Case Study: Brain Cancer Prediction

by: Shaiva Muthaiyah ### ID: AA.SC.P2MCA2301014

Objective

The primary objective of this case study is to delve into the intricate relationship between various genes, their expression levels, and their role in the development of cancer. To provide a clear and accessible overview of the data, we present a simple example that elucidates the meaning of these values and their corresponding functions.

Examples of Genes and their functions:

Consider the following table, which offers a concise insight into the nature of the data and the functions associated with the provided values:

Data table

ID	Sequence Source	Target Description	Gene Title	Gene Ontology Molecular Function
1007_s_at	Affymetrix Proprietary Database	mRNA /DEFINITION=HSU48705 Human receptor tyrosine kinase DDR gene, complete cds	discoidin domain receptor tyrosine kinase 1 /// microRNA 4640	0000166 // nucleotide binding // inferred from electronic annotation /// 0004672 // protein kinase activity // inferred from electronic annotation /// 0004713 // protein tyrosine kinase activity // inferred from electronic annotation /// 0004714 // transmembrane receptor protein tyrosine kinase activity // traceable author statement /// 0005515 // protein binding // inferred from physical interaction /// 0005518 // collagen binding // inferred from direct assay /// 0005518 // collagen binding // inferred from mutant phenotype /// 0005524 // ATP binding // inferred from electronic annotation /// 0016301 // kinase activity // inferred from electronic annotation /// 0016740 // transferase activity // inferred from electronic annotation /// 0016772 // transferase activity, transferring phosphorus-containing groups // inferred from electronic annotation /// 0038062 // protein tyrosine kinase collagen receptor activity // inferred from direct assay /// 0046872 // metal ion binding // inferred from electronic annotation
1053_at	GenBank	M87338 /FEATURE= /DEFINITION=HUMA1SBU Human replication factor C, 40-kDa subunit (A1) mRNA, complete cds	replication factor C (activator 1) 2, 40kDa	0000166 // nucleotide binding // inferred from electronic annotation /// 0003677 // DNA binding // inferred from electronic annotation /// 0005515 // protein binding // inferred from physical interaction /// 0005524 // ATP binding // inferred from electronic annotation /// 0016851 // magnesium chelatase activity // inferred from electronic annotation /// 0017111 // nucleoside-triphosphatase activity // inferred from electronic annotation
117_at	Affymetrix Proprietary Database	HSP70B Human heat-shock protein HSP70B' gene	heat shock 70kDa protein 6 (HSP70B')	0000166 // nucleotide binding // inferred from electronic annotation /// 0005524 // ATP binding // inferred from electronic annotation /// 0019899 // enzyme binding // inferred from physical interaction /// 0031072 // heat shock protein binding // inferred from physical interaction /// 0042623 // ATPase activity, coupled // inferred from direct assay /// 0051082 // unfolded protein binding // inferred from direct assay
121_at	GenBank	HSPAX8A H.sapiens Pax8 mRNA	paired box 8	0000979 // RNA polymerase II core promoter sequence-specific DNA binding // inferred from direct assay /// 0003677 // DNA binding // inferred from direct assay /// 0003677 // DNA binding // inferred from mutant phenotype /// 0003700 // sequence-specific DNA binding transcription factor activity // inferred from direct assay /// 0004996 // thyroid-stimulating hormone receptor activity // traceable author statement /// 0005515 // protein binding // inferred from physical interaction /// 0044212 // transcription regulatory region DNA binding // inferred from direct assay

Brief description of the cancers covered:

- ### Glioblastoma:

Glioblastoma is an aggressive and fast-growing type of glioma, which arises from the glial cells in the brain. It is the most common and malignant form of glioma. Glioblastomas are known for their infiltrative nature, making complete surgical removal challenging. They tend to recur despite aggressive treatment. Prognosis for glioblastoma is often poor due to its aggressive behavior and resistance to standard treatments.

- ### Astrocytoma:

Astrocytomas are tumors that originate from astrocytes, a type of glial cell. They can range from low-grade (slow-growing) to high-grade (more aggressive) forms. Low-grade astrocytomas may progress to higher grades over time. High-grade astrocytomas, like glioblastoma, can be challenging to treat effectively. Prognosis varies based on the grade of the tumor, with higher-grade astrocytomas generally having a more guarded prognosis.

- ## Oligodendroglioma:

Oligodendrogliomas arise from oligodendrocytes, another type of glial cell. They are characterized by a more well-defined border compared to other gliomas. Oligodendrogliomas can be slow-growing and are often associated with a more favorable prognosis compared to some other gliomas. Prognosis depends on factors such as the tumor grade, location, and the effectiveness of treatment. Higher-grade oligodendrogliomas may require more aggressive management.

- ## Glioblastoma Cell Line:

Glioblastoma cell lines are cultured cells derived from glioblastoma tumors. These cell lines are commonly used in laboratory research to study the biology of glioblastomas and test potential treatments. Glioblastoma cell lines provide a controlled environment for researchers to investigate the molecular and cellular mechanisms underlying glioblastoma development and progression. Research using glioblastoma cell lines contributes to the development of novel therapies and a deeper understanding of the disease at the cellular level.

Gene Expression Level Explanation:

The numerical value associated with gene expression level typically represents the quantity or abundance of messenger RNA (mRNA) for a specific gene in a given sample. Some details are listed below:

- Continuous Scale: Values are often reported on a continuous scale. The specific unit depends on the technology used (e.g., log-transformed intensities in microarrays or normalized counts in RNA sequencing).
- Magnitude of Expression: Higher values generally represent a higher abundance of mRNA for the corresponding gene in the given sample.

Preprocessing: Loading the Dataset and peeking into the data

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from tabulate import tabulate


# Dataset from https://sbcb.inf.ufrgs.br/cumida
dataset = pd.read_csv('/content/drive/MyDrive/Brain_GSE15824.csv')

from IPython.display import display, HTML

# Set the maximum number of rows and columns to display
pd.set_option('display.max_rows', None)
```

```
pd.set_option('display.max_columns', None)

# Randomly select 15 rows to display
random_rows = dataset.sample(15)
#print(dataset.size)

print("Different cancerous samples with corresponding genes and their
expression levels".title())
print("Here the 'type' will be the target(dependent) variable and the
expression levels will be independent")
# Our objective is to understand how gene expression contribute to
different cancers and how
#  they contribute to developing specific cancers.
# Gene Expression Values indicate the activity of a gene in expressing
its associated characteristics

# Display the DataFrame
display(HTML(random_rows.to_html()))
```

```
Different Cancerous Samples With Corresponding Genes And Their
Expression Levels
Here the 'type' will be the target(dependent) variable and the
expression levels will be independent

<IPython.core.display.HTML object>
```

Preprocessing: Checking for Missing Values

```
# Check for missing values
#print(dataset.isnull().sum())

# Separate features (X) and target variable (y)
# Remove the samples and type columns as they are non numeric
X = dataset.drop(['samples', 'type'], axis=1)

# Type will be the target cancer type
y = dataset['type']
```

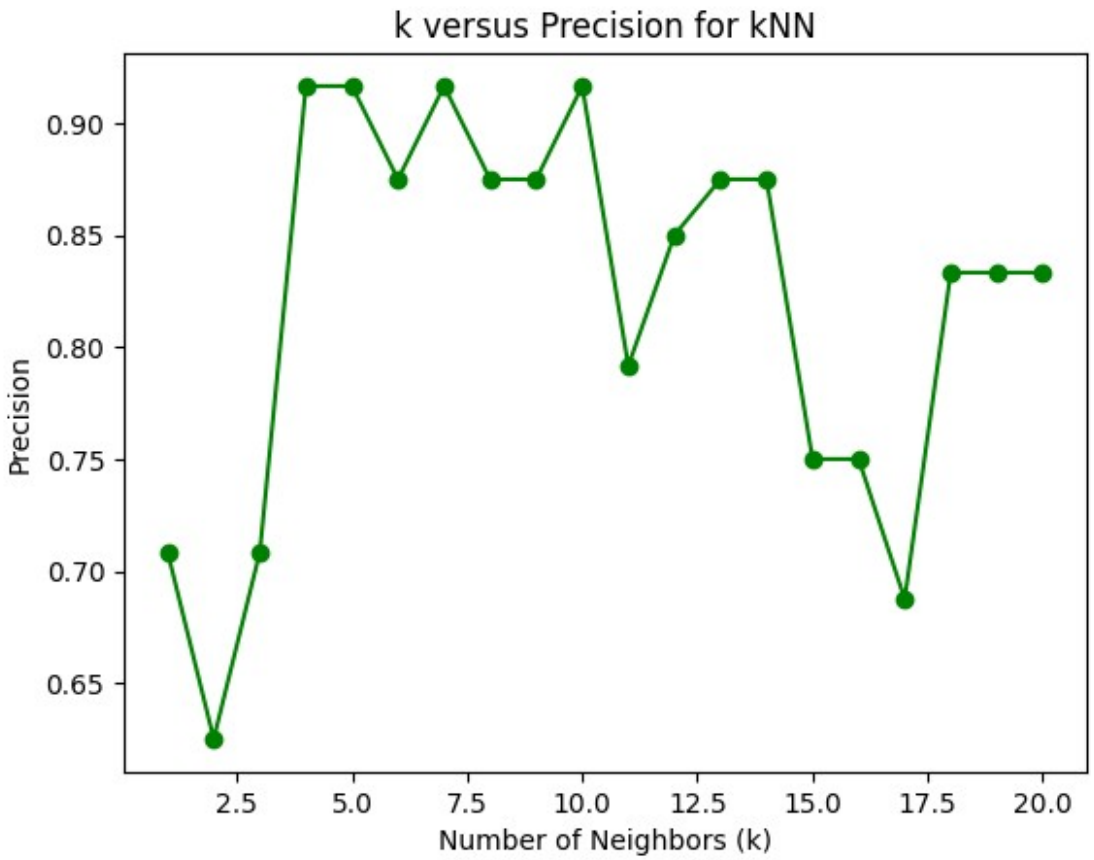
Model 1: KNN Algorithm

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

```
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score

knn_model = KNeighborsClassifier(n_neighbors=5)
knn_model.fit(X_train, y_train)
y_pred_knn = knn_model.predict(X_test)
accuracy_knn = accuracy_score(y_test, y_pred_knn)
print(f'kNN Accuracy: {accuracy_knn}')
```

```
kNN Accuracy: 0.875
```

```
from sklearn.metrics import precision_score
import matplotlib.pyplot as plt

# List to store precision values
precision_values = []

# Test different values of k
for k in range(1, 21):
    # Initialize the model with a value of k
    knn_model = KNeighborsClassifier(n_neighbors=k,
metric='euclidean')

    # Train the model with the datasets
    knn_model.fit(X_train, y_train)

    # Predict the value of y from the test dataset
    y_pred_knn = knn_model.predict(X_test)

    # Compare values of the predicted and the actual dataset
    precision = precision_score(y_test, y_pred_knn,
average='weighted', zero_division=1)

    # Add the precision values to the list
    precision_values.append(precision)

# Plotting the graph
plt.plot(range(1, 21), precision_values, marker='o', color='green')
plt.title('k versus Precision for kNN')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Precision')
plt.show()
```

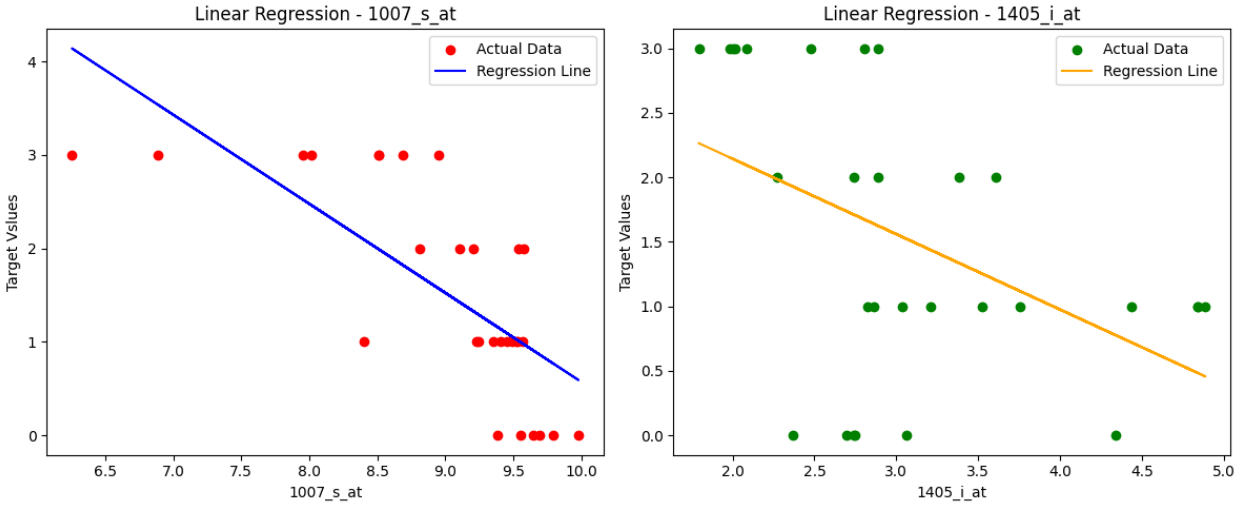


Model 2a: LINEAR REGRESSION

