

PCA on Normalized and Raw data, Tissue and Project labeling

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

normalized_exp=pd.read_csv(r"expression_data/Leaf_Root_normalized_data.csv",index_col='Unnamed: 0')
raw_exp= pd.read_csv(r"expression_data/Leaf_Root_raw_data.csv",index_col='Unnamed: 0').T
annot= pd.read_csv(r"expression_data/Leaf_Root_annotation.csv")
normalized_exp.loc[:, 'true_cluster']=np.array(annot.loc[annot['Sample']==normalized_exp.index, 'Tissue'])
raw_exp.loc[:, 'true_cluster']=np.array(annot.loc[annot['Sample']==raw_exp.index, 'Tissue'])
normalized_exp['true_cluster']=normalized_exp['true_cluster'].apply(lambda x:0 if x=="Root" else 1)
raw_exp['true_cluster']=normalized_exp['true_cluster'].apply(lambda x:0 if x=="Root" else 1)

normalized_exp.head()
```

| | rna-gnl WGS:JAHRHJ evm2.model.Chr01.1 | rna-gnl WGS:JAHRHJ evm2.model.Chr01.2 | rna-gnl WGS:JAHRHJ evm2.model.Chr01.3 |
|-------------|---------------------------------------|---------------------------------------|---------------------------------------|
| SRR10913924 | -0.271727 | 1.500167 | -0.271727 |
| SRR10913925 | -0.271727 | 1.267299 | -0.271727 |
| SRR10913926 | -0.271727 | 1.405773 | -0.271727 |
| SRR10913927 | -0.271727 | 1.369847 | -0.271727 |
| SRR10913928 | -0.271727 | 1.607823 | -0.271727 |

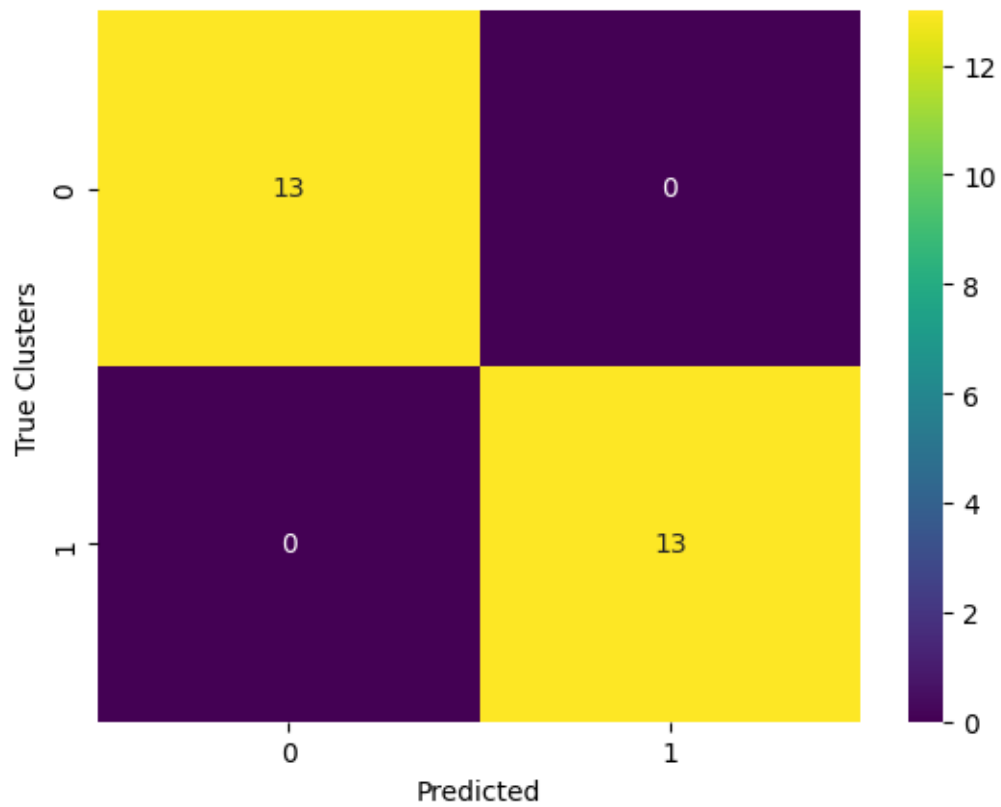
Normalized Data

Normalized: Tissue Label

```
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix, classification_report
tissue_label= pd.factorize(annot['Tissue'])[0]
project_label=pd.factorize(annot['Project'])[0]
k=2
kmeans=KMeans(n_clusters=k,n_init='auto')
kmeans.fit(normalized_exp.iloc[:,0:44770])
#column=f"kmeans_cluster_k{k}"
#normalized_exp[column]=kmeans.labels_
predicted=kmeans.labels_
confusion_mat=confusion_matrix(tissue_label, kmeans.labels_)

hp=sns.heatmap(confusion_mat,annot=True,cmap="viridis")
hp.set(ylabel='True Clusters',xlabel='Predicted')
```

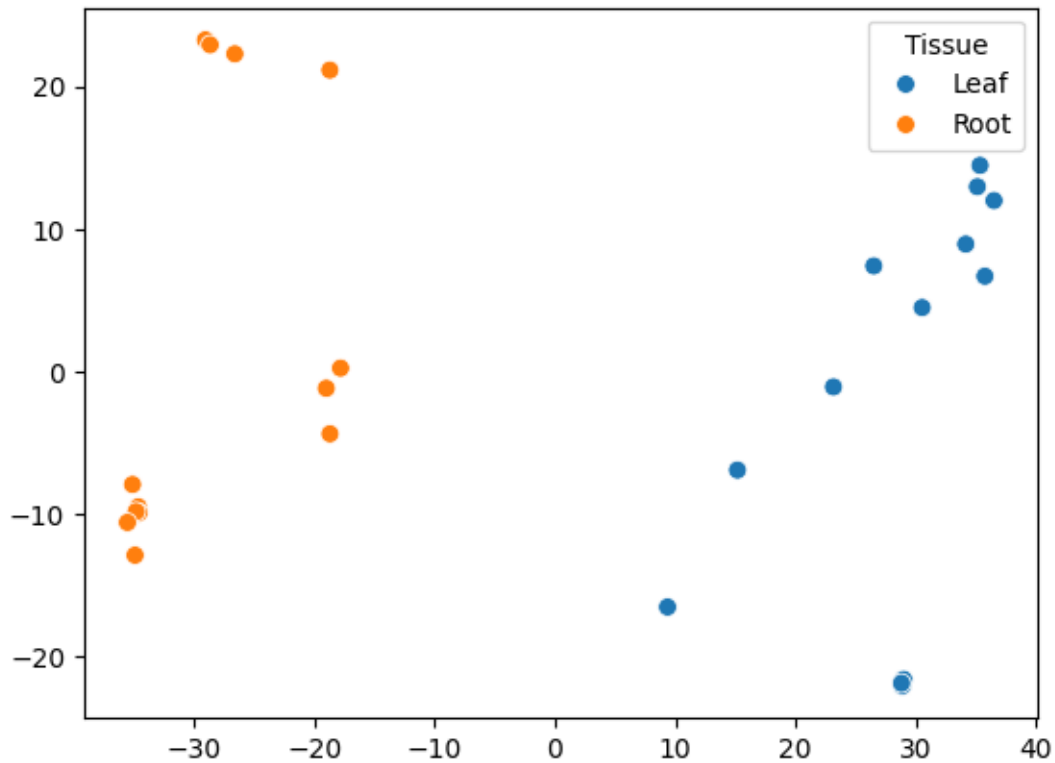
```
[Text(50.72222222222214, 0.5, 'True Clusters'),
Text(0.5, 23.52222222222222, 'Predicted')]
```



```
from sklearn.decomposition import PCA
pca = PCA()
transformed_data=pca.fit_transform(normalized_exp.iloc[:,0:44770])

sns.scatterplot(x=transformed_data[:,0],y=transformed_data[:,1],s=50,hue=annot['Tissue'])
```

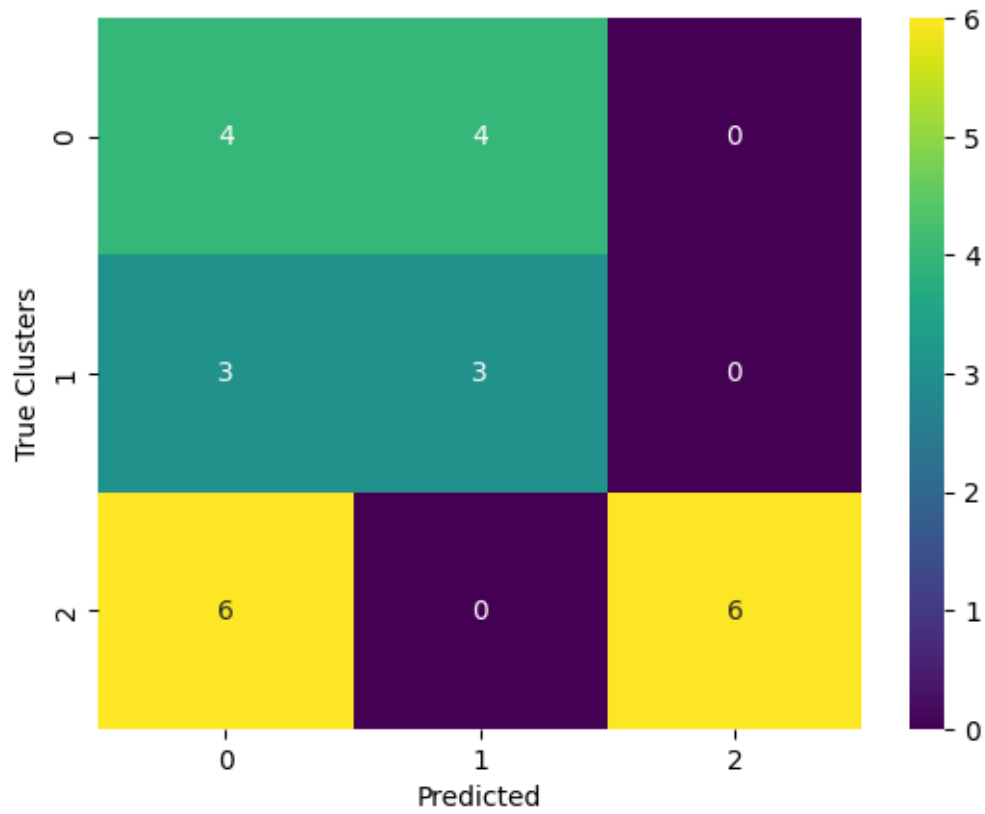
<Axes: >



Normalized: Project Label

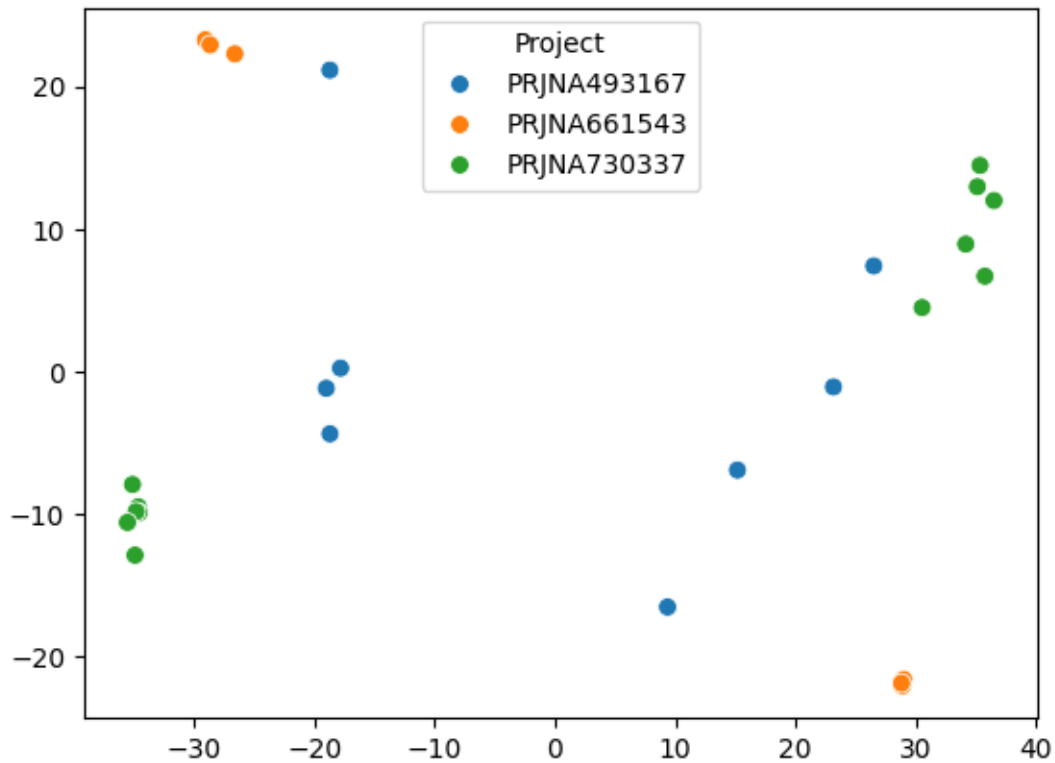
```
k=3
kmeans=KMeans(n_clusters=k,n_init='auto')
kmeans.fit(normalized_exp.iloc[:,0:44770])
#column=f"kmeans_cluster_k{k}"
predicted=kmeans.labels_
confusion_mat=confusion_matrix(project_label, predicted)
hp=sns.heatmap(confusion_mat,annot=True,cmap="viridis")
hp.set(ylabel='True Clusters',xlabel='Predicted')
```

```
[Text(50.72222222222214, 0.5, 'True Clusters'),
Text(0.5, 23.5222222222222, 'Predicted')]
```



```
sns.scatterplot(x=transformed_data[:,0],y=transformed_data[:,1],s=50,hue=annot['Project'])
```

<Axes: >



Raw Data

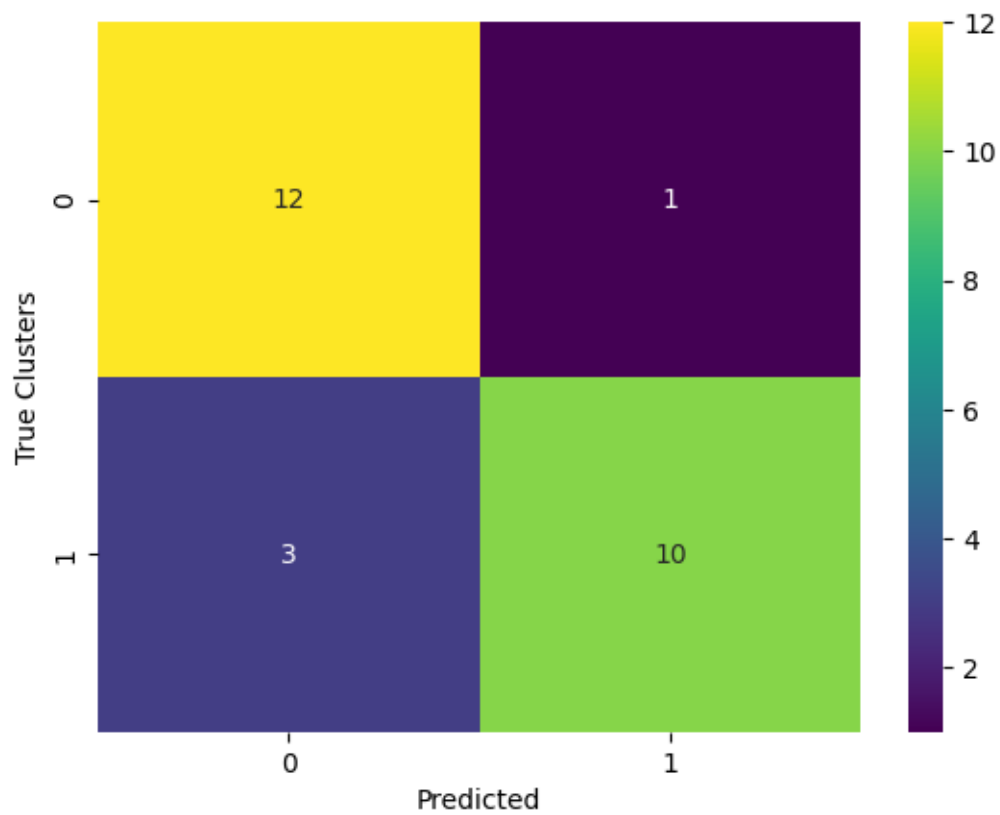
Raw: Tissue Label

```
from sklearn.cluster import KMeans
from sklearn.metrics import confusion_matrix, classification_report
tissue_label= pd.factorize(annot['Tissue'])[0]
project_label=pd.factorize(annot['Project'])[0]
k=2
kmeans=KMeans(n_clusters=k,n_init='auto')
kmeans.fit(raw_exp.iloc[:,0:44770])
#column=f"kmeans_cluster_k{k}"
#normalized_exp[column]=kmeans.labels_
predicted=kmeans.labels_
confusion_mat=confusion_matrix(tissue_label, kmeans.labels_)

hp=sns.heatmap(confusion_mat,annot=True,cmap="viridis")
```

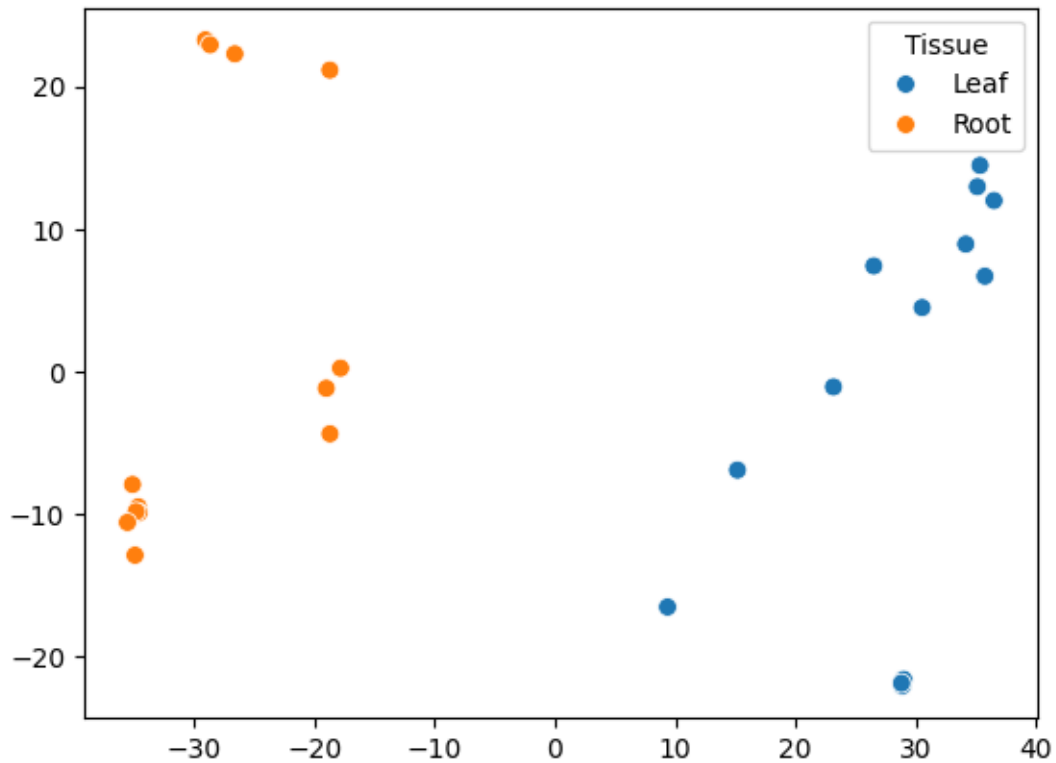
```
hp.set(ylabel='True Clusters',xlabel='Predicted')
```

```
[Text(50.72222222222214, 0.5, 'True Clusters'),  
Text(0.5, 23.5222222222222, 'Predicted')]
```



```
sns.scatterplot(x=transformed_data[:,0],y=transformed_data[:,1],s=50,hue=annot['Tissue'])
```

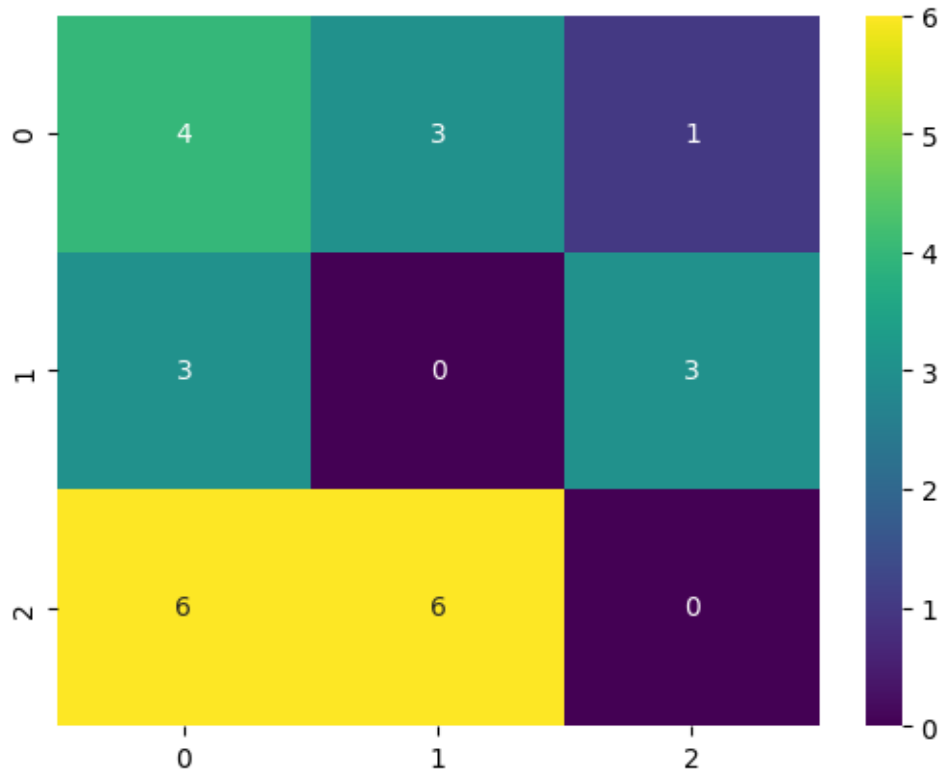
<Axes: >



Raw: Project Label

```
k=3
kmeans=KMeans(n_clusters=k,n_init='auto')
kmeans.fit(raw_exp.iloc[:,0:44770])
#column=f"kmeans_cluster_k{k}"
predicted=kmeans.labels_
confusion_mat=confusion_matrix(project_label, predicted)
sns.heatmap(confusion_mat,annot=True,cmap="viridis")
```

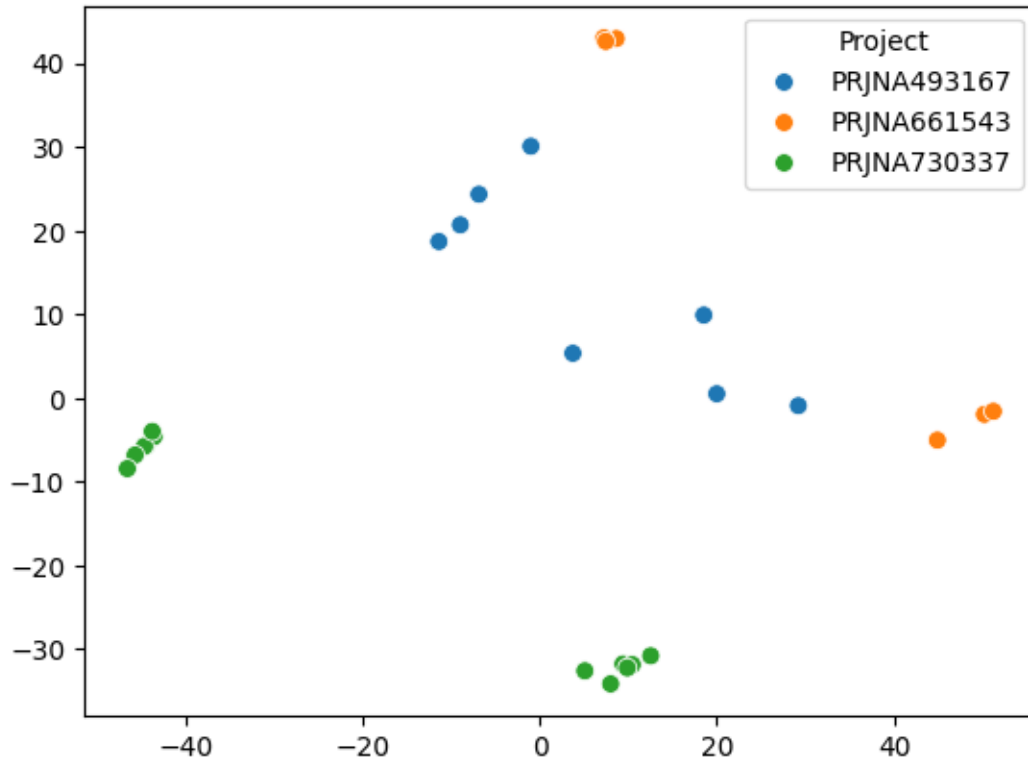
<Axes: >



```
transformed_data=pca.fit_transform(raw_exp.iloc[:,0:44770])
```

```
sns.scatterplot(x=transformed_data[:,0],y=transformed_data[:,1],s=50,hue=annot['Project'])
```

<Axes: >



References

Websites

ChatGPT

1- I want to apply Logistic regression on image data. (MNIST dataset from keras) . Do I have to flatten the features before hand? and convert 28*28 matrices to vectors? Yes, you're correct. When applying Logistic Regression to image data, such as the MNIST dataset, you need to flatten the 2D image matrices into 1D vectors¹².

The reason for this is that Logistic Regression, like many other machine learning algorithms, expects input data in a tabular form where each row represents an instance and each column represents a feature¹². In the case of image data, each pixel can be considered as a feature. So, a 28x28 pixel image from the MNIST dataset would be flattened into a 1D vector of 784 features (since $28 \times 28 = 784$).

Here's a simple example of how you might do this in Python:

Python

```
from keras.datasets import mnist import numpy as np
```

Load the MNIST dataset

```
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

Flatten the images

```
train_images = train_images.reshape((60000, 28 * 28)) test_images = test_images.reshape((10000, 28 * 28))
```

Convert to float and normalize

```
train_images = train_images.astype('float32') / 255 test_images = test_images.astype('float32') / 255
```

AI-generated code. Review and use carefully. More info on [FAQ](#). In this code, we first load the MNIST dataset using Keras. Then, we reshape the images from 28x28 matrices into 1D vectors of length 784 using the reshape function. Finally, we normalize the pixel values to be between 0 and 1 by converting the data type to float32 and dividing by 255 (since pixel values range from 0 to 255).

Once you have your flattened features, you can then apply Logistic Regression or any other machine learning algorithm that expects input data in this format¹².]

Questions:

Don't know how to interpret the results