SRECHARAN
SELVAM

SSELVAM

Homework 9

Instructions

This homework contains **3** concepts and **3** programming questions. In MS word or a similar text editor, write down the problem number and your answer for each problem. Combine all answers for concept questions in a single PDF file. Export/print the Jupyter notebook as a PDF file including the code you implemented and the outputs of the program. Make sure all plots and outputs are visible in the PDF.

Combine all answers into a single PDF named andrewID_hw9.pdf and submit it to Gradescope before the due date. Refer to the syllabus for late homework policy. Please assign each question a page by using the "Assign Questions and Pages" feature in Gradescope.

Here is a breakdown of the points for programming questions:

Name	Points
M9-L1-P1	15
M9-L1-P2	15
M9-HW1	60

Problem 1 (6 points)

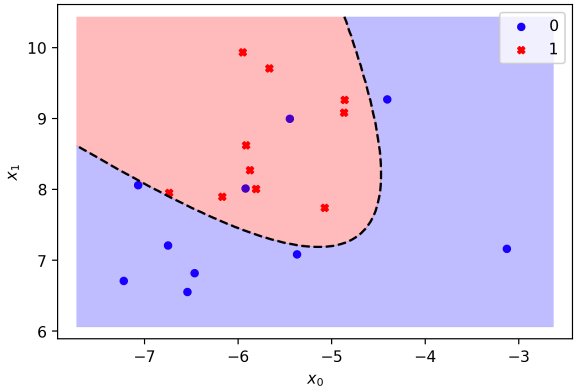
Provided the following ground truth vector, y = [-4, 8, 7, -15,

12] and
the prediction vector, y^ = [2, 9, -1, -16, 18], compute the MAE, MSE, and MAPE without the use of in-built functions

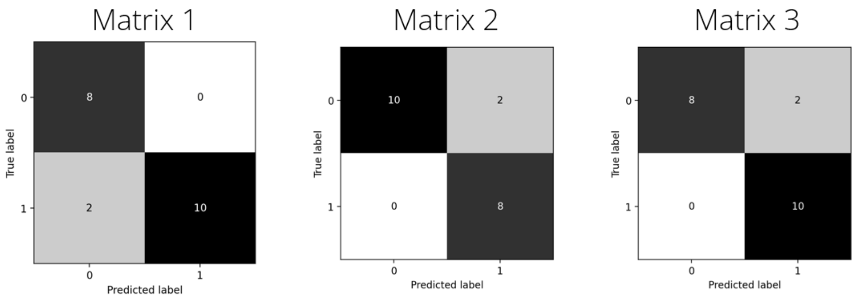
☐

Problem 2 (2 points)

Consider the following model and data, where we use the convention that 1 is the positive outcome.



Which of the following confusion matrices corresponds to the data and fitted model?



Matrix 1
Matrix 2
Matrix 3
True label
Predicted label

Matrix 1

True label \ Predicted label	0	1
0	8	0
1	2	10

Matrix 2

True label \ Predicted label	0	1
0	10	2
1	0	8

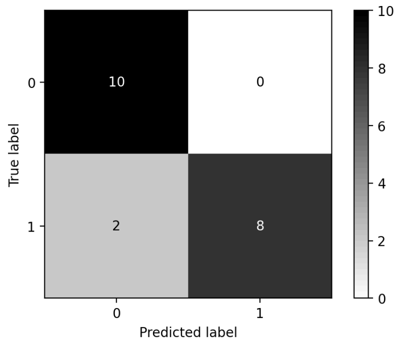
Matrix 3

True label \ Predicted label	0	1
0	8	2
1	0	10

☐

Problem 3 (2 points)

Provided the following confusion matrix, compute the TP, TN, FP, FN, recall, precision, and f1 score, where we use the convention that 1 is the positive outcome.



(Multiple choice, choose one)

1. 8, 10, 2, 0, 1.0, 0.833, 0.909
2. 8, 10, 0, 2, 1.0, 0.833, 0.909
3. 8, 10, 2, 0, 0.8, 1.0, 0.889
4. 8, 10, 0, 2, 0.8, 1.0, 0.899

ANSWERS:

PROBLEM 1 :-

Given , $y = [-4, 8, 7, -15, 12]$

$\hat{y} = [2, 9, -1, -16, 18]$

(i) MAE [Mean Absolute Error] :-

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |e_i|$$

$$\boxed{MAE = \frac{6 + 1 + 8 + 1 + 6}{5} = \frac{22}{5} = 4.4}$$

(ii) MSE [Mean Squared Error] :-

$$MSE = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

$$\boxed{MSE = \frac{36 + 1 + 64 + 1 + 36}{5} = \frac{138}{5} = 27.6}$$

(iii) MAPE [Mean Absolute Percentage Error] :-

$$MAPE = \left(\frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \right)$$

$$MAPE = \frac{1}{5} \left(\frac{6}{4} + \frac{1}{8} + \frac{8}{7} + \frac{1}{15} + \frac{6}{12} \right)$$

$$\boxed{MAPE = 0.667}$$

PROBLEM 2 :-

$TP = 10$

$TN = 8$

$FP = 2$

$FN = 0$

TN	FP
FN	TP

8	2
0	10

$\Longrightarrow$ MATRIX-3

PROBLEM 3:-

TP, TN, FP, FN, recall, precision and F1 score.

$TP = 8$

$TN = 10$

$FP = 0$

$FN = 2$

TN	FP
FN	TP

$$Recall = \frac{TP}{TP+FN} = 0.8$$

$$Precision = \frac{TP}{TP+FP} = 1$$

$$F1\ Score = \frac{2\,(Precision \times Recall)}{(Precision + Recall)} = 0.889$$

Answer : OPTION 4 : 8, 10, 2, 0, 0.8, 1.0, 0.889

M9-L1 Problem 1

Here, you will implement three loss functions from scratch in numpy: MAE, MSE, and MAPE.

In [5]:
```
import numpy as np

y_gt1 = np.array([1,2,3,4,5,6,7,8,9,10])
y_pred1 = np.array([1,1.3,3.1,4.6,5.9,5.9,6.4,9.2,8.1,10.5])

y_gt2 = np.array([-3.23594765, -3.74125693, -2.3040903 ,  0.        ,  0.30190142, -1.68434859,  1.10160357,  0.85874
y_pred2 = np.array([-3.17886560e+00, -3.72628642e+00, -2.28154027e+00, -2.42424242e-06, 2.96261368e-01, -1.70080838e-
```

Mean Absolute Error

Complete the definition for `MAE(y_gt, y_pred)` below.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| = \frac{1}{n}\sum_{i=1}^{n}|e_i|$$

`MAE(y_gt1, y_pred1)` should return 0.560.

In [6]:
```
def MAE(y_gt, y_pred):
    # YOUR CODE GOES HERE
    return np.mean(np.abs(y_gt-y_pred))

print(f"MAE(y_gt1, y_pred1) = {MAE(y_gt1, y_pred1):.3f}")
print(f"MAE(y_gt2, y_pred2) = {MAE(y_gt2, y_pred2):.3f}")
```

```
MAE(y_gt1, y_pred1) = 0.560
MAE(y_gt2, y_pred2) = 0.290
```

Mean Squared Error

Complete the definition for `MSE(y_gt, y_pred)` below.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 = \frac{1}{n}\sum_{i=1}^{n}(e_i)^2 = \frac{1}{n}e^Te$$

`MSE(y_gt1, y_pred1)` should return 0.454.

In [7]:
```
def MSE(y_gt, y_pred):
    # YOUR CODE GOES HERE
     return np.mean((y_gt-y_pred)**2)

print(f"MSE(y_gt1, y_pred1) = {MSE(y_gt1, y_pred1):.3f}")
print(f"MSE(y_gt2, y_pred2) = {MSE(y_gt2, y_pred2):.3f}")
```

```
MSE(y_gt1, y_pred1) = 0.454
MSE(y_gt2, y_pred2) = 0.174
```

Mean Absolute Percentage Error

Complete the definition for `MAPE(y_gt, y_pred, epsilon)` below.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - \hat{y}_i|}{|y_i| + \varepsilon} = \frac{1}{n}\sum_{i=1}^{n}\frac{|e_i|}{|y_i| + \varepsilon}$$

`MAPE(y_gt1, y_pred1, 1e-6)` should return 0.112.

In [9]:
```
def MAPE(y_gt, y_pred, epsilon=1e-6):
    # YOUR CODE GOES HERE
    return np.mean(np.abs((y_gt-y_pred)/(y_gt+epsilon)))

print(f"MAPE(y_gt1, y_pred1) = {MAPE(y_gt1, y_pred1):.3f}")
print(f"MAPE(y_gt2, y_pred2) = {MAPE(y_gt2, y_pred2):.3f}")
```

```
MAPE(y_gt1, y_pred1) = 0.112
MAPE(y_gt2, y_pred2) = 0.032
```

M9-L1 Problem 2

Recall the von Mises stress prediction problem from the module 6 homework. In this problem, you will compute the R^2 score for a few model predictions for a single shape in this dataset. You will also plot the predicted-vs-actual stress for each model.

In [9]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

float32 = np.float32
```

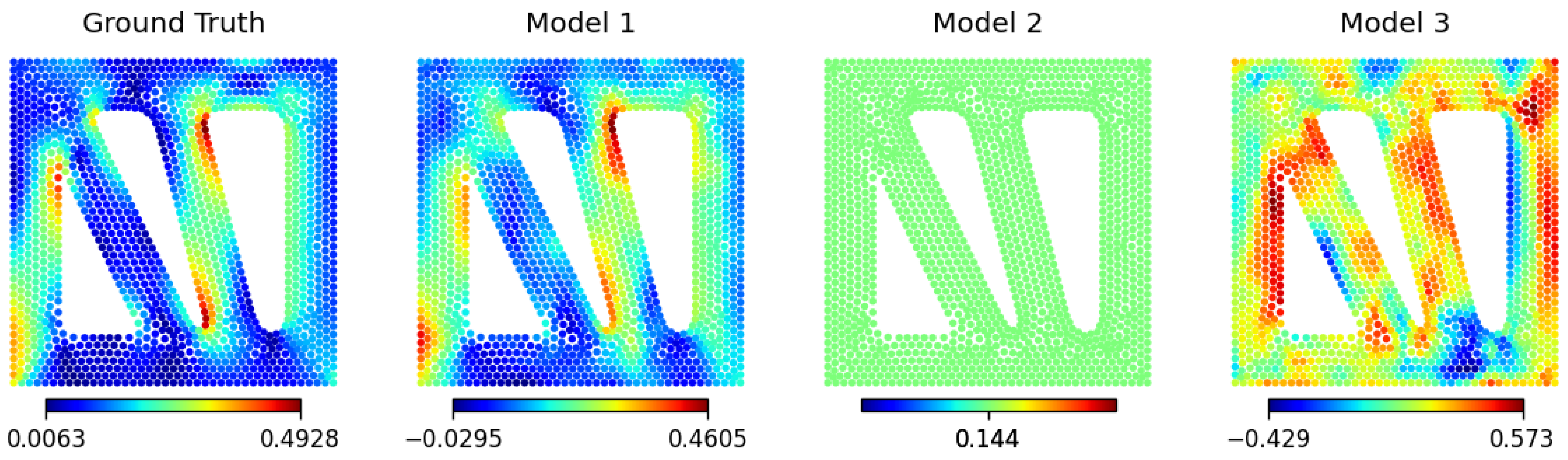
In [10]:
```
xs = np.array([0.        , 1.        , 1.        , 0.84258664, 0.8588491 , 0.86      , 0.86      , 0.859758  , 0.855
ys = np.array([1.        , 1.        , 0.        , 0.1574128 , 0.19719249, 0.24159406, 0.73021555, 0.7746335 , 0.818
gt = np.array([0.08839864, 0.08831116, 0.19658579, 0.12091552, 0.22751924, 0.2393936 , 0.26098105, 0.2698702 , 0.230
model1 = np.array([ 1.11028813e-01,  1.22077152e-01,  1.65442377e-01,  1.55162349e-01, 2.34626681e-01,  2.43337303e-
model2 = np.array([0.14395204, 0.14395204, 0.14395204, 0.14395204, 0.14395204, 0.14395204, 0.14395204, 0.14395204, 0
model3 = np.array([ 2.81959862e-01,  4.73277241e-01,  2.35924363e-01,  6.43386170e-02, 4.48577106e-04,  4.03367728e-
```

Visualizing data

Run the following cell to load the data and visualize the 3 model predictions.

- `gt` is the ground truth von Mises stress vector
- `model1` is the vector of stress predictions for model 1
- `model2` is the vector of stress predictions for model 2
- `model3` is the vector of stress predictions for model 3

In [11]:
```
def plot_shape(x, y, stress, lims=None):

    if lims is None:
        lims = [min(stress),max(stress)]

    plt.scatter(x,y,s=5,c=stress,cmap="jet",vmin=lims[0],vmax=lims[1])
    plt.colorbar(orientation="horizontal", shrink=.75, pad=0,ticks=lims)
    plt.axis("off")
    plt.axis("equal")

def plot_all(x, y, gt, model1, model2, model3):
    plt.figure(figsize=[12,3.2], dpi=120)
    plt.subplot(141)
    plot_shape(x, y, gt)
    plt.title("Ground Truth")

    plt.subplot(142)
    plot_shape(x, y, model1)
    plt.title("Model 1")

    plt.subplot(143)
    plot_shape(x, y, model2)
    plt.title("Model 2")

    plt.subplot(144)
    plot_shape(x, y, model3)
    plt.title("Model 3")

    plt.show()

plot_all(xs, ys, gt, model1, model2, model3)
```



Computing R^2

Calculate the R^2 value for each model and print the results.

In [12]:
```
# YOUR CODE GOES HERE
r2_model1 = r2_score(gt, model1)
r2_model2 = r2_score(gt, model2)
r2_model3 = r2_score(gt, model3)

print(f"The R^2 Score for Model 1 is = {r2_model1}")
print(f"The R^2 Score for Model 2 is = {r2_model2}")
print(f"The R^2 Score for Model 3 is = {r2_model3}")
```

```
The R^2 Score for Model 1 is = 0.8727994044645364
The R^2 Score for Model 2 is = 0.0
The R^2 Score for Model 3 is = -3.0451391287323784
```

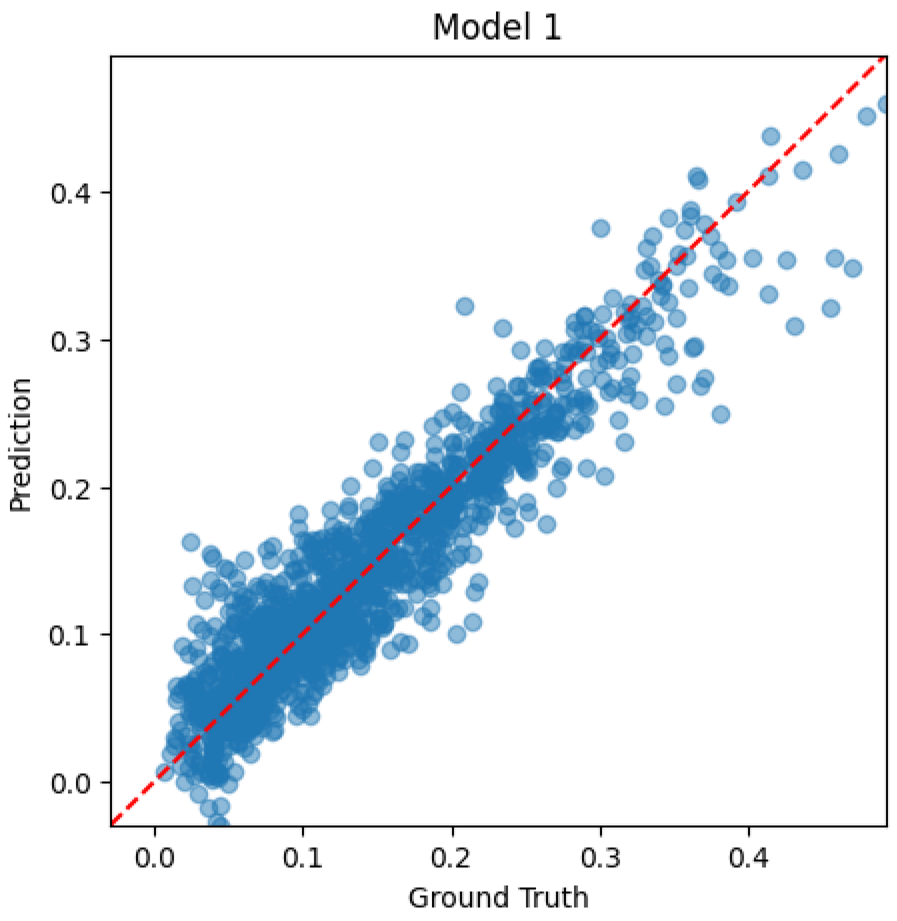
Plotting predictions vs ground truth

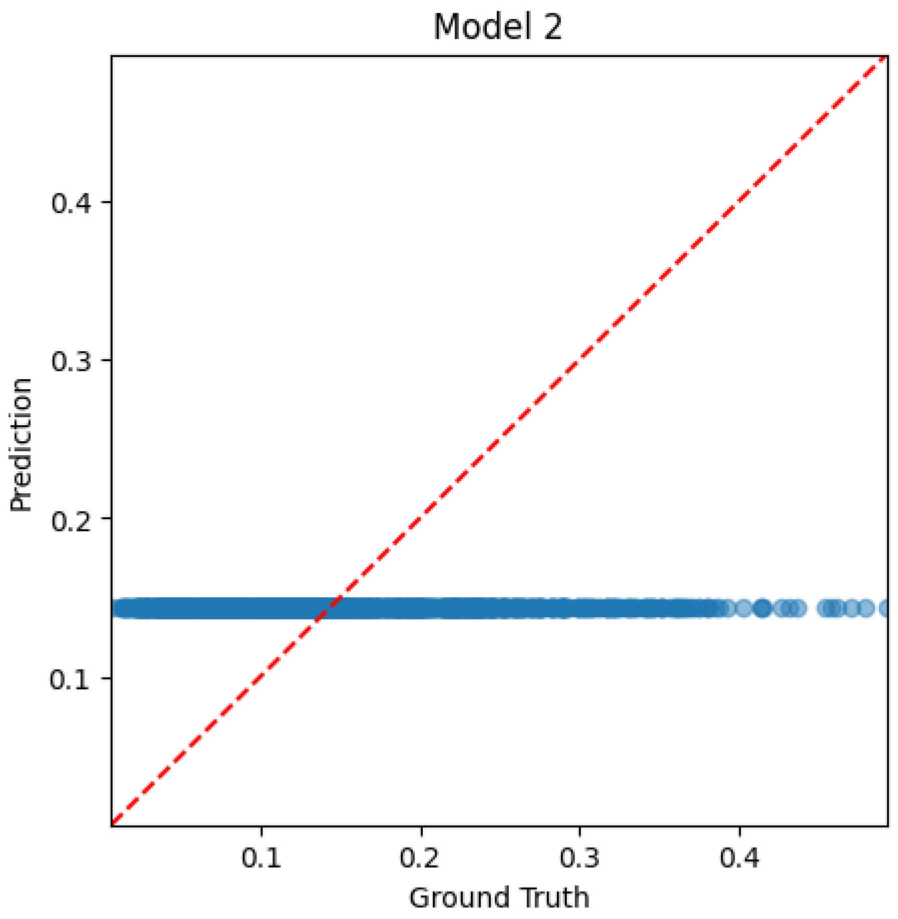
Complete the function definition below for `plot_r2(gt, pred, title)`

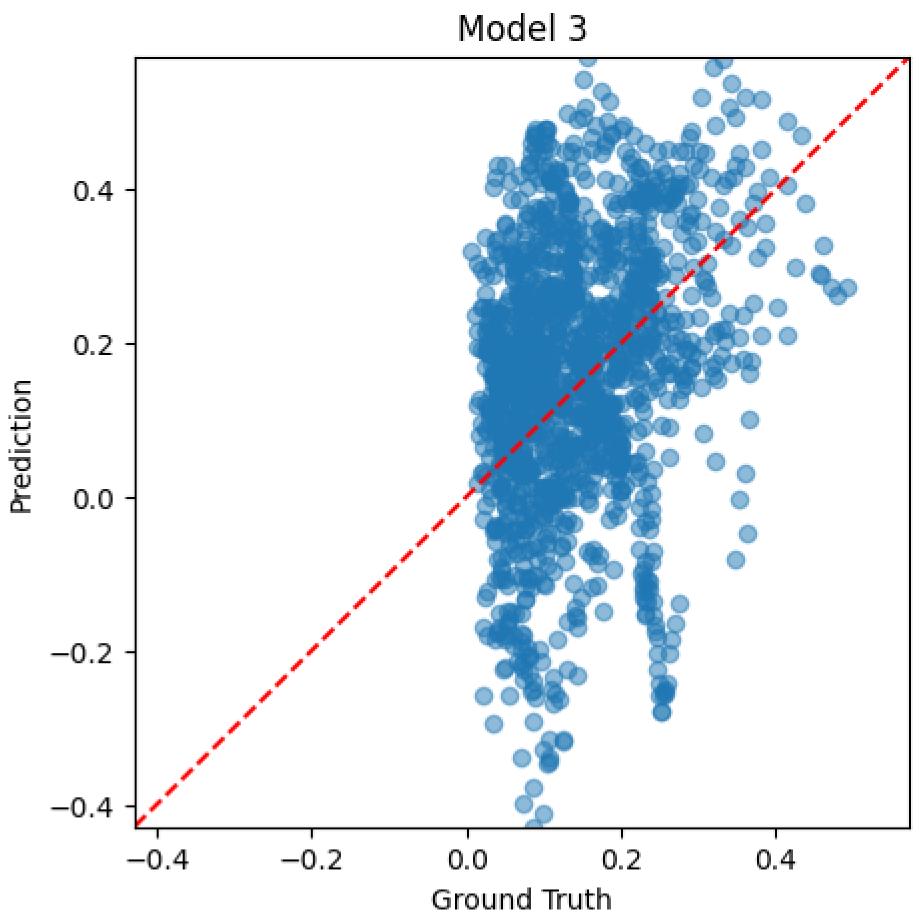
Then create plots for all 3 models.

In [15]:
```
def plot_r2(gt, pred, title):
    plt.figure(figsize=[5,5])

    # YOUR CODE GOES HERE
    plt.scatter(gt, pred, alpha=0.5)


    plt.plot([-1000,1000], [-1000,1000],"r--")

    all = np.concatenate([gt, pred])
    plt.xlim(np.min(all), np.max(all))
    plt.ylim(np.min(all), np.max(all))
    plt.xlabel("Ground Truth")
    plt.ylabel("Prediction")
    plt.title(title)
    plt.show()

plot_r2(gt, model1,"Model 1")
```

```
plot_r2(gt, model2,"Model 2")
plot_r2(gt, model3,"Model 3")
```




Model 2
Prediction
Ground Truth
0.1
0.2
0.3
0.4
0.1
0.2
0.3
0.4



Questions

1. Model 2 has an R^2 of exactly 0. Why?
2. Model 3 has an R^2 less than 0. What does this mean?

In [16]:
```
print("1) Model 2 has an R^2 score of exactly 0 which indicates that the model's predictions do not at all account fo
print("2) For Model 3, an R^2 score less than 0 suggests that the model is performing worse than a simple model that
```

```
1) Model 2 has an R^2 score of exactly 0 which indicates that the model's predictions do not at all account for the v
ariance in the ground truth data. This happens when a model simply predicts the mean of the target variable for all i
nputs, without actually learning from the features.

2) For Model 3, an R^2 score less than 0 suggests that the model is performing worse than a simple model that would a
lways predict the mean of the target variable. Since the ground truth values are not negative and the model is produc
ing negative predictions, this indicates that the model's predictions are systematically incorrect. This discrepancy
between the prediction range and the actual values contributes significantly to the poor R^2 score.
```

Problem 1:

Once again consider the plane-strain compression problem shown in "data/plane-strain.png". In this problem you are given node features for 100 parts. These node features have been extracted by processing each part shape using a neural network. You will train a neural network to von Mises stress at each node given its 60 features. Then you will analyze R^2 for the training and testing data, both for the full dataset and for individual shapes within each dataset.

Summary of deliverables

- Neural network model definition
- Training function
- Training loss curve
- Overall R^2 on training and testing data
- Predicted-vs-actual plots for training and testing data
- Histograms of R^2 distributions on training and testing shapes
- Median R^2 values across training and testing shapes

In [29]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import r2_score

import torch
from torch import nn, optim

def plot_shape(dataset, index, model=None, lims=None):
    x = dataset["coordinates"][index][:,0]
    y = dataset["coordinates"][index][:,1]

    if model is None:
        c = dataset["stress"][index]
    else:
        c = model(torch.tensor(dataset["features"][index])).detach().numpy().flatten()

    if lims is None:
        lims = [min(c),max(c)]
    plt.scatter(x,y,s=5,c=c,cmap="jet",vmin=lims[0],vmax=lims[1])
    plt.colorbar(orientation="horizontal", shrink=.75, pad=0,ticks=lims)
    plt.axis("off")
    plt.axis("equal")

def plot_shape_comparison(dataset, index, model, title=""):
    plt.figure(figsize=[6,3.2], dpi=120)
    plt.subplot(1,2,1)
    plot_shape(dataset,index)
    plt.title("Ground Truth",fontsize=9,y=.96)
    plt.subplot(1,2,2)
    c = dataset["stress"][index]
    plot_shape(dataset, index, model, lims = [min(c), max(c)])
    plt.title("Prediction",fontsize=9,y=.96)
    plt.suptitle(title)
    plt.show()

def load_dataset(path):
    dataset = np.load(path)
    coordinates = []
    features = []
    stress = []
    N = np.max(dataset[:,0].astype(int)) + 1
    split = int(N*.8)
    for i in range(N):
        idx = dataset[:,0].astype(int) == i
        data = dataset[idx,:]
        coordinates.append(data[:,1:3])
        features.append(data[:,3:-1])
        stress.append(data[:,-1])
    dataset_train = dict(coordinates=coordinates[:split], features=features[:split], stress=stress[:split])
    dataset_test = dict(coordinates=coordinates[split:], features=features[split:], stress=stress[split:])
    X_train, X_test = np.concatenate(features[:split], axis=0), np.concatenate(features[split:], axis=0)
    y_train, y_test = np.concatenate(stress[:split], axis=0), np.concatenate(stress[split:], axis=0)
    return dataset_train, dataset_test, X_train, X_test, y_train, y_test

def get_shape(dataset,index):
    X = torch.tensor(dataset["features"][index])
    Y = torch.tensor(dataset["stress"][index].reshape(-1,1))
    return X, Y
```

```
def plot_r2_distribution(r2s, title=""):
    plt.figure(dpi=120,figsize=(6,4))
    plt.hist(r2s, bins=10)
    plt.xlabel("$R^2$")
    plt.ylabel("Number of shapes")
    plt.title(title)
    plt.show()
```

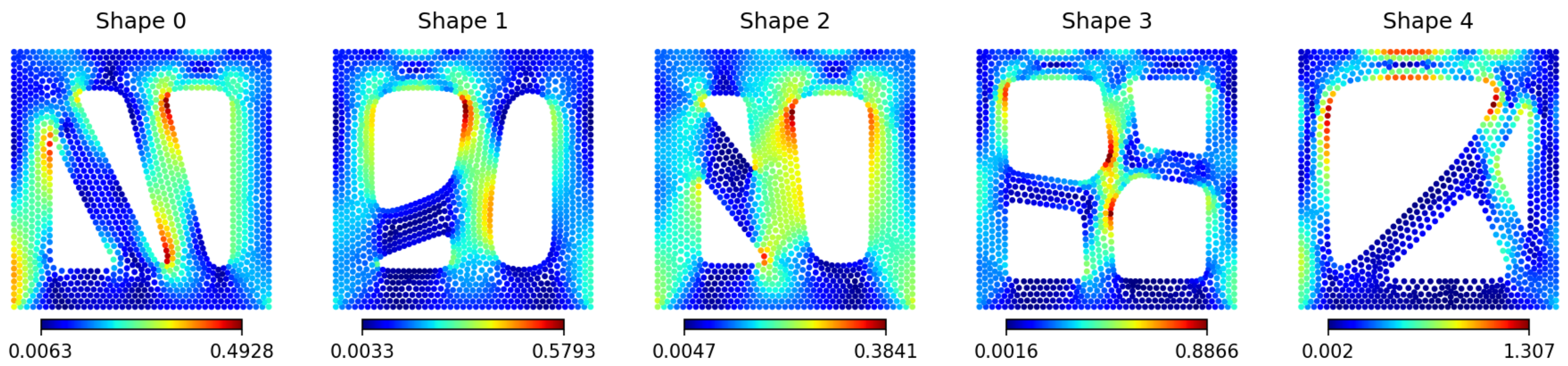
Loading the data

First, complete the code below to load the data and plot the von Mises stress fields for a few shapes.
You'll need to input the path of the data file, the rest is done for you.

All training node features and outputs are in `X_train` and `y_train`, respectively. Testing nodes are in `X_test`, `y_test`.

`dataset_train` and `dataset_test` contain more detailed information such as node coordinates, and they are separated by shape.
Get features and outputs for a shape by calling `get_shape(dataset,index)`. `N_train` and `N_test` are the number of training and testing shapes in each of these datasets.

In [30]:
```
data_path = "C:\\Users\\srech\\Downloads\\stress_nodal_features.npy"
dataset_train, dataset_test, X_train, X_test, y_train, y_test = load_dataset(data_path)
N_train = len(dataset_train["stress"])
N_test = len(dataset_test["stress"])

plt.figure(figsize=[15,3.2], dpi=150)
for i in range(5):
    plt.subplot(1,5,i+1)
    plot_shape(dataset_train,i)
    plt.title(f"Shape {i}")
plt.show()
```



Neural network to predict stress

Create a PyTorch neural network class `StressPredictor` below. This should be an MLP with 60 inputs (the given features) and 1 output (stress). The hidden layer sizes and activations are up to you.

In [32]:
```
class StressPredictor(nn.Module):
    # YOUR CODE GOES HERE
    def __init__(self):
        super(StressPredictor, self).__init__()
        self.network = nn.Sequential(
            nn.Linear(60, 120),
            nn.ReLU(),
            nn.Linear(120, 60),
            nn.ReLU(),
            nn.Linear(60, 1))

    def forward(self, x):
        return self.network(x)
```

Training function

Below, you should define a function `train(model, dataset, lr, epochs)` that will train `model` on the data in `dataset` with the Adam optimizer for `epochs` epochs with a learning rate of `lr`.

Because there are so many total nodes, you should treat each shape as a batch of nodes -- each epoch of training will require you to loop through each shape in the dataset in a random order, performing a step of gradient descent for each shape encountered. Your function should automatically generate a plot of the loss curve on training data.

- You can use the provided `get_shape` to access feature and output tensors for each shape.
- Use MSE as a your loss function.
- Look into `np.random.permutation()` for generating a random index order

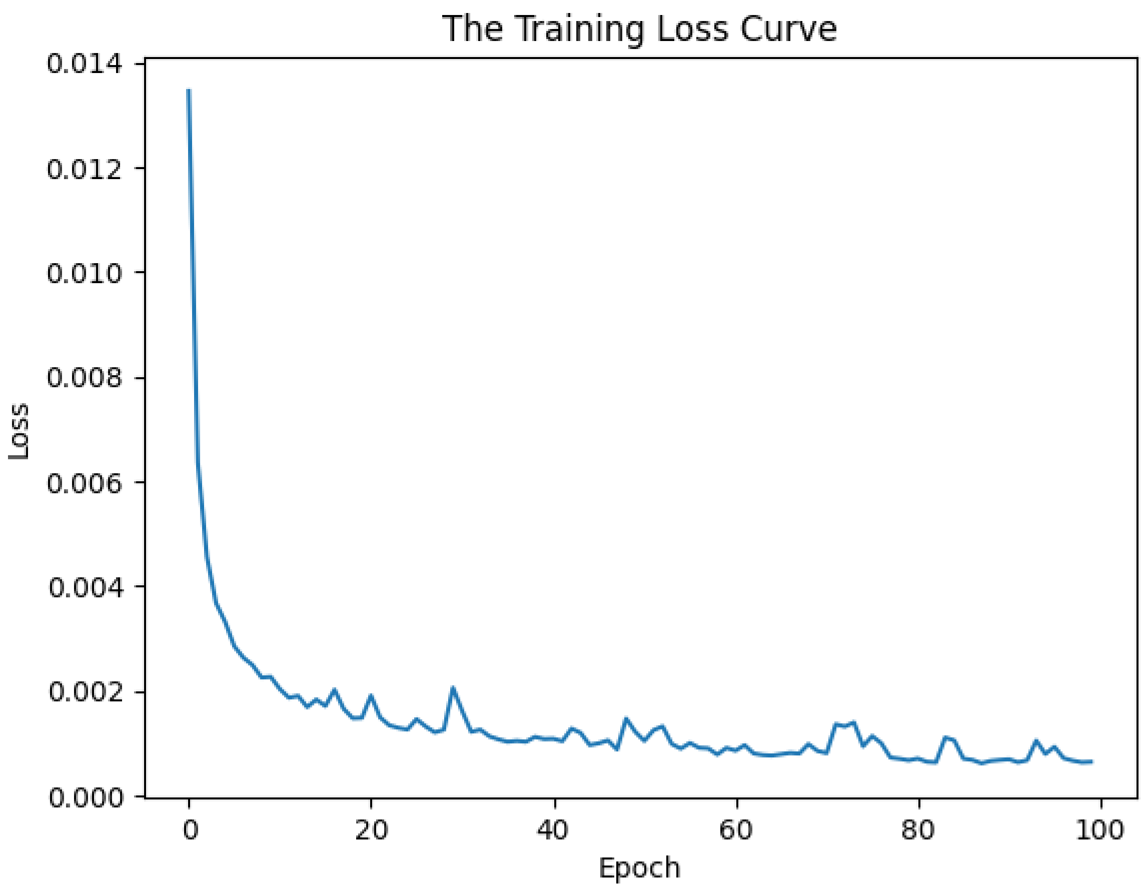
In [33]:
```python
def train(model, dataset, lr, epochs):
    # YOUR CODE GOES HERE
    opt = optim.Adam(model.parameters(), lr=lr)
    cr = nn.MSELoss()
    l = []

    for e in range(epochs):
        epoch_loss = 0.0
        o = np.random.permutation(len(dataset['features']))

        for i in o:
            f, t = get_shape(dataset,i)
            opt.zero_grad()
            ops = model(f.float())
            loss = cr(ops, t.float())
            loss.backward()
            opt.step()
            epoch_loss += loss.item()

        avg_loss = epoch_loss/len(o)
        print(f'Epoch [{e+1}/{epochs}], Loss: {avg_loss:.4f}')
        l.append(avg_loss)

    plt.plot(l)
    plt.xlabel('Epoch')
    plt.ylabel('Loss')
    plt.title('The Training Loss Curve')
    plt.show()
```

Training your Neural Network

Now, create your neural network model and run your train function on the training dataset `dataset_train`. Determining the right number of epochs and learning rate are up to you. The training loss curve should be shown.

In [34]:
```
# YOUR CODE GOES HERE
model = StressPredictor()
train(model, dataset_train, lr=0.001, epochs=100)
```

```
Epoch [1/100], Loss: 0.0135
Epoch [2/100], Loss: 0.0064
Epoch [3/100], Loss: 0.0046
Epoch [4/100], Loss: 0.0037
Epoch [5/100], Loss: 0.0033
Epoch [6/100], Loss: 0.0029
Epoch [7/100], Loss: 0.0026
Epoch [8/100], Loss: 0.0025
Epoch [9/100], Loss: 0.0023
Epoch [10/100], Loss: 0.0023
Epoch [11/100], Loss: 0.0020
Epoch [12/100], Loss: 0.0019
Epoch [13/100], Loss: 0.0019
Epoch [14/100], Loss: 0.0017
Epoch [15/100], Loss: 0.0018
Epoch [16/100], Loss: 0.0017
Epoch [17/100], Loss: 0.0020
Epoch [18/100], Loss: 0.0017
Epoch [19/100], Loss: 0.0015
Epoch [20/100], Loss: 0.0015
Epoch [21/100], Loss: 0.0019
Epoch [22/100], Loss: 0.0015
Epoch [23/100], Loss: 0.0013
Epoch [24/100], Loss: 0.0013
Epoch [25/100], Loss: 0.0013
Epoch [26/100], Loss: 0.0015
Epoch [27/100], Loss: 0.0013
Epoch [28/100], Loss: 0.0012
Epoch [29/100], Loss: 0.0013
Epoch [30/100], Loss: 0.0021
Epoch [31/100], Loss: 0.0016
Epoch [32/100], Loss: 0.0012
Epoch [33/100], Loss: 0.0013
Epoch [34/100], Loss: 0.0011
Epoch [35/100], Loss: 0.0011
Epoch [36/100], Loss: 0.0010
Epoch [37/100], Loss: 0.0010
Epoch [38/100], Loss: 0.0010
Epoch [39/100], Loss: 0.0011
Epoch [40/100], Loss: 0.0011
Epoch [41/100], Loss: 0.0011
Epoch [42/100], Loss: 0.0010
Epoch [43/100], Loss: 0.0013
Epoch [44/100], Loss: 0.0012
Epoch [45/100], Loss: 0.0010
Epoch [46/100], Loss: 0.0010
Epoch [47/100], Loss: 0.0011
Epoch [48/100], Loss: 0.0009
Epoch [49/100], Loss: 0.0015
Epoch [50/100], Loss: 0.0012
Epoch [51/100], Loss: 0.0010
Epoch [52/100], Loss: 0.0012
Epoch [53/100], Loss: 0.0013
Epoch [54/100], Loss: 0.0010
Epoch [55/100], Loss: 0.0009
Epoch [56/100], Loss: 0.0010
Epoch [57/100], Loss: 0.0009
Epoch [58/100], Loss: 0.0009
Epoch [59/100], Loss: 0.0008
Epoch [60/100], Loss: 0.0009
Epoch [61/100], Loss: 0.0009
Epoch [62/100], Loss: 0.0010
Epoch [63/100], Loss: 0.0008
Epoch [64/100], Loss: 0.0008
Epoch [65/100], Loss: 0.0008
Epoch [66/100], Loss: 0.0008
Epoch [67/100], Loss: 0.0008
Epoch [68/100], Loss: 0.0008
Epoch [69/100], Loss: 0.0010
Epoch [70/100], Loss: 0.0009
Epoch [71/100], Loss: 0.0008
Epoch [72/100], Loss: 0.0014
Epoch [73/100], Loss: 0.0013
Epoch [74/100], Loss: 0.0014
Epoch [75/100], Loss: 0.0009
Epoch [76/100], Loss: 0.0011
Epoch [77/100], Loss: 0.0010
Epoch [78/100], Loss: 0.0007
Epoch [79/100], Loss: 0.0007
Epoch [80/100], Loss: 0.0007
Epoch [81/100], Loss: 0.0007
Epoch [82/100], Loss: 0.0006
Epoch [83/100], Loss: 0.0006
Epoch [84/100], Loss: 0.0011
Epoch [85/100], Loss: 0.0011
Epoch [86/100], Loss: 0.0007
Epoch [87/100], Loss: 0.0007
Epoch [88/100], Loss: 0.0006
Epoch [89/100], Loss: 0.0007
Epoch [90/100], Loss: 0.0007
Epoch [91/100], Loss: 0.0007
Epoch [92/100], Loss: 0.0006
Epoch [93/100], Loss: 0.0007
Epoch [94/100], Loss: 0.0011
Epoch [95/100], Loss: 0.0008
Epoch [96/100], Loss: 0.0009
Epoch [97/100], Loss: 0.0007
Epoch [98/100], Loss: 0.0007
Epoch [99/100], Loss: 0.0006
Epoch [100/100], Loss: 0.0006
```



R^2 Score

Compute the R^2 Score on the training dataset. You will have to convert between tensors and arrays versions to use sklearn functions, or you can write your own function.

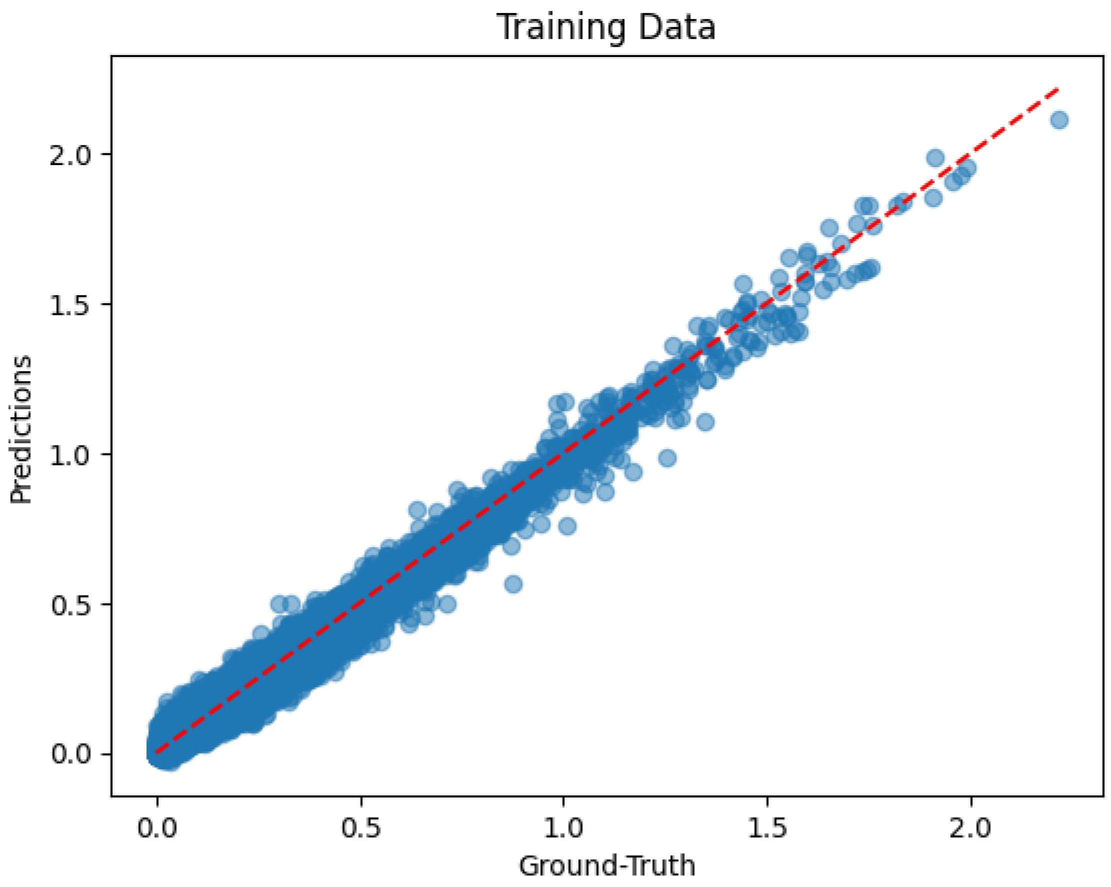
In [35]:
```
# YOUR CODE GOES HERE
from sklearn.metrics import r2_score
def compute_r2(model, dataset):
    pd = []
    g_t = []
    n = len(dataset['features'])
    for i in range(n):
        X, Y = get_shape(dataset, i)
        preds = model(X.float()).detach().numpy().flatten()
        pd.extend(preds)
        g_t.extend(Y.numpy().flatten())

    return r2_score(g_t, pd)

r2_train = compute_r2(model, dataset_train)
r2_test = compute_r2(model, dataset_test)
print(f'The R^2 on Training Data is = {r2_train}')
print(f'The R^2 on Testing Data is = {r2_test}')
```

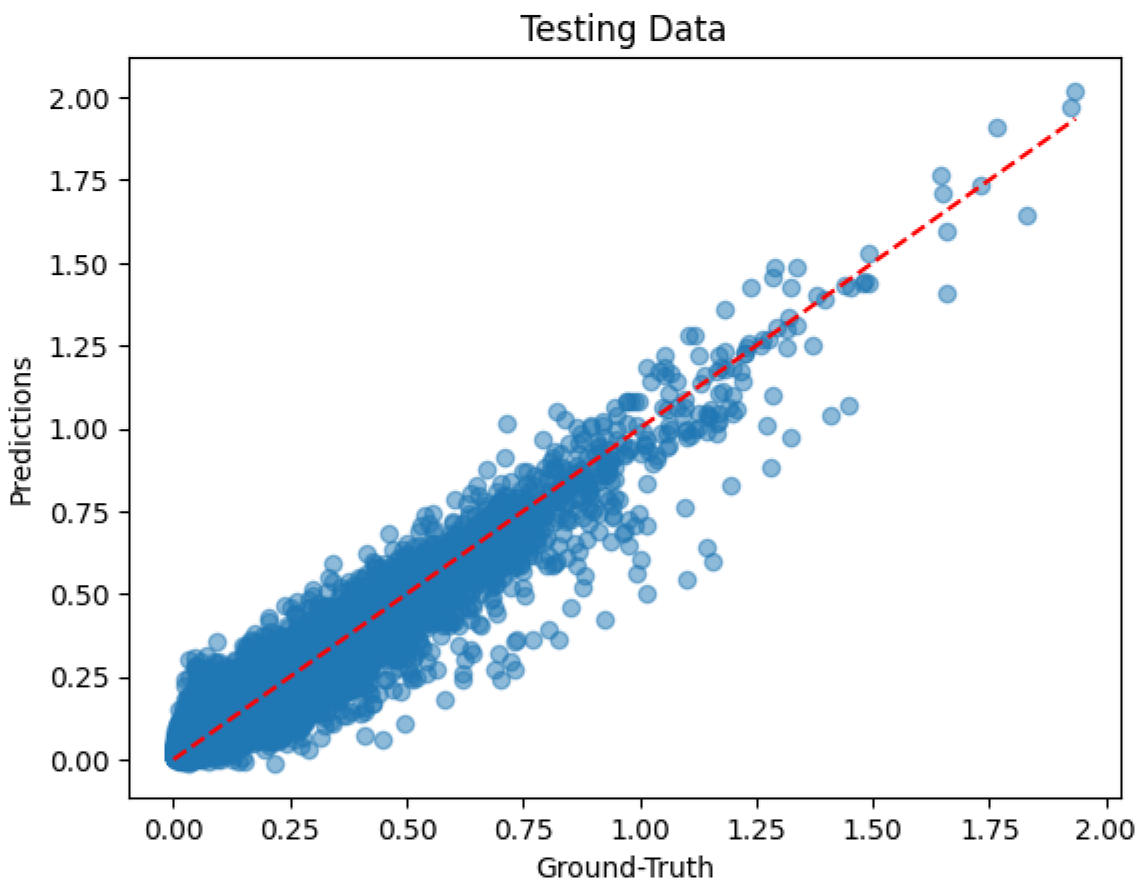
```
The R^2 on Training Data is = 0.9719782008857277
The R^2 on Testing Data is = 0.9213617267361874
```

R^2 Plots

Now, generate predicted-vs-actual plots that display both data and a theoretical best fit line. Make 2 such plots - one for training data and one for testing.

In [36]:
```
# YOUR CODE GOES HERE
def plot_r2(gt, pred, title=""):
    plt.scatter(gt, pred, alpha=0.5)
    plt.plot([min(gt), max(gt)], [min(gt), max(gt)], 'r--')
    plt.xlabel("Ground-Truth")
    plt.ylabel("Predictions")
    plt.title(title)
    plt.show()

plot_r2(y_train, model(torch.tensor(X_train).float()).detach().numpy(), title="Training Data")
plot_r2(y_test, model(torch.tensor(X_test).float()).detach().numpy(), title="Testing Data")
```


Training Data
Predictions
Ground-Truth
0.0
0.5
1.0
1.5
2.0



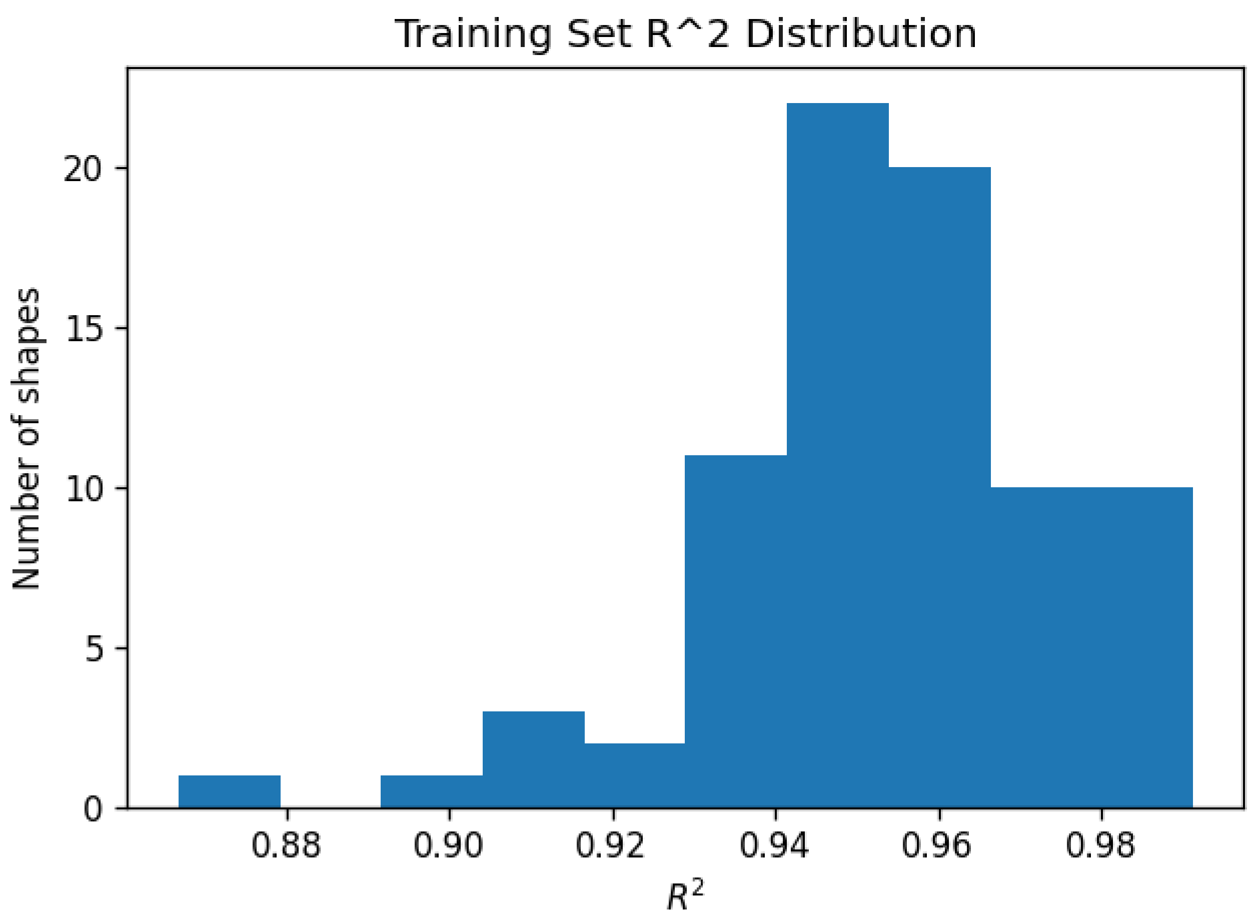
Individual Shape R^2

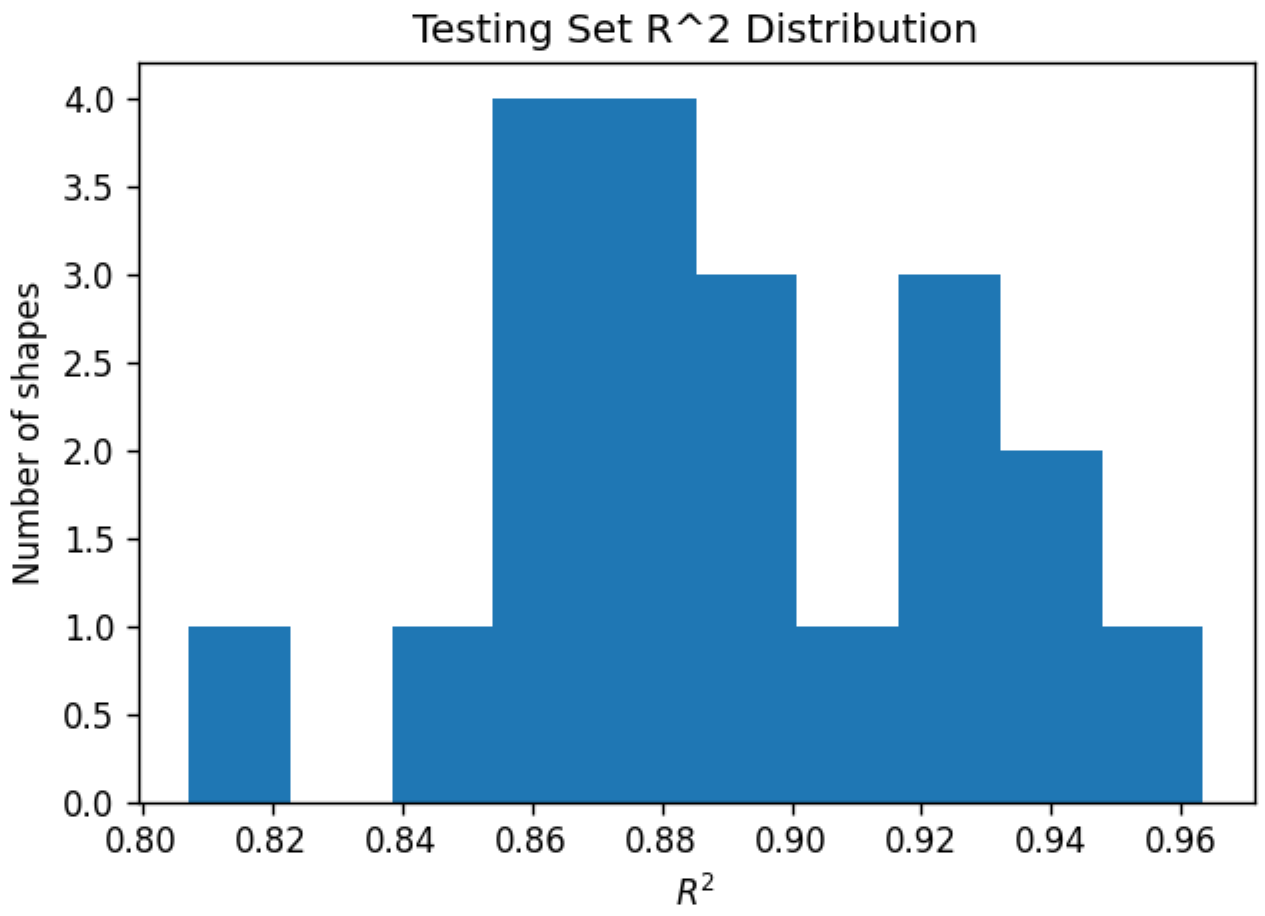
Because we have a unique problem where groups of nodes in a dataset form a single shape, we can compute an R^2 score for an individual shape. For each shape in the training set, compute an R^2 score. Then create a histogram of the values with the function `plot_r2_hist(r2s)`. Repeat for the testing set.

Report the median R^2 score across all training shapes, and the median across all testing shapes.

If your test median is below 0.85, try and tune your network size/training hyperparameters until it reaches this threshold.

In [37]:
```
# YOUR CODE GOES HERE
def compute_individual_r2(model, dataset):
    r2_scores = []

    for i in range(len(dataset['features'])):
        X, Y = get_shape(dataset, i)
        preds = model(X.float()).detach().numpy()
        r2_scores.append(r2_score(Y.numpy(), preds))

    return r2_scores

r2_scores_train = compute_individual_r2(model, dataset_train)
r2_scores_test = compute_individual_r2(model, dataset_test)
plot_r2_distribution(r2_scores_train, title="Training Set R^2 Distribution")
plot_r2_distribution(r2_scores_test, title="Testing Set R^2 Distribution")
print(f'Median R^2 for Training Shapes: {np.median(r2_scores_train)}')
print(f'Median R^2 for Testing Shapes: {np.median(r2_scores_test)}')
```


Training Set R^2 Distribution
Number of shapes
0
5
10
15
20
0.88
0.90
0.92
0.94
0.96
0.98
R^2



```
Median R^2 for Training Shapes: 0.9540753130663849
Median R^2 for Testing Shapes: 0.8893747615993984
```