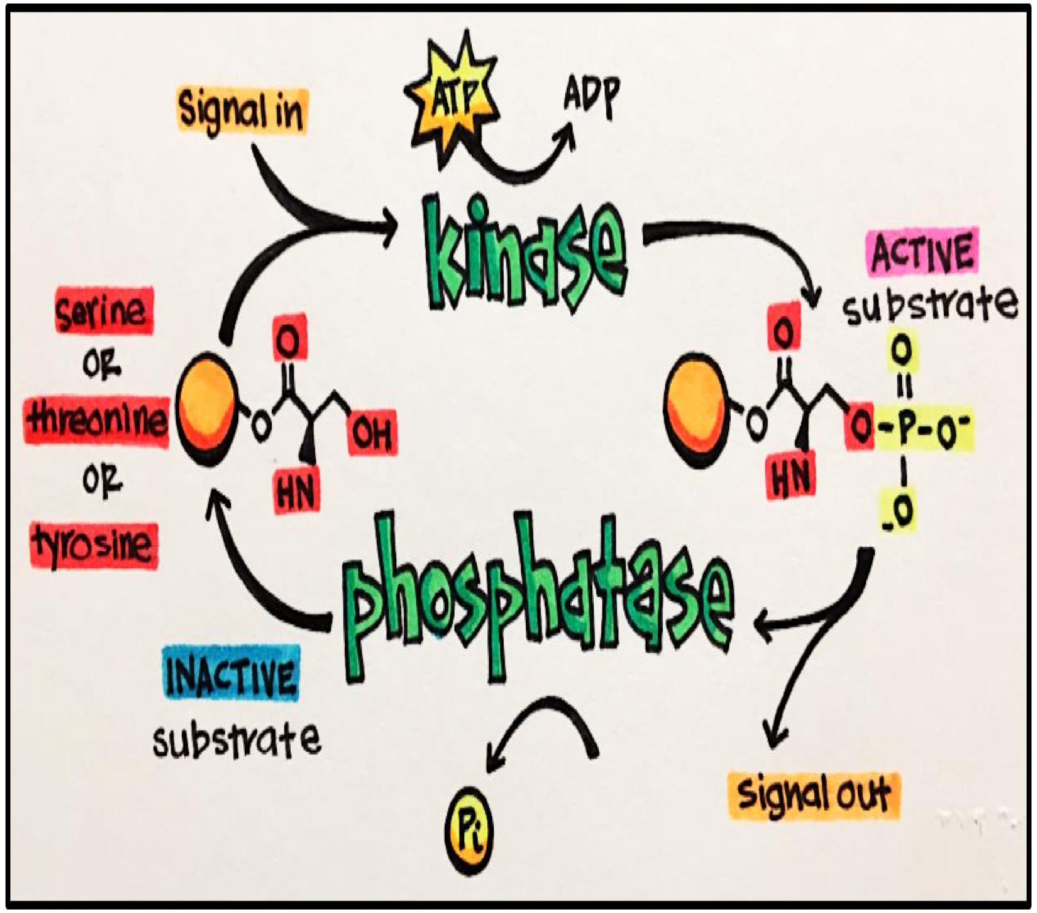
Compare the effectiveness of using embedding vectors versus sequence similarity for distinguishing kinase and phosphatase proteins family.

Presented by: Kiran Adhikari

Introduction

- Kinases and phosphatases play crucial roles in cellular signaling by adding or removing phosphate groups, respectively, from substrate proteins.
- Despite their opposing functions, these protein families often exhibit similarities at the sequence and structural levels due to shared substrate binding and catalytic mechanisms.
- The main role of kinases and phosphatases is to regulate post-translational modifications of proteins, which are essential to govern cellular signaling networks.
- Their interplay in phosphorylation and dephosphorylation processes is crucial for maintaining cellular homeostasis and responding to environmental cues and stimuli.

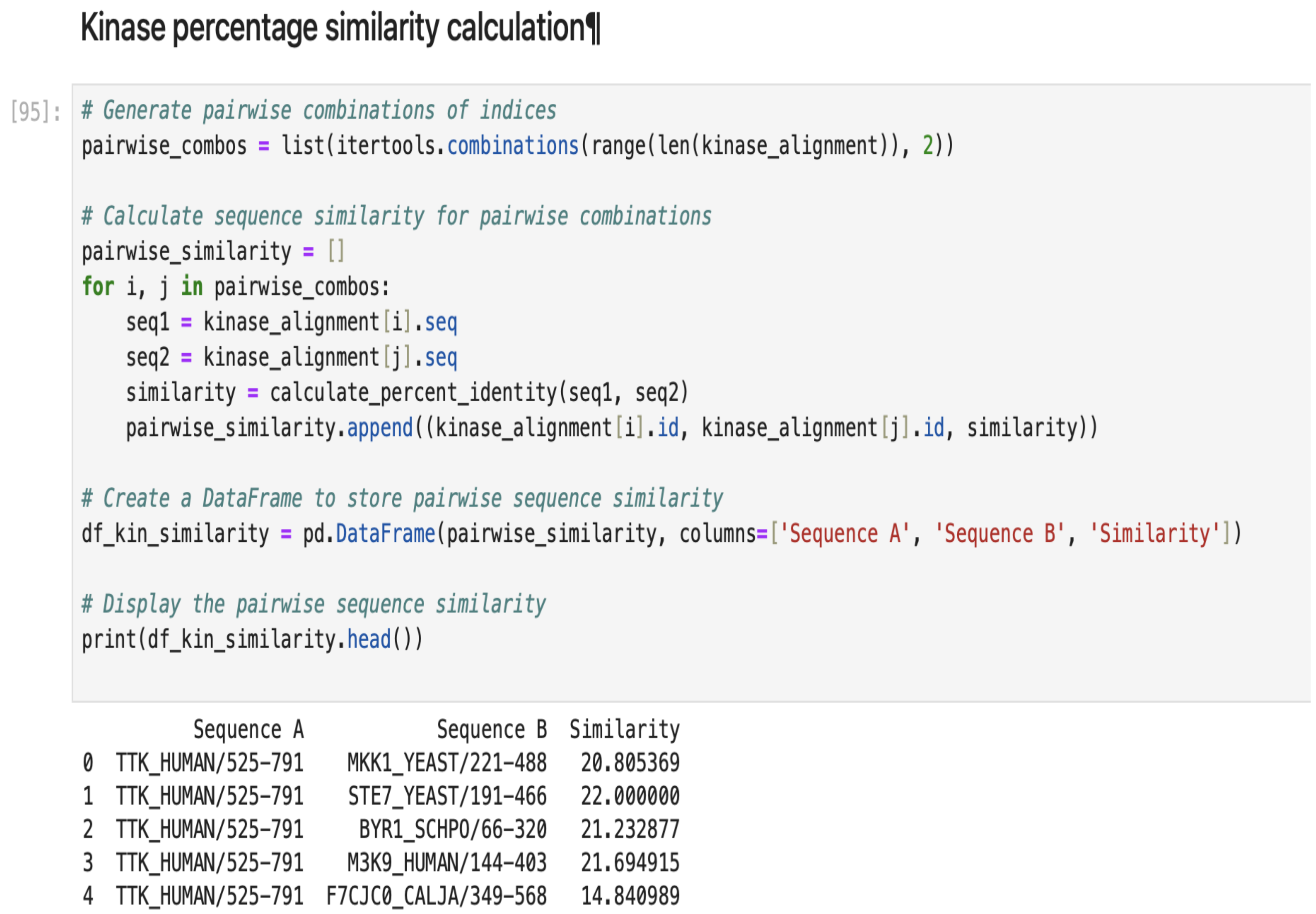


Objectives

- Calculate pairwise Percentage sequence similarity. When percentage sequence identity between their two sequences is sufficiently high, it is believed that pairs of protein have similar structures.
- Using embedding vectors as opposed the traditional way of using sequence similarity to find how well the 2 protein families can be distinguished.
- Using sequence similarity and embedding vector we can find if protein similar to its family without going into the expensive procedures of systematic experimental structure determination.

Data processing: Aligned sequence files

- Stockholm format is a multiple sequence alignment format usec by Pfam,
- Calculate percent identity two sequences without gap consideration.



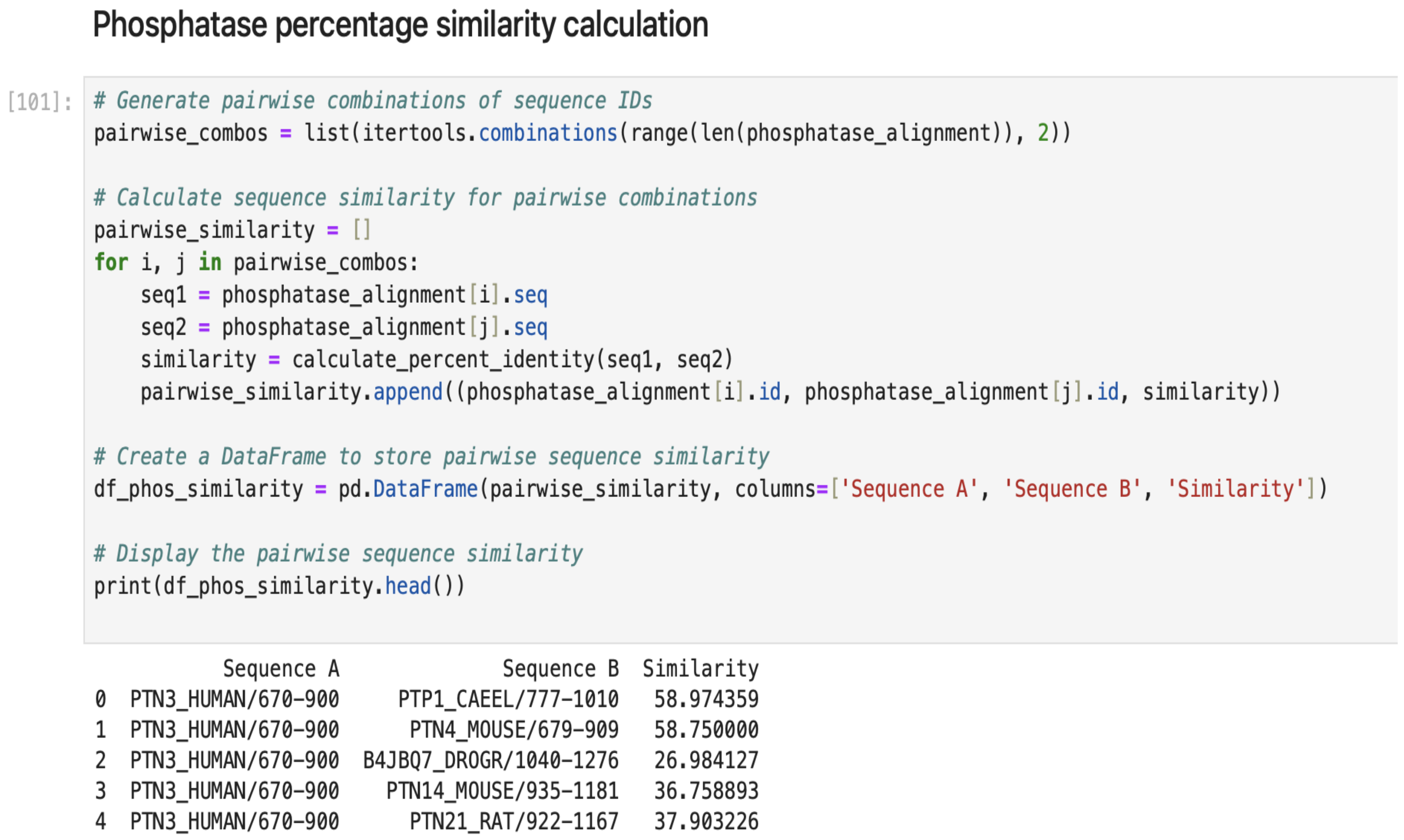
Kinase percentage similarity calculation¶

```
[95]: # Generate pairwise combinations of indices
pairwise_combos = list(itertools.combinations(range(len(kinase_alignment)), 2))

# Calculate sequence similarity for pairwise combinations
pairwise_similarity = []
for i, j in pairwise_combos:
    seq1 = kinase_alignment[i].seq
    seq2 = kinase_alignment[j].seq
    similarity = calculate_percent_identity(seq1, seq2)
    pairwise_similarity.append((kinase_alignment[i].id, kinase_alignment[j].id, similarity))

# Create a DataFrame to store pairwise sequence similarity
df_kin_similarity = pd.DataFrame(pairwise_similarity, columns=['Sequence A', 'Sequence B', 'Similarity'])

# Display the pairwise sequence similarity
print(df_kin_similarity.head())
```

```
          Sequence A          Sequence B  Similarity
0  TTK_HUMAN/525-791   MKK1_YEAST/221-488   20.805369
1  TTK_HUMAN/525-791   STE7_YEAST/191-466   22.000000
2  TTK_HUMAN/525-791    BYR1_SCHPO/66-320   21.232877
3  TTK_HUMAN/525-791   M3K9_HUMAN/144-403   21.694915
4  TTK_HUMAN/525-791 F7CJC0_CALJA/349-568   14.840989
```

Data processing: Aligned sequence files

- Average similarity for kin 22.990478569990916
- Average similarity for phosphatase: 26.545583996148252

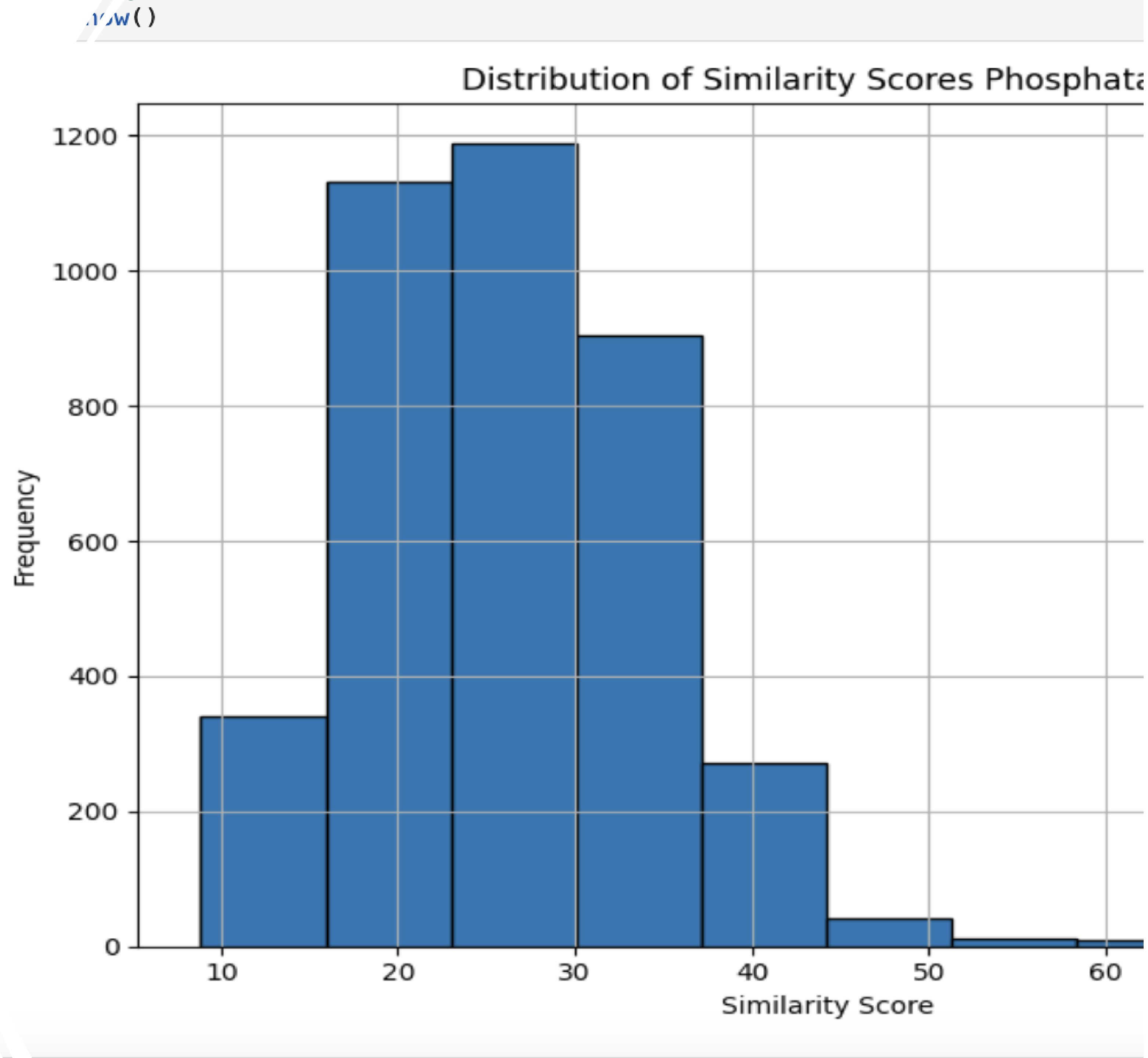


Phosphatase percentage similarity calculation

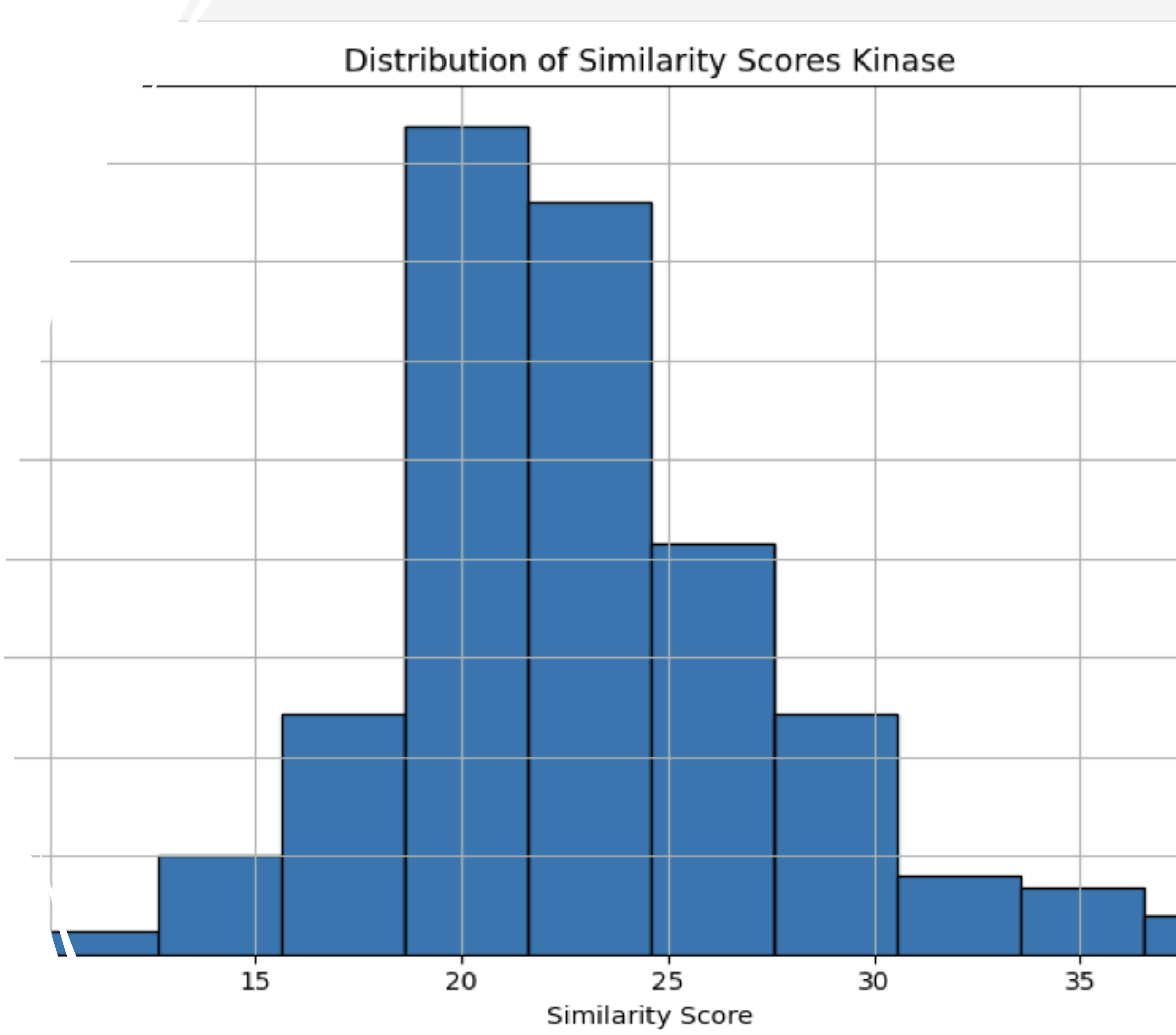
```
[101]: # Generate pairwise combinations of sequence IDs
pairwise_combos = list(itertools.combinations(range(len(phosphatase_alignment)), 2))

# Calculate sequence similarity for pairwise combinations
pairwise_similarity = []
for i, j in pairwise_combos:
    seq1 = phosphatase_alignment[i].seq
    seq2 = phosphatase_alignment[j].seq
    similarity = calculate_percent_identity(seq1, seq2)
    pairwise_similarity.append((phosphatase_alignment[i].id, phosphatase_alignment[j].id, similarity))

# Create a DataFrame to store pairwise sequence similarity
df_phos_similarity = pd.DataFrame(pairwise_similarity, columns=['Sequence A', 'Sequence B', 'Similarity'])

# Display the pairwise sequence similarity
print(df_phos_similarity.head())
```

```
          Sequence A           Sequence B  Similarity
0  PTN3_HUMAN/670-900    PTP1_CAEEL/777-1010   58.974359
1  PTN3_HUMAN/670-900     PTN4_MOUSE/679-909   58.750000
2  PTN3_HUMAN/670-900  B4JBQ7_DROGR/1040-1276   26.984127
3  PTN3_HUMAN/670-900    PTN14_MOUSE/935-1181   36.758893
4  PTN3_HUMAN/670-900     PTN21_RAT/922-1167   37.903226
```

Histogram of Sequence Similarity for Phosphatase



Bar Chart of Sequence Similarity for Kinase



Data Analysis: Embedding vectors.

Cosine similarity

Computed all-vs-all embedding vector similarities using cosine similarity.

For this embedding vector of Kinase and Phosphatase protein family were used.

Cosine similarity are useful to find how similar the data objects irrespective of their size, mostly vectors and it focus on the angle between two vectors.

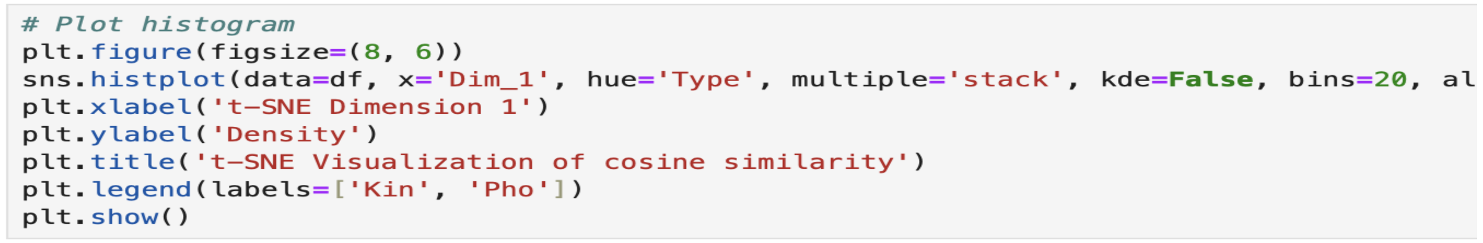
The smaller the angle, higher the cosine similarity.

```
# Get lengths of kinase embedding vector
n_kin = len(kinase_embedding_vectors)

# Concatenate embedding vectors
all_embedding_vectors = np.concatenate([kinase_embedding_vectors, phosphatase_embedding_vectors])

# Calculate cosine similarity
cos_similarity = cosine_similarity(all_embedding_vectors, all_embedding_vectors)
```

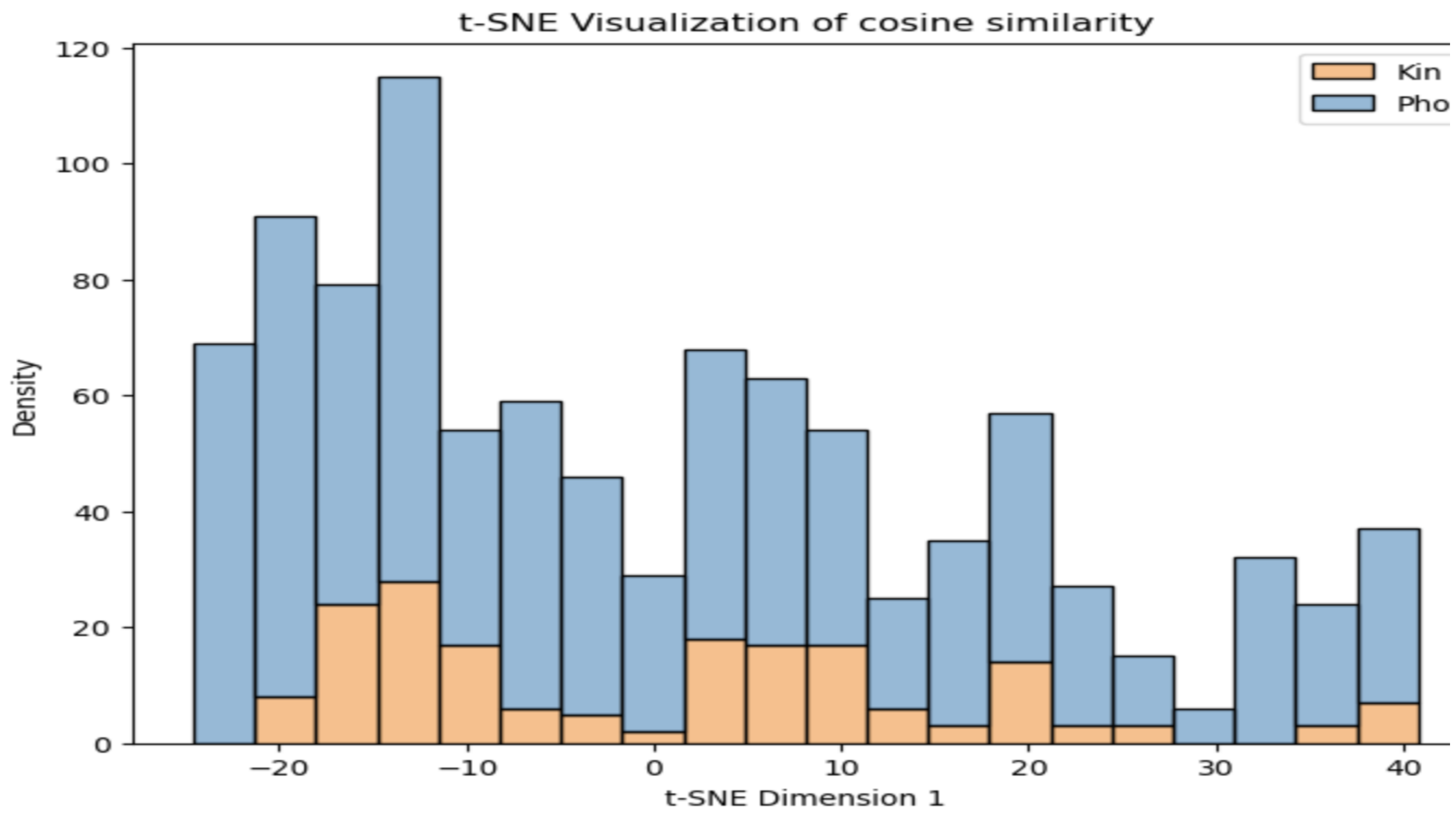
t-SNE

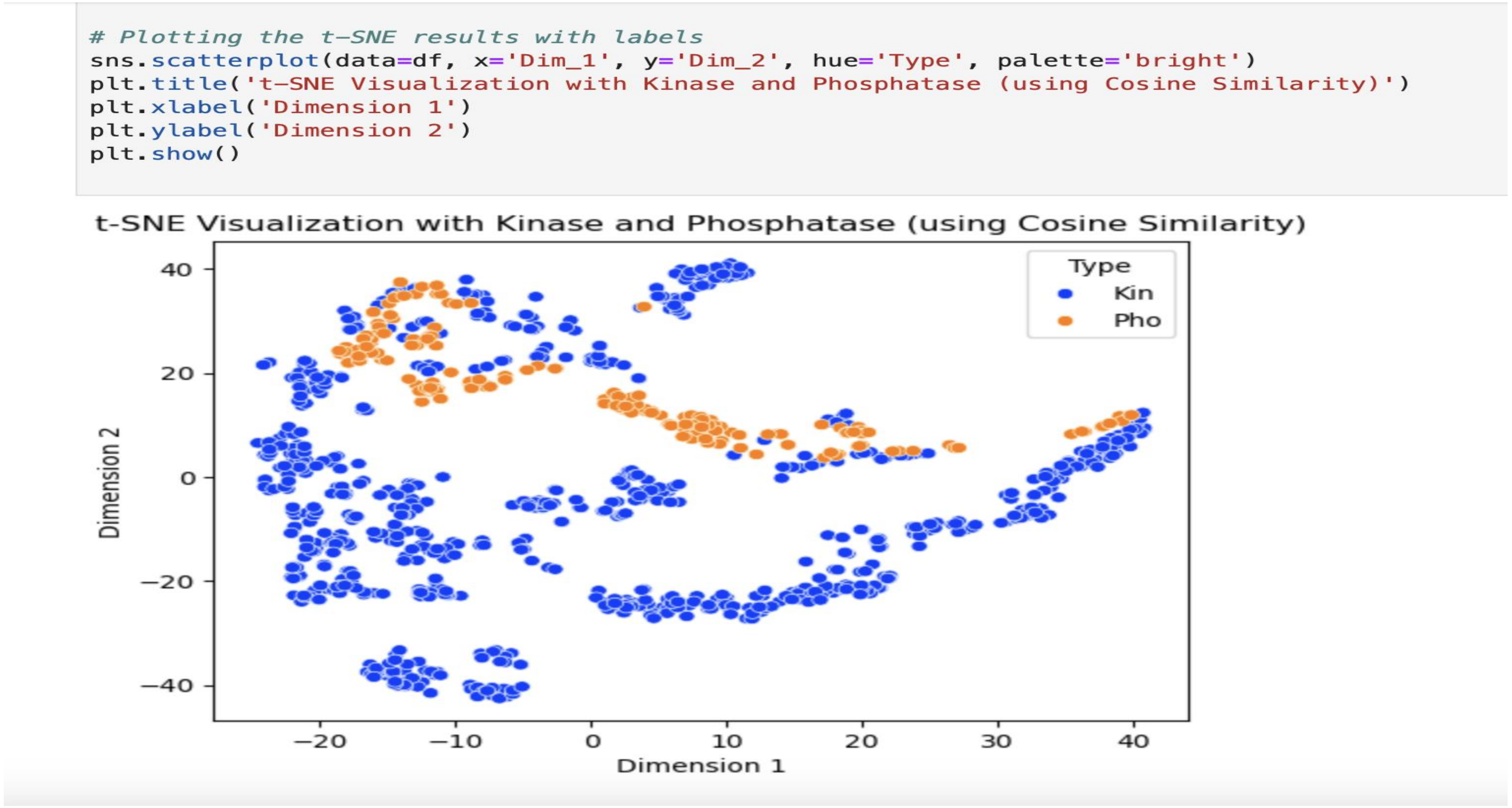
t-SNE is powerful technique for dimensionality reduction and data visualization.

- It is non-linear dimensionality reduction method in which algorithm allows us to separate data that cannot be separated by a linear line. t-SNE allows us to preserve relationship of pairwise similarities between vectors in a lower dimensional space.
- Here I visualized the data in one dimensional to understand the underlying pattern.

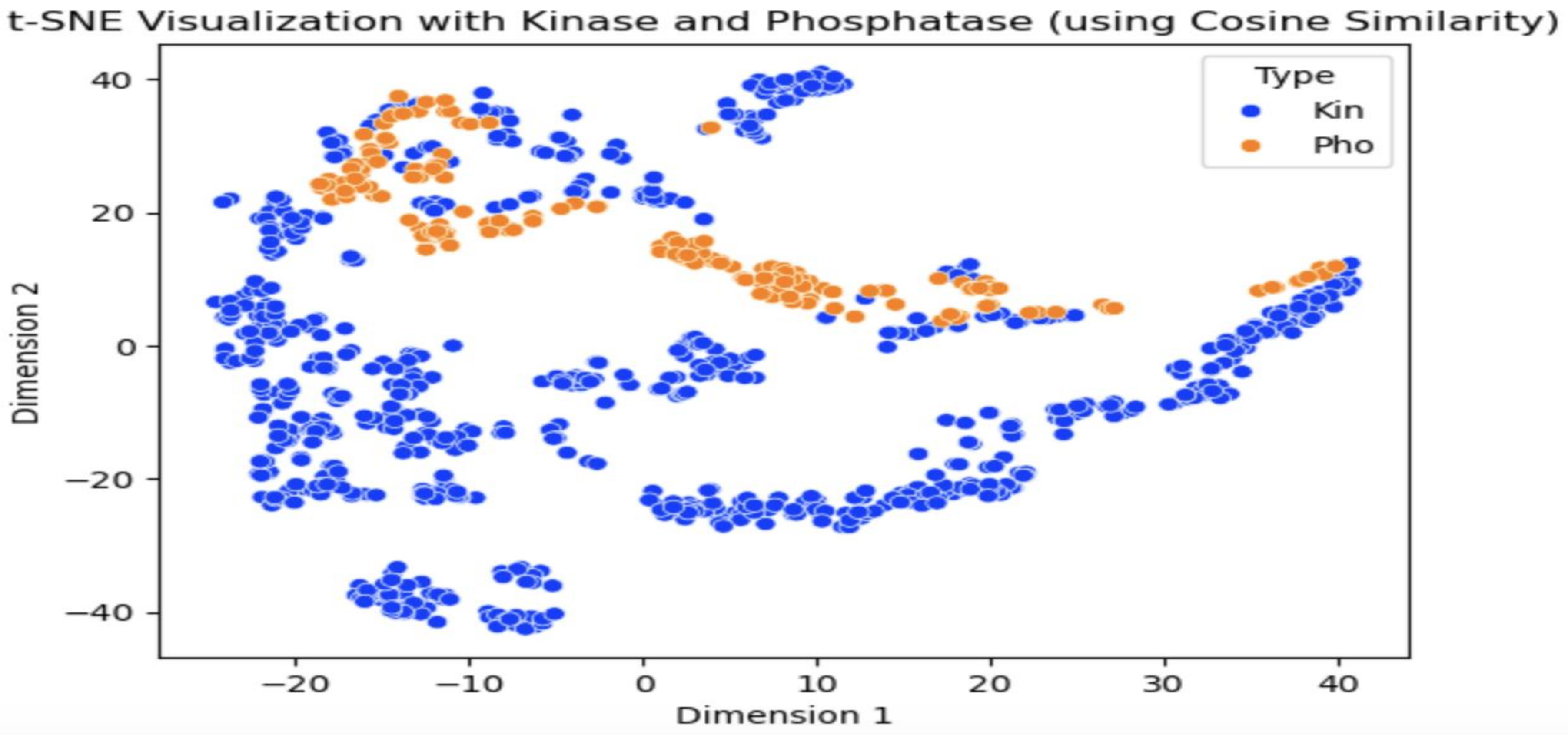


```
# Plot histogram
plt.figure(figsize=(8, 6))
sns.histplot(data=df, x='Dim_1', hue='Type', multiple='stack', kde=False, bins=20, al
plt.xlabel('t-SNE Dimension 1')
plt.ylabel('Density')
plt.title('t-SNE Visualization of cosine similarity')
plt.legend(labels=['Kin', 'Pho'])
plt.show()
```

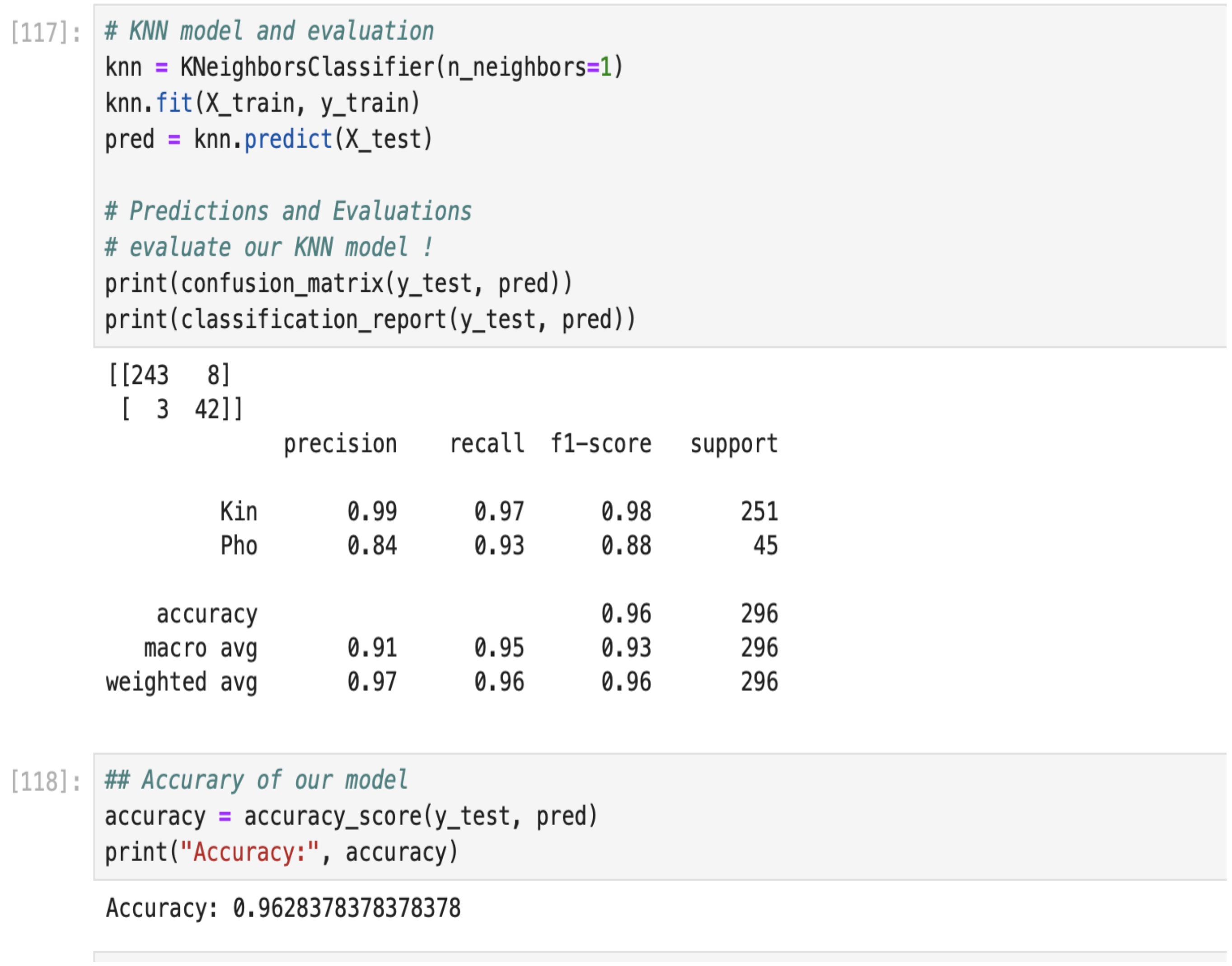




```
# Plotting the t-SNE results with labels
sns.scatterplot(data=df, x='Dim_1', y='Dim_2', hue='Type', palette='bright')
plt.title('t-SNE Visualization with Kinase and Phosphatase (using Cosine Similarity)')
plt.xlabel('Dimension 1')
plt.ylabel('Dimension 2')
plt.show()
```



KNN model
We are performing KNN supervised ML algorithm to classify kinase and Phosphatase family. Target Variable is y = df['Type'] Independent variable is X = df[['PosX', 'PosY']]



```
[117]: # KNN model and evaluation
       knn = KNeighborsClassifier(n_neighbors=1)
       knn.fit(X_train, y_train)
       pred = knn.predict(X_test)

       # Predictions and Evaluations
       # evaluate our KNN model !
       print(confusion_matrix(y_test, pred))
       print(classification_report(y_test, pred))
```

```
[[243   8]
 [  3  42]]
              precision    recall  f1-score   support

         Kin       0.99      0.97      0.98       251
         Pho       0.84      0.93      0.88        45

    accuracy                           0.96       296
   macro avg       0.91      0.95      0.93       296
weighted avg       0.97      0.96      0.96       296
```

```
[118]: ## Accurary of our model
       accuracy = accuracy_score(y_test, pred)
       print("Accuracy:", accuracy)
```

```
Accuracy: 0.9628378378378378
```

Conclusion:

From KNN Model:

- True Positive (TP): 243 kinase samples were correctly classified as kinases.
- False Positive (FP): 8 phosphate samples were incorrectly classified as kinases.
- False Negative (FN): 3 kinase samples were incorrectly classified as phosphates.
- True Negative (TN): 42 phosphate samples were correctly classified as phosphates.

- F1-Score:
- F1-Score for kinases: Harmonic mean of precision and recall for kinases ≈ 0.98
- F1-Score for phosphates: Harmonic mean of precision and recall for phosphates ≈ 0.88.