Final Exam - Group Portion - Supplementary Code Notebook

Data Science for Biology

Notebook developed by: *Max Staller*, *Kinsey Long*

Final Group Individual Portion Instructions

- The google doc above superceeds instructions in the group project final notebooks and it will have updates as corrections emerge.

This document should contain all of the supplementary code used in your group final project. You can copy over code from the warm-up if you use it. Your code must be well-documented with comments and should be modular with functions.

Submission instructions:

- Upload the Final Group Warm-up python notebook to Final Group Warm-up Autograder (one submission per group, add members to the assignment)
- Upload the Final Group Warm-up PDF to Final Group Warm-up Written Work (one submission per group, add members to the assignment)
- Upload the Final Group (Project) PDF to Final Group Project (one submission per group, add members to the assignment)
- Upload the Final Group (Project) Supplementary Code (this notebook) as a PDF to Final Group Project Supplementary Code (one submission per group, add members to the assignment)
- Upload the Final Group Academic Honesty Statement individually

We expect you to work together. You may divide up the steps of the analysis and perform those subparts individually or in pairs or trios. For each code module, clearly indicate which group members contributed. For each figure, in the paragraph after the figure indicate which team members contributed.

We want everyone to make both intellectual contributions and coding contributions. Coding contributions are writing code to perform a filter or analysis or create a plot. Intellectual contributions are deciding what filters to apply, what analyzes to perform, or how to plot the data.

Project Options

Option A: Glucose vs Galactose

- What is the global change in Tile activities (if any)?
- Are there some Tiles that show condition specific activity? (ie they are stronger in Galactose or in Glucose)
- What are the sequence features of these Tiles

Option B: Importance of Mig1 binding sites Compare P3promoter with P3promoter_Mig1TFBSremoved

- Is Mig 1 a strong or weak repressor?
- Is there a global pattern?
- The literature says that Mig1 should be inactive (sequestered in the cytoplasm) during growth in galactose. Test the hypothesis that the P3promoter in Galactose (Activity_SCgalactose) is more similar to - P3promoter_Mig1TFBSremoved than it is to P3promoter in Glucose (Activity_SCglucose).

Option C: Promoter-specific Activity

- Compare the activities of the 4 different promoters.
- Are there tiles that show strong promoter specificity? (they prefer one promoter over the others)
  - What are the sequence features of these tiles?

Option D: Consistent vs Variable ADs

- What ADs are the most consistent across all conditions?
  - Are there any ADs that have very consistent activity and show no condition specific activity?
  - What are their protein sequence features?
- What tiles are most variable across all conditions?
  - Are there tiles with very different activity in each condition?
  - Are these well measured or poorly measured?
- What are their protein sequence features?

Option E: Protein Stability

- Look at the mCherry signal, which reports on protein stability
- Look for protein sequence features associated with high stability or low stability tiles.

General analysis approaches to find protein sequence features
Once you have identified a set of interesting tiles, you can use any of these approaches to find the protein sequence features:
- Are known motifs associated with the phenotype?
- Or perform de novo motif finding. What protein sequence motifs control activity in different conditions?
- Machine learning to predict activity in a condition from sequence.

- Choose a condition and predict activity
- Use ordinary least squares regression models to describe how composition controls the phenotype or ANOVA.

Student Group Project - Supplementary Code File

Project Members: *double-click to type names here*
Project Choice: *double-click to type project choice here*

In [1]:
```
#Some packages you might want, feel free to add more
!pip install seaborn==0.13.2
!pip install scikit-learn==1.4.2
!pip install transformers
!pip install torch
!pip install python-dotenv

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import re
from collections import Counter
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import Ridge, Lasso, LinearRegression, ElasticNet
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressc
from sklearn.pipeline import make_pipeline
from sklearn.metrics import r2_score, mean_absolute_error, root_mean_squarec
from scipy.stats import pearsonr

from transformers import AutoTokenizer, AutoModel
import torch
from dotenv import load_dotenv
import os
from tqdm import tqdm
import joblib

# Set random seed
np.random.seed(42)
```

```
Requirement already satisfied: seaborn==0.13.2 in /srv/conda/lib/python3.11/
site-packages (0.13.2)
Requirement already satisfied: numpy!=1.24.0,>=1.20 in /srv/conda/lib/python
3.11/site-packages (from seaborn==0.13.2) (1.26.4)
Requirement already satisfied: pandas>=1.2 in /srv/conda/lib/python3.11/site
-packages (from seaborn==0.13.2) (2.2.2)
Requirement already satisfied: matplotlib!=3.6.1,>=3.4 in /srv/conda/lib/pyt
hon3.11/site-packages (from seaborn==0.13.2) (3.10.1)
Requirement already satisfied: contourpy>=1.0.1 in /srv/conda/lib/python3.1
1/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /srv/conda/lib/python3.11/sit
e-packages (from matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /srv/conda/lib/python3.1
1/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /srv/conda/lib/python3.1
1/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /srv/conda/lib/python3.11/
site-packages (from matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (24.2)
Requirement already satisfied: pillow>=8 in /srv/conda/lib/python3.11/site-p
ackages (from matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (9.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in /srv/conda/lib/python3.1
1/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /srv/conda/lib/python
3.11/site-packages (from matplotlib!=3.6.1,>=3.4->seaborn==0.13.2) (2.9.0.po
st0)
Requirement already satisfied: pytz>=2020.1 in /srv/conda/lib/python3.11/sit
e-packages (from pandas>=1.2->seaborn==0.13.2) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /srv/conda/lib/python3.11/s
ite-packages (from pandas>=1.2->seaborn==0.13.2) (2025.2)
Requirement already satisfied: six>=1.5 in /srv/conda/lib/python3.11/site-pa
ckages (from python-dateutil>=2.7->matplotlib!=3.6.1,>=3.4->seaborn==0.13.2)
(1.17.0)
Collecting scikit-learn==1.4.2
  Using cached scikit_learn-1.4.2-cp311-cp311-manylinux_2_17_x86_64.manylinu
x2014_x86_64.whl.metadata (11 kB)
Requirement already satisfied: numpy>=1.19.5 in /srv/conda/lib/python3.11/si
te-packages (from scikit-learn==1.4.2) (1.26.4)
Requirement already satisfied: scipy>=1.6.0 in /srv/conda/lib/python3.11/sit
e-packages (from scikit-learn==1.4.2) (1.15.1)
Requirement already satisfied: joblib>=1.2.0 in /srv/conda/lib/python3.11/si
te-packages (from scikit-learn==1.4.2) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /srv/conda/lib/python
3.11/site-packages (from scikit-learn==1.4.2) (3.6.0)
Using cached scikit_learn-1.4.2-cp311-cp311-manylinux_2_17_x86_64.manylinux2
014_x86_64.whl (12.1 MB)
Installing collected packages: scikit-learn
  Attempting uninstall: scikit-learn
    Found existing installation: scikit-learn 1.2.2
    Uninstalling scikit-learn-1.2.2:
      Successfully uninstalled scikit-learn-1.2.2
Successfully installed scikit-learn-1.4.2
Collecting transformers
  Using cached transformers-4.51.3-py3-none-any.whl.metadata (38 kB)
Collecting filelock (from transformers)
  Using cached filelock-3.18.0-py3-none-any.whl.metadata (2.9 kB)
Collecting huggingface-hub<1.0,>=0.30.0 (from transformers)
```

```
  Using cached huggingface_hub-0.31.1-py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: numpy>=1.17 in /srv/conda/lib/python3.11/site
-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /srv/conda/lib/python3.11/
site-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /srv/conda/lib/python3.11/site
-packages (from transformers) (6.0.2)
Collecting regex!=2019.12.17 (from transformers)
  Using cached regex-2024.11.6-cp311-cp311-manylinux_2_17_x86_64.manylinux20
14_x86_64.whl.metadata (40 kB)
Requirement already satisfied: requests in /srv/conda/lib/python3.11/site-pa
ckages (from transformers) (2.32.3)
Collecting tokenizers<0.22,>=0.21 (from transformers)
  Using cached tokenizers-0.21.1-cp39-abi3-manylinux_2_17_x86_64.manylinux20
14_x86_64.whl.metadata (6.8 kB)
Collecting safetensors>=0.4.3 (from transformers)
  Using cached safetensors-0.5.3-cp38-abi3-manylinux_2_17_x86_64.manylinux20
14_x86_64.whl.metadata (3.8 kB)
Requirement already satisfied: tqdm>=4.27 in /srv/conda/lib/python3.11/site-
packages (from transformers) (4.67.1)
Collecting fsspec>=2023.5.0 (from huggingface-hub<1.0,>=0.30.0->transformer
s)
  Using cached fsspec-2025.3.2-py3-none-any.whl.metadata (11 kB)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /srv/conda/lib/
python3.11/site-packages (from huggingface-hub<1.0,>=0.30.0->transformers)
(4.13.2)
Collecting hf-xet<2.0.0,>=1.1.0 (from huggingface-hub<1.0,>=0.30.0->transfor
mers)
  Using cached hf_xet-1.1.0-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x8
6_64.whl.metadata (494 bytes)
Requirement already satisfied: charset_normalizer<4,>=2 in /srv/conda/lib/py
thon3.11/site-packages (from requests->transformers) (3.4.1)
Requirement already satisfied: idna<4,>=2.5 in /srv/conda/lib/python3.11/sit
e-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /srv/conda/lib/python3.
11/site-packages (from requests->transformers) (2.3.0)
Requirement already satisfied: certifi>=2017.4.17 in /srv/conda/lib/python3.
11/site-packages (from requests->transformers) (2025.1.31)
Using cached transformers-4.51.3-py3-none-any.whl (10.4 MB)
Using cached huggingface_hub-0.31.1-py3-none-any.whl (484 kB)
Using cached regex-2024.11.6-cp311-cp311-manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (792 kB)
Using cached safetensors-0.5.3-cp38-abi3-manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (471 kB)
Using cached tokenizers-0.21.1-cp39-abi3-manylinux_2_17_x86_64.manylinux2014
_x86_64.whl (3.0 MB)
Using cached filelock-3.18.0-py3-none-any.whl (16 kB)
Using cached fsspec-2025.3.2-py3-none-any.whl (194 kB)
Using cached hf_xet-1.1.0-cp37-abi3-manylinux_2_17_x86_64.manylinux2014_x86_
64.whl (53.6 MB)
Installing collected packages: safetensors, regex, hf-xet, fsspec, filelock,
huggingface-hub, tokenizers, transformers
Successfully installed filelock-3.18.0 fsspec-2025.3.2 hf-xet-1.1.0 huggingf
ace-hub-0.31.1 regex-2024.11.6 safetensors-0.5.3 tokenizers-0.21.1 transform
ers-4.51.3
Collecting torch
```

```
  Using cached torch-2.7.0-cp311-cp311-manylinux_2_28_x86_64.whl.metadata (2
9 kB)
Requirement already satisfied: filelock in /srv/conda/lib/python3.11/site-pa
ckages (from torch) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /srv/conda/lib/p
ython3.11/site-packages (from torch) (4.13.2)
Collecting sympy>=1.13.3 (from torch)
  Using cached sympy-1.14.0-py3-none-any.whl.metadata (12 kB)
Collecting networkx (from torch)
  Using cached networkx-3.4.2-py3-none-any.whl.metadata (6.3 kB)
Requirement already satisfied: jinja2 in /srv/conda/lib/python3.11/site-pack
ages (from torch) (3.1.6)
Requirement already satisfied: fsspec in /srv/conda/lib/python3.11/site-pack
ages (from torch) (2025.3.2)
Collecting nvidia-cuda-nvrtc-cu12==12.6.77 (from torch)
  Using cached nvidia_cuda_nvrtc_cu12-12.6.77-py3-none-manylinux2014_x86_64.
whl.metadata (1.5 kB)
Collecting nvidia-cuda-runtime-cu12==12.6.77 (from torch)
  Using cached nvidia_cuda_runtime_cu12-12.6.77-py3-none-manylinux2014_x86_6
4.manylinux_2_17_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cuda-cupti-cu12==12.6.80 (from torch)
  Using cached nvidia_cuda_cupti_cu12-12.6.80-py3-none-manylinux2014_x86_64.
manylinux_2_17_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cudnn-cu12==9.5.1.17 (from torch)
  Using cached nvidia_cudnn_cu12-9.5.1.17-py3-none-manylinux_2_28_x86_64.wh
l.metadata (1.6 kB)
Collecting nvidia-cublas-cu12==12.6.4.1 (from torch)
  Using cached nvidia_cublas_cu12-12.6.4.1-py3-none-manylinux2014_x86_64.man
ylinux_2_17_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufft-cu12==11.3.0.4 (from torch)
  Using cached nvidia_cufft_cu12-11.3.0.4-py3-none-manylinux2014_x86_64.many
linux_2_17_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-curand-cu12==10.3.7.77 (from torch)
  Using cached nvidia_curand_cu12-10.3.7.77-py3-none-manylinux2014_x86_64.ma
nylinux_2_17_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cusolver-cu12==11.7.1.2 (from torch)
  Using cached nvidia_cusolver_cu12-11.7.1.2-py3-none-manylinux2014_x86_64.m
anylinux_2_17_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparse-cu12==12.5.4.2 (from torch)
  Using cached nvidia_cusparse_cu12-12.5.4.2-py3-none-manylinux2014_x86_64.m
anylinux_2_17_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-cusparselt-cu12==0.6.3 (from torch)
  Using cached nvidia_cusparselt_cu12-0.6.3-py3-none-manylinux2014_x86_64.wh
l.metadata (6.8 kB)
Collecting nvidia-nccl-cu12==2.26.2 (from torch)
  Using cached nvidia_nccl_cu12-2.26.2-py3-none-manylinux2014_x86_64.manylin
ux_2_17_x86_64.whl.metadata (2.0 kB)
Collecting nvidia-nvtx-cu12==12.6.77 (from torch)
  Using cached nvidia_nvtx_cu12-12.6.77-py3-none-manylinux2014_x86_64.manyli
nux_2_17_x86_64.whl.metadata (1.6 kB)
Collecting nvidia-nvjitlink-cu12==12.6.85 (from torch)
  Using cached nvidia_nvjitlink_cu12-12.6.85-py3-none-manylinux2010_x86_64.m
anylinux_2_12_x86_64.whl.metadata (1.5 kB)
Collecting nvidia-cufile-cu12==1.11.1.6 (from torch)
  Using cached nvidia_cufile_cu12-1.11.1.6-py3-none-manylinux2014_x86_64.man
ylinux_2_17_x86_64.whl.metadata (1.5 kB)
```

```
Collecting triton==3.3.0 (from torch)
  Using cached triton-3.3.0-cp311-cp311-manylinux_2_27_x86_64.manylinux_2_28
_x86_64.whl.metadata (1.5 kB)
Requirement already satisfied: setuptools>=40.8.0 in /srv/conda/lib/python3.
11/site-packages (from triton==3.3.0->torch) (75.6.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /srv/conda/lib/python3.
11/site-packages (from sympy>=1.13.3->torch) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /srv/conda/lib/python3.11/
site-packages (from jinja2->torch) (3.0.2)
Using cached torch-2.7.0-cp311-cp311-manylinux_2_28_x86_64.whl (865.2 MB)
Using cached nvidia_cublas_cu12-12.6.4.1-py3-none-manylinux2014_x86_64.manyl
inux_2_17_x86_64.whl (393.1 MB)
Using cached nvidia_cuda_cupti_cu12-12.6.80-py3-none-manylinux2014_x86_64.ma
nylinux_2_17_x86_64.whl (8.9 MB)
Using cached nvidia_cuda_nvrtc_cu12-12.6.77-py3-none-manylinux2014_x86_64.wh
l (23.7 MB)
Using cached nvidia_cuda_runtime_cu12-12.6.77-py3-none-manylinux2014_x86_64.
manylinux_2_17_x86_64.whl (897 kB)
Using cached nvidia_cudnn_cu12-9.5.1.17-py3-none-manylinux_2_28_x86_64.whl
(571.0 MB)
Using cached nvidia_cufft_cu12-11.3.0.4-py3-none-manylinux2014_x86_64.manyli
nux_2_17_x86_64.whl (200.2 MB)
Using cached nvidia_cufile_cu12-1.11.1.6-py3-none-manylinux2014_x86_64.manyl
inux_2_17_x86_64.whl (1.1 MB)
Using cached nvidia_curand_cu12-10.3.7.77-py3-none-manylinux2014_x86_64.many
linux_2_17_x86_64.whl (56.3 MB)
Using cached nvidia_cusolver_cu12-11.7.1.2-py3-none-manylinux2014_x86_64.man
ylinux_2_17_x86_64.whl (158.2 MB)
Using cached nvidia_cusparse_cu12-12.5.4.2-py3-none-manylinux2014_x86_64.man
ylinux_2_17_x86_64.whl (216.6 MB)
Using cached nvidia_cusparselt_cu12-0.6.3-py3-none-manylinux2014_x86_64.whl
(156.8 MB)
Using cached nvidia_nccl_cu12-2.26.2-py3-none-manylinux2014_x86_64.manylinux
_2_17_x86_64.whl (201.3 MB)
Using cached nvidia_nvjitlink_cu12-12.6.85-py3-none-manylinux2010_x86_64.man
ylinux_2_12_x86_64.whl (19.7 MB)
Using cached nvidia_nvtx_cu12-12.6.77-py3-none-manylinux2014_x86_64.manylinu
x_2_17_x86_64.whl (89 kB)
Using cached triton-3.3.0-cp311-cp311-manylinux_2_27_x86_64.manylinux_2_28_x
86_64.whl (156.5 MB)
Using cached sympy-1.14.0-py3-none-any.whl (6.3 MB)
Using cached networkx-3.4.2-py3-none-any.whl (1.7 MB)
Installing collected packages: nvidia-cusparselt-cu12, triton, sympy, nvidia
-nvtx-cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvi
dia-cufile-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cu
da-cupti-cu12, nvidia-cublas-cu12, networkx, nvidia-cusparse-cu12, nvidia-cu
fft-cu12, nvidia-cudnn-cu12, nvidia-cusolver-cu12, torch
  Attempting uninstall: sympy
    Found existing installation: sympy 1.12
    Uninstalling sympy-1.12:
      Successfully uninstalled sympy-1.12
Successfully installed networkx-3.4.2 nvidia-cublas-cu12-12.6.4.1 nvidia-cud
a-cupti-cu12-12.6.80 nvidia-cuda-nvrtc-cu12-12.6.77 nvidia-cuda-runtime-cu12
-12.6.77 nvidia-cudnn-cu12-9.5.1.17 nvidia-cufft-cu12-11.3.0.4 nvidia-cufile
-cu12-1.11.1.6 nvidia-curand-cu12-10.3.7.77 nvidia-cusolver-cu12-11.7.1.2 nv
idia-cusparse-cu12-12.5.4.2 nvidia-cusparselt-cu12-0.6.3 nvidia-nccl-cu12-2.
```

```
26.2 nvidia-nvjitlink-cu12-12.6.85 nvidia-nvtx-cu12-12.6.77 sympy-1.14.0 tor
ch-2.7.0 triton-3.3.0
Collecting python-dotenv
  Using cached python_dotenv-1.1.0-py3-none-any.whl.metadata (24 kB)
Using cached python_dotenv-1.1.0-py3-none-any.whl (20 kB)
Installing collected packages: python-dotenv
Successfully installed python-dotenv-1.1.0
```

In [2]:
```
#Feel free to import more packages
```

This is all Part 1 for Activity_SCgalactose!

In [3]:
```
#Start off by importing the data

data = pd.read_csv("gcn4-orthologs.csv")
data = data.rename(columns={data.columns[0]: "tile_id"})
data.head()
```

Out[3]:

	tile_id		DNAseq	
0	0	AAAAATTCTAGATTTGTTTCTTTGATGCAAGGATTCTCTGATGGTT...	KNSRFVSLMQGI
1	1	AAAAATTCTCCATCTGGTTTGGCTCATTCTTCTGTTTCTGGTGTTG...	KNSPSGLAHSS
2	2	AAACAAAATTCTCAAGTTGATCAATCTCCATTGTTGCCAGAAGAAG...	KQNSQVDQSPL
3	3	AAACAAAGAGCTACTCCATTGACTCCAGTTGTTCCAGAATCTGATG...	KQRATPLTPVVP
4	4	AAACAAAGATCTATTCCATTGTCTCCAATTGTTCCAGAATCTTCTG...	KQRSIPLSPIVP

5 rows × 35 columns

In [4]:
```
num_rows = data.shape[0]
num_cols = data.shape[1]

print(f"Number of rows: {num_rows}")
print(f"Number of columns: {num_cols}")
```

```
Number of rows: 19038
Number of columns: 35
```

In [5]:
```
read_threshold = 1000 # threshold for read count

df_gal = data[(data['TotalReads_BioRepA_BYS4'] >= read_threshold) & (data['T
print(f"Number of rows before filtering: {num_rows}")
print(f"Number of rows removed: {num_rows - df_gal.shape[0]}")
print(f"Number of rows after filtering: {df_gal.shape[0]}")
```

```
Number of rows before filtering: 19038
Number of rows removed: 9985
Number of rows after filtering: 9053
```

In [6]:
```
activity_col_gal = "Activity_SCgalactose"
df_gal = df_gal[df_gal[activity_col_gal] > 0]

print(f"Number of rows after filtering for non-zero {activity_col_gal}: {df_
```

```
Number of rows after filtering for non-zero Activity_SCgalactose: 8360
```

In [7]:
```python
# Check for duplicates
duplicates_gal = df_gal.duplicated(subset=["tile_id"], keep=False)
print(f"Number of duplicate rows: {duplicates_gal.sum()}")
```

```
Number of duplicate rows: 0
```

In [8]:
```python
df_with_seq_gal = df_gal.copy()
```

In [9]:
```python
# Global constants reused for galactose
AA_LIST = 'ACDEFGHIKLMNPQRSTVWY'
AA_COLS = [f"AA_{aa}" for aa in AA_LIST]

KD_HYDROPHOBICITY = {
    'A': 1.8, 'C': 2.5, 'D': -3.5, 'E': -3.5, 'F': 2.8,
    'G': -0.4, 'H': -3.2, 'I': 4.5, 'K': -3.9, 'L': 3.8,
    'M': 1.9, 'N': -3.5, 'P': -1.6, 'Q': -3.5, 'R': -4.5,
    'S': -0.8, 'T': -0.7, 'V': 4.2, 'W': -0.9, 'Y': -1.3
}

MOTIFS = [
    ("W..LF", "W..LF"),
    ("WFYL_WFYL", "[WFYL]..[WFYL][WFYL]"),
    ("WFL_WFL", "[WFL]..[WFL][WFL]"),
    ("DE_WFY", "[DE][WFY]"),
    ("DE_WF", "[DE][WF]"),
    ("DE_L", "[DE][L]"),
    ("DE_x_WFY", "[DE].[WFY]"),
    ("DE_xx_WFY", "[DE]..[WFY]"),
    ("FF", "FF"),
    ("F_F", "F.F"),
    ("F__F", "F..F"),
    ("FY_5x_FY", "[FY].....[FY]"),
    ("SP", "SP"),
    ("WFYL_3x_WFYL", "[WFYL]...[WFYL][WFYL]")
]

# Feature extraction functions
def amino_acid_count(sequence: str) -> dict[str, int]:
    c = Counter(sequence)
    return {f"AA_{aa}": c.get(aa, 0) for aa in AA_LIST}

def net_charge(seq: str) -> int:
    charge_map = {"K": 1, "R": 1, "D": -1, "E": -1}
    return sum(charge_map.get(a, 0) for a in seq)

def hydrophobicity(sequence: str) -> float:
    return sum(KD_HYDROPHOBICITY.get(aa, 0) for aa in sequence)

def motif_counts_dict(seq: str) -> dict[str, int]:
    return {name: len(re.findall(pat, seq)) for name, pat in MOTIFS}

# Galactose-specific application
def add_sequence_features_gal(df_gal: pd.DataFrame, seq_col: str = "ADseq")
    df_gal = df_gal.copy()
```

```
    # Basic features
    aa_expanded = df_gal[seq_col].apply(amino_acid_count).apply(pd.Series)
    motif_expanded = df_gal[seq_col].apply(motif_counts_dict).apply(pd.Serie

    df_gal["NetCharge"] = df_gal[seq_col].apply(net_charge)
    df_gal["Hydrophobicity"] = df_gal[seq_col].apply(hydrophobicity)
    df_gal["Length"] = df_gal[seq_col].str.len()

    # Combine into one dataframe
    result_df_gal = pd.concat([df_gal, aa_expanded, motif_expanded], axis=1)
    return result_df_gal
```

In [10]:
```
# Apply features to the filtered DataFrame
df_gal = add_sequence_features_gal(df_gal)
df_gal.iloc[:, 35:]
```

Out[10]:

	NetCharge	Hydrophobicity	Length	AA_A	AA_C	AA_D	AA_E	AA_F	AA_G
1	6	-34.0	40	2	0	2	0	0	4
7	1	-24.2	40	3	1	2	1	1	3
8	2	-36.7	40	2	0	3	0	0	2
12	-4	-13.8	40	6	0	4	2	4	3
18	6	-24.4	40	5	0	0	1	2	0
...	...	...	...	...	...	...	...	...	...
19021	-1	-33.9	40	2	0	3	1	4	0
19030	-6	19.9	40	3	1	6	0	3	1
19033	-7	-42.3	40	1	0	5	3	2	2
19036	2	27.8	40	5	2	1	0	2	3
19037	-4	-14.4	40	1	0	4	1	4	2

8360 rows × 37 columns

In [11]:
```
x_gal = df_gal.iloc[:, 35:]
y_gal = df_gal[activity_col_gal]


# now i structure the final prep data
df = pd.concat([x_gal, y_gal], axis=1)

assert df.isna().sum().sum() == 0, "Features contain NaNs"

# quick sanity check for missing values
print("Missing values in final data:")
print(df.isnull().sum())
```

```
Missing values in final data:
NetCharge               0
Hydrophobicity          0
Length                  0
AA_A                    0
AA_C                    0
AA_D                    0
AA_E                    0
AA_F                    0
AA_G                    0
AA_H                    0
AA_I                    0
AA_K                    0
AA_L                    0
AA_M                    0
AA_N                    0
AA_P                    0
AA_Q                    0
AA_R                    0
AA_S                    0
AA_T                    0
AA_V                    0
AA_W                    0
AA_Y                    0
W..LF                   0
WFYL_WFYL               0
WFL_WFL                 0
DE_WFY                  0
DE_WF                   0
DE_L                    0
DE_x_WFY                0
DE_xx_WFY               0
FF                      0
F_F                     0
F__F                    0
FY_5x_FY                0
SP                      0
WFYL_3x_WFYL            0
Activity_SCgalactose    0
dtype: int64
```

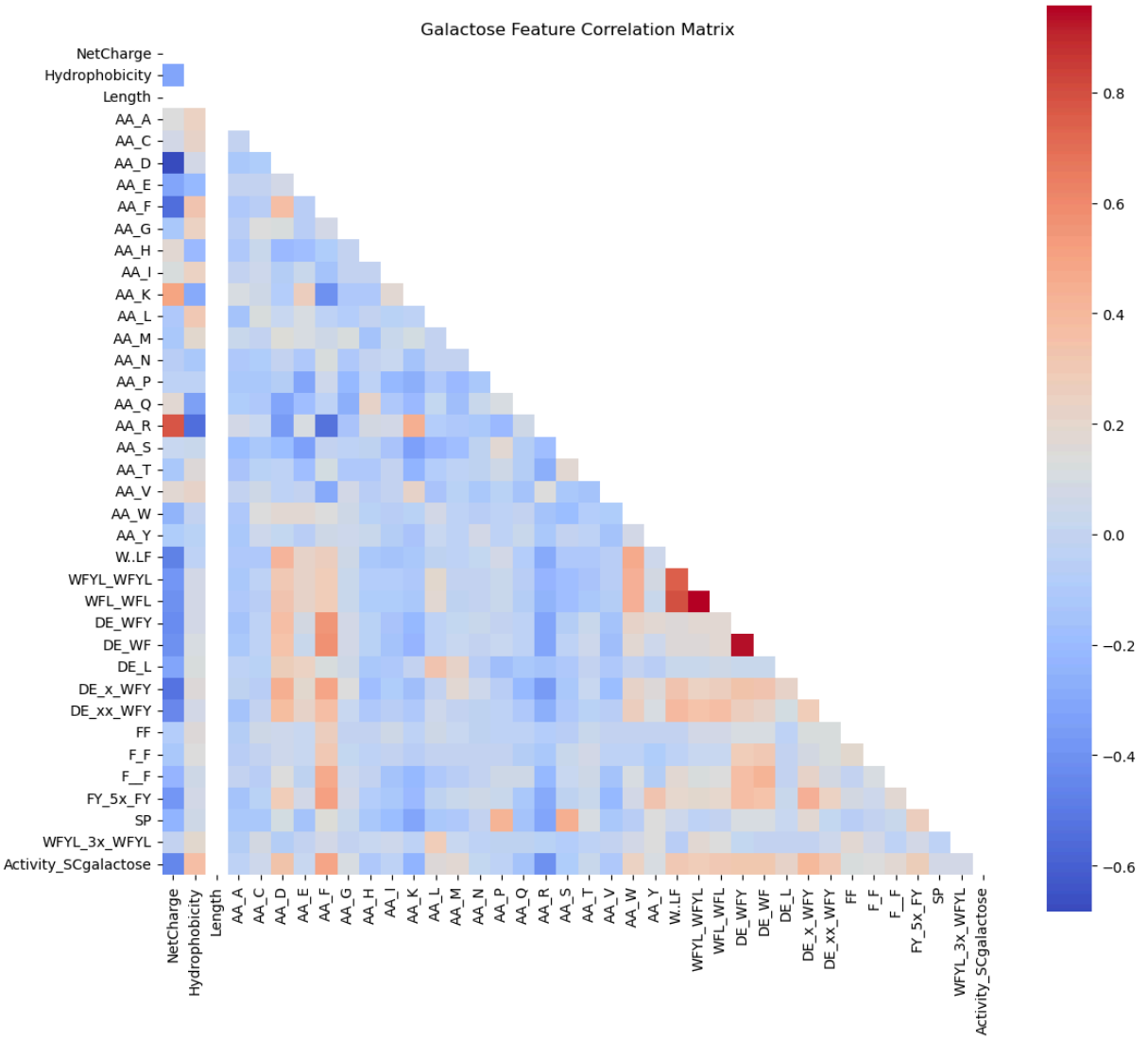
In [12]:
```
df.head()
```

Out[12]:

	NetCharge	Hydrophobicity	Length	AA_A	AA_C	AA_D	AA_E	AA_F	AA_G	AA
1	6	-34.0	40	2	0	2	0	0	4	
7	1	-24.2	40	3	1	2	1	1	3	
8	2	-36.7	40	2	0	3	0	0	2	
12	-4	-13.8	40	6	0	4	2	4	3	
18	6	-24.4	40	5	0	0	1	2	0	

5 rows × 38 columns

In [13]:
```
# Check correlation matrix
plt.figure(figsize=(12, 12))
corr = df.corr()
mask = np.triu(np.ones_like(corr, dtype=bool))
sns.heatmap(corr, mask=mask, cmap="coolwarm", annot=False, fmt=".2f", square
plt.title("Galactose Feature Correlation Matrix")
plt.tight_layout()
#plt.savefig("GCN4_correlation_matrix.png")
plt.show()
```



In [14]:
```
# Get top 10 positive and negative correlations with the target variable
corr_target = df.corr()[activity_col_gal].sort_values(ascending=False)
top_10_positive = corr_target.head(10)
top_10_negative = corr_target.tail(10)
print("Top 10 positive correlations with target:")
print(top_10_positive)

print("="*50)

print("Top 10 negative correlations with target:")
print(top_10_negative)
```

```
Top 10 positive correlations with target:
Activity_SCgalactose    1.000000
AA_F                    0.483265
DE_x_WFY                0.398081
Hydrophobicity          0.363599
AA_D                    0.338413
DE_WFY                  0.320236
DE_WF                   0.318454
W..LF                   0.309696
DE_xx_WFY               0.307296
WFL_WFL                 0.297927
Name: Activity_SCgalactose, dtype: float64
==================================================
Top 10 negative correlations with target:
AA_P        -0.036554
AA_A        -0.041080
AA_S        -0.089480
AA_V        -0.102987
AA_H        -0.106469
AA_Q        -0.160848
AA_K        -0.237128
AA_R        -0.408354
NetCharge   -0.445969
Length            NaN
Name: Activity_SCgalactose, dtype: float64
```

In [15]:
```
x_train_gal, x_test_gal, y_train_gal, y_test_gal = train_test_split(x_gal, y
```

In [16]:
```
linear_reg_gal = make_pipeline(StandardScaler(), LinearRegression())
linear_reg_gal.fit(x_train_gal, y_train_gal)
linear_reg_y_pred_gal = linear_reg_gal.predict(x_test_gal)

print("Linear Regression Results:")
print(f"R^2: {r2_score(y_test_gal, linear_reg_y_pred_gal):.4f}")
print(f"MAE: {mean_absolute_error(y_test_gal, linear_reg_y_pred_gal):.4f}")
print(f"RMSE: {root_mean_squared_error(y_test_gal, linear_reg_y_pred_gal):.4
print(f"Pearson Correlation: {pearsonr(y_test_gal, linear_reg_y_pred_gal)[0]
```

```
Linear Regression Results:
R^2: 0.3900
MAE: 32843.5195
RMSE: 42034.3811
Pearson Correlation: 0.6252
```

In [17]:
```
ridge_reg_gal = make_pipeline(StandardScaler(), Ridge(alpha=1.0))
ridge_reg_gal.fit(x_train_gal, y_train_gal)
ridge_reg_y_pred_gal = ridge_reg_gal.predict(x_test_gal)
print("Ridge Regression Results for SCgalactose:")
print(f"R^2: {r2_score(y_test_gal, ridge_reg_y_pred_gal):.3f}")
print(f"MAE: {mean_absolute_error(y_test_gal, ridge_reg_y_pred_gal):.3f}")
print(f"RMSE: {root_mean_squared_error(y_test_gal, ridge_reg_y_pred_gal):.3f
print(f"Pearson Correlation: {pearsonr(y_test_gal, ridge_reg_y_pred_gal)[0]:
```

```
Ridge Regression Results for SCgalactose:
R^2: 0.390
MAE: 32844.087
RMSE: 42034.659
Pearson Correlation: 0.625
```

In [18]:
```
lasso_reg_gal = make_pipeline(StandardScaler(), Lasso(alpha=1.0, max_iter=10
lasso_reg_gal.fit(x_train_gal, y_train_gal)
lasso_reg_y_pred_gal = lasso_reg_gal.predict(x_test_gal)

print("Lasso Regression Results:")
print(f"R^2: {r2_score(y_test_gal, lasso_reg_y_pred_gal)}")
print(f"MAE: {mean_absolute_error(y_test_gal, lasso_reg_y_pred_gal)}")
print(f"RMSE: {root_mean_squared_error(y_test_gal, lasso_reg_y_pred_gal)}")
print(f"Pearson Correlation: {pearsonr(y_test_gal, lasso_reg_y_pred_gal)[0]}
```

```
Lasso Regression Results:
R^2: 0.39002079335234063
MAE: 32843.99240486016
RMSE: 42034.388873136224
Pearson Correlation: 0.6252302898619648
```

In [19]:
```
grad_boost_reg_gal = make_pipeline(
    StandardScaler(),
    GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, max_depth
)
grad_boost_reg_gal.fit(x_train_gal, y_train_gal)
grad_boost_reg_y_pred_gal = grad_boost_reg_gal.predict(x_test_gal)
print("Gradient Boosting Regressor Results for SCgalactose:")
print(f"R^2: {r2_score(y_test_gal, grad_boost_reg_y_pred_gal):.3f}")
print(f"MAE: {mean_absolute_error(y_test_gal, grad_boost_reg_y_pred_gal):.3f
print(f"RMSE: {root_mean_squared_error(y_test_gal, grad_boost_reg_y_pred_gal
print(f"Pearson Correlation: {pearsonr(y_test_gal, grad_boost_reg_y_pred_gal
```

```
Gradient Boosting Regressor Results for SCgalactose:
R^2: 0.408
MAE: 32322.851
RMSE: 41409.985
Pearson Correlation: 0.639
```

In [20]:
```
elastic_reg_gal = make_pipeline(StandardScaler(), ElasticNet(alpha=1.0, l1_r
elastic_reg_gal.fit(x_train_gal, y_train_gal)
elastic_reg_y_pred_gal = elastic_reg_gal.predict(x_test_gal)

print("Elastic Net Regression Results:")
print(f"R^2: {r2_score(y_test_gal, elastic_reg_y_pred_gal)}")
print(f"MAE: {mean_absolute_error(y_test_gal, elastic_reg_y_pred_gal)}")
print(f"RMSE: {root_mean_squared_error(y_test_gal, elastic_reg_y_pred_gal)}"
print(f"Pearson Correlation: {pearsonr(y_test_gal, elastic_reg_y_pred_gal)[0
```

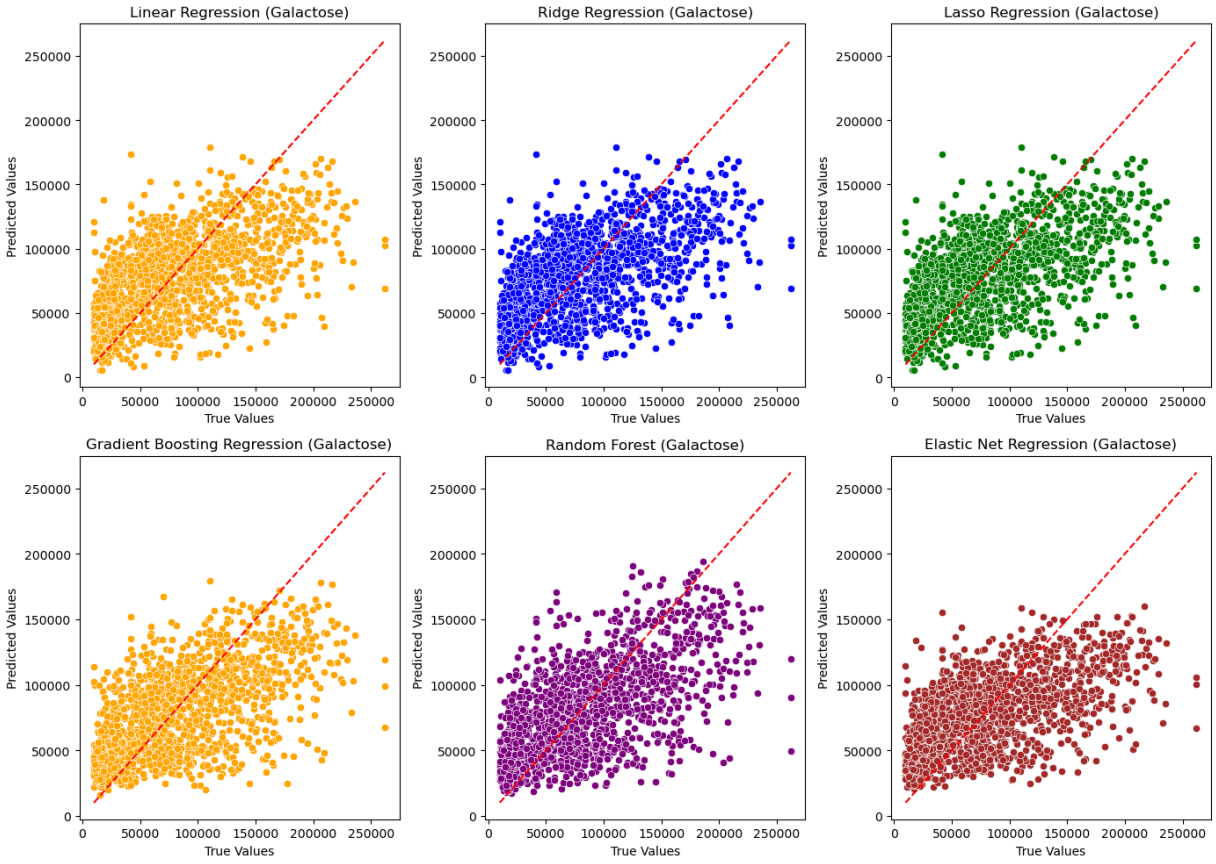
```
Elastic Net Regression Results:
R^2: 0.37084742609639665
MAE: 34093.24945767738
RMSE: 42689.9072424606
Pearson Correlation: 0.6131014048525667
```

In [21]:
```
random_forest_reg_gal = make_pipeline(
    StandardScaler(),
    RandomForestRegressor(n_estimators=100, random_state=42)
)
random_forest_reg_gal.fit(x_train_gal, y_train_gal)
random_forest_y_pred_gal = random_forest_reg_gal.predict(x_test_gal)

print("Random Forest Regressor Results for SCgalactose:")
print(f"R^2: {r2_score(y_test_gal, random_forest_y_pred_gal):.3f}")
print(f"MAE: {mean_absolute_error(y_test_gal, random_forest_y_pred_gal):.3f}
print(f"RMSE: {root_mean_squared_error(y_test_gal, random_forest_y_pred_gal)
print(f"Pearson Correlation: {pearsonr(y_test_gal, random_forest_y_pred_gal)
```

```
Random Forest Regressor Results for SCgalactose:
R^2: 0.429
MAE: 31196.194
RMSE: 40673.917
Pearson Correlation: 0.655
```

In [22]:
```
# Make subplots for each model on SCgalactose
fig, axes = plt.subplots(2, 3, figsize=(14, 10))
axes = axes.flatten()

# Ridge Regression
sns.scatterplot(x=y_test_gal, y=ridge_reg_y_pred_gal, ax=axes[1], color='blu
axes[1].set_title("Ridge Regression (Galactose)")
axes[1].set_xlabel("True Values")
axes[1].set_ylabel("Predicted Values")
sns.lineplot(x=y_test_gal, y=y_test_gal, ax=axes[1], color='red', linestyle=

# Lasso Regression
sns.scatterplot(x=y_test_gal, y=lasso_reg_y_pred_gal , ax=axes[2], color='gr
axes[2].set_title("Lasso Regression (Galactose)")
axes[2].set_xlabel("True Values")
axes[2].set_ylabel("Predicted Values")
sns.lineplot(x=y_test_gal, y=y_test_gal, ax=axes[2], color='red', linestyle=

# Gradient
sns.scatterplot(x=y_test_gal, y=grad_boost_reg_y_pred_gal, ax=axes[3], color
axes[3].set_title("Gradient Boosting Regression (Galactose)")
axes[3].set_xlabel("True Values")
axes[3].set_ylabel("Predicted Values")
sns.lineplot(x=y_test_gal, y=y_test_gal, ax=axes[3], color='red', linestyle=

# Linear Regression
sns.scatterplot(x=y_test_gal, y=linear_reg_y_pred_gal, ax=axes[0], color='or
axes[0].set_title("Linear Regression (Galactose)")
axes[0].set_xlabel("True Values")
axes[0].set_ylabel("Predicted Values")
sns.lineplot(x=y_test_gal, y=y_test_gal, ax=axes[0], color='red', linestyle=

# Random Forest
sns.scatterplot(x=y_test_gal, y=random_forest_y_pred_gal, ax=axes[4], color=
axes[4].set_title("Random Forest (Galactose)")
axes[4].set_xlabel("True Values")
axes[4].set_ylabel("Predicted Values")
```

```
sns.lineplot(x=y_test_gal, y=y_test_gal, ax=axes[4], color='red', linestyle=

# Elastic Net Regression
sns.scatterplot(x=y_test_gal, y=elastic_reg_y_pred_gal, ax=axes[5], color='b
axes[5].set_title("Elastic Net Regression (Galactose)")
axes[5].set_xlabel("True Values")
axes[5].set_ylabel("Predicted Values")
sns.lineplot(x=y_test_gal, y=y_test_gal, ax=axes[5], color='red', linestyle=

plt.tight_layout()
plt.show()
```



In [23]:
```
# Make df of results
results = {
    "Model": ["Linear (for Galactose)", "Ridge (for Galactose)", "Lasso (for
    "R^2": [
        r2_score(y_test_gal, linear_reg_y_pred_gal),
        r2_score(y_test_gal, ridge_reg_y_pred_gal),
        r2_score(y_test_gal, lasso_reg_y_pred_gal),
        r2_score(y_test_gal, grad_boost_reg_y_pred_gal),
        r2_score(y_test_gal, random_forest_y_pred_gal),
        r2_score(y_test_gal, elastic_reg_y_pred_gal)
    ],
    "MAE": [
        mean_absolute_error(y_test_gal, linear_reg_y_pred_gal),
        mean_absolute_error(y_test_gal, ridge_reg_y_pred_gal),
        mean_absolute_error(y_test_gal, lasso_reg_y_pred_gal),
        mean_absolute_error(y_test_gal, grad_boost_reg_y_pred_gal),
        mean_absolute_error(y_test_gal, random_forest_y_pred_gal),
        mean_absolute_error(y_test_gal, elastic_reg_y_pred_gal)
```

```
    ],
    "RMSE": [
        root_mean_squared_error(y_test_gal, linear_reg_y_pred_gal),
        root_mean_squared_error(y_test_gal, ridge_reg_y_pred_gal),
        root_mean_squared_error(y_test_gal, lasso_reg_y_pred_gal),
        root_mean_squared_error(y_test_gal, grad_boost_reg_y_pred_gal),
        root_mean_squared_error(y_test_gal, random_forest_y_pred_gal),
        root_mean_squared_error(y_test_gal, elastic_reg_y_pred_gal)
    ],
    "Pearson Correlation": [
        pearsonr(y_test_gal, linear_reg_y_pred_gal)[0],
        pearsonr(y_test_gal, ridge_reg_y_pred_gal)[0],
        pearsonr(y_test_gal, lasso_reg_y_pred_gal)[0],
        pearsonr(y_test_gal, grad_boost_reg_y_pred_gal)[0],
        pearsonr(y_test_gal, random_forest_y_pred_gal)[0],
        pearsonr(y_test_gal, elastic_reg_y_pred_gal)[0]
    ]
}
results_df = pd.DataFrame(results)
results_df = results_df.sort_values(by="R^2", ascending=False).reset_index(d
results_df.head(6)
```

Out[23]:

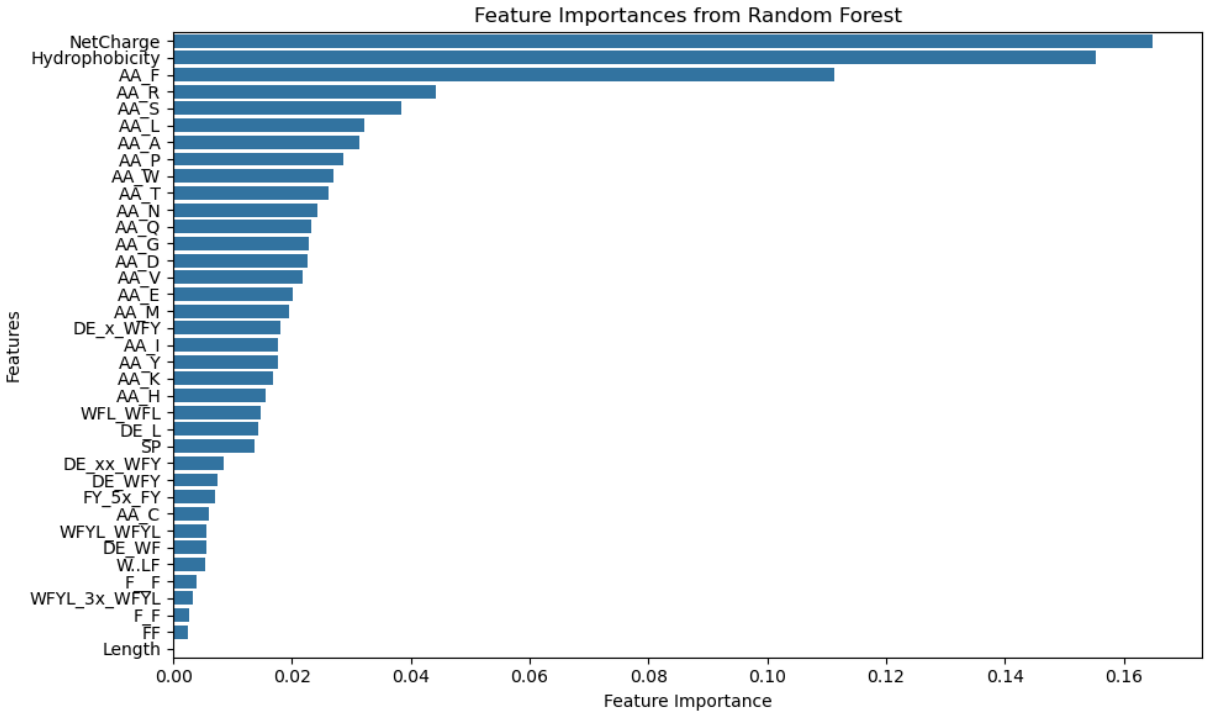
	Model	R^2	MAE	RMSE	Pearson Correlation
0	Random Forest (for Galactose)	0.428867	31196.193562	40673.917323	0.655263
1	Gradient Boosting (for Galactose)	0.408008	32322.850579	41409.985053	0.639404
2	Linear (for Galactose)	0.390021	32843.519523	42034.381059	0.625230
3	Lasso (for Galactose)	0.390021	32843.992405	42034.388873	0.625230
4	Ridge (for Galactose)	0.390013	32844.087143	42034.659486	0.625224
5	Elastic Net (for Galactose)	0.370847	34093.249458	42689.907242	0.613101

In [24]:
```
import os

# Create the directory if it doesn't exist
os.makedirs("./models", exist_ok=True)

# Save the model
joblib.dump(random_forest_reg_gal, "./models/random_forest_gal_model.pkl")
```

Out[24]:
```
['./models/random_forest_gal_model.pkl']
```

In [25]:
```
feature_importances = random_forest_reg_gal.named_steps['randomforestregress
feature_names = x_gal.columns
sorted_indices = np.argsort(feature_importances)[::-1]
sorted_feature_importances = feature_importances[sorted_indices]
sorted_feature_names = feature_names[sorted_indices]
plt.figure(figsize=(10, 6))
sns.barplot(x=sorted_feature_importances, y=sorted_feature_names)
```

```
plt.xlabel("Feature Importance")
plt.ylabel("Features")
plt.title("Feature Importances from Random Forest")
plt.tight_layout()
#plt.savefig("GCN4_feature_importances.png")
plt.show()
```



In [26]:

```
# Load the .env file
load_dotenv()
hf_token = os.getenv("HF_TOKEN")

# Load model and tokenizer once
model_id = "facebook/esm2_t6_8M_UR50D"
tokenizer = AutoTokenizer.from_pretrained(model_id, token=hf_token)
model = AutoModel.from_pretrained(model_id, token=hf_token)
model.eval()

def generate_embedding(sequence):
    """Generate pooled ESM embedding for a protein sequence."""
    input_ids = tokenizer(sequence, return_tensors="pt").input_ids
    attention_mask = torch.ones_like(input_ids)

    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        sequence_embeddings = outputs.last_hidden_state.squeeze()

    # Remove special tokens [CLS], [EOS]
    sequence_embeddings = sequence_embeddings[1:-1]

    # Mean-pool across residues
    pooled_embedding = sequence_embeddings.mean(dim=0)  # shape: (hidden_siz
    return pooled_embedding.cpu().numpy()

def save_embeddings_to_npz(data, sequence_column, save_path):
```

```
    """Generate embeddings for all sequences and save into a single NPZ file
    all_embeddings = []

    for sequence in tqdm(data[sequence_column], desc="Generating embeddings"
        embedding = generate_embedding(sequence)
        all_embeddings.append(embedding)

    all_embeddings = np.stack(all_embeddings)  # Shape: (num_sequences, hidd

    # Ensure the output directory exists
    os.makedirs(os.path.dirname(save_path), exist_ok=True)

    # Save the embeddings
    np.savez_compressed(save_path, embeddings=all_embeddings)
    print(f"Saved {len(all_embeddings)} embeddings to {save_path}")

# Comment out when not needed
save_embeddings_to_npz(df_with_seq_gal, sequence_column="ADseq", save_path="
```

```
Some weights of EsmModel were not initialized from the model checkpoint at f
acebook/esm2_t6_8M_UR50D and are newly initialized: ['pooler.dense.bias', 'p
ooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
Generating embeddings: 100%|██████████| 8360/8360 [02:08<00:00, 65.22it/s]
Saved 8360 embeddings to ./galactose_esm2_embeddings.npz
```

In [27]:
```
# Load the embeddings from the NPZ file
embeddings_data = np.load("./galactose_esm2_embeddings.npz")
embeddings_gal = embeddings_data["embeddings"]

# Check the shape of the loaded embeddings
print(f"Loaded embeddings shape: {embeddings_gal.shape}")
```

```
Loaded embeddings shape: (8360, 320)
```

In [28]:
```
# Prep embededings into a dataframe for feature concatenation
embeddings_df_gal = pd.DataFrame(embeddings_gal)
embeddings_df_gal.columns = [f"embedding_{i}" for i in range(embeddings_df_g
embeddings_df_gal.head(3)
```

Out[28]:

	embedding_0	embedding_1	embedding_2	embedding_3	embedding_4	embeddir
0	-0.025192	-0.386163	0.038171	0.142584	0.045324	-0.185
1	0.014334	-0.416917	0.232792	0.121774	-0.028585	-0.204
2	-0.005063	-0.397236	-0.005174	0.114196	0.172707	-0.098

3 rows × 320 columns

In [29]:
```
# the assert below is for me to chek for dimension mismatch
assert embeddings_df_gal.shape[0] == df_gal.shape[0], "Mismatch in number of

# Concatenate the embeddings with the original DataFrame
x_esm_gal = pd.concat([x_gal.reset_index(drop=True),
```

```
                    embeddings_df_gal.reset_index(drop=True)],
                   axis=1)
y_esm_gal = y_gal.copy()

print(f"X_esm shape: {x_esm_gal.shape}")
print(f"y_esm shape: {y_esm_gal.shape}")
```

```
X_esm shape: (8360, 357)
y_esm shape: (8360,)
```

In [30]:
```
x_esm_train_gal, x_esm_test_gal, y_esm_train_gal, y_esm_test_gal = train_tes
```

In [31]:
```
esm_linear_reg_gal = make_pipeline(StandardScaler(), LinearRegression())
esm_linear_reg_gal.fit(x_esm_train_gal, y_esm_train_gal)
esm_linear_reg_y_pred_gal = esm_linear_reg_gal.predict(x_esm_test_gal)

print("Linear Regression with ESM Embeddings Results:")
print(f"R^2: {r2_score(y_esm_test_gal, esm_linear_reg_y_pred_gal)}")
print(f"MAE: {mean_absolute_error(y_esm_test_gal, esm_linear_reg_y_pred_gal)
print(f"RMSE: {root_mean_squared_error(y_esm_test_gal, esm_linear_reg_y_prec
print(f"Pearson Correlation: {pearsonr(y_esm_test_gal, esm_linear_reg_y_prec
```

```
Linear Regression with ESM Embeddings Results:
R^2: 0.23537451368139295
MAE: 35825.27268704991
RMSE: 47062.133037609936
Pearson Correlation: 0.5585328051488605
```

In [32]:
```
esm_ridge_reg_gal = make_pipeline(StandardScaler(), Ridge(alpha=1.0))
esm_ridge_reg_gal.fit(x_esm_train_gal, y_esm_train_gal)
esm_ridge_reg_y_pred_gal = esm_ridge_reg_gal.predict(x_esm_test_gal)

# Just print for now
print("Ridge Regression Results (Galactose):")
print(f"R^2: {r2_score(y_esm_test_gal, esm_ridge_reg_y_pred_gal)}")
print(f"MAE: {mean_absolute_error(y_esm_test_gal, esm_ridge_reg_y_pred_gal)}
print(f"RMSE: {root_mean_squared_error(y_esm_test_gal, esm_ridge_reg_y_pred_
print(f"Pearson Correlation: {pearsonr(y_esm_test_gal, esm_ridge_reg_y_pred_
```

```
Ridge Regression Results (Galactose):
R^2: 0.4182259224695186
MAE: 31288.16019395661
RMSE: 41051.06280700234
Pearson Correlation: 0.6510301009021788
```

In [33]:
```
# Lasso Regression
esm_lasso_reg_gal = make_pipeline(StandardScaler(), Lasso(alpha=0.1))
esm_lasso_reg_gal.fit(x_esm_train_gal, y_esm_train_gal)
esm_lasso_reg_y_pred_gal = esm_lasso_reg_gal.predict(x_esm_test_gal)

print("Lasso Regression Results (Galactose):")
print(f"R^2: {r2_score(y_esm_test_gal, esm_lasso_reg_y_pred_gal)}")
print(f"MAE: {mean_absolute_error(y_esm_test_gal, esm_lasso_reg_y_pred_gal)}
print(f"RMSE: {root_mean_squared_error(y_esm_test_gal, esm_lasso_reg_y_pred_
print(f"Pearson Correlation: {pearsonr(y_esm_test_gal, esm_lasso_reg_y_pred_
```

```
Lasso Regression Results (Galactose):
R^2: 0.41798476734924805
MAE: 31299.851558059727
RMSE: 41059.57010277924
Pearson Correlation: 0.6509565711131717
```

```
/srv/conda/lib/python3.11/site-packages/sklearn/linear_model/_coordinate_des
cent.py:678: ConvergenceWarning: Objective did not converge. You might want
to increase the number of iterations, check the scale of the features or con
sider increasing regularisation. Duality gap: 4.921e+12, tolerance: 2.000e+0
9
  model = cd_fast.enet_coordinate_descent(
```

In [34]:
```
# Gradient Boosting Regression
esm_grad_boost_reg_gal = GradientBoostingRegressor(n_estimators=100, random_
esm_grad_boost_reg_gal.fit(x_esm_train_gal, y_esm_train_gal)
esm_grad_boost_reg_y_pred_gal = esm_grad_boost_reg_gal.predict(x_esm_test_ga

print("Gradient Boosting Regression Results (Galactose):")
print(f"R^2: {r2_score(y_esm_test_gal, esm_grad_boost_reg_y_pred_gal)}")
print(f"MAE: {mean_absolute_error(y_esm_test_gal, esm_grad_boost_reg_y_pred_
print(f"RMSE: {root_mean_squared_error(y_esm_test_gal, esm_grad_boost_reg_y_
print(f"Pearson Correlation: {pearsonr(y_esm_test_gal, esm_grad_boost_reg_y_
```

```
Gradient Boosting Regression Results (Galactose):
R^2: 0.41992196613836785
MAE: 31707.275120974387
RMSE: 40991.181131368074
Pearson Correlation: 0.6487077764125251
```

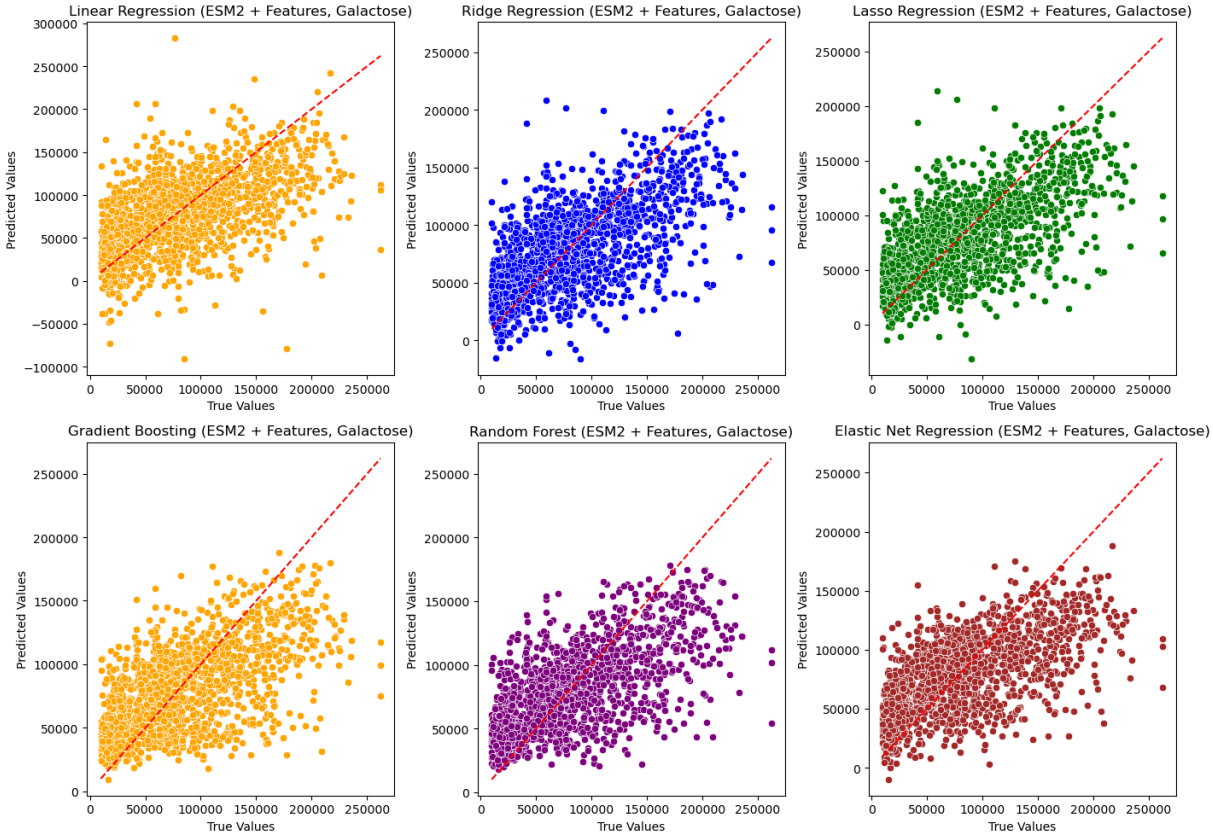
In [35]:
```
esm_elastic_reg_gal = make_pipeline(StandardScaler(), ElasticNet(alpha=1.0,
esm_elastic_reg_gal.fit(x_esm_train_gal, y_esm_train_gal)
esm_elastic_reg_y_pred_gal = esm_elastic_reg_gal.predict(x_esm_test_gal)

print("Elastic Net Regression Results:")
print(f"R^2: {r2_score(y_esm_test_gal, esm_elastic_reg_y_pred_gal)}")
print(f"MAE: {mean_absolute_error(y_esm_test_gal, esm_elastic_reg_y_pred_gal
print(f"RMSE: {root_mean_squared_error(y_esm_test_gal, esm_elastic_reg_y_pre
print(f"Pearson Correlation: {pearsonr(y_esm_test_gal, esm_elastic_reg_y_pre
```

```
Elastic Net Regression Results:
R^2: 0.4071577718770595
MAE: 32442.134869951667
RMSE: 41439.71765695648
Pearson Correlation: 0.639422269775672
```

In [36]:
```
# Random Forest Regression
esm_random_forest_reg_gal = RandomForestRegressor(n_estimators=100, random_s
esm_random_forest_reg_gal.fit(x_esm_train_gal, y_esm_train_gal)
esm_random_forest_y_pred_gal = esm_random_forest_reg_gal.predict(x_esm_test_

print("Random Forest Regression Results (Galactose):")
print(f"R^2: {r2_score(y_esm_test_gal, esm_random_forest_y_pred_gal)}")
print(f"MAE: {mean_absolute_error(y_esm_test_gal, esm_random_forest_y_pred_g
print(f"RMSE: {root_mean_squared_error(y_esm_test_gal, esm_random_forest_y_p
print(f"Pearson Correlation: {pearsonr(y_esm_test_gal, esm_random_forest_y_p
```

```
Random Forest Regression Results (Galactose):
R^2: 0.4331951856584767
MAE: 31497.89122649578
RMSE: 40519.491592854116
Pearson Correlation: 0.6593070432146668
```

In [37]:
```
# Make subplots for each model (Galactose)
fig, axes = plt.subplots(2, 3, figsize=(14, 10))
axes = axes.flatten()

# Linear Regression
sns.scatterplot(x=y_esm_test_gal, y=esm_linear_reg_y_pred_gal, ax=axes[0], c
axes[0].set_title("Linear Regression (ESM2 + Features, Galactose)")
axes[0].set_xlabel("True Values")
axes[0].set_ylabel("Predicted Values")
sns.lineplot(x=y_esm_test_gal, y=y_esm_test_gal, ax=axes[0], color='red', li

# Ridge Regression
sns.scatterplot(x=y_esm_test_gal, y=esm_ridge_reg_y_pred_gal, ax=axes[1], co
axes[1].set_title("Ridge Regression (ESM2 + Features, Galactose)")
axes[1].set_xlabel("True Values")
axes[1].set_ylabel("Predicted Values")
sns.lineplot(x=y_esm_test_gal, y=y_esm_test_gal, ax=axes[1], color='red', li

# Lasso Regression
sns.scatterplot(x=y_esm_test_gal, y=esm_lasso_reg_y_pred_gal, ax=axes[2], co
axes[2].set_title("Lasso Regression (ESM2 + Features, Galactose)")
axes[2].set_xlabel("True Values")
axes[2].set_ylabel("Predicted Values")
sns.lineplot(x=y_esm_test_gal, y=y_esm_test_gal, ax=axes[2], color='red', li

# Gradient Boosting Regression
sns.scatterplot(x=y_esm_test_gal, y=esm_grad_boost_reg_y_pred_gal, ax=axes[3
axes[3].set_title("Gradient Boosting (ESM2 + Features, Galactose)")
axes[3].set_xlabel("True Values")
axes[3].set_ylabel("Predicted Values")
sns.lineplot(x=y_esm_test_gal, y=y_esm_test_gal, ax=axes[3], color='red', li

# Random Forest Regression
sns.scatterplot(x=y_esm_test_gal, y=esm_random_forest_y_pred_gal, ax=axes[4]
axes[4].set_title("Random Forest (ESM2 + Features, Galactose)")
axes[4].set_xlabel("True Values")
axes[4].set_ylabel("Predicted Values")
sns.lineplot(x=y_esm_test_gal, y=y_esm_test_gal, ax=axes[4], color='red', li

# Elastic Net Regression
sns.scatterplot(x=y_esm_test_gal, y=esm_elastic_reg_y_pred_gal, ax=axes[5],
axes[5].set_title("Elastic Net Regression (ESM2 + Features, Galactose)")
axes[5].set_xlabel("True Values")
axes[5].set_ylabel("Predicted Values")
sns.lineplot(x=y_esm_test_gal, y=y_esm_test_gal, ax=axes[5], color='red', li

plt.tight_layout()
plt.show()
```



In [38]:
```
# Make df of results for Galactose
esm_results_gal = {
    "Model": ["Linear (ESM for Galactose)", "Ridge (ESM for Galactose)", "La
    "R^2": [
        r2_score(y_esm_test_gal, esm_linear_reg_y_pred_gal),
        r2_score(y_esm_test_gal, esm_ridge_reg_y_pred_gal),
        r2_score(y_esm_test_gal, esm_lasso_reg_y_pred_gal),
        r2_score(y_esm_test_gal, esm_grad_boost_reg_y_pred_gal),
        r2_score(y_esm_test_gal, esm_random_forest_y_pred_gal),
        r2_score(y_esm_test_gal, esm_elastic_reg_y_pred_gal)
    ],
    "MAE": [
        mean_absolute_error(y_esm_test_gal, esm_linear_reg_y_pred_gal),
        mean_absolute_error(y_esm_test_gal, esm_ridge_reg_y_pred_gal),
        mean_absolute_error(y_esm_test_gal, esm_lasso_reg_y_pred_gal),
        mean_absolute_error(y_esm_test_gal, esm_grad_boost_reg_y_pred_gal),
        mean_absolute_error(y_esm_test_gal, esm_random_forest_y_pred_gal),
        mean_absolute_error(y_esm_test_gal, esm_elastic_reg_y_pred_gal)
    ],
    "RMSE": [
        root_mean_squared_error(y_esm_test_gal, esm_linear_reg_y_pred_gal),
        root_mean_squared_error(y_esm_test_gal, esm_ridge_reg_y_pred_gal),
        root_mean_squared_error(y_esm_test_gal, esm_lasso_reg_y_pred_gal),
        root_mean_squared_error(y_esm_test_gal, esm_grad_boost_reg_y_pred_ga
        root_mean_squared_error(y_esm_test_gal, esm_random_forest_y_pred_gal
        root_mean_squared_error(y_esm_test_gal, esm_elastic_reg_y_pred_gal)
    ],
    "Pearson Correlation": [
        pearsonr(y_esm_test_gal, esm_linear_reg_y_pred_gal)[0],
        pearsonr(y_esm_test_gal, esm_ridge_reg_y_pred_gal)[0],
```

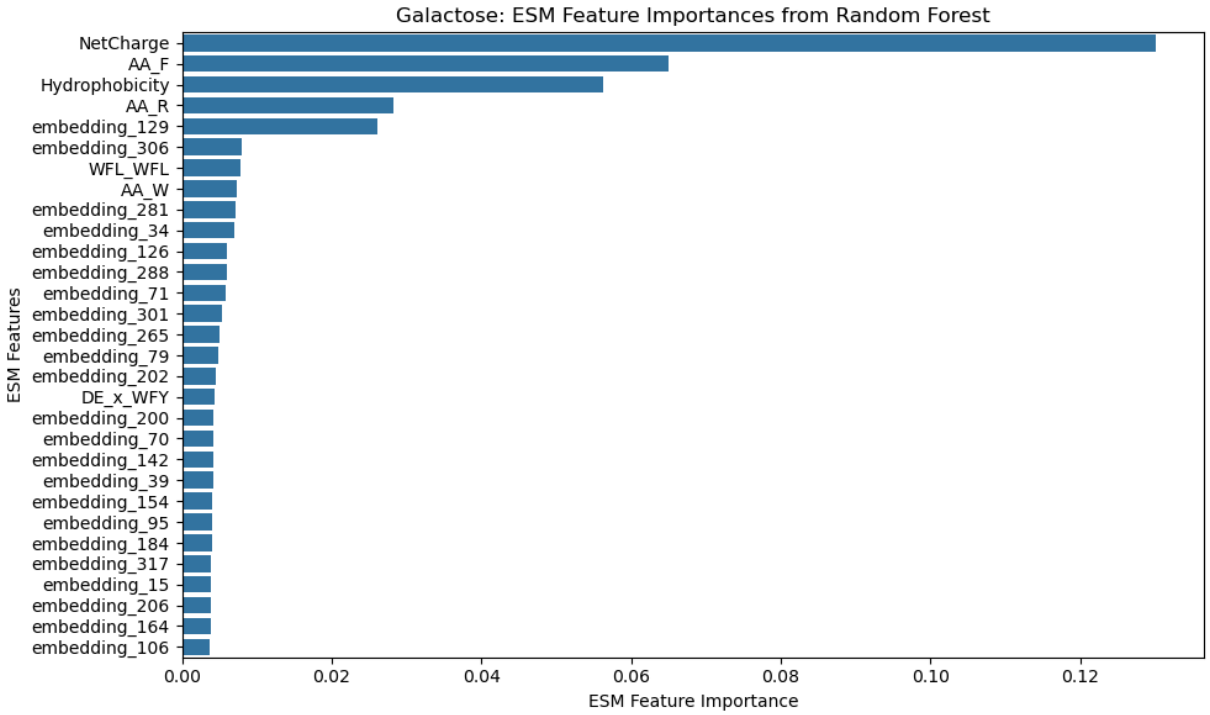
```
        pearsonr(y_esm_test_gal, esm_lasso_reg_y_pred_gal)[0],
        pearsonr(y_esm_test_gal, esm_grad_boost_reg_y_pred_gal)[0],
        pearsonr(y_esm_test_gal, esm_random_forest_y_pred_gal)[0],
        pearsonr(y_esm_test_gal, esm_elastic_reg_y_pred_gal)[0]
    ]
}
esm_results_df_gal = pd.DataFrame(esm_results_gal)
esm_results_df_gal = esm_results_df_gal.sort_values(by="R^2", ascending=Fals
esm_results_df_gal.head(6)
```

Out[38]:

	Model	R^2	MAE	RMSE	Pearson Correlation
0	Random Forest (ESM for Galactose)	0.433195	31497.891226	40519.491593	0.659307
1	Gradient Boosting (ESM for Galactose)	0.419922	31707.275121	40991.181131	0.648708
2	Ridge (ESM for Galactose)	0.418226	31288.160194	41051.062807	0.651030
3	Lasso (ESM for Galactose)	0.417985	31299.851558	41059.570103	0.650957
4	Elastic Net (ESM for Galactose)	0.407158	32442.134870	41439.717657	0.639422
5	Linear (ESM for Galactose)	0.235375	35825.272687	47062.133038	0.558533

In [39]:

```
# Get feature importances from galactose Random Forest model
esm_feature_importances_gal = esm_random_forest_reg_gal.feature_importances_
esm_feature_names_gal = x_esm_gal.columns
esm_sorted_indices_gal = np.argsort(esm_feature_importances_gal)[::-1]

top_k = 30

sorted_esm_feature_importances_gal = esm_feature_importances_gal[esm_sorted_
sorted_esm_feature_names_gal = esm_feature_names_gal[esm_sorted_indices_gal]

# Plot feature importances
plt.figure(figsize=(10, 6))
sns.barplot(x=sorted_esm_feature_importances_gal, y=sorted_esm_feature_names
plt.xlabel("ESM Feature Importance")
plt.ylabel("ESM Features")
plt.title("Galactose: ESM Feature Importances from Random Forest")
plt.tight_layout()
plt.show()
```



In [40]:
```
import joblib
joblib.dump(esm_random_forest_reg_gal, "./models/esm_random_forest_model_gal
```

Out[40]:
```
['./models/esm_random_forest_model_gal.pkl']
```

In [41]:
```
x_only_embeddings_gal = embeddings_df_gal.copy()
y_only_embeddings_gal = y_gal.copy()
```

In [42]:
```
x_only_embeddings_train_gal, x_only_embeddings_test_gal, y_only_embeddings_t
    x_only_embeddings_gal, y_only_embeddings_gal, test_size=0.2, random_stat
```

In [43]:
```
# Linear Regression
linear_reg_embeddings_gal = make_pipeline(StandardScaler(), LinearRegressior
linear_reg_embeddings_gal.fit(x_only_embeddings_train_gal, y_only_embeddings
linear_reg_embeddings_y_pred_gal = linear_reg_embeddings_gal.predict(x_only_
print("Linear Regression with ESM Embeddings Results:")
print(f"R^2: {r2_score(y_only_embeddings_test_gal, linear_reg_embeddings_y_p
print(f"MAE: {mean_absolute_error(y_only_embeddings_test_gal, linear_reg_emb
print(f"RMSE: {root_mean_squared_error(y_only_embeddings_test_gal, linear_re
print(f"Pearson Correlation: {pearsonr(y_only_embeddings_test_gal, linear_re
```

```
Linear Regression with ESM Embeddings Results:
R^2: 0.4110527533062649
MAE: 31659.84178433523
RMSE: 41303.36357407488
Pearson Correlation: 0.6459323314193213
```

In [44]:
```
# Ridge Regression for Galactose
ridge_reg_embeddings_gal = make_pipeline(StandardScaler(), Ridge(alpha=1.0))
ridge_reg_embeddings_gal.fit(x_only_embeddings_train_gal, y_only_embeddings_
ridge_reg_embeddings_gal_y_pred = ridge_reg_embeddings_gal.predict(x_only_em

print("Ridge Regression Results (Only ESM2 for Galactose):")
```

```
print(f"R^2: {r2_score(y_only_embeddings_test_gal, ridge_reg_embeddings_gal_
print(f"MAE: {mean_absolute_error(y_only_embeddings_test_gal, ridge_reg_embe
print(f"RMSE: {root_mean_squared_error(y_only_embeddings_test_gal, ridge_reg
print(f"Pearson Correlation: {pearsonr(y_only_embeddings_test_gal, ridge_reg
```

```
Ridge Regression Results (Only ESM2 for Galactose):
R^2: 0.4118090362051533
MAE: 31641.686909614087
RMSE: 41276.835678552525
Pearson Correlation: 0.6463498541257227
```

In [45]:
```
# Lasso Regression for Galactose
lasso_reg_embeddings_gal = make_pipeline(StandardScaler(), Lasso(alpha=1.0,
lasso_reg_embeddings_gal.fit(x_only_embeddings_train_gal, y_only_embeddings_
lasso_reg_embeddings_gal_y_pred = lasso_reg_embeddings_gal.predict(x_only_em

print("Lasso Regression Results (Only ESM2 for Galactose):")
print(f"R^2: {r2_score(y_only_embeddings_test_gal, lasso_reg_embeddings_gal_
print(f"MAE: {mean_absolute_error(y_only_embeddings_test_gal, lasso_reg_embe
print(f"RMSE: {root_mean_squared_error(y_only_embeddings_test_gal, lasso_reg
print(f"Pearson Correlation: {pearsonr(y_only_embeddings_test_gal, lasso_reg
```

```
Lasso Regression Results (Only ESM2 for Galactose):
R^2: 0.4116212790431727
MAE: 31646.29983044094
RMSE: 41283.4231676715
Pearson Correlation: 0.6462505196322523
```

In [46]:
```
# Gradient Boosting for Galactose
grad_boost_reg_embeddings_gal = make_pipeline(StandardScaler(), GradientBoos
grad_boost_reg_embeddings_gal.fit(x_only_embeddings_train_gal, y_only_embedd
grad_boost_reg_embeddings_gal_y_pred = grad_boost_reg_embeddings_gal.predict

print("Gradient Boosting Regressor Results (Only ESM2 for Galactose):")
print(f"R^2: {r2_score(y_only_embeddings_test_gal, grad_boost_reg_embeddings
print(f"MAE: {mean_absolute_error(y_only_embeddings_test_gal, grad_boost_reg
print(f"RMSE: {root_mean_squared_error(y_only_embeddings_test_gal, grad_boos
print(f"Pearson Correlation: {pearsonr(y_only_embeddings_test_gal, grad_boos
```

```
Gradient Boosting Regressor Results (Only ESM2 for Galactose):
R^2: 0.40962520299281513
MAE: 32104.871240031105
RMSE: 41353.39092650614
Pearson Correlation: 0.6409018746424546
```

In [47]:
```
# Random Forest Regression for Galactose
random_forest_reg_embeddings_gal = make_pipeline(StandardScaler(), RandomFor
random_forest_reg_embeddings_gal.fit(x_only_embeddings_train_gal, y_only_emb
random_forest_reg_embeddings_gal_y_pred = random_forest_reg_embeddings_gal.p

print("Random Forest Regressor Results (Only ESM2 for Galactose):")
print(f"R^2: {r2_score(y_only_embeddings_test_gal, random_forest_reg_embeddi
print(f"MAE: {mean_absolute_error(y_only_embeddings_test_gal, random_forest_
print(f"RMSE: {root_mean_squared_error(y_only_embeddings_test_gal, random_fo
print(f"Pearson Correlation: {pearsonr(y_only_embeddings_test_gal, random_fo
```

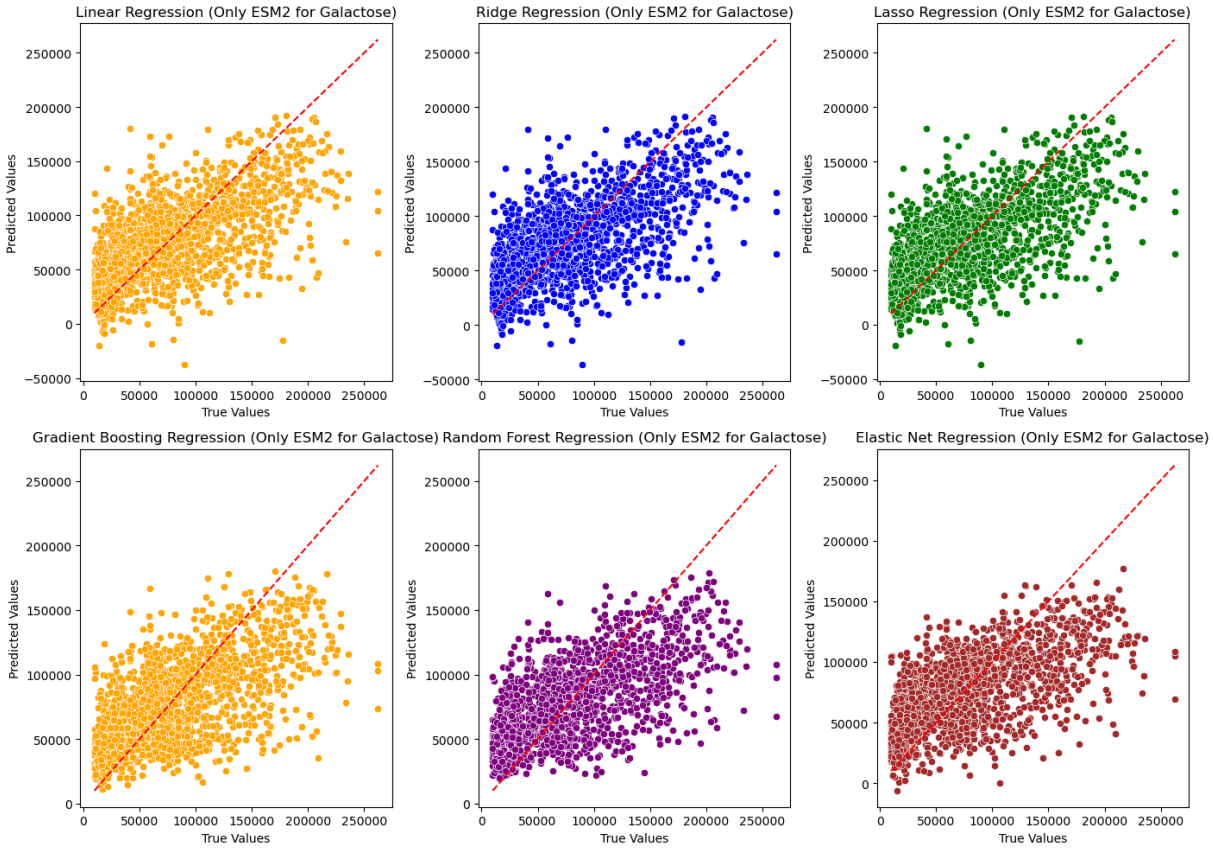
```
Random Forest Regressor Results (Only ESM2 for Galactose):
R^2: 0.427561054925541
MAE: 31798.413163874648
RMSE: 40720.3787362394
Pearson Correlation: 0.655734429092366
```

In [48]:
```
# Elastic Net
elastic_reg_embeddings_gal = make_pipeline(StandardScaler(), ElasticNet(alph
elastic_reg_embeddings_gal.fit(x_only_embeddings_train_gal, y_only_embedding
elastic_reg_embeddings_y_pred_gal = elastic_reg_embeddings_gal.predict(x_onl
print("Elastic Net Regression Results (Only ESM2):")
print(f"R^2: {r2_score(y_only_embeddings_test_gal, elastic_reg_embeddings_y_
print(f"MAE: {mean_absolute_error(y_only_embeddings_test_gal, elastic_reg_em
print(f"RMSE: {root_mean_squared_error(y_only_embeddings_test_gal, elastic_r
print(f"Pearson Correlation: {pearsonr(y_only_embeddings_test_gal, elastic_r
```

```
Elastic Net Regression Results (Only ESM2):
R^2: 0.39564814338396925
MAE: 32797.08958380816
RMSE: 41840.04593336029
Pearson Correlation: 0.6306033096953435
```

In [49]:
```
# subplots for Galactose
fig, axes = plt.subplots(2, 3, figsize=(14, 10))
axes = axes.flatten()

# Linear Regression
sns.scatterplot(x=y_only_embeddings_test_gal, y=linear_reg_embeddings_y_prec
axes[0].set_title("Linear Regression (Only ESM2 for Galactose)")
axes[0].set_xlabel("True Values")
axes[0].set_ylabel("Predicted Values")
sns.lineplot(x=y_only_embeddings_test_gal, y=y_only_embeddings_test_gal, ax=

# Ridge Regression (Galactose)
sns.scatterplot(x=y_only_embeddings_test_gal, y=ridge_reg_embeddings_gal_y_p
axes[1].set_title("Ridge Regression (Only ESM2 for Galactose)")
axes[1].set_xlabel("True Values")
axes[1].set_ylabel("Predicted Values")
sns.lineplot(x=y_only_embeddings_test_gal, y=y_only_embeddings_test_gal, ax=

# Lasso Regression (Galactose)
sns.scatterplot(x=y_only_embeddings_test_gal, y=lasso_reg_embeddings_gal_y_p
axes[2].set_title("Lasso Regression (Only ESM2 for Galactose)")
axes[2].set_xlabel("True Values")
axes[2].set_ylabel("Predicted Values")
sns.lineplot(x=y_only_embeddings_test_gal, y=y_only_embeddings_test_gal, ax=

# Gradient Boosting Regression (Galactose)
sns.scatterplot(x=y_only_embeddings_test_gal, y=grad_boost_reg_embeddings_ga
axes[3].set_title("Gradient Boosting Regression (Only ESM2 for Galactose)")
axes[3].set_xlabel("True Values")
axes[3].set_ylabel("Predicted Values")
sns.lineplot(x=y_only_embeddings_test_gal, y=y_only_embeddings_test_gal, ax=

# Random Forest Regression (Galactose)
sns.scatterplot(x=y_only_embeddings_test_gal, y=random_forest_reg_embeddings
```

```
axes[4].set_title("Random Forest Regression (Only ESM2 for Galactose)")
axes[4].set_xlabel("True Values")
axes[4].set_ylabel("Predicted Values")
sns.lineplot(x=y_only_embeddings_test_gal, y=y_only_embeddings_test_gal, ax=

# Elastic Net Regression
sns.scatterplot(x=y_only_embeddings_test_gal, y=elastic_reg_embeddings_y_pre
axes[5].set_title("Elastic Net Regression (Only ESM2 for Galactose)")
axes[5].set_xlabel("True Values")
axes[5].set_ylabel("Predicted Values")
sns.lineplot(x=y_only_embeddings_test_gal, y=y_only_embeddings_test_gal, ax=

plt.tight_layout()
plt.show()
```



In [50]:
```
# Make results df for Galactose
only_embeddings_results_gal = {
    "Model": ["Linear (Only ESM2 for Galactose)", "Ridge (Only ESM2 for Gala
    "R^2": [
        r2_score(y_only_embeddings_test_gal, linear_reg_embeddings_y_pred_ga
        r2_score(y_only_embeddings_test_gal, ridge_reg_embeddings_gal_y_prec
        r2_score(y_only_embeddings_test_gal, lasso_reg_embeddings_gal_y_prec
        r2_score(y_only_embeddings_test_gal, grad_boost_reg_embeddings_gal_y
        r2_score(y_only_embeddings_test_gal, random_forest_reg_embeddings_ga
        r2_score(y_only_embeddings_test_gal, elastic_reg_embeddings_y_pred_g
    ],
    "MAE": [
        mean_absolute_error(y_only_embeddings_test_gal, linear_reg_embedding
        mean_absolute_error(y_only_embeddings_test_gal, ridge_reg_embeddings
        mean_absolute_error(y_only_embeddings_test_gal, lasso_reg_embeddings
```

```
        mean_absolute_error(y_only_embeddings_test_gal, grad_boost_reg_embed
        mean_absolute_error(y_only_embeddings_test_gal, random_forest_reg_em
        mean_absolute_error(y_only_embeddings_test_gal, elastic_reg_embeddir
    ],
    "RMSE": [
        root_mean_squared_error(y_only_embeddings_test_gal, linear_reg_embed
        root_mean_squared_error(y_only_embeddings_test_gal, ridge_reg_embedd
        root_mean_squared_error(y_only_embeddings_test_gal, lasso_reg_embedd
        root_mean_squared_error(y_only_embeddings_test_gal, grad_boost_reg_e
        root_mean_squared_error(y_only_embeddings_test_gal, random_forest_re
        root_mean_squared_error(y_only_embeddings_test_gal, elastic_reg_embe
    ],
    "Pearson Correlation": [
        pearsonr(y_only_embeddings_test_gal, linear_reg_embeddings_y_pred_ga
        pearsonr(y_only_embeddings_test_gal, ridge_reg_embeddings_gal_y_pred
        pearsonr(y_only_embeddings_test_gal, lasso_reg_embeddings_gal_y_pred
        pearsonr(y_only_embeddings_test_gal, grad_boost_reg_embeddings_gal_y
        pearsonr(y_only_embeddings_test_gal, random_forest_reg_embeddings_ga
        pearsonr(y_only_embeddings_test_gal, elastic_reg_embeddings_y_pred_g
    ]
}

only_embeddings_results_df_gal = pd.DataFrame(only_embeddings_results_gal)
only_embeddings_results_df_gal = only_embeddings_results_df_gal.sort_values(
only_embeddings_results_df_gal.head(6)
```

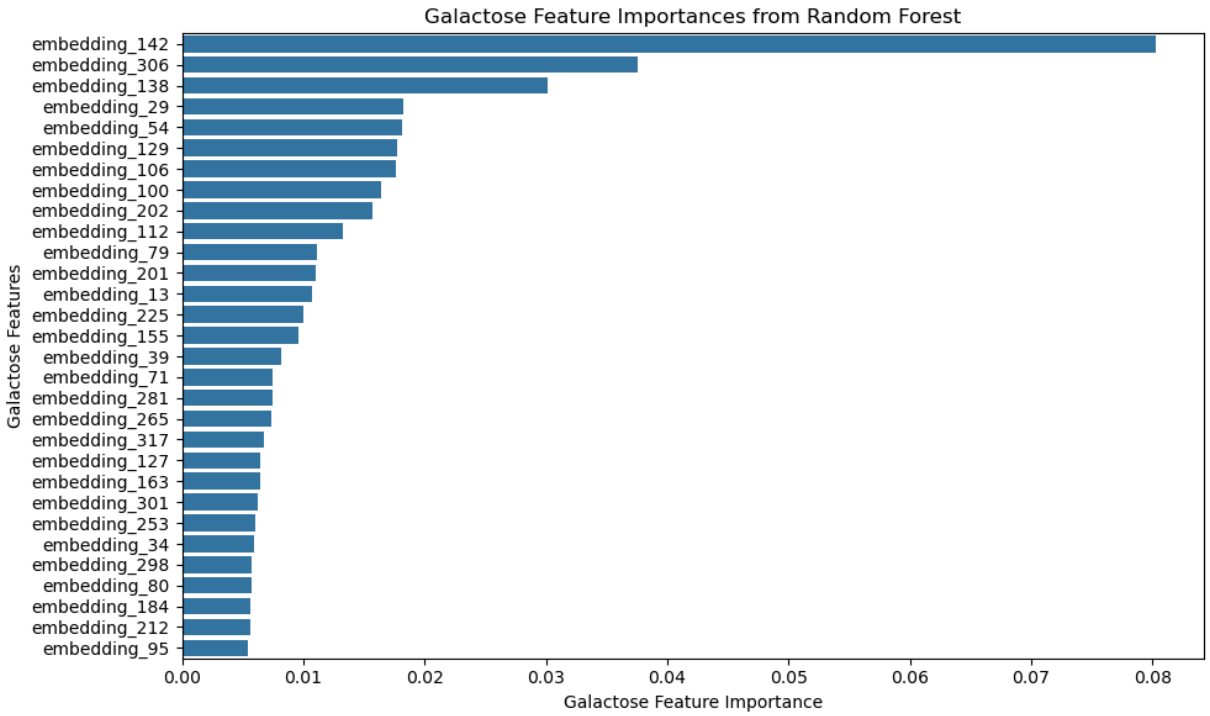
Out[50]:

	Model	R^2	MAE	RMSE	Pearson Correlation
0	Random Forest (Only ESM2 for Galactose)	0.427561	31798.413164	40720.378736	0.655734
1	Ridge (Only ESM2 for Galactose)	0.411809	31641.686910	41276.835679	0.646350
2	Lasso (Only ESM2 for Galactose)	0.411621	31646.299830	41283.423168	0.646251
3	Linear (Only ESM2 for Galactose)	0.411053	31659.841784	41303.363574	0.645932
4	Gradient Boosting (Only ESM2 for Galactose)	0.409625	32104.871240	41353.390927	0.640902
5	Elastic Net (Only ESM2 for Galactose)	0.395648	32797.089584	41840.045933	0.630603

In [51]:

```
# feature importances for Galactose
only_embeddings_feature_importances_gal = random_forest_reg_embeddings_gal.r
only_embeddings_feature_names_gal = x_only_embeddings_gal.columns
only_embeddings_sorted_indices_gal = np.argsort(only_embeddings_feature_impo

top_k = 30

sorted_only_embeddings_feature_importances_gal = only_embeddings_feature_imp
sorted_only_embeddings_feature_names_gal = only_embeddings_feature_names_gal
```

```
plt.figure(figsize=(10, 6))
sns.barplot(x=sorted_only_embeddings_feature_importances_gal, y=sorted_only_
plt.xlabel("Galactose Feature Importance")
plt.ylabel("Galactose Features")
plt.title("Galactose Feature Importances from Random Forest")
plt.tight_layout()
#plt.savefig("GCN4_feature_importances.png")
plt.show()
```



In [52]:
```
# Save the galactose model
joblib.dump(random_forest_reg_embeddings_gal, "./models/random_forest_model_
```

Out[52]:
```
['./models/random_forest_model_only_embeddings_gal.pkl']
```

Part 2 Started Here!

In [53]:
```
!pip install metapredict
```

```
Collecting metapredict
  Using cached metapredict-3.0.1-cp311-cp311-linux_x86_64.whl
Collecting cython (from metapredict)
  Using cached cython-3.1.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_
x86_64.whl.metadata (30 kB)
Requirement already satisfied: torch in /srv/conda/lib/python3.11/site-packa
ges (from metapredict) (2.7.0)
Requirement already satisfied: numpy in /srv/conda/lib/python3.11/site-packa
ges (from metapredict) (1.26.4)
Requirement already satisfied: matplotlib in /srv/conda/lib/python3.11/site-
packages (from metapredict) (3.10.1)
Collecting protfasta (from metapredict)
  Using cached protfasta-0.1.15.2-py3-none-any.whl
Requirement already satisfied: scipy in /srv/conda/lib/python3.11/site-packa
ges (from metapredict) (1.15.1)
Requirement already satisfied: urllib3 in /srv/conda/lib/python3.11/site-pac
kages (from metapredict) (2.3.0)
Requirement already satisfied: tqdm in /srv/conda/lib/python3.11/site-packag
es (from metapredict) (4.67.1)
Collecting pytorch_lightning (from metapredict)
  Using cached pytorch_lightning-2.5.1.post0-py3-none-any.whl.metadata (20 k
B)
Collecting getSequence>=2.2.1 (from metapredict)
  Using cached getSequence-2.2.1-py3-none-any.whl.metadata (20 kB)
Requirement already satisfied: requests in /srv/conda/lib/python3.11/site-pa
ckages (from getSequence>=2.2.1->metapredict) (2.32.3)
Requirement already satisfied: contourpy>=1.0.1 in /srv/conda/lib/python3.1
1/site-packages (from matplotlib->metapredict) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /srv/conda/lib/python3.11/sit
e-packages (from matplotlib->metapredict) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /srv/conda/lib/python3.1
1/site-packages (from matplotlib->metapredict) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /srv/conda/lib/python3.1
1/site-packages (from matplotlib->metapredict) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /srv/conda/lib/python3.11/
site-packages (from matplotlib->metapredict) (24.2)
Requirement already satisfied: pillow>=8 in /srv/conda/lib/python3.11/site-p
ackages (from matplotlib->metapredict) (9.2.0)
Requirement already satisfied: pyparsing>=2.3.1 in /srv/conda/lib/python3.1
1/site-packages (from matplotlib->metapredict) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /srv/conda/lib/python
3.11/site-packages (from matplotlib->metapredict) (2.9.0.post0)
Requirement already satisfied: PyYAML>=5.4 in /srv/conda/lib/python3.11/site
-packages (from pytorch_lightning->metapredict) (6.0.2)
Requirement already satisfied: fsspec>=2022.5.0 in /srv/conda/lib/python3.1
1/site-packages (from fsspec[http]>=2022.5.0->pytorch_lightning->metapredic
t) (2025.3.2)
Collecting torchmetrics>=0.7.0 (from pytorch_lightning->metapredict)
  Using cached torchmetrics-1.7.1-py3-none-any.whl.metadata (21 kB)
Requirement already satisfied: typing-extensions>=4.4.0 in /srv/conda/lib/py
thon3.11/site-packages (from pytorch_lightning->metapredict) (4.13.2)
Collecting lightning-utilities>=0.10.0 (from pytorch_lightning->metapredict)
  Using cached lightning_utilities-0.14.3-py3-none-any.whl.metadata (5.6 kB)
Requirement already satisfied: filelock in /srv/conda/lib/python3.11/site-pa
ckages (from torch->metapredict) (3.18.0)
Requirement already satisfied: sympy>=1.13.3 in /srv/conda/lib/python3.11/si
```

```
te-packages (from torch->metapredict) (1.14.0)
Requirement already satisfied: networkx in /srv/conda/lib/python3.11/site-pa
ckages (from torch->metapredict) (3.4.2)
Requirement already satisfied: jinja2 in /srv/conda/lib/python3.11/site-pack
ages (from torch->metapredict) (3.1.6)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /srv/cond
a/lib/python3.11/site-packages (from torch->metapredict) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /srv/con
da/lib/python3.11/site-packages (from torch->metapredict) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /srv/cond
a/lib/python3.11/site-packages (from torch->metapredict) (12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.5.1.17 in /srv/conda/li
b/python3.11/site-packages (from torch->metapredict) (9.5.1.17)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /srv/conda/li
b/python3.11/site-packages (from torch->metapredict) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /srv/conda/li
b/python3.11/site-packages (from torch->metapredict) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /srv/conda/l
ib/python3.11/site-packages (from torch->metapredict) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /srv/conda/
lib/python3.11/site-packages (from torch->metapredict) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /srv/conda/
lib/python3.11/site-packages (from torch->metapredict) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.3 in /srv/conda/l
ib/python3.11/site-packages (from torch->metapredict) (0.6.3)
Requirement already satisfied: nvidia-nccl-cu12==2.26.2 in /srv/conda/lib/py
thon3.11/site-packages (from torch->metapredict) (2.26.2)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /srv/conda/lib/p
ython3.11/site-packages (from torch->metapredict) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /srv/conda/
lib/python3.11/site-packages (from torch->metapredict) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /srv/conda/li
b/python3.11/site-packages (from torch->metapredict) (1.11.1.6)
Requirement already satisfied: triton==3.3.0 in /srv/conda/lib/python3.11/si
te-packages (from torch->metapredict) (3.3.0)
Requirement already satisfied: setuptools>=40.8.0 in /srv/conda/lib/python3.
11/site-packages (from triton==3.3.0->torch->metapredict) (75.6.0)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /srv/conda/lib/
python3.11/site-packages (from fsspec[http]>=2022.5.0->pytorch_lightning->me
tapredict) (3.11.16)
Requirement already satisfied: six>=1.5 in /srv/conda/lib/python3.11/site-pa
ckages (from python-dateutil>=2.7->matplotlib->metapredict) (1.17.0)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /srv/conda/lib/python3.
11/site-packages (from sympy>=1.13.3->torch->metapredict) (1.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in /srv/conda/lib/python3.11/
site-packages (from jinja2->torch->metapredict) (3.0.2)
Requirement already satisfied: charset_normalizer<4,>=2 in /srv/conda/lib/py
thon3.11/site-packages (from requests->getSequence>=2.2.1->metapredict) (3.
4.1)
Requirement already satisfied: idna<4,>=2.5 in /srv/conda/lib/python3.11/sit
e-packages (from requests->getSequence>=2.2.1->metapredict) (3.10)
Requirement already satisfied: certifi>=2017.4.17 in /srv/conda/lib/python3.
11/site-packages (from requests->getSequence>=2.2.1->metapredict) (2025.1.3
1)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /srv/conda/lib/pyt
hon3.11/site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.
```

```
5.0->pytorch_lightning->metapredict) (2.6.1)
Requirement already satisfied: aiosignal>=1.1.2 in /srv/conda/lib/python3.1
1/site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->py
torch_lightning->metapredict) (1.3.2)
Requirement already satisfied: attrs>=17.3.0 in /srv/conda/lib/python3.11/si
te-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->pytorc
h_lightning->metapredict) (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in /srv/conda/lib/python3.1
1/site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->py
torch_lightning->metapredict) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /srv/conda/lib/python
3.11/site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0-
>pytorch_lightning->metapredict) (6.4.3)
Requirement already satisfied: propcache>=0.2.0 in /srv/conda/lib/python3.1
1/site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->py
torch_lightning->metapredict) (0.3.1)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /srv/conda/lib/python3.1
1/site-packages (from aiohttp!=4.0.0a0,!=4.0.0a1->fsspec[http]>=2022.5.0->py
torch_lightning->metapredict) (1.20.0)
Using cached getSequence-2.2.1-py3-none-any.whl (140 kB)
Using cached cython-3.1.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x8
6_64.whl (3.2 MB)
Using cached pytorch_lightning-2.5.1.post0-py3-none-any.whl (823 kB)
Using cached lightning_utilities-0.14.3-py3-none-any.whl (28 kB)
Using cached torchmetrics-1.7.1-py3-none-any.whl (961 kB)
Installing collected packages: protfasta, lightning-utilities, cython, getSe
quence, torchmetrics, pytorch_lightning, metapredict
ERROR: pip's dependency resolver does not currently take into account all th
e packages that are installed. This behaviour is the source of the following
dependency conflicts.
macs2 2.2.9.1 requires Cython~=0.29, but you have cython 3.1.0 which is inco
mpatible.
Successfully installed cython-3.1.0 getSequence-2.2.1 lightning-utilities-0.
14.3 metapredict-3.0.1 protfasta-0.1.15.2 pytorch_lightning-2.5.1.post0 torc
hmetrics-1.7.1
```

In [54]:
```python
import pandas as pd
import numpy as np
from collections import Counter
import re
import joblib
from tqdm import tqdm
import metapredict as meta
import matplotlib.pyplot as plt

# Set random seed for reproducibility
np.random.seed(42)
```

In [55]:
```python
#Start off by importing the data

data = pd.read_csv("gcn4-orthologs.csv")
data = data.rename(columns={data.columns[0]: "tile_id"})
data.head()
```

Out[55]:

	tile_id		DNAseq	
0	0	AAAAATTCTAGATTTGTTTCTTTGATGCAAGGATTCTCTGATGGTT...	KNSRFVSLMQGI
1	1	AAAAATTCTCCATCTGGTTTGGCTCATTCTTCTGTTTCTGGTGTTG...	KNSPSGLAHSS
2	2	AAACAAAATTCTCAAGTTGATCAATCTCCATTGTTGCCAGAAGAAG...	KQNSQVDQSPL
3	3	AAACAAAGAGCTACTCCATTGACTCCAGTTGTTCCAGAATCTGATG...	KQRATPLTPVVP
4	4	AAACAAAGATCTATTCCATTGTCTCCAATTGTTCCAGAATCTTCTG...	KQRSIPLSPIVP

5 rows × 35 columns

In [56]:
```
# Sort by activity_SCgalactose
activity = "Activity_SCgalactose"
low_activity_df_gal = data.sort_values(by=[activity], ascending=True)
low_activity_df_gal = low_activity_df_gal[['ADseq', activity]].dropna()
low_activity_df_gal
```

Out[56]:

	ADseq	Activity_SCgalactose
8936	DPNDTVAMKRARNTLAARKSRQRKMQRFDELEDKIAKLEA	0.0
15739	SGASLFAGLQNDSPWEAPASAFTTINPHSVSGSTRTISPK	0.0
2024	TEARQNALNHRLQFDPASKLNKIQSTTIHDFSPLHTSAQP	0.0
2030	TEAARRSRARKMERMNQLEEKVEGLVGINSEIIKENSNLK	0.0
8994	DPVAVKRARNTEAARKSRARKLERQEEMERRIAELEKLLE	0.0
...	...	...
9074	DFSLGLDTTALDIAAALSQPKAKADVTPSSPMIRTTSSRA	262143.0
3439	RVSDSVSVPFGSGSTPSALFPKSLLSLPLVPRPAQATTTA	262143.0
11033	AARRSRARKMERMAQLEEKVEDLMNENSKMCDEVERLKAL	262143.0
17279	LRLSSSSRAGQPSSAAPLVNLGAISTSPLKMTFDSSNLNA	262143.0
18118	LFSTPLETDLSPLFDDIDVGVDAANWDSLFDEVPAEAPRE	262143.0

19038 rows × 2 columns

In [57]:
```
# Let's look at the min and max values of the activity
low_activity_df_gal.describe()
```

Out[57]:

	Activity_SCgalactose
count	19038.000000
mean	63422.829417
std	56236.541634
min	0.000000
25%	18045.696675
50%	50066.144585
75%	97417.912363
max	262143.000000

In Silico Directed Evolution Without loss of generality, let's simply pick the protein sequence with the lowest activity and do some a random pointwise mutation. We then repeat this process across generations. First I will add sequence features so that we can use the random forest model to predict the SCGlucose activity. The reason for this is because the model is very strict with the columns we provide for prediction. It must be named and ordered exactly the same as how we trained the model initially or scikit-learn will complain!

In [58]:
```
# Global constants reused for galactose
AA_LIST = 'ACDEFGHIKLMNPQRSTVWY'
AA_COLS = [f"AA_{aa}" for aa in AA_LIST]

KD_HYDROPHOBICITY = {
    'A': 1.8, 'C': 2.5, 'D': -3.5, 'E': -3.5, 'F': 2.8,
    'G': -0.4, 'H': -3.2, 'I': 4.5, 'K': -3.9, 'L': 3.8,
    'M': 1.9, 'N': -3.5, 'P': -1.6, 'Q': -3.5, 'R': -4.5,
    'S': -0.8, 'T': -0.7, 'V': 4.2, 'W': -0.9, 'Y': -1.3
}

MOTIFS = [
    ("W..LF", "W..LF"),
    ("WFYL_WFYL", "[WFYL]..[WFYL][WFYL]"),
    ("WFL_WFL", "[WFL]..[WFL][WFL]"),
    ("DE_WFY", "[DE][WFY]"),
    ("DE_WF", "[DE][WF]"),
    ("DE_L", "[DE][L]"),
    ("DE_x_WFY", "[DE].[WFY]"),
    ("DE_xx_WFY", "[DE]..[WFY]"),
    ("FF", "FF"),
    ("F_F", "F.F"),
    ("F__F", "F..F"),
    ("FY_5x_FY", "[FY].....[FY]"),
    ("SP", "SP"),
    ("WFYL_3x_WFYL", "[WFYL]...[WFYL][WFYL]")
]

# Feature extraction functions
```

```
def amino_acid_count(sequence: str) -> dict[str, int]:
    c = Counter(sequence)
    return {f"AA_{aa}": c.get(aa, 0) for aa in AA_LIST}

def net_charge(seq: str) -> int:
    charge_map = {"K": 1, "R": 1, "D": -1, "E": -1}
    return sum(charge_map.get(a, 0) for a in seq)

def hydrophobicity(sequence: str) -> float:
    return sum(KD_HYDROPHOBICITY.get(aa, 0) for aa in sequence)

def motif_counts_dict(seq: str) -> dict[str, int]:
    return {name: len(re.findall(pat, seq)) for name, pat in MOTIFS}

# Galactose-specific application
def add_sequence_features_gal(df_gal: pd.DataFrame, seq_col: str = "ADseq")
    df_gal = df_gal.copy()

    # Basic features
    aa_expanded = df_gal[seq_col].apply(amino_acid_count).apply(pd.Series)
    motif_expanded = df_gal[seq_col].apply(motif_counts_dict).apply(pd.Serie

    df_gal["NetCharge"] = df_gal[seq_col].apply(net_charge)
    df_gal["Hydrophobicity"] = df_gal[seq_col].apply(hydrophobicity)
    df_gal["Length"] = df_gal[seq_col].str.len()

    # Combine into one dataframe
    result_df_gal = pd.concat([df_gal, aa_expanded, motif_expanded], axis=1)
    return result_df_gal
```

In [59]:
```
low_activity_df_gal = add_sequence_features_gal(low_activity_df_gal, seq_col

# now we put Activity_SCglucose as the last column for convenience
low_activity_df_gal = low_activity_df_gal[[col for col in low_activity_df_ga
low_activity_df_gal.head()
```

Out[59]:

	ADseq	NetCharge	Hydrophobicity
8936	DPNDTVAMKRARNTLAARKSRQRKMQRFDELEDKIAKLEA	4	-51.3
15739	SGASLFAGLQNDSPWEAPASAFTTINPHSVSGSTRTISPK	0	-11.4
2024	TEARQNALNHRLQFDPASKLNKIQSTTIHDFSPLHTSAQP	1	-35.2
2030	TEAARRSRARKMERMNQLEEKVEGLVGINSEIIKENSNLK	2	-41.5
8994	DPVAVKRARNTEAARKSRARKLERQEEMERRIAELEKLLE	3	-52.4

5 rows × 39 columns

Great! Now what we're going to do is simply take the first row which corresponds to the ADseq with low SCglucose activity. We will also take note of its corresponding activity value for comparison later on!

In [60]:
```
protein_sequence_gal = low_activity_df_gal.iloc[0, 0]
activity_value_gal = low_activity_df_gal.iloc[0, -1]
print(f"Protein sequence for Galactose: {protein_sequence_gal}")
print(f"Activity value for Galactose: {activity_value_gal}")
```

```
Protein sequence for Galactose: DPNDTVAMKRARNTLAARKSRQRKMQRFDELEDKIAKLEA
Activity value for Galactose: 0.0
```

In [61]:
```
def simulate_mutations_over_generations_gal(
    initial_sequence_gal: str,
    aa_list: list[str],
    add_features_fn_gal,
    num_generations: int = 10,
    num_mutations_per_generation: int = 1_000,
    max_sequences_per_gen: int = 5_000,
):
    # Precompute the original disorder once
    original_disorder_gal = np.array(meta.predict_disorder(initial_sequence_

    current_sequences_gal = [initial_sequence_gal]
    all_dfs_gal = []

    # Begin generation loop
    for gen in tqdm(range(num_generations), desc="Generation"):
        parent_count_gal = len(current_sequences_gal)
        total_events_gal = num_mutations_per_generation

        # Randomly select parents, positions, and new amino acids
        parents_gal   = np.random.randint(0, parent_count_gal, size=total_ev
        positions_gal = np.random.randint(0, len(initial_sequence_gal), size
        new_aas_gal   = np.random.choice(aa_list, size=total_events_gal)

        # Generate mutated sequences (unique only)
        mutated_gal = {
            current_sequences_gal[p][:pos] + aa + current_sequences_gal[p][p
            for p, pos, aa in zip(parents_gal, positions_gal, new_aas_gal)
        }

        mutated_list_gal = list(mutated_gal)
        tqdm.write(f"Gen {gen+1}: {len(mutated_list_gal)} unique mutants cre

        # Optional: filter based on similarity to original disorder
        if len(mutated_list_gal) > max_sequences_per_gen:
            distances_gal = []
            for seq in tqdm(mutated_list_gal, desc="Disorder scoring"):
                sc_gal = np.array(meta.predict_disorder(seq))
                distances_gal.append(np.linalg.norm(sc_gal - original_disorc
            distances_gal = np.array(distances_gal)

            keep_idx_gal = np.argsort(distances_gal)[:max_sequences_per_gen]
            mutated_list_gal = [mutated_list_gal[i] for i in keep_idx_gal]
            tqdm.write(f"  → filtered down to {len(mutated_list_gal)} by dis

        # Add features
        df_gal = pd.DataFrame({"ADseq": mutated_list_gal})
        df_gal["Generation"] = gen + 1
```

```
        df_gal = add_features_fn_gal(df_gal, seq_col="ADseq")

        # Save for this generation
        all_dfs_gal.append(df_gal)
        current_sequences_gal = mutated_list_gal

    return pd.concat(all_dfs_gal, ignore_index=True)
```

In [62]:
```
result_gal = simulate_mutations_over_generations_gal(
    initial_sequence_gal=protein_sequence_gal,
    aa_list=list(AA_LIST),
    add_features_fn_gal=add_sequence_features_gal,  # make sure this is defi
    num_generations=5,
    num_mutations_per_generation=5000
)
```

```
Generation:   0%|          | 0/5 [00:00<?, ?it/s]
Gen 1: 759 unique mutants created
Generation:  20%|██        | 1/5 [00:00<00:00,  4.13it/s]
Gen 2: 4887 unique mutants created
Generation:  40%|████      | 2/5 [00:01<00:02,  1.07it/s]
Gen 3: 4988 unique mutants created
Generation:  60%|██████    | 3/5 [00:03<00:02,  1.29s/it]
Gen 4: 4989 unique mutants created
Generation:  80%|████████  | 4/5 [00:05<00:01,  1.46s/it]
Gen 5: 4991 unique mutants created
Generation: 100%|██████████| 5/5 [00:07<00:00,  1.42s/it]
```

In [63]:
```
result_gal.head()
```

Out[63]:

	ADseq	Generation	NetCharge	Hydrop
0	DPNDTVAMKRARFTLAARKSRQRKMQRFDELEDKIAKLEA	1	4	
1	DPNDTVAMKRARNTLAAPKSRQRKMQRFDELEDKIAKLEA	1	3	
2	DPNDTVAMKRARNTLAARKSRQRKMYRFDELEDKIAKLEA	1	4	
3	DPNDTVAMKRARNTLAARKSRQRKMQRFDELFDKIAKLEA	1	5	
4	DPNDTVAMKRARNTLAARKSRQRKMQRFDELEDKIAKVEA	1	4	

5 rows × 39 columns

In [64]:
```
# ============================
# ESM2 Feature Embedding Computation
# ============================

# Load the .env file
load_dotenv()
hf_token = os.getenv("HF_TOKEN")

# Load model and tokenizer once
model_id = "facebook/esm2_t6_8M_UR50D"
tokenizer = AutoTokenizer.from_pretrained(model_id, token=hf_token)
```

```
model = AutoModel.from_pretrained(model_id, token=hf_token)
model.eval()

def generate_embedding(sequence):
    """Generate pooled ESM embedding for a protein sequence."""
    input_ids = tokenizer(sequence, return_tensors="pt").input_ids
    attention_mask = torch.ones_like(input_ids)

    with torch.no_grad():
        outputs = model(input_ids, attention_mask=attention_mask)
        sequence_embeddings = outputs.last_hidden_state.squeeze()

    # Remove special tokens [CLS], [EOS]
    sequence_embeddings = sequence_embeddings[1:-1]

    # Mean-pool across residues
    pooled_embedding = sequence_embeddings.mean(dim=0)  # shape: (hidden_siz
    return pooled_embedding.cpu().numpy()

def save_embeddings_to_npz(data, sequence_column, save_path):
    """Generate embeddings for all sequences and save into a single NPZ file
    all_embeddings = []

    for sequence in tqdm(data[sequence_column], desc="Generating embeddings"
        embedding = generate_embedding(sequence)
        all_embeddings.append(embedding)

    all_embeddings = np.stack(all_embeddings)  # Shape: (num_sequences, hidd

    np.savez_compressed(save_path, embeddings=all_embeddings)
    print(f"Saved {len(all_embeddings)} embeddings to {save_path}")

    return all_embeddings
```

```
Some weights of EsmModel were not initialized from the model checkpoint at f
acebook/esm2_t6_8M_UR50D and are newly initialized: ['pooler.dense.bias', 'p
ooler.dense.weight']
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
```

In [65]:
```
result_gal.head()
```

Out[65]:

	ADseq	Generation	NetCharge	Hydrop
0	DPNDTVAMKRARFTLAARKSRQRKMQRFDELEDKIAKLEA	1	4	
1	DPNDTVAMKRARNTLAAPKSRQRKMQRFDELEDKIAKLEA	1	3	
2	DPNDTVAMKRARNTLAARKSRQRKMYRFDELEDKIAKLEA	1	4	
3	DPNDTVAMKRARNTLAARKSRQRKMQRFDELFDKIAKLEA	1	5	
4	DPNDTVAMKRARNTLAARKSRQRKMQRFDELEDKIAKVEA	1	4	

5 rows × 39 columns

In [66]:
```
model_name = "./models/esm_random_forest_model_gal.pkl"

# Load the regression model
reg_model = joblib.load(model_name)

if model_name.split("/")[-1].split("_")[0] == "esm":
    mutated_seq_embeddings = save_embeddings_to_npz(result_gal, "ADseq", "./
    embeddings_df = pd.DataFrame(mutated_seq_embeddings)
    embeddings_df.columns = [f"embedding_{i}" for i in range(embeddings_df.s
    mutated_feature =result_gal.drop(columns=["ADseq", "Generation"])
    mutated_feature = pd.concat([mutated_feature, embeddings_df], axis=1)

else:
    mutated_feature = result_gal.drop(columns=["ADseq", "Generation"])

# Make predictions on the mutated sequence
predictions = reg_model.predict(mutated_feature)

# Add predictions to the DataFrame
result_gal[activity] = predictions

result_gal.head()
```

```
Generating embeddings: 100%|██████████| 20614/20614 [04:55<00:00, 69.73it/s]
Saved 20614 embeddings to ./feature-embeddings/mutated_seq_embeddings.npz
```

Out[66]:

	ADseq	Generation	NetCharge	Hydrop
0	DPNDTVAMKRARFTLAARKSRQRKMQRFDELEDKIAKLEA	1	4	
1	DPNDTVAMKRARNTLAAPKSRQRKMQRFDELEDKIAKLEA	1	3	
2	DPNDTVAMKRARNTLAARKSRQRKMYRFDELEDKIAKLEA	1	4	
3	DPNDTVAMKRARNTLAARKSRQRKMQRFDELFDKIAKLEA	1	5	
4	DPNDTVAMKRARNTLAARKSRQRKMQRFDELEDKIAKVEA	1	4	

5 rows × 40 columns

In [67]:
```
#ADD CELLS AS NEEDED, MAKE SURE TO DOCUMENT YOUR CODE WITH COMMENTS
```

Naturally the next thing to do is to find which of these mutations is predicted to have the highest scglucose_activity. We'll then perform a more thorough analysis on this sequence

In [68]:
```
highest_activity_candidates = result_gal.sort_values(by=[activity], ascendir
highest_activity_candidates
```

Out[68]:

	ADseq	Generation	NetCharge	Hy
1789	DDNDTVAMKRRRNTLAARKSRQRKMQRFDELEDKIAKLEA	2	4	
8773	DDNDTVPMKRARNTLAARKSRQRKMQRFDMLEDKIAKLEA	3	4	
18562	HPNDTNACKRARNTLAARKRRQRKMQRFDKLEDKIAKLEA	5	8	
15724	DPNDTVAMKRCRNTLAARDSRQRKMQRHDELEDKIHVLEA	5	1	
9670	HPNDTVACKRARNTLAARKRRQRKMQRFDELEDKIAKLEA	3	6	
...	...	...	...	
13915	DPNDTVAMKRARNTLAARKSRQEKMQRFDEEEDKIAKLSS	4	2	
6832	DPNDTVAMKLARNTEAARKSRQRKMQRFDELEDKIAKLET	3	2	
10703	DPNDTVAMKRARTGLAARKSRERKMQRFNELEDKIAKLEA	4	4	
13086	DPNDTVAMERARNTLAARKSRQRKMQRFDELERKIAKLED	4	3	
2148	DPNDTVAMKRAENTLAARKSRQRKMQRFDELEGKIAKLEA	2	3	

20614 rows × 40 columns

Now let's take the top 500 rows and then further filter for the top 100 based off disorder score

In [69]:
```
top_500 = highest_activity_candidates.head(500)
top_500.head()
```

Out[69]:

	ADseq	Generation	NetCharge	Hy
1789	DDNDTVAMKRRRNTLAARKSRQRKMQRFDELEDKIAKLEA	2	4	
8773	DDNDTVPMKRARNTLAARKSRQRKMQRFDMLEDKIAKLEA	3	4	
18562	HPNDTNACKRARNTLAARKRRQRKMQRFDKLEDKIAKLEA	5	8	
15724	DPNDTVAMKRCRNTLAARDSRQRKMQRHDELEDKIHVLEA	5	1	
9670	HPNDTVACKRARNTLAARKRRQRKMQRFDELEDKIAKLEA	3	6	

5 rows × 40 columns

Now using vector norms, I want to sort by disorderedness

In [70]:
```
orig_seq_disorder = np.array(meta.predict_disorder(protein_sequence_gal))
top_500_disorder = np.array([meta.predict_disorder(seq) for seq in top_500["
top_500_disorder = np.array(top_500_disorder)


# now sort by the disorder
sorted_disorder = np.argsort(np.linalg.norm(top_500_disorder - orig_seq_disc
sorted_disorder = top_500.iloc[sorted_disorder]
```

```
sorted_disorder = sorted_disorder.reset_index(drop=True)
sorted_disorder.head(10)
```

Out[70]:

	ADseq	Generation	NetCharge	Hydro
0	DPNDTVAMKRARNQLAARKRRQRKMQRFDELEDKIAKLEA	2	5	
1	DPHDTVAMKRARNTLAARKRRQRKMQRFDELEDKIAKLEA	2	5	
2	DPNDTVAMKRARNTLAARKKRQRKMQRFDELEDKIAKLEA	2	5	
3	DPNDTVAMKRARNTLAARKKRQRKMQRFDELEDKIAKLEA	1	5	
4	HPNDTVAMKRARNTLAARKRRQRKMQRFDELEDKIAKLEA	2	6	
5	DPNDTVAMKRARNTLAARKRRQRKMQRFDELEDKIAKLEA	1	5	
6	DDNDTVAMKRARNTLAARKKRQRKMQRFDELEDKYAKLEA	3	4	
7	EPNDTVAMKRARNTLAARKKRQRKMQRFDELEDKWAKLEA	3	5	
8	DPNKTVAMKRARNSLAARKSRQRKMARFDELEDKIAKLEA	3	6	
9	DPNDTVAMKRARNTLAARNSRQRKMQRFDELEDKIAFLEA	3	2	

10 rows × 40 columns

Cross-Examination Between Original and Mutated Sequences

In [71]:
```
top_candidate = sorted_disorder.iloc[0]
print(f"Top candidate sequence: {top_candidate['ADseq']}")
print(f"Original sequence: {protein_sequence_gal}")
print(f"Original activity: {activity_value_gal}")
print(f"Predicted activity: {top_candidate[activity]}")
```

```
Top candidate sequence: DPNDTVAMKRARNQLAARKRRQRKMQRFDELEDKIAKLEA
Original sequence: DPNDTVAMKRARNTLAARKSRQRKMQRFDELEDKIAKLEA
Original activity: 0.0
Predicted activity: 126335.36038289992
```

In [73]:
```
# How these two are different
def compare_sequences(seq1: str, seq2: str) -> list[tuple[int, str, str]]:
    differences = []
    for i, (a, b) in enumerate(zip(seq1, seq2)):
        if a != b:
            differences.append((i, a, b))
    return differences
differences = compare_sequences(protein_sequence_gal, top_candidate["ADseq"]
print("Differences between original and top candidate:")
for pos, original_aa, new_aa in differences:
    print(f"Position {pos}: {original_aa} -> {new_aa}")
```

```
Differences between original and top candidate:
Position 13: T -> Q
Position 19: S -> R
```

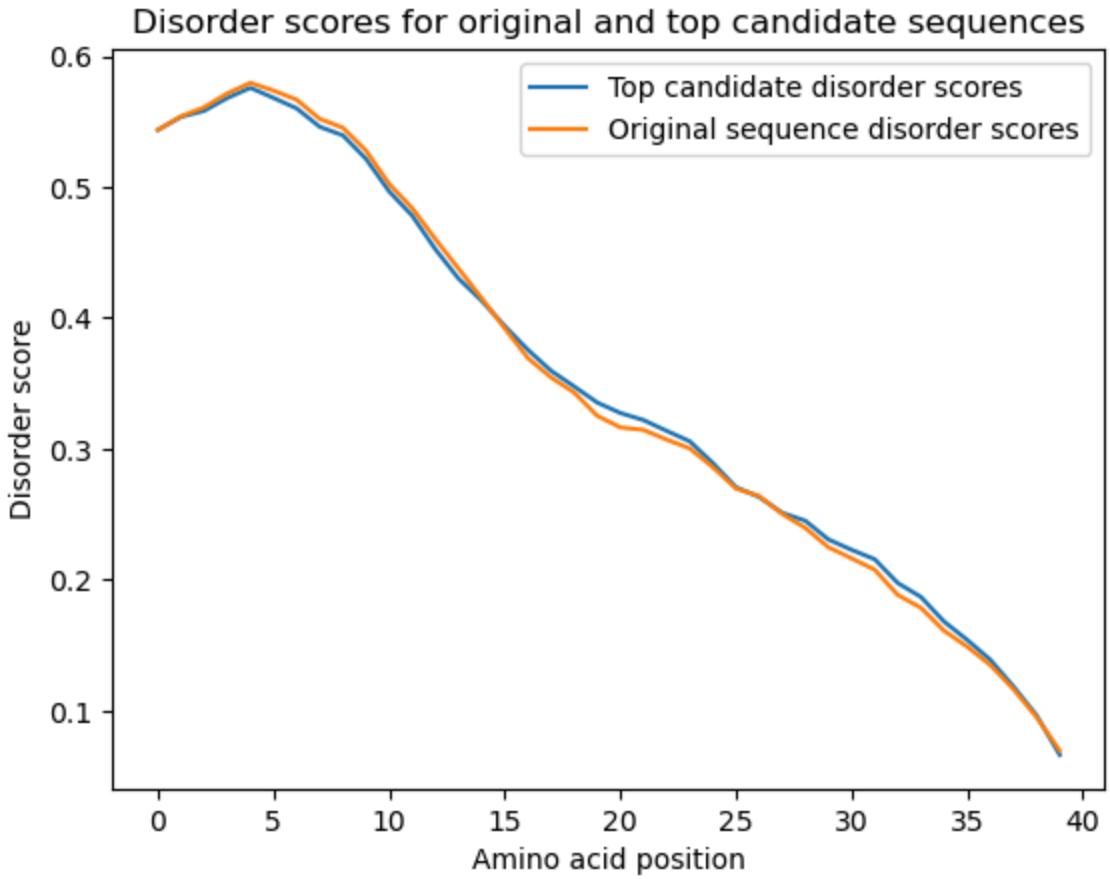
In [76]:
```
# Difference in net charge and hydrophobicity
original_net_charge = net_charge(protein_sequence_gal)
```

```
top_candidate_net_charge = net_charge(top_candidate["ADseq"])
print(f"Original net charge: {original_net_charge}")
print(f"Top candidate net charge: {top_candidate_net_charge}")
print(f"Net charge difference: {top_candidate_net_charge - original_net_char


original_hydrophobicity = hydrophobicity(protein_sequence_gal)
top_candidate_hydrophobicity = hydrophobicity(top_candidate["ADseq"])
print(f"Original hydrophobicity: {original_hydrophobicity}")
print(f"Top candidate hydrophobicity: {top_candidate_hydrophobicity}")
print(f"Hydrophobicity difference: {top_candidate_hydrophobicity - original_
```

```
Original net charge: 4
Top candidate net charge: 5
Net charge difference: 1
Original hydrophobicity: -51.30000000000001
Top candidate hydrophobicity: -57.80000000000001
Hydrophobicity difference: -6.5
```

Disorder Analysis with Meta Predict

In [78]:
```
top_candidate_disorder_scores = meta.predict_disorder(top_candidate["ADseq"]
original_sequence_disorder_scores = meta.predict_disorder(protein_sequence_g

plt.plot(top_candidate_disorder_scores, label="Top candidate disorder scores
plt.plot(original_sequence_disorder_scores, label="Original sequence disorde
plt.xlabel("Amino acid position")
plt.ylabel("Disorder score")
plt.title("Disorder scores for original and top candidate sequences")
plt.legend()
plt.show()
```



In [80]:
```
# Save the top 500 sequences to a CSV file
top_500 = sorted_disorder.head(500)
top_500.to_csv(f"top_500_sequences_{activity}.csv", index=False)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: