In [116…

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

1. Naive Bayes Classifier

In [40]:

```python
df = pd.read_csv('logistic.csv')
df.Y = df.Y.map({'M':'0', 'B':'1'})
df.Y = df.Y.astype(float)
df.head()
```

Out[40]:

	Y	X1	X2	X3	X4	X5	X6	X7	X8	X9	...	X21	X22	X23	X24	X25	X26	
0	0.0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	...	25.38	17.33	184.60	2019.0	0.1622	0.6656	0.
1	0.0	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	...	24.99	23.41	158.80	1956.0	0.1238	0.1866	0.
2	0.0	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	...	23.57	25.53	152.50	1709.0	0.1444	0.4245	0.
3	0.0	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	...	14.91	26.50	98.87	567.7	0.2098	0.8663	0.
4	0.0	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	...	22.54	16.67	152.20	1575.0	0.1374	0.2050	0.

5 rows × 31 columns

Normalizing the dataset

In [77]:

```python
# x = df.drop(columns='Y')
# y = df.Y
x = df.drop(columns='Y').values
y = df['Y'].values
x = (x - x.mean())/x.std()
```

a) Naive Bayes Classifier from scratch

In [111…

```python
from sklearn.model_selection import train_test_split

class NaiveBayes:
    def fit(self, X, y):
        n_samples, n_features = X.shape
        self._classes = np.unique(y)
        n_classes = len(self._classes)

        # calculating mean, var, and prior for each class
        self._mean = np.zeros((n_classes, n_features), dtype=np.float64)
        self._var = np.zeros((n_classes, n_features), dtype=np.float64)
        self._priors = np.zeros(n_classes, dtype=np.float64)

        for idx, c in enumerate(self._classes):
            X_c = X[y == c]
            self._mean[idx, :] = X_c.mean(axis=0)
            self._var[idx, :] = X_c.var(axis=0)
            self._priors[idx] = X_c.shape[0] / float(n_samples)

    def predict(self, X):
        y_pred = [self._predict(x) for x in X]
        return np.array(y_pred)

    def _predict(self, x):
        posteriors = []

        # calculating  probability for each class
        for idx, c in enumerate(self._classes):
            prior = np.log(self._priors[idx])
            posterior = np.sum(np.log(self._pdf(idx, x)))
            posterior = prior + posterior
            posteriors.append(posterior)

        # returning class with highest posterior probability
        return self._classes[np.argmax(posteriors)]

    def _pdf(self, class_idx, x):
        mean = self._mean[class_idx]
        var = self._var[class_idx]
        numerator = np.exp(-((x - mean) ** 2) / (2 * var))
        denominator = np.sqrt(2 * np.pi * var)
        return numerator / denominator
```

```python
def accuracy(y_true, y_pred):
        accuracy = np.sum(y_true == y_pred) / len(y_true)
        return accuracy*100

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=50)

nb = NaiveBayes()
nb.fit(X_train, y_train)
predictions = nb.predict(X_test)

print("Naive Bayes classification accuracy: ", accuracy(y_test, predictions))
```

```
Naive Bayes classification accuracy:  93.85964912280701
```

b) Evaluation Metrics

In [112…

```python
def evaluation_metrics(y_true, y_pred):
    TP = TN = FP = FN = 0

    for true, pred in zip(y_true, y_pred):
        if true == 1 and pred == 1:
            TP += 1
        elif true == 0 and pred == 0:
            TN += 1
        elif true == 0 and pred == 1:
            FP += 1
        elif true == 1 and pred == 0:
            FN += 1

    accuracyz = (TP + TN) / (TP + TN + FP + FN) * 100
    precision = TP / (TP + FP) * 100
    recall = TP / (TP + FN) * 100
    f1 = 2 * (precision * recall) / (precision + recall)

    return accuracyz, precision, recall, f1

accuracyz, precision, recall, f1 = evaluation_metrics(y_test, predictions)
print("Accuracy:", accuracyz)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
```

```
Accuracy: 93.85964912280701
Precision: 97.22222222222221
Recall: 93.33333333333333
F1-Score: 95.23809523809523
```

c) Implementing with sklearn GaussianNB

In [113…

```python
from sklearn.naive_bayes import GaussianNB

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=50)

model = GaussianNB()
model.fit(X_train, y_train)

y_pred1 = model.predict(X_test)
print("GaussianNB Accuracy:", accuracy(y_test, y_pred1 ))
```

```
GaussianNB Accuracy: 94.73684210526315
```

