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 [ ]:
```
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 [ ]:
```
# print the dataframe
print(df)
```

```
       season  year  month  hour holiday  weekday workingday weather   temp  \
0      spring     0      1     0   False        6      False   clear   9.84
1      spring     0      1     1   False        6      False   clear   9.02
2      spring     0      1     2   False        6      False   clear   9.02
3      spring     0      1     3   False        6      False   clear   9.84
4      spring     0      1     4   False        6      False   clear   9.84
...       ...   ...    ...   ...     ...      ...        ...     ...    ...
17374  spring     1     12    19   False        1       True   misty  10.66
17375  spring     1     12    20   False        1       True   misty  10.66
17376  spring     1     12    21   False        1       True   clear  10.66
17377  spring     1     12    22   False        1       True   clear  10.66
17378  spring     1     12    23   False        1       True   clear  10.66

       feel_temp  humidity  windspeed  count
0         14.395      0.81     0.0000     16
1         13.635      0.80     0.0000     40
2         13.635      0.80     0.0000     32
3         14.395      0.75     0.0000     13
4         14.395      0.75     0.0000      1
...          ...       ...        ...    ...
17374     12.880      0.60    11.0014    119
17375     12.880      0.60    11.0014     89
17376     12.880      0.60    11.0014     90
17377     13.635      0.56     8.9981     61
17378     13.635      0.65     8.9981     49

[17379 rows x 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 [ ]:
```
# write your code here

m=df[["temp","feel_temp","humidity","windspeed"]]
corr_m=m.corr()
print(corr_m)
print("\nfeel_temp and temp are most positively correlated.")
```

```
                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

feel_temp and temp are most positively correlated.
```

Histogram of temperatures [5pts]

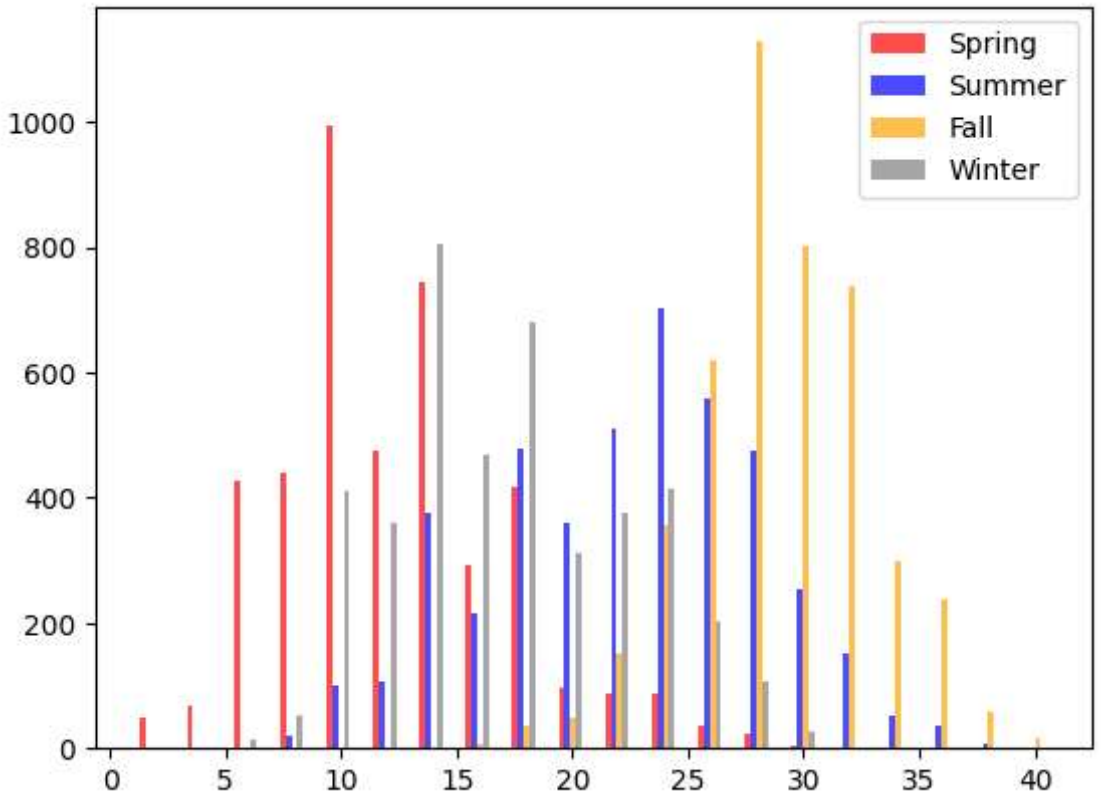
How does temperature change by season? You are asked to generate histograms of temperature values across seasons using different legends. Use histograms with shifts to avoid overlapping.

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 [ ]:
```
# write your code here
m = df[["temp","season"]]
spring = m.loc[m["season"]=="spring"]["temp"]
summer = m.loc[m["season"]=="summer"]["temp"]
fall = m.loc[m["season"]=="fall"]["temp"]
winter = m.loc[m["season"]=="winter"]["temp"]


#spring.plot.hist(bins=20,histtype = 'bar', alpha = 0.5, rwidth = 0.5,color="r",
#summer.plot.hist(bins=20,histtype = 'bar', alpha = 0.5, rwidth = 0.5, label = "
#fall.plot.hist(bins=20,histtype = 'bar', alpha = 0.5, rwidth = 0.5, label = "Fa
#winter.plot.hist(bins=20,histtype = 'bar', alpha = 0.8, rwidth = 0.5, label = "
plt.hist((spring,summer,fall,winter),bins=20,histtype = 'bar', alpha = 0.7, rwic
```

```
          color=("red","blue","orange","grey"))
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x2180dbc5d90>



Scatter plot [5pts]

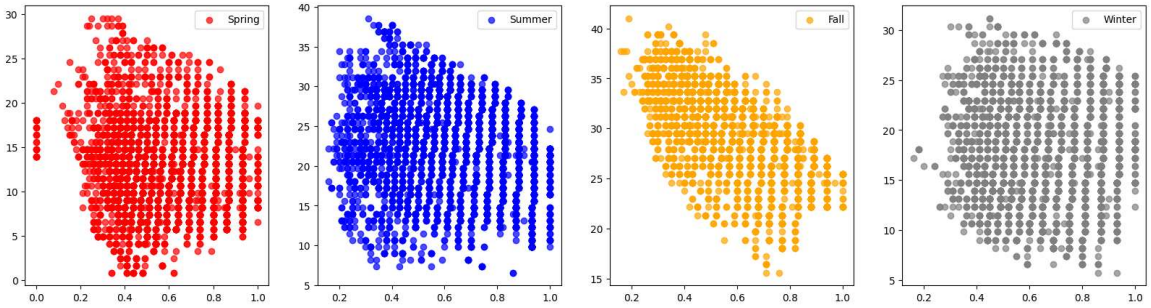
You are asked to generate a figure with four axes. Each axis should show a scatter plot of humidity v.s. temp within one season. You can use the alpha parameter in plt.scatter to adjust the transparency of points.

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

In [ ]:
```
# write your code here

ht = df[["humidity","temp","season"]]
spring = ht.loc[m["season"]=="spring"]
summer = ht.loc[m["season"]=="summer"]
fall = ht.loc[m["season"]=="fall"]
winter = ht.loc[m["season"]=="winter"]
plt.figure(figsize=(20,5))

plt.subplot(1,4,1)
plt.scatter(spring['humidity'], spring['temp'], label="Spring", facecolor="red",
plt.legend()
plt.subplot(1,4,2)
plt.scatter(summer['humidity'], summer['temp'], label="Summer", facecolor="blue"
plt.legend()
plt.subplot(1,4,3)
plt.scatter(fall['humidity'], fall['temp'], label="Fall", facecolor="orange",alp
plt.legend()
plt.subplot(1,4,4)
```

```
plt.scatter(winter['humidity'], winter['temp'], label="Winter", facecolor="grey"
plt.legend()
```

Out[ ]: <matplotlib.legend.Legend at 0x21810c58e50>



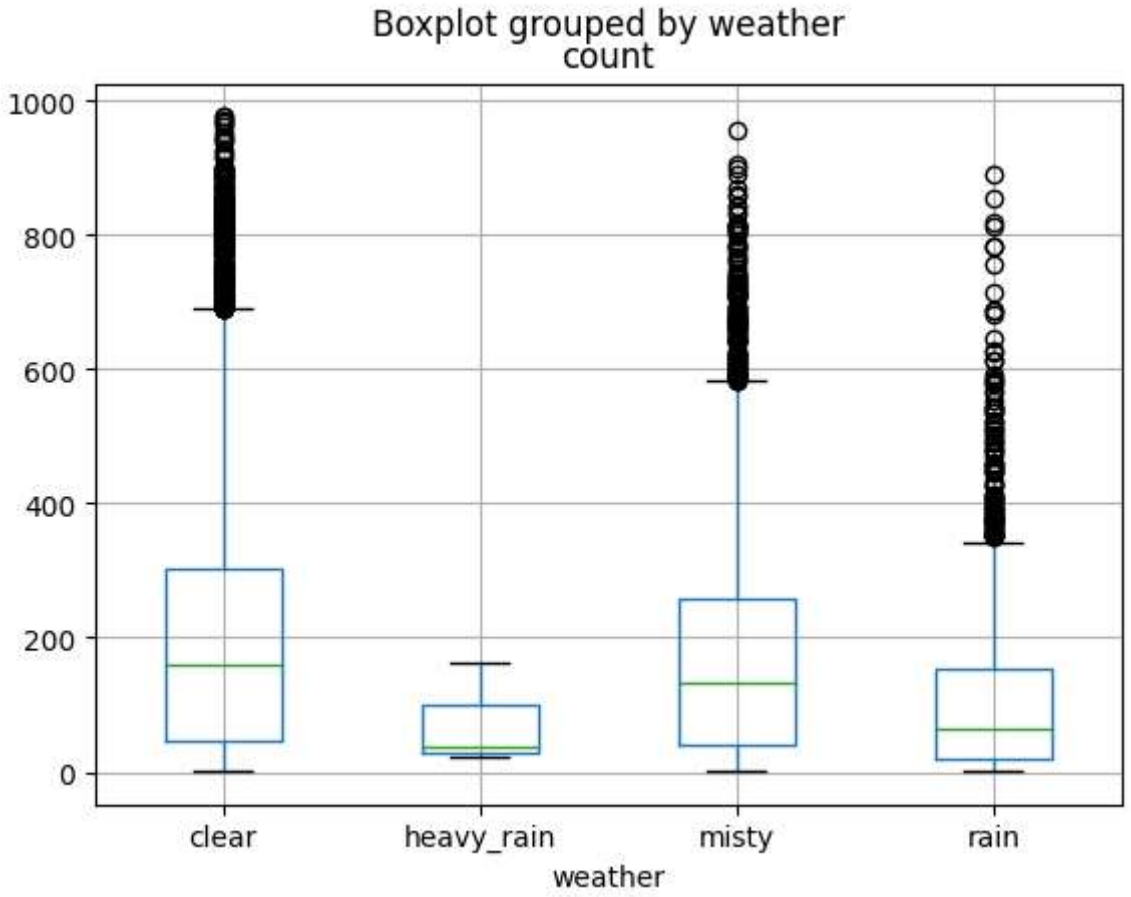
Boxplot [5pts]

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 [ ]:
```
# write your code here
df.boxplot(by ='weather', column =['count'])
```

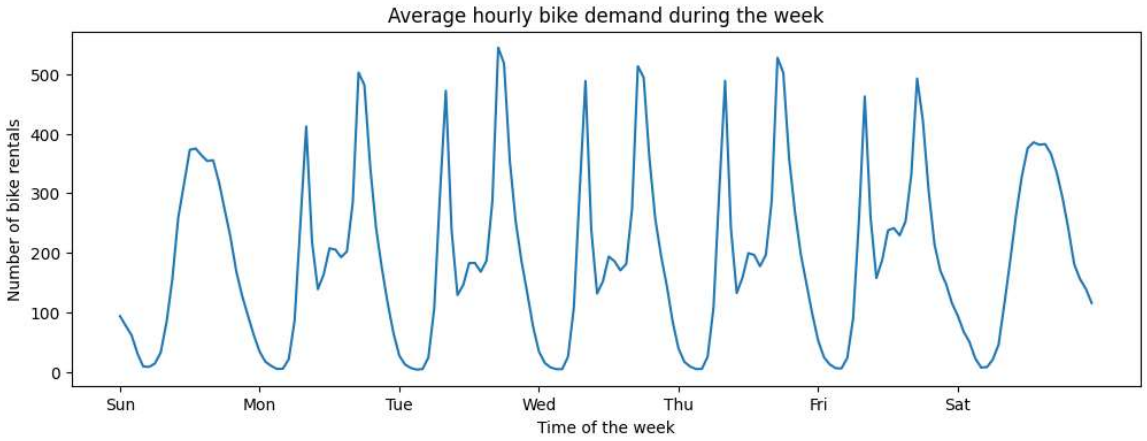
Out[ ]: <AxesSubplot: title={'center': 'count'}, xlabel='weather'>



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

# write your code here
filtered_df.groupby(['weekday','hour'])['count'].mean().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.

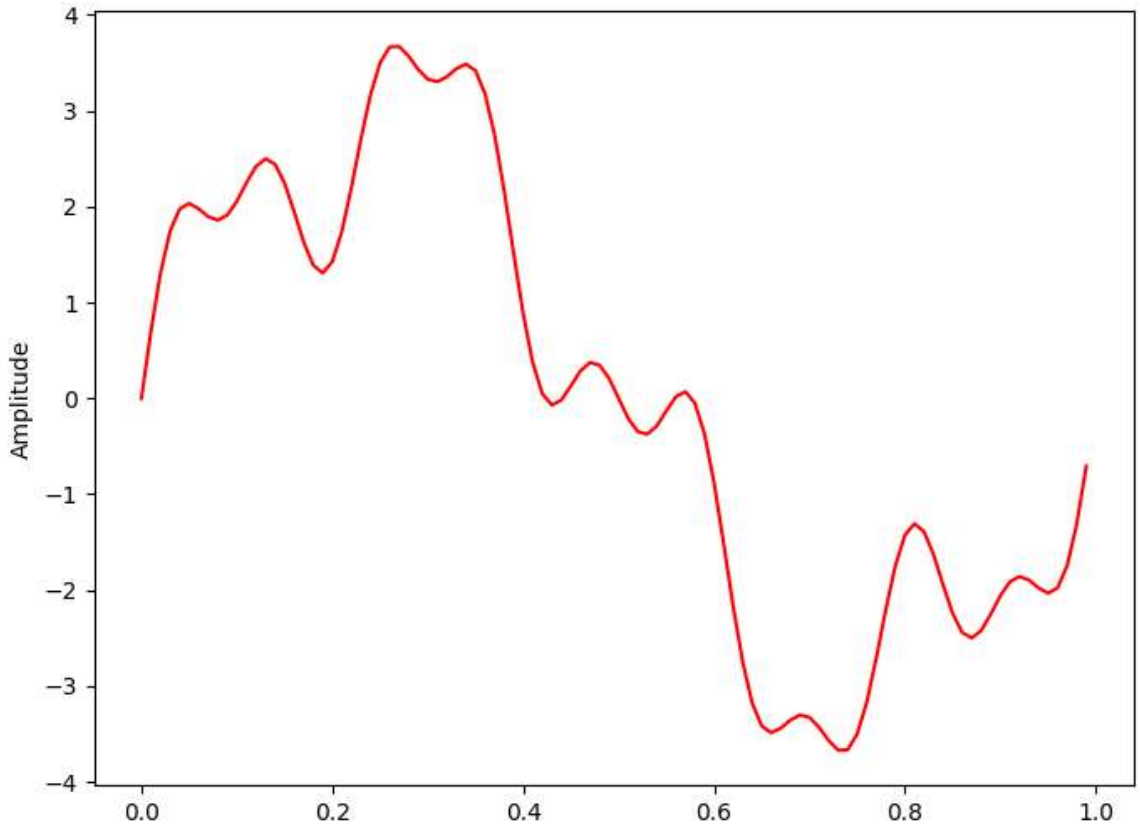
Fourier Transform is a widely used method to understand sequential data. While this will be covered in future lectures, in this notebook, we want to show how to visualize time series data from the frequency domain using FFT and its inverse in Python. A fast Fourier transform (FFT) is an algorithm that 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 [ ]:
```
from numpy.fft import fft, ifft
import numpy as np
```

Example

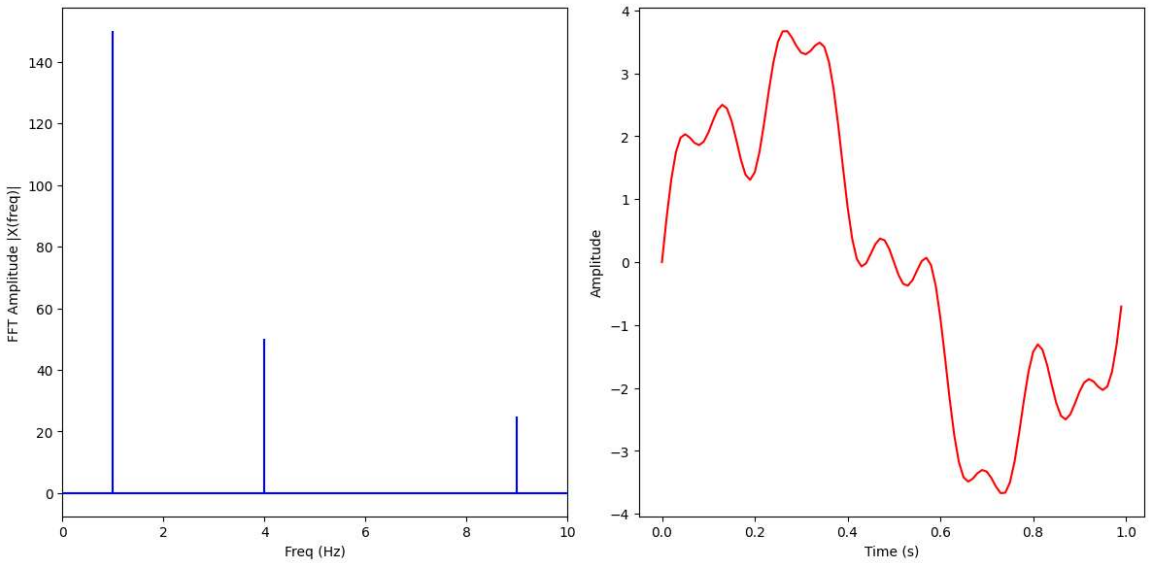
Below is an example of a sequence composed of signals from three different frequencies. We visualize this complicated sequence using the plt.plot function.

In [ ]:
```
# 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 [ ]:
```
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?

From plot 1, it is clear to see that at freq = 1, 4, 9, signals exist. From plot 2, it is hard to see how different sine graphs influencing each other.

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 [ ]:
```
# write your code here
sr = 168
# sampling interval
ts = 1.0/sr
t = np.arange(0,1,ts)
```

```
x = filtered_df.groupby(['weekday','hour'])['count'].mean()
X = fft(x)
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()
```