Université Abdelmalek Essaadi Faculté des Sciences et Techniques –Tanger Département Génie Informatique

# Lab 1

PyTorch

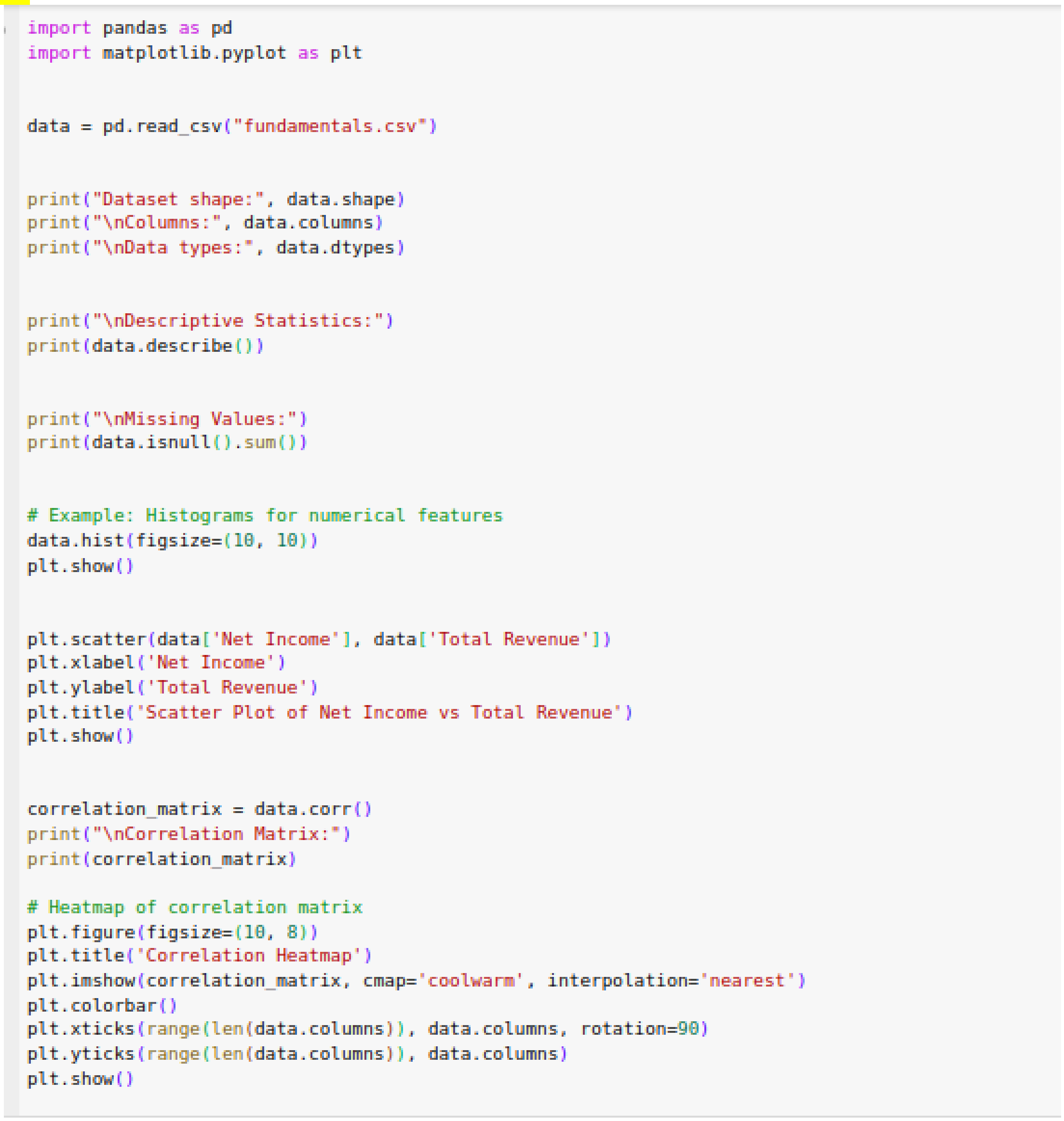
Réalisé par :
KAISSI Houda

**Objective :** The main purpose behind this lab is to get familiar with Pytorch library to do Classification and Regression tasks by establishing DNN/MLP architectures.

Work to do:

Part one regression:

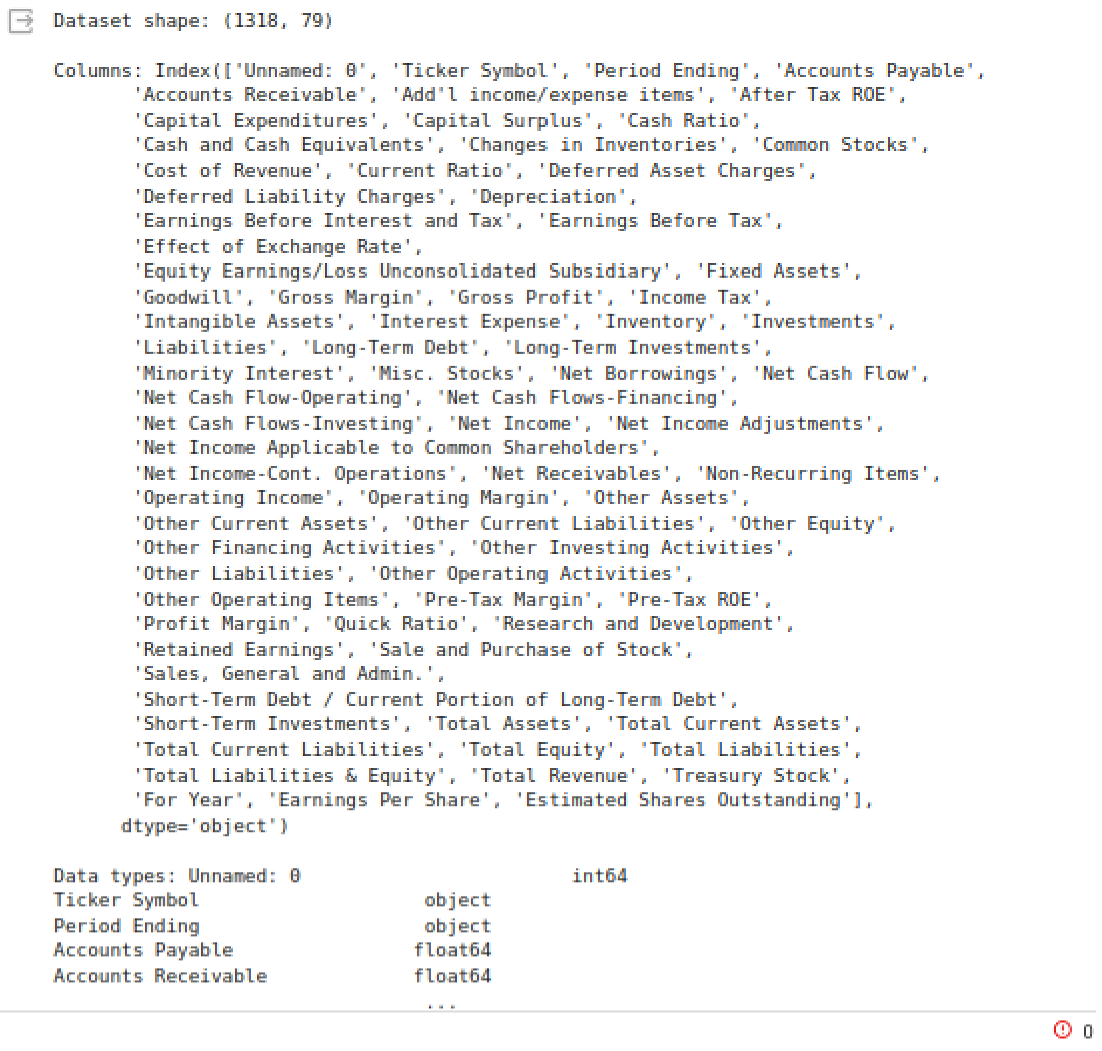
1. Apply the Exploratory data analysis technics to understand and visualize the given Dataset.



```python
import pandas as pd
import matplotlib.pyplot as plt


data = pd.read_csv("fundamentals.csv")


print("Dataset shape:", data.shape)
print("\nColumns:", data.columns)
print("\nData types:", data.dtypes)


print("\nDescriptive Statistics:")
print(data.describe())


print("\nMissing Values:")
print(data.isnull().sum())


# Example: Histograms for numerical features
data.hist(figsize=(10, 10))
plt.show()


plt.scatter(data['Net Income'], data['Total Revenue'])
plt.xlabel('Net Income')
plt.ylabel('Total Revenue')
plt.title('Scatter Plot of Net Income vs Total Revenue')
plt.show()


correlation_matrix = data.corr()
print("\nCorrelation Matrix:")
print(correlation_matrix)

# Heatmap of correlation matrix
plt.figure(figsize=(10, 8))
plt.title('Correlation Heatmap')
plt.imshow(correlation_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.xticks(range(len(data.columns)), data.columns, rotation=90)
plt.yticks(range(len(data.columns)), data.columns)
plt.show()
```

-Load the dataset
-Display basic information about the dataset
-Display descriptive statistics

-Check for missing values
-Visualize data distribution and relationships
-Scatter plot for two numerical features

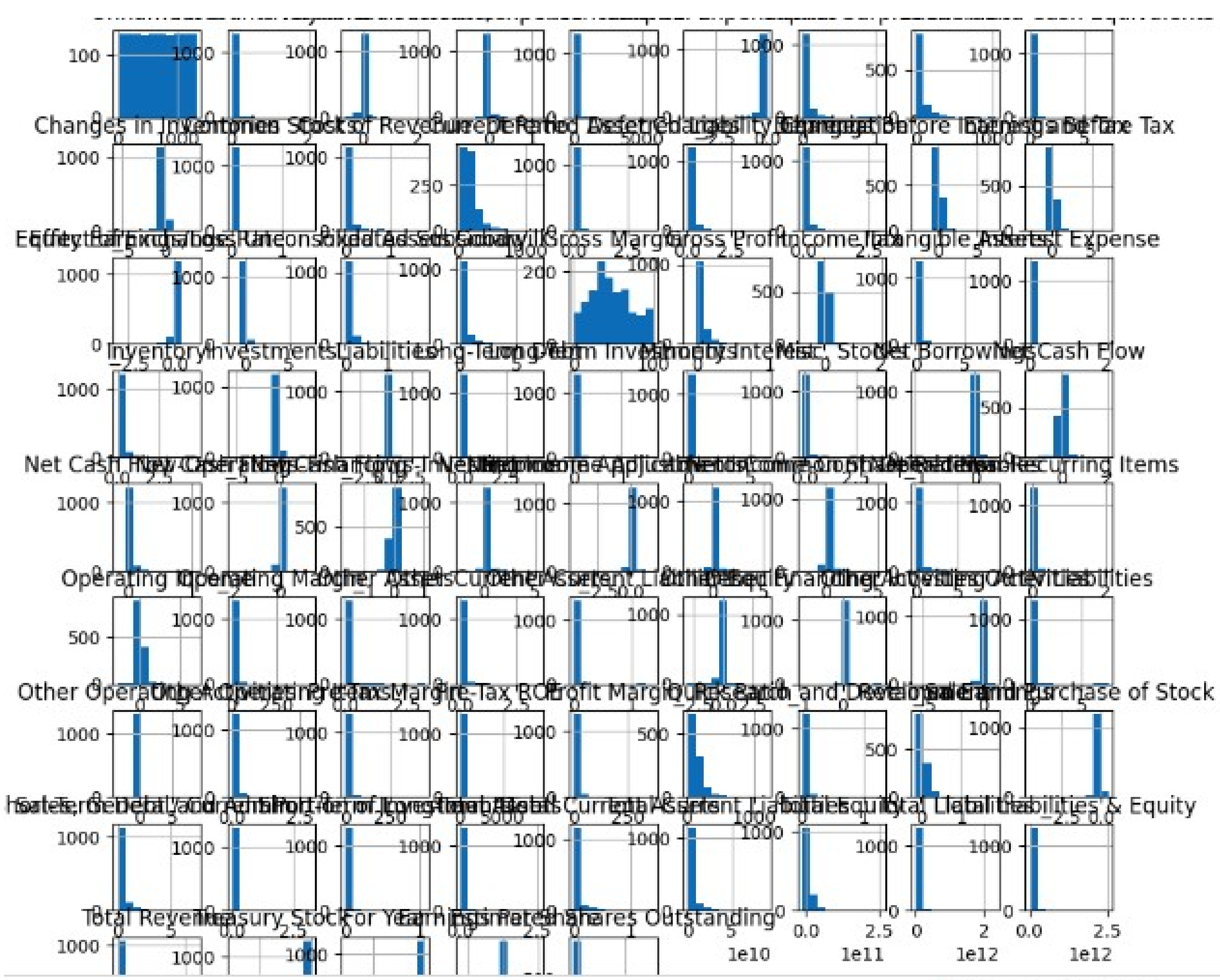


```
Dataset shape: (1318, 79)

Columns: Index(['Unnamed: 0', 'Ticker Symbol', 'Period Ending', 'Accounts Payable',
       'Accounts Receivable', 'Add'l income/expense items', 'After Tax ROE',
       'Capital Expenditures', 'Capital Surplus', 'Cash Ratio',
       'Cash and Cash Equivalents', 'Changes in Inventories', 'Common Stocks',
       'Cost of Revenue', 'Current Ratio', 'Deferred Asset Charges',
       'Deferred Liability Charges', 'Depreciation',
       'Earnings Before Interest and Tax', 'Earnings Before Tax',
       'Effect of Exchange Rate',
       'Equity Earnings/Loss Unconsolidated Subsidiary', 'Fixed Assets',
       'Goodwill', 'Gross Margin', 'Gross Profit', 'Income Tax',
       'Intangible Assets', 'Interest Expense', 'Inventory', 'Investments',
       'Liabilities', 'Long-Term Debt', 'Long-Term Investments',
       'Minority Interest', 'Misc. Stocks', 'Net Borrowings', 'Net Cash Flow',
       'Net Cash Flow-Operating', 'Net Cash Flows-Financing',
       'Net Cash Flows-Investing', 'Net Income', 'Net Income Adjustments',
       'Net Income Applicable to Common Shareholders',
       'Net Income-Cont. Operations', 'Net Receivables', 'Non-Recurring Items',
       'Operating Income', 'Operating Margin', 'Other Assets',
       'Other Current Assets', 'Other Current Liabilities', 'Other Equity',
       'Other Financing Activities', 'Other Investing Activities',
       'Other Liabilities', 'Other Operating Activities',
       'Other Operating Items', 'Pre-Tax Margin', 'Pre-Tax ROE',
       'Profit Margin', 'Quick Ratio', 'Research and Development',
       'Retained Earnings', 'Sale and Purchase of Stock',
       'Sales, General and Admin.',
       'Short-Term Debt / Current Portion of Long-Term Debt',
       'Short-Term Investments', 'Total Assets', 'Total Current Assets',
       'Total Current Liabilities', 'Total Equity', 'Total Liabilities',
       'Total Liabilities & Equity', 'Total Revenue', 'Treasury Stock',
       'For Year', 'Earnings Per Share', 'Estimated Shares Outstanding'],
      dtype='object')

Data types: Unnamed: 0                          int64
Ticker Symbol                    object
Period Ending                    object
Accounts Payable                float64
Accounts Receivable             float64
                                  ...
```

0

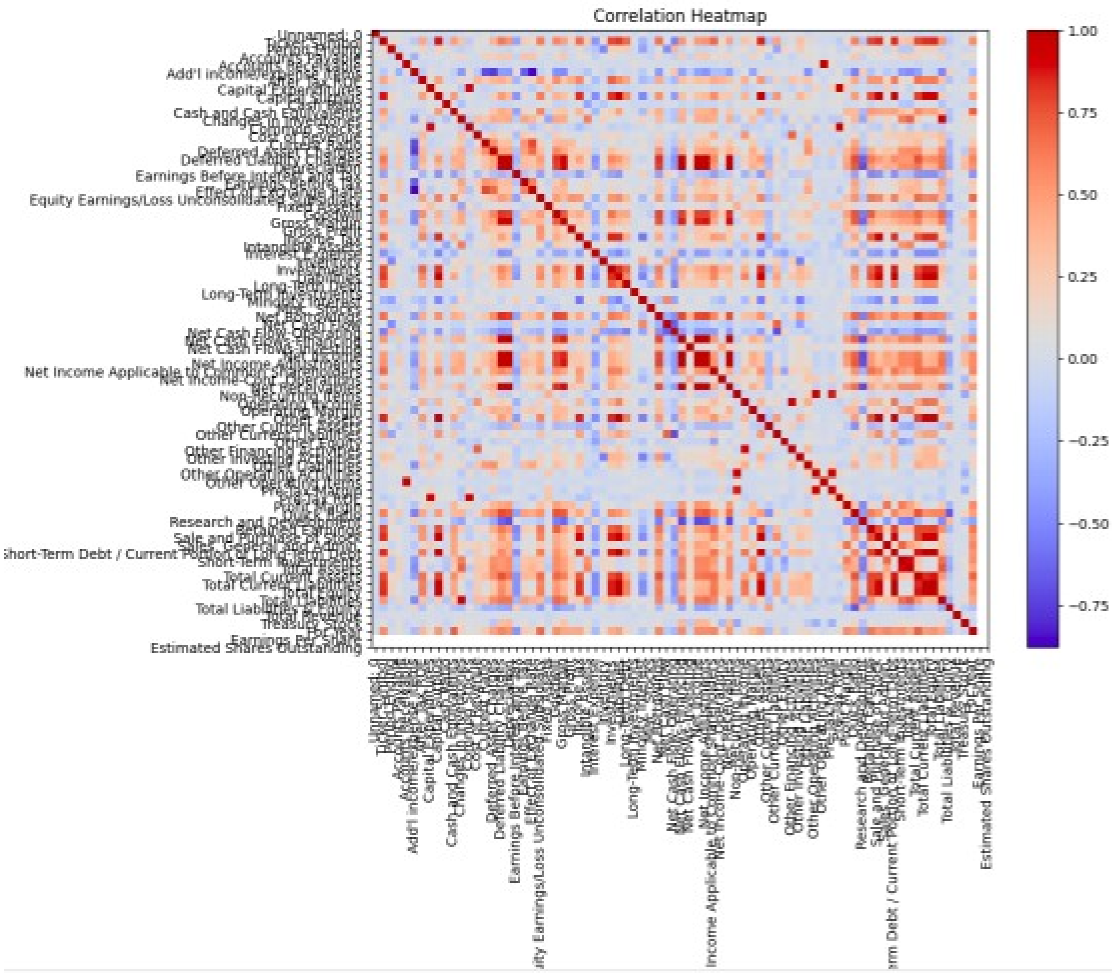
```
Accounts Receivable             float64
                                 ...
Total Revenue                   float64
Treasury Stock                  float64
For Year                        float64
Earnings Per Share              float64
Estimated Shares Outstanding    float64
Length: 79, dtype: object

Descriptive Statistics:
        Unnamed: 0  Accounts Payable  Accounts Receivable  \
count  1318.000000      1.318000e+03         1.318000e+03
mean    658.500000      4.797284e+09        -6.488739e+07
std     380.618138      1.493426e+10         8.417679e+08
min       0.000000      0.000000e+00        -6.452000e+09
25%     329.250000      5.705820e+08        -1.121675e+08
50%     658.500000      1.438700e+09        -1.785250e+07
75%     987.750000      3.712500e+09         7.159750e+06
max    1317.000000      2.069390e+11         2.266400e+10

       Add'l income/expense items  After Tax ROE  Capital Expenditures  \
count                1.318000e+03    1318.000000          1.318000e+03
mean                 5.677452e+07      45.017451         -1.205638e+09
std                  5.869530e+08     244.351628          2.547535e+09
min                 -6.768000e+09       0.000000         -3.798500e+10
25%                 -3.045000e+06      10.000000         -1.190792e+09
50%                  2.803500e+06      16.000000         -3.647495e+08
75%                  3.545000e+07      26.000000         -1.311085e+08
max                  1.161300e+10    5789.000000          5.000000e+06

       Capital Surplus   Cash Ratio  Cash and Cash Equivalents  \
count     1.318000e+03  1087.000000               1.318000e+03
mean      5.652385e+09    75.612695               8.368660e+09
std       1.158267e+10   107.843263               5.266895e+10
min      -7.215000e+08     0.000000               2.100000e+04
25%       4.902875e+08    18.000000               3.346500e+08
50%       2.172970e+09    42.000000               9.555000e+08
75%       6.201975e+09    88.000000               2.429250e+09
max       1.082880e+11  1041.000000               7.281110e+11
```

```
Missing Values:
Unnamed: 0                        0
Ticker Symbol                     0
Period Ending                     0
Accounts Payable                  0
Accounts Receivable               0
                                ...
Total Revenue                     1
Treasury Stock                    1
For Year                        130
Earnings Per Share              158
Estimated Shares Outstanding    158
Length: 79, dtype: int64
          [illegible]
```

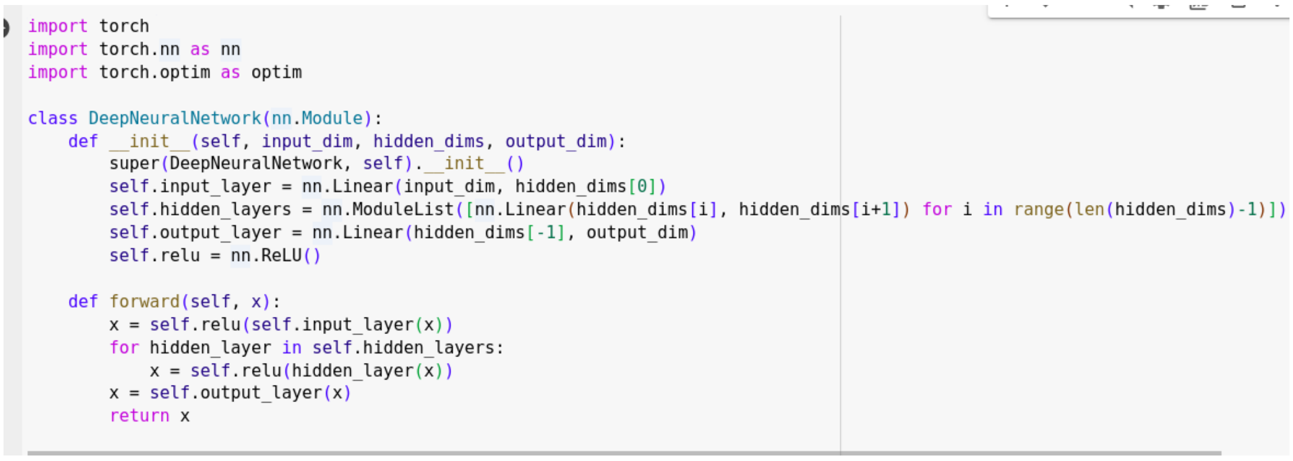


0 s terminée à 15:36

```
7 rows x 77 columns]
```



2. Establish a Deep Neural network Architecture by using PyTorch library to handle the regression task.



```python
import torch
import torch.nn as nn
import torch.optim as optim

class DeepNeuralNetwork(nn.Module):
    def __init__(self, input_dim, hidden_dims, output_dim):
        super(DeepNeuralNetwork, self).__init__()
        self.input_layer = nn.Linear(input_dim, hidden_dims[0])
        self.hidden_layers = nn.ModuleList([nn.Linear(hidden_dims[i], hidden_dims[i+1]) for i in range(len(hidden_dims)-1)])
        self.output_layer = nn.Linear(hidden_dims[-1], output_dim)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.input_layer(x))
        for hidden_layer in self.hidden_layers:
            x = self.relu(hidden_layer(x))
        x = self.output_layer(x)
        return x
```

This code defines a feedforward neural network with multiple hidden layers, each followed by a ReLU activation function. The network architecture is flexible and can handle different input and output dimensions, as well as varying numbers of hidden layers and units.

3. By using GridSearch tool from sklearn library chose the best hyper-parameters (learning rate , optimizers, epoch, model architecture, etc) that will give an efficient model.

- Convert Pandas Series to NumPy array for target variable y
-Create a PyTorchRegressorWrapper object
-Define the parameter grid
-Define the GridSearchCV object
-Fit the model to the training data
-Summarize results
- Evaluate the best model on the test set

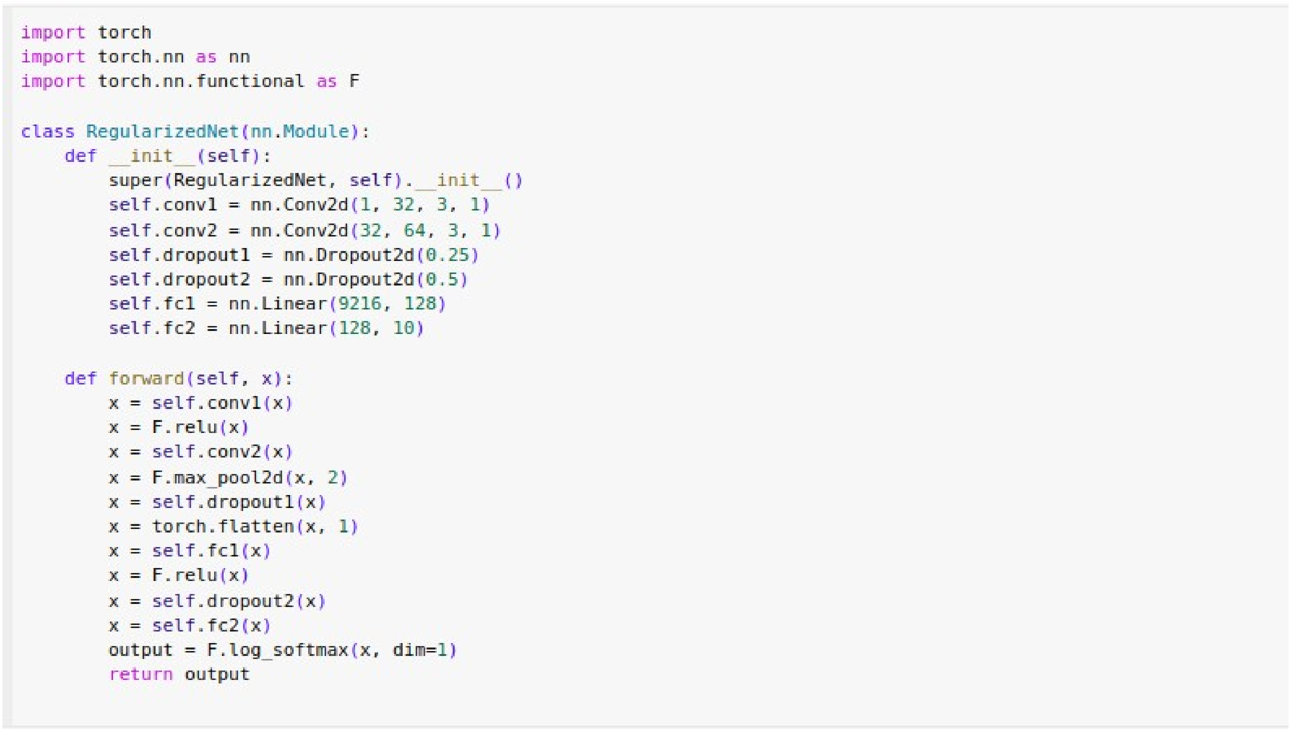
```python
 from sklearn.base import BaseEstimator, RegressorMixin
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import GridSearchCV
import torch
import torch.nn as nn
import torch.optim as optim

class DeepNeuralNetwork(nn.Module):
    def __init__(self, input_dim, hidden_dims, output_dim):
        super(DeepNeuralNetwork, self).__init__()
        self.input_layer = nn.Linear(input_dim, hidden_dims[0])
        self.hidden_layers = nn.ModuleList([nn.Linear(hidden_dims[i], hidden_dims[i+1]) for i in range(len(hidden_dims) - 1)])
        self.output_layer = nn.Linear(hidden_dims[-1], output_dim)
        self.relu = nn.ReLU()

    def forward(self, x):
        x = self.relu(self.input_layer(x))
        for layer in self.hidden_layers:
            x = self.relu(layer(x))
        x = self.output_layer(x)
        return x

y_train_np = y_train.values


pytorch_regressor_wrapper = PyTorchRegressorWrapper(model=DeepNeuralNetwork)


param_grid = {
    'learning_rate': [0.001, 0.01, 0.1],
    'optimizer_class': [optim.Adam, optim.SGD]
}


grid = GridSearchCV(estimator=pytorch_regressor_wrapper, param_grid=param_grid, scoring='neg_mean_squared_error', cv=3)
```

```python
best_model = grid_result.best_estimator_
y_pred = best_model.predict(X_test_scaled)
mse = mean_squared_error(y_test, y_pred)
print("Mean Squared Error on Test Set:", mse)
```

4. Visualize the two graph (Loss / Epochs) and (Accuracy / Epochs) for both training and test data, give your interpretation.

```python
plt.figure(figsize=(10, 5))
plt.plot(train_losses, label='Training Loss', color='blue')
plt.plot(test_losses, label='Test Loss', color='red')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Loss vs Epochs')
plt.legend()
plt.show()

# Plot Accuracy vs Epochs
plt.figure(figsize=(10, 5))
plt.plot(train_accuracies, label='Training Accuracy', color='blue')
plt.plot(test_accuracies, label='Test Accuracy', color='red')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Accuracy vs Epochs')
plt.legend()
plt.show()
```

5. Apply several regularization techniques on your architecture then compare the obtained result with the first model.

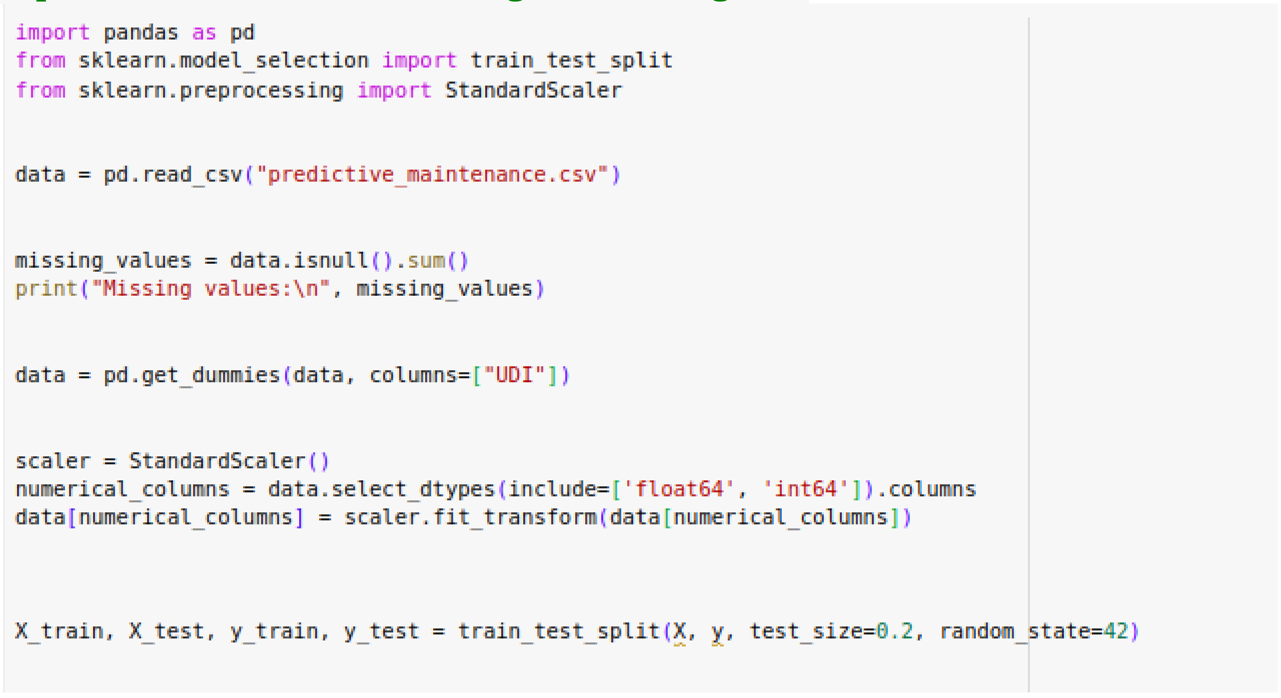


```python
import torch
import torch.nn as nn
import torch.nn.functional as F

class RegularizedNet(nn.Module):
    def __init__(self):
        super(RegularizedNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

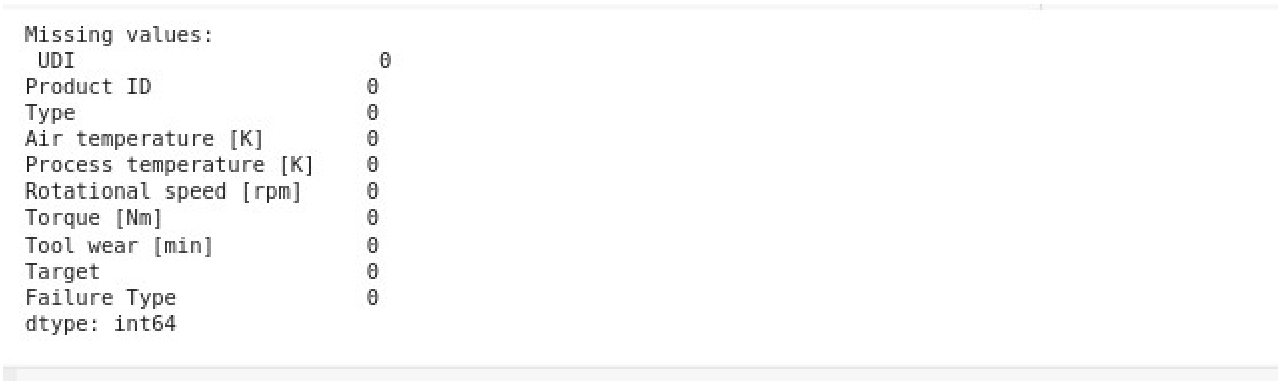
Part two multi class classification:

1. Apply the per-processing technics and the given dataset to clean, standardization/normalization of the data.

-Check for missing values
-Encode categorical variables
-Standardize or normalize numerical features
-Split the dataset into training and testing sets

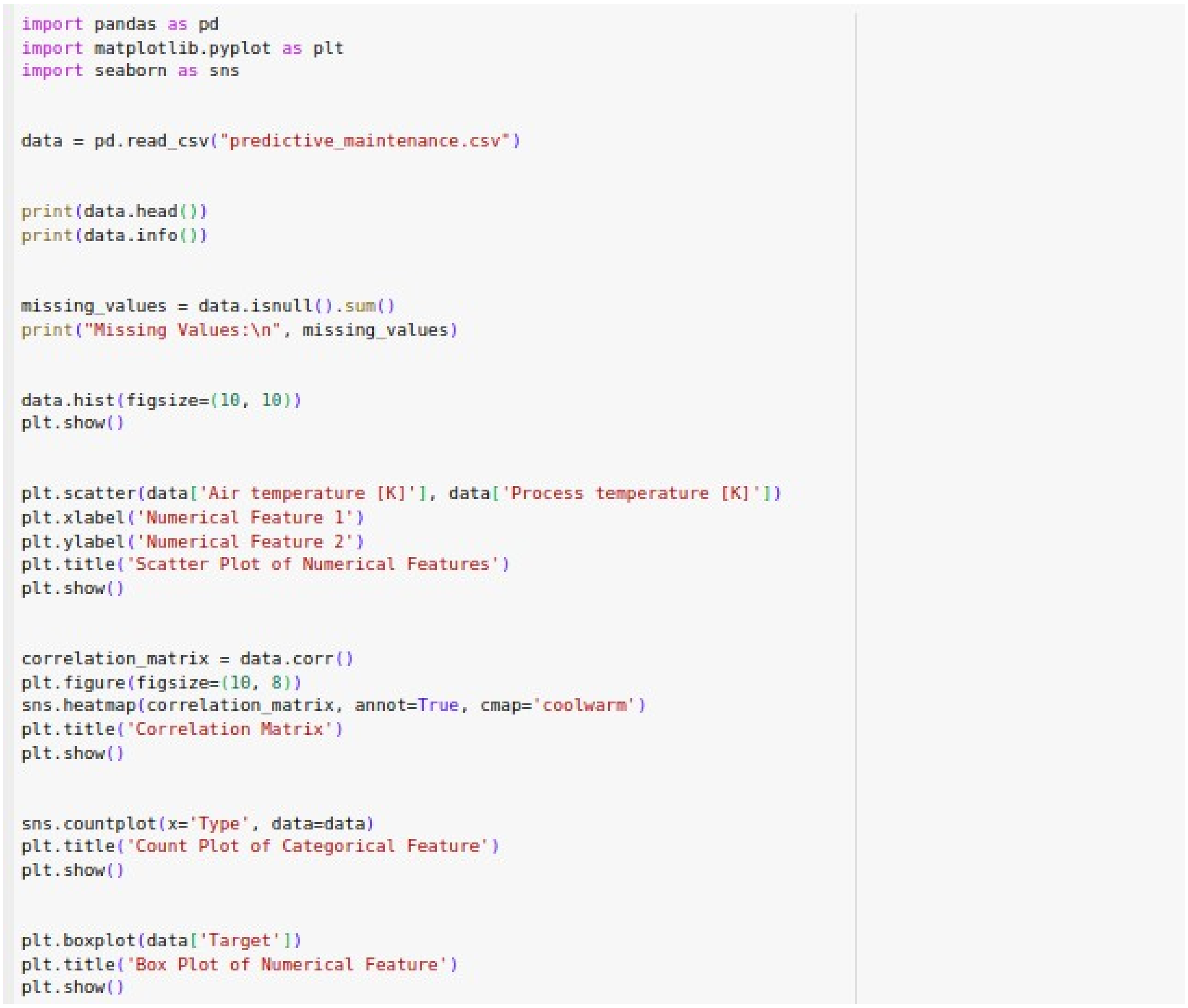


```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler


data = pd.read_csv("predictive_maintenance.csv")


missing_values = data.isnull().sum()
print("Missing values:\n", missing_values)


data = pd.get_dummies(data, columns=["UDI"])


scaler = StandardScaler()
numerical_columns = data.select_dtypes(include=['float64', 'int64']).columns
data[numerical_columns] = scaler.fit_transform(data[numerical_columns])



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


```
Missing values:
 UDI                         0
Product ID                  0
Type                        0
Air temperature [K]         0
Process temperature [K]     0
Rotational speed [rpm]      0
Torque [Nm]                 0
Tool wear [min]             0
Target                      0
Failure Type                0
dtype: int64
```

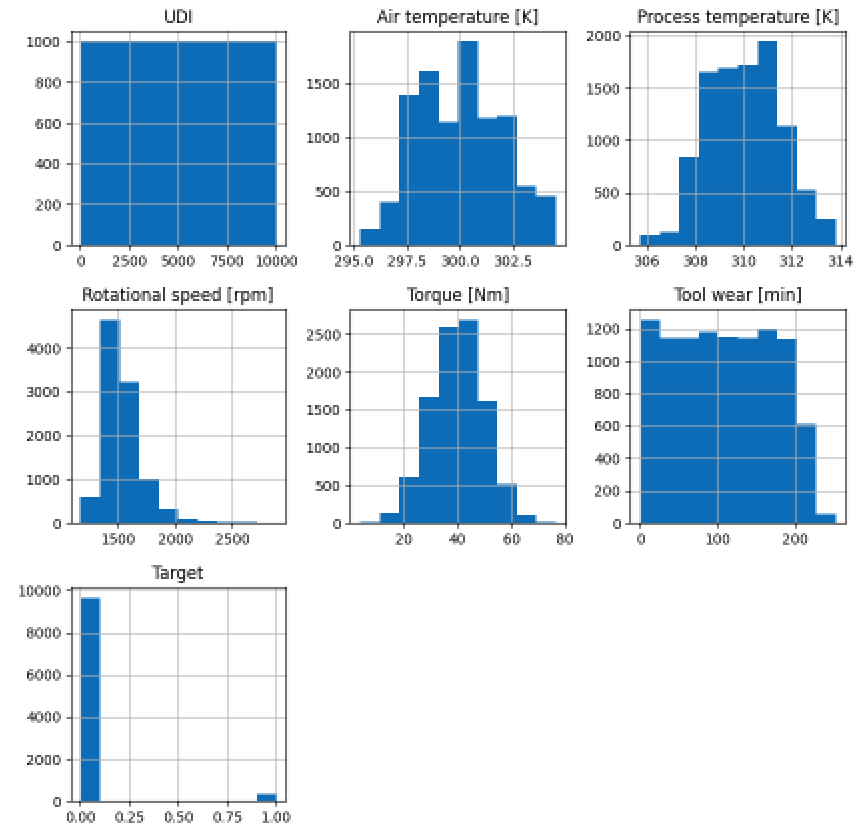
2. Apply the Exploratory data analysis technics to understand and visualize the given Dataset.

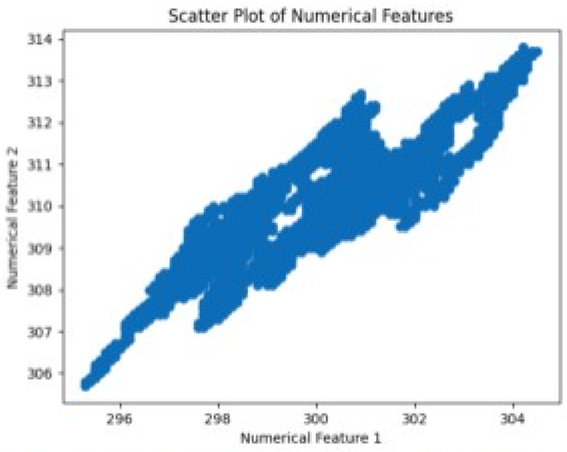
Explore the Dataset
Handle Missing Values (if any)
Visualize Distributions
Visualize Relationships
Correlation Analysis
Explore Categorical Features

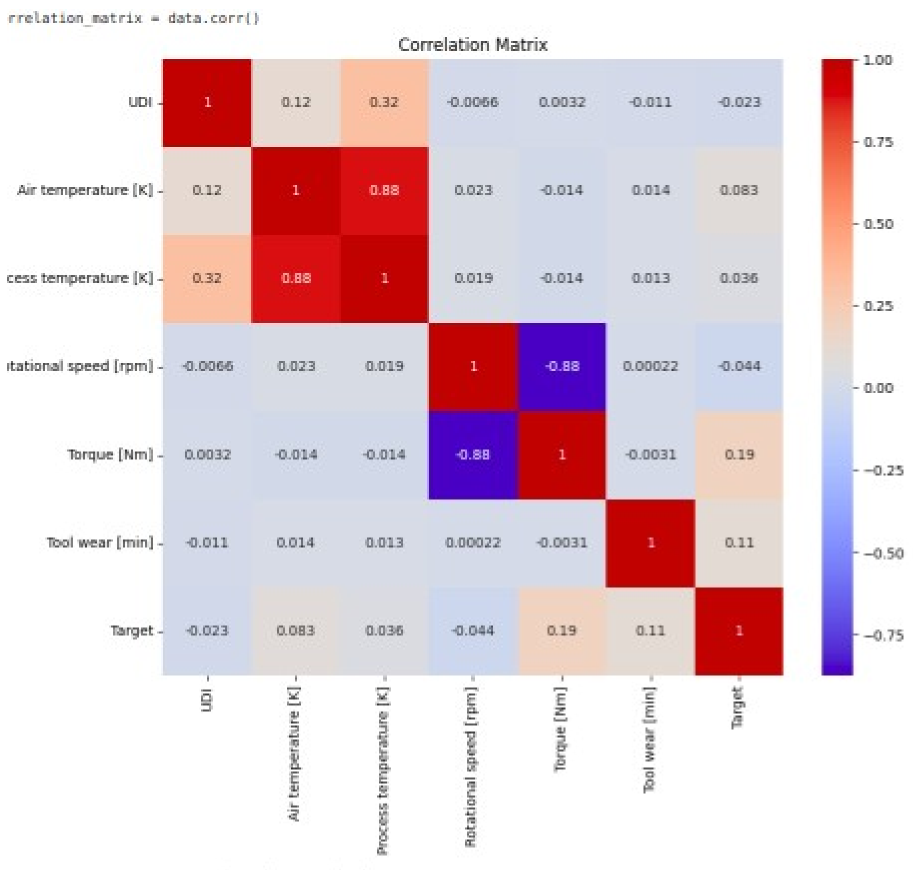


```python
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns


data = pd.read_csv("predictive_maintenance.csv")


print(data.head())
print(data.info())


missing_values = data.isnull().sum()
print("Missing Values:\n", missing_values)


data.hist(figsize=(10, 10))
plt.show()


plt.scatter(data['Air temperature [K]'], data['Process temperature [K]'])
plt.xlabel('Numerical Feature 1')
plt.ylabel('Numerical Feature 2')
plt.title('Scatter Plot of Numerical Features')
plt.show()


correlation_matrix = data.corr()
plt.figure(figsize=(10, 8))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix')
plt.show()


sns.countplot(x='Type', data=data)
plt.title('Count Plot of Categorical Feature')
plt.show()


plt.boxplot(data['Target'])
plt.title('Box Plot of Numerical Feature')
plt.show()
```

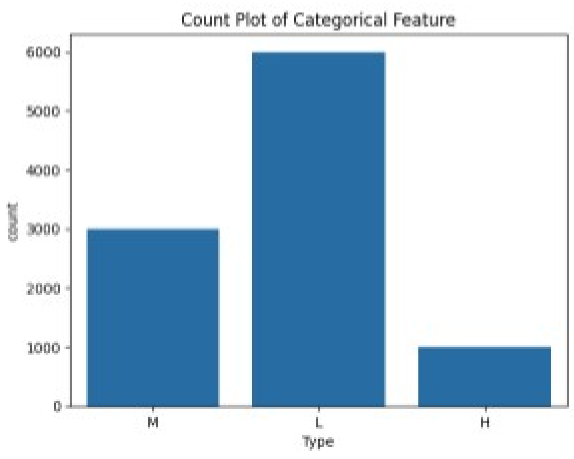
```
dtype: int64
```


UDI
Air temperature [K]
Process temperature [K]
Rotational speed [rpm]
Torque [Nm]
Tool wear [min]
Target


Scatter Plot of Numerical Features
Numerical Feature 1
Numerical Feature 2

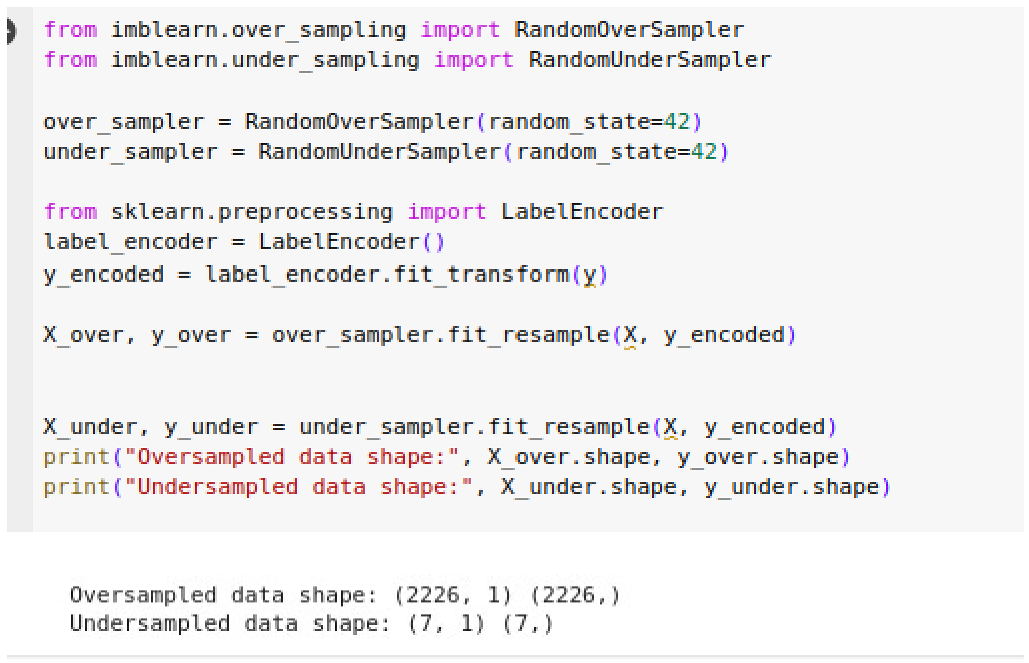


```
rrelation_matrix = data.corr()
```



3. Apply data augmentation technics to balance the given Dataset.

```
 'X' is  feature matrix and 'y' is  target vector
```

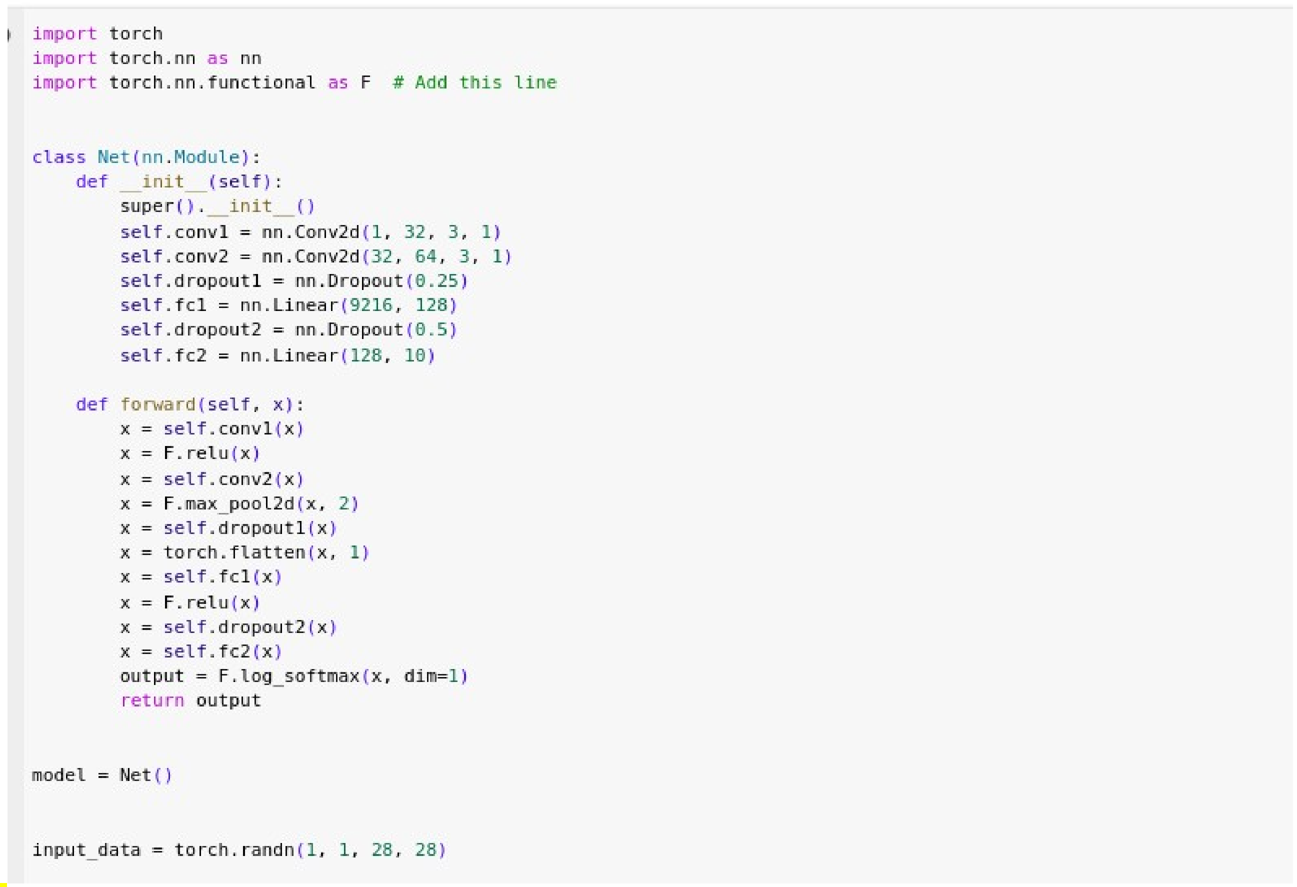


```
from imblearn.over_sampling import RandomOverSampler
from imblearn.under_sampling import RandomUnderSampler

over_sampler = RandomOverSampler(random_state=42)
under_sampler = RandomUnderSampler(random_state=42)

from sklearn.preprocessing import LabelEncoder
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

X_over, y_over = over_sampler.fit_resample(X, y_encoded)


X_under, y_under = under_sampler.fit_resample(X, y_encoded)
print("Oversampled data shape:", X_over.shape, y_over.shape)
print("Undersampled data shape:", X_under.shape, y_under.shape)
```

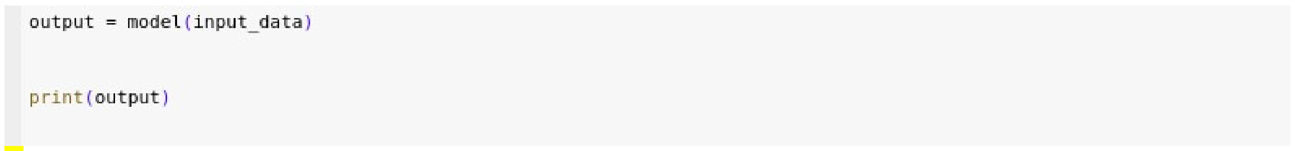
```
Oversampled data shape: (2226, 1) (2226,)
Undersampled data shape: (7, 1) (7,)
```

4. Establish a Deep Neural network Architecture by using PyTorch library to handle the multiclass classification task.

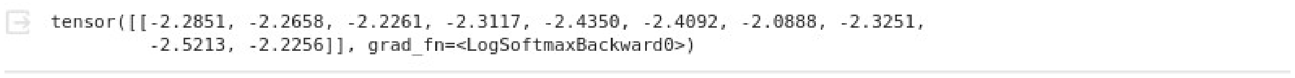
Definel'architecture  of network
Create an instance of the network
Assume 'input_data' is the input tensor of appropriate shape
Pass the input data through the network to get the output
Print the output tensor



```python
import torch
import torch.nn as nn
import torch.nn.functional as F  # Add this line


class Net(nn.Module):
    def __init__(self):
        super().__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout(0.25)
        self.fc1 = nn.Linear(9216, 128)
        self.dropout2 = nn.Dropout(0.5)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


model = Net()


input_data = torch.randn(1, 1, 28, 28)
```

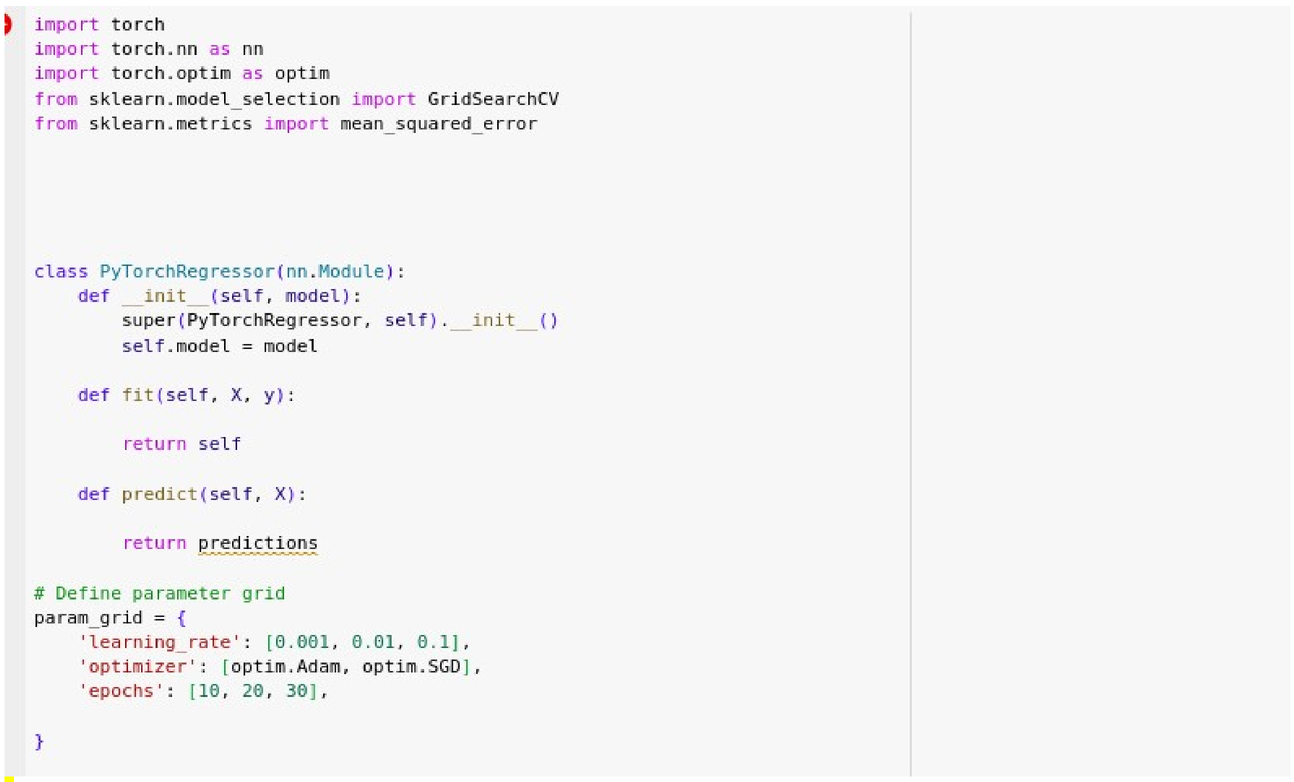


```python
output = model(input_data)


print(output)
```

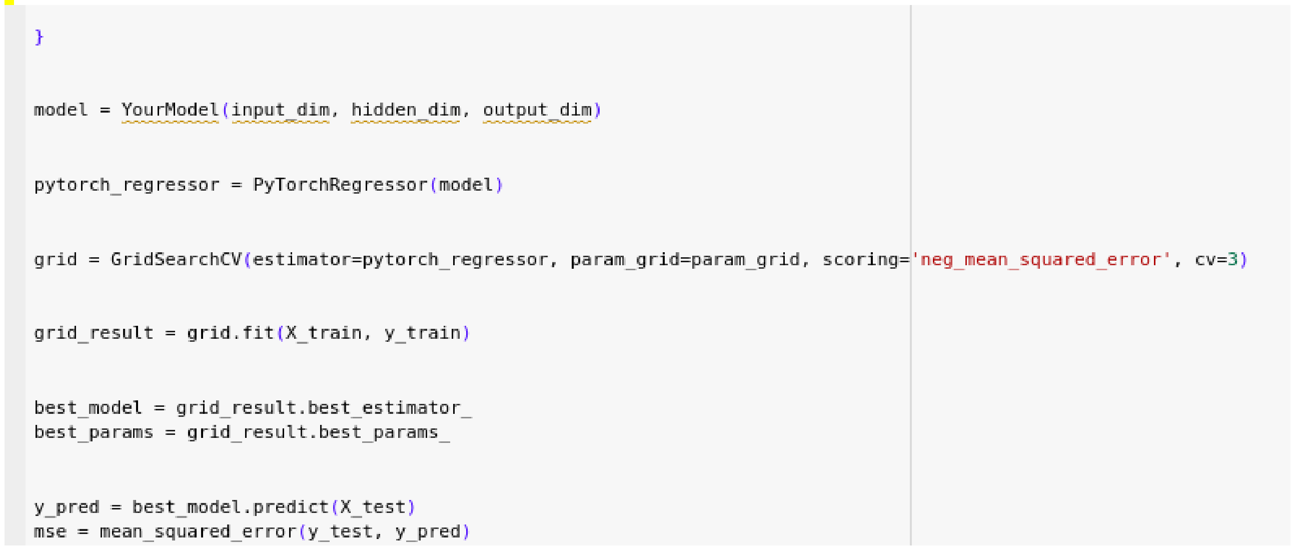


```
tensor([[-2.2851, -2.2658, -2.2261, -2.3117, -2.4350, -2.4092, -2.0888, -2.3251,
         -2.5213, -2.2256]], grad_fn=<LogSoftmaxBackward0>)
```

5. By using GridSearch tool from sklearn library chose the best hyper-parameters (learning rate , optimizers, epoch, model architecture, etc) that will give an efficient model.

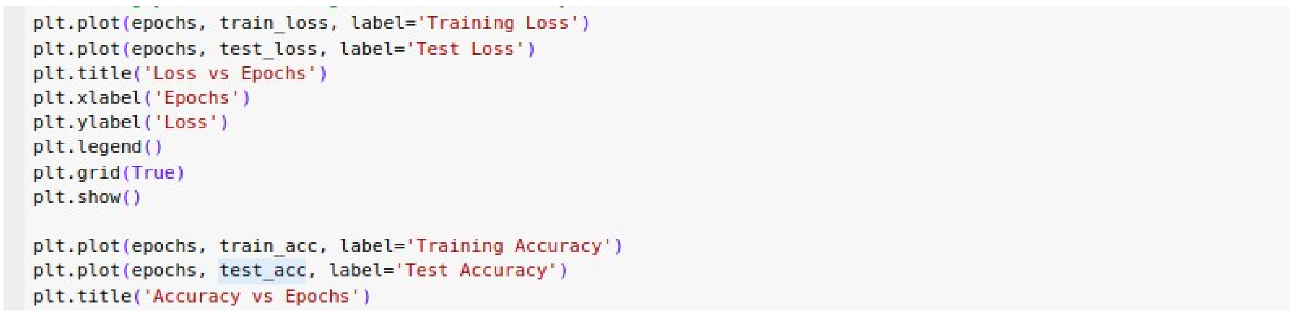


```python
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error




class PyTorchRegressor(nn.Module):
    def __init__(self, model):
        super(PyTorchRegressor, self).__init__()
        self.model = model

    def fit(self, X, y):

        return self

    def predict(self, X):

        return predictions

# Define parameter grid
param_grid = {
    'learning_rate': [0.001, 0.01, 0.1],
    'optimizer': [optim.Adam, optim.SGD],
    'epochs': [10, 20, 30],

}
```



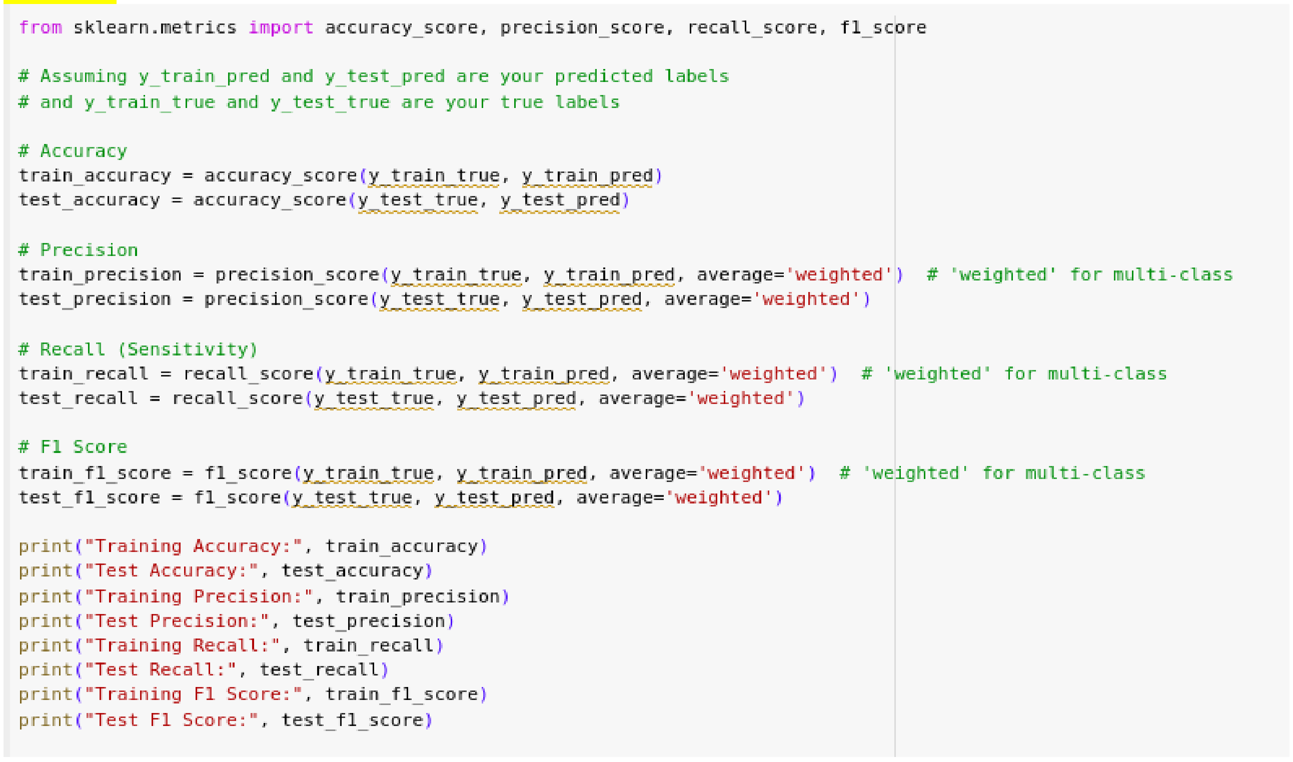
```python
}


model = YourModel(input_dim, hidden_dim, output_dim)


pytorch_regressor = PyTorchRegressor(model)


grid = GridSearchCV(estimator=pytorch_regressor, param_grid=param_grid, scoring='neg_mean_squared_error', cv=3)


grid_result = grid.fit(X_train, y_train)


best_model = grid_result.best_estimator_
best_params = grid_result.best_params_


y_pred = best_model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
```

6. Visualize the two graph (Loss / Epochs) and (Accuracy / Epochs) for both training and test data, give your interpretation.



```
plt.plot(epochs, train_loss, label='Training Loss')
plt.plot(epochs, test_loss, label='Test Loss')
plt.title('Loss vs Epochs')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)
plt.show()

plt.plot(epochs, train_acc, label='Training Accuracy')
plt.plot(epochs, test_acc, label='Test Accuracy')
plt.title('Accuracy vs Epochs')
```

7. Calculate metrics like accuracy, sensitivity, f1 score, etc, on both training and test dataset.



```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

# Assuming y_train_pred and y_test_pred are your predicted labels
# and y_train_true and y_test_true are your true labels

# Accuracy
train_accuracy = accuracy_score(y_train_true, y_train_pred)
test_accuracy = accuracy_score(y_test_true, y_test_pred)

# Precision
train_precision = precision_score(y_train_true, y_train_pred, average='weighted')  # 'weighted' for multi-class
test_precision = precision_score(y_test_true, y_test_pred, average='weighted')

# Recall (Sensitivity)
train_recall = recall_score(y_train_true, y_train_pred, average='weighted')  # 'weighted' for multi-class
test_recall = recall_score(y_test_true, y_test_pred, average='weighted')

# F1 Score
train_f1_score = f1_score(y_train_true, y_train_pred, average='weighted')  # 'weighted' for multi-class
test_f1_score = f1_score(y_test_true, y_test_pred, average='weighted')

print("Training Accuracy:", train_accuracy)
print("Test Accuracy:", test_accuracy)
print("Training Precision:", train_precision)
print("Test Precision:", test_precision)
print("Training Recall:", train_recall)
print("Test Recall:", test_recall)
print("Training F1 Score:", train_f1_score)
print("Test F1 Score:", test_f1_score)
```

8. Apply several regularization techniques on your architecture then compare the obtained result with the first model.

```
import torch
import torch.nn as nn
import torch.nn.functional as F

class RegularizedNet(nn.Module):
    def __init__(self):
        super(RegularizedNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output


model = RegularizedNet()
```

```

model = RegularizedNet()



output = model(input_data)


print(output)
```

```
tensor([[-2.5653, -2.3127, -2.3356, -2.1195, -2.2491, -2.3800, -2.2352, -2.2281,
         -2.3599, -2.3018]], grad_fn=<LogSoftmaxBackward0>)
```