Data Visualization Part 2 Student Project

Matriculation Number: 12203030

Name: Shivam Goyal

The final project consists of visualization and text exercises. For the visualization exercises the following steps are required:

- Perform all exercises in a Jupyter notebook.
- Add your matriculation number to the Jupyter notebook!
- Write Python code to visualize the given data (see detailed instructions below).
- Comment your code to describe what your code does.
- Recreate the shown example figures using the provided data.
- Export the final notebook to an html and upload it to iLearn.

1: Visualization Exercise 1

Three alternative visualizations of the same artificial data shall be re-created. All three visualizations show the same fictitious genomic annotations together with fictitious RNA binding protein data. The visualizations are an example for RNA binding protein signals as well as the genomic annotations. Recreate each of the shown figures. Two different datasets are provided for this task:

10_project_data_annotations.csv 10_project_data_signals.csv The 10_project_data_annotations.csv file contains fictitious genomic information as visualized in all bottom panels of the example plots. Each horizontal line represents a transcript. A transcript can contain multiple exons (grey rectangles). Transcripts can be located on the '+' or on the '-' strand of the DNA.

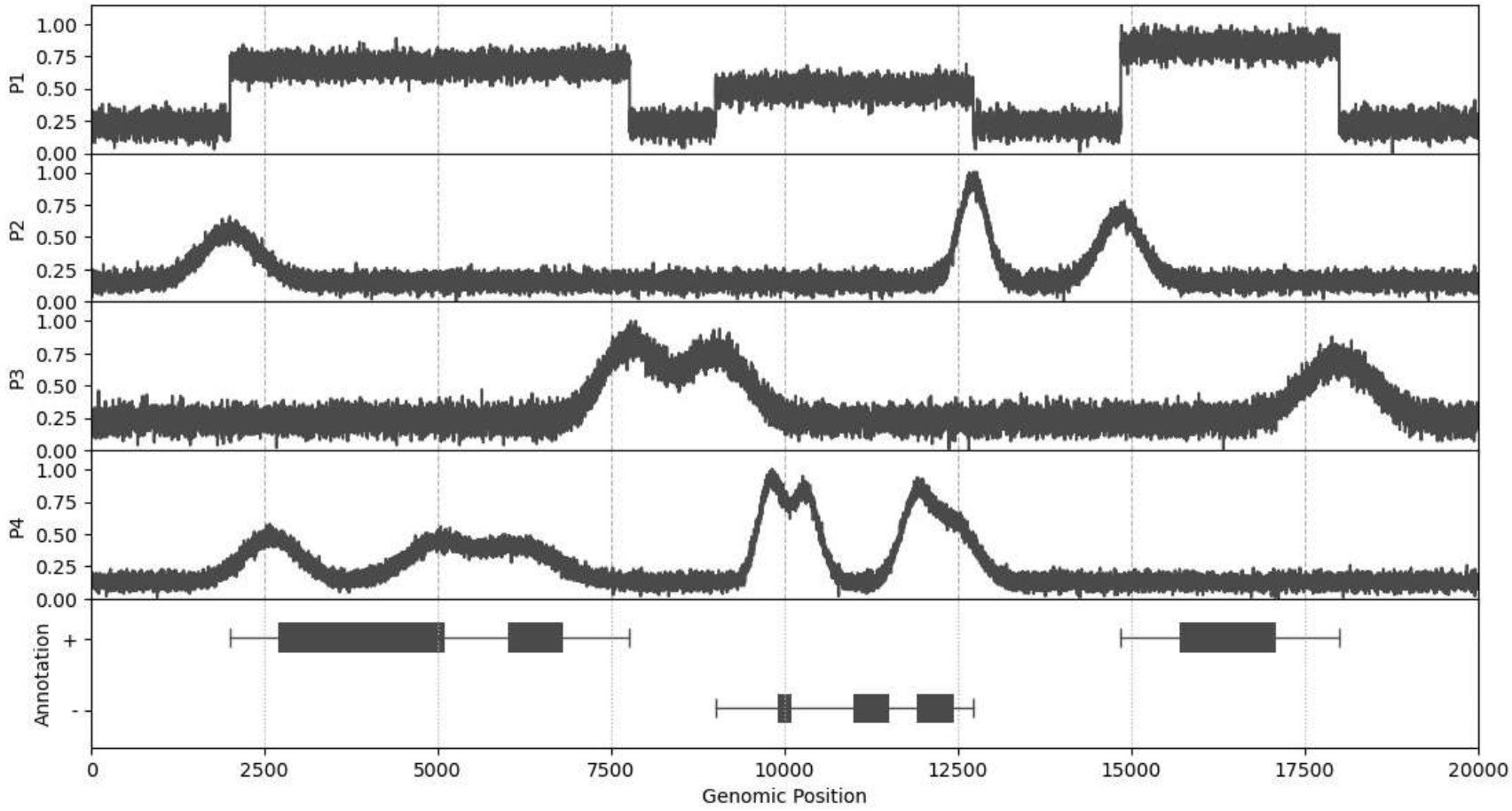
10_project_data_signals.csv contains fictitious signals of four RNA binding proteins (P1, P2, P3, P4).

In [13]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from matplotlib.gridspec import GridSpec
import itertools
```

1.1: Version 1

In [15]:
```
#Import annotation data
Annotation = pd.read_csv('10_project_data_annotation.csv')

#Import signal data
Signal = pd.read_csv('10_project_data_signals.csv')
#set x axis range
x1 = np.arange(0, 20000, 1)

fig, axs = plt.subplots(5, 1, sharex=True)

fig.set_size_inches(13,7)
fig.subplots_adjust(hspace=0)


i = 0

#Plot Protein data in different subplots
for P in Signal.columns:
    axs[i].plot(x1, Signal[[P]], color='#4f4f4f')
    axs[i].set_xlim(0, 20000)
    axs[i].set_yticks((0.0, 0.25, 0.5,0.75,1.0))
    axs[i].set_ylim(0, 1.15)
    axs[i].xaxis.grid(linestyle='--')
    axs[i].set_ylabel(P)
    i += 1


#seperate list for trans
Trans = Annotation[Annotation['type'].str.contains('transcript')]


Annotation['diff'] = Annotation['stop'] - Annotation['start']

#seperate list for Exon
Exon = Annotation[Annotation['type'].str.contains('exon')]


for rows in range(len(Trans)):
    if Trans.iloc[rows,4] == '+':
        y = 0.93
    elif Trans.iloc[rows,4] == '-':
```

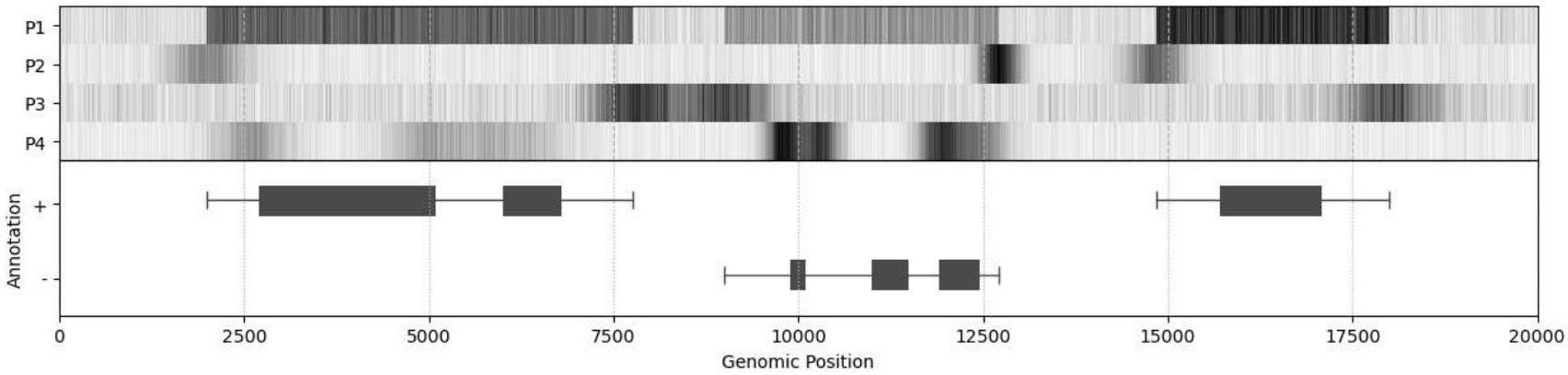
```
        y = 0.33
    g = [(Trans.iloc[rows,2], y), (Trans.iloc[rows,3], y)]

    xs, ys = zip(*g)

    axs[i].plot(xs, ys, '|-', lw=1, color='#4f4f4f', ms=10)

axs[i].set_ylim(0, 1.25)
axs[i].set_xlim(0, 20000)
axs[i].set_xlabel('Genomic Position')
axs[i].set_ylabel('Annotation')
axs[i].set_yticks([0.31, 0.91])
axs[i].set_yticklabels(['-', '+'])
axs[i].xaxis.grid(linestyle=':')

# Exon Plot
for rows in range(len(Exon)):
    if Exon.iloc[rows,4] == '+':
        y = 0.8
        h = 0.25
    elif Exon.iloc[rows,4] == '-':
        y = 0.2
        h = 0.25
    axs[i].broken_barh([(Exon.iloc[rows,2], Exon.iloc[rows,5])], (y, h), facecolors='#4f4f4f')

plt.show()
```



1.2: Version 2

In [8]:
```
fig, axs = plt.subplots(2, 1, sharex=True)
fig.set_size_inches(15,3.15)
fig.subplots_adjust(hspace=0)

#reverse order as per requirement
Signal2 = Signal[['P4', 'P3', 'P2', 'P1']]
#transpose for proper dimensions
Signal2 = np.transpose(Signal2)

x = np.arange(-0.5, 20000, 1)
y = np.arange(-0.5, 4.5, 1)
```

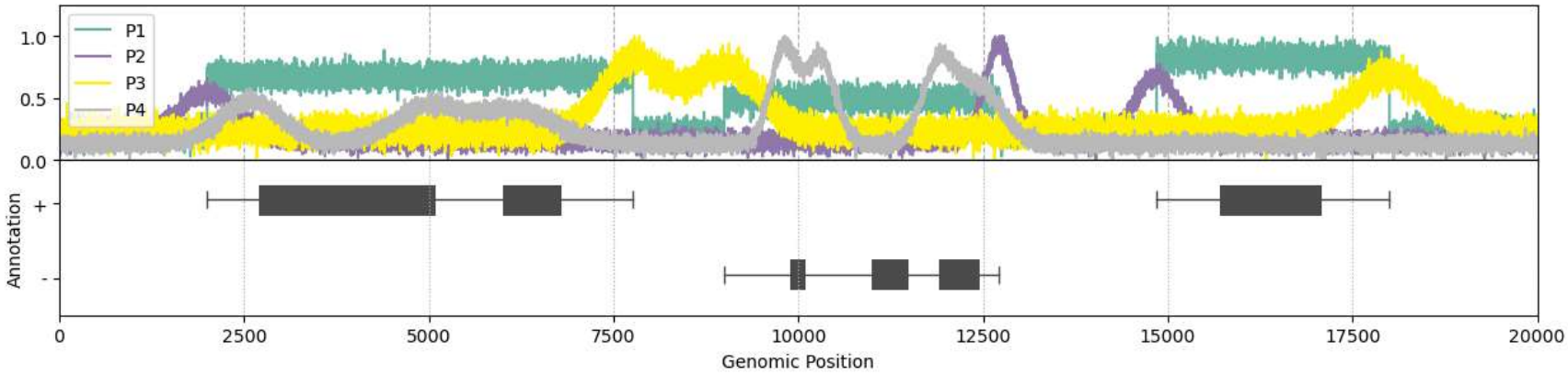
```
# using colormesh
axs[0].pcolormesh(x, y, Signal2, cmap='Greys')
axs[0].set_yticks([0.0, 1.0, 2.0, 3.0])
axs[0].set_yticklabels(['P4', 'P3', 'P2', 'P1'])
axs[0].xaxis.grid(linestyle='--')


for rows in range(len(Trans)):
    if Trans.iloc[rows,4] == '+':
        y = 0.93
    elif Trans.iloc[rows,4] == '-':
        y = 0.33
    g = [(Trans.iloc[rows,2], y), (Trans.iloc[rows,3], y)]
    xs, ys = zip(*g)
    axs[1].plot(xs, ys, '|-', lw=1, color='#4f4f4f', ms=10)

axs[1].set_ylim(0, 1.25)
axs[1].set_xlim(0, 20000)
axs[1].set_xlabel('Genomic Position')
axs[1].set_ylabel('Annotation')
axs[1].set_yticks([0.3, 0.9])
axs[1].set_yticklabels(['-', '+'])
axs[1].xaxis.grid(linestyle=':')

for rows in range(len(Exon)):
    if Exon.iloc[rows,4] == '+':
        y = 0.8
        h = 0.25
    elif Exon.iloc[rows,4] == '-':
        y = 0.2
        h = 0.25
    axs[1].broken_barh([(Exon.iloc[rows,2], Exon.iloc[rows,5])], (y, h), facecolors='#4f4f4f')

plt.show()
```



1.3: Version 3

In [9]:
```
t = np.arange(0, 20000, 1)

fig, axs = plt.subplots(2, 1, sharex=True)
fig.set_size_inches(15,3.15)
fig.subplots_adjust(hspace=0)

#plotting protiens in a single plot
axs[0].plot(x1,Signal["P1"], label = "P1", color = "#65b7a4")
axs[0].plot(x1,Signal["P2"], label = "P2", color = "#947cab")
axs[0].plot(x1,Signal["P3"], label = "P3", color = "#fff200")
axs[0].plot(x1,Signal["P4"], label = "P4", color = "#bdbdbd")
axs[0].set_yticks(np.arange(0.0, 1.25, 0.5))
axs[0].set_ylim(0, 1.25)
axs[0].xaxis.grid(linestyle='--')
axs[0].legend(loc='upper left', ncol=1)

for rows in range(len(Trans)):
    if Trans.iloc[rows,4] == '+':
        y = 0.93
    elif Trans.iloc[rows,4] == '-':
        y = 0.33
    g = [(Trans.iloc[rows,2], y), (Trans.iloc[rows,3], y)]
    xs, ys = zip(*g)
    axs[1].plot(xs, ys, '|-', lw=1, color='#4f4f4f', ms=10)

axs[1].set_ylim(0, 1.25)
axs[1].set_xlim(0, 20000)
```

```
axs[1].set_xlabel('Genomic Position')
axs[1].set_ylabel('Annotation')
axs[1].set_yticks([0.3, 0.9])
axs[1].set_yticklabels(['-', '+'])
axs[1].xaxis.grid(linestyle=':')

for rows in range(len(Exon)):
    if Exon.iloc[rows,4] == '+':
        y = 0.8
        h = 0.25
    elif Exon.iloc[rows,4] == '-':
        y = 0.2
        h = 0.25
    axs[1].broken_barh([(Exon.iloc[rows,2], Exon.iloc[rows,5])], (y, h), facecolors='#4f4f4f')

plt.show()
```



1.4: Discussion

Discuss the pros and cons between the different visualization approaches.

There are many different visualizing techniques which offers different approaches to visualizations. They excel in some ways and lacks in others. They should be selected based on their pros and cons. Below are discussed the first three approaches used in this assignment

Version 1 Pro:

1) As a seperate plot for each protien is created, it is easy to identify different trends and patterns. 2) Plotting the protien graphs as subplots helps in visualizing all of them at the same time

cons: 1) This visualization technique is not viable with high number of protiens 2) Plotting of graphs in layers makes comparitive analysis difficult

Version 2 Pro:

1) It is better for visualizing high and low intensities of signals 2) It helps us identify regions of interests within the protiens data

cons: 1) It is not possible to track individual values

Version 3 Pro:

1) Easy to visualize all the data in a single plot 2) Better for comparative study

cons: 1) Graphs becomes unclear and difficult to track at some points,and it will increase with higher number of protiens

2: Visualization Exercises

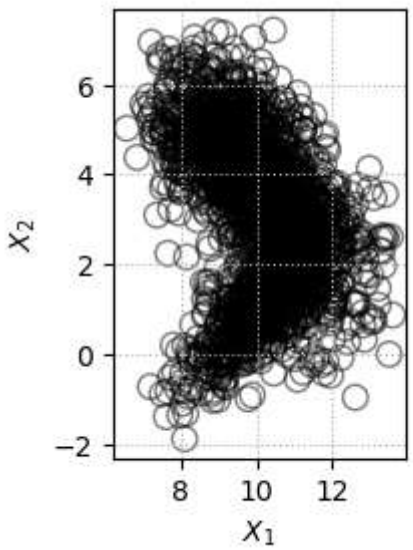
In this task, two additional plots shall be added create a figure with multiple panels. Recreate the shown figure. Note, that the bottom part of the figure is one of your solutions from the first exercise. Two additional datasets are provided:

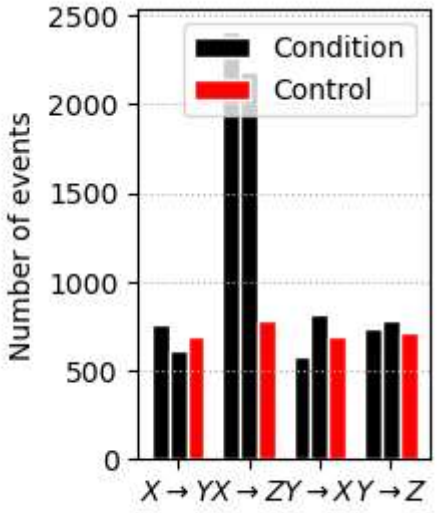
10_project_data_scatter.csv contains the data needed to create the shown scatter plot 10_project_data_barplot.csv contains the data needed to create the shown bar plot.

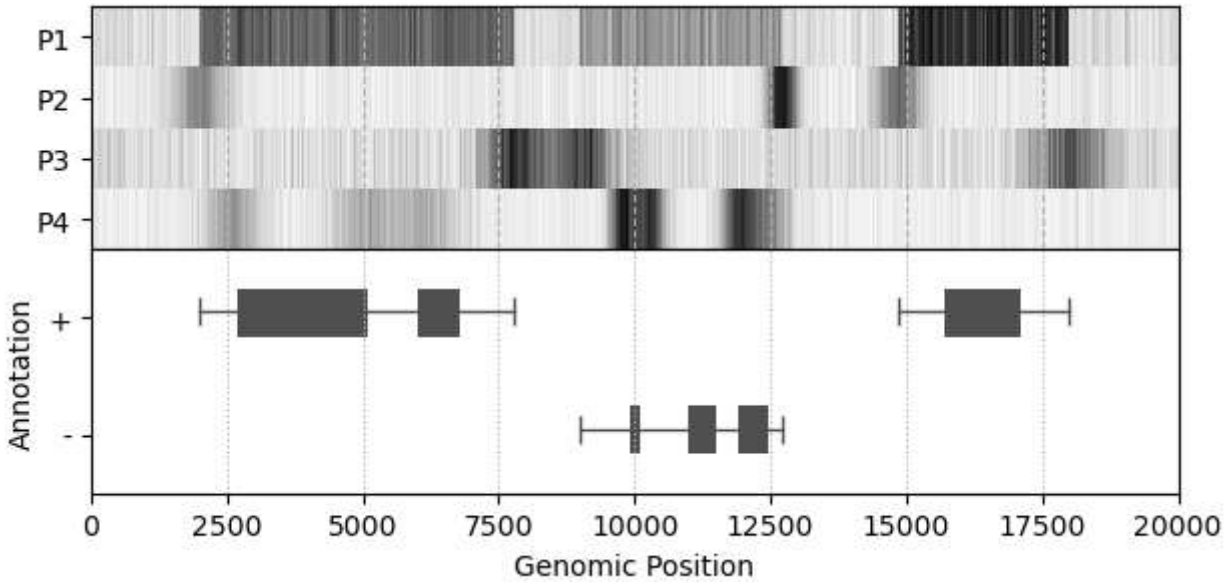
In [11]:
```
#Creating grid space to create a figure with multiple panels.
gs = GridSpec(2, 3,height_ratios=[2, 1])

fig1, axs1 = plt.subplots(1, 2)
#Scatter plot data
SPData = pd.read_csv('10_project_data_scatter.csv')
#scatter plot in 1st row and 1st column
ax1=plt.subplot(gs[0,0])
x1 = SPData['x1']
x2 = SPData['x2']
plt.xticks(np.arange(8, 12+1, 2))
plt.xlabel('$X_{1}$')
plt.ylabel('$X_{2}$')
```

```
plt.scatter(x1, x2, s=80, alpha=0.5, facecolors='none', edgecolors='black')
plt.grid(linestyle=':')

# Bar graph in  1st row and 2nd column
ax2=plt.subplot(gs[0,2])


BPData = pd.read_csv('10_project_data_barplot.csv')

barWidth = 0.25

labels = [r'$X{\rightarrow}Y$', r'$X{\rightarrow}Z$', r'$Y{\rightarrow}X$', r'$Y{\rightarrow}Z$']
b1 = BPData['condition_a_sample_1']
b2 = BPData['condition_a_sample_2']
b3 = BPData['control']

r1 = np.arange(len(b1))
r2 = [x + barWidth for x in r1]
r3 = [x + barWidth for x in r2]


plt.bar(r1, b1, color='black', width=barWidth, edgecolor='white', label='Condition')
plt.bar(r2, b2, color='black', width=barWidth, edgecolor='white')
plt.bar(r3, b3, color='red', width=barWidth, edgecolor='white', label='Control')

plt.ylabel('Number of events')
plt.xticks(r1+barWidth, labels)
plt.grid(axis='y',linestyle=':')

plt.legend()

# 1.2 version fraph in second row and all columns
ax3=plt.subplot(gs[1,:])


fig, axs = plt.subplots(2, 1, sharex=True)
ax3.remove()
fig.set_size_inches(7,3.15)
fig.subplots_adjust(hspace=0)


Signal2 = Signal[['P4', 'P3', 'P2', 'P1']]

Signal2 = np.transpose(Signal2)
```

```
x = np.arange(-0.5, 20000, 1)
y = np.arange(-0.5, 4.5, 1)

axs[0].pcolormesh(x, y, Signal2, cmap='Greys')
axs[0].set_yticks([0.0, 1.0, 2.0, 3.0])
axs[0].set_yticklabels(['P4', 'P3', 'P2', 'P1'])
axs[0].xaxis.grid(linestyle='--')


for rows in range(len(Trans)):
    if Trans.iloc[rows,4] == '+':
        y = 0.93
    elif Trans.iloc[rows,4] == '-':
        y = 0.33
    g = [(Trans.iloc[rows,2], y), (Trans.iloc[rows,3], y)]
    xs, ys = zip(*g)
    axs[1].plot(xs, ys, '|-', lw=1, color='#4f4f4f4f', ms=10)

axs[1].set_ylim(0, 1.25)
axs[1].set_xlim(0, 20000)
axs[1].set_xlabel('Genomic Position')
axs[1].set_ylabel('Annotation')
axs[1].set_yticks([0.3, 0.9])
axs[1].set_yticklabels(['-', '+'])
axs[1].xaxis.grid(linestyle=':')

for rows in range(len(Exon)):
    if Exon.iloc[rows,4] == '+':
        y = 0.8
        h = 0.25
    elif Exon.iloc[rows,4] == '-':
        y = 0.2
        h = 0.25
    axs[1].broken_barh([(Exon.iloc[rows,2], Exon.iloc[rows,5])], (y, h), facecolors='#4f4f4f4f')

plt.show()
```







3: K-mer Counts In Sequence Data

The file 10_project_data_dna_sequences.txt consists of DNA sequences. Each line is a single DNA sequence. The figure shows that some k-mers are concentrated (occur more often) in some regions of the sequences. To recreate the figure you need to count the occurrences of a kmer at each position for all sequences.

Example: imagine to search for "xx" and "yy" in following sequences.

TCTAAGGxxyyG xxCGTGGyyCGT TTTxxACCGyyG AGCxxACAACyy TAxxCAAyyAGT The counts for "xx" and "yy" would be the following:

[1 0 1 2 0 0 0 1 0 0 0 0] [0 0 0 0 0 0 0 2 0 2 1 0] Recreate the shown figure be providing code to find the 2-mers in the given dataset as well as the code to display the 2-mer occurences per position.

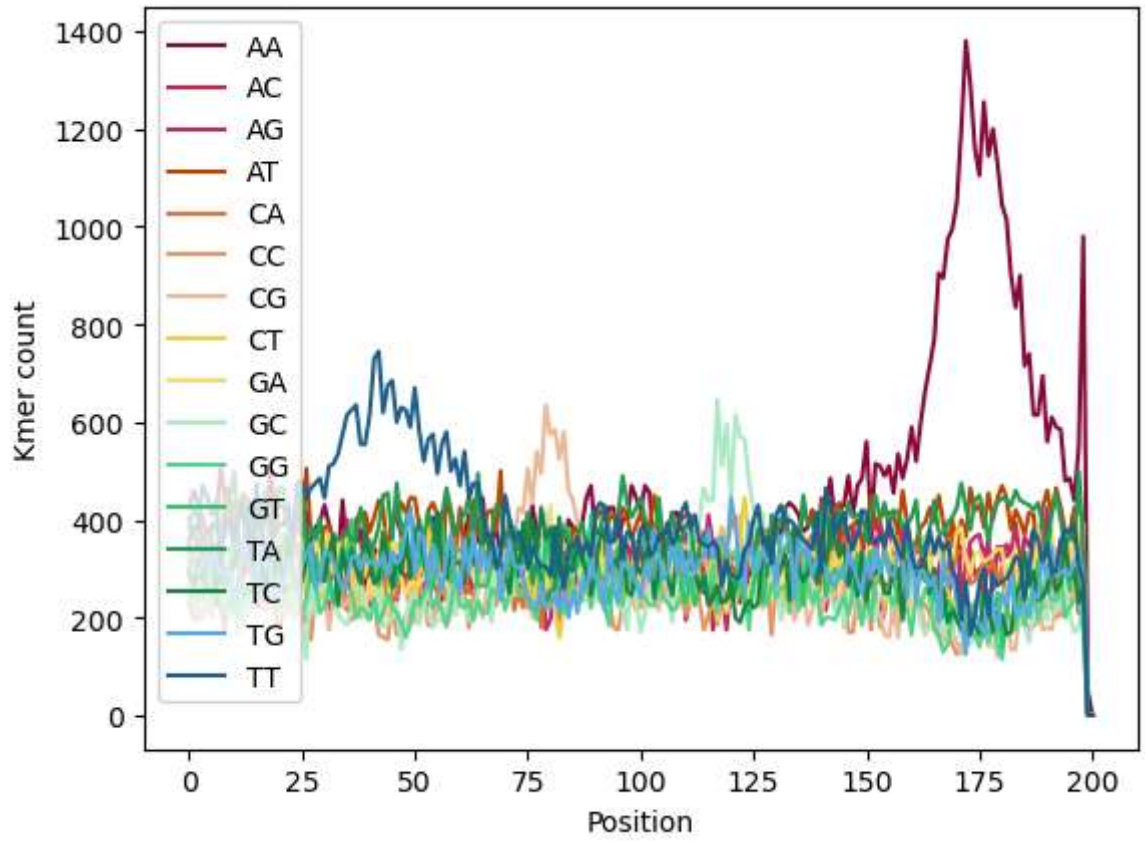
In [22]:
```
kmer=[]
x=0
with open('10_project_data_dna_sequences.txt','r') as f:
    lines = f.readlines()
    for line in lines:
        x=max(x,len(line))
        for j in range(len(line)-2):
            temp=line[j:j+2]
            if temp in kmer:
                continue
            else:
                kmer.append(temp)
    f.close()
iter = itertools.repeat(0, x)


for i in kmer:
    i=list(iter)


with open('10_project_data_dna_sequences.txt','r') as f:
    lines = f.readlines()
    #Iterating through every index of the line
    for line in lines:
        for j in range(len(line)-1):
#Appending zeroes. List index corresponds to position to the index on the line. Making the lenth of the lists equivalent


#comparing at every index of the line it matches the required pattern
            if line.find("AA",j,j+2)!=-1:
                AA[j]+=1
            if line.find("AC",j,j+2)!=-1:
                AC[j]+=1
            if line.find("AG",j,j+2)!=-1:
                AG[j]+=1
            if line.find("AT",j,j+2)!=-1:
```

```
                AT[j]+=1
            if line.find("CA",j,j+2)!=-1:
                CA[j]+=1
            if line.find("CC",j,j+2)!=-1:
                CC[j]+=1
            if line.find("CG",j,j+2)!=-1:
                CG[j]+=1
            if line.find("CT",j,j+2)!=-1:
                CT[j]+=1
            if line.find("GA",j,j+2)!=-1:
                GA[j]+=1
            if line.find("GC",j,j+2)!=-1:
                GC[j]+=1
            if line.find("GG",j,j+2)!=-1:
                GG[j]+=1
            if line.find("GT",j,j+2)!=-1:
                GT[j]+=1
            if line.find("TA",j,j+2)!=-1:
                TA[j]+=1
            if line.find("TC",j,j+2)!=-1:
                TC[j]+=1
            if line.find("TG",j,j+2)!=-1:
                TG[j]+=1
            if line.find("TT",j,j+2)!=-1:
                TT[j]+=1
fig, axs = plt.subplots(1, 1, sharex=True)

AAxaxis=[]
AAyaxis=[]

ACxaxis=[]
ACyaxis=[]

AGxaxis=[]
AGyaxis=[]

ATxaxis=[]
ATyaxis=[]

CAxaxis=[]
CAyaxis=[]

CCxaxis=[]
CCyaxis=[]
```

```
CGxaxis=[]
CGyaxis=[]

CTxaxis=[]
CTyaxis=[]

GAxaxis=[]
GAyaxis=[]

GCxaxis=[]
GCyaxis=[]

GGxaxis=[]
GGyaxis=[]

GTxaxis=[]
GTyaxis=[]

TAxaxis=[]
TAyaxis=[]

TCxaxis=[]
TCyaxis=[]

TGxaxis=[]
TGyaxis=[]

TTxaxis=[]
TTyaxis=[]

#Making the list of x axis and y axis for every k mer
for i in range(len(AA)):
    AAxaxis.append(i)
    AAyaxis.append(AA[i])

for i in range(len(AC)):
    ACxaxis.append(i)
    ACyaxis.append(AC[i])

for i in range(len(AG)):
    AGxaxis.append(i)
    AGyaxis.append(AG[i])

for i in range(len(AT)):
    ATxaxis.append(i)
```

```
    ATyaxis.append(AT[i])

for i in range(len(CA)):
    CAxaxis.append(i)
    CAyaxis.append(CA[i])

for i in range(len(CC)):
    CCxaxis.append(i)
    CCyaxis.append(CC[i])

for i in range(len(CG)):
    CGxaxis.append(i)
    CGyaxis.append(CG[i])

for i in range(len(CT)):
    CTxaxis.append(i)
    CTyaxis.append(CT[i])

for i in range(len(GA)):
    GAxaxis.append(i)
    GAyaxis.append(GA[i])

for i in range(len(GC)):
    GCxaxis.append(i)
    GCyaxis.append(GC[i])

for i in range(len(GG)):
    GGxaxis.append(i)
    GGyaxis.append(GG[i])

for i in range(len(GT)):
    GTxaxis.append(i)
    GTyaxis.append(GT[i])

for i in range(len(TA)):
    TAxaxis.append(i)
    TAyaxis.append(TA[i])

for i in range(len(TC)):
    TCxaxis.append(i)
    TCyaxis.append(TC[i])

for i in range(len(TG)):
    TGxaxis.append(i)
    TGyaxis.append(TG[i])
```

```
for i in range(len(TT)):
    TTxaxis.append(i)
    TTyaxis.append(TT[i])


axs.plot(AAxaxis,AAyaxis, label = "AA", color = "#8a0c40")
axs.plot(ACxaxis,ACyaxis, label = "AC", color = "#C12E52")
axs.plot(AGxaxis,AGyaxis, label = "AG", color = "#C12E6A")
axs.plot(ATxaxis,ATyaxis, label = "AT", color = "#BA4A00")
axs.plot(CAxaxis,CAyaxis, label = "CA", color = "#DC7633")
axs.plot(CCxaxis,CCyaxis, label = "CC", color = "#E59866")
axs.plot(CGxaxis,CGyaxis, label = "CG", color = "#EDBB99")
axs.plot(CTxaxis,CTyaxis, label = "CT", color = "#F4D03F")
axs.plot(GAxaxis,GAyaxis, label = "GA", color = "#F7DC6F")
axs.plot(GCxaxis,GCyaxis, label = "GC", color = "#ABEBC6")
axs.plot(GGxaxis,GGyaxis, label = "GG", color = "#58D68D")
axs.plot(GTxaxis,GTyaxis, label = "GT", color = "#2ECC71")
axs.plot(TAxaxis,TAyaxis, label = "TA", color = "#239B56")
axs.plot(TCxaxis,TCyaxis, label = "TC", color = "#1D8348")
axs.plot(TGxaxis,TGyaxis, label = "TG", color = "#5DADE2")
axs.plot(TTxaxis,TTyaxis, label = "TT", color = "#1F618D")



plt.legend(loc='upper left')
axs.set_xlabel('Position')
axs.set_ylabel('Kmer count')
plt.show()
```


AA
AC
AG
AT
CA
CC
CG
CT
GA
GC
GG
GT
TA
TC
TG
TT
Kmer count
Position
0
200
400
600
800
1000
1200
1400
0
25
50
75
100
125
150
175
200

In [ ]: