# NLP with Spacy

In [1]:
```
! conda install spacy
```

```
Channels:
 - defaults
Platform: win-64
Collecting package metadata (repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.
```

In [1]:
```
import spacy
import pandas as pd
import numpy as np
from collections import Counter, defaultdict
import re
from textblob import TextBlob
import matplotlib.pyplot as plt
import seaborn as sns
```

In [2]:
```
import subprocess
import sys
```

In [3]:
```
# Install the English model
subprocess.check_call([sys.executable, "-m", "spacy", "download", "en_core_web_s
```

Out[3]: 0

In [4]:
```
# Load spaCy model (download with: python -m spacy download en_core_web_sm)
try:
    nlp = spacy.load("en_core_web_sm")
    print("✓ spaCy English model loaded successfully")
except OSError:
    print("Please install spaCy English model: python -m spacy download en_core_
    exit()
```

```
✓ spaCy English model loaded successfully
```

In [5]:
```
class AmazonReviewsNLP:
    def __init__(self):
        self.nlp = nlp
        self.reviews_data = []
        self.processed_results = []

    def load_data(self, file_path=None, sample_size=1000):
        """
        Load Amazon reviews data. If no file provided, create sample data.
```

```
        Expected format for real data: __label__1 review_text or __label__2 revi
        """
        if file_path:
            try:
                with open(file_path, 'r', encoding='utf-8') as f:
                    lines = f.readlines()[:sample_size]

                for line in lines:
                    line = line.strip()
                    if line.startswith('__label__'):
                        parts = line.split(' ', 1)
                        if len(parts) == 2:
                            label = parts[0].replace('__label__', '')
                            text = parts[1]
                            self.reviews_data.append({
                                'label': int(label),
                                'text': text,
                                'sentiment': 'positive' if int(label) == 2 else
                            })
                print(f"✓ Loaded {len(self.reviews_data)} reviews from file")
            except FileNotFoundError:
                print(f"File {file_path} not found. Creating sample data instead
                self.create_sample_data()
        else:
            self.create_sample_data()

    def create_sample_data(self):
        """Create sample Amazon review data for demonstration"""
        sample_reviews = [
            {
                'label': 2,
                'text': "I absolutely love my new iPhone 15 Pro from Apple! The
                'sentiment': 'positive'
            },
            {
                'label': 1,
                'text': "The Samsung Galaxy phone I ordered was defective. Poor
                'sentiment': 'negative'
            },
            {
                'label': 2,
                'text': "Sony WH-1000XM4 headphones are incredible! The noise ca
                'sentiment': 'positive'
            },
            {
                'label': 1,
                'text': "Nike Air Max shoes were uncomfortable and overpriced. T
                'sentiment': 'negative'
            },
            {
                'label': 2,
                'text': "The MacBook Pro from Apple exceeded my expectations. Fa
                'sentiment': 'positive'
            },
            {
                'label': 1,
                'text': "Dell laptop arrived damaged. Customer service was unhel
                'sentiment': 'negative'
            },
            {
```

```
            'label': 2,
            'text': "Amazon Echo Dot with Alexa is so convenient! The voice
            'sentiment': 'positive'
        },
        {
            'label': 1,
            'text': "The Kindle Fire tablet from Amazon was slow and buggy.
            'sentiment': 'negative'
        },
        {
            'label': 2,
            'text': "Love my new Canon EOS camera! The image quality is prof
            'sentiment': 'positive'
        },
        {
            'label': 1,
            'text': "Microsoft Surface Pro was overheating constantly. Poor
            'sentiment': 'negative'
        }
    ]

    self.reviews_data = sample_reviews
    print(f"✓ Created {len(self.reviews_data)} sample reviews for demonstrat

def process_reviews(self):
    """Process all reviews for NER and sentiment analysis"""
    print("Processing reviews...")

    for i, review in enumerate(self.reviews_data):
        text = review['text']

        # Extract entities
        entities = self.extract_entities(text)

        # Analyze sentiment
        sentiment_analysis = self.analyze_sentiment_rule_based(text)

        # Store results
        result = {
            'review_id': i,
            'original_text': text,
            'original_label': review['label'],
            'original_sentiment': review['sentiment'],
            'extracted_entities': entities,
            'sentiment_analysis': sentiment_analysis,
            'products_found': entities['products'],
            'brands_found': entities['brands']
        }

        self.processed_results.append(result)

    print(f"✓ Processed {len(self.processed_results)} reviews")

def extract_entities(self, text):
    """Extract named entities using spaCy NER"""
    doc = self.nlp(text)
    entities = {
        'products': [],
        'brands': [],
        'organizations': [],
        'money': []
```

```
            'money': [],
            'all_entities': []
        }

        # Common product/brand keywords to help identify entities
        product_keywords = ['iphone', 'galaxy', 'macbook', 'kindle', 'echo', 'su
        brand_keywords = ['apple', 'samsung', 'sony', 'nike', 'adidas', 'dell',

        for ent in doc.ents:
            entity_info = {
                'text': ent.text,
                'label': ent.label_,
                'description': spacy.explain(ent.label_)
            }
            entities['all_entities'].append(entity_info)

            # Classify entities
            if ent.label_ in ['PRODUCT', 'WORK_OF_ART'] or any(keyword in ent.te
                entities['products'].append(ent.text)
            elif ent.label_ in ['ORG', 'PERSON'] or any(keyword in ent.text.lowe
                entities['brands'].append(ent.text)
            elif ent.label_ == 'ORG':
                entities['organizations'].append(ent.text)
            elif ent.label_ == 'MONEY':
                entities['money'].append(ent.text)

        # Additional pattern matching for products and brands
        text_lower = text.lower()
        for keyword in product_keywords:
            if keyword in text_lower and keyword.title() not in entities['produc
                entities['products'].append(keyword.title())

        for keyword in brand_keywords:
            if keyword in text_lower and keyword.title() not in entities['brands
                entities['brands'].append(keyword.title())

        return entities
    def analyze_sentiment_rule_based(self, text):
        """Rule-based sentiment analysis"""
        # Positive and negative word lists
        positive_words = [
            'love', 'amazing', 'fantastic', 'excellent', 'great', 'perfect', 'wo
            'awesome', 'brilliant', 'outstanding', 'superb', 'incredible', 'best
            'good', 'nice', 'beautiful', 'impressive', 'satisfied', 'happy', 'pl
        ]

        negative_words = [
            'hate', 'terrible', 'awful', 'horrible', 'bad', 'worst', 'disappoint
            'poor', 'defective', 'broken', 'useless', 'waste', 'overpriced', 'sl
            'buggy', 'crashed', 'damaged', 'unhelpful', 'uncomfortable', 'disapp
        ]

        # Intensifiers
        intensifiers = ['very', 'extremely', 'really', 'absolutely', 'completely

        text_lower = text.lower()
        words = text_lower.split()

        positive_score = 0
        negative_score = 0
```

```
        for i, word in enumerate(words):
            # Check for intensifiers
            multiplier = 1
            if i > 0 and words[i-1] in intensifiers:
                multiplier = 2

            if word in positive_words:
                positive_score += (1 * multiplier)
            elif word in negative_words:
                negative_score += (1 * multiplier)

        # Handle negations (simple approach)
        negation_words = ['not', 'no', 'never', 'nothing', 'nobody', 'nowhere',
        for neg_word in negation_words:
            if neg_word in text_lower:
                # Flip scores if negation is present
                positive_score, negative_score = negative_score * 0.5, positive_
                break

        # Determine overall sentiment
        if positive_score > negative_score:
            sentiment = 'positive'
            confidence = positive_score / (positive_score + negative_score + 1)
        elif negative_score > positive_score:
            sentiment = 'negative'
            confidence = negative_score / (positive_score + negative_score + 1)
        else:
            sentiment = 'neutral'
            confidence = 0.5

        return {
            'sentiment': sentiment,
            'confidence': confidence,
            'positive_score': positive_score,
            'negative_score': negative_score
        }
    def display_results(self):
        """Display analysis results"""
        print("\n" + "="*80)
        print("AMAZON REVIEWS NLP ANALYSIS RESULTS")
        print("="*80)

        # Summary statistics
        total_reviews = len(self.processed_results)
        correct_predictions = sum(1 for r in self.processed_results
                                  if r['sentiment_analysis']['sentiment'] == r['or
        accuracy = correct_predictions / total_reviews * 100

        print(f"\nSUMMARY STATISTICS:")
        print(f"Total Reviews Processed: {total_reviews}")
        print(f"Sentiment Analysis Accuracy: {accuracy:.1f}%")
        all_products = []
        all_brands = []
        for result in self.processed_results:
            all_products.extend(result['products_found'])
            all_brands.extend(result['brands_found'])

        product_counts = Counter(all_products)
        brand_counts = Counter(all_brands)
```

```
        print(f"\nENTITY EXTRACTION SUMMARY:")
        print(f"Unique Products Found: {len(set(all_products))}")
        print(f"Unique Brands Found: {len(set(all_brands))}")
        print(f"Most Common Products: {dict(product_counts.most_common(5))}")
        print(f"Most Common Brands: {dict(brand_counts.most_common(5))}")

        # Detailed results for each review
        print(f"\nDETAILED ANALYSIS:")
        print("-" * 80)

        for i, result in enumerate(self.processed_results):
            print(f"\nReview #{i+1}:")
            print(f"Text: {result['original_text'][:100]}{'...' if len(result['o
            print(f"Original Sentiment: {result['original_sentiment']}")
            print(f"Predicted Sentiment: {result['sentiment_analysis']['sentimen
            print(f"Products Found: {result['products_found']}")
            print(f"Brands Found: {result['brands_found']}")

            # Show all entities found
            if result['extracted_entities']['all_entities']:
                print("All Named Entities:")
                for entity in result['extracted_entities']['all_entities']:
                    print(f"  • {entity['text']} ({entity['label']}: {entity['de

            print("-" * 40)

    def create_visualizations(self):
        """Create visualizations of the analysis results"""
        # Sentiment distribution
        sentiments = [r['sentiment_analysis']['sentiment'] for r in self.process
        sentiment_counts = Counter(sentiments)

        # Product and brand frequency
        all_products = []
        all_brands = []
        for result in self.processed_results:
            all_products.extend(result['products_found'])
            all_brands.extend(result['brands_found'])

        product_counts = Counter(all_products)
        brand_counts = Counter(all_brands)

        # Create plots
        fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(15, 10))

        # Sentiment distribution
        ax1.pie(sentiment_counts.values(), labels=sentiment_counts.keys(), autop
        ax1.set_title('Sentiment Distribution')

        # Accuracy comparison
        original_sentiments = [r['original_sentiment'] for r in self.processed_r
        predicted_sentiments = [r['sentiment_analysis']['sentiment'] for r in se
        accuracy_data = {'Original': Counter(original_sentiments), 'Predicted':

        x = np.arange(len(set(original_sentiments)))
        width = 0.35
        ax2.bar(x - width/2, [accuracy_data['Original']['positive'], accuracy_da
                width, label='Original', alpha=0.7)
        ax2.bar(x + width/2, [accuracy_data['Predicted'].get('positive', 0), acc
                width, label='Predicted', alpha=0.7)
        ax2.set_xlabel('Sentiment')
```

```
        ax2.set_ylabel('Count')
        ax2.set_title('Original vs Predicted Sentiment')
        ax2.set_xticks(x)
        ax2.set_xticklabels(['Positive', 'Negative'])
        ax2.legend()

        # Top products
        if product_counts:
            top_products = dict(product_counts.most_common(5))
            ax3.barh(list(top_products.keys()), list(top_products.values()))
            ax3.set_title('Most Mentioned Products')
            ax3.set_xlabel('Frequency')

        # Top brands
        if brand_counts:
            top_brands = dict(brand_counts.most_common(5))
            ax4.barh(list(top_brands.keys()), list(top_brands.values()))
            ax4.set_title('Most Mentioned Brands')
            ax4.set_xlabel('Frequency')

        plt.tight_layout()
        plt.show()
```

In [6]:

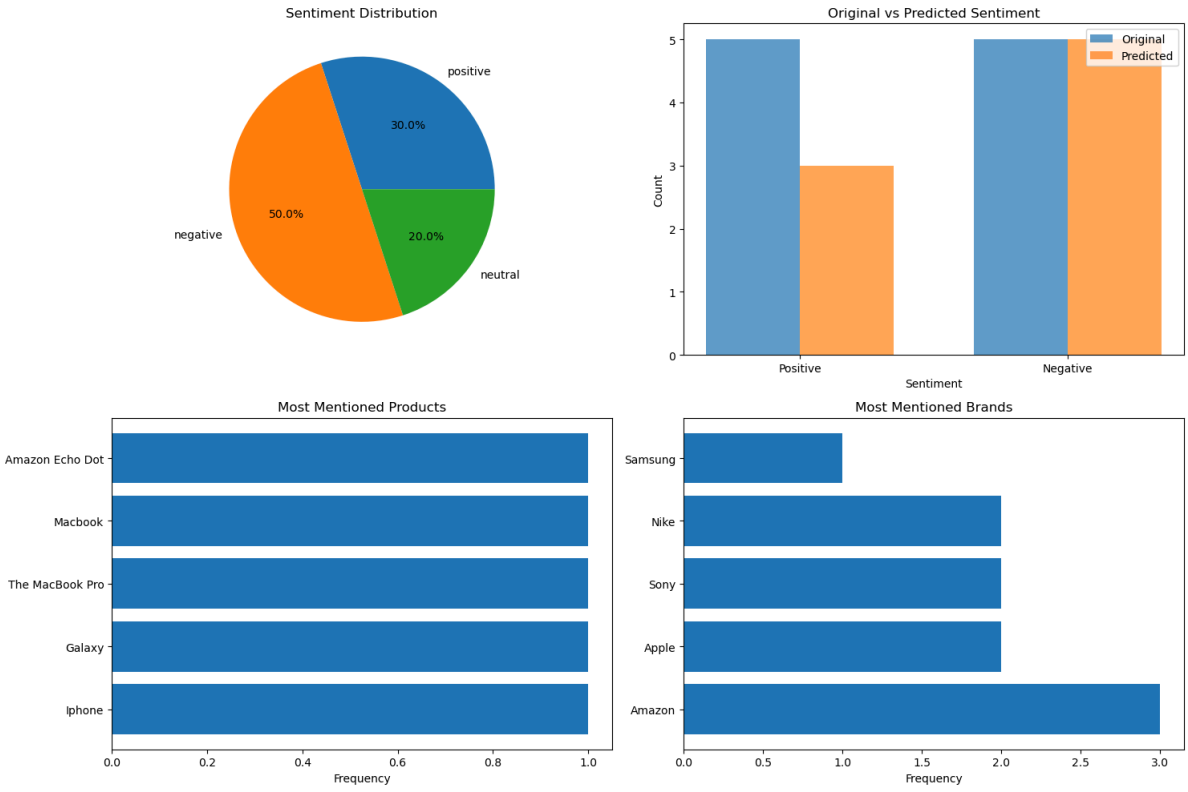
```
# Main execution
def main():
    print("Amazon Reviews NLP Analysis with spaCy")
    print("=" * 50)

    # Initialize analyzer
    analyzer = AmazonReviewsNLP()

    # Load data (replace with actual file path if available)
    # analyzer.load_data('path/to/amazon_reviews.txt', sample_size=1000)
    analyzer.load_data()  # Uses sample data

    # Process reviews
    analyzer.process_reviews()

    # Display results
    analyzer.display_results()

    # Create visualizations
    try:
        analyzer.create_visualizations()
    except Exception as e:
        print(f"Visualization error: {e}")
        print("Make sure matplotlib is installed: pip install matplotlib seaborn

if __name__ == "__main__":
    main()

# Additional utility functions for extended analysis
def analyze_entity_sentiment_correlation(analyzer):
    """Analyze correlation between specific entities and sentiment"""
    entity_sentiment = defaultdict(list)

    for result in analyzer.processed_results:
        sentiment = result['sentiment_analysis']['sentiment']
```

```
            for product in result['products_found']:
                entity_sentiment[product].append(sentiment)
            for brand in result['brands_found']:
                entity_sentiment[brand].append(sentiment)

    print("\nENTITY-SENTIMENT CORRELATION:")
    print("-" * 40)
    for entity, sentiments in entity_sentiment.items():
        if len(sentiments) > 1:  # Only show entities with multiple mentions
            positive_ratio = sentiments.count('positive') / len(sentiments)
            print(f"{entity}: {positive_ratio:.2f} positive ratio ({len(sentimen
```

```
Amazon Reviews NLP Analysis with spaCy
==================================================
✓ Created 10 sample reviews for demonstration
Processing reviews...
✓ Processed 10 reviews

================================================================================
AMAZON REVIEWS NLP ANALYSIS RESULTS
================================================================================

SUMMARY STATISTICS:
Total Reviews Processed: 10
Sentiment Analysis Accuracy: 60.0%

ENTITY EXTRACTION SUMMARY:
Unique Products Found: 10
Unique Brands Found: 12
Most Common Products: {'Iphone': 1, 'Galaxy': 1, 'The MacBook Pro': 1, 'Macbook':
1, 'Amazon Echo Dot': 1}
Most Common Brands: {'Amazon': 3, 'Apple': 2, 'Sony': 2, 'Nike': 2, 'Samsung': 1}

DETAILED ANALYSIS:
--------------------------------------------------------------------------------

Review #1:
Text: I absolutely love my new iPhone 15 Pro from Apple! The camera quality is ama
zing and the battery lif...
Original Sentiment: positive
Predicted Sentiment: positive (confidence: 0.800)
Products Found: ['Iphone']
Brands Found: ['Apple']
All Named Entities:
  • 15 (CARDINAL: Numerals that do not fall under another type)
  • Apple (ORG: Companies, agencies, institutions, etc.)
----------------------------------------

Review #2:
Text: The Samsung Galaxy phone I ordered was defective. Poor build quality and the
screen cracked after on...
Original Sentiment: negative
Predicted Sentiment: negative (confidence: 0.750)
Products Found: ['Galaxy']
Brands Found: ['Amazon', 'Samsung']
All Named Entities:
  • one day (DATE: Absolute or relative dates or periods)
  • Amazon (ORG: Companies, agencies, institutions, etc.)
----------------------------------------
```

```
Review #3:
Text: Sony WH-1000XM4 headphones are incredible! The noise cancellation works perf
ectly and the sound qual...
Original Sentiment: positive
Predicted Sentiment: neutral (confidence: 0.500)
Products Found: []
Brands Found: ['Sony', 'Sony']
All Named Entities:
  • Sony (ORG: Companies, agencies, institutions, etc.)
  • Sony (ORG: Companies, agencies, institutions, etc.)
----------------------------------------

Review #4:
Text: Nike Air Max shoes were uncomfortable and overpriced. The Adidas version was
much better. Would not ...
Original Sentiment: negative
Predicted Sentiment: positive (confidence: 0.333)
Products Found: []
Brands Found: ['Nike', 'Adidas', 'Nike']
All Named Entities:
  • Nike (ORG: Companies, agencies, institutions, etc.)
  • Adidas (PERSON: People, including fictional)
  • Nike (ORG: Companies, agencies, institutions, etc.)
----------------------------------------

Review #5:
Text: The MacBook Pro from Apple exceeded my expectations. Fast processor and grea
t display. Perfect for w...
Original Sentiment: positive
Predicted Sentiment: positive (confidence: 0.667)
Products Found: ['The MacBook Pro', 'Macbook']
Brands Found: ['Apple']
All Named Entities:
  • The MacBook Pro (ORG: Companies, agencies, institutions, etc.)
  • Apple (ORG: Companies, agencies, institutions, etc.)
----------------------------------------

Review #6:
Text: Dell laptop arrived damaged. Customer service was unhelpful and the product
quality was poor. Waste ...
Original Sentiment: negative
Predicted Sentiment: negative (confidence: 0.667)
Products Found: []
Brands Found: ['Dell']
----------------------------------------

Review #7:
Text: Amazon Echo Dot with Alexa is so convenient! The voice recognition is accura
te and it integrates wel...
Original Sentiment: positive
Predicted Sentiment: neutral (confidence: 0.500)
Products Found: ['Amazon Echo Dot', 'Echo']
Brands Found: ['Alexa', 'Amazon']
All Named Entities:
  • Amazon Echo Dot (ORG: Companies, agencies, institutions, etc.)
  • Alexa (ORG: Companies, agencies, institutions, etc.)
----------------------------------------

Review #8:
```

```
Text: The Kindle Fire tablet from Amazon was slow and buggy. Battery died quickly
and apps crashed frequen...
Original Sentiment: negative
Predicted Sentiment: negative (confidence: 0.667)
Products Found: ['The Kindle Fire', 'Kindle']
Brands Found: ['Amazon', 'apps']
All Named Entities:
  • The Kindle Fire (LOC: Non-GPE locations, mountain ranges, bodies of water)
  • Amazon (ORG: Companies, agencies, institutions, etc.)
  • apps (PERSON: People, including fictional)
----------------------------------------

Review #9:
Text: Love my new Canon EOS camera! The image quality is professional-grade and th
e autofocus is lightning...
Original Sentiment: positive
Predicted Sentiment: negative (confidence: 0.333)
Products Found: []
Brands Found: ['EOS', 'Canon']
All Named Entities:
  • EOS (ORG: Companies, agencies, institutions, etc.)
----------------------------------------

Review #10:
Text: Microsoft Surface Pro was overheating constantly. Poor thermal design and cu
stomer support was terri...
Original Sentiment: negative
Predicted Sentiment: negative (confidence: 0.500)
Products Found: ['Surface Pro', 'Surface']
Brands Found: ['Microsoft']
All Named Entities:
  • Microsoft (ORG: Companies, agencies, institutions, etc.)
  • Surface Pro (PERSON: People, including fictional)
----------------------------------------
```




In [7]:

```
def export_results_to_csv(analyzer, filename='amazon_reviews_analysis.csv'):
```

```
def export_results_to_csv(analyzer, filename='amazon_reviews_analysis.csv'):
    """Export analysis results to CSV"""
    data = []
    for result in analyzer.processed_results:
        data.append({
            'review_id': result['review_id'],
            'original_text': result['original_text'],
            'original_sentiment': result['original_sentiment'],
            'predicted_sentiment': result['sentiment_analysis']['sentiment'],
            'confidence': result['sentiment_analysis']['confidence'],
            'products_found': ', '.join(result['products_found']),
            'brands_found': ', '.join(result['brands_found']),
            'total_entities': len(result['extracted_entities']['all_entities'])
        })

    df = pd.DataFrame(data)
    df.to_csv(filename, index=False)
    print(f"✓ Results exported to {filename}")
```

In [8]:
```
analyzer = AmazonReviewsNLP()
analyzer.load_data()
analyzer.process_reviews()
```

```
✓ Created 10 sample reviews for demonstration
Processing reviews...
✓ Processed 10 reviews
```

In [9]:
```
export_results_to_csv(analyzer, 'my_results.csv')
```

```
✓ Results exported to my_results.csv
```

## Machine learning with Scikit Learn

Decision Trees

In [10]:
```
# Import necessary libraries
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, class
from sklearn.preprocessing import LabelEncoder
```

In [11]:
```
# Load the dataset
from sklearn.datasets import load_iris
iris = load_iris()
```

In [12]:
```
# Create a DataFrame for better visualization and handling
iris_df = pd.DataFrame(data=np.c_[iris['data'], iris['target']],
                       columns=iris['feature_names'] + ['target'])
```

In [13]:
```
# Display basic information about the dataset
print("=== Dataset Information ===")
print(iris_df.info())
print("\nFirst 5 rows:")
print(iris_df.head())
print("=== No of rows and columns ===")
print(iris_df.shape)
```

```
=== Dataset Information ===
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   target             150 non-null    float64
dtypes: float64(5)
memory usage: 6.0 KB
None

First 5 rows:
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2   
1                4.9               3.0                1.4               0.2   
2                4.7               3.2                1.3               0.2   
3                4.6               3.1                1.5               0.2   
4                5.0               3.6                1.4               0.2   

   target
0     0.0
1     0.0
2     0.0
3     0.0
4     0.0
=== No of rows and columns ===
(150, 5)
```

In [14]:
```
# Check for missing values
print("\n=== Missing Values ===")
print(iris_df.isnull().sum())
```

```
=== Missing Values ===
sepal length (cm)    0
sepal width (cm)     0
petal length (cm)    0
petal width (cm)     0
target               0
dtype: int64
```

There are no null values in our dataset.

In [15]:
```
# Separate features (X) and target (y)
#separate the features (sepal length, sepal width, petal length, petal width) fr
X = iris_df.drop('target', axis=1)
y = iris_df['target']
```

In [16]:
```
#convert  any string labels to numbers
le = LabelEncoder()
y_encoded = le.fit_transform(y)
print("\nOriginal target values:", y.unique())
print("Encoded target values:", np.unique(y_encoded))
```

```
Original target values: [0. 1. 2.]
Encoded target values: [0 1 2]
```

In [17]:
```
# Split the data into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_
```

In [18]:
```
 #Initialize the Decision Tree Classifier
dt_classifier = DecisionTreeClassifier(random_state=42)
```

In [19]:
```
#  train the decision tree on our training data using the fit() method.
dt_classifier.fit(X_train, y_train)
```

Out[19]:
```
DecisionTreeClassifier(random_state=42)
```
**In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.**
**On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.**

In [20]:
```
# Make predictions on the test set
y_pred = dt_classifier.predict(X_test)
```

In [21]:
```
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
# For multiclass classification, we use average parameter
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
```

In [22]:
```
print("\n=== Model Evaluation ===")
print(f"Accuracy: {accuracy:.4f}")
print(f"Precision: {precision:.4f}")
print(f"Recall: {recall:.4f}")
```

```
=== Model Evaluation ===
Accuracy: 1.0000
Precision: 1.0000
Recall: 1.0000
```

In [23]:
```
# Detailed classification report
print("\n=== Classification Report ===")
print(classification_report(y_test, y_pred, target_names=iris.target_names))
```

```
=== Classification Report ===
              precision    recall  f1-score   support

      setosa       1.00      1.00      1.00        10
  versicolor       1.00      1.00      1.00         9
   virginica       1.00      1.00      1.00        11

    accuracy                           1.00        30
   macro avg       1.00      1.00      1.00        30
weighted avg       1.00      1.00      1.00        30
```

In [24]:

```
# Feature importance
# Determine the most important for the decision tree's predictions.
print("\n=== Feature Importance ===")
for name, importance in zip(iris.feature_names, dt_classifier.feature_importance
    print(f"{name}: {importance:.4f}")
```

```
=== Feature Importance ===
sepal length (cm): 0.0000
sepal width (cm): 0.0167
petal length (cm): 0.9061
petal width (cm): 0.0772
```

## MNIST Handwritten Digit Classification with CNN

In [25]:

```
import sys
print(f"Python executable: {sys.executable}")
print(f"Python version: {sys.version}")
```

```
Python executable: C:\Users\priscillah\anaconda3\envs\tf_env\python.exe
Python version: 3.10.13 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:15:5
7) [MSC v.1916 64 bit (AMD64)]
```

In [26]:

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np
import matplotlib.pyplot as plt
```

In [27]:

```
# Load MNIST dataset
(X_train, y_train), (X_test, y_test) = keras.datasets.mnist.load_data()
```

In [28]:

```
# Preprocess the data
# Normalize pixel values to [0, 1]
X_train = X_train.astype("float32") / 255
X_test = X_test.astype("float32") / 255
```

In [29]:

```
# Reshape images to include channel dimension (28, 28, 1)
X_train = np.expand_dims(X_train, -1)
X_test = np.expand_dims(X_test, -1)
```

In [30]:
```
# Convert labels to one-hot encoding
y_train = keras.utils.to_categorical(y_train, 10)
y_test = keras.utils.to_categorical(y_test, 10)
```

In [31]:
```
# Print dataset shape
print("Training data shape:", X_train.shape)
print("Training labels shape:", y_train.shape)
print("Test data shape:", X_test.shape)
print("Test labels shape:", y_test.shape)
```

```
Training data shape: (60000, 28, 28, 1)
Training labels shape: (60000, 10)
Test data shape: (10000, 28, 28, 1)
Test labels shape: (10000, 10)
```

In [32]:
```
# Build the CNN model
model = keras.Sequential([
    # First convolutional block
    layers.Conv2D(32, kernel_size=(3, 3), activation="relu", input_shape=(28, 28
    layers.MaxPooling2D(pool_size=(2, 2)),

    # Second convolutional block
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
    layers.MaxPooling2D(pool_size=(2, 2)),

    # Classifier head
    layers.Flatten(),
    layers.Dropout(0.5),  # Regularization
    layers.Dense(10, activation="softmax")
])
```

In [33]:
```
# Compile the model
model.compile(optimizer="adam",
              loss="categorical_crossentropy",
              metrics=["accuracy"])
```

In [34]:
```
# Display model architecture
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)
```

```
 flatten (Flatten)           (None, 1600)              0

 dropout (Dropout)           (None, 1600)              0

 dense (Dense)               (None, 10)                16010

=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

In [35]:

```
# Train the model
batch_size = 128
epochs = 15

history = model.fit(X_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_split=0.1)
```

```
Epoch 1/15
422/422 [==============================] - 17s 39ms/step - loss: 0.3779 - accurac
y: 0.8863 - val_loss: 0.0818 - val_accuracy: 0.9770
Epoch 2/15
422/422 [==============================] - 16s 39ms/step - loss: 0.1134 - accurac
y: 0.9654 - val_loss: 0.0548 - val_accuracy: 0.9852
Epoch 3/15
422/422 [==============================] - 17s 40ms/step - loss: 0.0836 - accurac
y: 0.9749 - val_loss: 0.0459 - val_accuracy: 0.9880
Epoch 4/15
422/422 [==============================] - 16s 38ms/step - loss: 0.0703 - accurac
y: 0.9784 - val_loss: 0.0403 - val_accuracy: 0.9878
Epoch 5/15
422/422 [==============================] - 17s 40ms/step - loss: 0.0609 - accurac
y: 0.9810 - val_loss: 0.0381 - val_accuracy: 0.9892
Epoch 6/15
422/422 [==============================] - 17s 40ms/step - loss: 0.0561 - accurac
y: 0.9819 - val_loss: 0.0369 - val_accuracy: 0.9893
Epoch 7/15
422/422 [==============================] - 18s 42ms/step - loss: 0.0513 - accurac
y: 0.9840 - val_loss: 0.0350 - val_accuracy: 0.9905
Epoch 8/15
422/422 [==============================] - 16s 37ms/step - loss: 0.0486 - accurac
y: 0.9848 - val_loss: 0.0308 - val_accuracy: 0.9913
Epoch 9/15
422/422 [==============================] - 16s 37ms/step - loss: 0.0442 - accurac
y: 0.9862 - val_loss: 0.0304 - val_accuracy: 0.9923
Epoch 10/15
422/422 [==============================] - 16s 37ms/step - loss: 0.0418 - accurac
y: 0.9863 - val_loss: 0.0308 - val_accuracy: 0.9923
Epoch 11/15
422/422 [==============================] - 15s 36ms/step - loss: 0.0407 - accurac
y: 0.9872 - val_loss: 0.0284 - val_accuracy: 0.9923
Epoch 12/15
422/422 [==============================] - 16s 37ms/step - loss: 0.0378 - accurac
y: 0.9880 - val_loss: 0.0303 - val_accuracy: 0.9913
Epoch 13/15
422/422 [==============================] - 16s 37ms/step - loss: 0.0357 - accurac
```

```
y: 0.9887 - val_loss: 0.0276 - val_accuracy: 0.9922
Epoch 14/15
422/422 [==============================] - 16s 37ms/step - loss: 0.0349 - accurac
y: 0.9887 - val_loss: 0.0265 - val_accuracy: 0.9928
Epoch 15/15
422/422 [==============================] - 15s 37ms/step - loss: 0.0332 - accurac
y: 0.9894 - val_loss: 0.0250 - val_accuracy: 0.9935
```

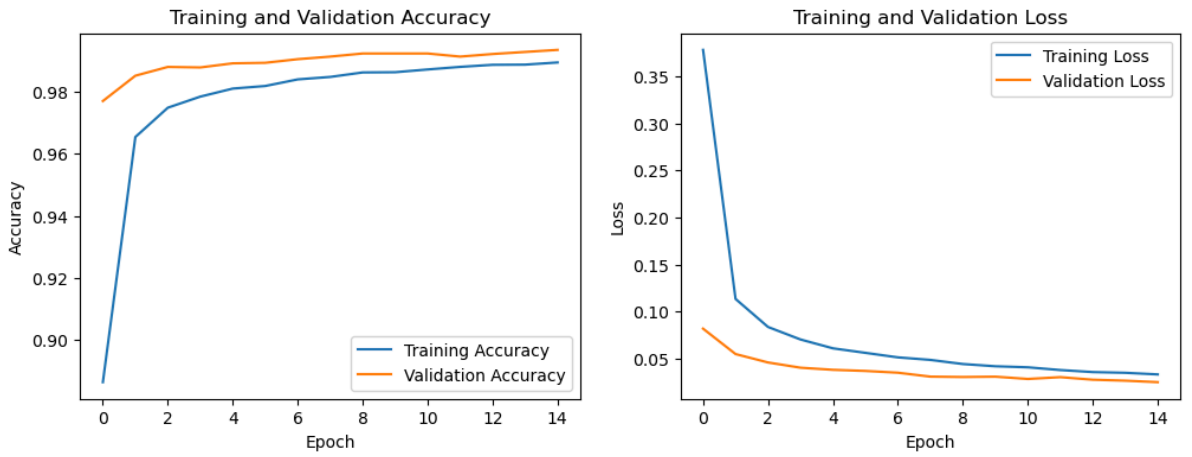
In [36]:

```
# Evaluate on test set
test_loss, test_acc = model.evaluate(X_test, y_test, verbose=0)
print(f"\nTest accuracy: {test_acc:.4f}")
```

```
Test accuracy: 0.9915
```

In [37]:

```
# Plot training history
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```




In [38]:

```
# Visualize predictions on sample images
sample_images = X_test[:5]
sample_labels = np.argmax(y_test[:5], axis=1)
```

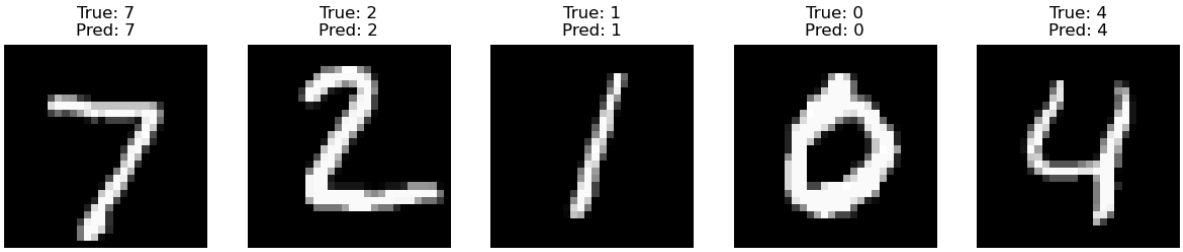
In [39]:

```
# Make predictions
predictions = model.predict(sample_images)
predicted_labels = np.argmax(predictions, axis=1)
```

```
1/1 [==============================] - 0s 170ms/step
```

In [40]:

```
# Display the images with predictions
plt.figure(figsize=(15, 3))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(sample_images[i].reshape(28, 28), cmap='gray')
    plt.title(f"True: {sample_labels[i]}\nPred: {predicted_labels[i]}")
    plt.axis('off')
plt.show()
```




In [ ]: