

```
!pip install pyreadr
import pandas as pd
import numpy as np
import pyreadr

result = pyreadr.read_r('/content/exgr_test.rds')
exgr_test = result[None]
```

▼ Task 1: How many unique transcripts are there?

```
#transcript_num = len(exgr_test['transcript_id'].unique())
transcript_num = len(exgr_test['transcript_name'].unique())
print("The value of the unique transcript is: " + str(transcript_num))
```

The value of the unique transcript is: 234485

▼ Task 2: How many unique exons are there?

```
len(exgr_test['exon_id'].unique())
exon_count = len(exgr_test['exon_name'].unique())
print("The number of unique exon is:" + str(exon_count))
```

The number of unique exon is:760040

Here, number of unique exons' ID is 509 larger than the number of unique exons' name. This is because the same exons (same name) are labeled as different ID in ChrX and ChrY.

```
new_table = exgr_test[["exon_id", "exon_name"]].copy()
shortened_table = new_table.drop_duplicates(subset="exon_id", keep="first")
s2 = new_table.drop_duplicates(subset="exon_name", keep="first")

difference = pd.merge(shortened_table, s2, on="exon_id", how="left", indicator=True)
difference = difference[difference['_merge'] == 'left_only']
difference = difference.drop(columns='_merge')
```

▼ Task 3: what is the average length of an exon? What is the median length?

```

exon_avg_length = np.mean(exgr_test['width'])
exon_med_length = np.median(exgr_test['width'])
print("The average length of an exon is:" + str(exon_avg_length))
print("The median length is:" + str(exon_med_length ))

```

```

The average length of an exon is:262.9930635885628
The median length is:130.0

```

Task 4: Find the length of the introns between the exons. (length must be a positive number)

```

import time

# Start the timer
start_time = time.time()
# Filter rows based on condition for strand '+'
data1 = exgr_test[exgr_test['strand'] == '+']

# Calculate length column using vectorized operations
a = data1['start'].values[0:]
b = data1['end'].values[:-1]
b = np.concatenate(([0],b))
c = a - b -1
data1['length'] = c

# Filter rows where rank is not equal to 1
data1_filtered = data1[data1['rank'].ne(1)]
data1_filtered['introns_rank'] = data1_filtered['rank'] -1
introns_plus = data1_filtered[['transcript_id', 'transcript_name', 'strand', 'introns_rank', 'le

# Filter rows based on condition for strand '-'
data2 = exgr_test[exgr_test['strand'] == '-']

# Calculate length column using vectorized operations
x1 = data2['start'].values[:-1]
# add a not important number so that two column reduced by data1$start[i] - data1$end[i-1]
x2 = np.concatenate(([0],x1))
x3 = data2['end'].values[0:]
x4 = x2 - x3 -1
data2['length'] = x4

# when rank equal 1, there should be no intron before that one.
data2_filtered = data2[data2['rank'].ne(1)]
data2_filtered['introns_rank'] = data2_filtered['rank'] -1
introns_minus = data2_filtered[['transcript_id', 'transcript_name', 'strand', 'introns_rank', 'l

```

```
# combine the table of negative and positive strands
introns_table = combined_df = pd.concat([introns_plus, introns_minus])
```

```
# End the timer
end_time = time.time()
```

```
# Compute the elapsed time
elapsed_time = end_time - start_time
```

```
print(f"Elapsed time: {elapsed_time} seconds")
introns_table
#introns_table.to_csv('data.txt', sep='\t', index=False)
```

Elapsed time: 0.5012466907501221 seconds

	transcript_id	transcript_name	strand	introns_rank	length
1	1	ENST00000456328.2	+	1	385
2	1	ENST00000456328.2	+	2	499
4	2	ENST00000450305.2	+	1	121
5	2	ENST00000450305.2	+	2	385
6	2	ENST00000450305.2	+	3	277
...
1460943	234447	ENST00000399966.9_PAR_Y	-	1	1161
1460945	234448	ENST00000507418.6_PAR_Y	-	1	385
1460946	234448	ENST00000507418.6_PAR_Y	-	2	277
1460947	234448	ENST00000507418.6_PAR_Y	-	3	168
1460948	234448	ENST00000507418.6_PAR_Y	-	4	78

1226501 rows × 5 columns

▼ Bonus question

```
import math
```

```
# Separate data by the +/- strand
data1 = exgr_test[exgr_test['strand'] == '+']
data2 = exgr_test[exgr_test['strand'] == '-']
```

```

# Positive strand
record1 = {
    'transcript_id': [],
    'transcript_name': [],
    'exon': [],
    'start': [],
    'end': [],
    'width': [],
    'strand': [],
    'l1': [],
    'l2': [],
    'u1': [],
    'u2': []
}

# Loop over the positive list, leave the last case to deal individually
for i in range(len(data1) - 1):
    # Count the l1 region by case: at rank 1 or not
    if data1['rank'].iloc[i] == 1:
        l1 = 0
    else:
        last_intro = data1['start'].iloc[i] - data1['end'].iloc[i - 1] - 1
        if last_intro < 200:
            l1 = data1['start'].iloc[i] - math.ceil(last_intro / 2)
        else:
            l1 = data1['start'].iloc[i] - 100

    # Count the l2 and u1 region, also see if the region is larger than 200 bp
    if data1['width'].iloc[i] < 200:
        count = math.ceil(data1['width'].iloc[i] / 2)
        l2 = data1['start'].iloc[i] + count
        u1 = l2 + 1
    else:
        l2 = data1['start'].iloc[i] + 100
        u1 = data1['end'].iloc[i] - 100

    # Count the u2 regions by case: if it's the last tank inside its transcript
    if data1['rank'].iloc[i + 1] == 1:
        u2 = 0
    else:
        next_intro = data1['start'].iloc[i + 1] - data1['end'].iloc[i] - 1
        if next_intro < 200:
            u2 = data1['start'].iloc[i + 1] - math.ceil(next_intro / 2) - 1
        else:
            u2 = data1['end'].iloc[i] + 100

    # Record l1, l2, u1, u2 regions and their positions
    record1['transcript_id'].append(data1['transcript_id'].iloc[i])
    record1['transcript_name'].append(data1['transcript_name'].iloc[i])
    record1['exon'].append(data1['exon_name'].iloc[i])
    record1['start'].append(data1['start'].iloc[i])

```

```

record1['end'].append(data1['end'].iloc[i])
record1['width'].append(data1['width'].iloc[i])
record1['strand'].append("+")
record1['l1'].append(l1)
record1['l2'].append(l2)
record1['u1'].append(u1)
record1['u2'].append(u2)

# Edge case (last row of the data)
# Count the l2 and u2 regions at the edge case
if data1['width'].iloc[-1] < 200:
    count = math.ceil(data1['width'].iloc[-1] / 2)
    l2 = data1['start'].iloc[-1] + count
    u1 = l2 + 1
else:
    l2 = data1['start'].iloc[-1] + 100
    u1 = data1['end'].iloc[-1] - 100

# Count the l1 and u2 regions at the edge case
if data1['rank'].iloc[-1] == 1: # The edge case is an individual transcript
    l1 = 0
    u2 = 0
else: # The edge case isn't a single transcript
    last_intro = data1['start'].iloc[-1] - data1['end'].iloc[-2] - 1
    if last_intro < 200:
        l1 = data1['start'].iloc[-1] - math.ceil(last_intro / 2)
    else:
        l1 = data1['start'].iloc[-1] - 100
    u2 = 0

# Record l1, l2, u1, u2 regions at the edge case
record1['transcript_id'].append(data1['transcript_id'].iloc[-1])
record1['transcript_name'].append(data1['transcript_name'].iloc[-1])
record1['exon'].append(data1['exon_name'].iloc[-1])
record1['start'].append(data1['start'].iloc[-1])
record1['end'].append(data1['end'].iloc[-1])
record1['width'].append(data1['width'].iloc[-1])
record1['strand'].append("+")
record1['l1'].append(l1)
record1['l2'].append(l2)
record1['u1'].append(u1)
record1['u2'].append(u2)

record2 = {
    'transcript_id': [],
    'transcript_name': [],
    'exon': [],
    'start': [],
    'end': [],
    'width': [],

```

```

'strand': [],
'l1': [],
'l2': [],
'u1': [],
'u2': []
}

```

```

# Loop over the positive list, leave the last case to deal individually
for i in range(len(data2) - 1):
    # Count the l1 region by case: at rank 1 or not
    if data2['rank'].iloc[i] == 1:
        u2 = 0
    else:
        last_intro = data2['start'].iloc[i - 1] - data2['end'].iloc[i] - 1
        if last_intro < 200:
            u2 = data2['end'].iloc[i] + math.ceil(last_intro / 2)
        else:
            u2 = data2['end'].iloc[i] + 100

    # Count the l2 and u1 region, also see if the region is larger than 200 bp
    if data2['width'].iloc[i] < 200:
        count = math.ceil(data2['width'].iloc[i] / 2)
        l2 = data2['start'].iloc[i] + count
        u1 = l2 + 1
    else:
        u1 = data2['start'].iloc[i] + 100
        l2 = data2['end'].iloc[i] - 100

    # Count the u2 regions by case: if it's the last tank inside its transcript
    if data2['rank'].iloc[i + 1] == 1:
        l1 = 0
    else:
        next_intro = data2['start'].iloc[i] - data2['end'].iloc[i + 1] - 1
        if next_intro < 200:
            l1 = data2['end'].iloc[i + 1] + math.ceil(next_intro / 2) + 1
        else:
            l1 = data2['start'].iloc[i] - 100

    # Record l1, l2, u1, u2 regions and their positions
    record2['transcript_id'].append(data2['transcript_id'].iloc[i])
    record2['transcript_name'].append(data2['transcript_name'].iloc[i])
    record2['exon'].append(data2['exon_name'].iloc[i])
    record2['start'].append(data2['start'].iloc[i])
    record2['end'].append(data2['end'].iloc[i])
    record2['width'].append(data2['width'].iloc[i])
    record2['strand'].append("-")
    record2['l1'].append(l1)
    record2['l2'].append(l2)
    record2['u1'].append(u1)
    record2['u2'].append(u2)

```

```

# Edge case (last row of the data)
# Count the l2 and u2 regions at the edge case
if data2['width'].iloc[-1] < 200:
    count = math.ceil(data2['width'].iloc[-1] / 2)
    l2 = data2['start'].iloc[-1] + count
    u1 = l2 + 1
else:
    u1 = data2['start'].iloc[-1] + 100
    l2 = data2['end'].iloc[-1] - 100

# Count the l1 and u2 regions at the edge case
if data2['rank'].iloc[-1] == 1: # The edge case is an individual transcript
    l1 = 0
    u2 = 0
else: # The edge case isn't a single transcript
    last_intro = data2['start'].iloc[-2] - data2['end'].iloc[-1] - 1
    if last_intro < 200:
        u2 = data2['end'].iloc[-1] + math.ceil(last_intro / 2)
    else:
        u2 = data2['end'].iloc[-1] + 100
    l1 = 0

# Record l1, l2, u1, u2 regions at the edge case
record2['transcript_id'].append(data2['transcript_id'].iloc[-1])
record2['transcript_name'].append(data2['transcript_name'].iloc[-1])
record2['exon'].append(data2['exon_name'].iloc[-1])
record2['start'].append(data2['start'].iloc[-1])
record2['end'].append(data2['end'].iloc[-1])
record2['width'].append(data2['width'].iloc[-1])
record2['strand'].append("-")
record2['l1'].append(l1)
record2['l2'].append(l2)
record2['u1'].append(u1)
record2['u2'].append(u2)



region = pd.DataFrame.from_dict(record1)
region2 = pd.DataFrame.from_dict(record2)
combined_table = pd.concat([region, region2])

# Filter out rows with missing values
Bouns_table = combined_table.dropna().reset_index(drop=True)

Bouns_table
#Bouns_table.to_csv('Bouns_table.txt', sep='\t', index=False)

```



t_name	exon	start	end	width	strand	l1	l2	u1	u2	
56328.2	ENSE00002234944.1	11869	12227	359	+	0	11969	12127	12327	
56328.2	ENSE00003582793.1	12613	12721	109	+	12513	12668	12669	12821	
56328.2	ENSE00002312635.1	13221	14409	1189	+	13121	13321	14309	0	
50305.2	ENSE00001948541.1	12010	12057	48	+	0	12034	12035	12117	
50305.2	ENSE00001671638.2	12179	12227	49	+	12118	12204	12205	12327	
...	
37409.1	ENSE00001544488.1	5826	5891	66	-	0	5859	5860	0	
37416.2	ENSE00001544487.2	7446	7514	69	-	0	7481	7482	0	
37416.2	ENSE00001434974.2	14149	14673	525	-	0	14573	14249	0	
37459.1	ENSE00001544476.1	14674	14742	69	-	0	14709	14710	0	
37461.2	ENSE00001544473.2	15956	16023	68	-	0	15990	15991	0	