```
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import mean_squared_error, accuracy_score
from sklearn.preprocessing import LabelEncoder

# Encode the target variable
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)


# Since y is a pandas Series
df_mapping = pd.DataFrame({'Original': y.unique(), 'Encoded':
label_encoder.transform(y.unique())})
# Convert DataFrame to HTML using tabulate
html_table = tabulate(df_mapping, headers='keys', tablefmt='html',
showindex=False)
# Display HTML table in Google Colab
display(HTML(html_table))


# We select two independent variables to predict the target feature
```

```
X_two_features = X[['1007_s_at', '1405_i_at']]

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_two_features,
y_encoded, test_size=0.2, random_state=42)

# Create and fit the linear regression models for each feature
linear_regressor_1 = LinearRegression()
linear_regressor_2 = LinearRegression()

linear_regressor_1.fit(X_train[['1007_s_at']], y_train)
linear_regressor_2.fit(X_train[['1405_i_at']], y_train)

# Plotting the regression lines side by side
plt.figure(figsize=(12, 5))

# Plot for '1007_s_at' which is the first independent variable
plt.subplot(1, 2, 1)
plt.scatter(X_train[['1007_s_at']], y_train, color='red',
label='Actual Data')
plt.plot(X_train[['1007_s_at']],
linear_regressor_1.predict(X_train[['1007_s_at']]), color='blue',
label='Regression Line')
plt.title('Linear Regression - 1007_s_at')
plt.xlabel('1007_s_at')
plt.ylabel('Target Vslues')
plt.legend()

# Plot for '1405_i_at' which is the second independent variable
plt.subplot(1, 2, 2)
plt.scatter(X_train[['1405_i_at']], y_train, color='green',
label='Actual Data')
plt.plot(X_train[['1405_i_at']],
linear_regressor_2.predict(X_train[['1405_i_at']]), color='orange',
label='Regression Line')
plt.title('Linear Regression - 1405_i_at')
plt.xlabel('1405_i_at')
plt.ylabel('Target Values')
plt.legend()

plt.tight_layout()
plt.show()

# For this dataset linear regression is not ideal as our target values
resemble a classification problem
```

```
<IPython.core.display.HTML object>
```



Model 2b: LOGISTIC REGRESSION

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.preprocessing import LabelEncoder
import pandas as pd
from tabulate import tabulate
from IPython.display import HTML, display

# Since is a pandas DataFrame with multiple target variables
# Encode the target variable
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Since y is a pandas Series
df_mapping = pd.DataFrame({'Original': y.unique(), 'Encoded':
label_encoder.transform(y.unique())})
# Convert DataFrame to HTML using tabulate
html_table = tabulate(df_mapping, headers='keys', tablefmt='html',
showindex=False)
# Display HTML table in Google Colab
display(HTML(html_table))

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.2, random_state=42)

# Train logistic regression model
# Since this is a classification problem logistic regression will be
effective
# We deal with multiple classes so we employ multiple logistic
regression
```

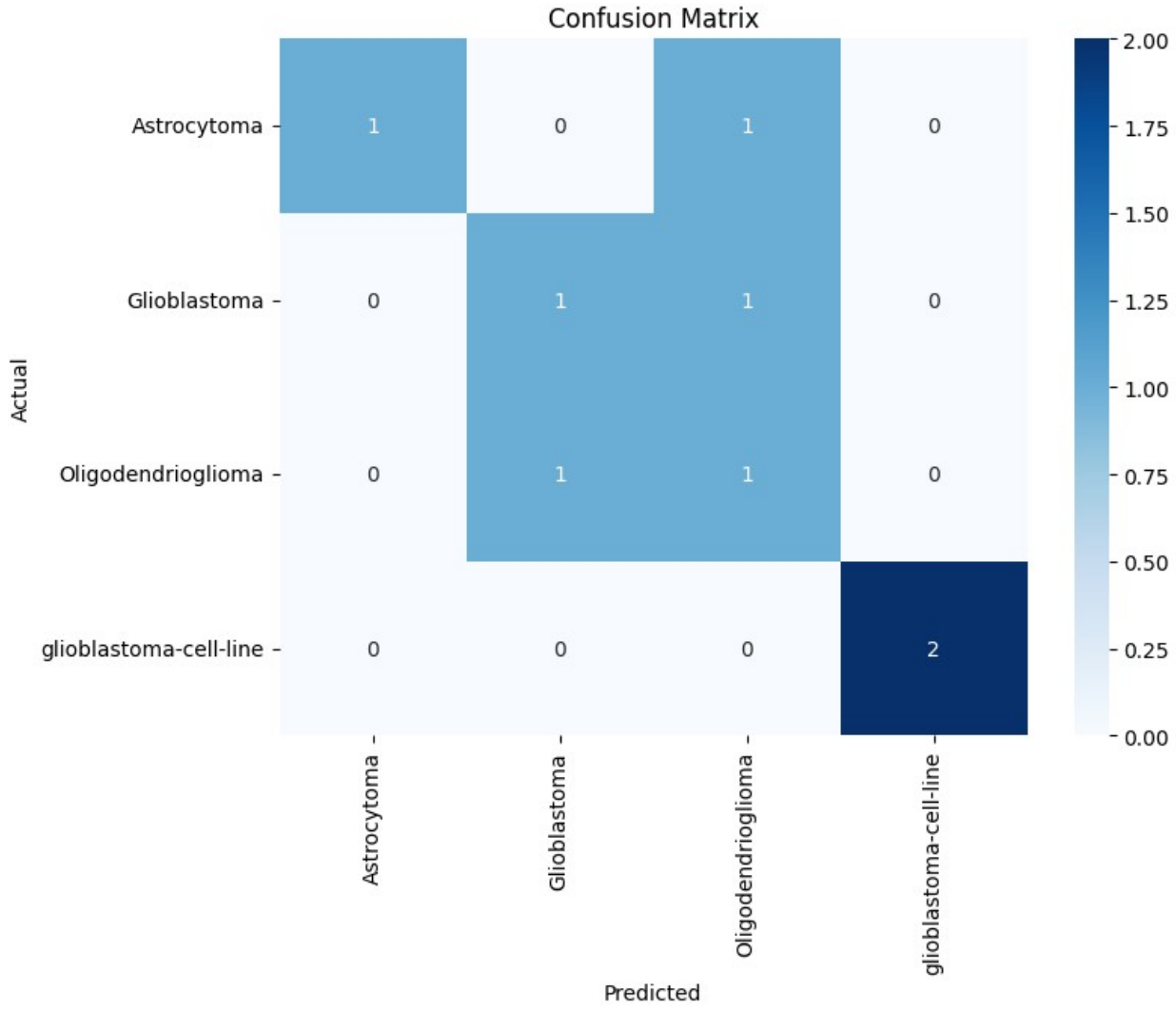
```
logistic_regressor = LogisticRegression(max_iter=500,
multi_class='ovr')
logistic_regressor.fit(X_train, y_train)

# Predictions on the test set
y_pred = logistic_regressor.predict(X_test)

# Evaluate the performance
accuracy = accuracy_score(y_test, y_pred)


# Function to generate a stylized classification report table
def generate_classification_report_table(y_true, y_pred,
target_names):
    report_dict = classification_report(y_true, y_pred,
target_names=target_names, output_dict=True)
    report_df = pd.DataFrame(report_dict).transpose()

    # Stylize the table
    styled_report = report_df.style.background_gradient(cmap='Blues',
subset=pd.IndexSlice[:, :'f1-score'])

    return styled_report

# Generate and display the classification report table
classification_report_table =
generate_classification_report_table(y_test, y_pred,
label_encoder.classes_)
classification_report_table
```

```
<IPython.core.display.HTML object>
```

```
<pandas.io.formats.style.Styler at 0x7e7523c23010>
```

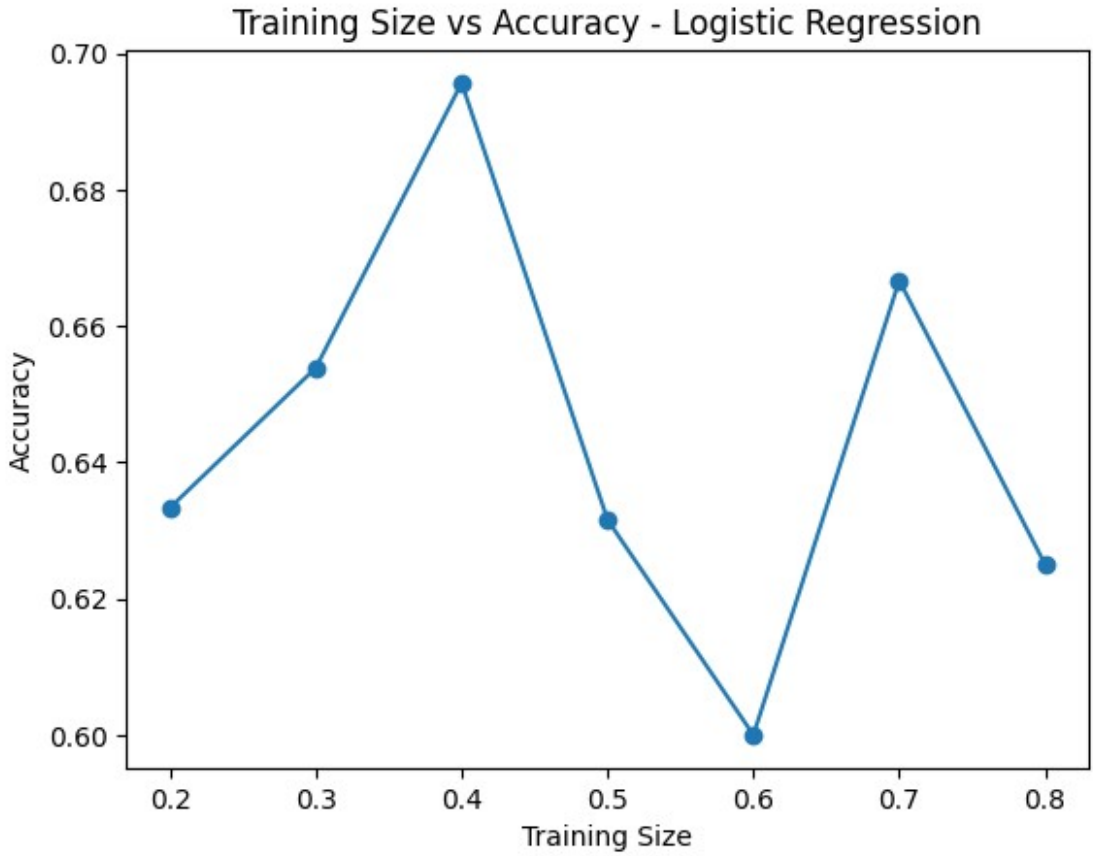
```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

# Function to plot a confusion matrix heatmap
def plot_confusion_matrix_heatmap(y_true, y_pred, classes):
    cm = confusion_matrix(y_true, y_pred)
    plt.figure(figsize=(8, 6))
    sns.heatmap(cm, annot=True, fmt="d", cmap="Blues",
xticklabels=classes, yticklabels=classes)
    plt.title('Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

# Plot confusion matrix heatmap
```

```
plot_confusion_matrix_heatmap(y_test, y_pred,
classes=label_encoder.classes_)
```

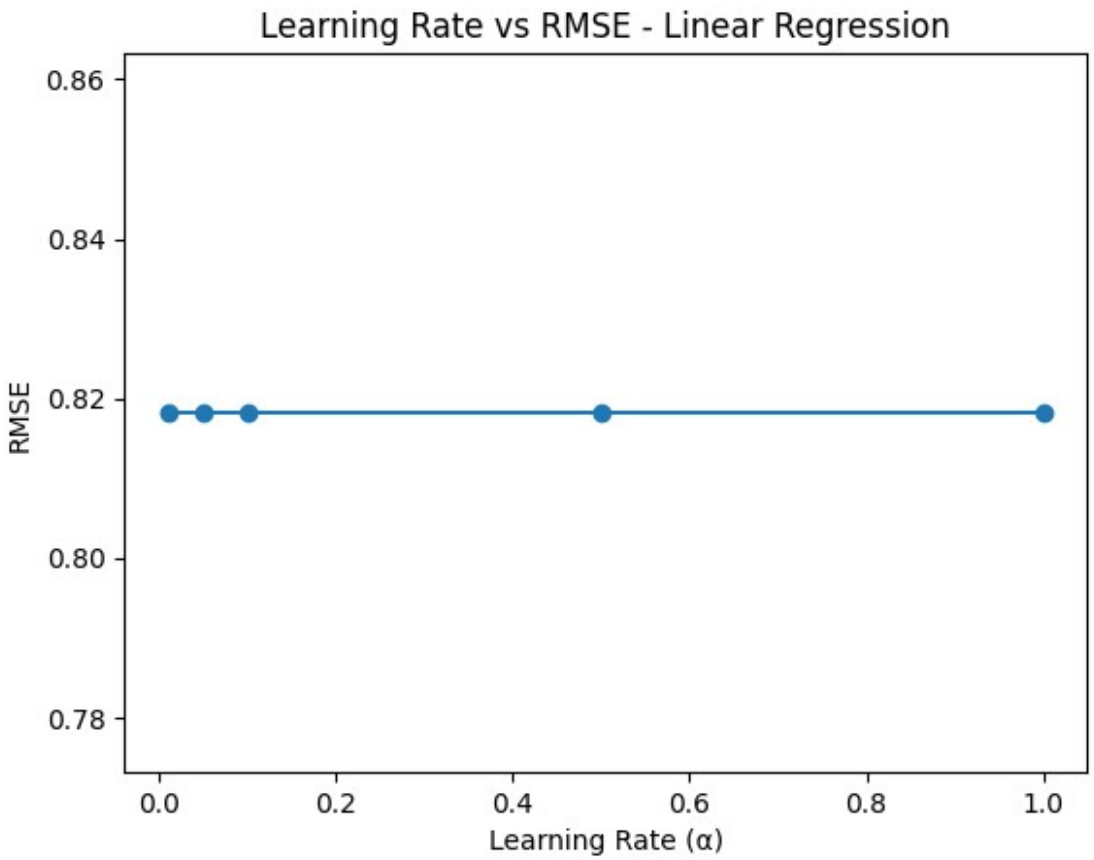



```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
import matplotlib.pyplot as plt
import numpy as np

# Assuming y is a pandas Series
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Define training sizes
training_sizes = [0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8]
accuracy_values = []
```

```
# Create a model in every loop for each training size and predict the
accuracy
for size in training_sizes:
    X_train_partial, X_test, y_train_partial, y_test =
train_test_split(X, y_encoded, train_size=size, random_state=42)
    model = LogisticRegression(max_iter=500, multi_class='ovr')
    model.fit(X_train_partial, y_train_partial)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
    accuracy_values.append(accuracy)


# Plot the graph
plt.plot(training_sizes, accuracy_values, marker='o')
plt.title('Training Size vs Accuracy - Logistic Regression')
plt.xlabel('Training Size')
plt.ylabel('Accuracy')
plt.show()
```



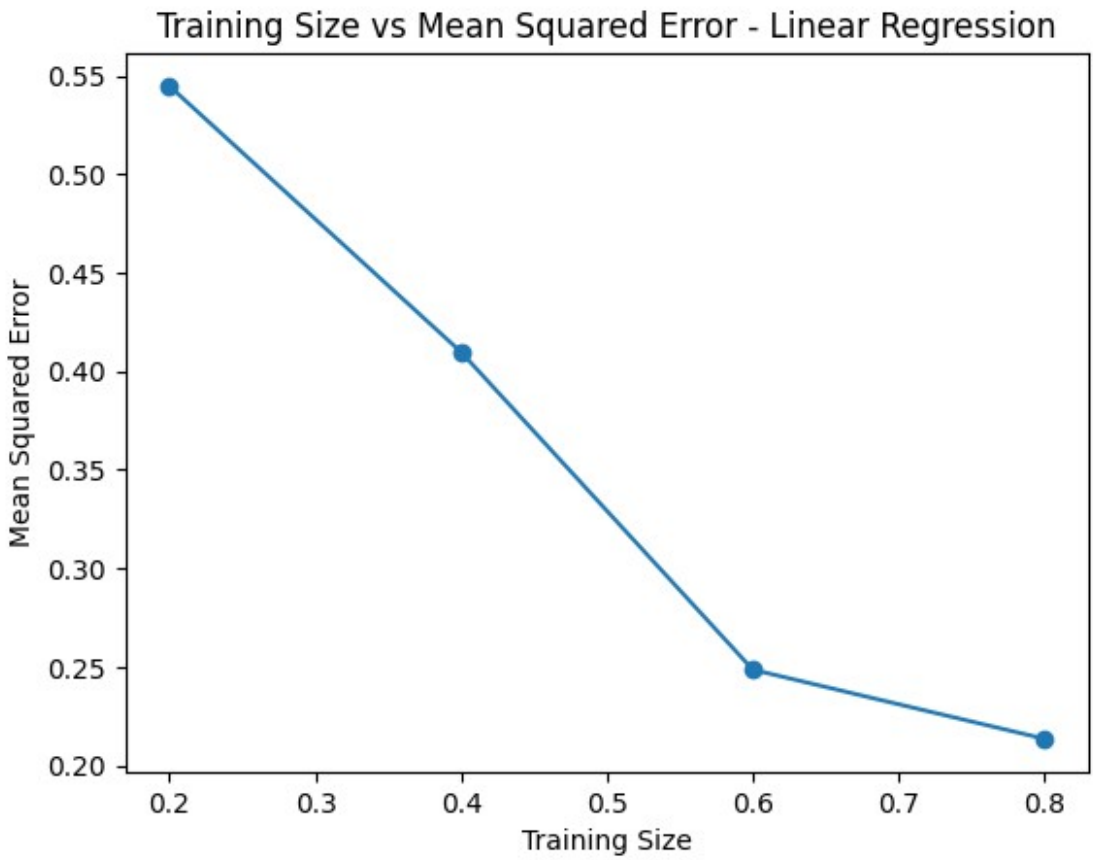
```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
import numpy as np

# Encode the target variable for linear regression
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# We select two independent variables to predict the target feature
X_two_features = X[['1007_s_at', '1405_i_at']]

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_two_features,
y_encoded, test_size=0.2, random_state=42)

# Vary learning rates (α)
learning_rates = [0.01, 0.05, 0.1, 0.5, 1.0]
rmse_values = []

# Loop for creating the rmse_value list
for alpha in learning_rates:
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    rmse_values.append(rmse)

# Plot the graph
plt.plot(learning_rates, rmse_values, marker='o')
plt.title('Learning Rate vs RMSE - Linear Regression')
plt.xlabel('Learning Rate (α)')
plt.ylabel('RMSE')
plt.show()

# Linear Regression Models are not as efficient as our dataset creates
a Classfication Problem
# Reducing the complexity by using two independent variables does not
drastically change the output
```




```
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
from IPython.display import HTML


# Encode the target variable for linear regression
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)



# Assuming y is a pandas Series
df_mapping = pd.DataFrame({'Original': y.unique(), 'Encoded':
label_encoder.transform(y.unique())})
# Convert DataFrame to HTML using tabulate
html_table = tabulate(df_mapping, headers='keys', tablefmt='html',
showindex=False)

# Display HTML table in Google Colab
display(HTML(html_table))
```

```
# Training values
training_sizes = [0.2, 0.4, 0.6, 0.8]

# collect mean squared error values in this array
mse_values = []


for size in training_sizes:
    X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
train_size=size, test_size=0.2, random_state=42)
    model = LinearRegression()
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)  # Predict on the test set
    mse = mean_squared_error(y_test, y_pred)
    mse_values.append(mse)

plt.plot(training_sizes, mse_values, marker='o')
plt.title('Training Size vs Mean Squared Error - Linear Regression')
plt.xlabel('Training Size')
plt.ylabel('Mean Squared Error')
plt.show()
```

```
<IPython.core.display.HTML object>
```



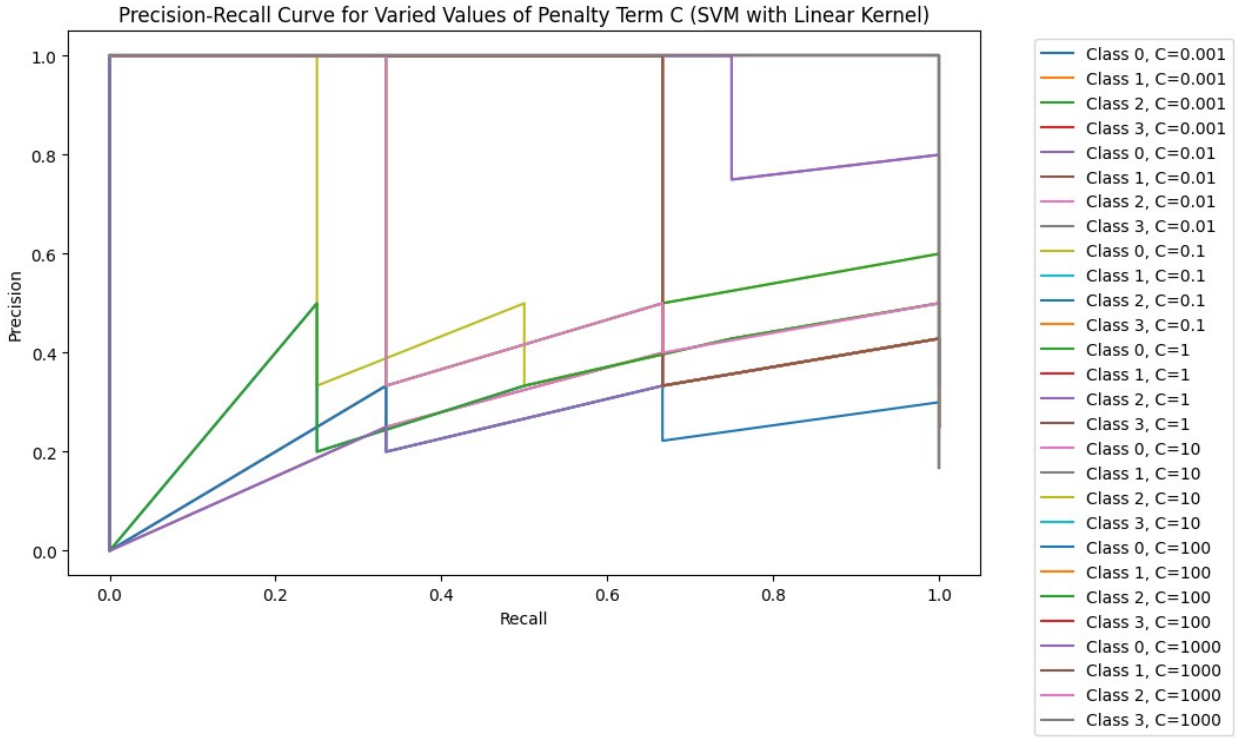
Model 3: SVM

```
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_recall_curve, auc

# Encode the target variable
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)



# Since y is a pandas Series
df_mapping = pd.DataFrame({'Original': y.unique(), 'Encoded':
label_encoder.transform(y.unique())})
# Convert DataFrame to HTML using tabulate
html_table = tabulate(df_mapping, headers='keys', tablefmt='html',
showindex=False)
# Display HTML table in Google Colab
display(HTML(html_table))
```

```
# Split the dataset into training and testing sets (70:30 split)
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.3, random_state=42)


# Varied values for penalty term C
penalty_terms = [0.001, 0.01, 0.1, 1, 10, 100, 1000]


# Initialize lists to store precision and recall values for each C and
class
precision_values = []
recall_values = []

for C in penalty_terms:
    # Create SVM model with a linear kernel and specified C
    model = SVC(kernel='linear', C=C, probability=True,
decision_function_shape='ovo')  # Use 'ovo' for one-vs-one
    model.fit(X_train, y_train)

    # Predict probabilities for each class
    y_probs = model.predict_proba(X_test)

    # Compute precision and recall for each class
    for class_index in range(model.classes_.shape[0]):
        y_true_class = (y_test == class_index).astype(int)
        y_probs_class = y_probs[:, class_index]
        precision, recall, _ = precision_recall_curve(y_true_class,
y_probs_class)
        auc_score = auc(recall, precision)

        # Append precision and recall values for the current class and
C
        precision_values.append(precision)
        recall_values.append(recall)

# Plot the graph with varied values for the penalty term C
plt.figure(figsize=(10, 6))

for i, C in enumerate(penalty_terms):
    for j in range(model.classes_.shape[0]):
        plt.plot(recall_values[i * model.classes_.shape[0] + j],
precision_values[i * model.classes_.shape[0] + j], label=f'Class {j},
C={C}')

plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve for Varied Values of Penalty Term C
(SVM with Linear Kernel)')
plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
```

```
plt.show()
```

```
<IPython.core.display.HTML object>
```



Model 4: PCA

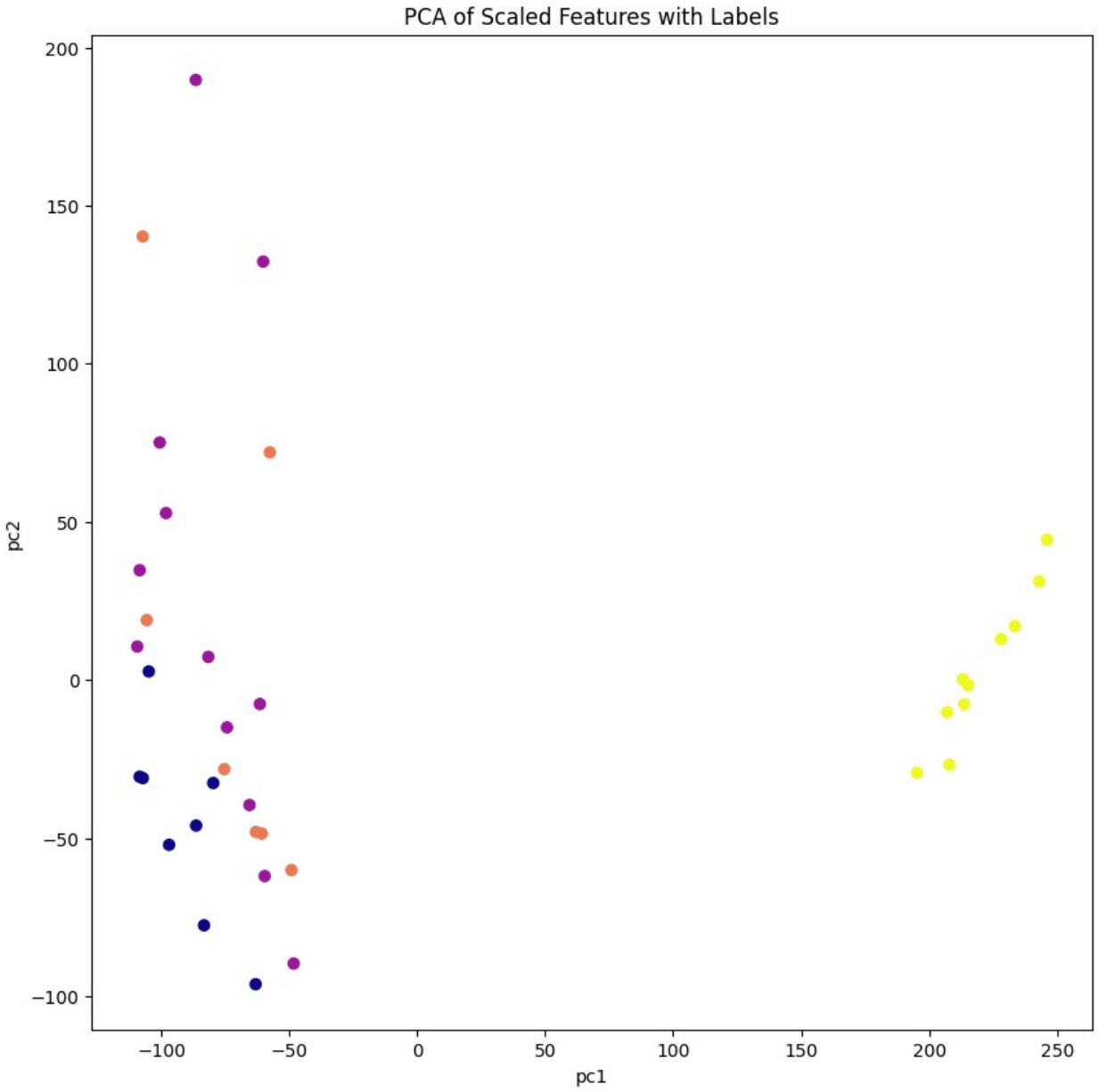
```
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler


# Scale data before applying PCA
scaling=StandardScaler()

# Use fit and transform method
scaling.fit(X)
Scaled_data=scaling.transform(X)

# Encode the target variable since it is a string
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)
print(y_encoded)

print(Scaled_data)

# Set the n_components=3
```

```
principal=PCA(n_components=3)
principal.fit(Scaled_data)
x=principal.transform(Scaled_data)

# Get the loadings (components) and feature names
# loadings = principal.components_
# feature_names = X.columns

# DataFrame to display the loadings
# loadings_df = pd.DataFrame(loadings, columns=feature_names)

# Display the loadings
# print("Loadings (Components):")
# print(loadings_df)


# Check the dimensions of data after PCA
print("Shape of the matrix:", x.shape)
```

```
[1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3
3 3]
[[ 0.51485652 -0.47532374  2.28076255 ... -0.72523253 -0.1913693
  -0.71325652]
 [ 0.67360952 -0.09899706  2.74048195 ... -1.17676418 -0.57586824
  -0.37931182]
 [-0.49665147 -1.03866539 -0.59204849 ...  1.17202793  0.44382522
   3.51464582]
 ...
 [-0.21512912  1.05889194 -0.63055049 ...  1.22063778  0.35400931
  -0.62310649]
 [-2.89636242  1.68022482 -0.34560151 ...  0.1969349   1.28506597
   1.11740952]
 [-2.66487837  1.57326542 -0.18375443 ...  0.82590862  2.05891413
   0.17847857]]
Shape of the matrix: (37, 3)
```

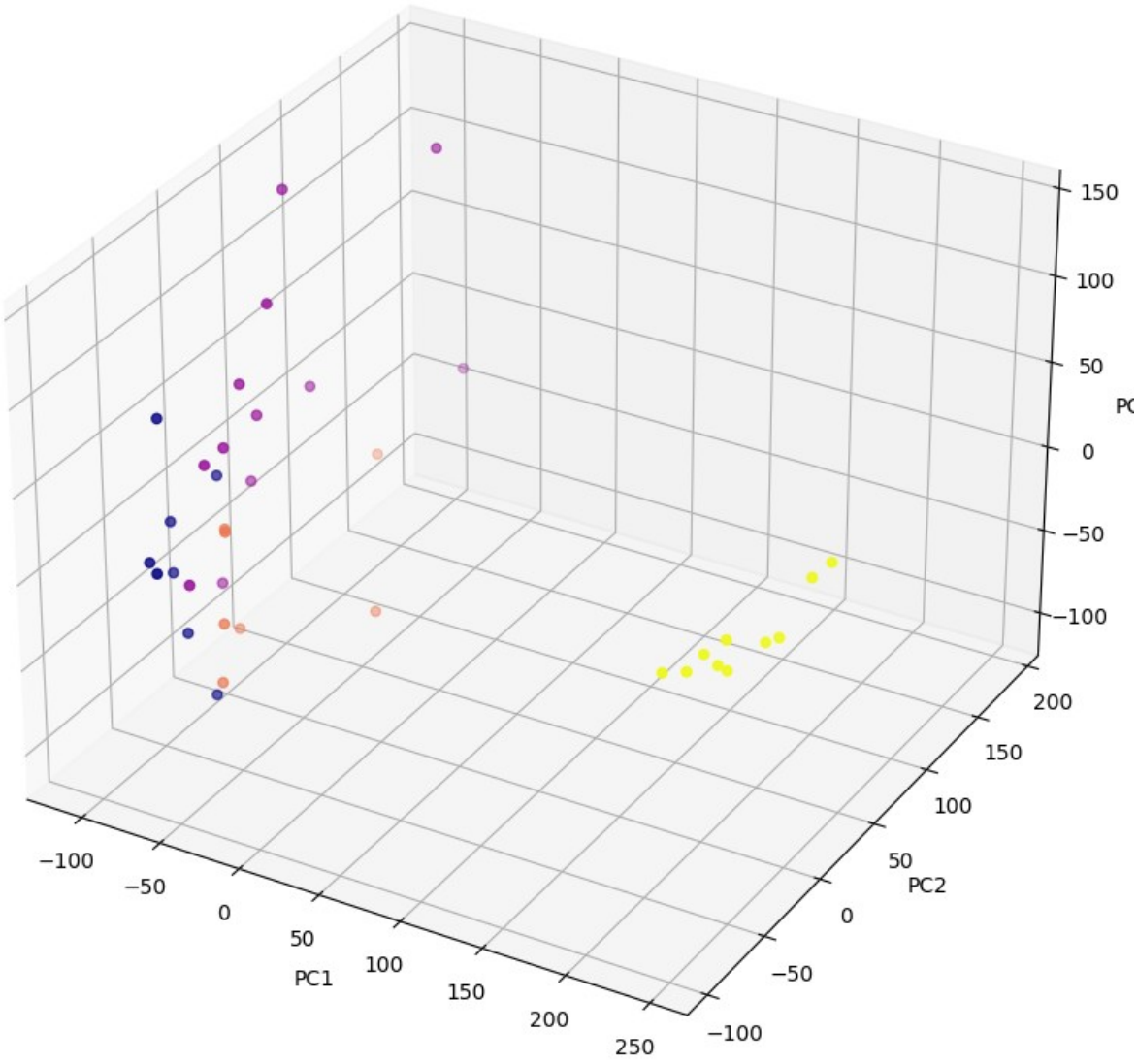
```
plt.figure(figsize=(10,10))
plt.scatter(x[:, 0], x[:, 1], c=y_encoded, cmap='plasma')
plt.xlabel('pc1')
plt.ylabel('pc2')
plt.title('PCA of Scaled Features with Labels')
```

```
Text(0.5, 1.0, 'PCA of Scaled Features with Labels')
```




```
# import libraries for 3d graph
from mpl_toolkits.mplot3d import Axes3D
fig = plt.figure(figsize=(10,10))

# choose projection 3d for creating a 3d graph
axis = fig.add_subplot(111, projection='3d')

# x[:,0]is pc1,x[:,1] is pc2 while x[:,2] is pc3
axis.scatter(x[:,0],x[:,1],x[:,2], c=y_encoded,cmap='plasma')
axis.set_xlabel("PC1", fontsize=10)
axis.set_ylabel("PC2", fontsize=10)
axis.set_zlabel("PC3", fontsize=10)
```

```
Text(0.5, 0, 'PC3')
```




```
# Access the principal components
components = principal.components_

# Access the explained variance ratio
explained_variance_ratio = principal.explained_variance_ratio_

# Print the results
print("Principal Components:")
print(components)
```

```
print("\nExplained Variance Ratio:")
print(explained_variance_ratio)
```

```
Principal Components:
[[-5.74628212e-03  6.70659850e-03 -1.42020741e-03 ...  3.96644790e-03
   6.02433907e-03  8.97256306e-05]
 [-3.40217271e-03 -3.28840670e-03  1.05571527e-03 ...  1.51028619e-03
   5.13227287e-03  6.58586432e-03]
 [ 2.64063607e-04  8.78507462e-04  1.08661081e-02 ...  3.79746701e-03
  -6.93891546e-04 -1.81370783e-03]]

Explained Variance Ratio:
[0.33568379 0.06897215 0.0631215 ]
```