`generate_dummy_data` function utilized by NetREm demo

By: Saniya Khullar, Xiang Huang, Raghu Ramesh, John Svaren, Daifeng Wang

Daifeng Wang Lab

:) Please note that this function *generate_dummy_data* focuses on generating random X and y data. We set a random seed for generating y and a random seed for generating X so that our demo results are reproducible :).

In our biological example, X would be gene expression data for the N candidate Transcription Factors (TFs), our predictors, and y would be the gene expression data for the target gene (TG), which we believe to be regulated by a subset of these N TFs.

Explanation of *generate_dummy_data* function:

Please note that in this tutorial, we will go over our function *generate_dummy_data* in more details. We use this function to generate dummy X and y training and testing data for our NetREm demo (our toy example). Here, we will expand on that toy example and provide more details and clarifications as well as more features that could be adjusted, if needed. In general, this function can be extended and applied in cases where the user would like to generate random X and y data where each of the X variables have a predetermined correlation with the y variable. Please note that we do not consider pairwise correlations among the X predictors.

In our example, we will create 5 predictors [X_1, X_2, X_3, X_4, X_5], where X_1 has a strongly positive correlation with y of 0.9 while X_5 has a strongly negative correlation with y of -0.8. X_3 and X_4 have rather weak correlations with y of 0.1 and -0.2, respectively.

Here, we build Y data based on a normal distribution (specified mean μ and standard deviation). M is the # of samples we want to generate. Thus, Y is a vector with M elements. Then, this class returns X for a set of N predictors (each with M # of samples) based on a list of N respective correlation values. For instance, if N = 5 predictors (the Transcription Factors (TFs): [TF_1, TF_2, TF_3, TF_4, TF_5]), our respective predictors are: [X_1, X_2, X_3, X_4, X_5]. Then, we may have a respective list of Pearson correlation (r) values given by *corrVals* = [r_1, r_2, r_3, r_4, r_5] = [cor(TF_1, y), cor(TF_2, y), cor(TF_3, y), cor(TF_4, y), cor(TF_5, y)]. Then, *generate_dummy_data* will generate X, an input matrix of those 5 predictors (based on a similar distribution as Y) with these respective Pearson correlations (r).

Parameter	Definition	More information
M	# of samples (data points) to generate.	Default: 100
N	# of predictors (X values):	[X_1, ..., X_N]
X	Input numpy array matrix (list of lists) each list corresponds to a sample. Here, rows are samples and columns are predictors.	Dimensions: M rows by N columns
y	Input numpy array list with 1 value for each sample.	Dimensions: List of M entries

In this demo, we specify: *corrVals*: [cor(TF_1, y) = 0.9, cor(TF_2, y) = 0.5, cor(TF_3, y) = 0.1, cor(TF_4, y) = -0.2, cor(TF_5, y) = -0.8]. = 0.9, cor(TF_2, y) = 0.5, cor(TF_3, y) = 0.1, cor(TF_4, y) = -0.2, cor(TF_5, y) = -0.8].

We also specify `num_samples_M` is 5,000 samples. Thus, y, our response variable vector, will be a vector with 100 entries, 1 for each sample. Since we have N = 5 predictors, our X matrix will be 5,000 rows by 5 columns (samples by predictors: M by N).

```
generate_dummy_data(
 corrVals ,
 num_samples_M = 10000,
 train_data_percent = 70,
 mu = 0,
 std_dev = 1,
 iters_to_generate_X = 100,
 orthogonal_X = False,
 ortho_scalar = 10,
 view_input_corrs_plot = False,
 verbose = True,
 rand_seed_x = 123,
 rand_seed_y = 2023
)
```

In [2]:

```python
from DemoDataBuilderXandY import generate_dummy_data
import plotly.express as px
import numpy as np
import pandas as pd

corrVals = [0.9, 0.5, 0.1, -0.2, -0.8]
M = 50000
train_data_percent = 70
view_corrVals_plot = True # for visual aid, to see a plot of input correlations

dummy_data = generate_dummy_data(corrVals = corrVals,
                    num_samples_M = M,
                    train_data_percent = train_data_percent,
                    view_input_corrs_plot = view_corrVals_plot)
```

```
:) same_train_test_data = False
```

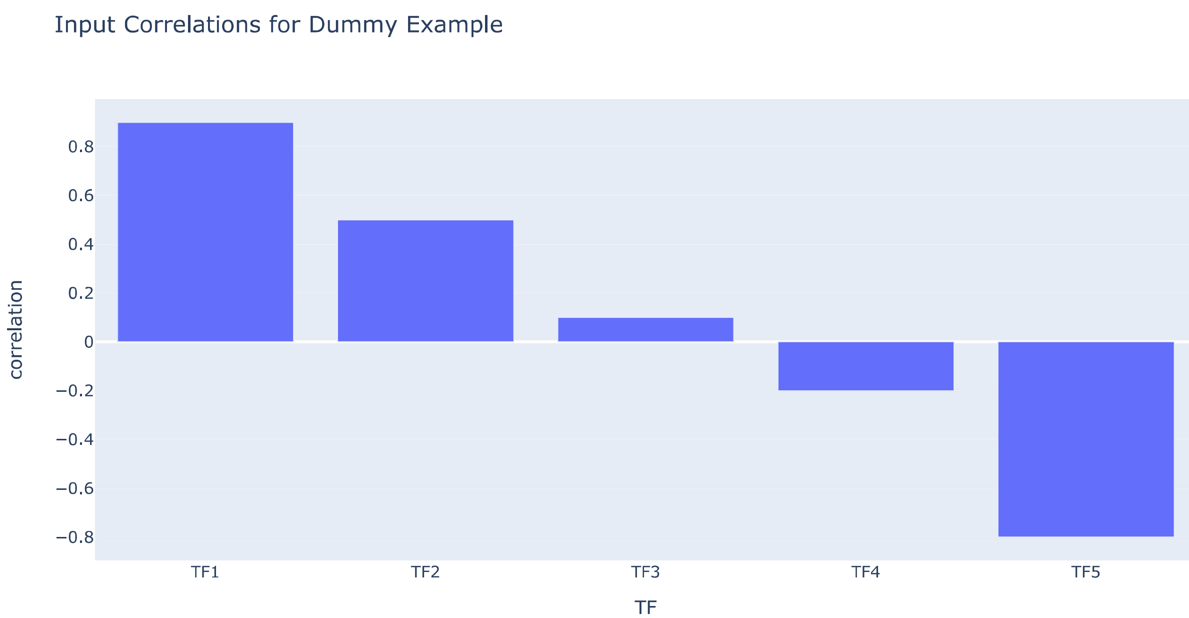
Generating predictors: 100%   5/5 [00:00<00:00, 294.74it/s]

```
Please note that since we hold out 30.0% of our 50000 samples for testing, we have:
X_train = 35000 rows (samples) and 5 columns (N = 5 predictors) for training.
X_test = 15000 rows (samples) and 5 columns (N = 5 predictors) for testing.
y_train = 35000 corresponding rows (samples) for training.
y_test = 15000 corresponding rows (samples) for testing.
```

100%   5/5 [00:00<00:00, 354.84it/s]

100%   5/5 [00:00<00:00, 348.02it/s]

100%   5/5 [00:00<00:00, 355.43it/s]



In [3]:

```python
X_original = dummy_data.X
y_original = dummy_data.y

print(f"X_original dimensions (M = {X_original.shape[0]} rows, N = {X_original.shape[1]} columns)")
print(f"y_original dimensions (M = {y_original.shape[0]} rows, 1 column)")
```

```
X_original dimensions (M = 50000 rows, N = 5 columns)
y_original dimensions (M = 50000 rows, 1 column)
```

We have generated:

- X_{train} and y_{train} for training and building a machine learning model
- X_{test} and y_{test} for testing our machine learning model on new, unseen real-world data (evaluating generalizability of model)

In [4]:

```python
# training data sets:
X_train = dummy_data.get_X_train()
y_train = dummy_data.get_y_train()

print(f":) Training Data information: {train_data_percent}% of original data")
print(f"X_train dimensions (M_train = {X_train.shape[0]} rows, N = {X_train.shape[1]} columns)")
print(f"y_train dimensions (M_train = {y_train.shape[0]} rows, 1 column)")
print("")

# testing data sets:
X_test = dummy_data.get_X_test()
y_test = dummy_data.get_y_test()
print(f":) Testing Data information: {dummy_data.test_data_percent}% of original data")
print(f"X_test dimensions (M_test = {X_test.shape[0]} rows, N = {X_test.shape[1]} columns)")
print(f"y_test dimensions (M_test = {y_test.shape[0]} rows, 1 column)")
print("")
```

```
:) Training Data information: 70% of original data
X_train dimensions (M_train = 35000 rows, N = 5 columns)
y_train dimensions (M_train = 35000 rows, 1 column)

:) Testing Data information: 30% of original data
X_test dimensions (M_test = 15000 rows, N = 5 columns)
y_test dimensions (M_test = 15000 rows, 1 column)
```

For more clarity, please note that we can view the datasets as Pandas dataframes:

In [7]:

```python
# We can view the original y data: (M = 100 samples)
y_df = dummy_data.view_original_y_df()
y_df
```

Out[7]:

	y
0	0.601721
1	1.151619
2	-1.359462
3	0.222055
4	-0.775868
...	...
49995	-0.445056
49996	0.103515
49997	-0.446794
49998	1.845294
49999	-0.073774

50000 rows × 1 columns

In [8]:
```python
# We can view the original X data: (M = 100 samples)
X_df = dummy_data.view_original_X_df()
X_df
```

Out[8]:

	TF1	TF2	TF3	TF4	TF5
0	0.060212	1.157731	0.341336	-1.606832	-0.823771
1	1.743612	-1.530726	-0.312241	1.012032	-1.435383
2	-1.523260	-0.767257	1.347948	-0.358711	0.822745
3	0.002555	2.013962	2.198329	0.941011	0.057121
4	-0.383786	0.896405	-1.009631	1.310682	-0.128220
...	...	...	...	...	...
49995	-0.735020	0.072683	0.399550	1.313636	-0.364718
49996	0.048522	-0.402600	0.902739	0.127789	-0.221559
49997	-0.101323	-0.099031	-0.545232	0.020850	-0.041131
49998	1.478381	0.995326	-0.552561	-1.215034	-1.426920
49999	-0.791031	-1.718334	-0.260494	-0.420600	0.105212

50000 rows × 5 columns

In [9]:
```python
# We can view the X_training data:
X_train_df = dummy_data.view_X_train_df()
X_train_df
```

Out[9]:

	TF1	TF2	TF3	TF4	TF5
0	0.012406	0.922764	-0.247662	0.197411	-0.160290
1	0.683566	-1.944551	1.126392	-0.117505	-0.222134
2	-0.874310	0.259708	-0.844884	1.606723	1.962838
3	0.227996	-0.365244	0.612245	-0.147671	-0.297991
4	1.065428	-0.477365	-0.924536	-0.645075	-0.356331
...	...	...	...	...	...
34995	0.059203	-0.714533	-1.258057	0.088533	0.638672
34996	0.975696	0.630414	0.394688	-0.416213	-0.530217
34997	1.100133	0.685614	-0.626559	-0.407492	-0.472842
34998	0.380724	0.544623	0.599941	1.862734	-0.269971
34999	0.854005	1.081615	0.866324	-1.006917	-1.427055

35000 rows × 5 columns

In [10]:
```python
# to view pairwise correlations among the X predictors in the training dataset
X_train_df.corr()
```

Out[10]:

	TF1	TF2	TF3	TF4	TF5
TF1	1.000000	0.446801	0.087681	-0.187984	-0.723031
TF2	0.446801	1.000000	0.040418	-0.105962	-0.403371
TF3	0.087681	0.040418	1.000000	-0.025694	-0.076525
TF4	-0.187984	-0.105962	-0.025694	1.000000	0.168489
TF5	-0.723031	-0.403371	-0.076525	0.168489	1.000000

In [11]:

```
# We can view the y_training data:
y_train_df = dummy_data.view_y_train_df()
y_train_df
```

Out[11]:

	y
0	0.442239
1	0.666132
2	-1.263846
3	-0.008571
4	0.664666
...	...
34995	0.011464
34996	0.469827
34997	0.922917
34998	0.335860
34999	1.688821

35000 rows × 1 columns

In [12]:

```
# We can view the X_testing data:
X_test_df = dummy_data.view_X_test_df()
# or just directly say:
# X_test_df = dummy_data.X_test_df
X_test_df
```

Out[12]:

	TF1	TF2	TF3	TF4	TF5
0	-1.653509	-0.327008	1.019585	1.811873	1.072599
1	1.142105	1.096250	2.004035	-2.026071	-1.593196
2	-0.322888	-0.187026	1.993765	0.019833	0.271462
3	-0.407612	0.050315	0.047314	-0.465817	-0.750017
4	-0.147236	-1.492924	1.889835	-0.037268	-0.167145
...	...	...	...	...	...
14995	-0.449012	0.806672	-1.684291	-0.629418	0.280621
14996	0.709095	1.189145	1.564877	-0.560968	-0.644792
14997	0.712328	1.603306	1.135285	0.622087	0.454745
14998	-0.108016	-0.358597	1.055580	-0.877273	-1.142462
14999	-0.035035	0.879663	-0.256335	-1.857476	-0.349704

15000 rows × 5 columns

In [13]:

```
# to view pairwise correlations among the X predictors in the testing dataset
X_test_df.corr()
```

Out[13]:

	TF1	TF2	TF3	TF4	TF5
TF1	1.000000	0.445547	0.087696	-0.171964	-0.723251
TF2	0.445547	1.000000	0.039867	-0.094735	-0.394644
TF3	0.087696	0.039867	1.000000	-0.021768	-0.070333
TF4	-0.171964	-0.094735	-0.021768	1.000000	0.158131
TF5	-0.723251	-0.394644	-0.070333	0.158131	1.000000

In [18]:

```python
# We can view the y_testing data:
y_test_df = dummy_data.view_y_test_df()
# or just directly say:
# y_test_df = dummy_data.y_test_df
y_test_df
```

Out[18]:

	y
0	-1.957836
1	1.904234
2	0.338825
3	-0.237264
4	-0.336472
...	...
14995	-0.789841
14996	0.777740
14997	0.739777
14998	0.432590
14999	0.342612

In [19]:

```python
X_train = dummy_data.get_X_train()
y_train = dummy_data.get_y_train()

X_test = dummy_data.get_X_test()
y_test = dummy_data.get_y_test()
```

In [20]:

```python
# We can view the X_training data:
X_train_df = dummy_data.view_X_train_df()
X_train_df
```

Out[20]:

	TF1	TF2	TF3	TF4	TF5
0	0.012406	0.922764	-0.247662	0.197411	-0.160290
1	0.683566	-1.944551	1.126392	-0.117505	-0.222134
2	-0.874310	0.259708	-0.844884	1.606723	1.962838
3	0.227996	-0.365244	0.612245	-0.147671	-0.297991
4	1.065428	-0.477365	-0.924536	-0.645075	-0.356331
...	...	...	...	...	...
34995	0.059203	-0.714533	-1.258057	0.088533	0.638672
34996	0.975696	0.630414	0.394688	-0.416213	-0.530217
34997	1.100133	0.685614	-0.626559	-0.407492	-0.472842
34998	0.380724	0.544623	0.599941	1.862734	-0.269971
34999	0.854005	1.081615	0.866324	-1.006917	-1.427055

35000 rows × 5 columns

Please note that *X_original* (X_group = overall) should have the correlations that were specified in *corrVals*.

If we split the data into training and testing datasets, the correlations of predictors with y may sadly not be exactly what we specified in *corrVals*. Alas, in the real world, we may not know the ground truth correlations and our random partitioning (for training and testing datasets), though done randomly (and by using widely-used Python packages for machine learning) may not reflect the true data distribution. However, these are some of the trade-offs that come with partitioning the data into training and testing datasets.

If this is worrisome, please use *train_data_percent = 100* so that all of the training data is also used for testing. In essence, there is no testing data. Then, please try to utilize cross-validation (CV) measures to evaluate your datasets or other ways to partition the data.

We can view the metrics of the differences by looking at the *combined_correlations_df* below:

In [21]:

```
combined_correlations_df = dummy_data.combined_correlations_df
combined_correlations_df
```

Out[21]:

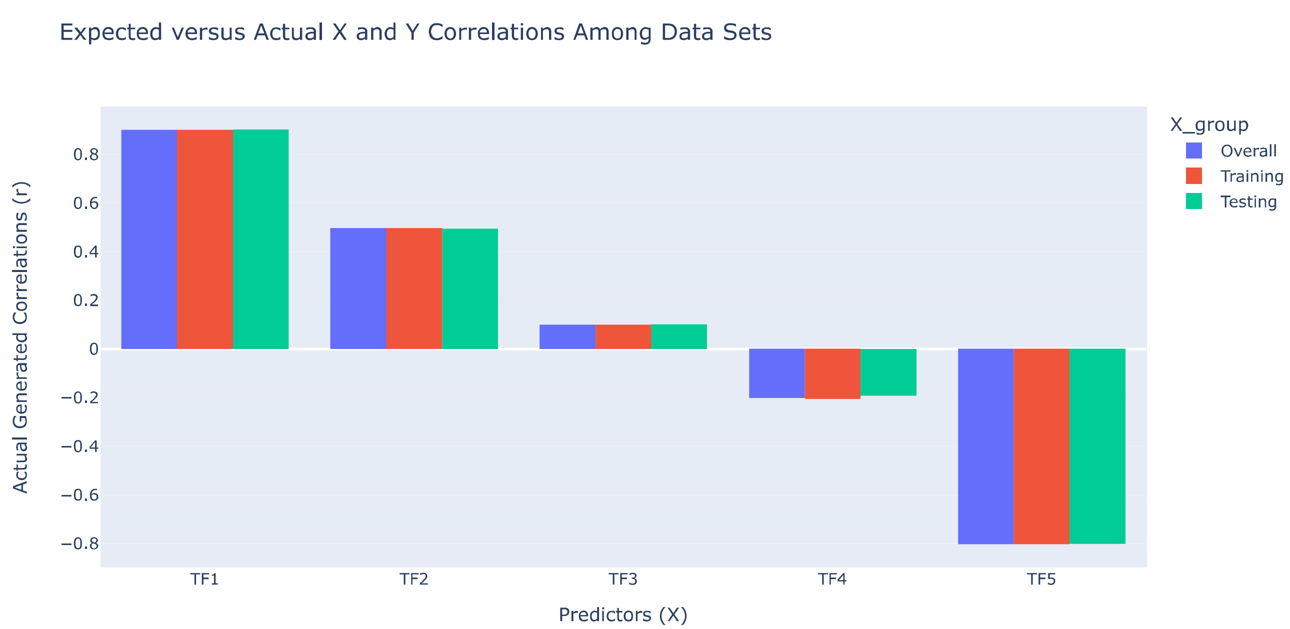
	i	predictor	expected_corr_with_Y	actual_corr	difference	X_group	num_samples
0	0	TF1	0.9	0.900976	0.000976	Overall	unique 50000
1	1	TF2	0.5	0.496949	0.003051	Overall	unique 50000
2	2	TF3	0.1	0.100172	0.000172	Overall	unique 50000
3	3	TF4	-0.2	-0.201952	0.001952	Overall	unique 50000
4	4	TF5	-0.8	-0.802429	0.002429	Overall	unique 50000
0	0	TF1	0.9	0.900615	0.000615	Training	unique 35000
1	1	TF2	0.5	0.497784	0.002216	Training	unique 35000
2	2	TF3	0.1	0.100018	0.000018	Training	unique 35000
3	3	TF4	-0.2	-0.206051	0.006051	Training	unique 35000
4	4	TF5	-0.8	-0.802839	0.002839	Training	unique 35000
0	0	TF1	0.9	0.901819	0.001819	Testing	unique 15000
1	1	TF2	0.5	0.494940	0.005060	Testing	unique 15000
2	2	TF3	0.1	0.100533	0.000533	Testing	unique 15000
3	3	TF4	-0.2	-0.192267	0.007733	Testing	unique 15000
4	4	TF5	-0.8	-0.801457	0.001457	Testing	unique 15000

In [22]:

```
fig = px.histogram(combined_correlations_df, x="predictor", y="actual_corr",
                   color='X_group', barmode='group',
                           title = "Expected versus Actual X and Y Correlations Among Data Sets")

# Rename axis labels
fig.update_xaxes(title_text='Predictors (X)')
fig.update_yaxes(title_text='Actual Generated Correlations (r)')

fig.show()
```

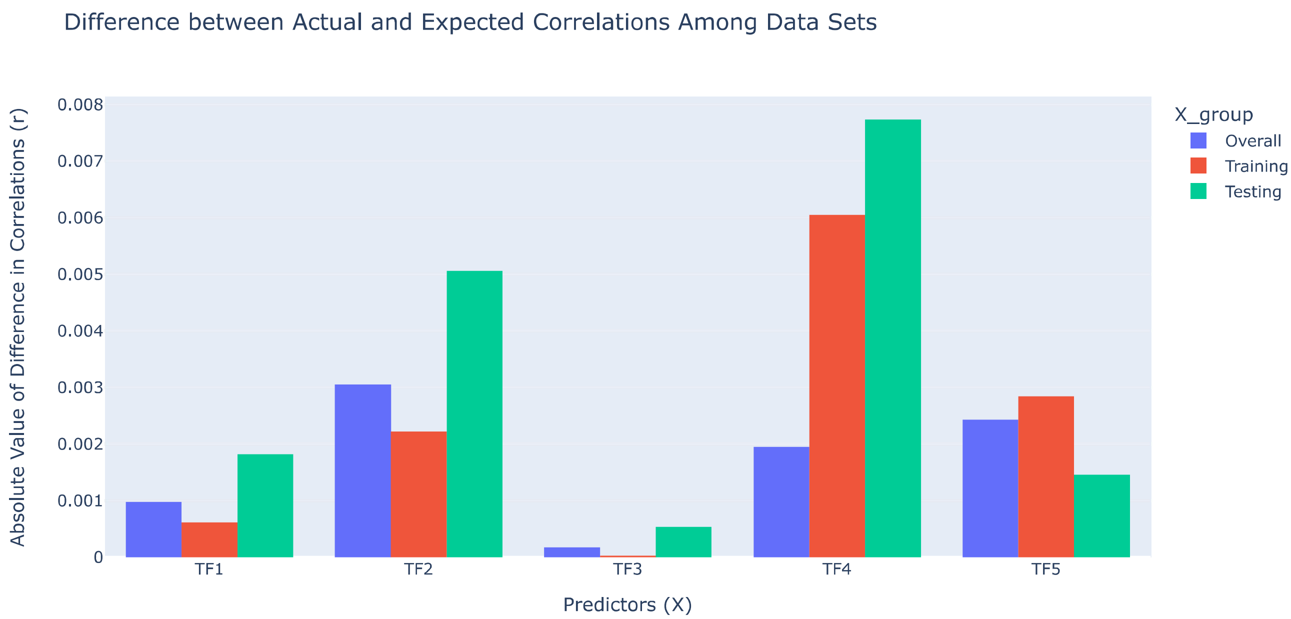


In [23]:

```
fig = px.histogram(combined_correlations_df, x="predictor", y="difference",
             color='X_group', barmode='group',
            title = "Difference between Actual and Expected Correlations Among Data Sets")

# Rename axis labels
fig.update_xaxes(title_text='Predictors (X)')
fig.update_yaxes(title_text='Absolute Value of Difference in Correlations (r)')

fig.show()
```



Please note that we can also view specific data points in the training and testing data for our predictors by accessing the *combined_train_test_x_and_y_df* dataframe and by visualizing scatterplots of X versus y values for each predictor (by calling the *view_train_vs_test_data_for_predictor* class method) as below:

In [24]:

```
combined_train_test_x_and_y_df = dummy_data.combined_train_test_x_and_y_df
combined_train_test_x_and_y_df
```

Out[24]:

	TF1	TF2	TF3	TF4	TF5	info	y
0	0.012406	0.922764	-0.247662	0.197411	-0.160290	training	0.442239
1	0.683566	-1.944551	1.126392	-0.117505	-0.222134	training	0.666132
2	-0.874310	0.259708	-0.844884	1.606723	1.962838	training	-1.263846
3	0.227996	-0.365244	0.612245	-0.147671	-0.297991	training	-0.008571
4	1.065428	-0.477365	-0.924536	-0.645075	-0.356331	training	0.664666
...	...	...	...	...	...	...	...
14995	-0.449012	0.806672	-1.684291	-0.629418	0.280621	testing	-0.789841
14996	0.709095	1.189145	1.564877	-0.560968	-0.644792	testing	0.777740
14997	0.712328	1.603306	1.135285	0.622087	0.454745	testing	0.739777
14998	-0.108016	-0.358597	1.055580	-0.877273	-1.142462	testing	0.432590
14999	-0.035035	0.879663	-0.256335	-1.857476	-0.349704	testing	0.342612

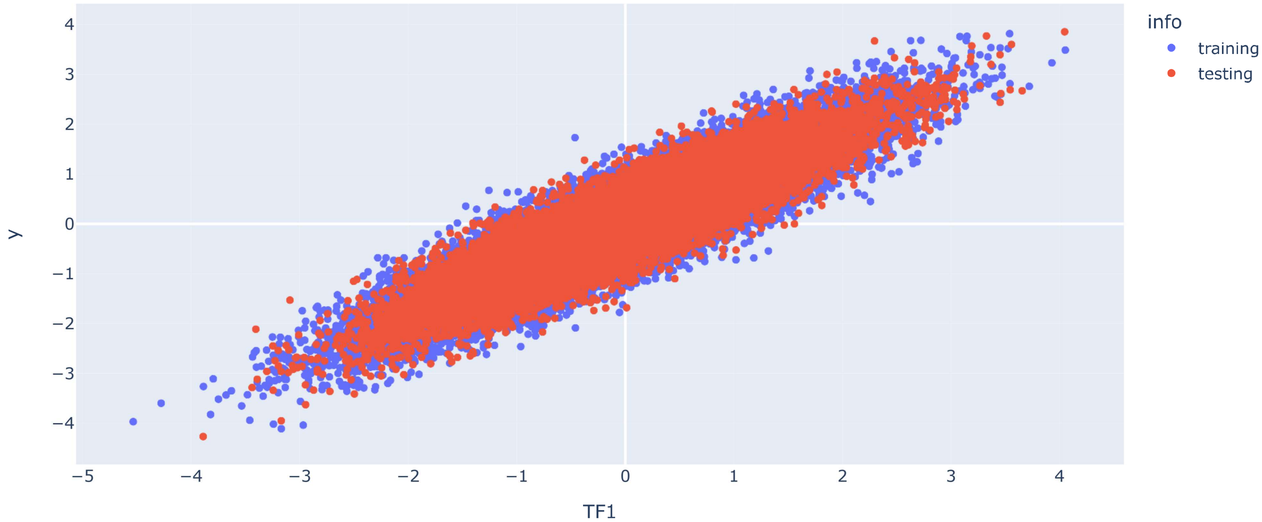
50000 rows × 7 columns

In [25]:

```
1 dummy_data.view_train_vs_test_data_for_predictor("TF1")
```

```
  predictor  actual_corr   X_group   num_samples
0       TF1     0.900976   Overall  unique 50000
0       TF1     0.900615  Training  unique 35000
0       TF1     0.901819   Testing  unique 15000
```

Training Versus Testing Data Points for Predictor: TF1

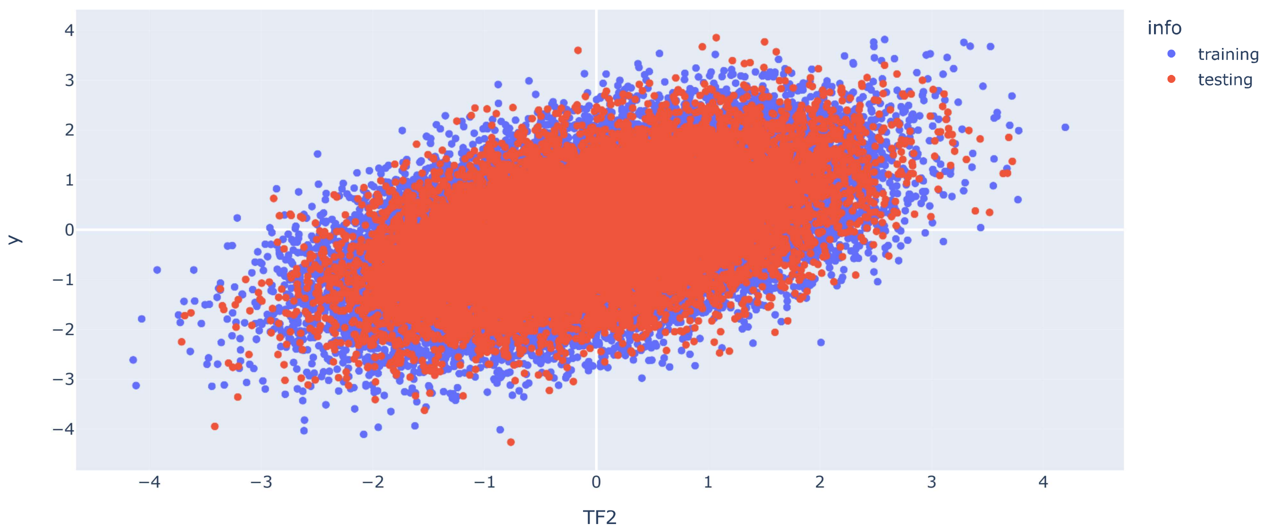


In [26]:

```
1 dummy_data.view_train_vs_test_data_for_predictor("TF2")
```

```
  predictor  actual_corr   X_group   num_samples
1       TF2     0.496949   Overall  unique 50000
1       TF2     0.497784  Training  unique 35000
1       TF2     0.494940   Testing  unique 15000
```

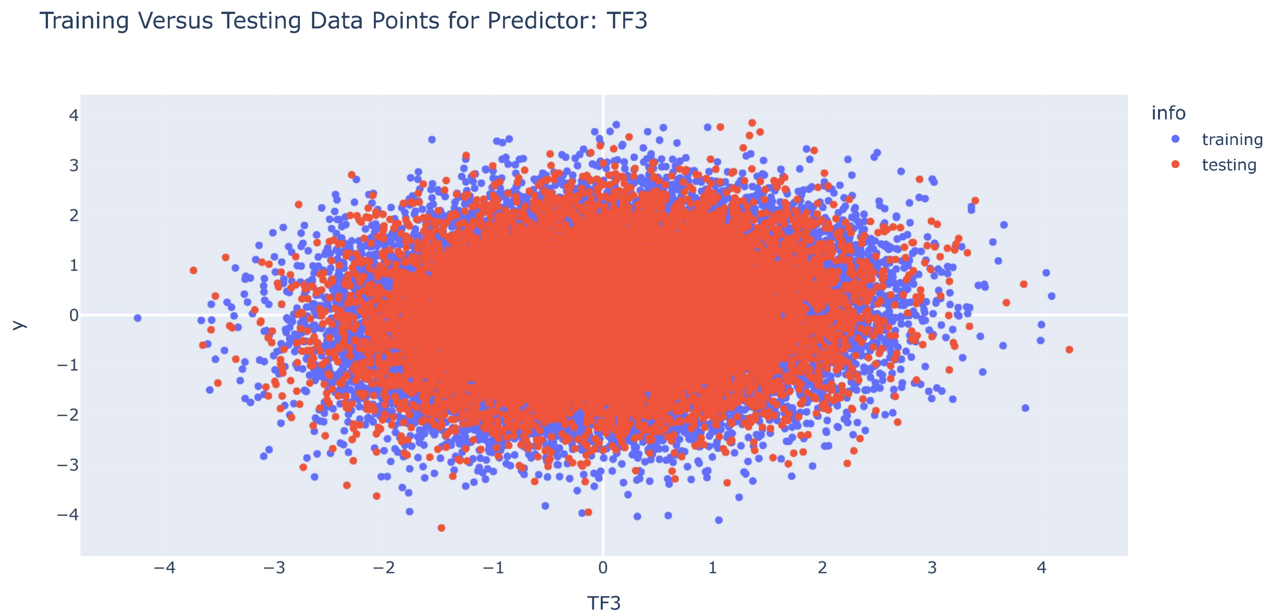
Training Versus Testing Data Points for Predictor: TF2



In [27]:

```
dummy_data.view_train_vs_test_data_for_predictor("TF3")
```

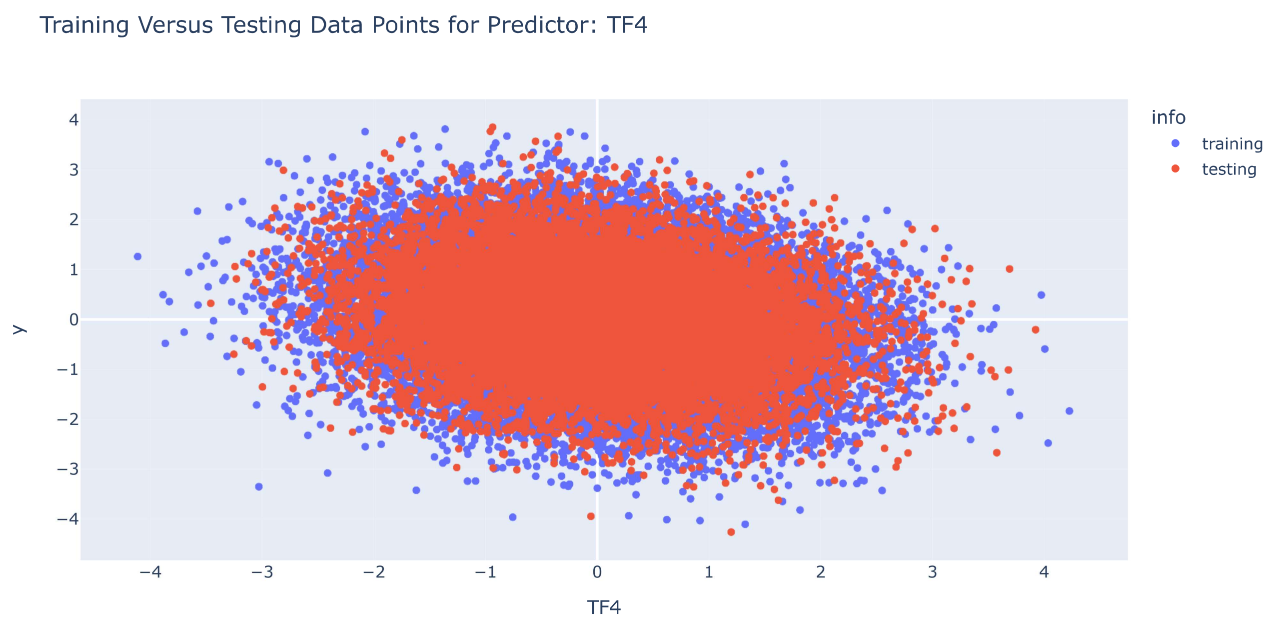
```
  predictor  actual_corr    X_group   num_samples
2       TF3     0.100172    Overall  unique 50000
2       TF3     0.100018   Training  unique 35000
2       TF3     0.100533    Testing  unique 15000
```



In [28]:

```
dummy_data.view_train_vs_test_data_for_predictor("TF4")
```

```
  predictor  actual_corr    X_group   num_samples
3       TF4    -0.201952    Overall  unique 50000
3       TF4    -0.206051   Training  unique 35000
3       TF4    -0.192267    Testing  unique 15000
```



In [29]:
```
dummy_data.view_train_vs_test_data_for_predictor("TF5")
```

	predictor	actual_corr	X_group	num_samples
4	TF5	-0.802429	Overall	unique 50000
4	TF5	-0.802839	Training	unique 35000
4	TF5	-0.801457	Testing	unique 15000