Chapter 2: Training the Perceptron Model on the Iris Data Set

- **What is a perceptron?**
  - A perceptron is a single-layer neural network, the simplest kind of classifier.
  - It takes several inputs (features), multiplies each by a weight, adds a bias, and passes the result through a step function.
  - The goal: Find weights & bias such that it separates the classes with a straight line (or hyperplane in higher dimensions).

- **Why two classes and two features?**
  - Iris dataset has 3 classes; but perceptron is for binary classification.
  - Setosa and Versicolor are linearly separable; this lets the perceptron find a perfect boundary.
  - Two features = easier 2D visualization.

- **How does it learn?**
  - The perceptron starts with random weights.
  - For each data point:
    - Makes a prediction
    - Compares to true label
    - Updates the weights if it made a mistake
  - Repeats for several epochs (passes over all data)
  - The "decision boundary" is the line (in 2D) where the perceptron output flips from one class to another.

- **Full Code Walk Through:**
  - Step 1: Import the important libraries:-

```
# Use pip3 install numpy in the terminal
# Use pip3 install pandas in the terminal
# Use pip3 install matplotlib in the terminal

# Import essential libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Explanation:

- numpy for numerical operations
- pandas for handling datasets
- matplotlib for visualization

  - Step 2: Load the Iris Data Set:-

Chapter 2: Training the Perceptron Model on the Iris Data Set

```
# Read the CSV into a pandas DataFrame
df = pd.read_csv( url , header=None, encoding='utf-8')

# Show the last 5 rows to confirm successful load
print(df.tail())
```

```
     0    1    2    3               4
145  6.7  3.0  5.2  2.3  Iris-virginica
146  6.3  2.5  5.0  1.9  Iris-virginica
147  6.5  3.0  5.2  2.0  Iris-virginica
148  6.2  3.4  5.4  2.3  Iris-virginica
149  5.9  3.0  5.1  1.8  Iris-virginica
```

Explanation:

- url: Link to Iris dataset
- read_csv: Loads the data. header=None because the CSV file doesn't have a header row.
- print(df.tail()): Check that we loaded it correctly

➢ Step 3: Select Two Classes (Setosa vs. Versicolor) and Two Features

```
# Select only the first 100 rows (Iris-setosa and Iris-versicolor)
y = df.iloc[0:100, 4].values   # Species column
# Convert class labels to integers: setosa = -1, versicolor = 1
y = np.where(y == 'Iris-setosa', -1, 1)

# Select two features: sepal length (0) and petal length (2)
X = df.iloc[0:100, [0, 2]].values
```

Explanation:

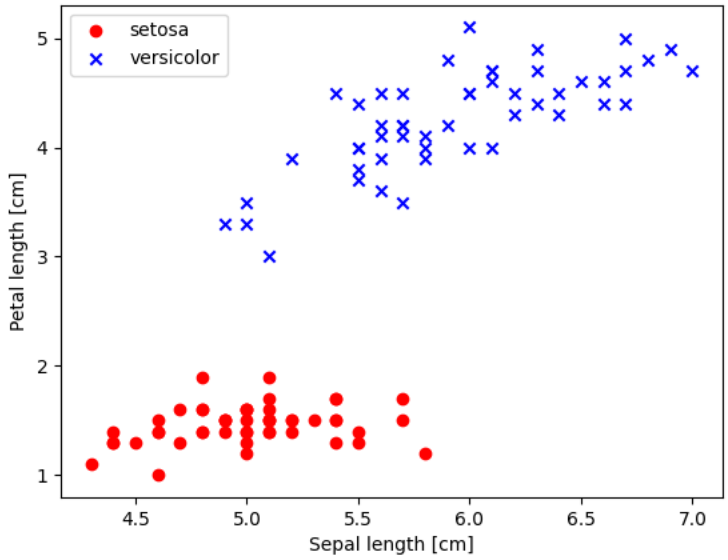
- iloc[0:100, 4]: Selects class labels for first 100 flowers
- np.where: Translates 'Iris-setosa' → -1, others → 1 (here, only versicolor)
- iloc[0:100, [0, 2]]: First 100 rows, columns 0 and 2 (sepal length and petal length)

➢ Step 4: Visualize the data:-

```
# Create a scatter plot: red for setosa, blue for versicolor
plt.scatter(X[:50, 0], X[:50, 1], color='red', marker='o', label='setosa')
plt.scatter(X[50:100, 0], X[50:100, 1], color='blue', marker='x', label='versicolor')

plt.xlabel('Sepal length [cm]')
plt.ylabel('Petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```

Chapter 2: Training the Perceptron Model on the Iris Data Set



Explanation:

- First 50 are setosa, next 50 are versicolor
- Each point = one flower
- Red circles = setosa, blue x = versicolor
- We'll see if the two classes are separated

➢ Step 5: Implement the perceptron:-

Chapter 2: Training the Perceptron Model on the Iris Data Set

```python
class Perceptron(object):
    """
    Perceptron classifier.

    Parameters
    ----------
    eta : float
        Learning rate (between 0.0 and 1.0)
    n_iter : int
        Passes over the training dataset (epochs)

    Attributes
    ----------
    w_ : 1d-array
        Weights after fitting
    errors_ : list
        Number of misclassifications (updates) in each epoch
    """
    def __init__(self, eta=0.01, n_iter=10):
        self.eta = eta
        self.n_iter = n_iter

    def fit(self, X, y):
        """Fit training data."""
        self.w_ = np.zeros(1 + X.shape[1])  # Initialize weights (including bias)
        self.errors_ = []

        for _ in range(self.n_iter):
            errors = 0
            for xi, target in zip(X, y):
                # Calculate the update value (learning rule)
                update = self.eta * (target - self.predict(xi))
                # Update the weights (skip index 0, which is the bias)
                self.w_[1:] += update * xi
                self.w_[0] += update   # bias
                # Count updates
                errors += int(update != 0.0)
            self.errors_.append(errors)
        return self

    def net_input(self, X):  1 usage
        """Calculate net input (weighted sum plus bias)"""
        return np.dot(X, self.w_[1:]) + self.w_[0]

    def predict(self, X):  1 usage
        """Return class label after unit step"""
        return np.where(self.net_input(X) >= 0.0, 1, -1)
```

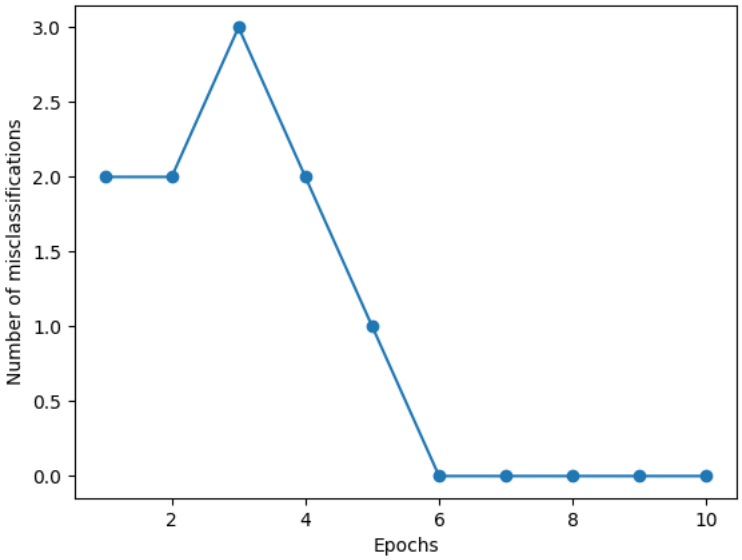
Explanation:

- __init__: Sets learning rate and number of epochs
- fit: Trains the perceptron; for each epoch, and each example, it predicts, calculates update, and adjusts weights
- net_input: Weighted sum plus bias (the value before the threshold)
- predict: Applies threshold: if net input >= 0, class 1; else, class -1

Chapter 2: Training the Perceptron Model on the Iris Data Set

- Step 5: Implement the perceptron:-

```
# Plot misclassifications (errors) per epoch
plt.plot( *args: range(1, len(ppn.errors_) + 1), ppn.errors_, marker='o')
plt.xlabel('Epochs')
plt.ylabel('Number of misclassifications')
plt.show()
```



Explanation:

- eta=0.1: Learning rate
- n_iter=10: Number of passes over the data
- fit(X, y): Start learning
- ppn.errors_: How many mistakes per epoch (should drop to zero if linearly separable)
- Plot shows if/when perceptron converges

- Step 7: Visualize the Decision Boundary:-

Chapter 2: Training the Perceptron Model on the Iris Data Set

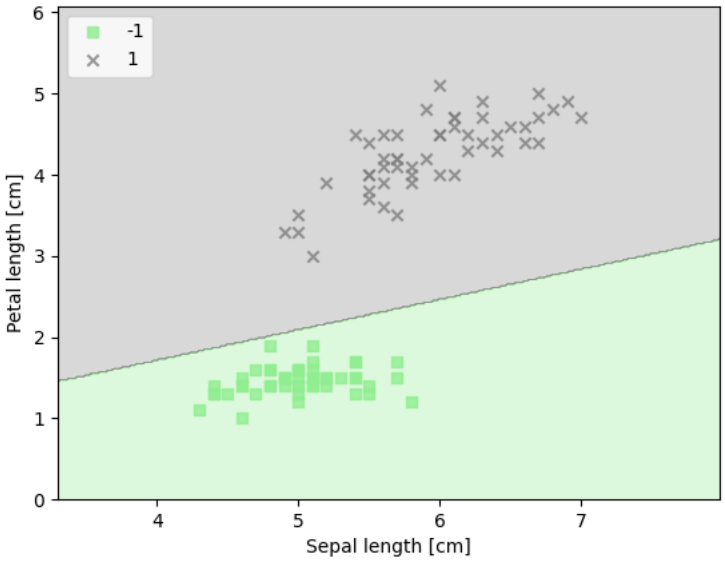
```python
from matplotlib.colors import ListedColormap

def plot_decision_regions(X, y, classifier, resolution=0.02):  1 usage
    # Setup marker generator and color map
    markers = ('s', 'x')
    colors = ('lightgreen', 'gray')
    cmap = ListedColormap(colors[:len(np.unique(y))])

    # Plot the decision surface
    x1_min, x1_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    x2_min, x2_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx1, xx2 = np.meshgrid(np.arange(x1_min, x1_max, resolution),
                           np.arange(x2_min, x2_max, resolution))
    Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.3, cmap=cmap)
    plt.xlim(xx1.min(), xx1.max())
    plt.ylim(xx2.min(), xx2.max())

    # Plot class samples
    for idx, cl in enumerate(np.unique(y)):
        plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],
                    alpha=0.8, c=colors[idx],
                    marker=markers[idx], label=cl)

# Now, use this function to plot decision regions
plot_decision_regions(X, y, classifier=ppn)
plt.xlabel('Sepal length [cm]')
plt.ylabel('Petal length [cm]')
plt.legend(loc='upper left')
plt.show()
```



Chapter 2: Training the Perceptron Model on the Iris Data Set

Explanation:

- We create a grid covering the 2D feature space.
- For every point in the grid, ask perceptron: is this setosa or versicolor?
- Color the region accordingly.
- Overlay the actual training points.
- The line between colored regions is the decision boundary.

Absolutely, let's wrap this up with a thorough, beginner-friendly summary of what you just achieved with the Iris dataset and the perceptron model—connecting concept, code, and visual output into one coherent understanding. This is the kind of summary you could also add as a final section in your Word document or your GitHub README.

❖ **Perceptron and the Iris Dataset — Beginner-Friendly Deep Dive**

➢ What Did I Just Do?

- I've just completed a classic experiment in machine learning:
  - I taught a computer to distinguish between two types of flowers (Setosa and Versicolor) using just their measurements.
  - I did this using a simple neural network model called a Perceptron—the foundational building block of modern AI.

➢ Why Is This Project Important?

- Hands-on intuition: The Iris dataset and perceptron algorithm are classic ML "first steps" because they are simple enough to visualize, but powerful enough to introduce fundamental concepts.
- Visual clarity: You get to see how an algorithm "draws a line" to separate categories in data—something that underlies all modern classification, from spam filters to facial recognition.
- Foundational skills: By coding the algorithm from scratch (rather than just using a library), I have learned how learning actually happens in machines.

➢ Step-by-Step: What Happened and Why

- Data Loading & Exploration
  - I started by loading the Iris dataset—a small table where each row represents a flower, and each column is a measurement (like petal length).

Chapter 2: Training the Perceptron Model on the Iris Data Set

- I focused only on two types of flowers (Setosa and Versicolor) and just two features (sepal length and petal length) to keep things easy to visualize.
- I plotted these points on a 2D graph. Instantly, you could see two "clouds" of points—this hinted that a straight line could probably separate them.

- What is a Perceptron?
  - Imagine a simple decision-maker: it takes a couple of numbers as input, multiplies them by some weights, adds a "bias" (just another number), and checks if the result is above or below zero.
  - If above zero, it calls it one class (say, Versicolor). If below, it calls it the other (Setosa).
  - The perceptron doesn't know the best weights to use at first. But it learns by making mistakes, and then adjusting the weights to do better next time.

- The Learning Process
  - The perceptron "looks" at each flower, guesses its class, checks if it's right, and if not, updates its weights a bit.
  - It does this over all the data, several times (called epochs), each time hopefully making fewer mistakes.
  - If the two classes can be separated by a straight line, the perceptron will find it.

- Visualizing Progress
  - I plotted the number of mistakes (misclassifications) at each epoch.
  - When the number of errors drops to zero and stays there, the perceptron has "learned" to separate the flowers perfectly (at least for the training data).

- Visualizing the Decision Boundary
  - I created a decision region plot: this shows which side of the line the perceptron predicts as Setosa, and which as Versicolor.
  - The actual flower points are overlaid on top.
  - The decision boundary (the dividing line) is the perceptron's "rule" for classifying new flowers.

Chapter 2: Training the Perceptron Model on the Iris Data Set

- Big Picture: What Did I Learn?
  - Data Representation: Real-world objects (flowers) can be represented as numbers and visualized.
  - Classification: Simple algorithms can "draw a line" to separate groups in data.
  - Learning from Mistakes: The perceptron is a primitive AI—it learns by seeing what it got wrong and adjusting.
  - The Power (and Limitations) of Linear Models: Perceptrons work only when the data is "linearly separable." If there's no straight line that can divide the two classes, the perceptron can't solve it.

- Where Does This Lead Next?
  - More Complex Models: What if the data isn't perfectly linearly separable? This is where more advanced algorithms (logistic regression, SVMs, neural networks) come in.
  - More Features: Using more than two features (dimensions) is typical in real problems. Decision boundaries become planes or hyperplanes I can't visualize, but the math is the same.
  - Multiclass Classification: Classifying more than two types—handled by combining several perceptrons or using more advanced techniques.

Key Takeaway (Explain Like I'm 10):

You taught your computer to tell apart two types of flowers just by looking at their measurements, by drawing a line between them. You did this by writing an algorithm that learns by trying, failing, and adjusting, and you got to see its decision in a picture.