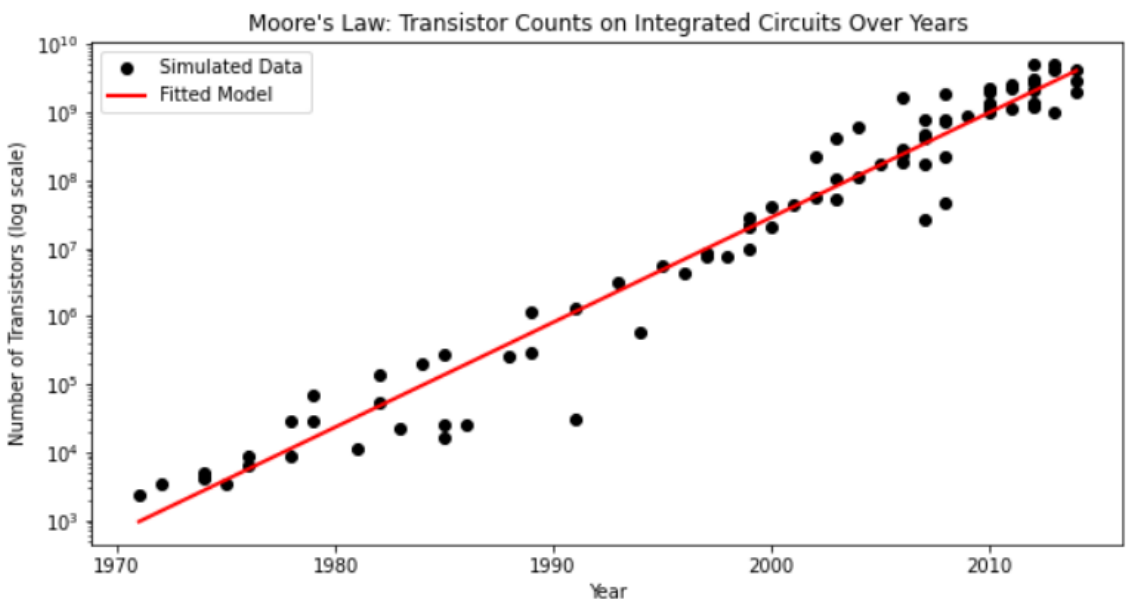
```python
# 10.4
import pandas as pd

csvname = 'transistor_counts.csv'
data = np.asarray(pd.read_csv(csvname,header = None))
x = data[:,0].reshape(-1, 1)
x.shape = (len(x),1)
y = data[:,1].reshape(-1, 1)
y.shape = (len(y),1)

# Logarithmic transformation of the y data
log_y = np.log(y)

# Set up the Least Squares problem
X_ls = np.hstack((np.ones_like(x), x))  # Add a column of ones for the intercept
weights_ls = np.linalg.inv(X_ls.T @ X_ls) @ X_ls.T @ log_y

# Fit the model to the transformed data
log_predicted_y = X_ls @ weights_ls

# Translate the model back to the original data space
predicted_y = np.exp(log_predicted_y)

# Plotting the results
plt.figure(figsize=(10, 5))
plt.scatter(x, y, label='Simulated Data', color='black')
plt.plot(x, predicted_y, label='Fitted Model', color='red', linewidth=2)
plt.yscale('log')  # Use a logarithmic scale for the y-axis to match the transformation
plt.xlabel('Year')
plt.ylabel('Number of Transistors (log scale)')
plt.title("Moore's Law: Transistor Counts on Integrated Circuits Over Years")
plt.legend()

plt.show()

# Output the weights for inspection
weights_ls
```



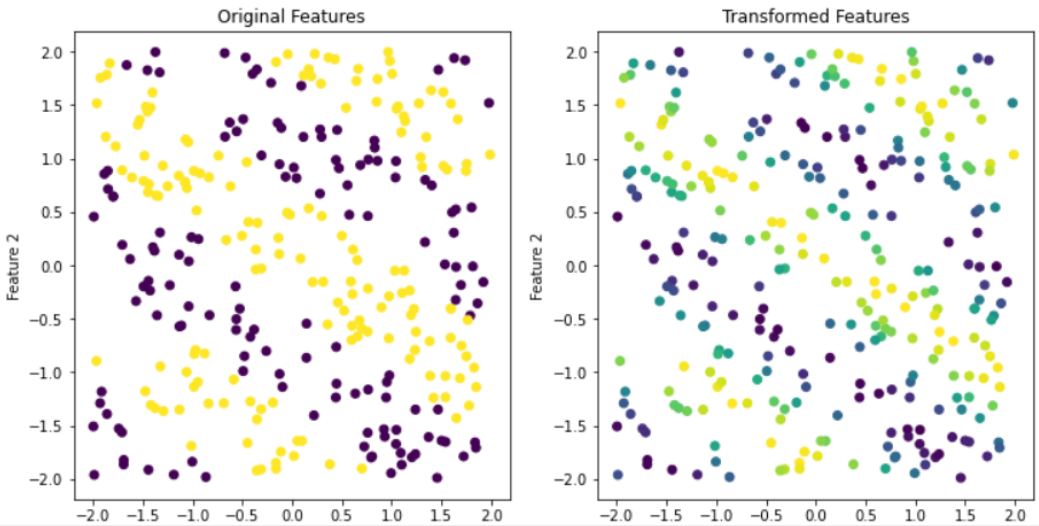

```
array([[-6.93553490e+02],
       [ 3.55358877e-01]])
```

```
import numpy as np
from autograd import grad
import autograd.numpy as anp

data = np.loadtxt('diagonal_stripes.csv', delimiter=',')

x = data[:2,:]
y = data[2,:]

def feature(x, w):
    sin_feature = anp.sin(w[0] * x[0, :] + w[1] * x[1, :] + w[2])
    return (w[3] + w[4] * sin_feature).T

def softmax_cost(w, x, y):
    predictions = feature(x, w)
    return anp.sum(anp.log(1 + anp.exp(-y * predictions))) / float(anp.size(y))

def gradient_descent(cost_func, x, y, initial_w, alpha, max_its):
    compute_gradient  = grad(cost_func)
    w = initial_w
    for iteration  in range(max_its):
        w -= alpha * compute_gradient (w, x, y)
    return w

def predict(x, w):
    predictions = feature(x, w)
    return anp.sign(predictions)

def calculate_misclassifications(x, y, w):
    predictions = predict(x, w)
    return np.sum(predictions != y)

# Initialize weights and run gradient descent
initial_w = np.array([1.0, 1.0, 1.0, 1.0, 1.0])
learning_rate = 0.1
iterations = 2000
```

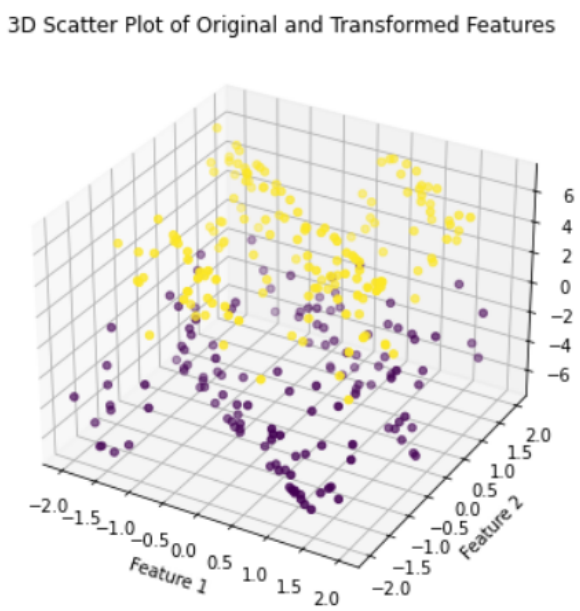
```
# Run gradient descent
optimized_w = gradient_descent(softmax_cost, x, y, initial_w, learning_rate, iterations)

# Predict using optimized weights
y_pred = predict(x, optimized_w)

# Calculate accuracy
accuracy = np.mean(y_pred == y) * 100

# Output the optimized weights and the accuracy
print(optimized_w, 'accuracy: ', accuracy)

misclassifications = calculate_misclassifications(x, y, optimized_w)
print('number of misclassification:', misclassifications)
```

```
[2.00088994e+00 3.00822368e+00 1.00407791e+00 1.61793061e-03
 6.77555078e+00] accuracy:  100.0
number of misclassification: 0
```

```
import matplotlib.pyplot as plt

transformed_features = feature(x, optimized_w)

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.scatter(x[0, :], x[1, :], c=y, cmap='viridis')
plt.title("Original Features")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")

plt.subplot(1, 2, 2)
plt.scatter(x[0, :], x[1, :], c=transformed_features, cmap='viridis')
plt.title("Transformed Features")
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")

plt.show()
```



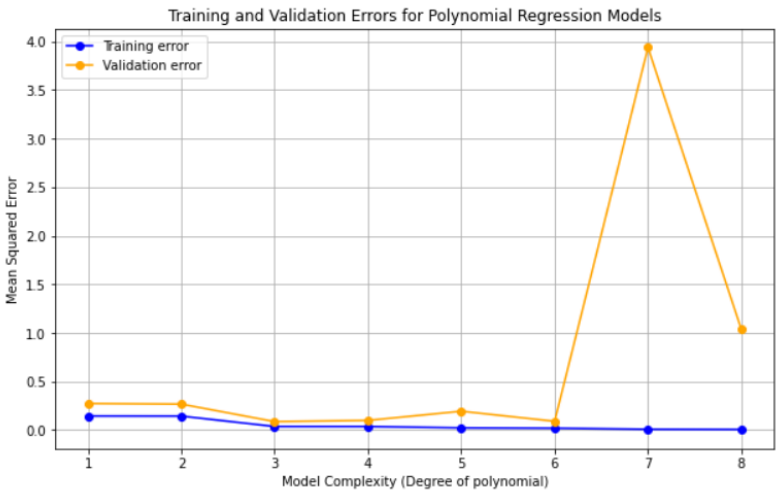

```
from mpl_toolkits.mplot3d import Axes3D

transformed_features = feature(x, optimized_w)

fig = plt.figure(figsize=(8, 6))
ax = fig.add_subplot(111, projection='3d')

scatter = ax.scatter(x[0, :], x[1, :], transformed_features, c=y, cmap='viridis')

ax.set_xlabel('Feature 1')
ax.set_ylabel('Feature 2')
ax.set_zlabel('Transformed Feature')
ax.set_title('3D Scatter Plot of Original and Transformed Features')

plt.show()
```



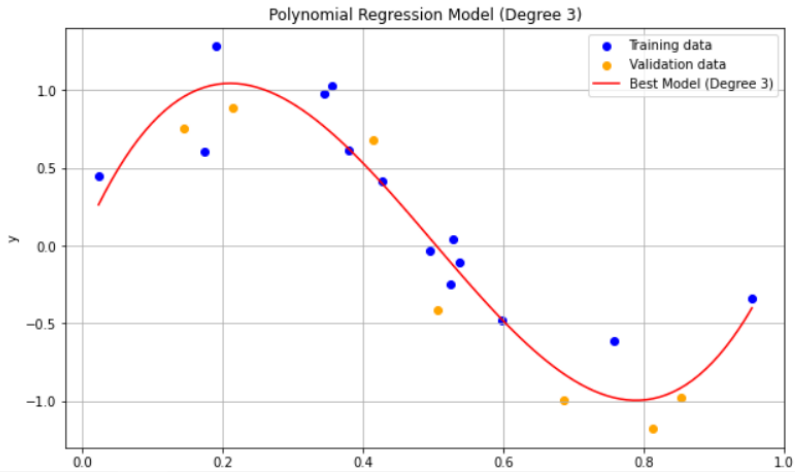
```python
# 11.1
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.preprocessing import PolynomialFeatures
from sklearn.linear_model import LinearRegression

# Load the dataset
csvname = 'noisy_sin_sample.csv'
data = np.loadtxt(csvname, delimiter=',')
x = data[:-1, :].T
y = data[-1:, :].T

# Print the shapes of the loaded data for confirmation
print(f'Shape of x: {np.shape(x)}')
print(f'Shape of y: {np.shape(y)}')

# Split the data into training and validation sets
x_train, x_val, y_train, y_val = train_test_split(x, y, test_size=1/3, random_state=42)

# Initialize arrays to store errors
train_errors = []
val_errors = []

# Test polynomial models of degrees 1 through 8
for degree in range(1, 9):
    # Create polynomial features
    poly = PolynomialFeatures(degree=degree)
    x_train_poly = poly.fit_transform(x_train)
    x_val_poly = poly.transform(x_val)

    # Fit the Linear Regression model
    model = LinearRegression()
    model.fit(x_train_poly, y_train)

    # Predict on training and validation set
    y_train_pred = model.predict(x_train_poly)
    y_val_pred = model.predict(x_val_poly)

    # Calculate and store the errors
    train_error = mean_squared_error(y_train, y_train_pred)
    val_error = mean_squared_error(y_val, y_val_pred)
    train_errors.append(train_error)
    val_errors.append(val_error)
```

```python
# Plot training and validation errors
plt.figure(figsize=(10, 6))
plt.plot(range(1, 9), train_errors, label='Training error', marker='o', color='blue')
plt.plot(range(1, 9), val_errors, label='Validation error', marker='o', color='orange')
plt.xlabel('Model Complexity (Degree of polynomial)')
plt.ylabel('Mean Squared Error')
plt.title('Training and Validation Errors for Polynomial Regression Models')
plt.legend()
plt.grid(True)
plt.show()

# Find the degree of polynomial with the lowest validation error
best_degree = np.argmin(val_errors) + 1  # +1 because degree count starts from 1
print(f'Best model degree: {best_degree} with validation error: {min(val_errors)}')

# Fit the model with the best degree on the entire dataset
poly = PolynomialFeatures(degree=best_degree)
x_poly = poly.fit_transform(x)
model = LinearRegression()
model.fit(x_poly, y)

# Plot the best model
x_linspace = np.linspace(x.min(), x.max(), 100).reshape(-1, 1)
x_linspace_poly = poly.transform(x_linspace)
y_linspace_pred = model.predict(x_linspace_poly)

plt.figure(figsize=(10, 6))
plt.scatter(x_train, y_train, label='Training data', color='blue')
plt.scatter(x_val, y_val, label='Validation data', color='orange')
plt.plot(x_linspace, y_linspace_pred, label=f'Best Model (Degree {best_degree})', color='red')
plt.xlabel('X')
plt.ylabel('y')
plt.title(f'Polynomial Regression Model (Degree {best_degree})')
plt.legend()
plt.grid(True)
plt.show()
```

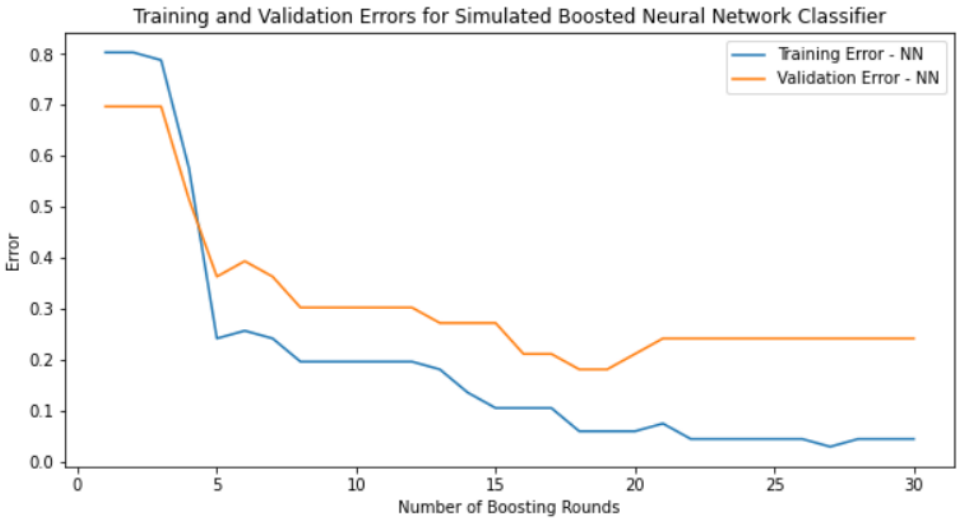
```
Shape of x: (21, 1)
Shape of y: (21, 1)
```




```
Best model degree: 3 with validation error: 0.08637378785747374
```




```python
# 11.3
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import train_test_split
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

# Load the data
data_path = 'new_circle_data.csv'
data = np.loadtxt(data_path, delimiter=',')

# Split the data into features and target
X = data[:-1, :].T
y = data[-1, :].T

# Split the data into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=1/3, random_state=42)

# Number of boosting rounds
M = 30

# Initialize lists to keep track of errors
train_errors_nn = []
val_errors_nn = []

# Define a neural network classifier as the base learner
base_nn_clf = MLPClassifier(hidden_layer_sizes=(10,), learning_rate_init=0.01, max_iter=1, warm_start=True)

# Manually simulate boosting using neural networks
for m in range(M):
    # Fit the model
    base_nn_clf.fit(X_train, y_train)

    # Calculate training and validation errors
    train_error = 1 - accuracy_score(y_train, base_nn_clf.predict(X_train))
    val_error = 1 - accuracy_score(y_val, base_nn_clf.predict(X_val))
    train_errors_nn.append(train_error)
    val_errors_nn.append(val_error)
    base_nn_clf.set_params(max_iter=base_nn_clf.max_iter + 1)

# Plotting the training and validation errors
plt.figure(figsize=(10, 5))
plt.plot(range(1, M + 1), train_errors_nn, label='Training Error - NN')
plt.plot(range(1, M + 1), val_errors_nn, label='Validation Error - NN')
plt.xlabel('Number of Boosting Rounds')
plt.ylabel('Error')
plt.title('Training and Validation Errors for Simulated Boosted Neural Network Classifier')
plt.legend()
plt.show()
```


Training and Validation Errors for Simulated Boosted Neural Network Classifier
Training Error - NN
Validation Error - NN
Error
Number of Boosting Rounds
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
5
10
15
20
25
30