# Main

November 27, 2025

## 1 Data Import and Analysis

### 1.1 Import Python libraries

This cell loads the Python libraries that we use for the analysis:

- `pandas` and `numpy` for handling tables of gene expression values.
- `matplotlib` and `seaborn` for making plots.
- `os` and `re` for handling file paths and regular expressions (used to parse sample names).
- `sklearn` tools for scaling data and clustering (used later if we find time-dependent genes).

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import os
import re
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering

# Set plot style
sns.set_theme(style="whitegrid")
```

This cell reads the main expression file `GSE196728_Peru_raw.xlsx` from the `Data/` folder and stores it in a table called `df`.

- Rows represent genes.
- Columns represent samples collected from different people, at different altitudes and times.
- The column `Unnamed: 0` contains gene names (symbols).
- The other columns have names like `sub100_SL22.raw`, which encode subject ID, altitude and time of sampling.

The `display(df.head())` and `df.shape` lines show the first few rows and the overall size of the dataset, to confirm that the file loaded correctly and to understand its structure.

[2]:
```
excel_path = os.path.join('Data', 'GSE196728_Peru_raw.xlsx')
try:
    df = pd.read_excel(excel_path)
    print("Excel file loaded successfully.")
    display(df.head())
```

```
    df.shape
except Exception as e:
    print(f"Error loading Excel file: {e}")
```

```
Excel file loaded successfully.
```

| | Unnamed: 0 | sub100_SL22.raw | sub101_SL18.raw | sub102_SL18.raw | \ |
|---|---|---|---|---|---|
| 0 | A1BG-AS1 | 2 | 2 | 2 | |
| 1 | A2M-AS1 | 10 | 2 | 4 | |
| 2 | A4GALT | 0 | 0 | 0 | |
| 3 | AAAS | 23 | 7 | 14 | |
| 4 | AACS | 12 | 7 | 11 | |

| | sub103_SL10.raw | sub104_SL2.raw | sub10_SL22.raw | sub11_SL14.raw | \ |
|---|---|---|---|---|---|
| 0 | 2 | 3 | 5 | 13 | |
| 1 | 1 | 2 | 1 | 4 | |
| 2 | 0 | 0 | 0 | 0 | |
| 3 | 15 | 20 | 47 | 33 | |
| 4 | 7 | 7 | 8 | 23 | |

| | sub12_SL2.raw | sub13_SL18.raw | … | sub67_RI2.raw | sub68_RI6.raw | \ |
|---|---|---|---|---|---|---|
| 0 | 15 | 14 | … | 7 | 11 | |
| 1 | 17 | 9 | … | 8 | 5 | |
| 2 | 0 | 1 | … | 0 | 7 | |
| 3 | 85 | 47 | … | 37 | 32 | |
| 4 | 37 | 25 | … | 25 | 27 | |

| | sub69_RI2.raw | sub70_RI18.raw | sub7_RI14.raw | sub83_RI2.raw | sub85_RI2.raw | \ |
|---|---|---|---|---|---|---|
| 0 | 8 | 4 | 7 | 8 | 4 | |
| 1 | 6 | 4 | 5 | 9 | 8 | |
| 2 | 2 | 3 | 0 | 4 | 0 | |
| 3 | 43 | 21 | 87 | 48 | 30 | |
| 4 | 21 | 13 | 23 | 17 | 14 | |

| | sub8_RI6.raw | sub98_RI14.raw | sub99_RI22.raw |
|---|---|---|---|
| 0 | 8 | 4 | 4 |
| 1 | 12 | 2 | 16 |
| 2 | 0 | 0 | 0 |
| 3 | 78 | 12 | 35 |
| 4 | 24 | 10 | 19 |

```
[5 rows x 145 columns]
```

This cell performs two key preparation steps:

1. **Select only sea-level (SL) samples**
    - It uses a regular expression to identify columns whose names match the pattern `sub<subject>_SL<time>.raw`.
    - Only those columns are kept, together with the gene name column.

- This removes all samples from higher altitudes (Puno and La Rinconada), so that the analysis here focuses on baseline sea-level conditions.

2. **Rename columns to human-readable labels**

   - Column names like `sub100_SL22.raw` are renamed to `Sub100_22`.
   - In this format:
     - `100` is the subject identifier.
     - `22` is the sampling time (22:00).

After this cell, `df` contains:

- 12,726 genes.
- 48 sea-level samples, each labeled as `Sub<subject>_<time>`.

This gives us a clean sea-level dataset for time-of-day analysis, without mixing in altitude effects.

[3]:
```
pattern = re.compile(r'^sub(?P<subject>\d+)_SL(?P<time>\d+)\.raw$', re.
 ↪IGNORECASE)
sample_columns = []
rename_map = {}
for col in df.columns:
    match = pattern.fullmatch(str(col))
    if match:
        sample_columns.append(col)
        subject = match.group('subject')
        timepoint = match.group('time')
        rename_map[col] = f"Sub{subject}_{timepoint}"

if 'Unnamed: 0' not in df.columns:
    raise KeyError("Expected 'Unnamed: 0' column was not found in the dataframe.
 ↪")

columns_to_keep = ['Unnamed: 0'] + sample_columns
df = df.loc[:, columns_to_keep].rename(columns={'Unnamed: 0': 'Genes',␣
 ↪**rename_map})
print(f"Filtered dataframe to {df.shape[0]} rows and {df.shape[1]} columns.")
display(df.head())
df.shape
```

```
Filtered dataframe to 12726 rows and 49 columns.
```

| | Genes | Sub100_22 | Sub101_18 | Sub102_18 | Sub103_10 | Sub104_2 | Sub10_22 | \ |
|---|---|---|---|---|---|---|---|---|
| 0 | A1BG-AS1 | 2 | 2 | 2 | 2 | 3 | 5 | |
| 1 | A2M-AS1 | 10 | 2 | 4 | 1 | 2 | 1 | |
| 2 | A4GALT | 0 | 0 | 0 | 0 | 0 | 0 | |
| 3 | AAAS | 23 | 7 | 14 | 15 | 20 | 47 | |
| 4 | AACS | 12 | 7 | 11 | 7 | 7 | 8 | |

| | Sub11_14 | Sub12_2 | Sub13_18 | … | Sub79_6 | Sub80_2 | Sub86_14 | Sub87_6 | \ |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 13 | 15 | 14 | … | 8 | 3 | 10 | 3 | |

```
1          4        17          9  …         9         2         19         3
2          0         0          1  …         0         0          1         0
3         33        85         47  …        45         8         44        10
4         23        37         25  …        18        12         20         3

   Sub88_22  Sub89_2  Sub92_2  Sub94_18  Sub95_18  Sub96_6
0         2        10        5         5         6        5
1         4        13        1         5         3        4
2         2         0        0         0         4        0
3        24        35       19        46        44       32
4         9        23        8        16        17       14

[5 rows x 49 columns]
```

[3]:
```
(12726, 49)
```

This cell checks which time point each subject was sampled at.

- It parses the `Sub<subject>_<time>` column names.
- For each subject, it records the corresponding time.
- It then prints lines such as `Subject 1: 10`, meaning subject 1 was sampled at 10:00 at sea level.

This confirms that, in this dataset, each subject appears at exactly one time point at sea level. There are no repeated measurements of the same person across different times.

[4]:
```
subject_pattern = re.compile(r'^Sub(?P<subject>\d+)_(?P<time>\d+)$')
subject_timepoints = {}
for col in df.columns:
    match = subject_pattern.match(str(col))
    if match:
        subj = match.group('subject')
        time = int(match.group('time'))
        subject_timepoints.setdefault(subj, set()).add(time)

for subj in sorted(subject_timepoints, key=lambda x: int(x)):
    times = sorted(subject_timepoints[subj])
    formatted_times = ','.join(str(t) for t in times)
    print(f'Subject {subj}: {formatted_times}')
```

```
Subject 1: 10
Subject 2: 6
Subject 3: 22
Subject 10: 22
Subject 11: 14
Subject 12: 2
Subject 13: 18
Subject 15: 6
Subject 16: 22
Subject 17: 18
```

```
Subject 18: 10
Subject 20: 14
Subject 21: 6
Subject 22: 14
Subject 23: 2
Subject 24: 22
Subject 33: 10
Subject 34: 14
Subject 36: 10
Subject 37: 14
Subject 38: 10
Subject 40: 10
Subject 57: 2
Subject 58: 10
Subject 59: 14
Subject 60: 18
Subject 62: 2
Subject 63: 18
Subject 73: 22
Subject 75: 6
Subject 76: 14
Subject 77: 6
Subject 78: 22
Subject 79: 6
Subject 80: 2
Subject 86: 14
Subject 87: 6
Subject 88: 22
Subject 89: 2
Subject 92: 2
Subject 94: 18
Subject 95: 18
Subject 96: 6
Subject 100: 22
Subject 101: 18
Subject 102: 18
Subject 103: 10
Subject 104: 2
```

This cell inverts the mapping and summarizes the sampling design by time of day.

- For each sea-level sample, it extracts the time (2, 6, 10, 14, 18, 22).
- It groups subjects by time and prints lines like:
  - Time 2: 8 subjects (12,23,57,62,80,89,92,104)
  - Time 6: 8 subjects (2,15,21,75,77,79,87,96)
  - …

This confirms that at sea level we have a **balanced cross-sectional design**:

- 6 time points over the 24-hour day.
- 8 independent subjects at each time point.

We use this design in all the downstream time-of-day analyses.

[5]:
```
time_counts = {}
for col in df.columns:
    match = subject_pattern.match(str(col))
    if match:
        time = int(match.group('time'))
        subj = match.group('subject')
        time_counts.setdefault(time, set()).add(subj)

for time in sorted(time_counts):
    subjects = time_counts[time]
    count = len(subjects)
    subject_list = ','.join(sorted(subjects, key=lambda x: int(x)))
    print(f'Time {time}: {count} subjects ({subject_list})')
```

```
Time 2: 8 subjects (12,23,57,62,80,89,92,104)
Time 6: 8 subjects (2,15,21,75,77,79,87,96)
Time 10: 8 subjects (1,18,33,36,38,40,58,103)
Time 14: 8 subjects (11,20,22,34,37,59,76,86)
Time 18: 8 subjects (13,17,60,63,94,95,101,102)
Time 22: 8 subjects (3,10,16,24,73,78,88,100)
```

### 1.2 "Do these cancer-related genes show similar time-of-day patterns across different people at sea level?"

This cell constructs a simple "metadata" table describing each sample.

- It loops over all expression columns in `df` (the `Sub<subject>_<time>` columns).
- For each one, it extracts:
  - `sample`: the column name (e.g. `Sub100_22`).
  - `subject`: the numeric subject ID (e.g. 100).
  - `time`: the sampling time in hours (e.g. 22).

It combines this information into a small table called `meta` with three columns:

- `sample`, `subject`, `time`.

We later merge `meta` with gene expression values, so that we always know which person and time each measurement comes from.

[6]:
```
expr = df.copy()  # df after renaming, with 'Genes' + SubX_time columns
subject_pattern = re.compile(r'^Sub(?P<subject>\d+)_(?P<time>\d+)$')

meta_rows = []
for col in expr.columns:
    m = subject_pattern.match(col)
    if m:
```

```
        meta_rows.append({
            "sample": col,
            "subject": int(m.group("subject")),
            "time": int(m.group("time"))
        })

meta = pd.DataFrame(meta_rows)
```

This cell focuses the analysis on a curated circadian gene set. Steps:

1. **Read the Reactome circadian clock file**
   - It loads `REACTOME_CIRCADIAN_CLOCK.v2025.1.Hs (1).tsv`, which lists genes associated with the Reactome "Circadian Clock" pathway.
2. **Extract the gene symbols**
   - It finds the row called `GENE_SYMBOLS` and splits the comma-separated list into individual gene names.
3. **Compute the overlap with our dataset**
   - It creates a set of gene names present in our sea-level expression data (`df['Genes']`).
   - It intersects this set with the Reactome circadian gene set.
   - It prints:
     - Total genes in our dataset.
     - Total genes in the Reactome circadian pathway.
     - Number of genes common to both (99 in this case).
   - It also displays that list of overlapping genes.

The result is a list of 99 circadian-related genes that are actually measured in the sea-level dataset. These are the genes we follow in the time-of-day analysis below.

[7]:
```
reactome_path = os.path.join('Data', 'REACTOME_CIRCADIAN_CLOCK.v2025.1.Hs (1).
 ↪tsv')
reactome_df = pd.read_csv(reactome_path, sep='\t', names=['STANDARD_NAME',␣
 ↪'VALUE'], header=0)
gene_symbols_row = reactome_df.loc[reactome_df['STANDARD_NAME'] ==␣
 ↪'GENE_SYMBOLS', 'VALUE']
if gene_symbols_row.empty:
    raise ValueError('Could not locate GENE_SYMBOLS entry in the Reactome file.
 ↪')
reactome_genes = [gene.strip() for gene in gene_symbols_row.iloc[0].split(',')␣
 ↪if gene.strip()]

if 'Genes' not in df.columns:
    raise KeyError("Expected 'Genes' column to be present in the dataframe. Run␣
 ↪the filtering cell first.")
dataset_genes = df['Genes'].astype(str).str.strip()
dataset_gene_set = set(dataset_genes)
reactome_gene_set = set(reactome_genes)
```

```
overlap_genes = sorted(dataset_gene_set & reactome_gene_set)

print(f'Total genes in dataset: {len(dataset_gene_set)}')
print(f'Total genes in Reactome set: {len(reactome_gene_set)}')
print(f'Intersecting genes: {len(overlap_genes)}')
display(pd.DataFrame({'Reactome_Circadian_Genes': overlap_genes}))
```

```
Total genes in dataset: 12726
Total genes in Reactome set: 112
Intersecting genes: 99
```

| | Reactome_Circadian_Genes |
|---|---|
| 0 | ADRM1 |
| 1 | ATF2 |
| 2 | BHLHE40 |
| 3 | BHLHE41 |
| 4 | BTRC |
| .. | … |
| 94 | TGS1 |
| 95 | UBA52 |
| 96 | UBB |
| 97 | UBC |
| 98 | UBE2D1 |

[99 rows x 1 columns]

This cell creates a smaller expression matrix that contains only the overlapping circadian genes.

- It filters `df` to keep only rows whose gene name is in `overlap_genes`.
- The resulting table `df_intersection` has:
  - 99 rows (the circadian genes present in the dataset).
  - 49 columns (the 48 sea-level samples plus the `Genes` column).

This reduces the dataset to a focused panel of circadian-related genes at sea level, which makes the subsequent analysis and interpretation more targeted.

[8]:
```
if 'overlap_genes' not in locals():
    raise NameError('Run the Reactome overlap cell first to compute␣
 ↪overlap_genes.')

df_intersection = df[df['Genes'].isin(overlap_genes)].copy()
print(f'Intersection dataset shape: {df_intersection.shape}')
display(df_intersection.head())
```

```
Intersection dataset shape: (99, 49)
```

| | Genes | Sub100_22 | Sub101_18 | Sub102_18 | Sub103_10 | Sub104_2 | Sub10_22 | \ |
|---|---|---|---|---|---|---|---|---|
| 238 | ADRM1 | 96 | 49 | 87 | 67 | 52 | 149 | |
| 761 | ATF2 | 10 | 0 | 7 | 2 | 11 | 7 | |
| 1030 | BHLHE40 | 10 | 12 | 12 | 23 | 12 | 32 | |
| 1031 | BHLHE41 | 0 | 0 | 1 | 0 | 2 | 0 | |

```
1151     BTRC          7          1          4          0          3          3

      Sub11_14  Sub12_2  Sub13_18  …  Sub79_6  Sub80_2  Sub86_14  Sub87_6  \
238        178      304       254  …      205       67       203       67
761         23       20        19  …       10       10         8        4
1030        30       45        31  …       59       21        30       18
1031         0        1         1  …        1        0         0        0
1151        10       18        12  …        7        9        13        6

      Sub88_22  Sub89_2  Sub92_2  Sub94_18  Sub95_18  Sub96_6
238        105      203      123       124       168      110
761          5       16        3         6         8        4
1030        23       25       28        23        35       13
1031         0        1        0         0         2        1
1151        11       21        4         4        16        7

[5 rows x 49 columns]
```

### 1.3 Log-transform, standardize and reshape into a "temporal" table

This cell prepares the circadian gene data for statistical analysis.

1. **Extract the numeric expression matrix**
   - It takes `df_intersection` and keeps only the expression columns (drops the `Genes` column).
   - It sets gene names as the index.
2. **Log2 transform the counts**
   - It computes `log2(count + 1)` for each gene–sample entry.
   - This reduces skewness and makes differences more interpretable.
3. **Per-gene z-scoring**
   - For each gene, it computes the mean and standard deviation across all 48 sea-level samples.
   - It converts the log expression to a z-score:
     - `(value - gene_mean) / gene_sd`.
   - This answers "how many standard deviations above or below its own average is this gene in this sample?"
4. **Reshape into long format**
   - It melts the matrices into a single table with columns:
     - `Genes`, `sample`, `expression` (raw count),
     - `log_expression`, `log_zscore`.
   - It adds `subject` and `time` to each row by parsing the `sample` name.
   - It renames `Genes` to `gene`.

The final table `temporal_df` has:

- 4,752 rows = 99 genes × 48 samples.

- 7 columns: `gene`, `sample`, `expression`, `log_expression`, `log_zscore`, `subject`, `time`.

Each row now represents the expression of one circadian gene in one subject at one time point at sea level, with normalized values ready for plotting and modelling.

[9]:
```
if 'df_intersection' not in locals():
    raise NameError('Run the intersection cell first to compute df_intersection.
 ↪')

expression_cols = [col for col in df_intersection.columns if col != 'Genes']
expr_matrix = df_intersection.set_index('Genes')[expression_cols]

log_expr = np.log2(expr_matrix + 1)
mean_per_gene = log_expr.mean(axis=1)
std_per_gene = log_expr.std(axis=1).replace(0, np.nan)
log_expr_z = log_expr.sub(mean_per_gene, axis=0).div(std_per_gene, axis=0)

long_expression = expr_matrix.reset_index().melt(id_vars='Genes',␣
 ↪var_name='sample', value_name='expression')
long_log = log_expr.reset_index().melt(id_vars='Genes', var_name='sample',␣
 ↪value_name='log_expression')
long_z = log_expr_z.reset_index().melt(id_vars='Genes', var_name='sample',␣
 ↪value_name='log_zscore')
temporal_df = long_expression.merge(long_log, on=['Genes', 'sample']).
 ↪merge(long_z, on=['Genes', 'sample'])

pattern = re.compile(r'^Sub(?P<subject>\d+)_(?P<time>\d+)$')
subject_matches = temporal_df['sample'].apply(lambda x: pattern.match(str(x)))
temporal_df['subject'] = subject_matches.apply(lambda m: m.group('subject') if␣
 ↪m else None)
temporal_df['time'] = subject_matches.apply(lambda m: int(m.group('time')) if m␣
 ↪else None)
temporal_df = temporal_df.rename(columns={'Genes': 'gene'})

print(f'Temporal dataframe shape: {temporal_df.shape}')
display(temporal_df.head())
temporal_df.shape
```

```
Temporal dataframe shape: (4752, 7)
```

| | gene | sample | expression | log_expression | log_zscore | subject | time |
|---|---|---|---|---|---|---|---|
| 0 | ADRM1 | Sub100_22 | 96 | 6.599913 | -0.752335 | 100 | 22 |
| 1 | ATF2 | Sub100_22 | 10 | 3.459432 | 0.376461 | 100 | 22 |
| 2 | BHLHE40 | Sub100_22 | 10 | 3.459432 | -1.960934 | 100 | 22 |
| 3 | BHLHE41 | Sub100_22 | 0 | 0.000000 | -1.182521 | 100 | 22 |
| 4 | BTRC | Sub100_22 | 7 | 3.000000 | -0.119535 | 100 | 22 |

[9]: (4752, 7)

This cell generates one figure per gene to visualize its time-of-day pattern at sea level.

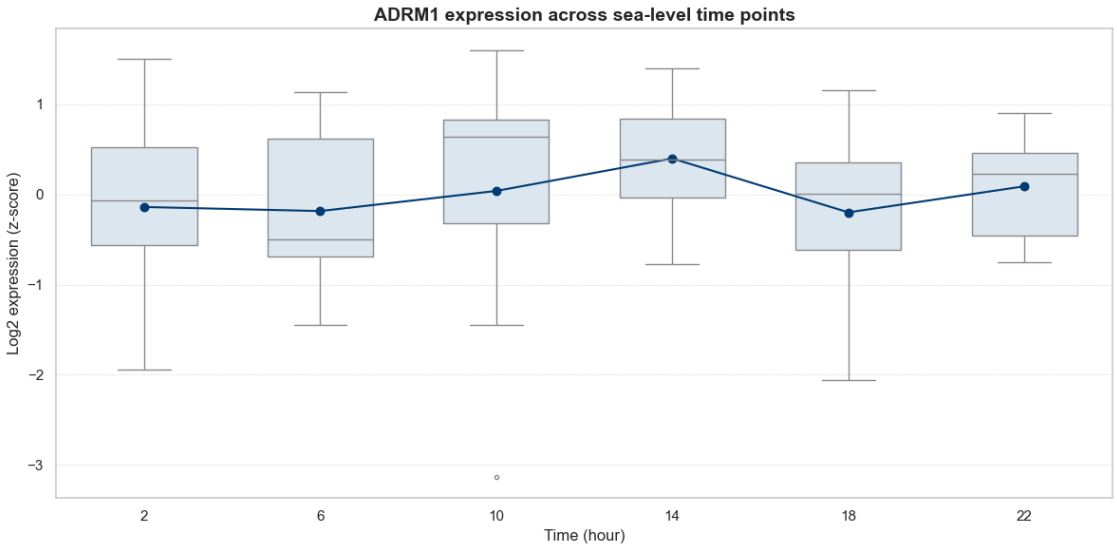
For each of the 99 circadian genes:

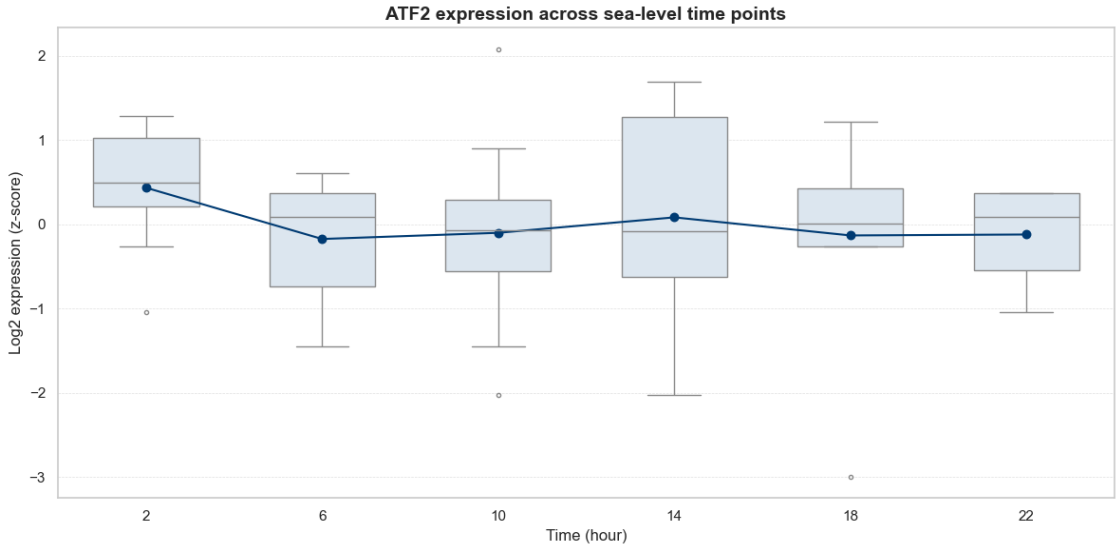
- It extracts all rows in `temporal_df` for that gene.
- It draws:
  - A boxplot of z-scored log expression (`log_zscore`) at each time point.
  - A dashed line connecting the mean at each time point (to highlight the average trend).
- It labels axes:
  - X-axis: sampling time (2, 6, 10, 14, 18, 22).
  - Y-axis: "Log2 expression z-score".
- It saves each figure to `Data/plots/` with the file name `<GENE>_temporal_expression.png`.

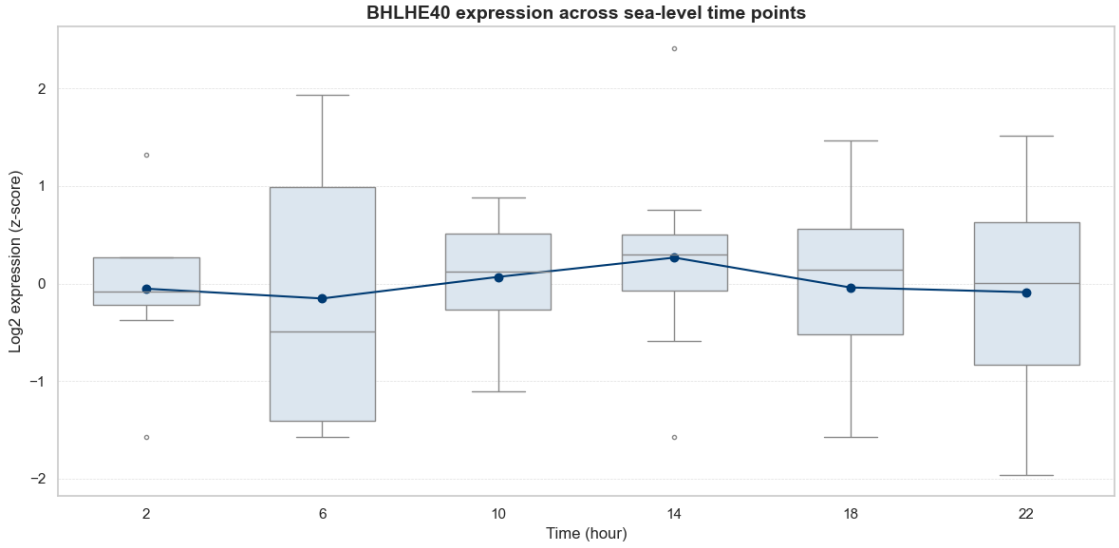
These plots provide a quick visual check of whether any circadian genes clearly rise or fall at specific times of day across the group of sea-level subjects.

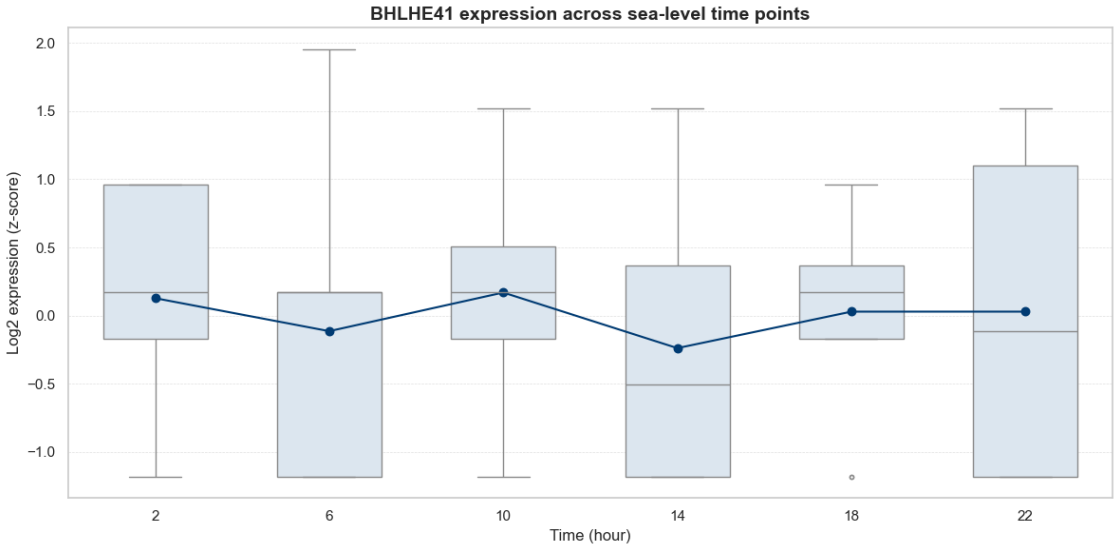
[10]:
```
if 'temporal_df' not in locals():
    raise NameError('Run the temporal formatting cell first to compute␣
 ↪temporal_df.')

plots_dir = os.path.join('Data', 'plots')
os.makedirs(plots_dir, exist_ok=True)

unique_genes = temporal_df['gene'].unique()
for gene in unique_genes:
    gene_df = temporal_df[temporal_df['gene'] == gene]

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

    # Boxplot with soft neutral color
    sns.boxplot(
        data=gene_df,
        x='time',
        y='log_zscore',
        color='#D9E6F2',
        width=0.6,
        fliersize=3
    )

    # Overlay the mean trend line
    sns.pointplot(
        data=gene_df,
        x='time',
        y='log_zscore',
        color='#003B73',
        linestyle='-',
        marker='o',
        markersize=6,
        linewidth=1.5,
```

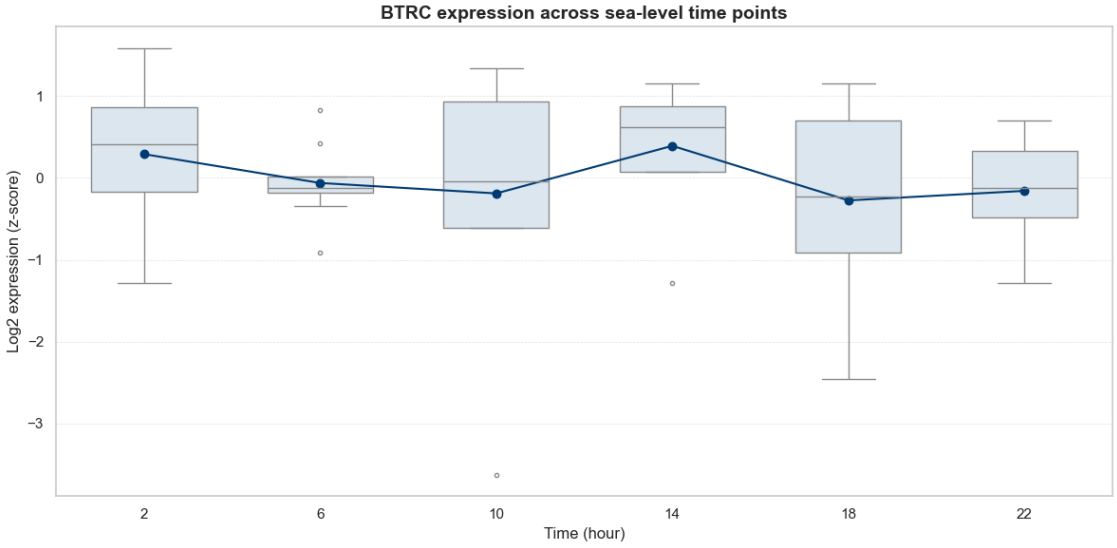
```
        errorbar=None
    )

    # Titles and labels
    plt.title(f'{gene} expression across sea-level time points', fontsize=14,␣
↪weight='bold')
    plt.xlabel('Time (hour)', fontsize=12)
    plt.ylabel('Log2 expression (z-score)', fontsize=12)

    # Soft clean background
    plt.grid(axis='y', linestyle='--', linewidth=0.5, alpha=0.6)

    # Improve spacing
    plt.tight_layout()

    # Save high-quality version
    plot_path = os.path.join(plots_dir, f"{gene}_temporal_expression.png")
    plt.savefig(plot_path, dpi=300)

    plt.show()
    plt.close()
```

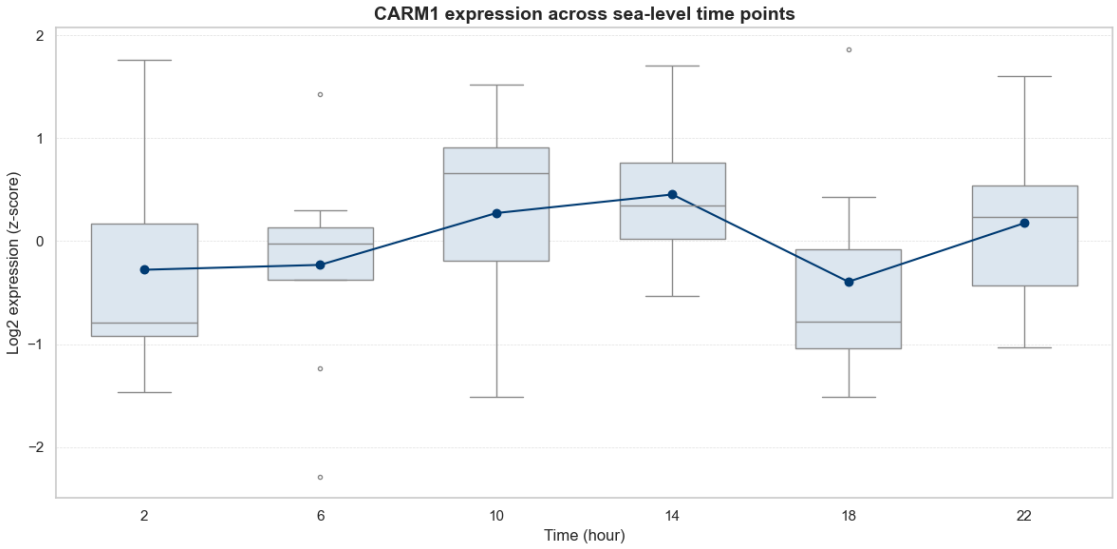


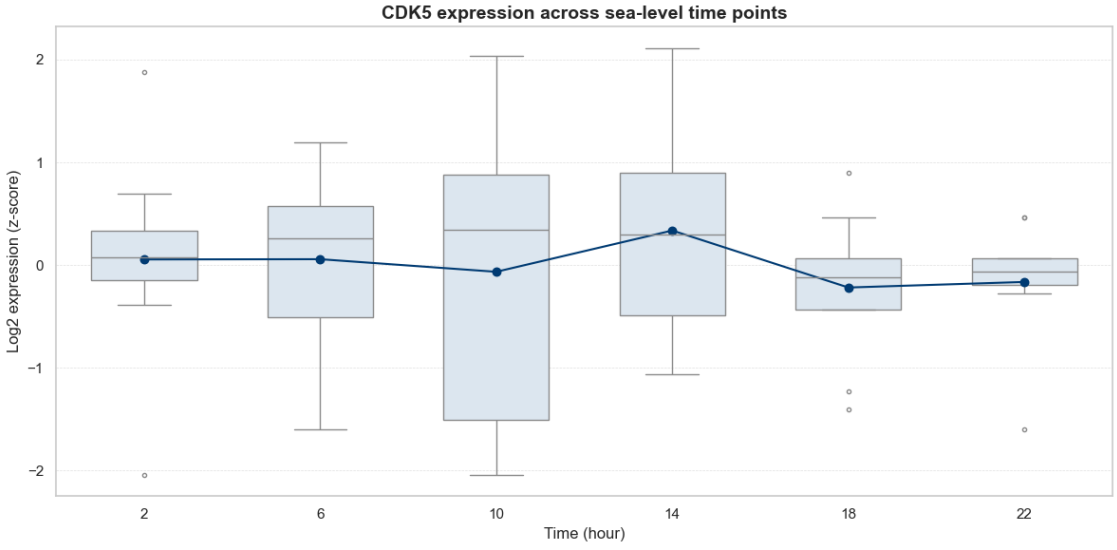

ATF2 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

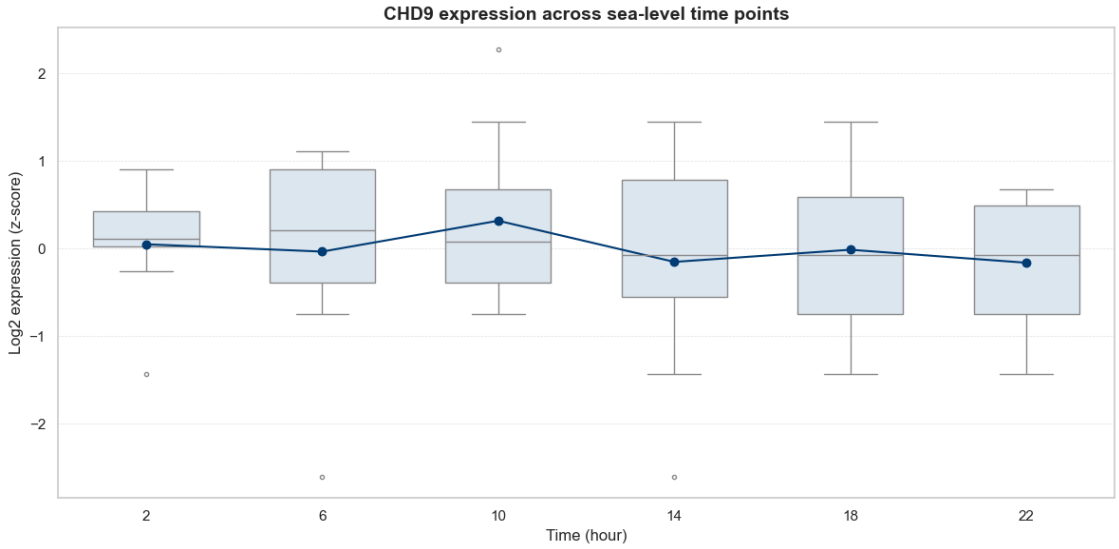

BHLHE40 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

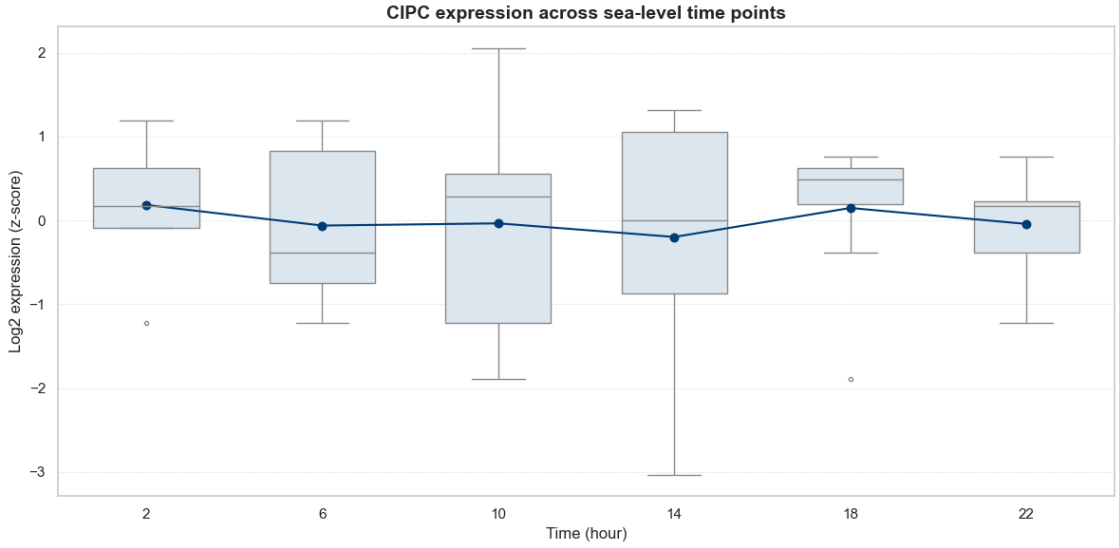

BHLHE41 expression across sea-level time points
Log2 expression (z-score)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
2
6
10
14
18
22
Time (hour)

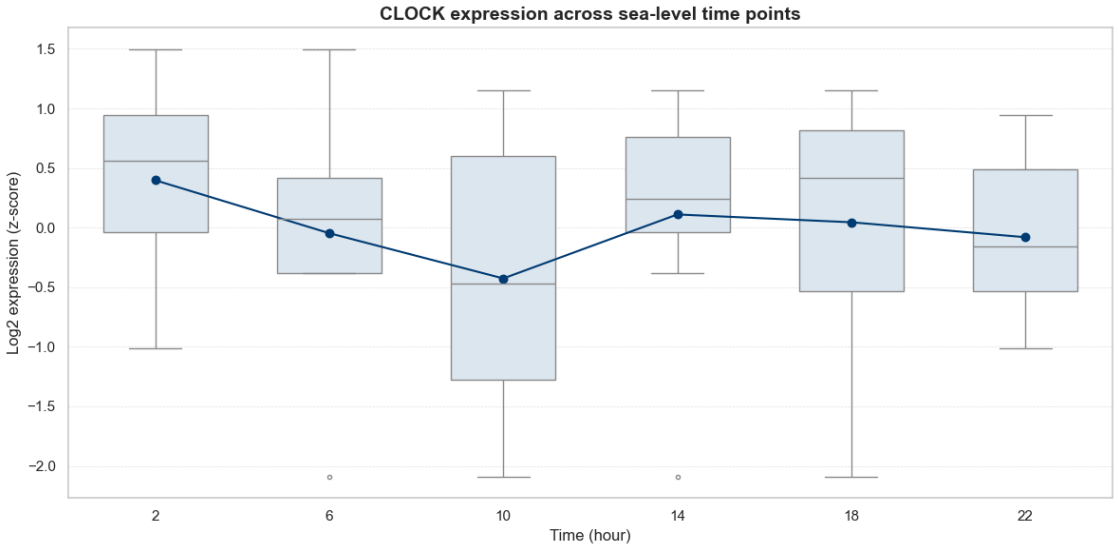

BTRC expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

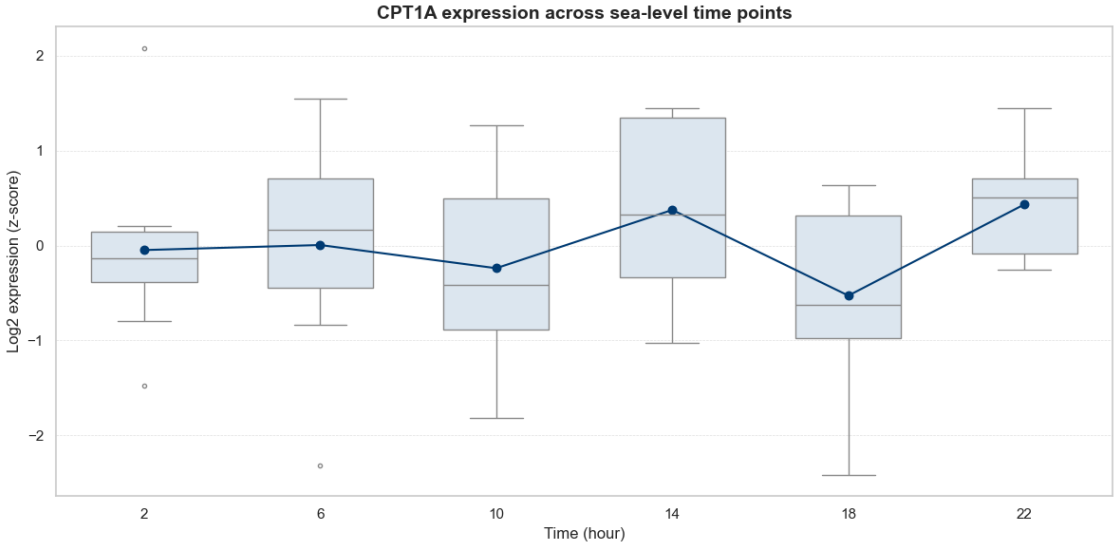

CARM1 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

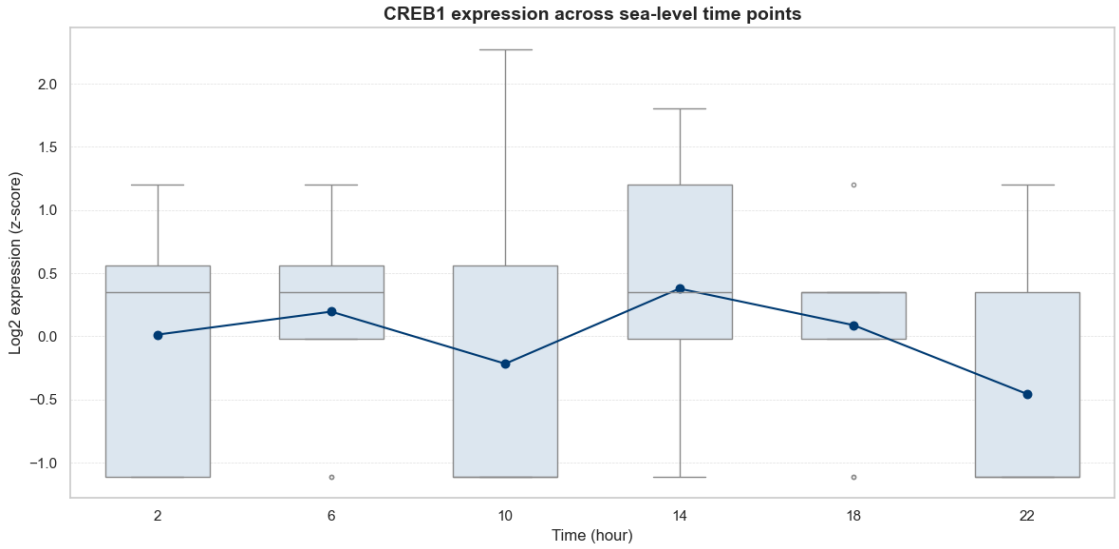

CDK5 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

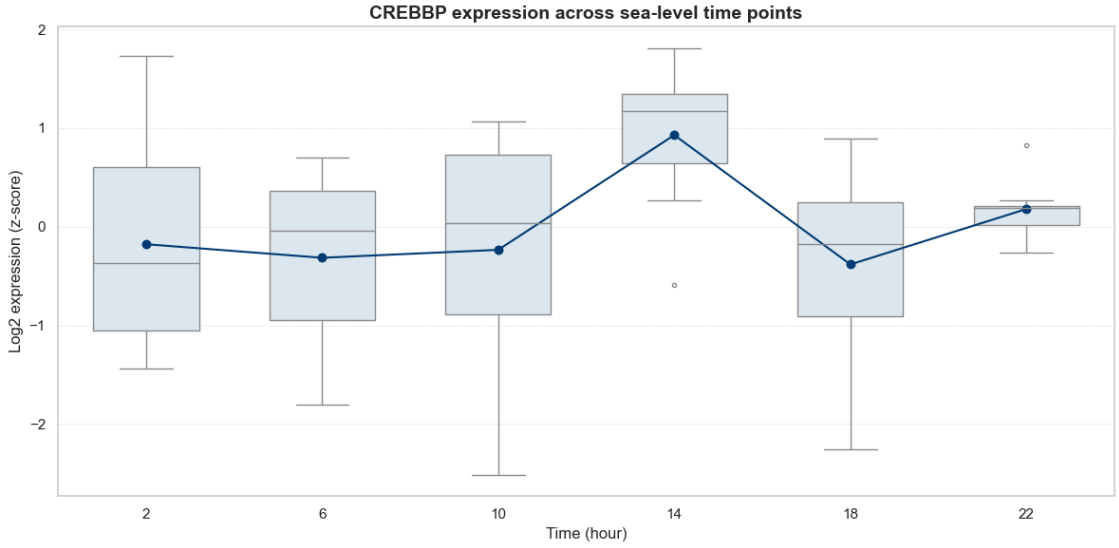

CHD9 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

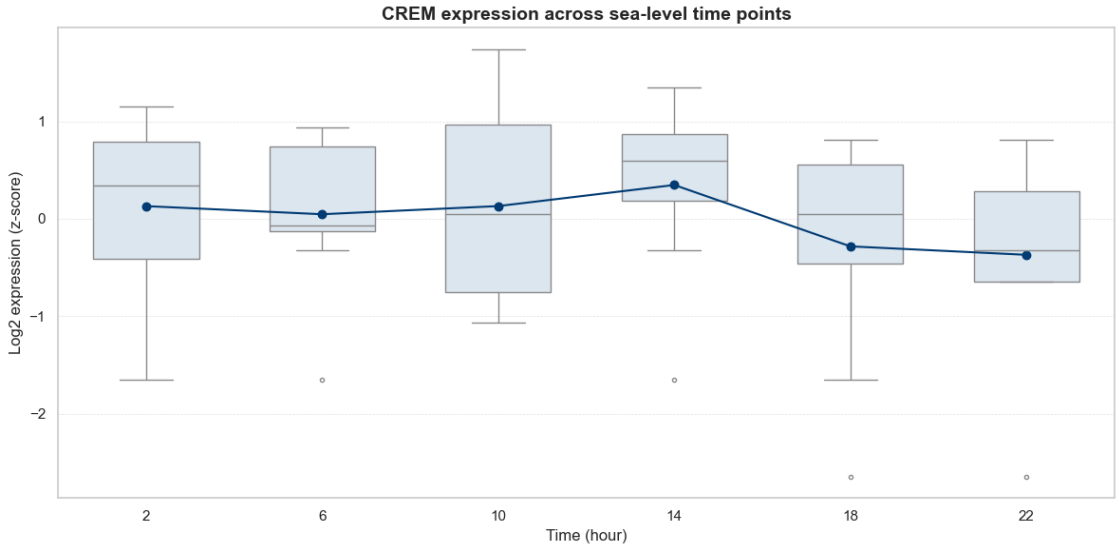

CIPC expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

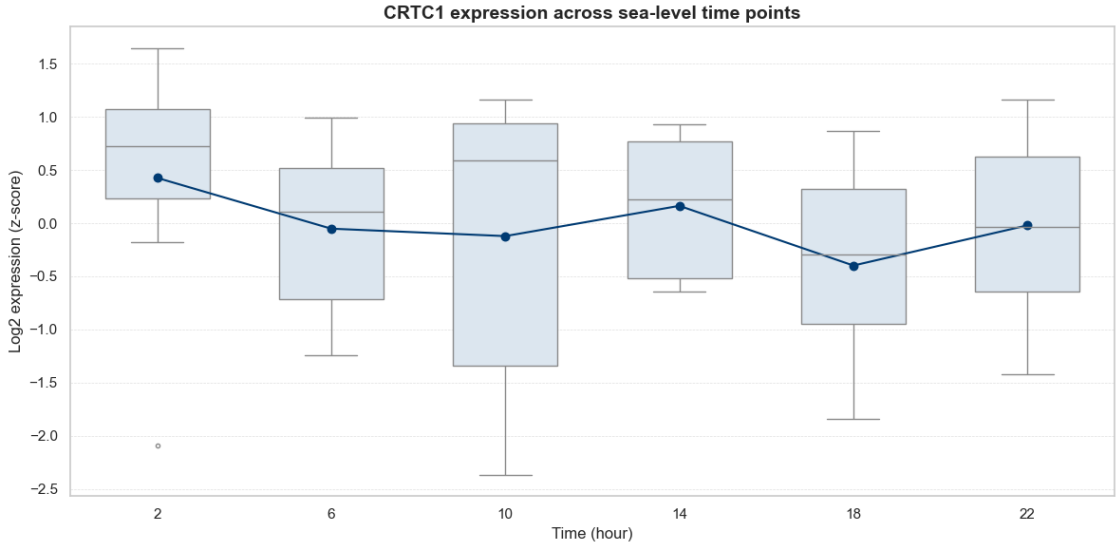

CLOCK expression across sea-level time points
Log2 expression (z-score)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
2
6
10
14
18
22
Time (hour)

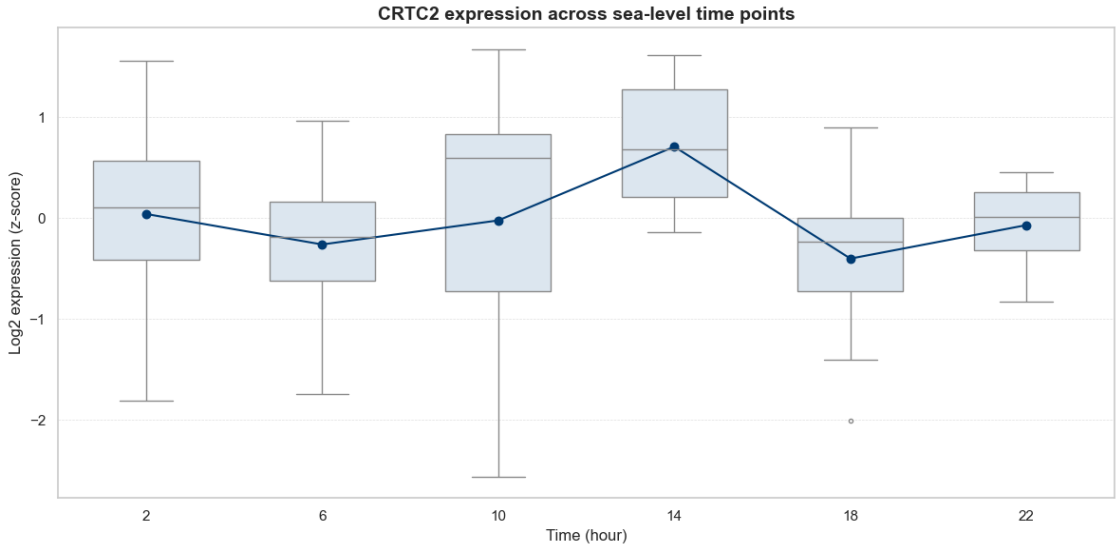

CPT1A expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

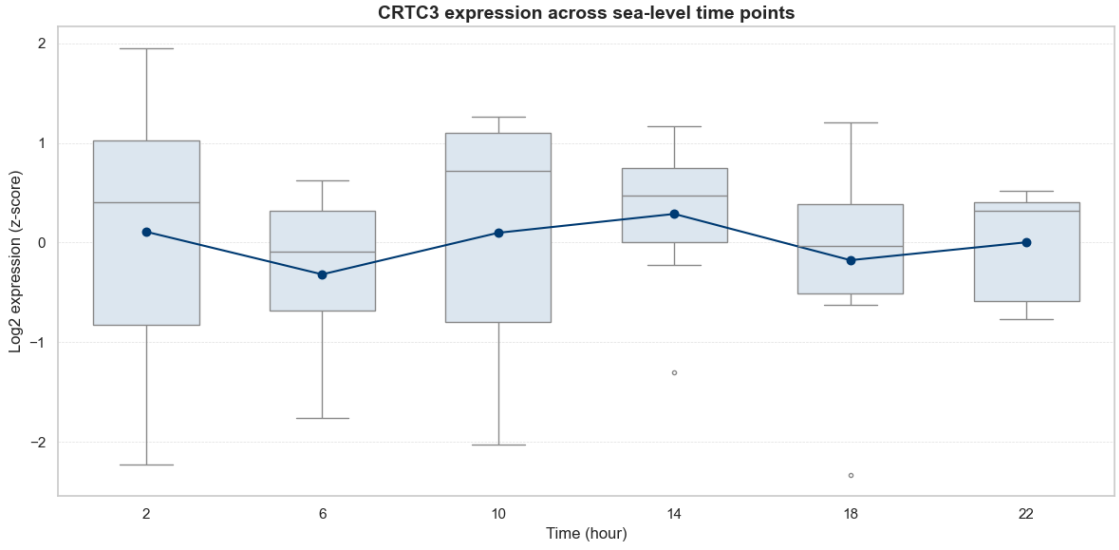

CREB1 expression across sea-level time points
Log2 expression (z-score)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
2
6
10
14
18
22
Time (hour)

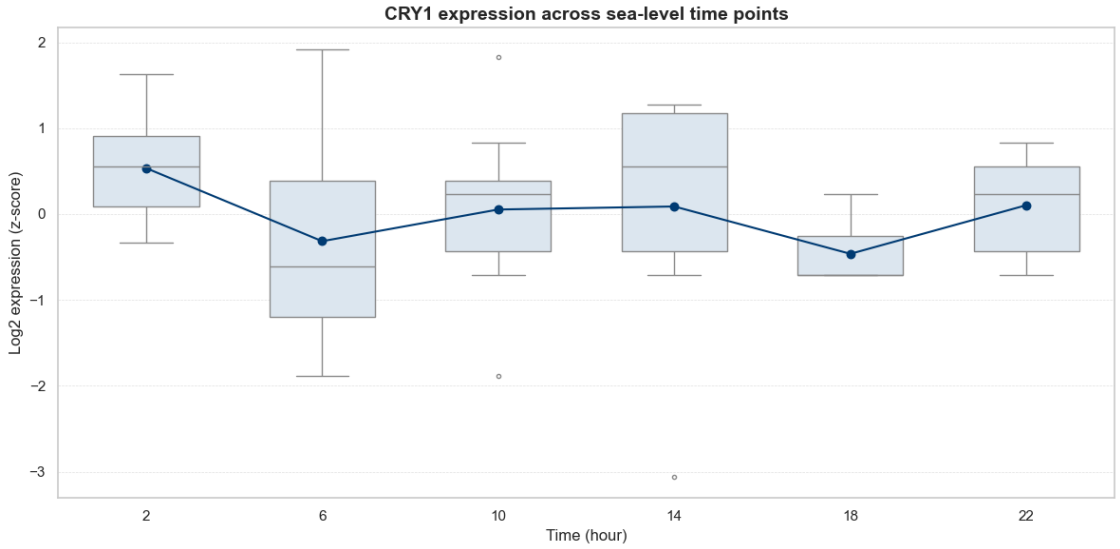

CREBBP expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

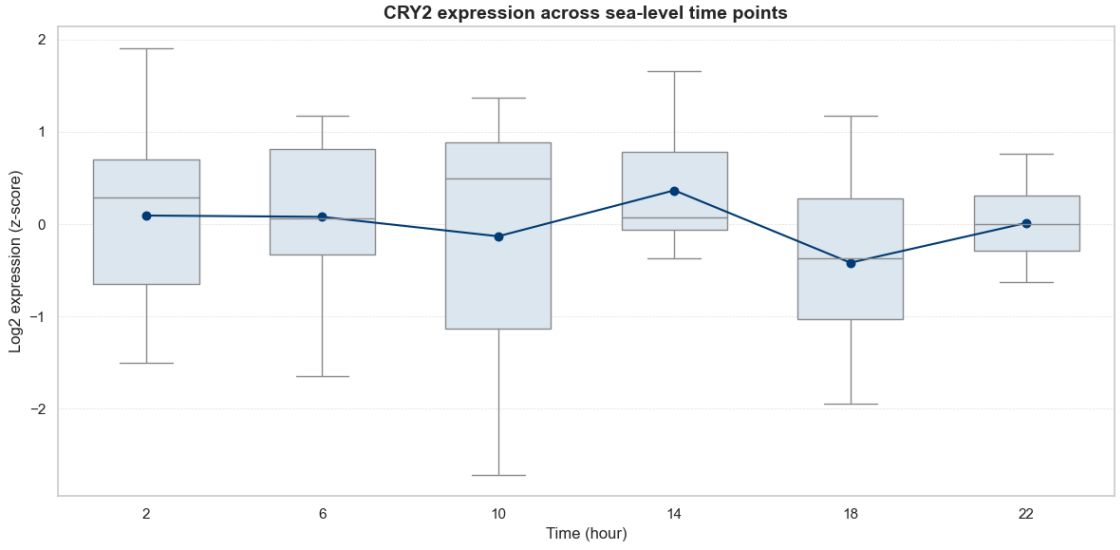

CREM expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

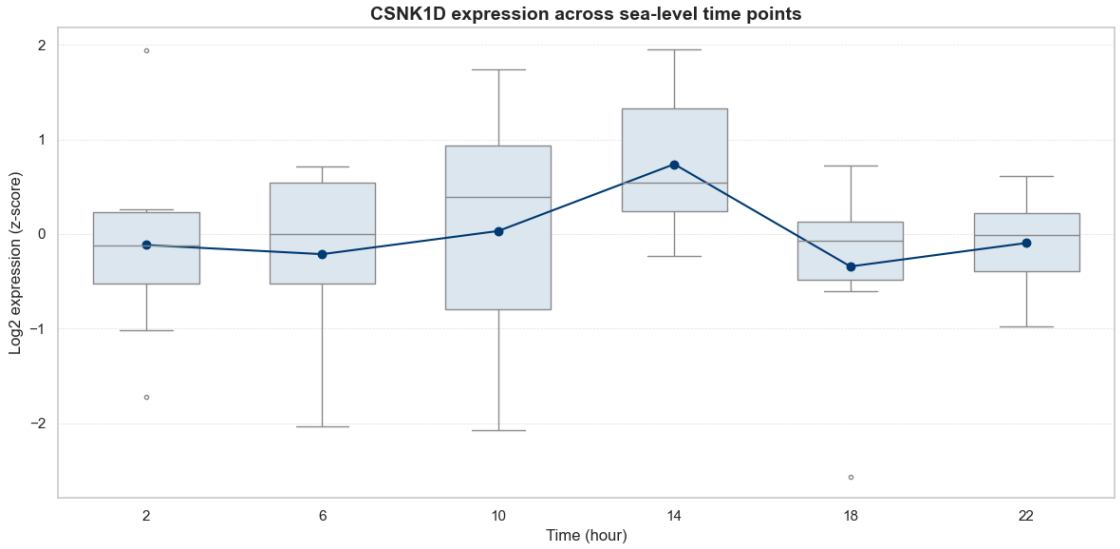

CRTC1 expression across sea-level time points
Log2 expression (z-score)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
2
6
10
14
18
22
Time (hour)

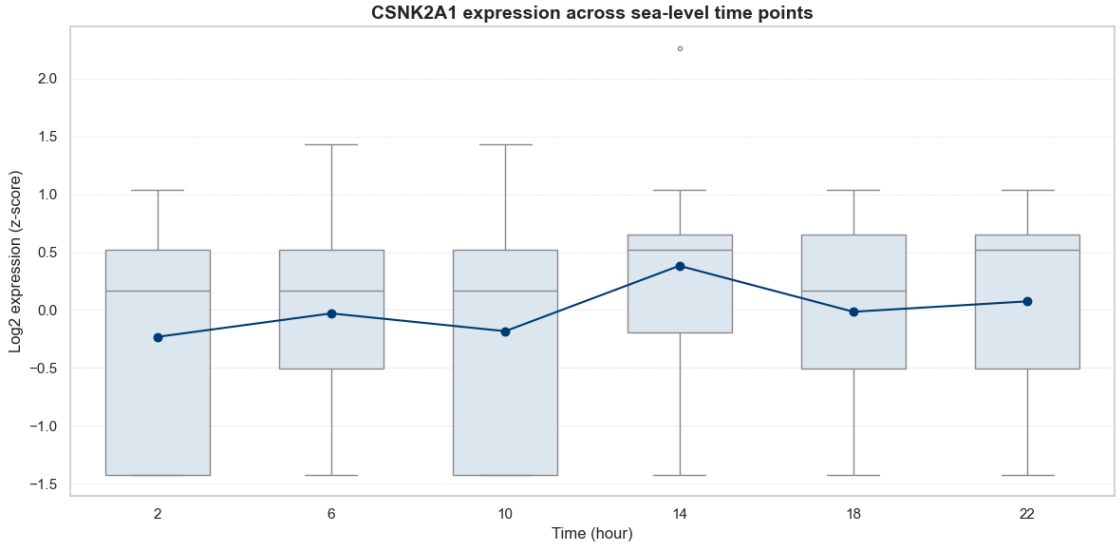

CRTC2 expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

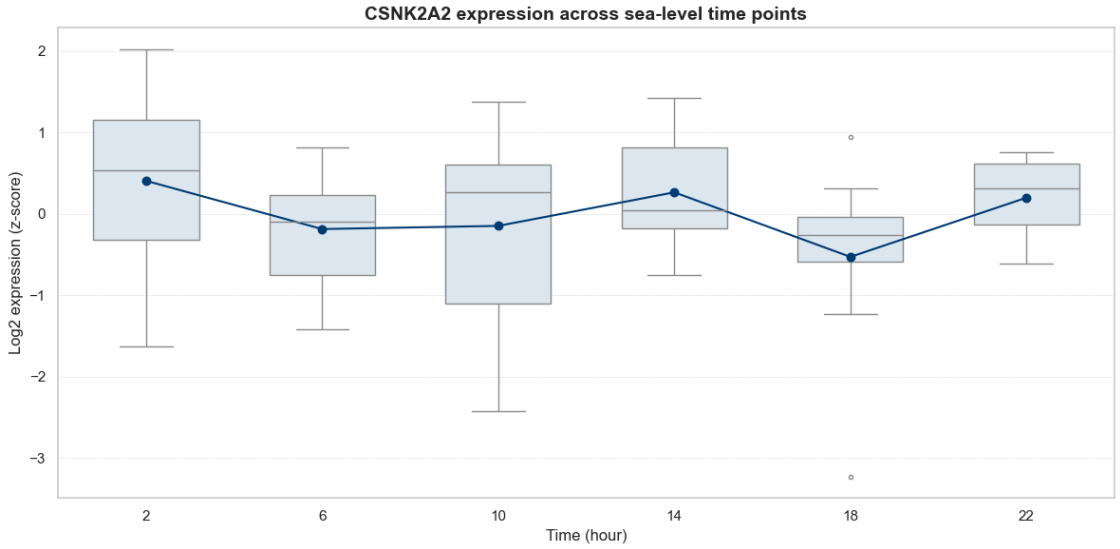

CRTC3 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

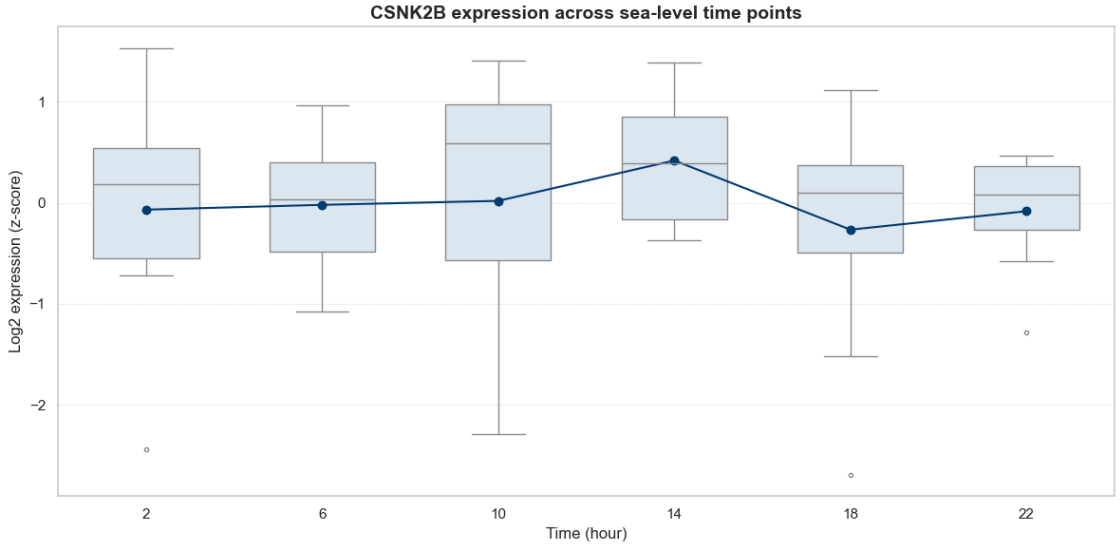

CRY1 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

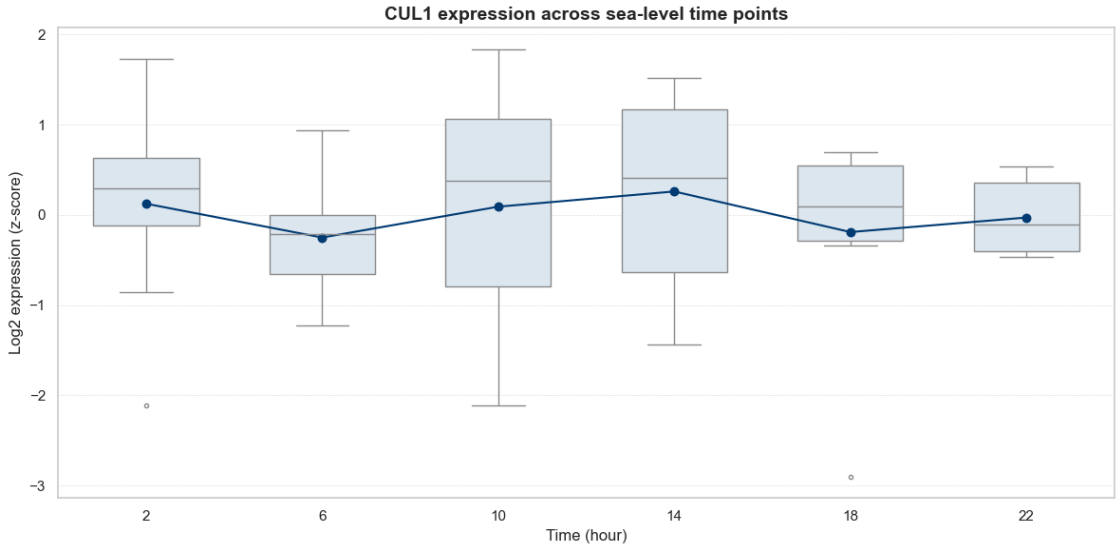

CRY2 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

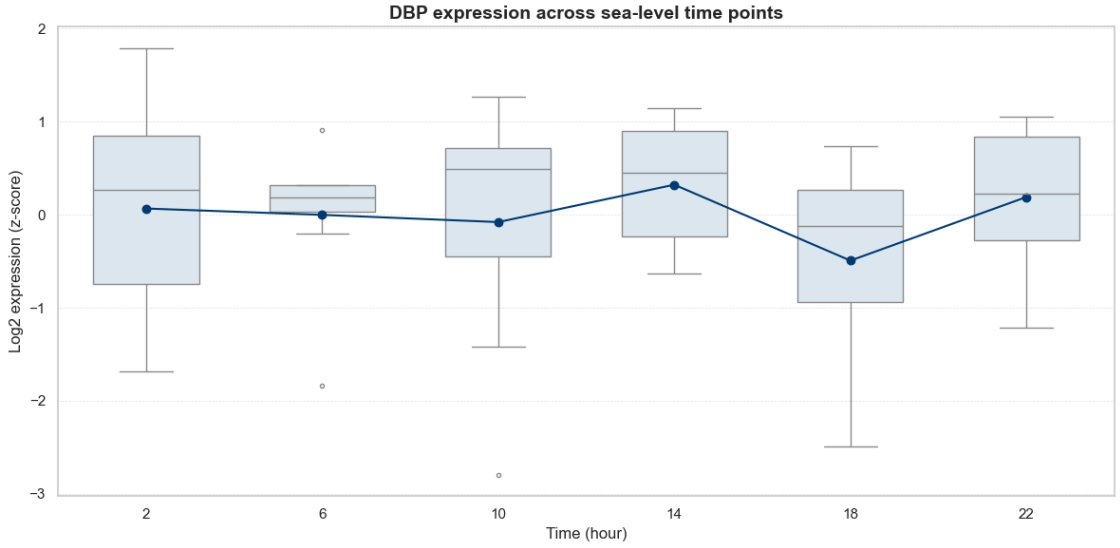

CSNK1D expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

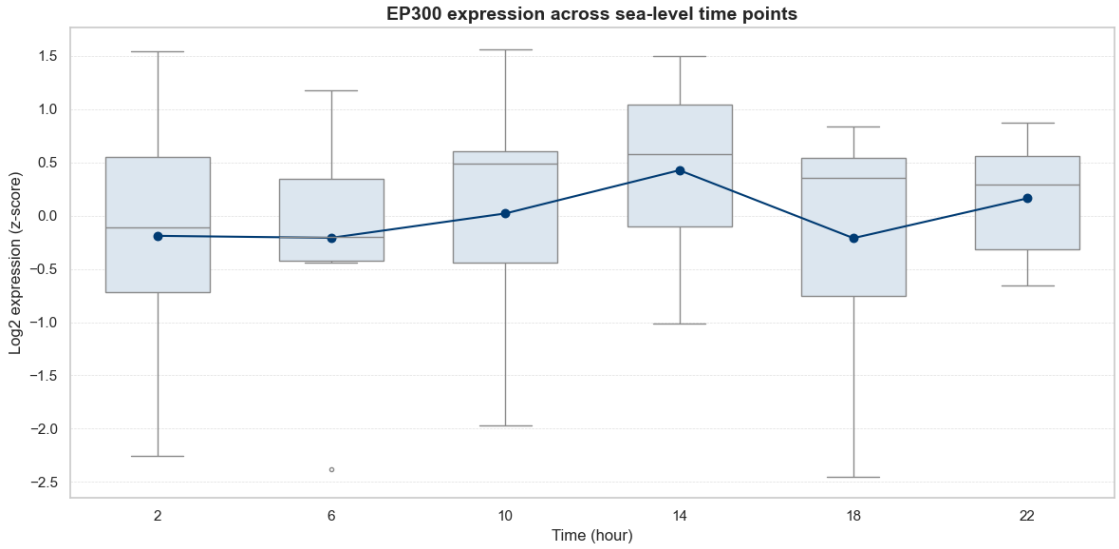

CSNK2A1 expression across sea-level time points
Log2 expression (z-score)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
2
6
10
14
18
22
Time (hour)

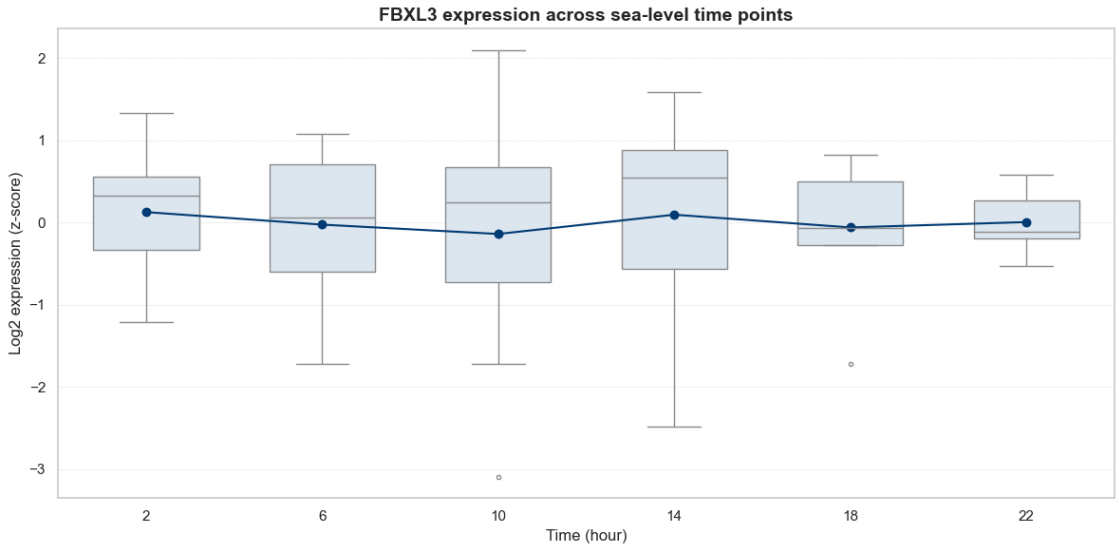

CSNK2A2 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

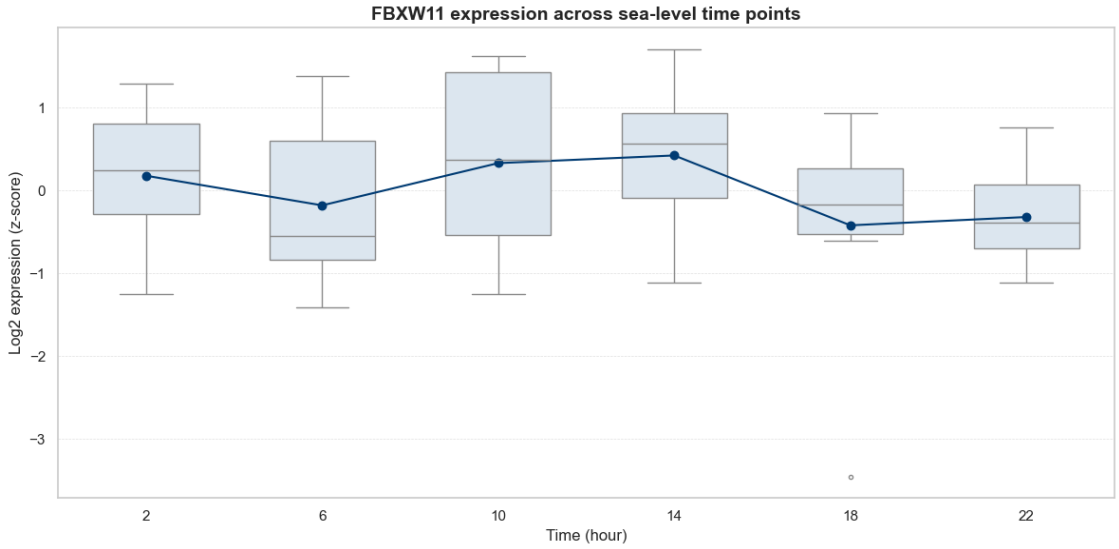

CSNK2B expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

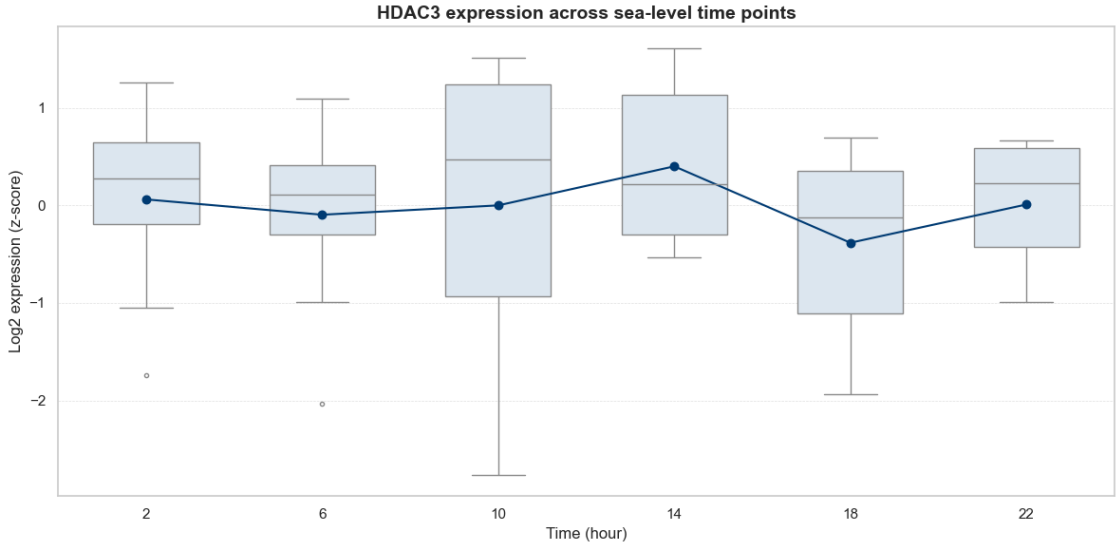

CUL1 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

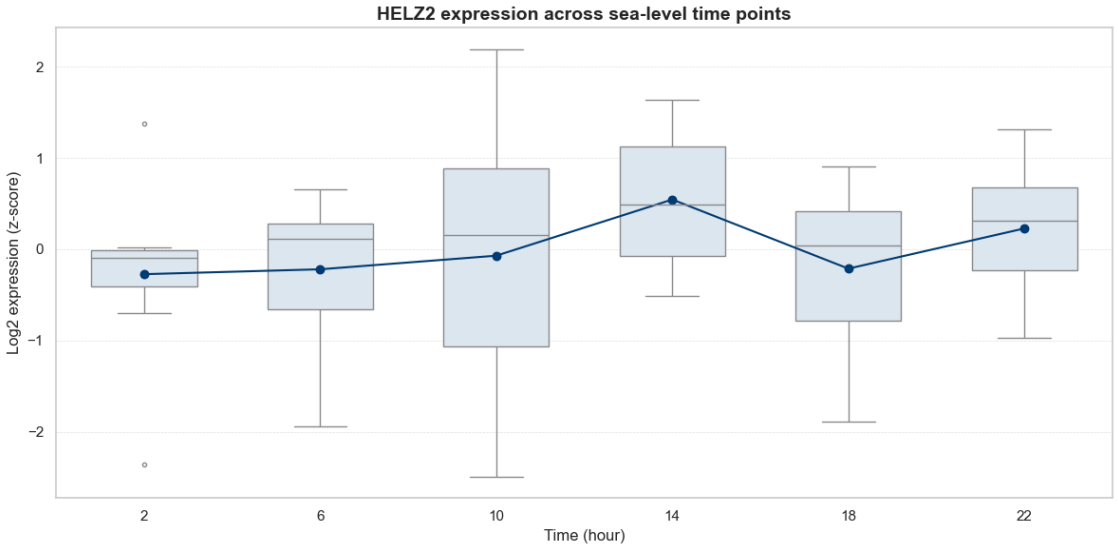

DBP expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

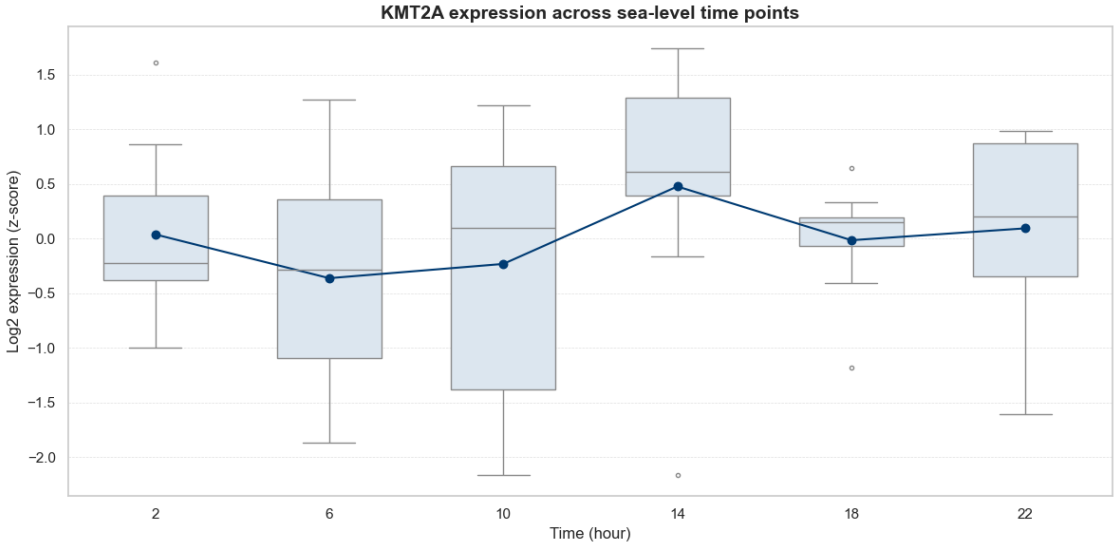

EP300 expression across sea-level time points
Log2 expression (z-score)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
2
6
10
14
18
22
Time (hour)

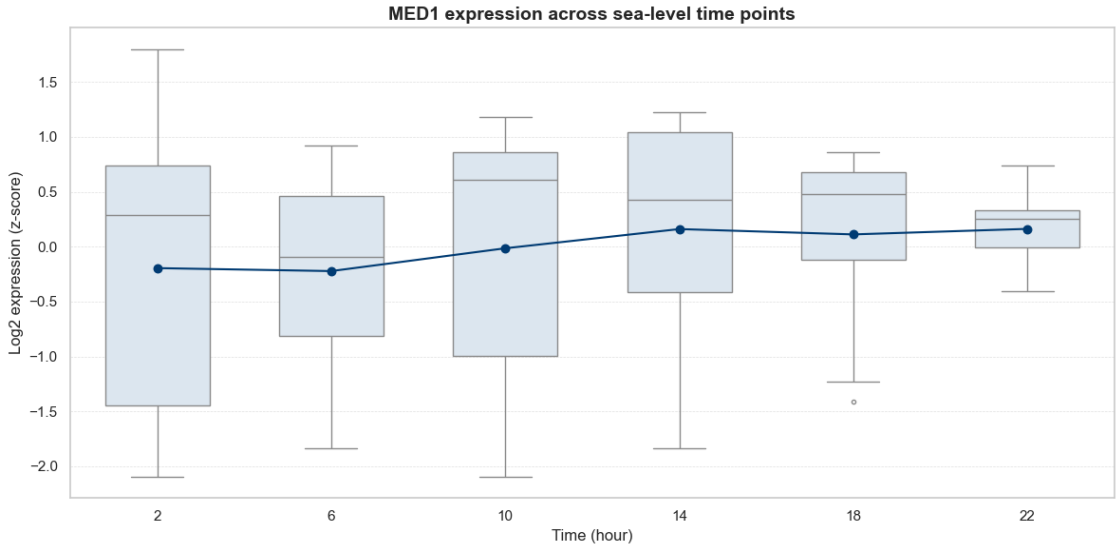

FBXL3 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

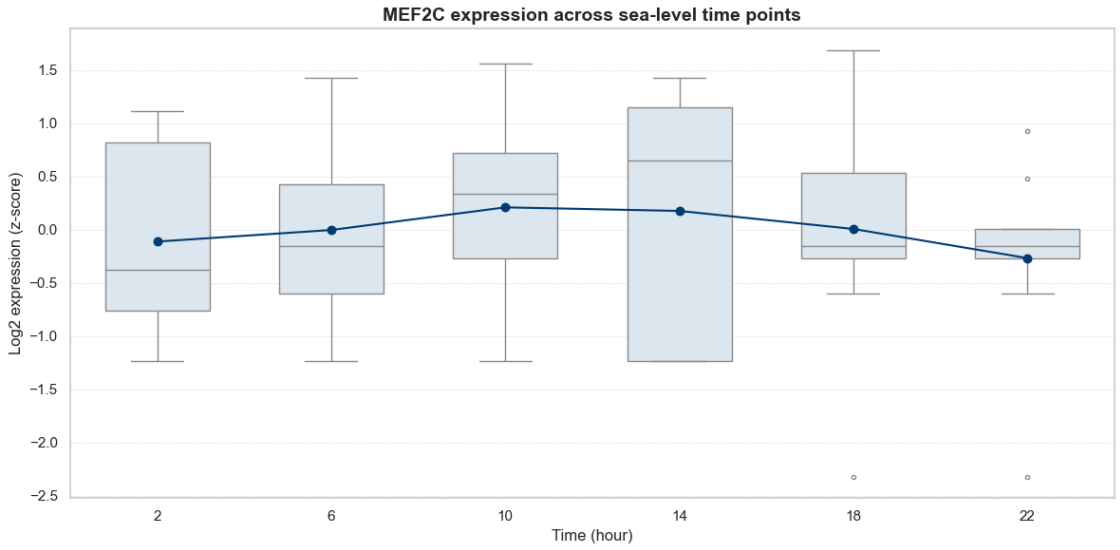

FBXW11 expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

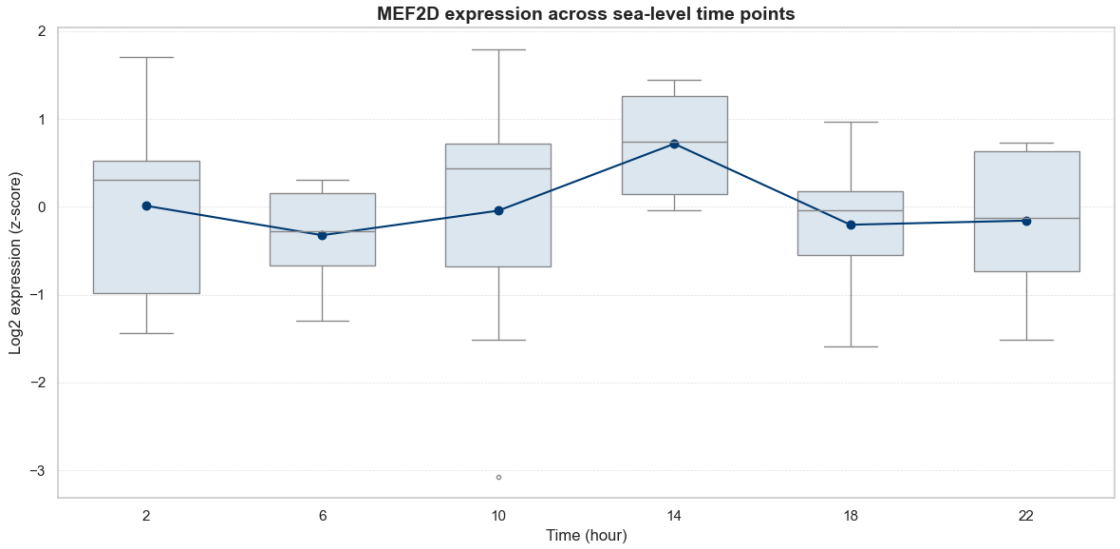

HDAC3 expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

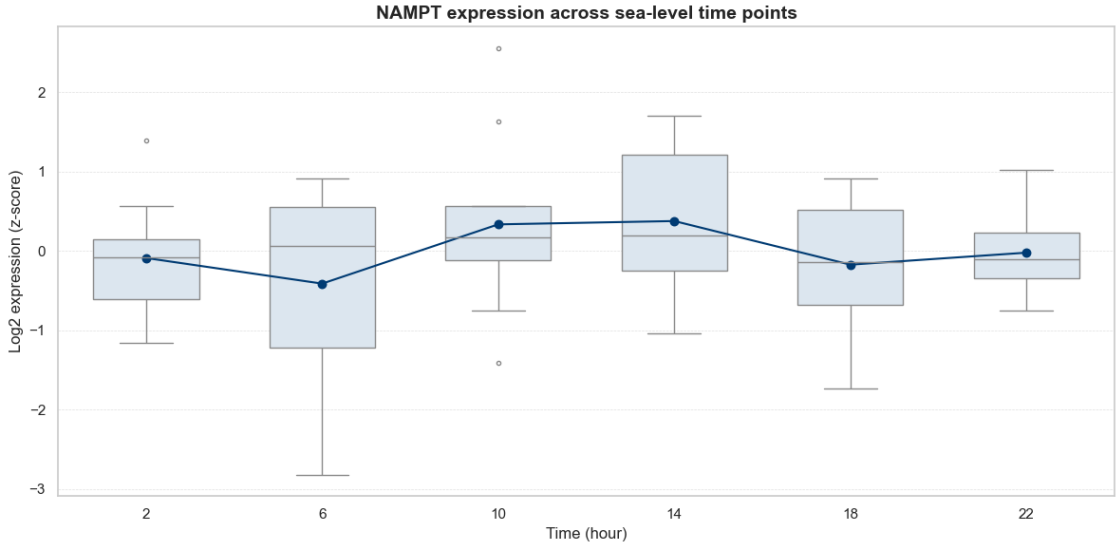

HELZ2 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

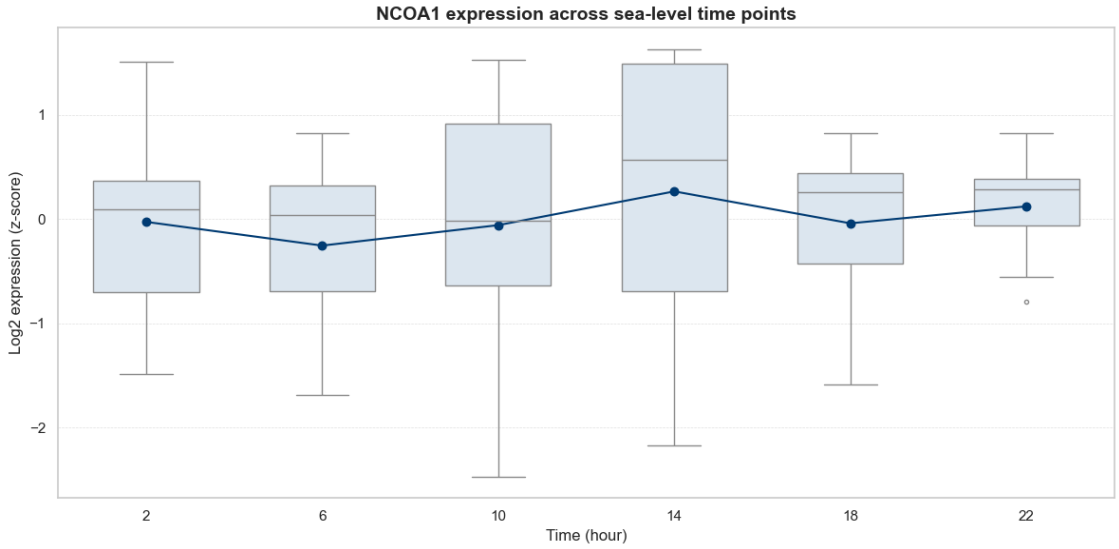

KMT2A expression across sea-level time points
Log2 expression (z-score)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
2
6
10
14
18
22
Time (hour)

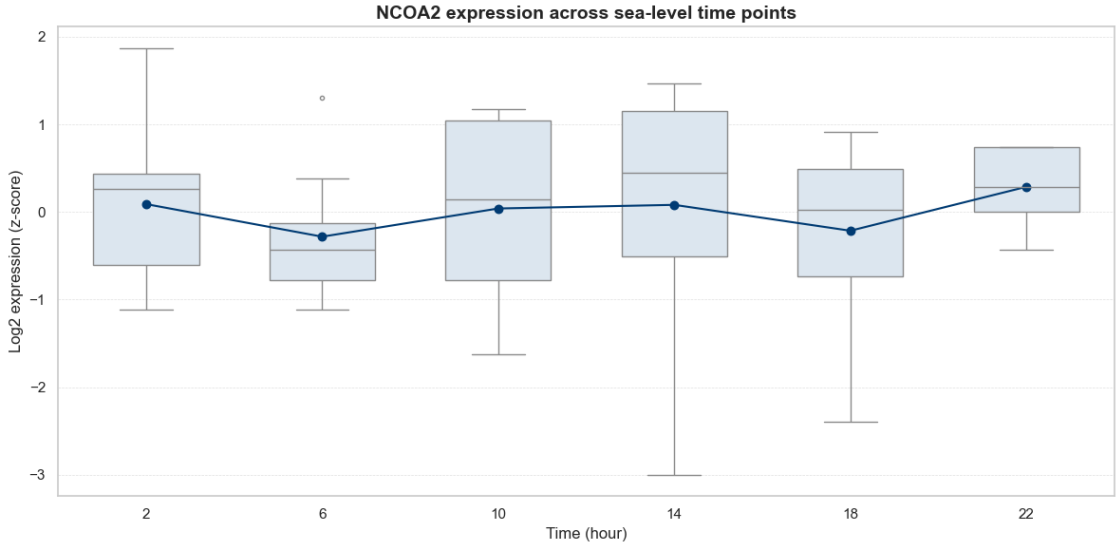

MED1 expression across sea-level time points
Log2 expression (z-score)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
2
6
10
14
18
22
Time (hour)

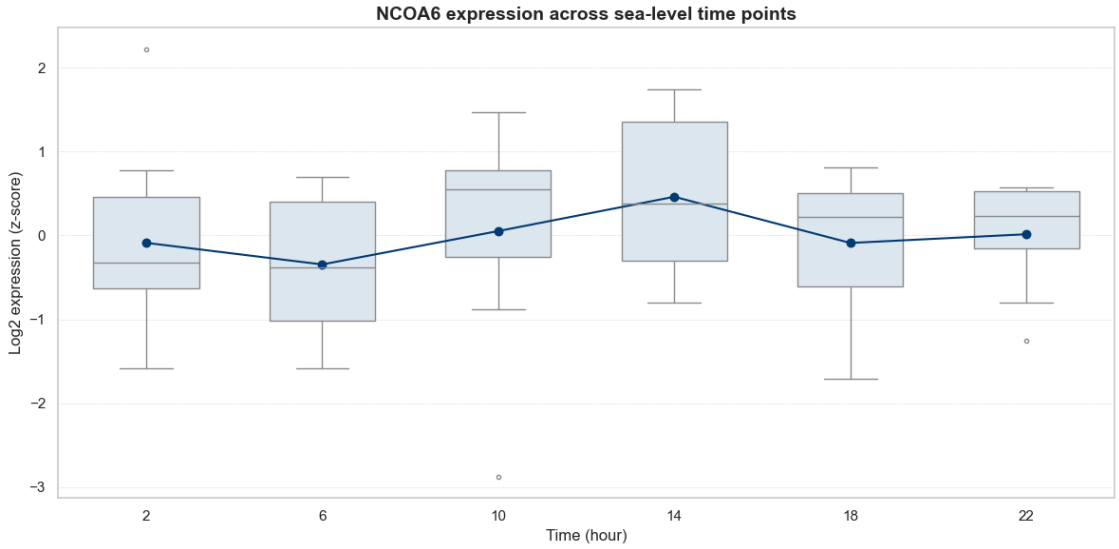

MEF2C expression across sea-level time points
Log2 expression (z-score)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
2
6
10
14
18
22
Time (hour)

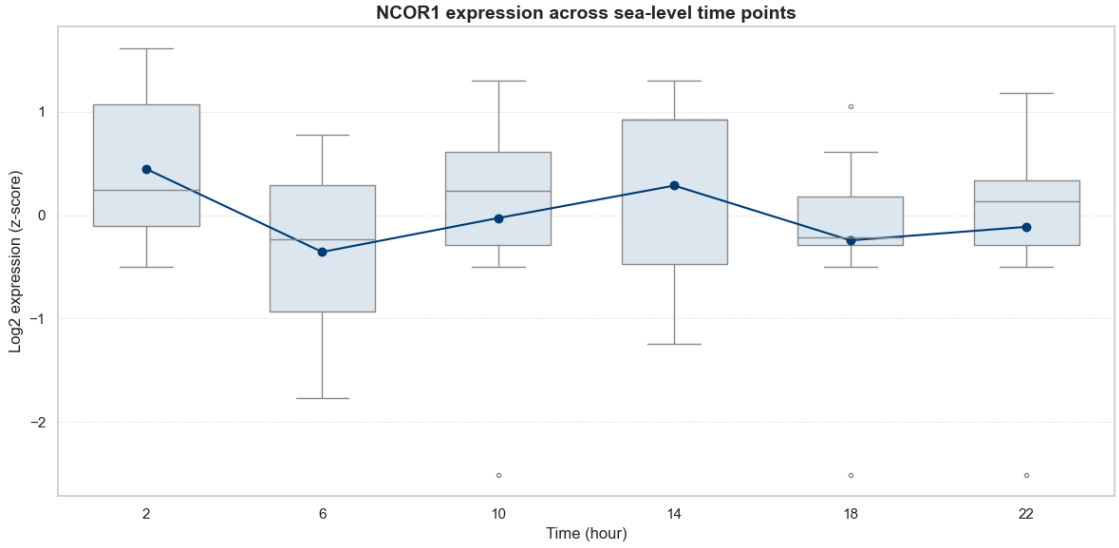

MEF2D expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

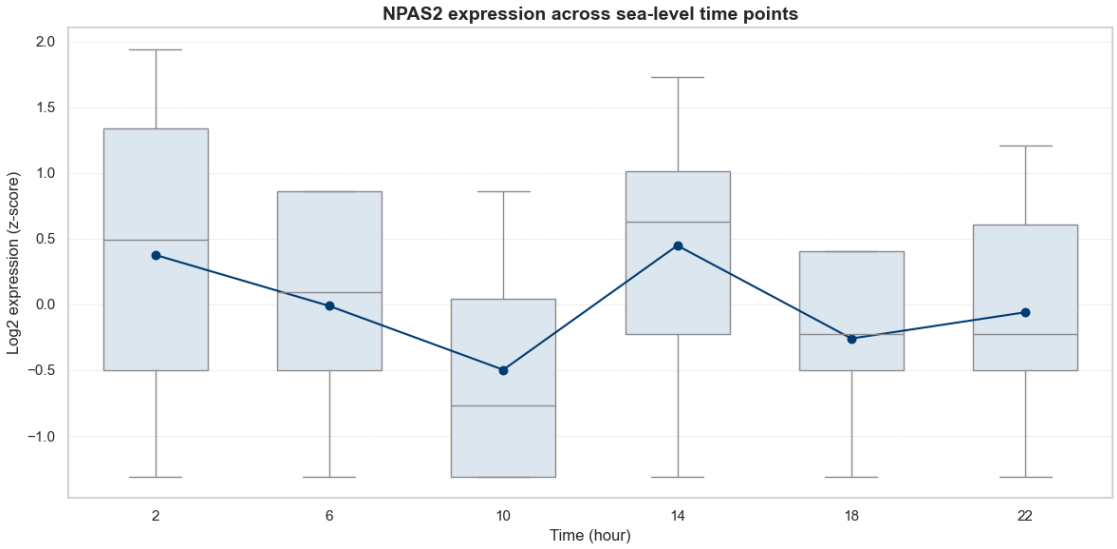

NAMPT expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

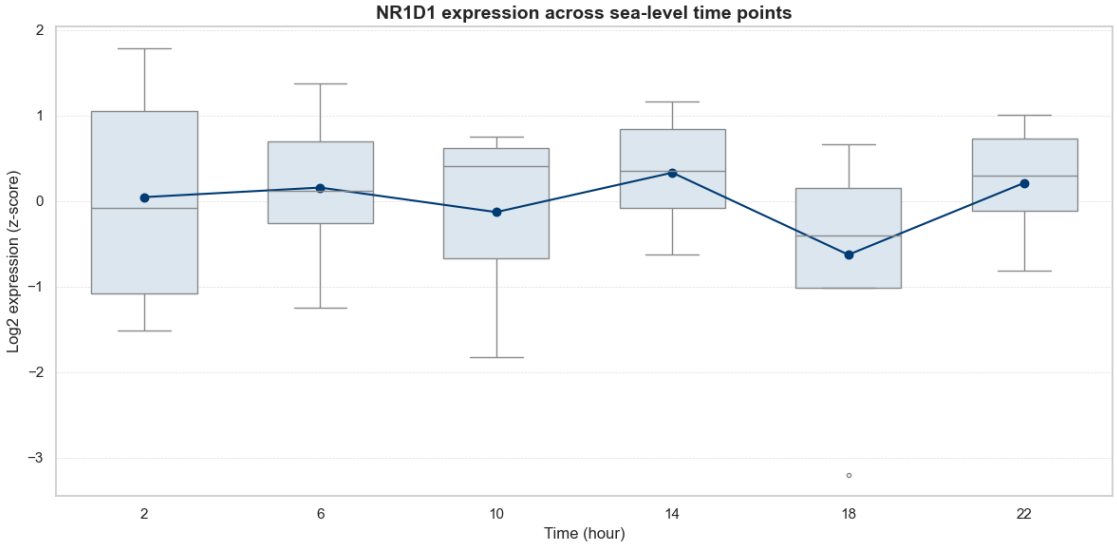

NCOA1 expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

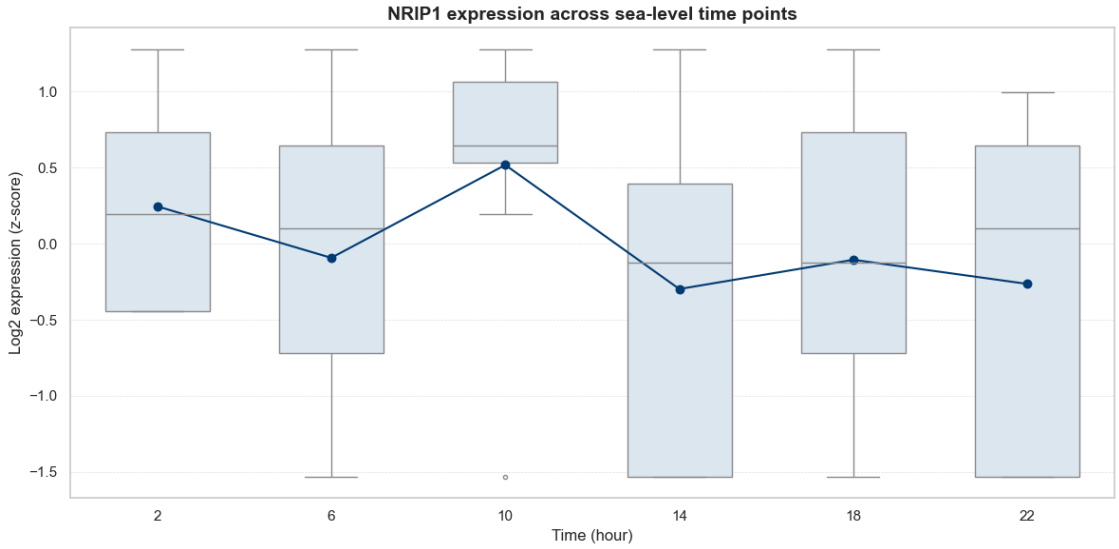

NCOA2 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

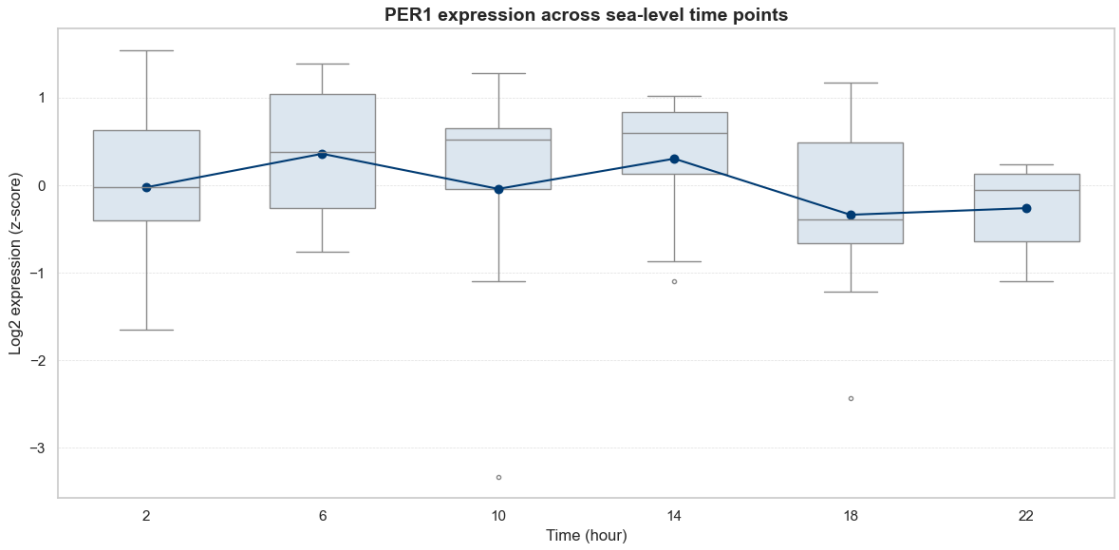

NCOA6 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

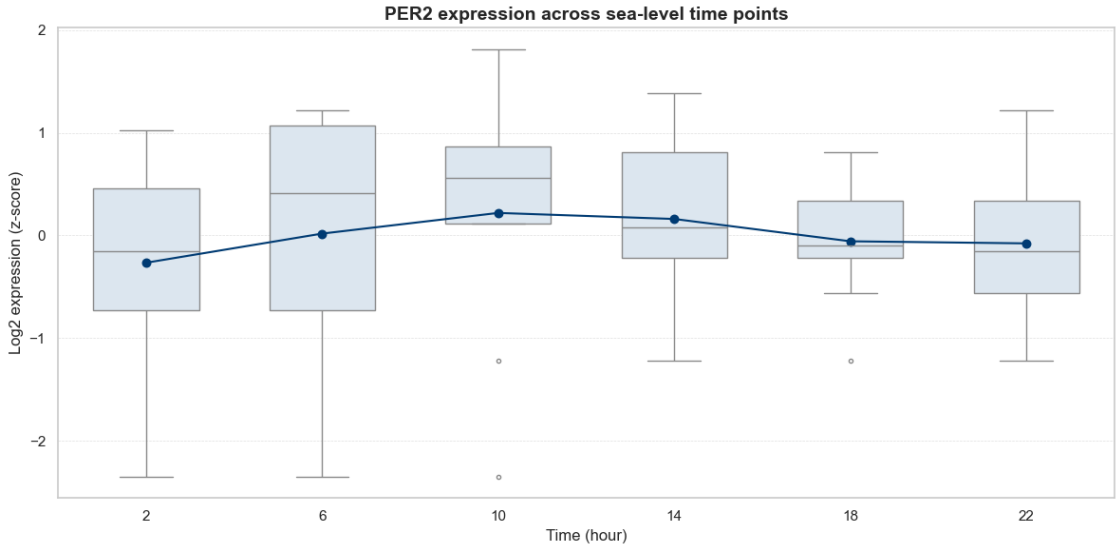

NCOR1 expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

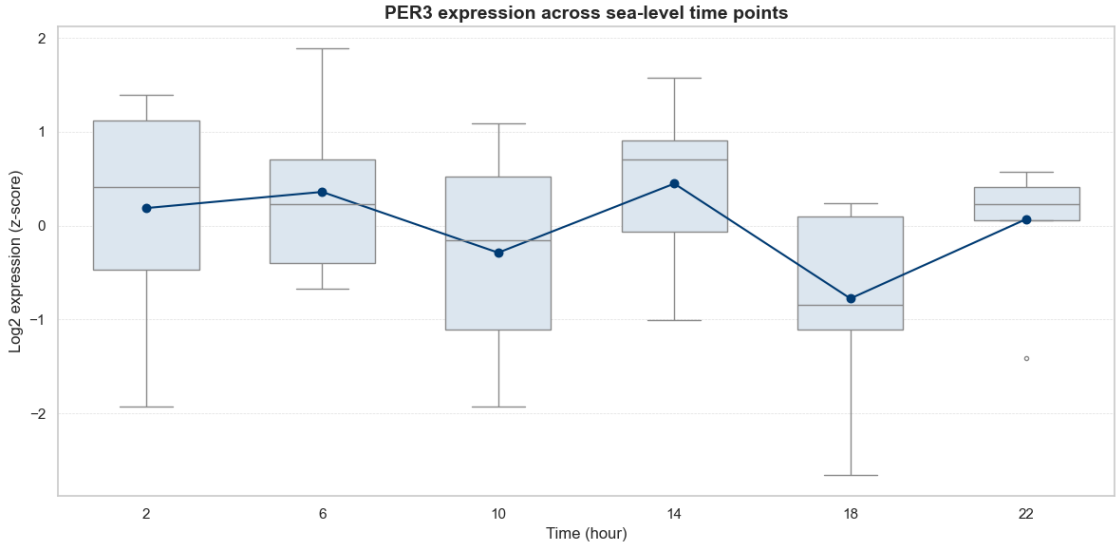

NPAS2 expression across sea-level time points
Log2 expression (z-score)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
2
6
10
14
18
22
Time (hour)

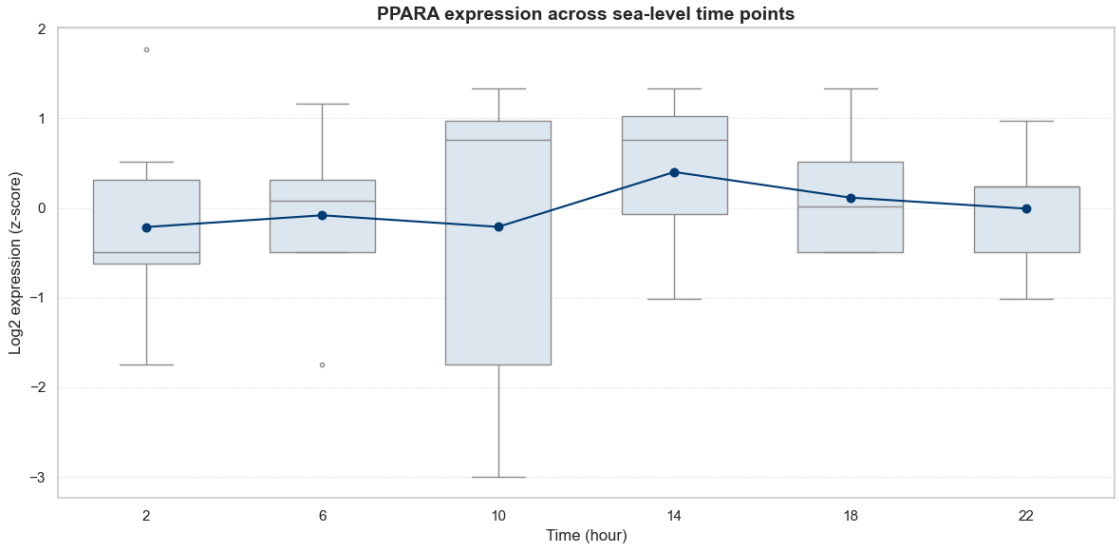

NR1D1 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

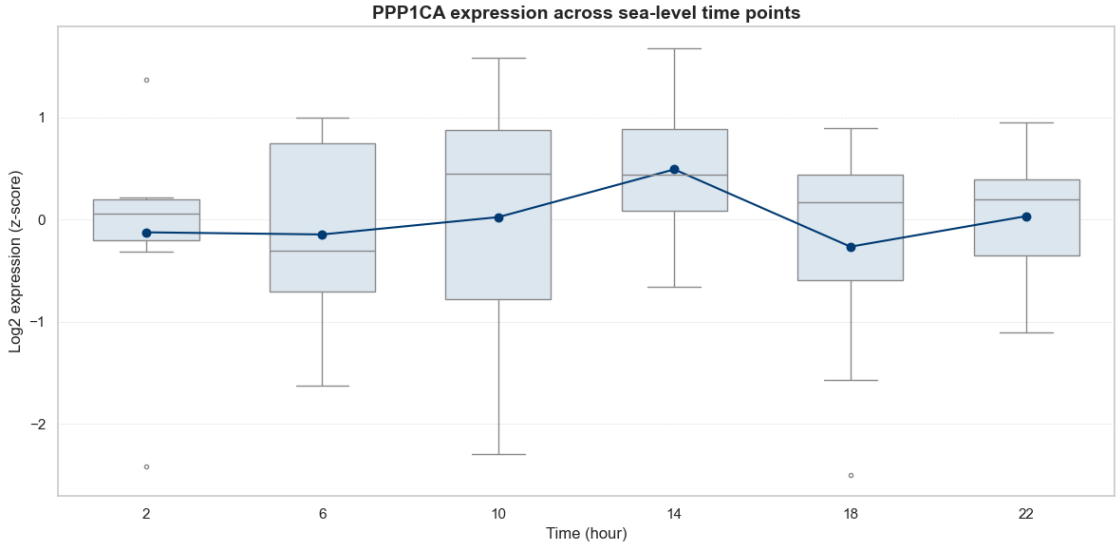

NRIP1 expression across sea-level time points
Log2 expression (z-score)
1.0
0.5
0.0
−0.5
−1.0
−1.5
2
6
10
14
18
22
Time (hour)

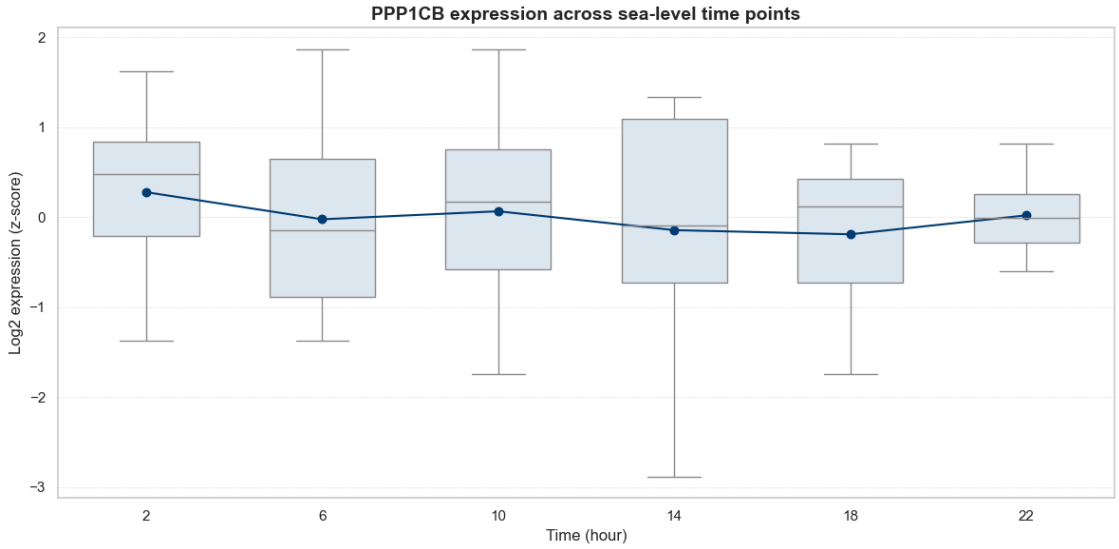

PER1 expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

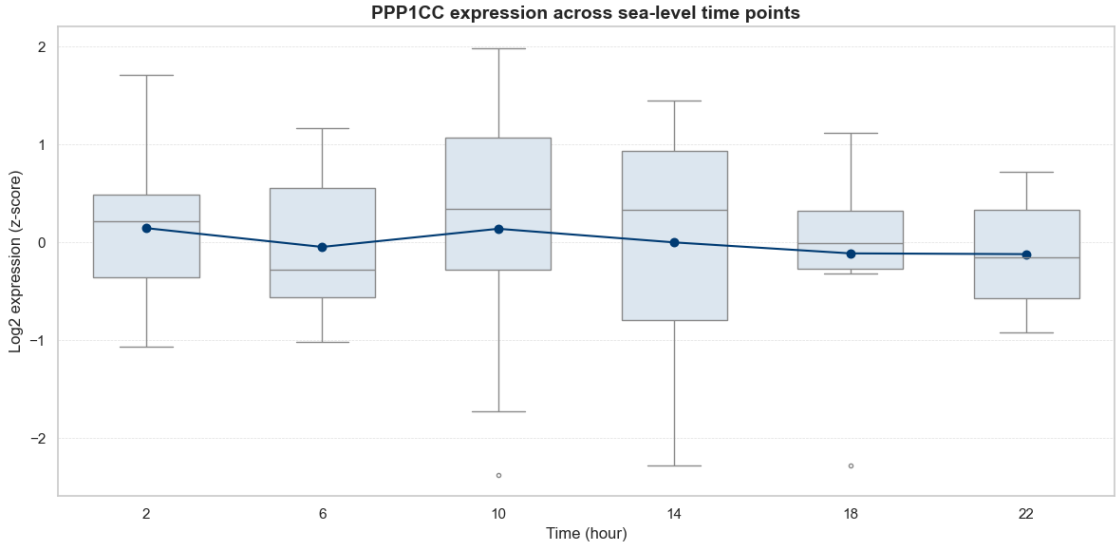

PER2 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

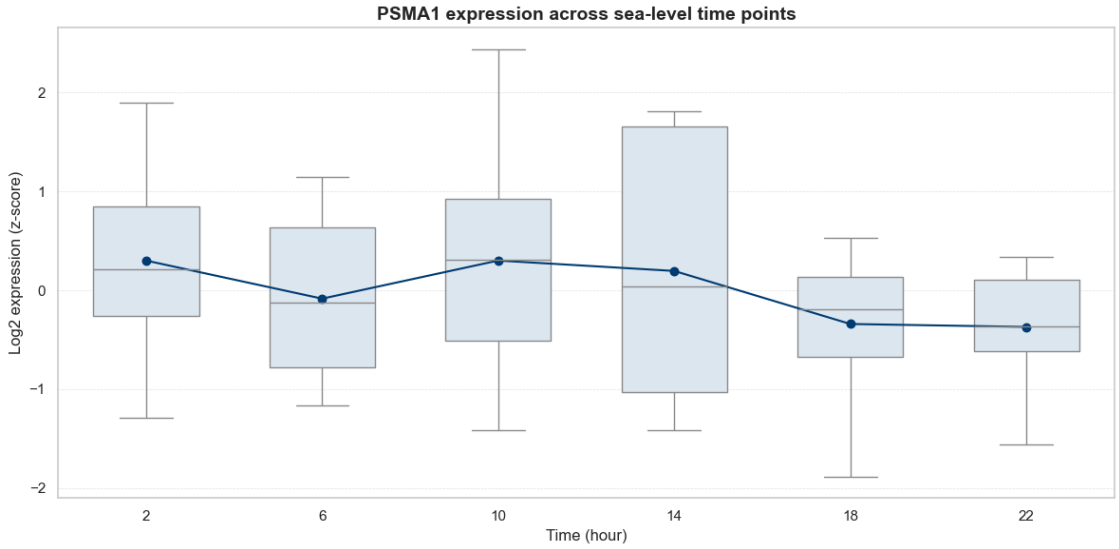

PER3 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

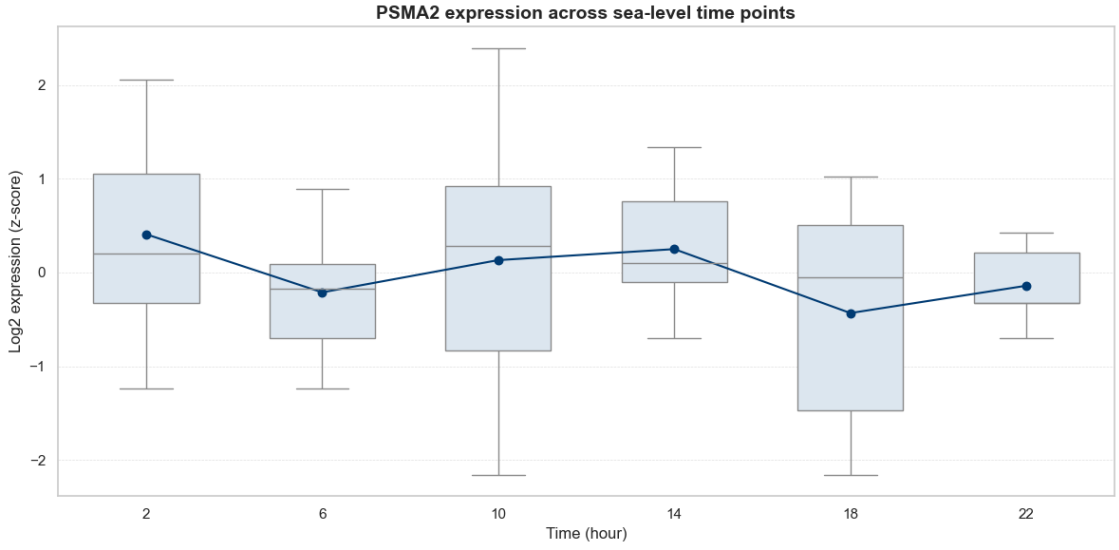

PPARA expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

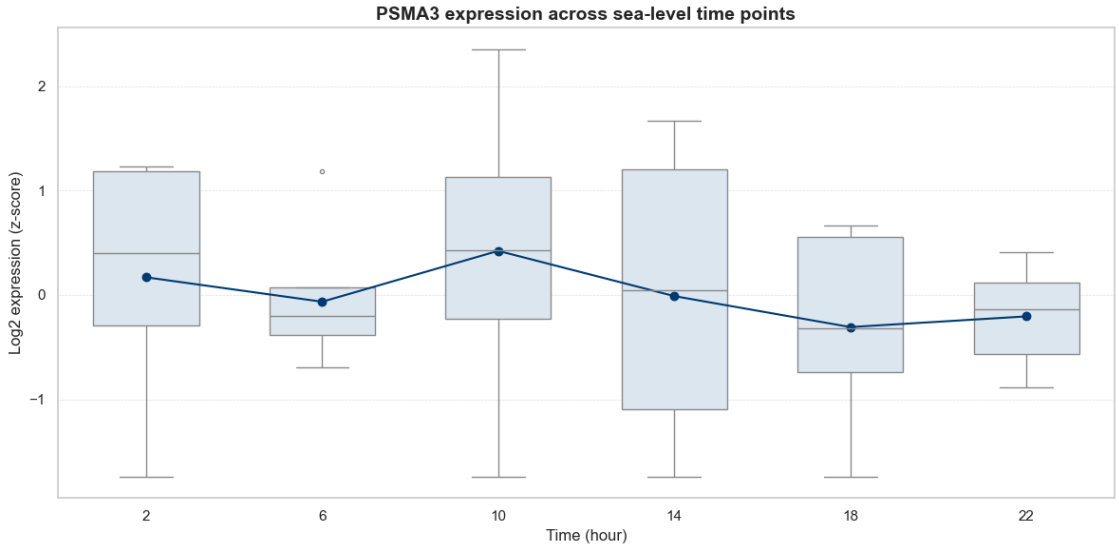

PPP1CA expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

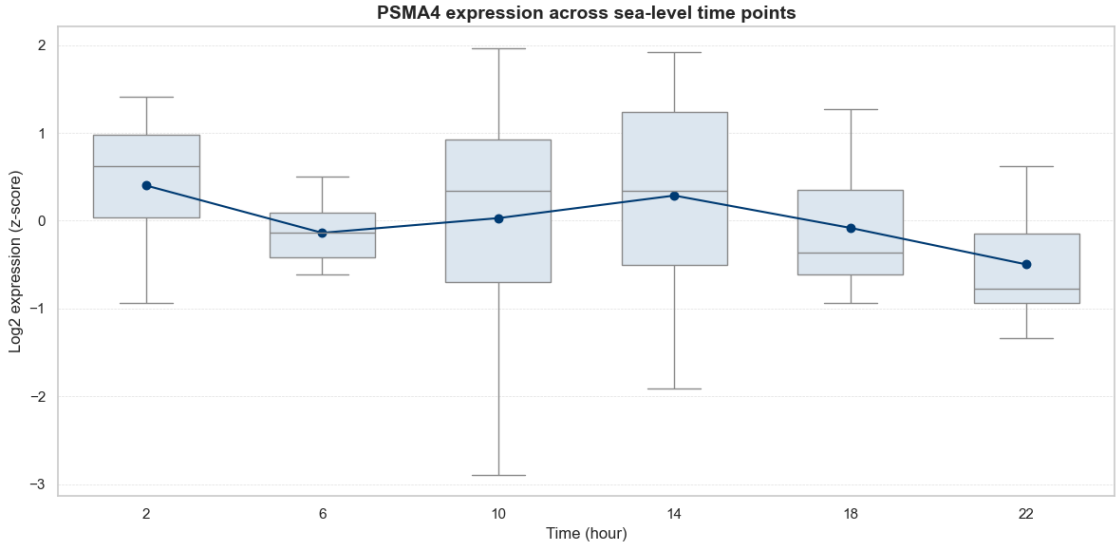

PPP1CB expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

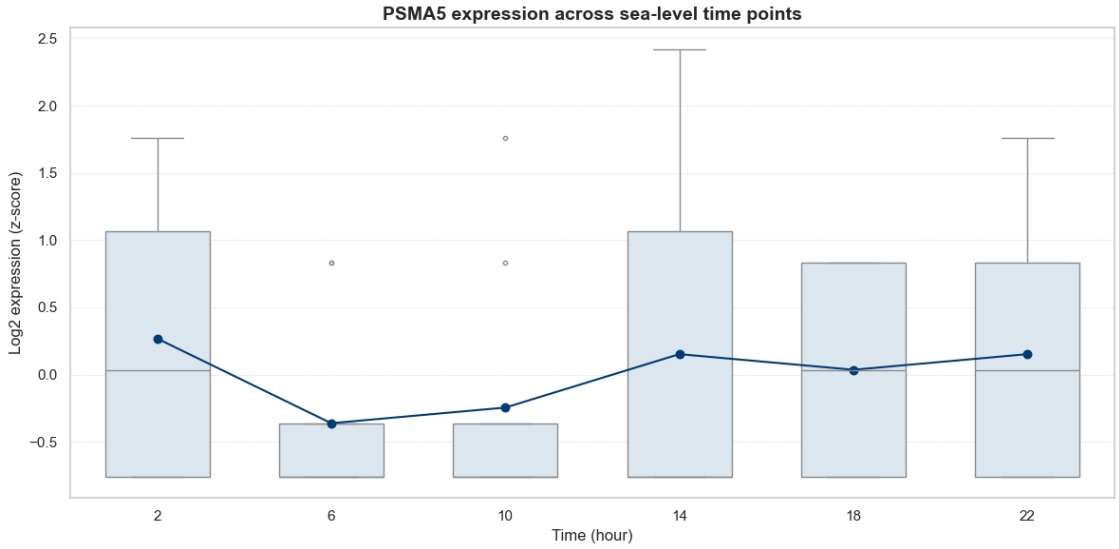

PPP1CC expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

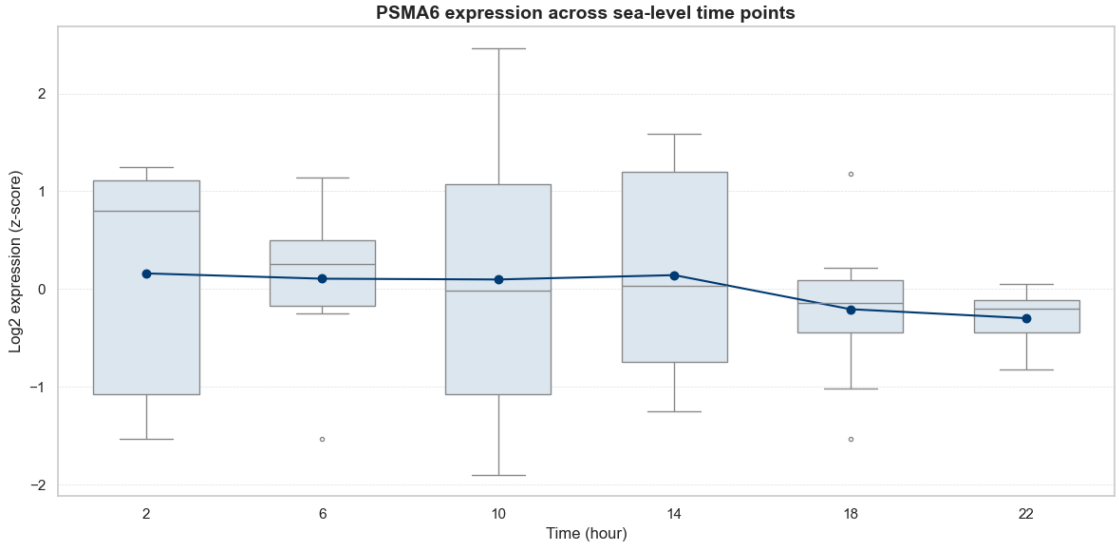

PSMA1 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

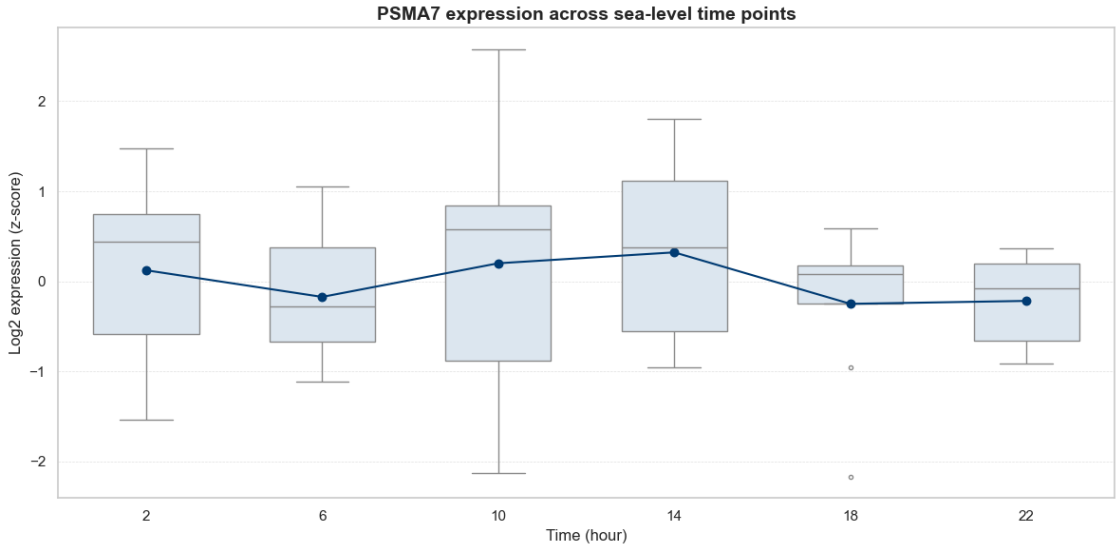

PSMA2 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

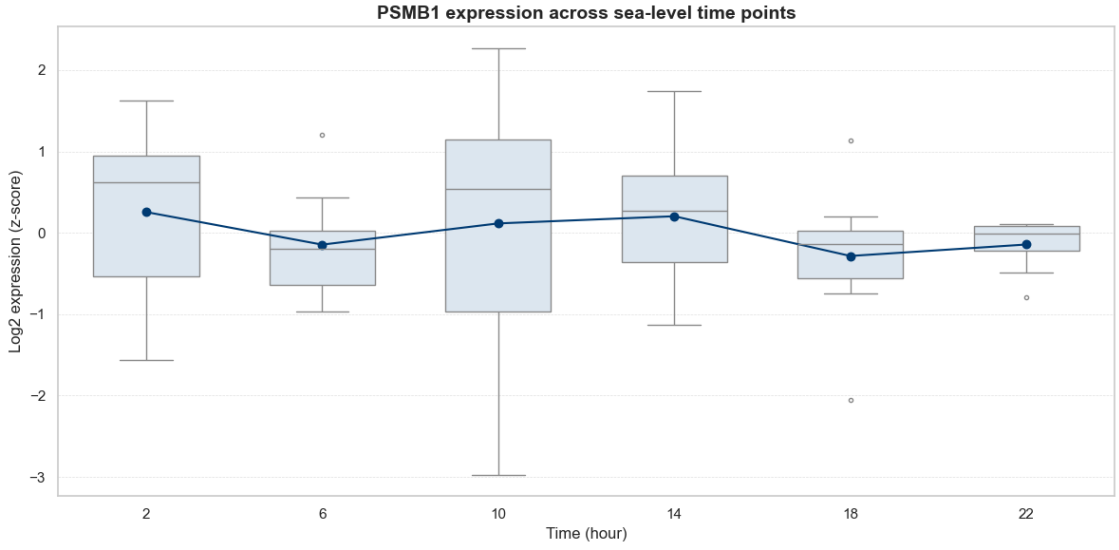

PSMA3 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
2
6
10
14
18
22
Time (hour)

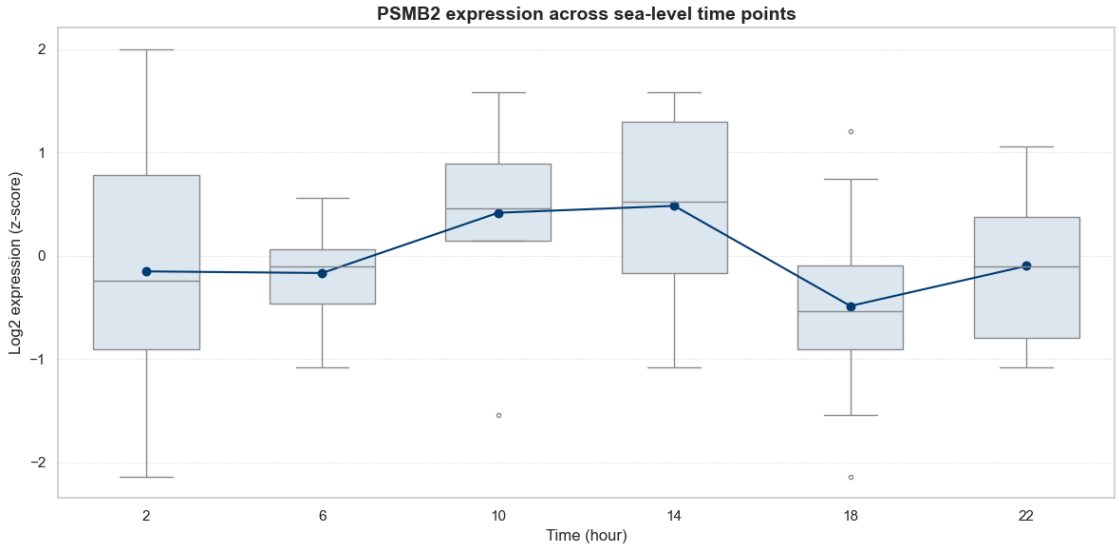

PSMA4 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

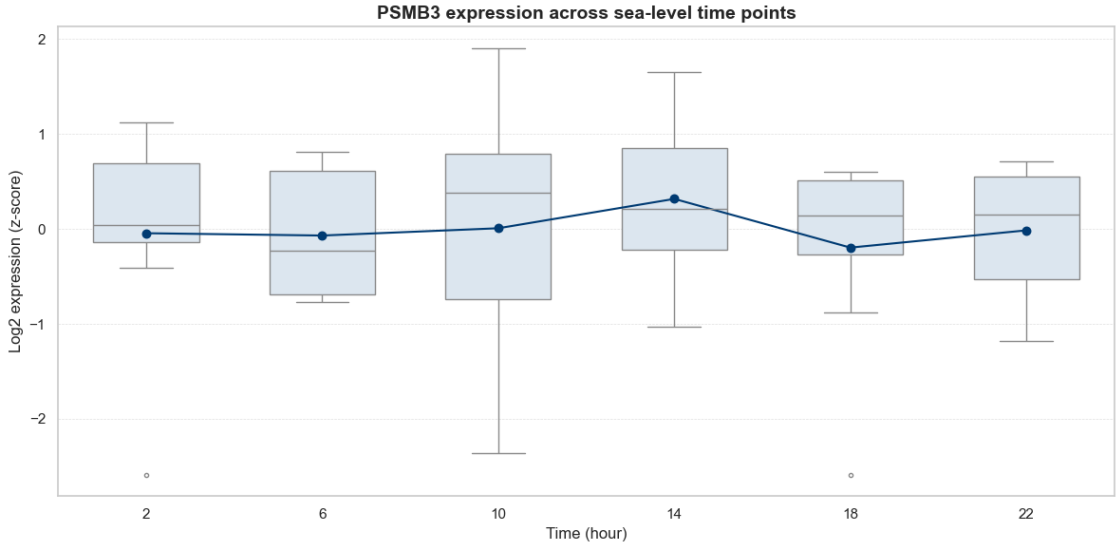

PSMA5 expression across sea-level time points
Log2 expression (z-score)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
2
6
10
14
18
22
Time (hour)

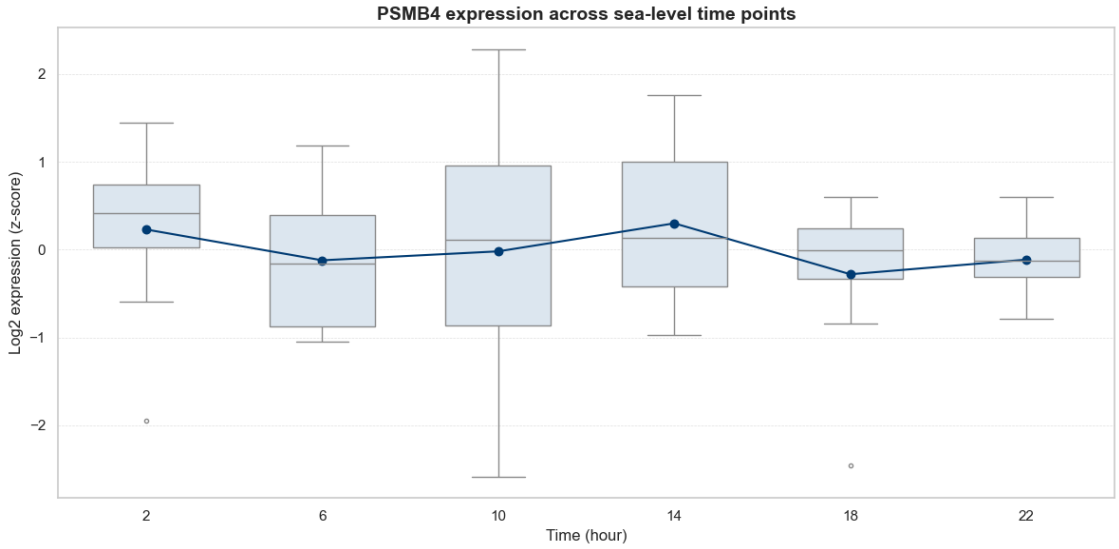

PSMA6 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

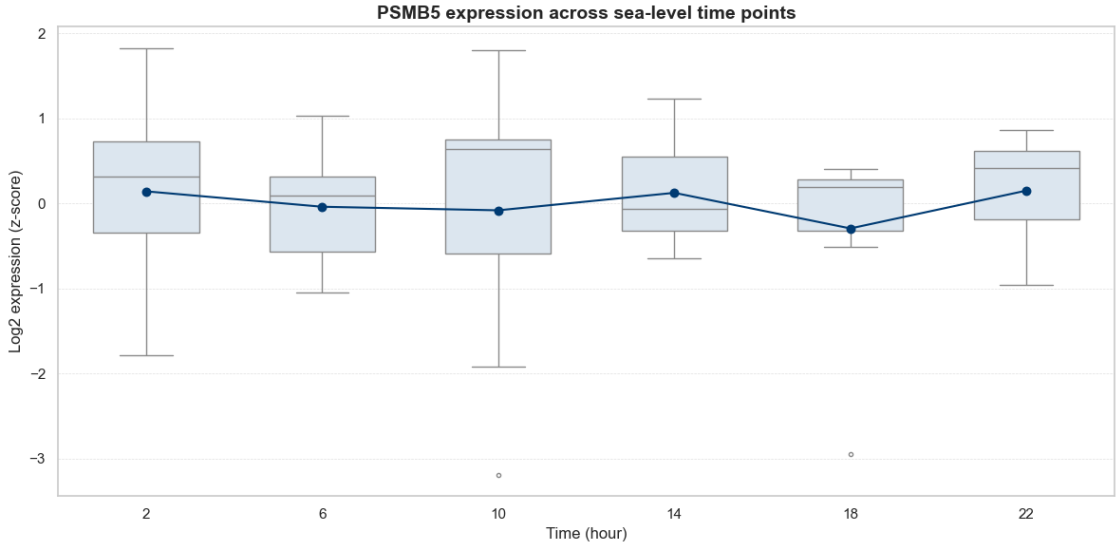

PSMA7 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

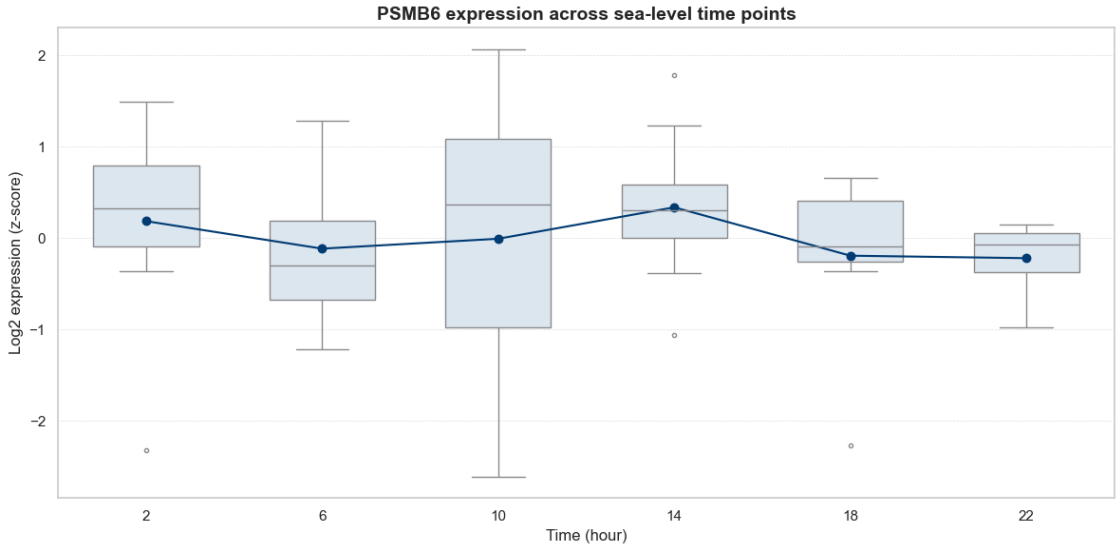

PSMB1 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

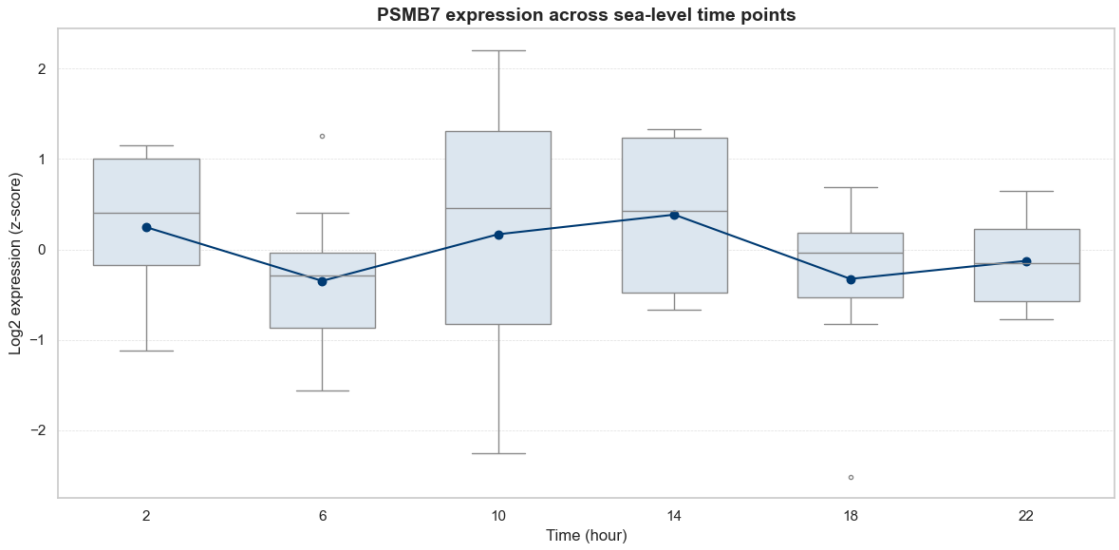

PSMB2 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

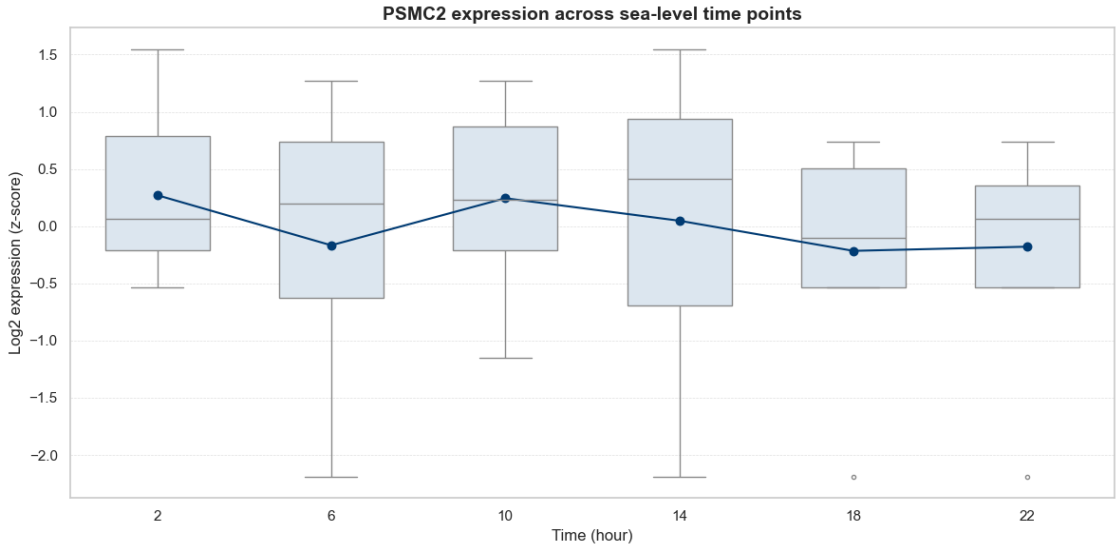

PSMB3 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

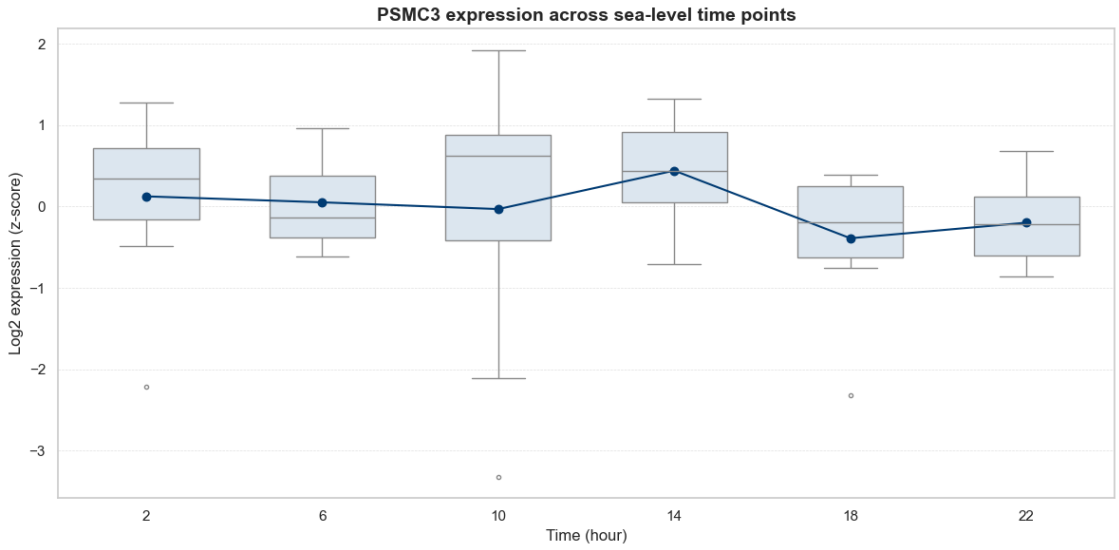

PSMB4 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

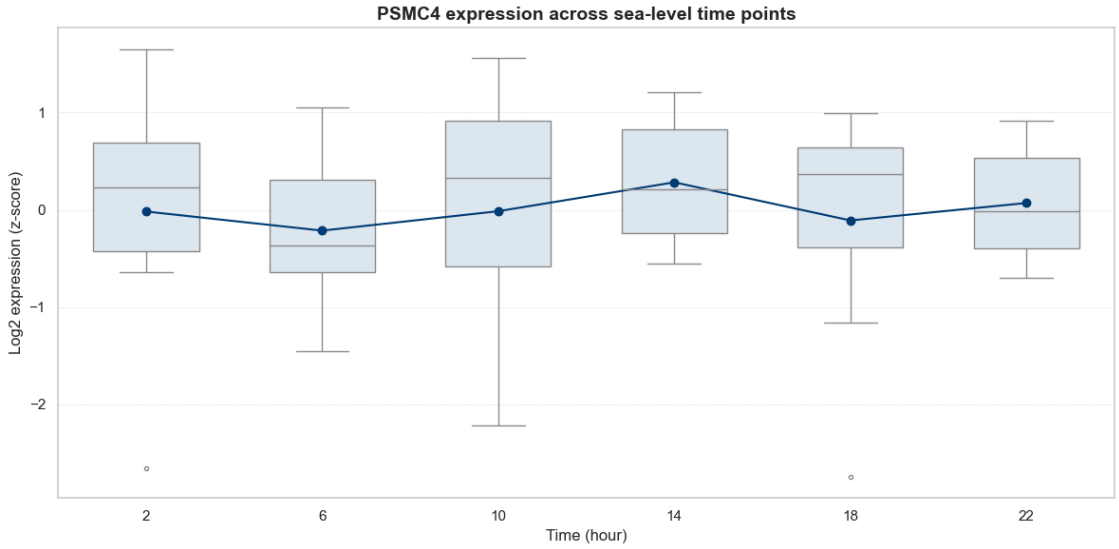

PSMB5 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

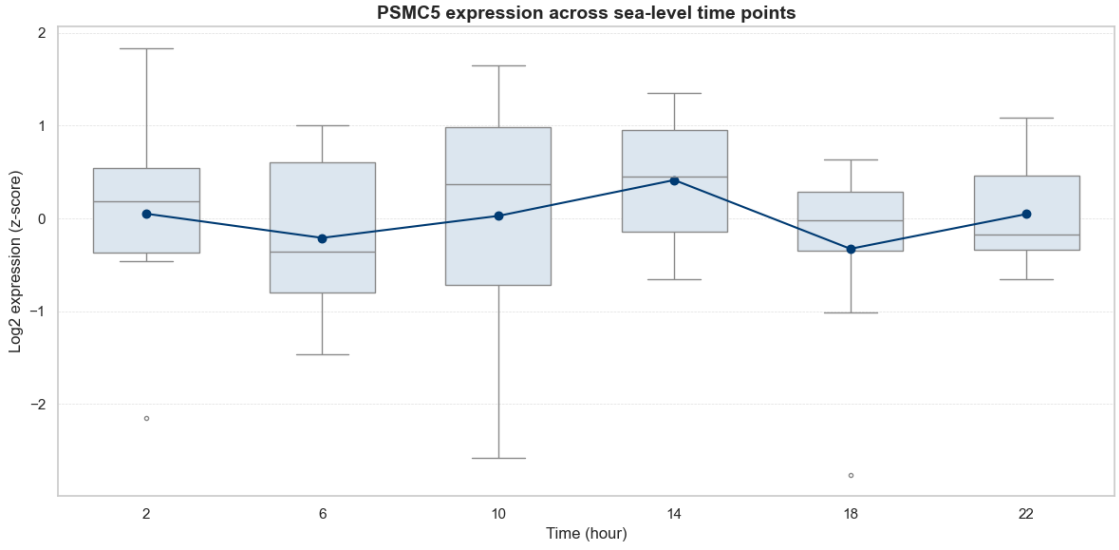

PSMB6 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

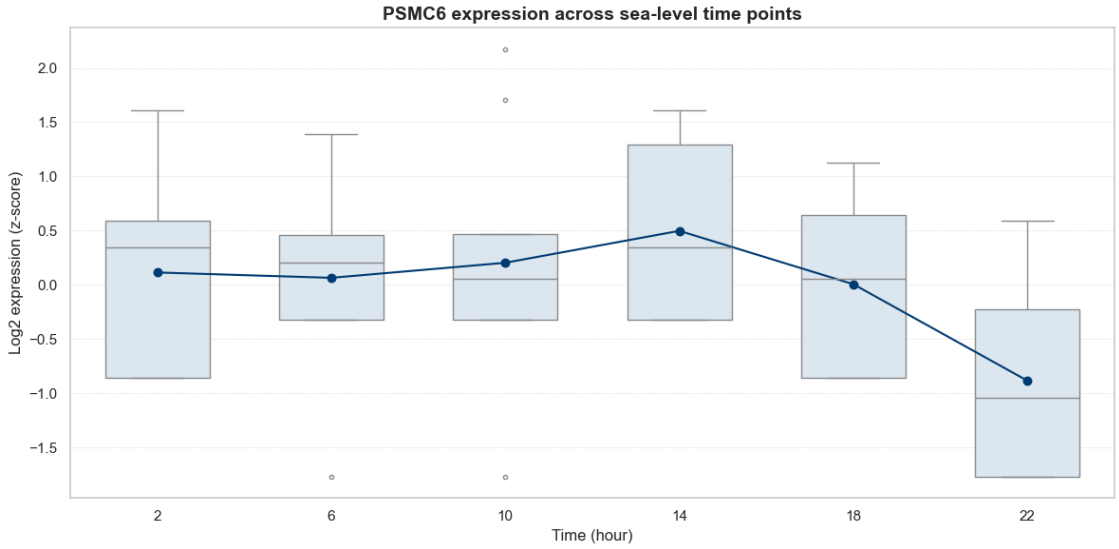

PSMB7 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

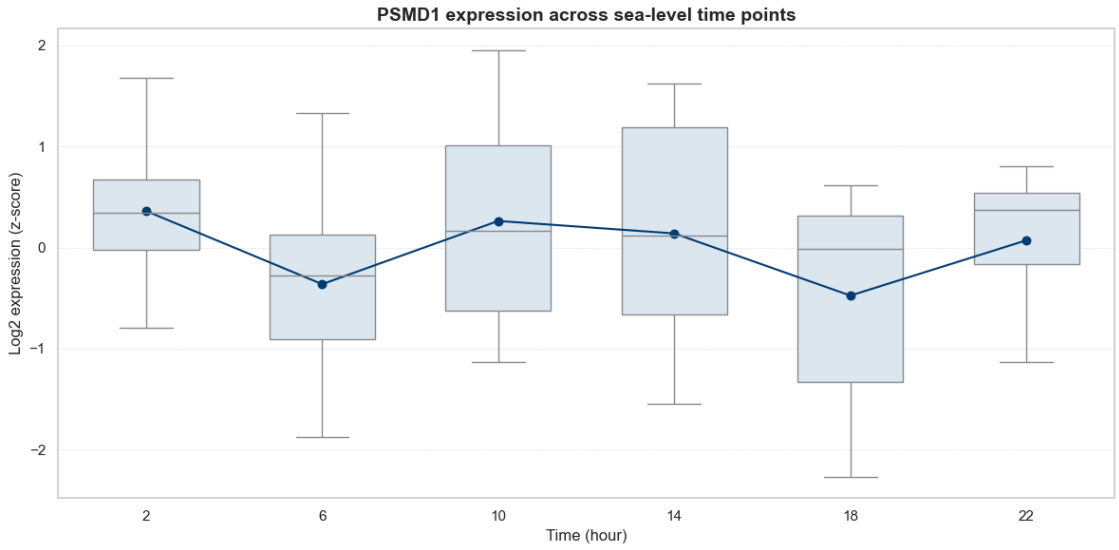

PSMC2 expression across sea-level time points
Log2 expression (z-score)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
2
6
10
14
18
22
Time (hour)

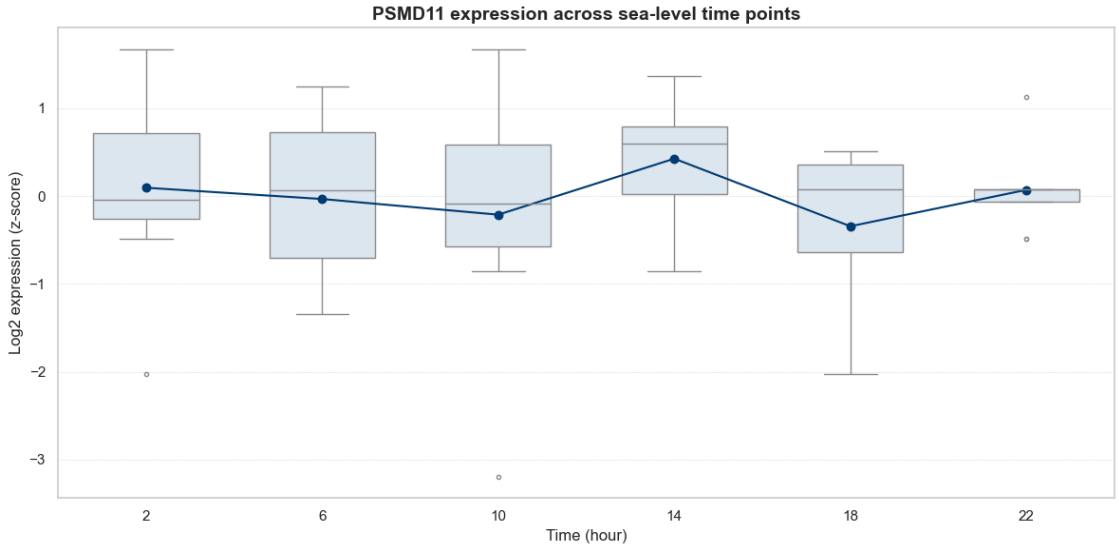

PSMC3 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

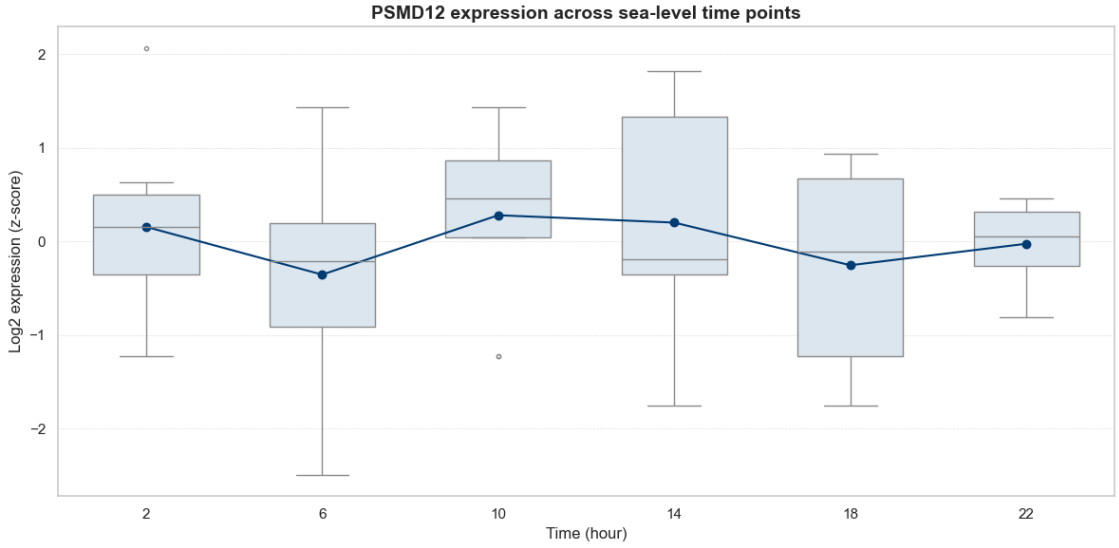

PSMC4 expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

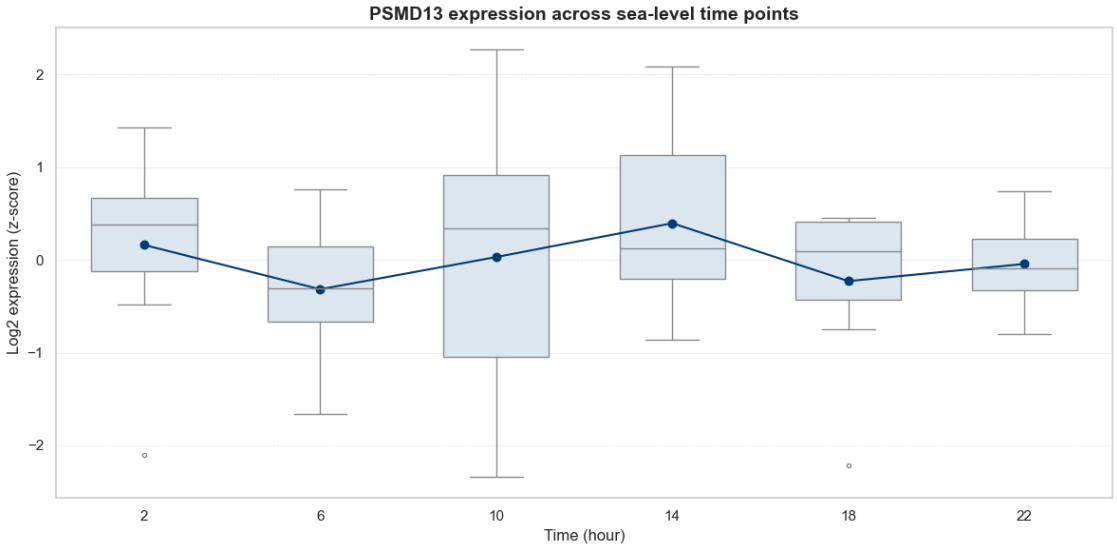

PSMC5 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

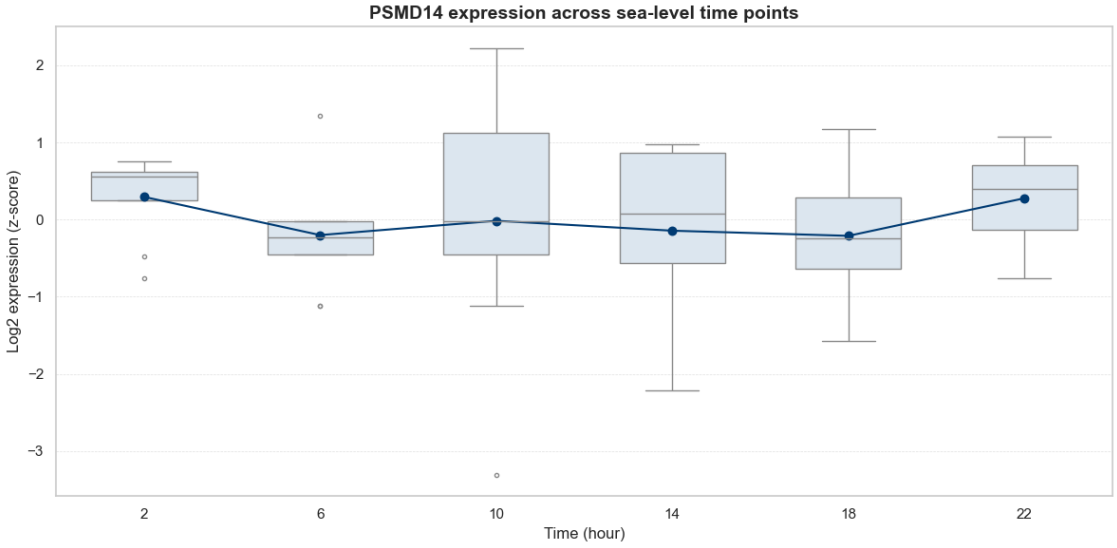

PSMC6 expression across sea-level time points
Log2 expression (z-score)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
2
6
10
14
18
22
Time (hour)

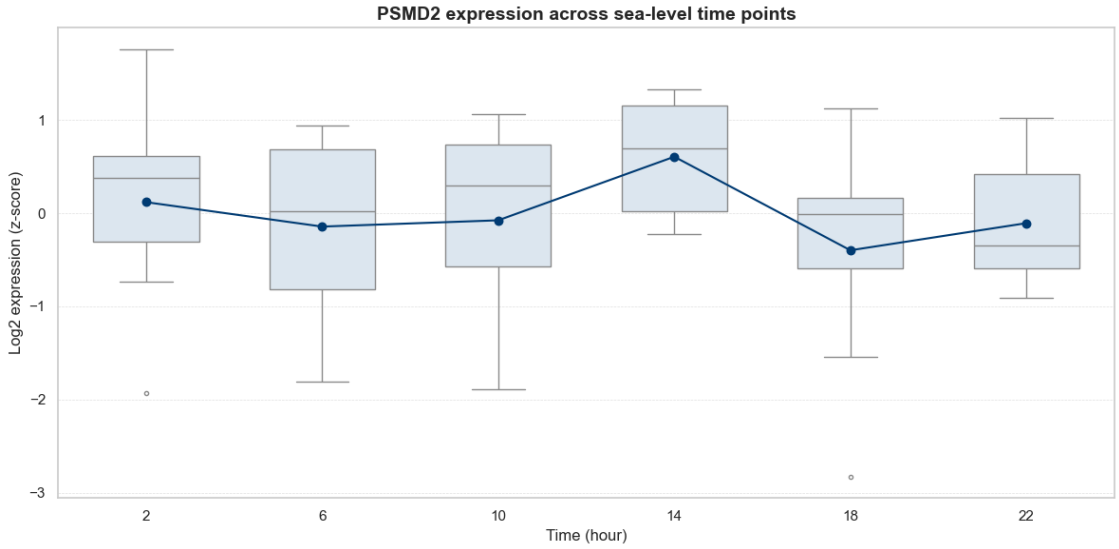

PSMD1 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

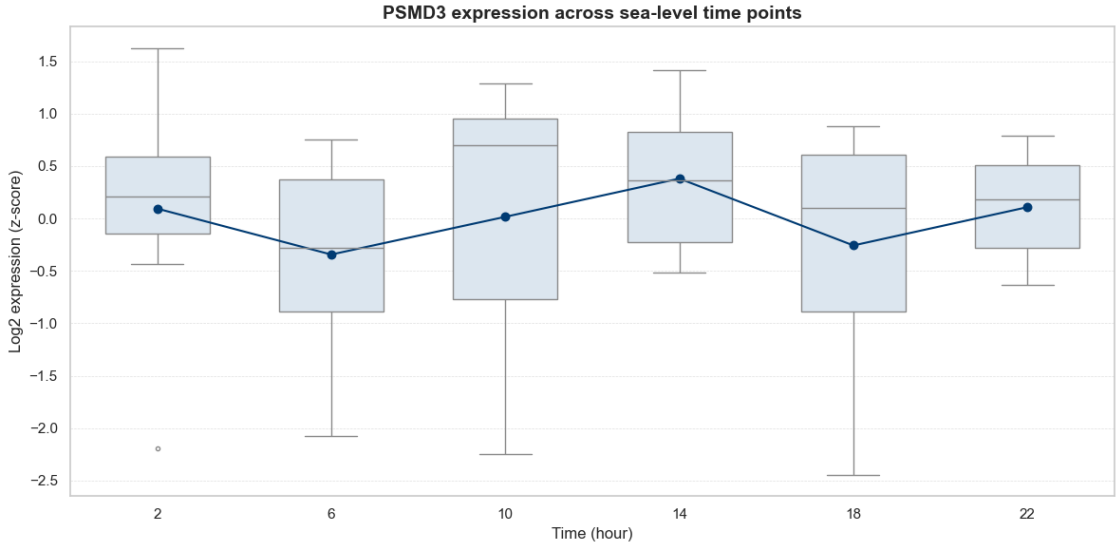

PSMD11 expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

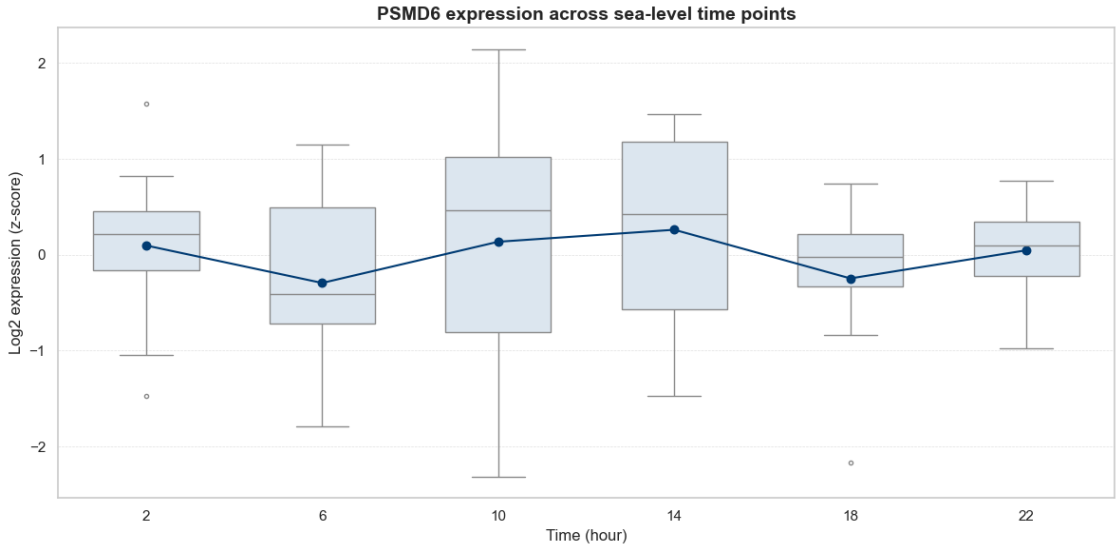

PSMD12 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

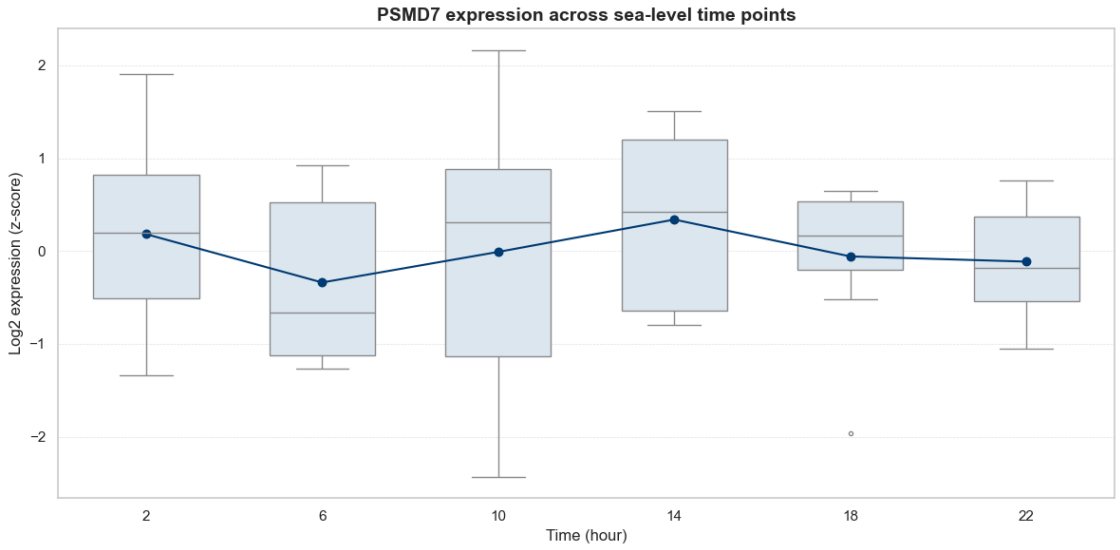

PSMD13 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

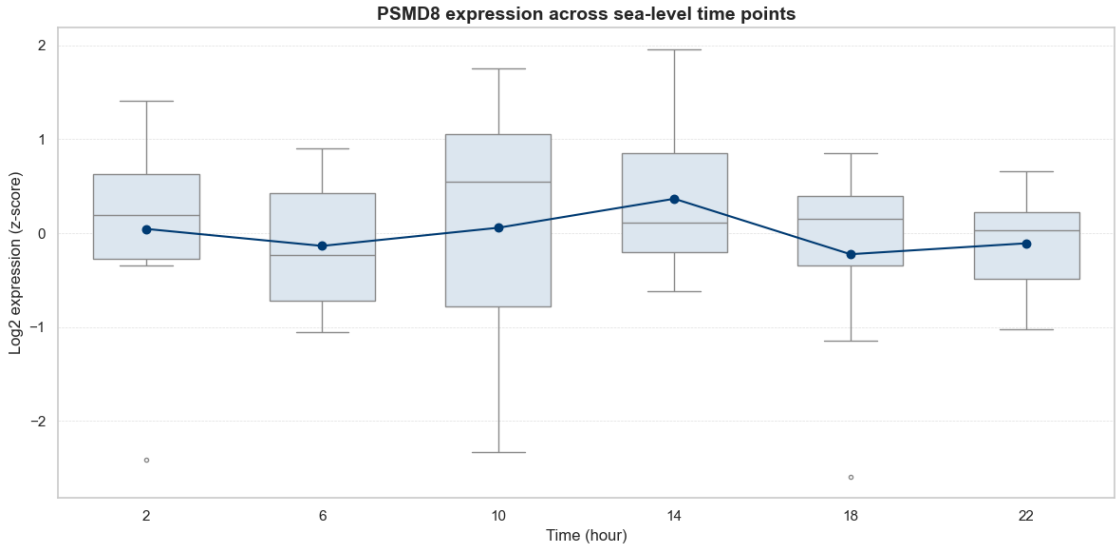

PSMD14 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

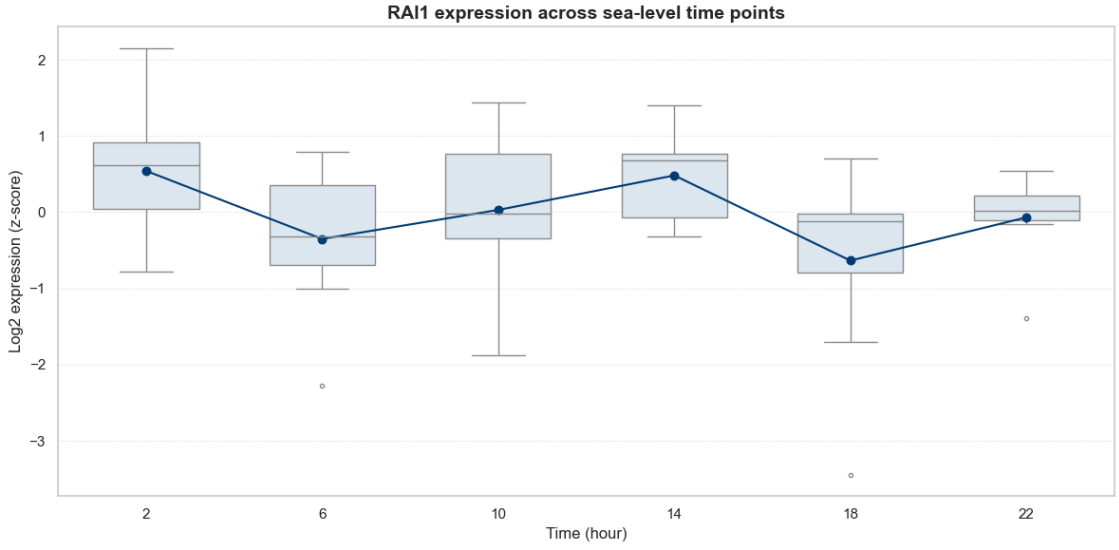

PSMD2 expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

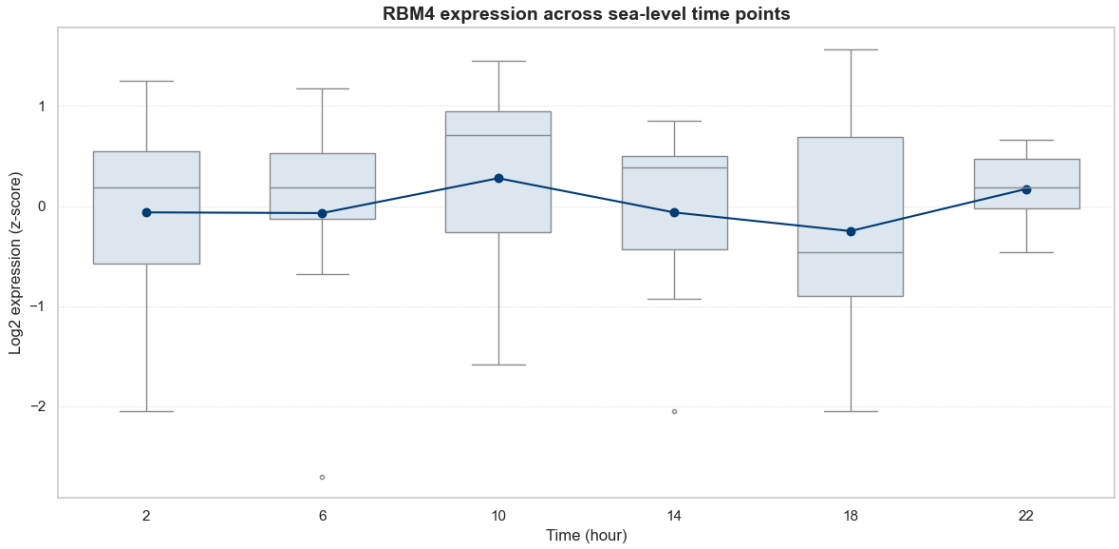

PSMD3 expression across sea-level time points
Log2 expression (z-score)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
−2.5
2
6
10
14
18
22
Time (hour)

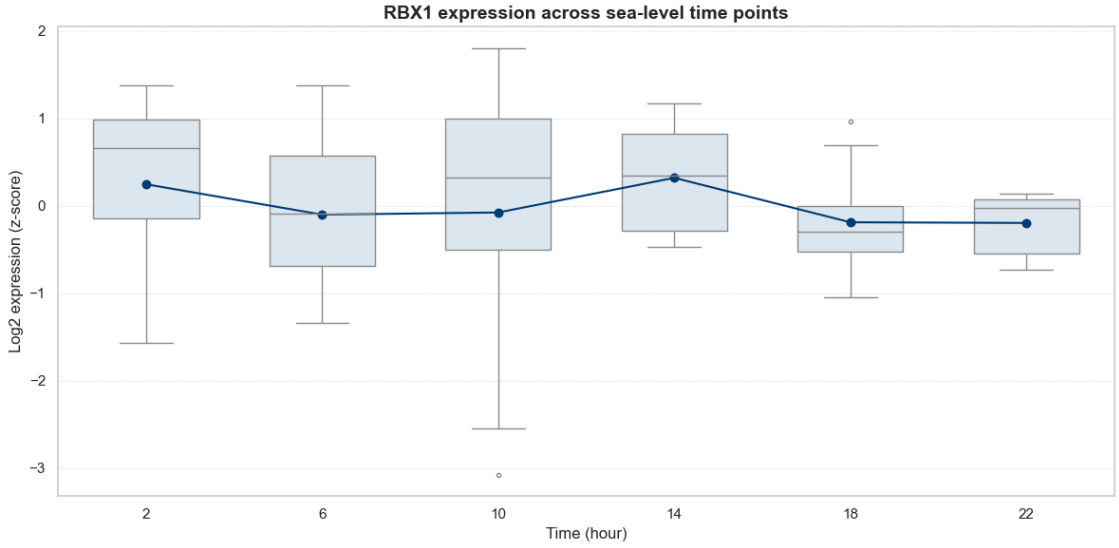

PSMD6 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

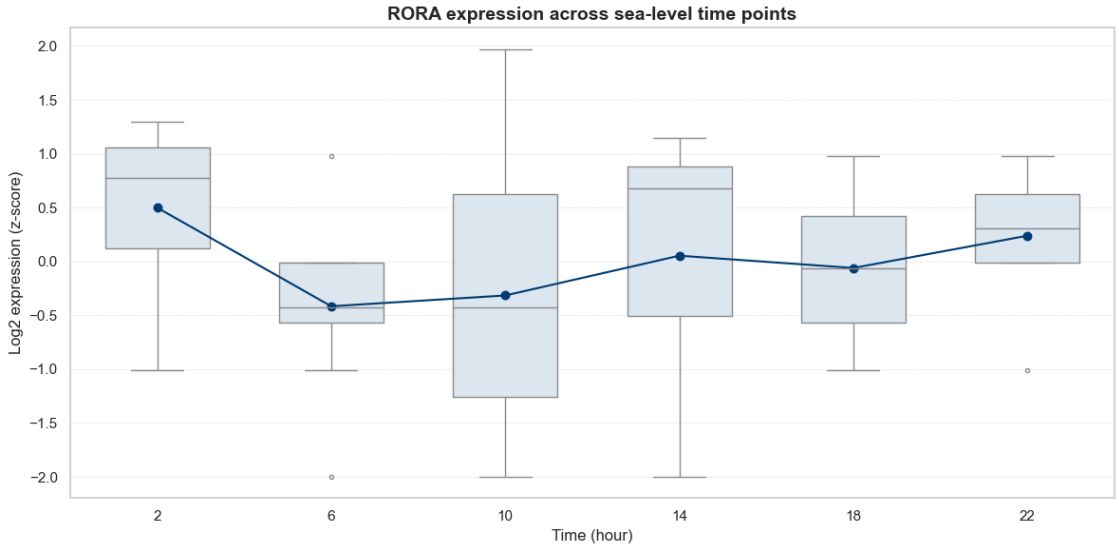

PSMD7 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

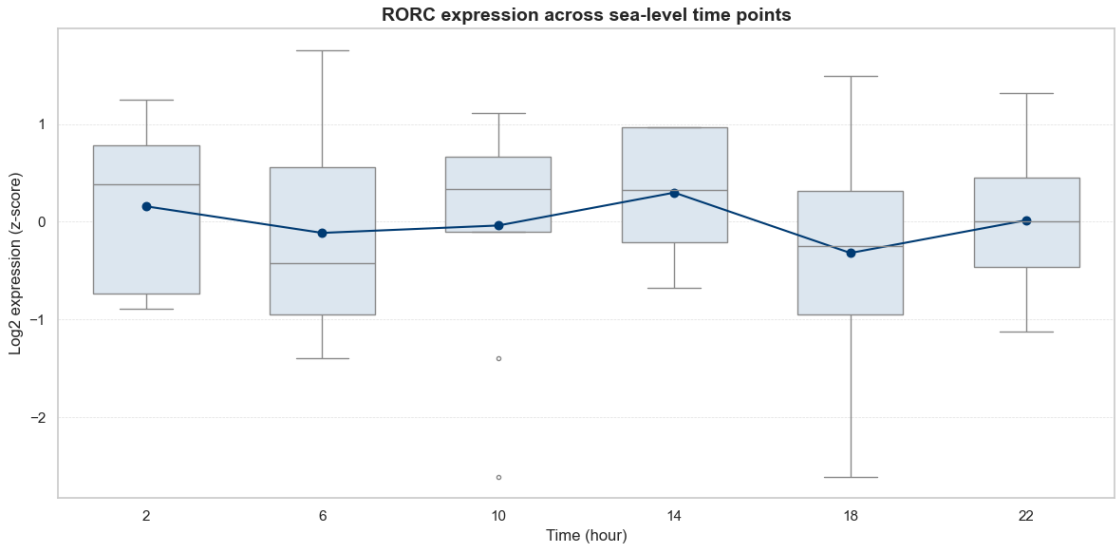

PSMD8 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

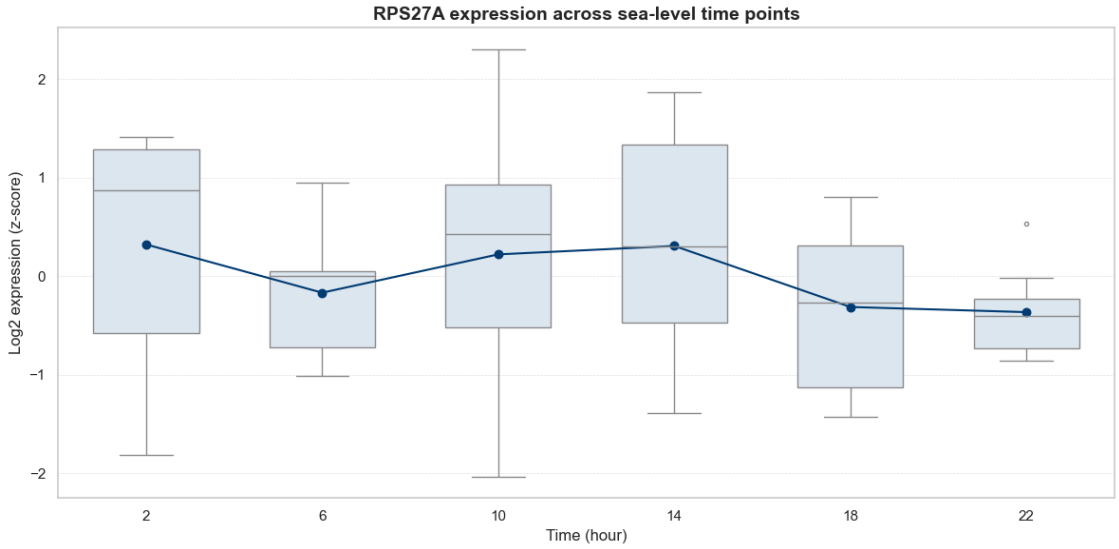

RAI1 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

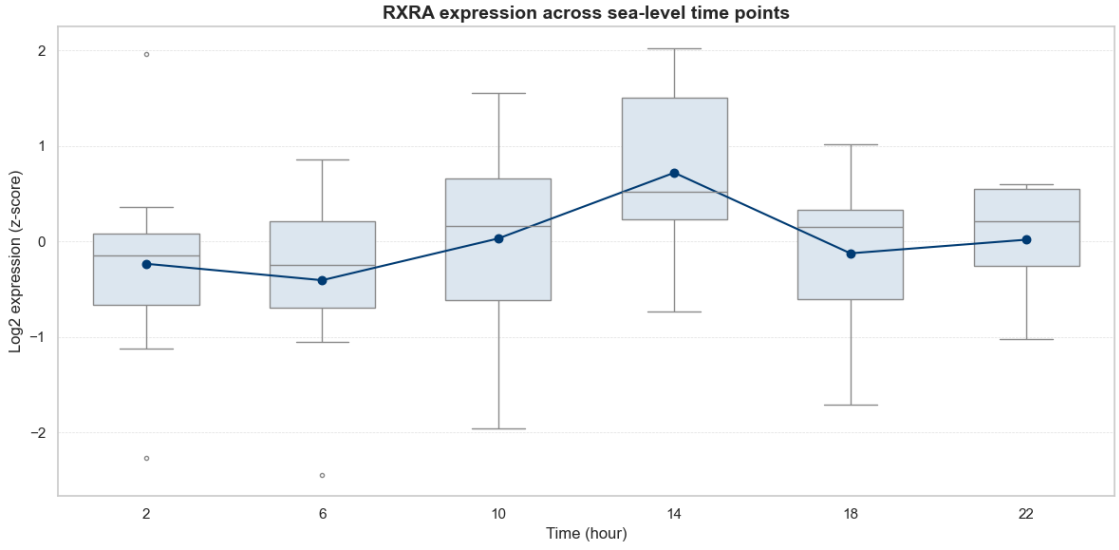

RBM4 expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

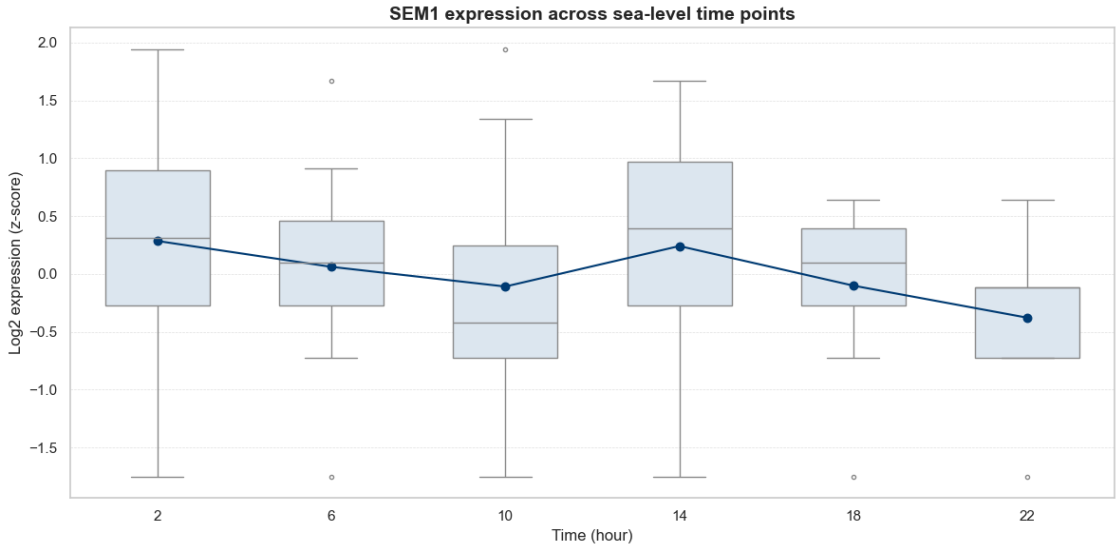

RBX1 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)

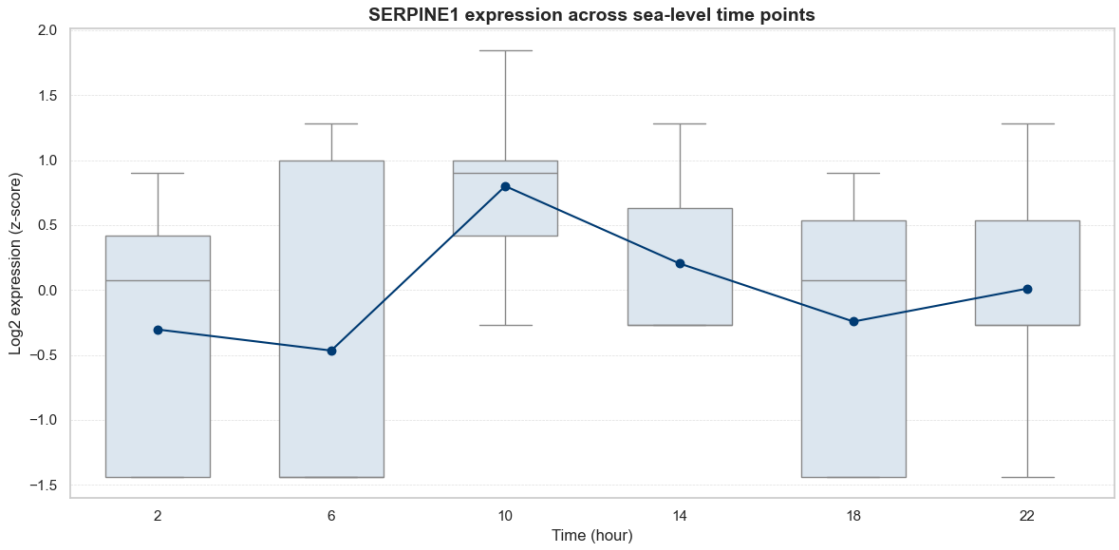

RORA expression across sea-level time points
Log2 expression (z-score)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−2.0
2
6
10
14
18
22
Time (hour)

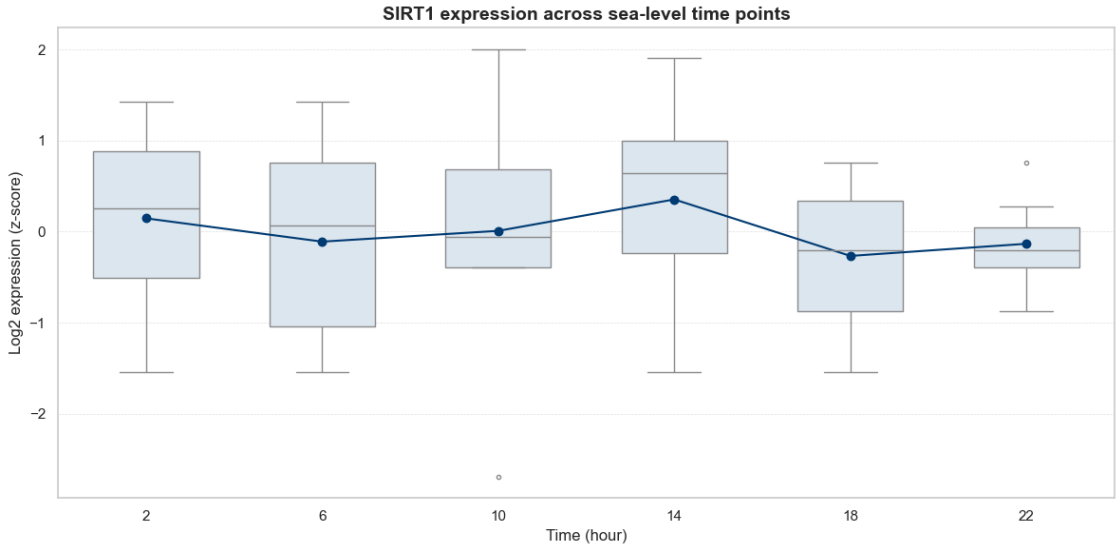

RORC expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

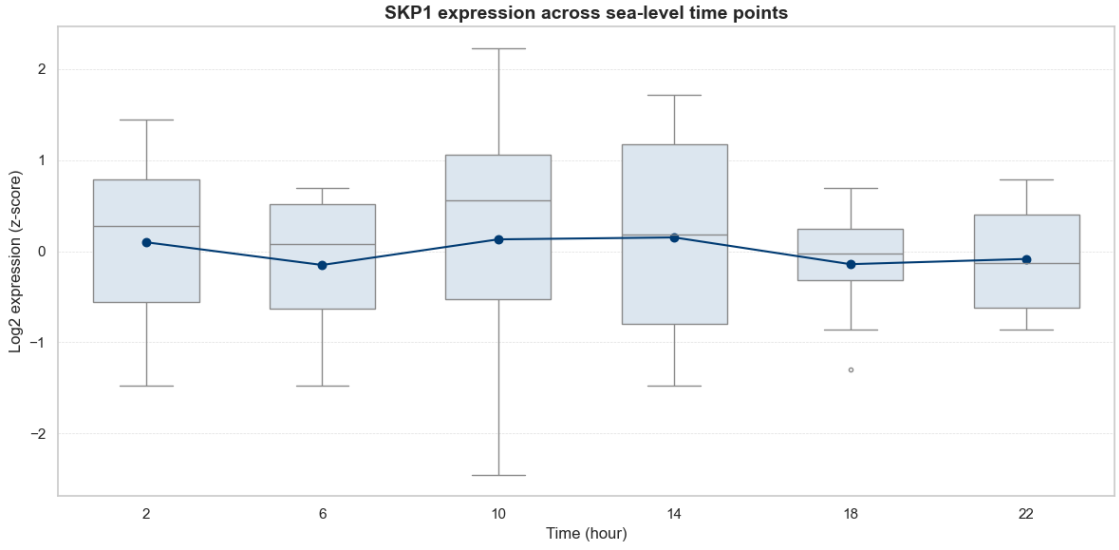

RPS27A expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

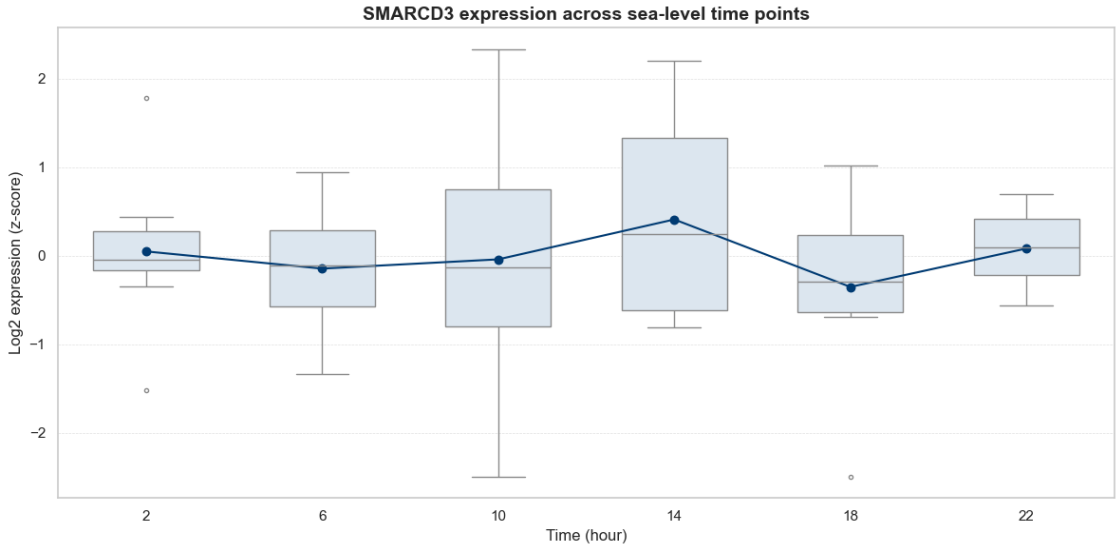

RXRA expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

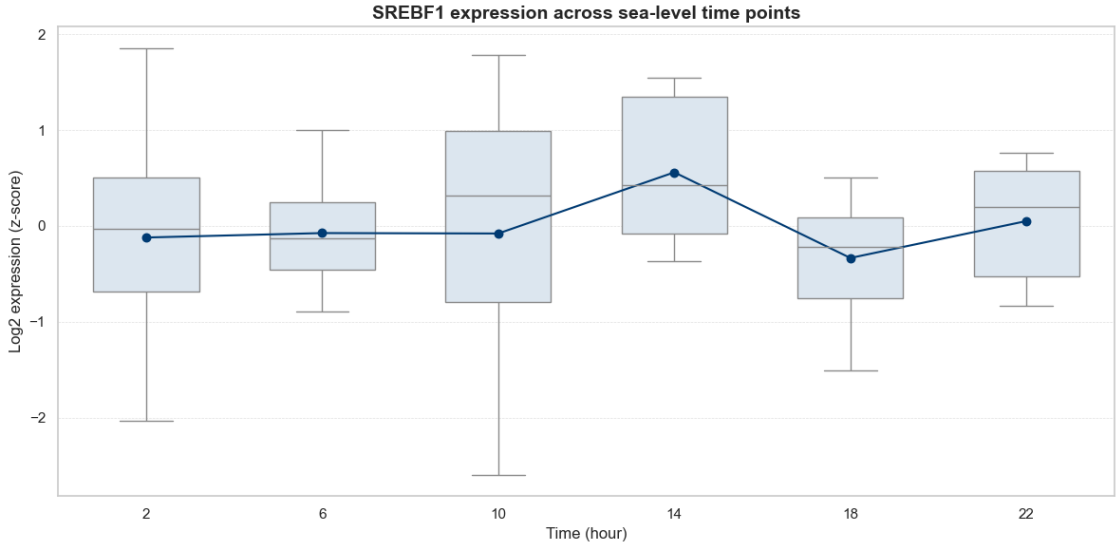

SEM1 expression across sea-level time points
Log2 expression (z-score)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
2
6
10
14
18
22
Time (hour)

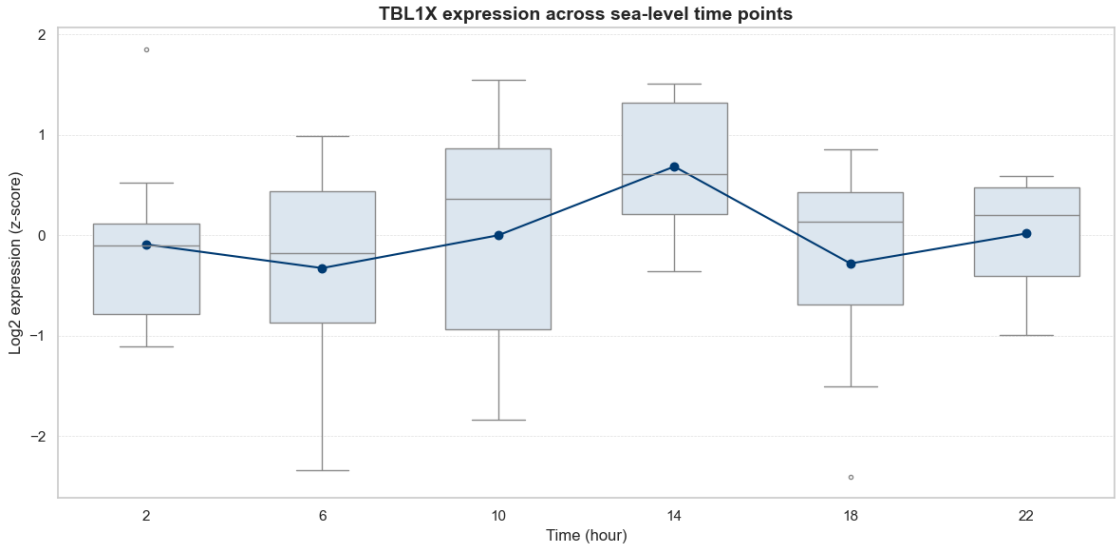

SERPINE1 expression across sea-level time points
Log2 expression (z-score)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
2
6
10
14
18
22
Time (hour)

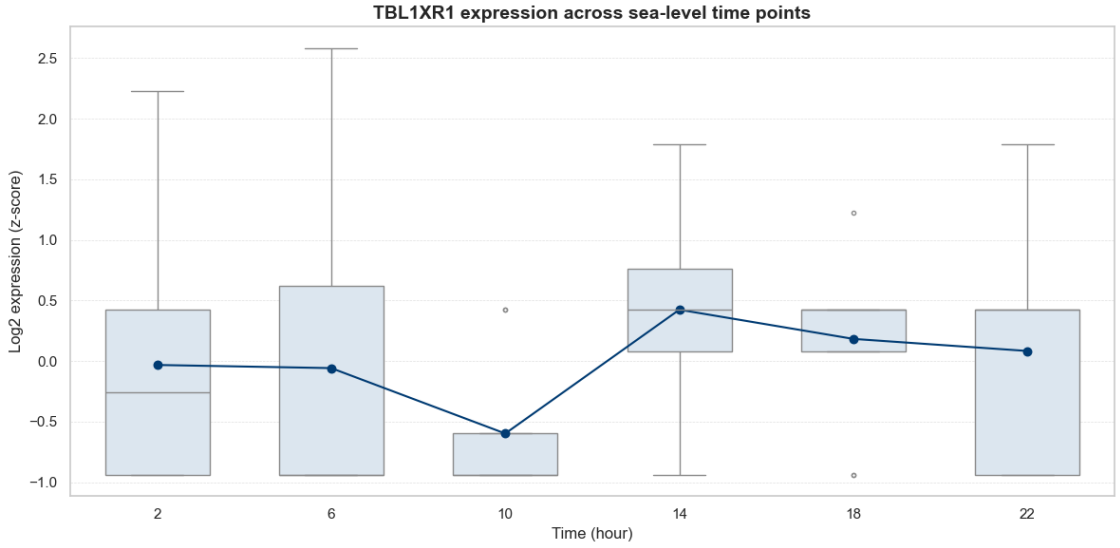

SIRT1 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

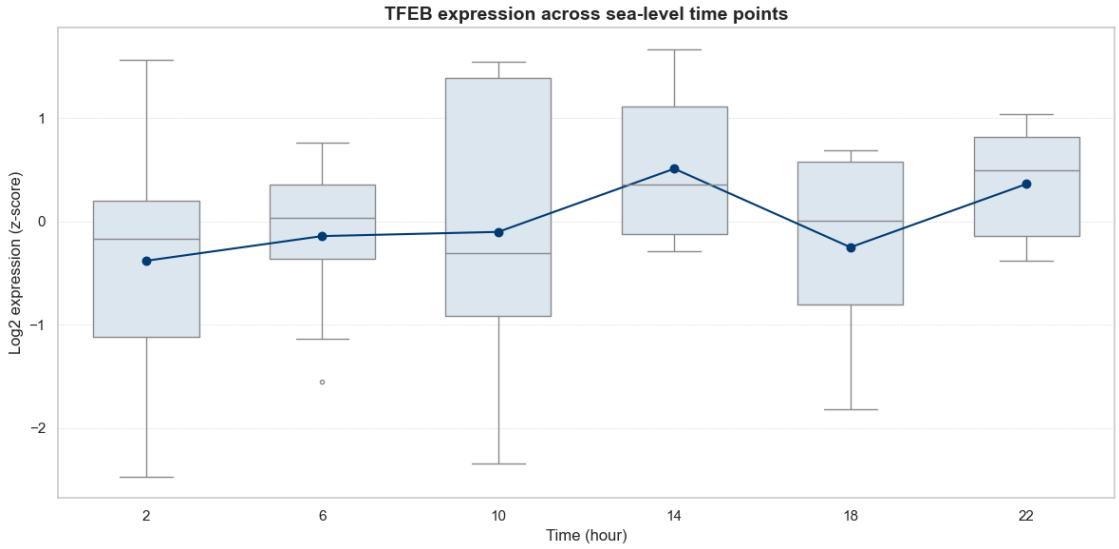

SKP1 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

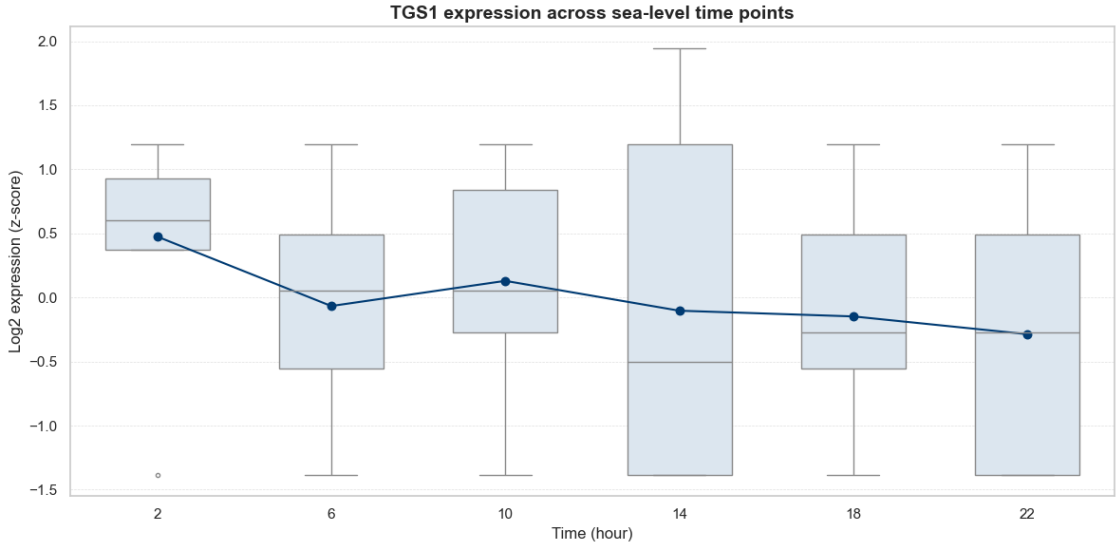

SMARCD3 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

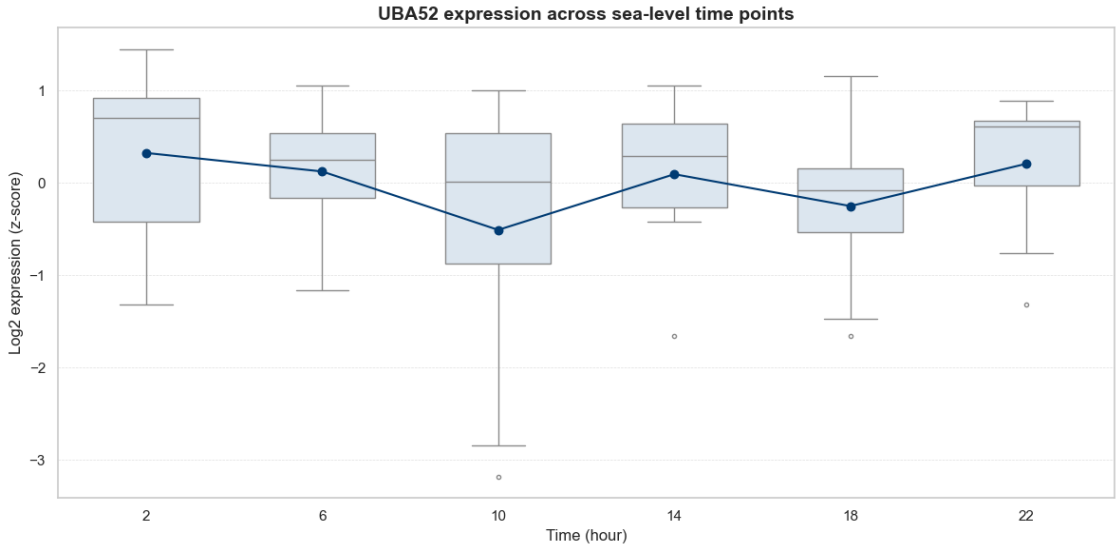

SREBF1 expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

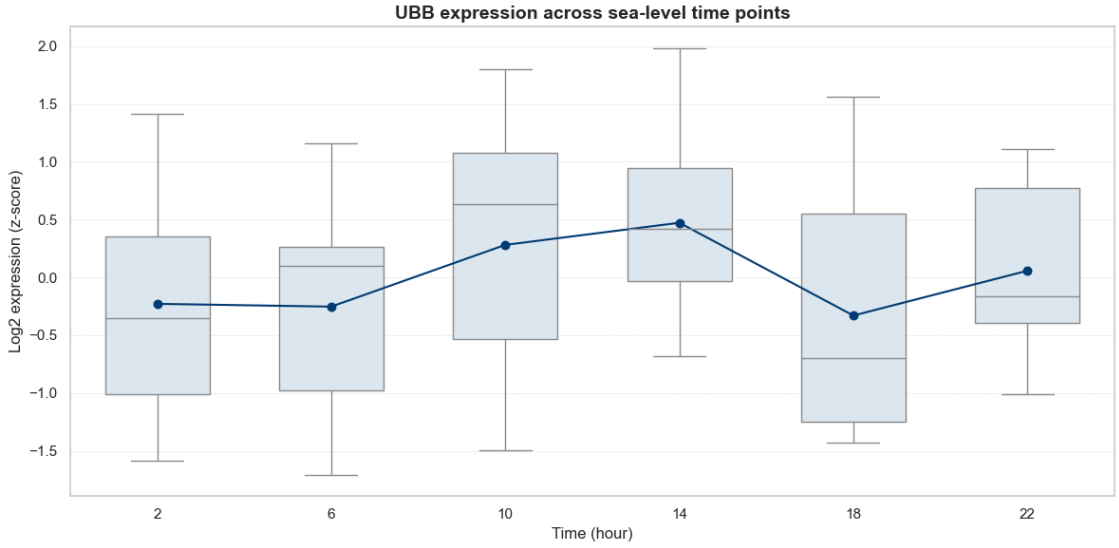

TBL1X expression across sea-level time points
Log2 expression (z-score)
2
1
0
−1
−2
2
6
10
14
18
22
Time (hour)

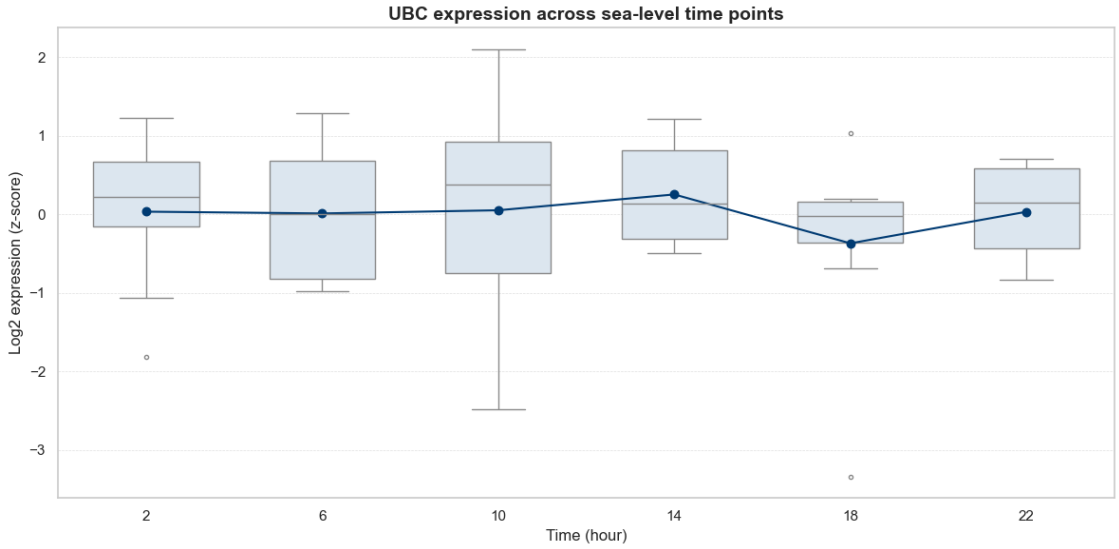

TBL1XR1 expression across sea-level time points
Log2 expression (z-score)
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
2
6
10
14
18
22
Time (hour)

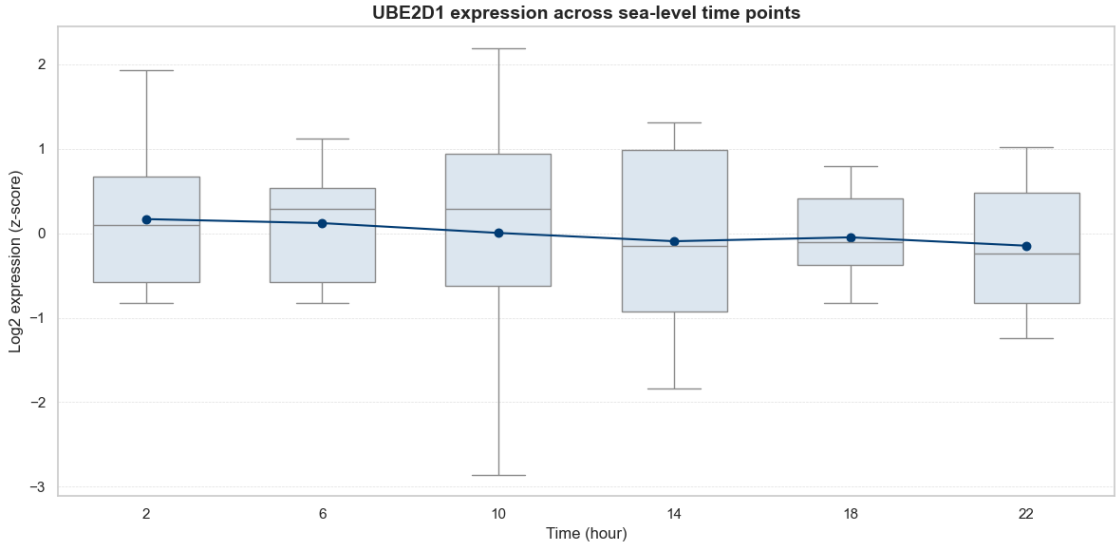

TFEB expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
2
6
10
14
18
22
Time (hour)


TGS1 expression across sea-level time points
Log2 expression (z-score)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
2
6
10
14
18
22
Time (hour)


UBA52 expression across sea-level time points
Log2 expression (z-score)
1
0
−1
−2
−3
2
6
10
14
18
22
Time (hour)


UBB expression across sea-level time points
Log2 expression (z-score)
2.0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
2
6
10
14
18
22
Time (hour)




## 1.4 Test for time-of-day effects using ANOVA

This cell tests, for each gene, whether average expression differs between the six time points.

Steps:

1. **Prepare the data**
   - It creates `long_df` with columns:
     - `gene`, `time`, `z_expr` (the z-scored log expression).
2. **Fit a one-way ANOVA per gene**

- For each gene, it fits a statistical model:
  - `z_expr ~ C(time)`
  - Here `C(time)` treats the six time values as categorical groups.
- It extracts the p-value for the overall "time" effect from the ANOVA table.

3. **Collect and sort results**

- It stores, for each gene:
  - the gene name,
  - the p-value for time.
- It sorts genes by increasing p-value and displays the top five.

In this sea-level circadian gene subset, the smallest p-values are around 0.07, and no gene reaches the conventional $p < 0.05$ threshold. This suggests that, within this dataset, we do not detect strong and statistically significant differences in mean expression across the six time points for any of these Reactome circadian genes.

[11]:
```
if 'temporal_df' not in locals():
    raise NameError('Run the temporal formatting cell first to compute␣
 ↪temporal_df.')

import statsmodels.api as sm
from statsmodels.formula.api import ols

long_df = temporal_df[['gene', 'time', 'log_zscore']].dropna().
 ↪rename(columns={'log_zscore': 'z_expr'})
results = []
for gene in long_df['gene'].unique():
    tmp = long_df[long_df['gene'] == gene]
    if tmp['time'].nunique() < 2:
        results.append({'gene': gene, 'p_time': float('nan')})
        continue
    model = ols('z_expr ~ C(time)', data=tmp).fit()
    anova_table = sm.stats.anova_lm(model, typ=2)
    pval = anova_table.loc['C(time)', 'PR(>F)']
    results.append({'gene': gene, 'p_time': pval})

time_effects = pd.DataFrame(results).sort_values('p_time')
display(time_effects.head())
```

```
        gene    p_time
12    CREBBP  0.070395
67     PSMC6  0.112670
78      RAI1  0.125033
44      PER3  0.125465
86  SERPINE1  0.127829
```

### 1.5 Attempt to cluster time-dependent genes (none found)

This cell is designed to cluster genes that show significant time-of-day effects, but it also reports when that set is empty.

- It defines a p-value threshold (`pval_threshold = 0.05`).
- It selects genes from `time_effects` whose `p_time` is below this threshold.
- It prints how many such genes exist.
- If any genes passed the threshold, it would:
  - Calculate the mean z-score per gene per time.
  - Standardize those temporal profiles.
  - Run hierarchical clustering to group genes with similar time patterns.

In the current results, there are **no genes with $p < 0.05$**, so:

- It prints:
  - "Significant genes ($p < 0.05$): 0"
  - "No significant genes found to cluster."
- No clustering is performed.

The conclusion is that, for this sea-level circadian gene set, we do not have a subset of genes with robustly different expression profiles across the six time points, at least under this statistical threshold.

[12]:
```
if 'temporal_df' not in locals() or 'time_effects' not in locals():
    raise NameError('Ensure temporal_df and time_effects are computed before␣
 ↪clustering.')

pval_threshold = 0.05
sig_genes = time_effects[time_effects['p_time'] < pval_threshold]['gene']
print(f'Significant genes (p < {pval_threshold}): {len(sig_genes)}')

if sig_genes.empty:
    print('No significant genes found to cluster.')
else:
    times = sorted(temporal_df['time'].dropna().unique())
    mean_profiles = (temporal_df[temporal_df['gene'].isin(sig_genes)]
                     .groupby(['gene', 'time'])['log_zscore']
                     .mean()
                     .unstack())
    mean_profiles = mean_profiles.reindex(columns=times)
    mean_profiles = mean_profiles.apply(lambda row: row.fillna(row.mean()),␣
 ↪axis=1).fillna(0)

    scaler = StandardScaler()
    scaled_profiles = scaler.fit_transform(mean_profiles)
    n_clusters = min(4, len(mean_profiles))
    if n_clusters < 2:
        print('Not enough genes for clustering (need at least 2).')
    else:
```

```
        clustering = AgglomerativeClustering(n_clusters=n_clusters)
        labels = clustering.fit_predict(scaled_profiles)
        cluster_df = (pd.DataFrame({'gene': mean_profiles.index, 'cluster':␣
↪labels})
                      .merge(time_effects, on='gene', how='left')
                      .sort_values(['cluster', 'p_time']))
        print(f'Generated {n_clusters} clusters.')
        display(cluster_df)
```

```
Significant genes (p < 0.05): 0
No significant genes found to cluster.
```

### 1.6 Cosinor-type analysis for circadian rhythmicity

This cell tests each gene for a smooth 24-hour-like rhythm using a simple cosinor model.

1. **Prepare the design variables**
   - It takes `temporal_df` and keeps `gene`, `time`, `log_zscore` (renamed to `z_expr`).
   - It converts each time (2, 6, 10, 14, 18, 22 hours) into a phase in radians:
     - `phase_rad = 2 * (time / 24)`.
   - It computes two predictors:
     - `cos_t = cos(phase_rad)`,
     - `sin_t = sin(phase_rad)`.
2. **Fit a cosinor model per gene**
   - For each gene, it fits:
     - `z_expr ~ cos_t + sin_t`.
   - From the fitted model, it computes:
     - `amp`: the rhythmic amplitude, based on the fitted cosine and sine coefficients.
     - `phase`: the estimated peak phase, derived from those coefficients.
   - It uses an F-test to compute a p-value for the hypothesis that both rhythmic terms are zero (no 24-hour component).
3. **Collect results**
   - It creates `cos_df` with one row per gene:
     - `gene`, `amp`, `phase`, `p`.
   - It sorts genes by p-value and displays the top five.

This analysis asks a more specifically "circadian" question than the ANOVA: is there an approximately sinusoidal 24-hour pattern? In this sea-level subset of Reactome circadian genes, the smallest p-value is about 0.089, so none of the genes show statistically significant 24-hour rhythmicity at $p < 0.05$ under this model.

[13]:
```
if 'temporal_df' not in locals():
    raise NameError('Run the temporal formatting cell first to compute␣
↪temporal_df.')
```

```
long_df = temporal_df[['gene', 'time', 'log_zscore']].dropna().
 ↪rename(columns={'log_zscore': 'z_expr'})
long_df = long_df[long_df['time'].notna()].copy()
long_df['phase_rad'] = 2 * np.pi * (long_df['time'] / 24.0)
long_df['cos_t'] = np.cos(long_df['phase_rad'])
long_df['sin_t'] = np.sin(long_df['phase_rad'])

results_cos = []
for gene in long_df['gene'].unique():
    tmp = long_df[long_df['gene'] == gene]
    if tmp['time'].nunique() < 2:
        results_cos.append({'gene': gene, 'amp': np.nan, 'phase': np.nan, 'p':␣
 ↪np.nan})
        continue
    model = ols('z_expr ~ cos_t + sin_t', data=tmp).fit()
    beta_cos = model.params.get('cos_t', np.nan)
    beta_sin = model.params.get('sin_t', np.nan)
    amp = np.sqrt(beta_cos**2 + beta_sin**2) if np.isfinite(beta_cos) and np.
 ↪isfinite(beta_sin) else np.nan
    phase = np.arctan2(-beta_sin, beta_cos) if np.isfinite(beta_cos) and np.
 ↪isfinite(beta_sin) else np.nan
    pval = float(model.f_test('cos_t = sin_t = 0').pvalue)
    results_cos.append({'gene': gene, 'amp': amp, 'phase': phase, 'p': pval})

cos_df = pd.DataFrame(results_cos).sort_values('p')
display(cos_df.head())
```

```
        gene       amp     phase         p
67     PSMC6  0.446530 -2.829505  0.089273
86  SERPINE1  0.375480  3.067899  0.186170
84      RXRA  0.374810  2.409253  0.187331
57     PSMB2  0.342682 -2.886764  0.248739
81      RORA  0.319594  0.441740  0.299620
```

### 1.7 Attempt to cluster genes by circadian phase (none significant)

This final analysis step would group genes by their estimated circadian phase, but only if any genes were found to be significantly rhythmic.

- It filters `cos_df` to keep genes with:
  - $p < 0.05$,
  - non-missing amplitude and phase.
- If that set (`cos_sig`) is not empty, it:
  - Wraps phases to the interval $[0, 2\ )$.
  - Converts phase to a peak hour in $[0, 24)$.
  - Represents each gene's phase on the unit circle (`cos_phase`, `sin_phase`).
  - Clusters genes into up to four phase clusters.
  - Summarizes each cluster by average peak hour and amplitude.

  - Plots amplitude vs peak hour and labels each point by gene name.

In this sea-level dataset:

- No gene has $p < 0.05$ in the cosinor analysis.
- The cell prints:
  - "No significant rhythmic genes to analyze for phase similarity."
- No clustering or phase plot is produced.

Biologically, this means we do not identify any genes in this Reactome circadian subset that show a robust, statistically significant 24-hour rhythm at sea level under the current sampling and noise conditions, so it is not meaningful to group them by "circadian phase" in this context.

[14]:
```
if 'cos_df' not in locals():
    raise NameError('Run the cosinor analysis cell first to compute cos_df.')

signif_threshold = 0.05
cos_sig = cos_df[(cos_df['p'] < signif_threshold) & cos_df['amp'].notna() &␣
 ↪cos_df['phase'].notna()].copy()
if cos_sig.empty:
    print('No significant rhythmic genes to analyze for phase similarity.')
else:
    cos_sig['phase_wrapped'] = (cos_sig['phase'] % (2 * np.pi))
    cos_sig['peak_hour'] = (cos_sig['phase_wrapped'] / (2 * np.pi)) * 24
    cos_sig['cos_phase'] = np.cos(cos_sig['phase_wrapped'])
    cos_sig['sin_phase'] = np.sin(cos_sig['phase_wrapped'])
    n_clusters = min(4, len(cos_sig))
    if n_clusters < 2:
        cos_sig['phase_cluster'] = 0
        print('Fewer than two significant genes; skipping clustering.')
    else:
        clustering = AgglomerativeClustering(n_clusters=n_clusters)
        cos_sig['phase_cluster'] = clustering.fit_predict(cos_sig[['cos_phase',␣
 ↪'sin_phase']])
        print(f'Generated {n_clusters} phase clusters for rhythmic genes.')

    cluster_summary = (cos_sig.groupby('phase_cluster')
                       .agg(mean_peak_hour=('peak_hour', 'mean'),
                            mean_amp=('amp', 'mean'),
                            genes=('gene', lambda x: list(x)))
                       .reset_index())
    display(cluster_summary)

    plt.figure(figsize=(8, 5))
    sns.scatterplot(data=cos_sig, x='peak_hour', y='amp', hue='phase_cluster',␣
 ↪palette='tab10', s=80)
    for _, row in cos_sig.iterrows():
        plt.text(row['peak_hour'], row['amp'], row['gene'], fontsize=8,␣
 ↪ha='left', va='bottom')
```

```
    plt.xlabel('Peak phase (hours)')
    plt.ylabel('Rhythmic amplitude')
    plt.title('Circadian phase vs amplitude for significant genes')
    plt.tight_layout()
    plot_path = os.path.join('Data', 'plots', 'circadian_phase_clusters.png')
    os.makedirs(os.path.dirname(plot_path), exist_ok=True)
    plt.savefig(plot_path, dpi=150)
    plt.show()
    plt.close()
    print(f'Saved phase/amplitude plot to {plot_path}')
```

```
No significant rhythmic genes to analyze for phase similarity.
```

## 1.8 Summary

In this notebook we re-analysed the whole-blood RNA-seq data from the Peru high-altitude study, focusing **only on sea-level (SL) samples** and a **curated Reactome circadian clock gene set**. We:

- Imported the full expression matrix (12,726 genes × 145 samples).
- Selected only sea-level samples (`sub*_SL*.raw`), giving **48 SL samples** across **6 time points** (2, 6, 10, 14, 18, 22 h), with **8 different subjects per time point** (cross-sectional design, not longitudinal).
- Identified **99 circadian-related genes** from the Reactome "Circadian Clock" pathway that are present in this dataset.
- Built a "temporal" table where each row is one gene in one subject at one time point (4,752 rows total), using log2-transformed counts and per-gene z-scores.

This gave us a clean, SL-only dataset suitable for asking whether circadian genes show time-of-day structure in whole blood.

---

## 1.9 What the data show

### 1.9.1 1. Visual patterns

Boxplots of each circadian gene across the six time points show:

- Most genes have **substantial between-subject variability** at each time point.
- Many genes display **subtle trends** (for example, slightly higher median expression around 14 h for several transcriptional regulators), but there is **no strong, consistent oscillation** across all six time points.

Overall, the visual impression is "noisy with mild hints of time-of-day effects," rather than clear rhythmic peaks and troughs.

---

### 1.9.2 2. ANOVA: differences in mean expression across time

We fitted a one-way ANOVA for each gene:

z-scored log-expression ~ time (2, 6, 10, 14, 18, 22 h as categorical groups)

- The smallest p-values were for genes such as **CREBBP, PSMC6, RAI1, PER3, SERPINE1** (p  0.07–0.13).
- **No gene reached p < 0.05**, so we did not identify any circadian-pathway genes with a statistically significant difference in mean expression between the six time points at sea level.

This means that, given the sample size and noise level, we cannot say that any of these canonical circadian genes are "time-of-day regulated" at SL in whole blood using this strict criterion.

---

#### 1.9.3 3. Cosinor analysis: testing for ~24-hour rhythms

To specifically ask for circadian-like patterns, we fitted for each gene a simple cosinor model:

z-scored log-expression ~ cos(2  · time/24) + sin(2  · time/24)

From this we derived:

- An amplitude (strength of a 24-h component).
- A phase (estimated peak time).
- A p-value for the joint null hypothesis of "no 24-hour component."

Findings:

- The most "rhythmic-looking" genes by this metric included **PSMC6, SERPINE1, RXRA, PSMB2, RORA**, with amplitudes around 0.3–0.45, but the best p-value was  0.089.
- Again, **no gene achieved p < 0.05**, so we did not detect robust 24-hour rhythmicity in any Reactome circadian gene at sea level.

Because of this, downstream clustering of "rhythmic genes" by phase was not performed (there were no significant genes to cluster).

---

### 1.10 Interpret

1. **The analysis is technically working and reproduces a key negative result.**
   Our pipeline behaves as expected and matches the original paper's conclusion that classical circadian methods fail to detect strong rhythms in this dataset, likely due to field conditions (night-time awakenings, sleep disruption, etc.) and the fact that whole blood is a heterogeneous tissue.

2. **At sea level, canonical clock genes do not show strong, statistically robust time-of-day regulation in whole blood.**
   Within this study design (different subjects at each time point, 8 per time), any time-of-day variation in circadian genes appears to be **small compared with inter-individual variability**. This suggests that, under "real-world" conditions, whole-blood expression of these pathway genes is not a sensitive readout of circadian phase.

3. **There are "suggestive" but underpowered trends.**
   A handful of genes (for example **CREBBP, PER3, PSMC6, SERPINE1, RXRA, RORA**) show the lowest p-values and modest amplitudes. These may genuinely have some

time-of-day modulation, but the current sample size and study design do not provide enough statistical power to call them significant.

4. **Important design limitation: cross-sectional, not longitudinal.**
   Each time point has different subjects, so we are estimating **population-level** time-of-day differences, not within-person rhythms. This structure inherently makes it harder to detect circadian patterns, because between-person differences in baseline expression dominate the signal.

---

### 1.11 Usecases

Even though we do not see strong circadian signals in canonical clock genes at sea level, this analysis gives us three concrete things for the cancer-focused work:

1. **Baseline expectation.**
   We should *not* expect large, clean circadian amplitudes for most genes in this blood dataset. For cancer-related genes, even modest and noisy day–night differences could still be biologically meaningful, especially if they are directionally consistent within a gene set.

2. **Validated analysis pipeline.**
   The current code reliably:

   - Parses sample metadata.
   - Normalises expression.
   - Performs time-of-day and cosinor tests on an arbitrary gene set.
     We can now apply the same pipeline to **cancer gene lists** (e.g. breast-cancer signatures, DNA damage genes, cell-cycle genes) with confidence that technical issues are unlikely to explain the results.

3. **Next, use a simpler contrast that has more power.**
   The original study gained power by pooling time points into **"light" (10–18 h) vs "dark" (22–6 h)** windows instead of modelling all six times separately. For cancer-related genes, we can:

   - Collapse samples into light vs dark at sea level.
   - Test for differences in expression between these two phases, gene by gene or at the pathway level.
   - Compare the effect sizes for cancer genes against those for the circadian-pathway genes analysed here.

This gives a clear, realistic starting point for asking whether cancer-relevant pathways show any systematic day–night bias in circulating blood cells, even though classical clock genes themselves do not show strong rhythmicity in this particular dataset.

---