In [115…

```python
accuracyz, precision, recall, f1 = evaluation_metrics(y_test, y_pred1)
print("Accuracy:", accuracyz)
print("Precision:", precision)
print("Recall:", recall)
print("F1-Score:", f1)
```

```
Accuracy: 94.73684210526315
Precision: 97.26027397260275
Recall: 94.66666666666667
F1-Score: 95.94594594594595
```

In [ ]:

2. MLP Regressor with PyTorch

In [146…

```python
import torch
import torch.nn as nn
import torch.optim as optim
from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

```python
from torch.utils.data import DataLoader, TensorDataset
import torch.nn.functional as F
from tqdm import trange, tqdm
```

Data Pre-processing

In [147…

```python
data = fetch_california_housing()
X, y = data.data, data.target


X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.2, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

In [148…

```python
y_train.shape
```

Out[148… (16512,)

Converting to PyTorch tensors

In [149…

```python
X_train, y_train = torch.tensor(X_train, dtype=torch.float32), torch.tensor(y_train, dtype=torch.float32).view(
X_val, y_val = torch.tensor(X_val, dtype=torch.float32), torch.tensor(y_val, dtype=torch.float32).view(-1, 1)
X_test, y_test = torch.tensor(X_test, dtype=torch.float32), torch.tensor(y_test, dtype=torch.float32).view(-1,

# Creatig the DataLoaders
batch_size = 64
train_loader = DataLoader(TensorDataset(X_train, y_train), batch_size=batch_size, shuffle=True)
val_loader = DataLoader(TensorDataset(X_val, y_val), batch_size=batch_size)
test_loader = DataLoader(TensorDataset(X_test, y_test), batch_size=batch_size)
```

In [153…

```python
# Class MLP ---
class CustomMLP(nn.Module):
    def __init__(self, input_size, hidden_layers, activation_fn):
        super(CustomMLP, self).__init__()
        self.layers = nn.ModuleList()
        self.activation_fn = activation_fn

        # creating hidden layers
        prev_size = input_size
        for hidden_size in hidden_layers:
            self.layers.append(nn.Linear(prev_size, hidden_size))
            prev_size = hidden_size

        # Output layer
        self.output = nn.Linear(prev_size, 1)

    def forward(self, x):
        for layer in self.layers:
            x = self.activation_fn(layer(x))
        return self.output(x)


# Initializing  the model, optimizer, loss function...
input_size = X_train.shape[1]
hidden_layers = [64, 32, 16]
activation_fn = F.relu

model = CustomMLP(input_size, hidden_layers, activation_fn).to(device)
optimizer = optim.Adam(model.parameters(), lr=0.001)
criterion = nn.MSELoss()
```

In [162…

```python
#Training and Validation...
def train_epoch(model, dataloader, criterion, optimizer, device):
    model.train()
    train_loss = 0
    for X_batch, y_batch in dataloader:
        X_batch, y_batch = X_batch.to(device), y_batch.to(device)

        # Forward pass
        predictions = model(X_batch)
        loss = criterion(predictions, y_batch)

        # Backward pass
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

```
        train_loss += loss.item() * X_batch.size(0)

    return train_loss / len(dataloader.dataset)

def validate_epoch(model, dataloader, criterion, device):
    model.eval()
    val_loss = 0
    with torch.no_grad():
        for X_batch, y_batch in dataloader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
            predictions = model(X_batch)
            loss = criterion(predictions, y_batch)
            val_loss += loss.item() * X_batch.size(0)

    return val_loss / len(dataloader.dataset)

device = "cpu"
model.to(device)
n_epochs = 50

train_losses = []
val_losses = []

for epoch in trange(n_epochs, desc="Epochs"):
    train_loss = train_epoch(model, train_loader, criterion, optimizer, device)
    train_losses.append(train_loss)

    val_loss = validate_epoch(model, val_loader, criterion, device)
    val_losses.append(val_loss)

    if epoch % 10 == 0:
        print(f"Epoch {epoch+1}/{n_epochs} - train Loss: {train_loss:.4f} - validation Loss: {val_loss:.4f}")
```

```
Epochs:   2%|▏         | 1/50 [00:02<02:18,  2.82s/it]
Epoch 1/50 - train Loss: 0.2346 - validation Loss: 0.2691
Epochs:  22%|██        | 11/50 [00:18<01:03,  1.62s/it]
Epoch 11/50 - train Loss: 0.2319 - validation Loss: 0.2743
Epochs:  42%|████      | 21/50 [00:37<00:54,  1.88s/it]
Epoch 21/50 - train Loss: 0.2281 - validation Loss: 0.2657
Epochs:  62%|██████    | 31/50 [01:01<00:44,  2.32s/it]
Epoch 31/50 - train Loss: 0.2236 - validation Loss: 0.2650
Epochs:  82%|████████  | 41/50 [01:14<00:10,  1.12s/it]
Epoch 41/50 - train Loss: 0.2216 - validation Loss: 0.2634
Epochs: 100%|██████████| 50/50 [01:25<00:00,  1.72s/it]
```

Evaluation on Test dataset

In [163…

```
def test_model(model, dataloader, criterion, device):
    model.eval()
    test_loss = 0
    with torch.no_grad():
        for X_batch, y_batch in dataloader:
            X_batch, y_batch = X_batch.to(device), y_batch.to(device)
            predictions = model(X_batch)
            loss = criterion(predictions, y_batch)
            test_loss += loss.item() * X_batch.size(0)

    return test_loss / len(dataloader.dataset)

test_loss = test_model(model, test_loader, criterion, device)
print(f"Test Loss: {test_loss:.4f}")
```

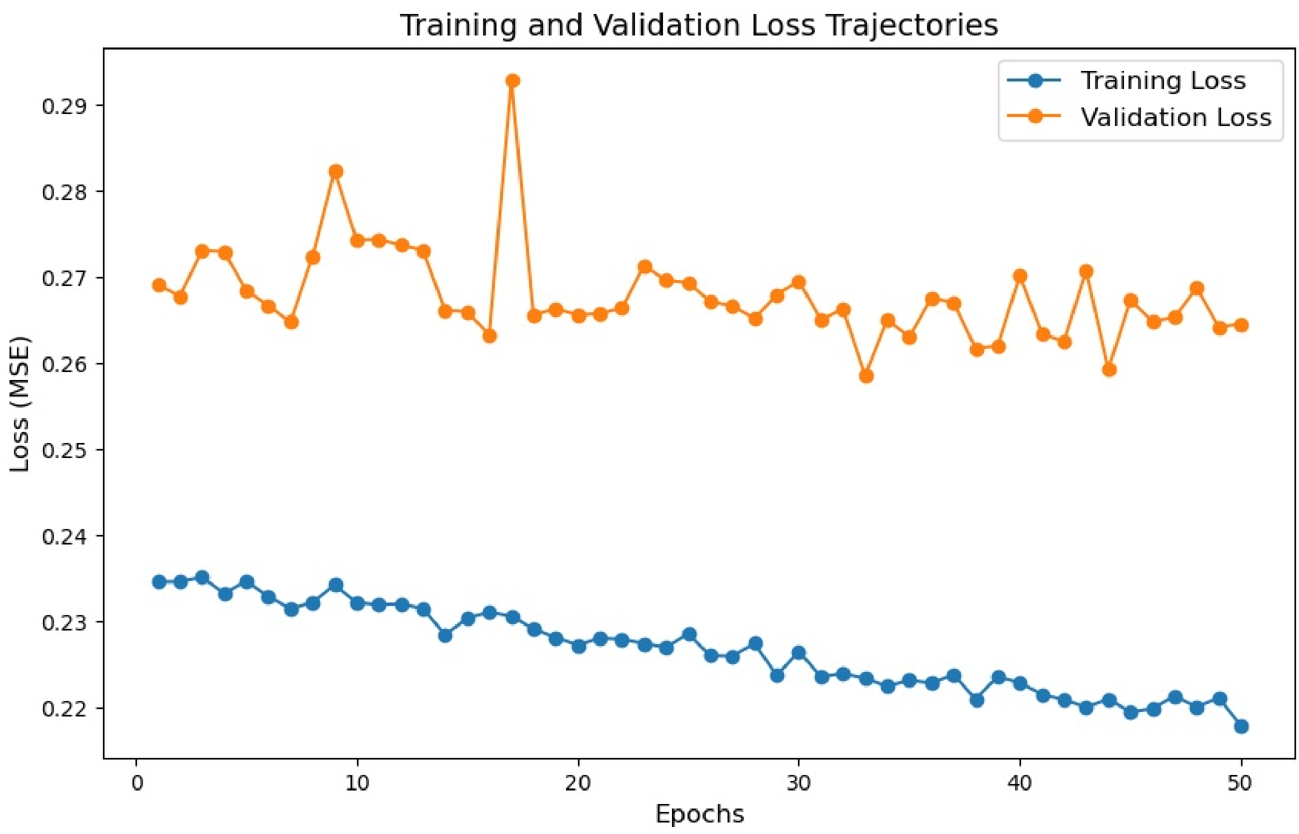
```
Test Loss: 0.2635
```

Plot training and validation loss trajectories

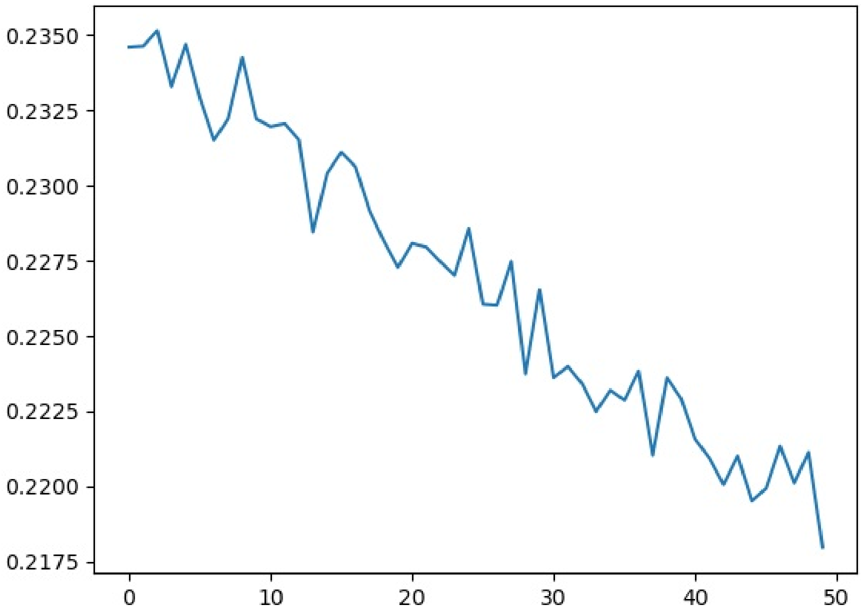
In [164…

```
plt.figure(figsize=(10, 6))
plt.plot(range(1, n_epochs + 1), train_losses, label="Training Loss", marker='o')
plt.plot(range(1, n_epochs + 1), val_losses, label="Validation Loss", marker='o')

plt.xlabel("Epochs", fontsize=12)
plt.ylabel("Loss (MSE)", fontsize=12)
plt.title("Training and Validation Loss Trajectories", fontsize=14)
plt.legend(fontsize=12)
plt.show()
```



In [165…
```
plt.plot(train_losses)
plt.show()
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js