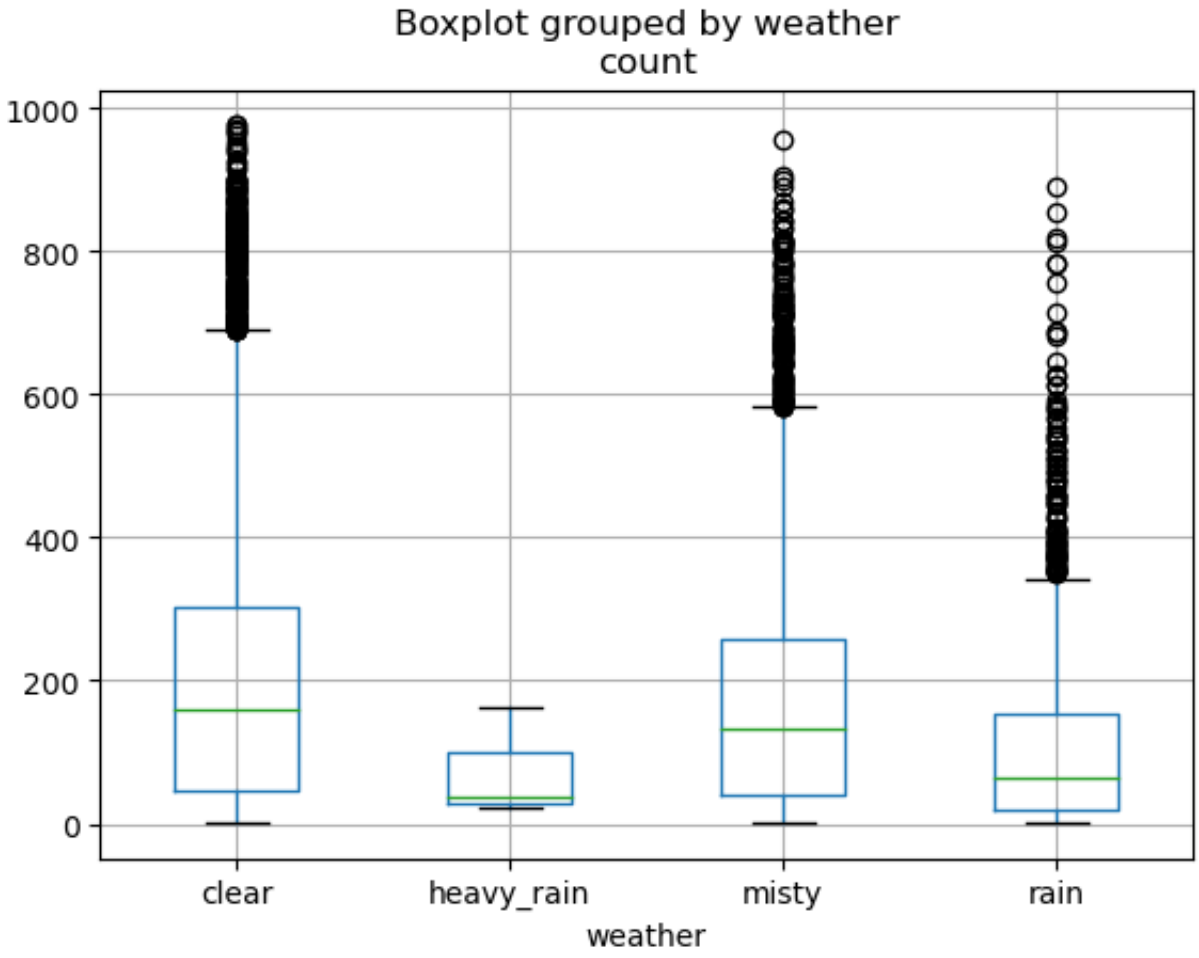
Answer: Fall is most unique concerning humidity and temperature because unlike other seasons (spring, fall, and winter) that have have data distributed for all values of humidity and temperature and does not have clear pattern, the datapoints of fall seem to have a relationship between humidity and temperature (as humidity for fall increases, the temperature decreases).

Boxplot [5pts]

Pandas.dataframe support the function boxplot(column, by, *other params). Here "column" takes one or a list of column names, and "by" takes another column name. One boxplot will be done per value of columns in "by".

Please use this function to draw boxplots of the "count" column across different "weather" conditions. In this plot, the x-axis should list four weather conditions, and the y-axis should represent the bike sharing count.

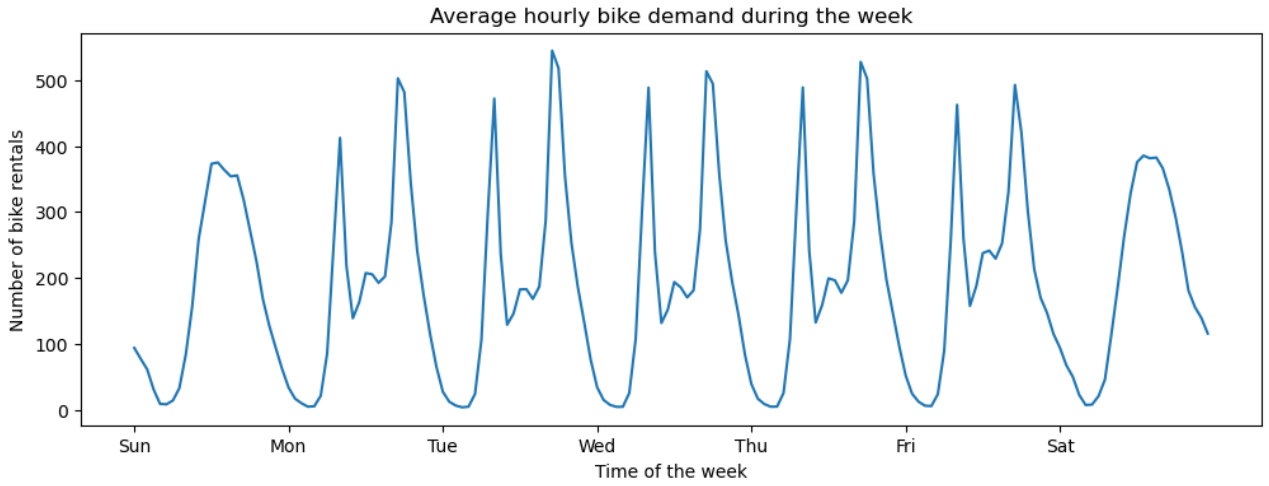
In [11]:
```
df.boxplot('count','weather')
plt.show()
```



Time series [5pts]

Now let's have a look at time series data. Group the filtered_df by "weekday" and "hour", then calculate the mean of "count". Plot how the mean of "count" changes in a week using plot() function.

In [12]:
```
filtered_df = df[['weekday','hour','count']]
fig, ax = plt.subplots(figsize=(12, 4))

# write your code here

filtered_df.groupby(['weekday','hour']).mean()['count'].plot()



# Set up ticks and labels.
_ = ax.set(
    title="Average hourly bike demand during the week",
    xticks=[i * 24 for i in range(7)],
    xticklabels=["Sun", "Mon", "Tue", "Wed", "Thu", "Fri", "Sat"],
    xlabel="Time of the week",
    ylabel="Number of bike rentals",
)
```



Fast Fourier Transform

You should see that the mean of "count" shows a daily pattern from Monday to Friday.

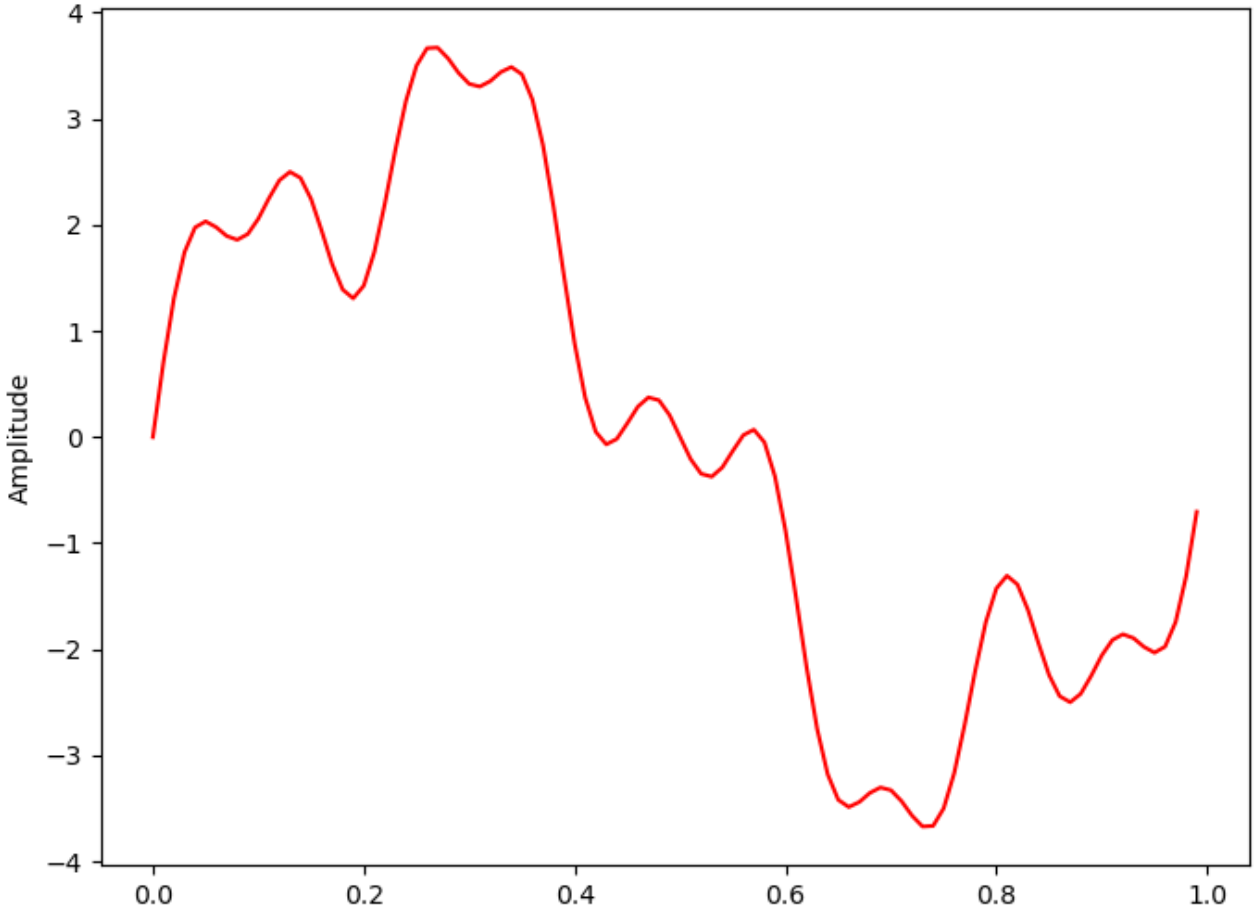
Fourier Transform is a widely used method to understand sequential data. While this will be covered in future lectures, in this notebook, we want to show how to visualize time series data from the frequency domain using FFT and its inverse in Python. A fast Fourier transform (FFT) is an algorithm that computes the discrete Fourier transform (DFT) of a sequence or its inverse (IDFT). Fourier analysis converts a signal from its original domain (often time or space) to a representation in the frequency domain and vice versa.

In [13]:
```
from numpy.fft import fft, ifft
import numpy as np
```

Example

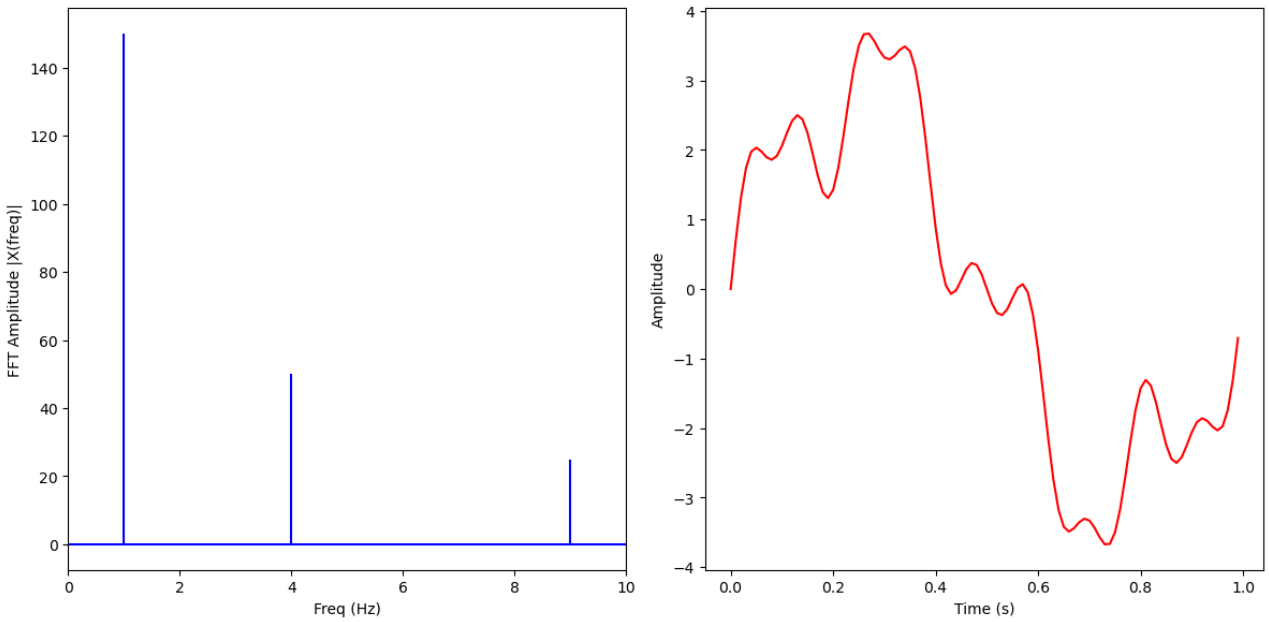
Below is an example of a sequence composed of signals from three different frequencies. We visualize this complicated sequence using the plt.plot function.

In [14]:
```
# sampling rate
sr = 100
# sampling interval
ts = 1.0/sr
t = np.arange(0,1,ts)

freq = 1.
x = 3*np.sin(2*np.pi*freq*t)

freq = 4
x += np.sin(2*np.pi*freq*t)

freq = 9
x += 0.5* np.sin(2*np.pi*freq*t)

plt.figure(figsize = (8, 6))
plt.plot(t, x, 'r')
plt.ylabel('Amplitude')

plt.show()
```



In the next cell, we convert the sequence from the time domain into the frequency domain using the FFT function supported by numpy. The result of FFT is shown in the first subplot. We can observe that FFT recovers all three frequencies we planted. Then we use the IFFT function to convert the signal back to the time domain and plot it in the second subplot.

In [15]:
```
X = fft(x)
N = len(X)
n = np.arange(N)
T = N/sr
freq = n/T

plt.figure(figsize = (12, 6))
plt.subplot(121)

plt.stem(freq, np.abs(X), 'b', \
         markerfmt=" ", basefmt="-b")
plt.xlabel('Freq (Hz)')
plt.ylabel('FFT Amplitude |X(freq)|')
plt.xlim(0, 10)

plt.subplot(122)
plt.plot(t, ifft(X), 'r')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.tight_layout()
plt.show()
```



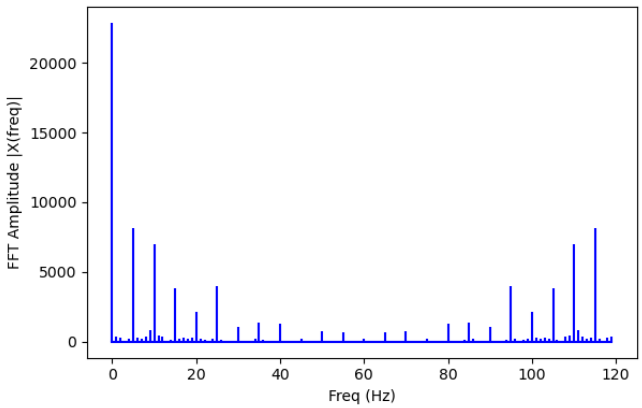
Question: What do you observe in the above plots?

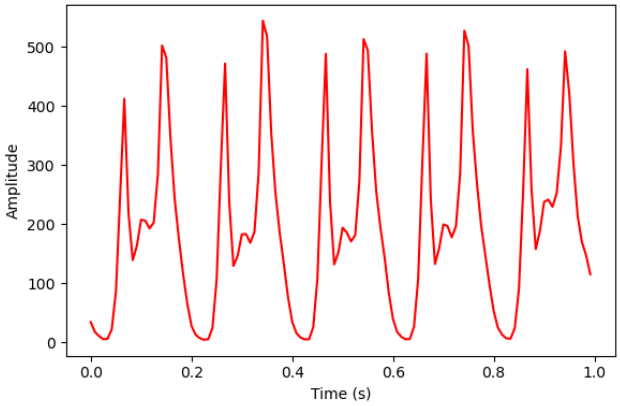
Answer: The graph on the left shows both the changes in amplitude of the graph on the right with each frequency (Hz) value and the number of times each change in amplitude of the graph on the right occurs. For example, on the graph on the right, the amplitude changes with frequency (Hz) of 1 about 150 times, of 4 about 50 times, of 9 about 25 times.

FFT on the sequential data [5pts]

Follow the worked out example in the previous cell by applying the FFT function on the mean of the "count" from Monday to Friday. After that, use the IFFT function to convert the signal back to the time domain. Plot both results in one figure.

In [17]:
```
filtered_df = df[['weekday','hour','count']]
indexDrop = filtered_df[ (filtered_df['weekday'] == 0.0)].index
filtered_df.drop(indexDrop, inplace=True)
indexDrop = filtered_df[ (filtered_df['weekday'] == 6.0)].index
filtered_df.drop(indexDrop, inplace=True)
graphTo = filtered_df.groupby(['weekday','hour']).mean()['count']




# sampling rate
sr = 120
# sampling interval
ts = 1.0/sr
t = np.arange(0,1,ts)




X = fft(graphTo)
N = len(X)
n = np.arange(N)
T = N/sr
freq = n/T



plt.figure(figsize = (12, 4))
plt.subplot(121)

plt.stem(freq, np.abs(X), 'b', \
         markerfmt=" ", basefmt="-b")
plt.xlabel('Freq (Hz)')
plt.ylabel('FFT Amplitude |X(freq)|')

plt.subplot(122)
plt.plot(t, ifft(X), 'r')
plt.xlabel('Time (s)')
plt.ylabel('Amplitude')
plt.tight_layout()
plt.show()
```


FFT Amplitude |X(freq)|
Freq (Hz)
0
5000
10000
15000
20000
0
20
40
60
80
100
120


Amplitude
Time (s)
0
100
200
300
400
500
0.0
0.2
0.4
0.6
0.8
1.0

In [ ]: