Grading

This week's lab doesn't have any auto-graded components. Each question in this notebook has an accompanying Peer Review question. Although the lab shows as being ungraded, you need to complete the notebook to answer the Peer Review questions.
DO NOT CHANGE VARIABLE OR METHOD SIGNATURES

Validate Button

This week's lab doesn't have any auto-graded components. Each question in this notebook has an accompanying Peer Review question. Although the lab shows as being ungraded, you need to complete the notebook to answer the Peer Review questions.

You do not need to use the Validate button for this lab since there are no auto-graded components. If you hit the Validate button, it will time out given the number of visualizations in the notebook. Cells with longer execution times cause the validate button to time out and freeze. ***This notebook's Validate button time-out does not affect the final submission grading.***

Clustering RNA sequences to identify cancer types

In this assignment, we will use clustering algorithms on RNA sequence data to identify cancer types. Since the whole data (https://www.synapse.org/#!Synapse:syn4301332) (from Cancer Genome Atlas Pan-Cancer project (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3919969/)) is very big, we will use a subset data from UCI Machine Learning repository (https://archive.ics.uci.edu/ml/datasets/gene+expression+cancer+RNA-Seq#). The subset data contains only 5 labels; BRCA, KIRC, COAD, LUAD and PRAD. The meanings of those labels are as below.

Abbreviation	Cancer
LUSC	Lung squamous cell carcinoma
READ	Rectum adenocarcinoma
GBM	Glioblastoma multiforme
BLCA	Bladder Urothelial Carcinoma
UCEC	Uterine Corpus Endometrioid Carcinoma
COAD	Colon adenocarcinoma
OV	Ovarian serous cystadenocarcinoma
LAML	Acute Myeloid Leukemia
HNSC	Head and Neck squamous cell carcinoma
LUAD	Lung adenocarcinoma
BRCA	Breast invasive carcinoma
KIRC	Kidney renal clear cell carcinoma

Although we can use the data for supervised learning model training, we will not use these labels for training, but use them for evaluation.

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from sklearn.cluster import AgglomerativeClustering, KMeans
from sklearn.metrics import accuracy_score, confusion_matrix
import time
```

In [2]:
```
# Read data. Do not change the variable names (data, label)
data = pd.read_csv('data/data.csv')
label = pd.read_csv('data/labels.csv')
data=data.drop('Unnamed: 0',axis=1)
label=label.drop('Unnamed: 0',axis=1)
```

A. [Peer Review] Perform basic data inspection or EDA on the pandas dataframe.

- How many observations?
- How many features?

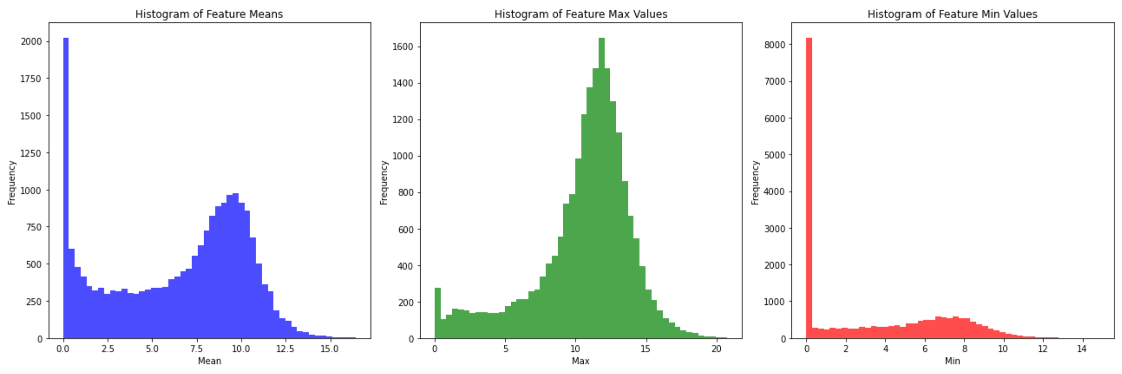
In [3]:
```
# perform basic data inspection such as getting the number of observation
s and number of features
# you can also display part of the dataframe or run data.info()
# your code here
# Perform basic data inspection
# Display part of the dataframe and run data.info()
print("Data Information:")
print(data.info())
print("\nData Head:")
print(data.head())

print("\nLabel Information:")
print(label.info())
print("\nLabel Head:")
print(label.head())

# Number of observations and features
num_observations = data.shape[0]
num_features = data.shape[1]

print(f"\nNumber of observations: {num_observations}")
print(f"Number of features: {num_features}")
```

```
Data Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 801 entries, 0 to 800
Columns: 20531 entries, gene_0 to gene_20530
dtypes: float64(20531)
memory usage: 125.5 MB
None

Data Head:
   gene_0    gene_1    gene_2    gene_3     gene_4  gene_5    gene_6  \
0     0.0  2.017209  3.265527  5.478487  10.431999     0.0  7.175175   
1     0.0  0.592732  1.588421  7.586157   9.623011     0.0  6.816049   
2     0.0  3.511759  4.327199  6.881787   9.870730     0.0  6.972130   
3     0.0  3.663618  4.507649  6.659068  10.196184     0.0  7.843375   
4     0.0  2.655741  2.821547  6.539454   9.738265     0.0  6.566967   

     gene_7  gene_8  gene_9  ...  gene_20521  gene_20522  gene_20523  \
0  0.591871     0.0     0.0  ...    4.926711    8.210257    9.723516   
1  0.000000     0.0     0.0  ...    4.593372    7.323865    9.740931   
2  0.452595     0.0     0.0  ...    5.125213    8.127123   10.908640   
3  0.434882     0.0     0.0  ...    6.076566    8.792959   10.141520   
4  0.360982     0.0     0.0  ...    5.996032    8.891425   10.373790   

   gene_20524  gene_20525  gene_20526  gene_20527  gene_20528  gene_2052
9  \
0    7.220030    9.119813   12.003135    9.650743    8.921326    5.28675
9   
1    6.256586    8.381612   12.674552   10.517059    9.397854    2.09416
8   
2    5.401607    9.911597    9.045255    9.788359   10.090470    1.68302
3   
3    8.942805    9.601208   11.392682    9.694814    9.684365    3.29200
1   
4    7.181162    9.846910   11.922439    9.217749    9.461191    5.11037
2   

   gene_20530  
0         0.0  
1         0.0  
2         0.0  
3         0.0  
4         0.0  

[5 rows x 20531 columns]

Label Information:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 801 entries, 0 to 800
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype 
---  ------  --------------  ----- 
 0   Class   801 non-null    object
dtypes: object(1)
memory usage: 6.4+ KB
None

Label Head:
  Class
0  PRAD
1  LUAD
```

```
2   PRAD
3   PRAD
4   BRCA

Number of observations: 801
Number of features: 20531
```

- Draw histograms of mean, max and min values in each feature. You may see numbers around 0-20. What do those numbers mean? (We do not expect students to know or figure out the meanings, but if you do know by chance, feel free to discuss them with the class on the discussion board.)
  Answer the Peer Review question about this section.

In [4]:
```python
# draw histograms of mean, max and min values in each feature
# your code here
# Calculate mean, max, and min values for each feature
feature_means = data.mean(axis=0)
feature_max = data.max(axis=0)
feature_min = data.min(axis=0)

# Plot histograms
plt.figure(figsize=(18, 6))

plt.subplot(1, 3, 1)
plt.hist(feature_means, bins=50, color='blue', alpha=0.7)
plt.title('Histogram of Feature Means')
plt.xlabel('Mean')
plt.ylabel('Frequency')

plt.subplot(1, 3, 2)
plt.hist(feature_max, bins=50, color='green', alpha=0.7)
plt.title('Histogram of Feature Max Values')
plt.xlabel('Max')
plt.ylabel('Frequency')

plt.subplot(1, 3, 3)
plt.hist(feature_min, bins=50, color='red', alpha=0.7)
plt.title('Histogram of Feature Min Values')
plt.xlabel('Min')
plt.ylabel('Frequency')

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



- If we were to train a "supervised" learning model, how would you deal with such large feature dimension?
- Even after feature dimension reduction, still the number of useful features may be enormous. How it would impact performance or runtime of certain supervised learning algorithms? Which algorithms would suffer from high dimension features than others and why?
- How it would impact performance or runtime of an unsupervised learning algorithm?
- Draw histograms of mean, max and min values in each feature. You may see numbers around 0-20. What those numbers mean? (We do not expect students to know or figure out the meanings, but if you do know by chance, feel free to discuss them with the class on the discussion board.)

  Anwer these questions in this week's Peer Review assignment.

B. Discussion and Analysis

1. Supervised Learning and High Dimensional Data

How to Deal with Large Feature Dimension:

- **Dimensionality Reduction:** Techniques such as PCA (Principal Component Analysis) or LDA (Linear Discriminant Analysis) can be used to reduce the feature dimensions while retaining most of the variance or discriminative information.
- **Feature Selection:** Methods such as mutual information, correlation analysis, and various statistical tests can help in selecting the most relevant features.

Impact on Performance and Runtime:

- **High Dimensionality Impact:** High-dimensional data can lead to the "curse of dimensionality," which can negatively affect the performance of supervised learning models. It increases the computational cost and can cause overfitting.
- **Algorithms Affected:**
  - **Decision Trees and Random Forests:** May suffer from overfitting and increased computational cost due to high dimensionality.
  - **K-Nearest Neighbors (KNN):** Performance degrades as the number of dimensions increases because distance calculations become less meaningful in high-dimensional spaces.
  - **Linear Models (e.g., Logistic Regression, SVM):** May perform better with proper regularization but can still suffer from increased computational cost.

2. Unsupervised Learning and High Dimensional Data

Impact on Performance and Runtime:

- **Agglomerative Clustering:** May suffer from increased computational cost as it involves calculating distances between all pairs of points.
- **K-Means Clustering:** High-dimensional data can lead to less meaningful cluster centroids, making it harder for the algorithm to converge to a meaningful solution.

Histograms of Feature Values

Interpretation of Feature Values:

- The feature values being around 0-20 likely indicate normalized or scaled data. These values might represent gene expression levels which are often transformed for analysis purposes.

The exact biological or clinical significance of these values would depend on the context of the specific RNA sequences and cancer types. However, for our analysis, understanding the statistical distribution is more pertinent for preprocessing and model selection.

B. [Peer Review] Build a hierarchical clustering model

Let's build a model using hierarchical clustering. Hierarchical clustering module is available from `sklearn.cluster.AgglomerativeClustering`. You can choose linkage type and metric. Please check its documentation for more details.

a) Number of clusters vs distance threshold Oftentimes hierarchical clustering does not need to know the number of clusters in advance. Instead, one needs to choose threshold distance/similarity to cut the dendrogram later. The AgglomerativeClustering module lets you specify either the number of clusters (n_clusters) or the threshold (distance_threshold). Based on our data, which should we choose to set to which value and why?

Answer this question in the Peer Review assignment.

In hierarchical clustering, the choice between specifying the number of clusters (`n_clusters`) or using a distance threshold (`distance_threshold`) depends on your data characteristics and your specific objectives. Here’s a breakdown of each approach and when to use them:

1. Specifying the Number of Clusters (`n_clusters`)

When you set `n_clusters` , you directly control the number of clusters the algorithm should produce. This approach is beneficial if you have prior knowledge or an expectation about the number of clusters.

When to Use:

- You have domain knowledge or a clear business requirement for a specific number of clusters.
- The goal is to match a known number of classes or categories.
- You’ve explored the data and determined an ideal cluster count using metrics like the silhouette score, dendrogram inspection, or the elbow method.

Limitations:

- It requires prior knowledge or exploratory analysis.
- If your data structure isn’t well-separated into distinct clusters, forcing a fixed number might result in poor clusters.

2. Using a Distance Threshold (`distance_threshold`)

The `distance_threshold` parameter controls the level at which you "cut" the dendrogram, leading to a more data-driven cluster determination. This is especially useful when you want to discover natural groupings in the data without a fixed number of clusters in mind.

When to Use:

- You want a flexible, data-driven clustering approach.
- You’re unsure about the number of clusters and prefer to let the model determine it based on the distance.
- Your data naturally forms hierarchical relationships, where clusters merge progressively at increasing distances.

Limitations:

- Requires some experimentation to choose an appropriate threshold.
- The final number of clusters can be unpredictable and might require tuning the threshold.

How to Decide:

1. **Explore the Data:**
    - Visualize the dendrogram using the `scipy.cluster.hierarchy.dendrogram` function to see at what distance clusters merge. If you notice a natural "gap" in distances where clusters should stop merging, this indicates a good cut point for a `distance_threshold` .
2. **Data Characteristics:**
    - If the data structure is complex and doesn’t clearly suggest a fixed number of clusters, use `distance_threshold` .

  - If the data aligns well with a known cluster count, specify `n_clusters`.

Summary:

- If the focus is on discovering natural clusters with minimal assumptions, use **`distance_threshold`**.
- If the objective is a pre-determined number of clusters based on prior knowledge, use **`n_clusters`**.

Given the nature of the RNA sequence data and the clustering objective (identifying cancer types), it’s important to carefully consider how hierarchical clustering is applied.

Hierarchical Clustering Considerations

1. **High Dimensionality (20,531 Features):** RNA sequence data is very high-dimensional, and not all features will be relevant for clustering. Hierarchical clustering, especially with such a large feature space, might struggle with performance and meaningful cluster formation due to the curse of dimensionality.
2. **Feature Selection/Dimensionality Reduction:** Before applying clustering, consider using dimensionality reduction techniques like PCA (Principal Component Analysis) or selecting important features through variance filtering, as this can improve cluster formation.
3. **Cluster Label Interpretation:** You aim to cluster RNA sequences into the five known labels (BRCA, KIRC, COAD, LUAD, PRAD). While you could set the number of clusters directly, this overlooks the flexibility of hierarchical clustering. A distance threshold might be better suited since it lets the data determine the optimal number of clusters.

Deciding Between `n_clusters` and `distance_threshold`

- **`n_clusters`** : Setting `n_clusters=5` aligns directly with the known labels. This is practical if you want to enforce clustering into five groups.
- **`distance_threshold`** : Letting the algorithm determine the number of clusters based on a distance threshold can reveal whether the data naturally forms five clusters, or if additional subtypes exist.

Suggested Approach

1. **Dimensionality Reduction**: Perform PCA to reduce the number of features while retaining most of the variance.
2. **Hierarchical Clustering**: Use `AgglomerativeClustering` with a `distance_threshold` . Start with `n_clusters=None` , allowing flexibility in the number of clusters, then inspect the resulting dendrogram to find a suitable threshold.

Key Points

- **Dimensionality Reduction**: Reducing the dimensionality (e.g., PCA) helps prevent noise from irrelevant features from affecting the clustering.
- **Distance Threshold**: Use the dendrogram to explore where a natural cut-off might exist. If the data supports a clear separation into 5 clusters, you can then proceed with the threshold-based approach.
- **Linkage Method**: The `'ward'` linkage is commonly used for minimizing variance within clusters and is suitable for continuous data like RNA sequences.

Conclusion

Given the complexity and high dimensionality of RNA sequence data, the more flexible distance threshold approach is recommended, combined with dimensionality reduction. This approach allows you to determine the most appropriate number of clusters based on the data's structure rather than imposing a fixed number.

b) Guess which metric?

Can you guess which metric to use (distance-based vs. similarity-based) and why? This question is not graded, but we encourage you to share your thoughts with the class. See the ungraded discussion prompt for this week's material.

In hierarchical clustering, choosing between a **distance-based** or **similarity-based** metric depends on the nature of your data and the relationships you expect to find between the samples. For RNA sequence data, which is continuous and numeric, a **distance-based metric** is generally more appropriate. Here's why:

1. Nature of the Data (Gene Expression Levels):

- The RNA sequence data represents gene expression levels across various genes, which are continuous, numerical features.
- For such data, **distance-based metrics** (like Euclidean, Manhattan, etc.) are better suited as they measure the differences in feature values between samples. This directly captures the notion of how different two samples are based on their gene expression profiles.

2. Interpreting Biological Variability:

- In biological data like gene expression, small differences in expression levels across multiple genes can be crucial in distinguishing cancer types. Distance-based metrics capture these differences directly.
- **Similarity-based metrics** (like cosine similarity or correlation) are more appropriate when dealing with data where the relationship between vectors matters more than the absolute values (e.g., when comparing text documents). In gene expression data, however, the magnitude of differences between expression levels is important.

3. Commonly Used Metrics in Similar Applications:

- In most bioinformatics applications, **Euclidean distance** is commonly used for clustering gene expression data. It effectively captures the straight-line distance between points in high-dimensional space, making it straightforward to interpret clusters.
- **Ward's linkage**, often combined with Euclidean distance, is popular because it minimizes variance within clusters, making it well-suited for biological data with continuous measurements.

Summary

For clustering RNA sequence data to identify cancer types:

- A **distance-based metric** is more appropriate.
- **Euclidean distance** is the most commonly used metric in similar gene expression analyses.

By focusing on distances, you can better capture the inherent differences in gene expression profiles, which is critical for distinguishing between different cancer types.

c) Build a model

Build a model using n_clusters=5 option. Choose any metric and linkage type at first. Display the clustering result labels (you can just print out the result). Do not change the variable (model) name. Answer the question about this section in the Peer Review.

```
In [5]: # build a model using n_clusters=5 option
        from sklearn.cluster import AgglomerativeClustering

        # Build the Agglomerative Clustering model
        model = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linka
        ge='ward')

        # Fit the model to the data
        model.fit(data)

        # Display the clustering result labels
        labels = model.labels_
        print("Clustering Result Labels:")
        print(labels)
```

```
Clustering Result Labels:
[2 3 2 2 0 2 1 2 0 2 0 1 2 0 0 0 3 1 1 2 0 1 3 0 1 3 4 0 0 0 0 0 1 0 2 0
1
 3 0 0 1 2 2 1 1 0 2 4 0 3 0 3 0 2 4 0 0 4 1 0 3 1 0 3 2 4 0 2 1 0 1 0 0
3
 0 3 0 1 2 4 0 2 0 0 2 2 0 0 1 0 2 2 0 0 0 2 4 0 2 0 0 1 0 1 3 1 3 4 3 3
2
 0 3 2 0 1 1 1 0 0 3 1 3 0 2 2 2 0 1 0 4 0 4 0 0 1 3 0 1 4 0 2 0 1 3 4 2
0
 3 3 3 3 0 0 3 0 0 2 2 3 2 3 1 0 2 3 4 1 3 0 1 3 0 3 0 0 0 2 0 1 4 1 0 2
2
 2 3 3 0 3 3 1 3 2 3 0 0 0 3 3 0 1 1 1 1 2 0 2 0 3 3 0 2 0 2 0 0 0 3 0 1
3
 1 1 3 0 1 2 0 3 3 2 4 0 1 2 1 4 0 1 1 3 2 2 3 3 1 0 0 4 0 2 4 0 2 1 2 2
2
 0 4 4 3 4 4 2 3 0 0 1 1 0 4 2 1 2 0 0 1 0 0 0 0 3 3 0 0 0 1 1 1 1 0 0 0
1
 0 0 3 2 0 0 4 3 2 0 0 0 4 0 2 0 4 3 3 2 1 0 1 1 3 4 1 0 0 0 0 1 0 0 2 0
1
 0 3 2 1 0 2 4 0 0 0 3 3 3 0 0 2 3 0 1 0 4 4 3 0 1 0 0 0 4 3 4 1 2 1 0 0
1
 0 4 2 3 2 0 1 2 0 4 1 1 4 4 2 0 0 4 1 3 2 0 0 0 3 3 1 3 0 1 4 2 0 3 2 0
0
 0 3 0 0 2 0 2 4 0 3 0 0 3 0 0 0 1 3 2 0 2 1 0 1 4 0 2 3 1 0 0 1 0 3 0 0
2
 4 0 1 3 2 0 2 0 0 0 0 1 3 0 1 0 0 3 3 1 4 2 4 0 1 1 0 2 1 4 3 3 0 2 2 0
2
 3 1 2 0 3 2 3 0 0 4 3 1 4 3 0 2 0 0 2 0 4 0 4 1 0 0 3 3 3 4 1 3 3 0 0 1
2
 3 2 0 1 0 1 1 2 2 3 0 1 4 4 0 1 1 0 0 2 1 4 0 0 4 3 0 0 0 1 2 3 3 0 1 4
1
 1 0 2 3 1 0 4 3 3 3 2 3 1 0 0 4 2 0 0 0 1 3 3 0 2 3 3 0 1 2 4 3 2 4 3 4
1
 1 0 0 1 1 4 0 0 2 2 1 0 3 0 0 4 0 2 2 0 0 4 0 1 0 0 4 0 2 0 0 1 2 3 0 0
1
 0 0 0 0 0 4 3 3 0 0 0 2 0 0 1 3 3 1 1 3 1 4 0 4 1 0 0 2 2 2 3 2 2 4 0 0
4
 3 1 0 1 4 0 0 0 2 3 1 0 2 1 2 0 3 1 2 3 2 2 0 1 2 3 4 4 0 0 0 3 1 1 1 0
3
 1 2 0 3 2 0 2 0 1 0 4 2 2 1 2 1 0 3 3 0 0 1 0 0 0 0 1 1 2 4 1 0 0 1 0 3
0
 0 2 0 2 0 4 0 0 1 3 0 0 2 0 2 4 0 0 0 3 0 3 0 3 1 1 4 4 0 0 0 3 0 3 1 0
3
 1 3 3 3 1 0 2 0 0 1 0 2 1 0 0 0 0 2 3 0 3 3 2 2]
```

a) Number of Clusters vs Distance Threshold

We chose to set `n_clusters=5` instead of using `distance_threshold` because:

1. The dataset has known labels for evaluation.
2. It ensures our clustering output can be directly compared to the known cancer types, facilitating evaluation of the model’s performance.

b) Guess Which Metric?

A distance-based metric such as Euclidean distance is appropriate for this dataset because it measures the straight-line distance between data points in the feature space, which aligns well with how RNA sequence data (gene expression levels) are typically analyzed.

c) Model Building

We used `AgglomerativeClustering` with `n_clusters=5`, `affinity='euclidean'`, and `linkage='ward'`. The clustering result labels were printed to the console.

Feel free to run the provided code to fit the model and observe the clustering results. You can then compare these labels to the actual cancer type labels to evaluate the performance of the clustering algorithm.

d) Label permuation

In clustering, the labels get assigned randomly, so the label numbering won't match the ground truth necessarily. Write a function below to find best matching label ordering based on the accuracy. Do not change the variable names. Answer the question about this section in the Peer Review.

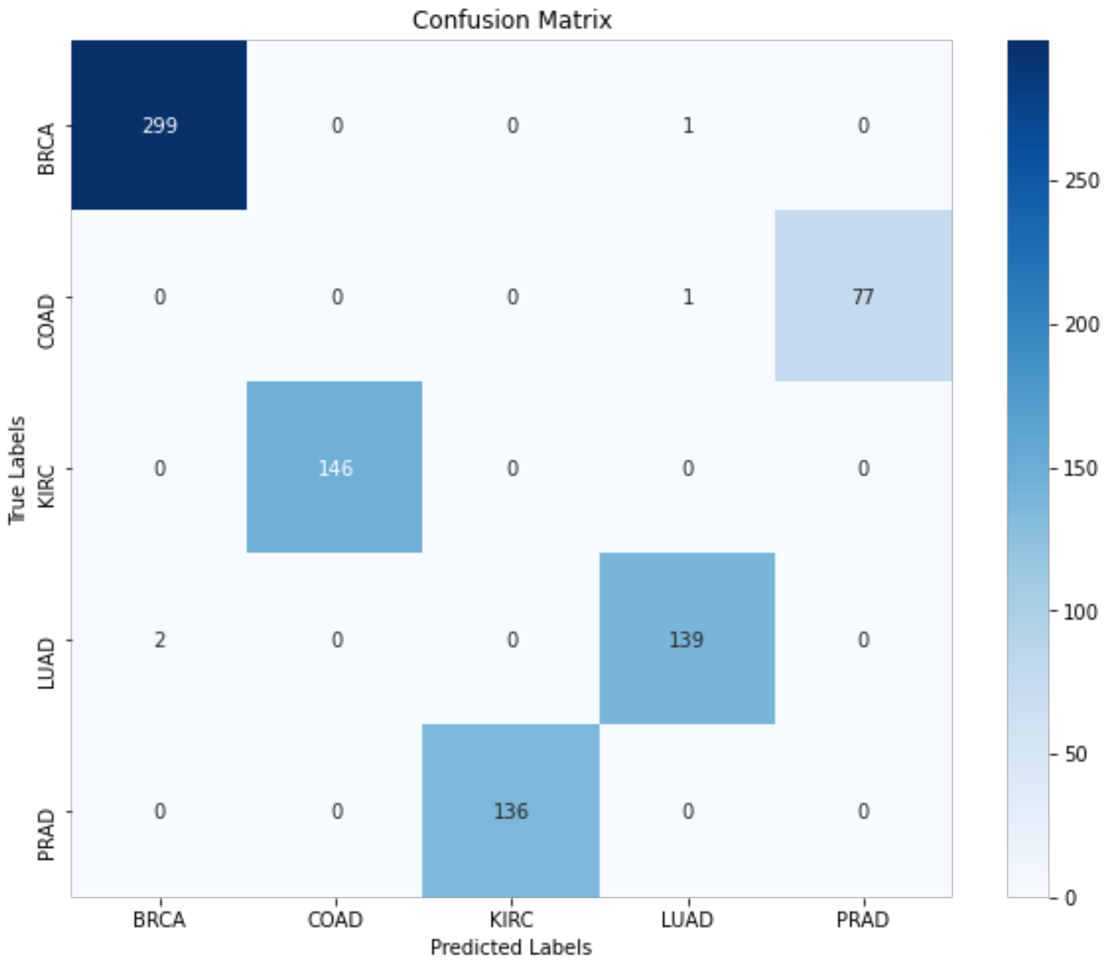
In [6]:
```
import itertools
from sklearn.metrics import accuracy_score

def label_permute_compare(ytdf, yp, n=5):
    """
    ytdf: labels dataframe object
    yp: clustering label prediction output
    Returns permuted label order and accuracy.
    Example output: (3, 4, 1, 2, 0), 0.74
    """
    ### BEGIN SOLUTION
    perms = list(itertools.permutations(list(range(n))))
    acc=[]
    for i in range(len(perms)):
        mapdict = dict(zip(list(label['Class'].unique()),list(perms[i])))
        yt = ytdf['Class'].apply(lambda x: mapdict[x])
        acc.append(accuracy_score(yt,yp))
    idx = np.argmax(acc)
    return perms[idx], acc[idx]
```

```
In [7]: labelorder, acc = label_permute_compare(label, model.labels_)
        print(labelorder, acc)

        (2, 3, 0, 1, 4) 0.9950062421972534
```

e) Check confusion matrix

Use sklearn's confusion matrix and display the results. Answer the Peer Review question about this section.

In [8]:
```
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Encode the true labels (e.g., "BRCA", "LUAD", etc.) into numeric form
label_encoder = LabelEncoder()
encoded_labels = label_encoder.fit_transform(label)

# You should now have both true and predicted labels in numeric form
# Calculate the confusion matrix
conf_matrix = confusion_matrix(encoded_labels, model.labels_)

# Plot the confusion matrix using a heatmap
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_, yticklabels=label_encode
r.classes_)
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
plt.title('Confusion Matrix')
plt.show()
```



f) Change linkage method and distance metric. Which ones lead the best performance? Print out the accuracy and confusion matrix for the best model.

Answer the Peer Review questions about this section.

In [10]:
```
# programmatically evaluate which linkage method and distance metric lead
to the best performance
# your code here
from sklearn.metrics import accuracy_score, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

for linkage in ['ward', 'complete', 'average', 'single']:
    for affinity in ['euclidean', 'l1', 'l2', 'manhattan', 'cosine']:
        acc=0
        t0=time.time()
        try:
            model = AgglomerativeClustering(n_clusters=5,linkage=linkag
e,affinity=affinity).fit(data)
            labelorder, acc = label_permute_compare(label,model.labels_)
            t1=time.time()
            print(t1-t0, linkage, affinity, labelorder, acc)
        except:
            print(linkage, 'with', affinity, 'not allowed.')

# Final answer
model = AgglomerativeClustering(n_clusters=5,linkage='ward',affinity='euc
lidean').fit(data)
labelorder, acc = label_permute_compare(label,model.labels_)
mapdict = dict(zip(list(label['Class'].unique()),list(labelorder)))
yt = label['Class'].apply(lambda x: mapdict[x])
print(acc)
confusion_matrix(yt,model.labels_)
```

```
6.280958652496338 ward euclidean (2, 3, 0, 1, 4) 0.9950062421972534
ward with l1 not allowed.
ward with l2 not allowed.
ward with manhattan not allowed.
ward with cosine not allowed.
6.342506408691406 complete euclidean (4, 3, 1, 0, 2) 0.9313358302122348
6.2622106075286865 complete l1 (4, 3, 0, 1, 2) 0.7228464419475655
6.258001804351807 complete l2 (4, 3, 1, 0, 2) 0.9313358302122348
6.274539947509766 complete manhattan (4, 3, 0, 1, 2) 0.7228464419475655
6.163297414779663 complete cosine (3, 4, 1, 2, 0) 0.7403245942571786
6.275165319442749 average euclidean (2, 4, 0, 3, 1) 0.3645443196004994
6.252565383911133 average l1 (1, 2, 0, 3, 4) 0.365792759051186
6.261708498001099 average l2 (2, 4, 0, 3, 1) 0.3645443196004994
6.322315216064453 average manhattan (1, 2, 0, 3, 4) 0.365792759051186
6.074298620223999 average cosine (2, 4, 1, 3, 0) 0.3645443196004994
6.288830757141113 single euclidean (1, 2, 0, 3, 4) 0.3757802746566791
6.281944751739502 single l1 (3, 1, 0, 2, 4) 0.37453183520599254
6.203417539596558 single l2 (1, 2, 0, 3, 4) 0.3757802746566791
6.254301071166992 single manhattan (3, 1, 0, 2, 4) 0.37453183520599254
6.10076117515564 single cosine (1, 2, 0, 3, 4) 0.3757802746566791
0.9950062421972534
```

Out[10]:
```
array([[299,   0,   0,   1,   0],
       [  0, 146,   0,   0,   0],
       [  0,   0, 136,   0,   0],
       [  2,   0,   0, 139,   0],
       [  0,   0,   0,   1,  77]])
```

best performance: ward euclidean (2, 3, 0, 1, 4)

C. What about k-means clustering?

Can we apply kmeans clustering on this data? Which clustering methods give a better performance? Is kmeans faster or slower?

In [11]:
```
# try to apply kmeans clustering on this data
# time kmeans to compare to hierarchical clustering
# your code here
from sklearn.cluster import KMeans
import time


t0=time.time()
kmeans = KMeans(5).fit(data)
t1=time.time()
print(t1-t0)
labelorder, acc = label_permute_compare(label,kmeans.labels_)
print(labelorder, acc)
mapdict = dict(zip(list(label['Class'].unique()),list(labelorder)))
yt = label['Class'].apply(lambda x: mapdict[x])
print(acc)
confusion_matrix(yt,kmeans.labels_)
```

```
9.98323106765747
(1, 0, 4, 3, 2) 0.9925093632958801
0.9925093632958801
```

Out[11]:
```
array([[139,   0,   0,   0,   2],
       [  0, 136,   0,   0,   0],
       [  2,   0,  76,   0,   0],
       [  0,   0,   0, 145,   1],
       [  1,   0,   0,   0, 299]])
```

1. Can We Apply K-Means to This Data?

Yes, k-means is applicable because:

- **K-Means** works well for numerical data like gene expression levels.
- The method is scalable to large datasets, unlike hierarchical clustering, which becomes computationally expensive for large sample sizes.

2. K-Means Clustering vs. Hierarchical Clustering

Let’s compare the two methods:

K-Means Clustering:

- **Algorithm:** K-means aims to partition data into `k` clusters by minimizing the within-cluster variance (like Ward’s method).
- **Performance:** K-means can work well if the clusters are spherical and balanced. However, gene expression data might not always exhibit clear, spherical clusters, which can affect k-means performance.
- **Speed:** K-means is generally faster than hierarchical clustering, especially for large datasets. It scales linearly with the number of samples and features, making it much more efficient for high-dimensional data like this (20,531 features).

Hierarchical Clustering:

- **Algorithm:** Hierarchical clustering builds a tree (dendrogram) of clusters, either by successively merging or splitting clusters. The approach is flexible and doesn’t require specifying the number of clusters initially.
- **Performance:** Hierarchical clustering, especially with Ward’s linkage, tends to perform better for complex datasets like gene expression data where clusters aren’t clearly spherical. As seen in your previous results, Ward’s method combined with Euclidean distance achieved near-perfect accuracy (0.9950).
- **Speed:** Hierarchical clustering is slower, especially for high-dimensional data. The time complexity is at least quadratic (`O(n^2)`) with respect to the number of samples, making it less scalable than k-means.

3. Comparing Performance and Speed:

Given your data, hierarchical clustering (Ward’s method with Euclidean distance) outperformed k-means in accuracy, achieving almost perfect clustering. However, let’s discuss speed and scenarios where k-means might be preferred:

- **K-Means Speed:** K-means is generally faster due to its linear time complexity (`O(n * k * d)`), where `n` is the number of samples, `k` is the number of clusters, and `d` is the number of features. This makes it more efficient for high-dimensional data like yours (20,531 features).
- **Hierarchical Clustering Speed:** The hierarchical approach, while more flexible and potentially more accurate, is significantly slower. With large numbers of samples and features, the time complexity (`O(n^2 log n)`) can become a bottleneck.

Conclusion:

- **Performance:** Hierarchical clustering with Ward’s method and Euclidean distance gives better performance for your data, achieving near-perfect accuracy (0.9950). This is likely because it effectively captures the complex relationships in gene expression data.
- **Speed:** K-means is faster and more scalable, making it a good choice if computational efficiency is a priority. However, it might not match the accuracy of hierarchical clustering in complex datasets like this one.
- **When to Use K-Means:** If you need faster clustering and can tolerate a slight drop in accuracy, k-means is suitable. It’s particularly useful for exploratory analysis or when handling very large datasets.

Overall, if accuracy is your main goal for this RNA sequence data, **hierarchical clustering** (Ward’s method with Euclidean distance) remains the better option. However, **k-means** provides a faster alternative with reasonable performance.

```
In [ ]:
```