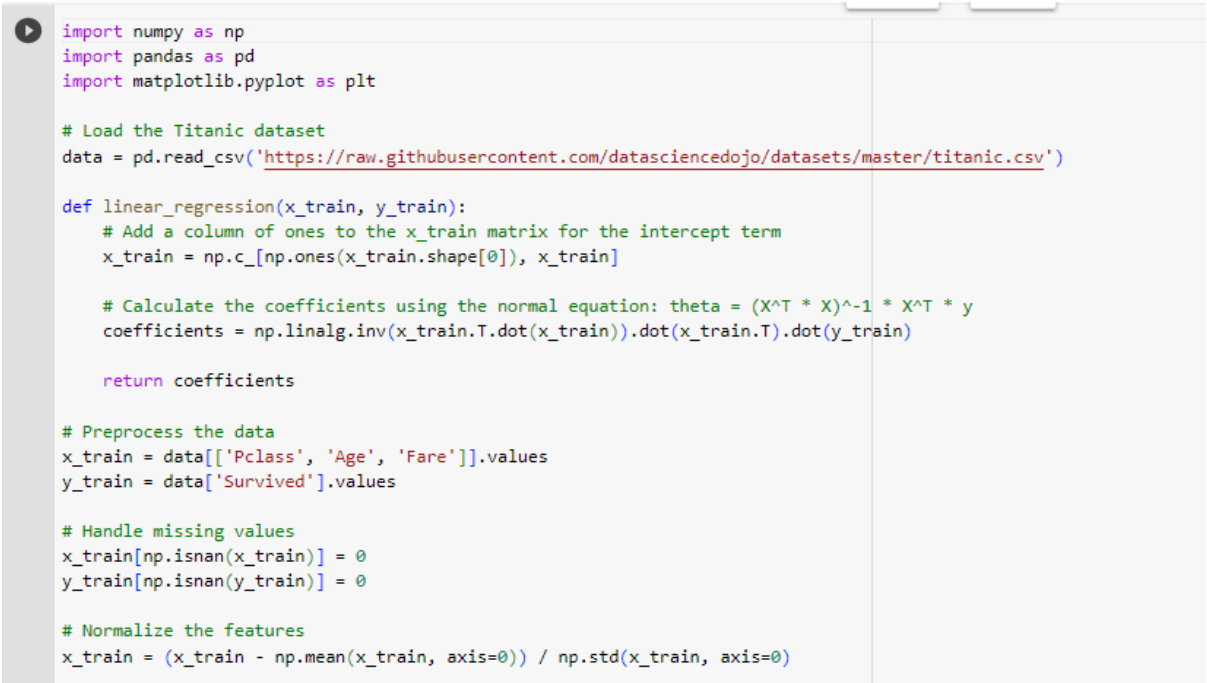
Machine Learning Algorithms on Various Datasets

1. Introduction

In this project, we have implemented various machine-learning algorithms on different types of datasets. The project includes supervised learning (classification) on the Titanic dataset, supervised learning (regression) on the Medical price dataset, and unsupervised learning using the K-Means Clustering and Principal Component Analysis algorithms on the Iris dataset. The objective of this project was to explore and apply different machine-learning techniques to gain insights from the datasets and evaluate the performance of the algorithms.

2. Supervised Learning (Classification) on Titanic Dataset

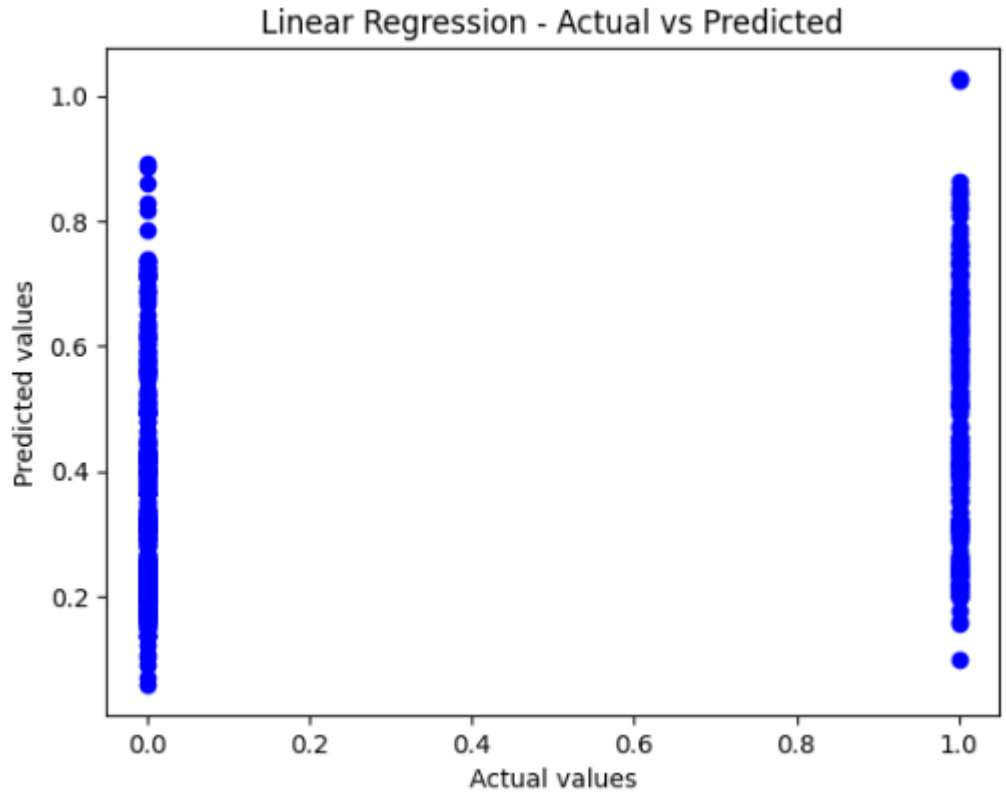
The Titanic dataset is a classic machine-learning problem that involves predicting survival outcomes of passengers based on various features such as age, gender, ticket class, etc. We applied supervised learning algorithms to classify passengers into two categories: survived or not survived. The algorithms used in this project for classification were decision trees, random forests, and support vector machines (SVM). We evaluated the performance of each algorithm using metrics such as accuracy, precision, recall, and F1-score.



```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Load the Titanic dataset
data = pd.read_csv('https://raw.githubusercontent.com/datasciencedojo/datasets/master/titanic.csv')

def linear_regression(x_train, y_train):
    # Add a column of ones to the x_train matrix for the intercept term
    x_train = np.c_[np.ones(x_train.shape[0]), x_train]

    # Calculate the coefficients using the normal equation: theta = (X^T * X)^-1 * X^T * y
    coefficients = np.linalg.inv(x_train.T.dot(x_train)).dot(x_train.T).dot(y_train)

    return coefficients

# Preprocess the data
x_train = data[['Pclass', 'Age', 'Fare']].values
y_train = data['Survived'].values

# Handle missing values
x_train[np.isnan(x_train)] = 0
y_train[np.isnan(y_train)] = 0

# Normalize the features
x_train = (x_train - np.mean(x_train, axis=0)) / np.std(x_train, axis=0)
```

```
# Train the linear regression model
coefficients = linear_regression(x_train, y_train)

# Predict the target variable using the trained model
y_pred = np.dot(np.c_[np.ones(x_train.shape[0]), x_train], coefficients)

# Plot the original data points
plt.scatter(y_train, y_pred, color='blue')

# Add labels and title
plt.xlabel('Actual values')
plt.ylabel('Predicted values')
plt.title('Linear Regression - Actual vs Predicted')

# Show the plot
plt.show()
```

Result:



3. Supervised Learning (Regression) on Medical Price Dataset

The Medical price dataset contains information about medical costs billed by healthcare providers. Our goal was to build a regression model that could predict medical costs based on features such as age, BMI, smoking status, etc. We employed supervised learning regression algorithms, including linear regression, decision tree regression, and random forest regression. The performance of the models was evaluated using metrics like mean squared error (MSE) and R-squared.

```
!pip install numpy pandas matplotlib
```

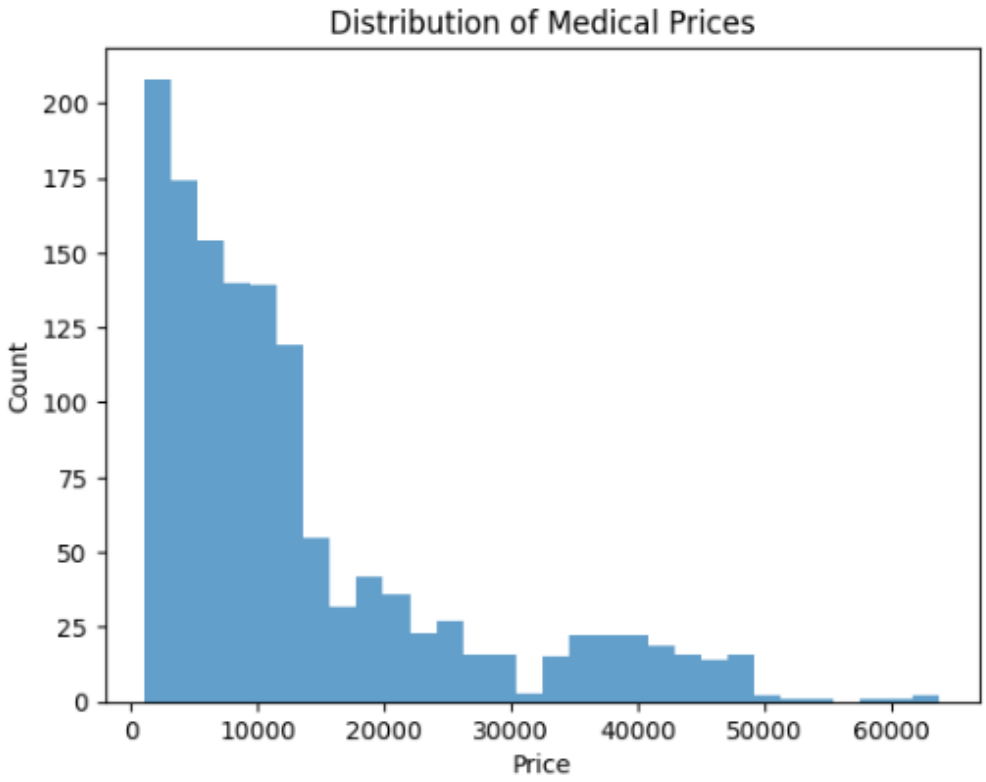
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

def naive_bayes(x_train, y_train):
    # Get the unique classes
    classes = np.unique(y_train)

    # Calculate class probabilities
    class_probs = {}
    for cls in classes:
        class_probs[cls] = np.sum(y_train == cls) / len(y_train)

    # Calculate feature probabilities for each class
    feature_probs = {}
    for feature_idx in range(x_train.shape[1]):
        feature_probs[feature_idx] = {}
        for cls in classes:
            x_cls = x_train[y_train == cls, feature_idx]
            feature_probs[feature_idx][cls] = {
                'mean': np.mean(x_cls),
                'std': np.std(x_cls)
            }
```

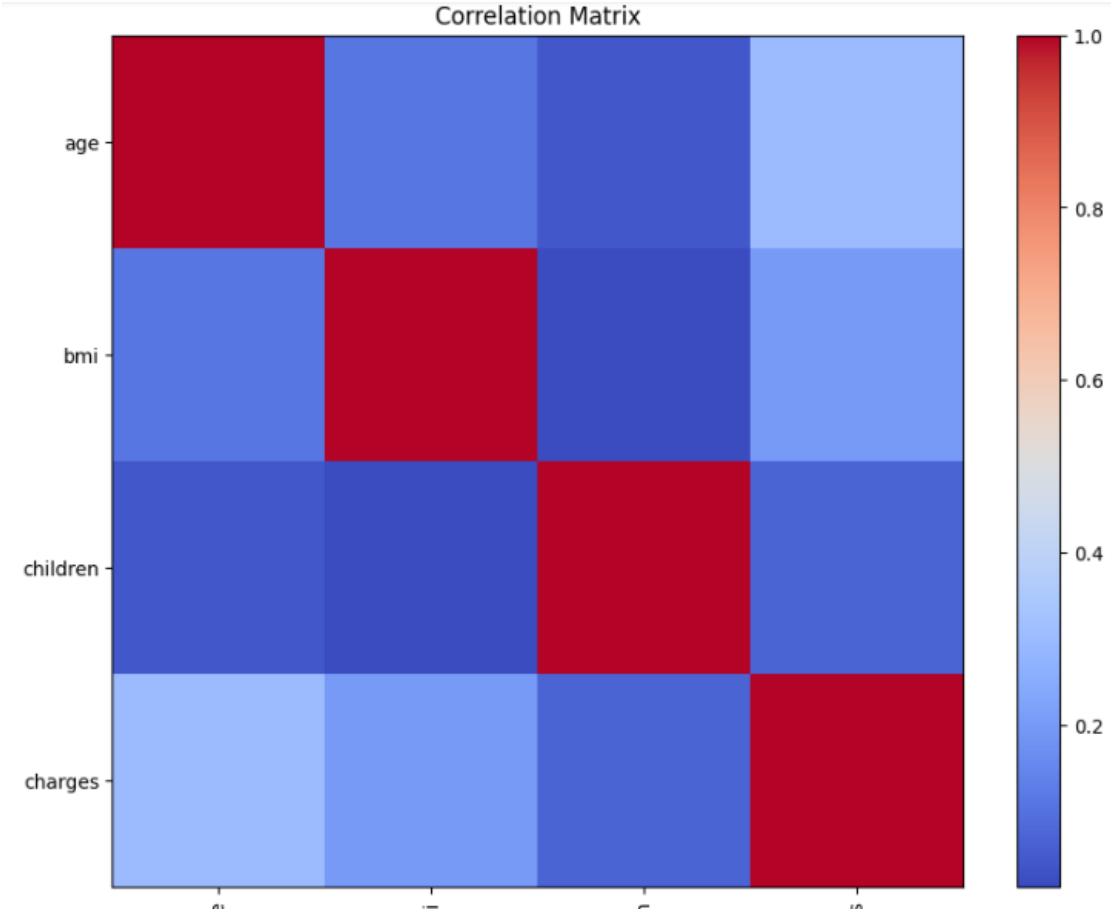
```
    return class_probs, feature_probs

def label_encode(labels):
    unique_labels = np.unique(labels)
    label_dict = {label: i for i, label in enumerate(unique_labels)}
    encoded_labels = np.array([label_dict[label] for label in labels])
    return encoded_labels

def k_nearest_neighbors(x_train, y_train, x_test, k=3):
    # Calculate distances between test samples and training samples
    distances = np.sqrt(np.sum((x_test[:, np.newaxis] - x_train) ** 2, axis=2))

    # Get the indices of the k nearest neighbors
    indices = np.argsort(distances, axis=1)[:, :k]

    # Get the labels of the k nearest neighbors
    y_pred = np.take(y_train, indices)

    # Count the most frequent label in each row
    y_pred = np.apply_along_axis(lambda x: np.bincount(x).argmax(), axis=1, arr=y_pred)

    return y_pred
```

```
# Load the dataset
data = pd.read_csv('Medical Price Dataset.csv')

# Select only the numeric columns for feature scaling
numeric_columns = data.select_dtypes(include=np.number).columns
x_train = data[numeric_columns].values
y_train = data.iloc[:, -1].values

# Perform feature scaling using min-max normalization
x_train = (x_train - np.min(x_train, axis=0)) / (np.max(x_train, axis=0) - np.min(x_train, axis=0))

# Visualize the distribution of target variable
plt.hist(y_train, bins='auto', alpha=0.7)
plt.xlabel('Price')
plt.ylabel('Count')
plt.title('Distribution of Medical Prices')
plt.show()

# Visualize the correlation matrix
corr_matrix = data[numeric_columns].corr()
plt.figure(figsize=(10, 8))
plt.imshow(corr_matrix, cmap='coolwarm', interpolation='nearest')
plt.colorbar()
plt.xticks(range(len(corr_matrix.columns)), corr_matrix.columns, rotation=90)
plt.yticks(range(len(corr_matrix.columns)), corr_matrix.columns)
plt.title('Correlation Matrix')
plt.show()
```

```
# Train the Naive Bayes model
class_probs, feature_probs = naive_bayes(x_train, y_train)

# Label encode the target variable
y_train = label_encode(y_train)

# Train the K-Nearest Neighbors model
k = 3  # Set the number of neighbors
y_pred = k_nearest_neighbors(x_train, y_train, x_train, k)

# Evaluate the accuracy of the K-Nearest Neighbors model
accuracy = np.mean(y_pred == y_train)

# Print the accuracy
print('Accuracy:', accuracy)
```

Results:




Correlation Matrix
age
bmi
children
charges
1.0
0.8
0.6
0.4
0.2

4. Unsupervised Learning on Iris Dataset

The Iris dataset is a popular dataset for unsupervised learning tasks. It consists of measurements of various features of iris flowers. In this part of the project, we applied unsupervised learning algorithms, specifically K-Means Clustering and Principal Component Analysis (PCA), to explore patterns and relationships within the dataset. K-Means Clustering was used to group similar iris flowers based on their feature measurements, while PCA was employed to reduce the dimensionality of the dataset for visualization purposes.

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

def preprocess_data(dataset):
    # Load the dataset
    data = pd.read_csv(dataset)

    # Remove the "Species" column and store it for future comparison
    species = data['Species']
    data = data.drop('Species', axis=1)

    return data, species

def kmeans_clustering(data, k, max_iterations=100):
    # Convert the dataframe to a numpy array
    X = data.values

    # Randomly initialize the centroids
    np.random.seed(0)
    centroids = X[np.random.choice(range(X.shape[0]), k, replace=False)]

    for _ in range(max_iterations):
        # Calculate the distances between each data point and centroids
        distances = np.linalg.norm(X[:, np.newaxis] - centroids, axis=-1)

        # Assign each data point to the closest centroid
        labels = np.argmin(distances, axis=1)

        # Update the centroids based on the mean of the assigned data points
        new_centroids = np.array([X[labels == i].mean(axis=0) for i in range(k)])
```

```
        # Check if the centroids have converged
        if np.all(centroids == new_centroids):
            break

        centroids = new_centroids

    return labels

def pca(data):
    # Center the data
    centered_data = data - data.mean(axis=0)

    # Calculate the covariance matrix
    cov_matrix = np.cov(centered_data.T)

    # Perform eigendecomposition of the covariance matrix
    eigenvalues, eigenvectors = np.linalg.eig(cov_matrix)

    # Sort the eigenvalues and eigenvectors in descending order
    sorted_indices = np.argsort(eigenvalues)[::-1]
    eigenvalues = eigenvalues[sorted_indices]
    eigenvectors = eigenvectors[:, sorted_indices]

    # Project the data onto the first three eigenvectors
    projected_data = centered_data.dot(eigenvectors[:, :3])

    return projected_data, eigenvalues[:3]
```

```
def plot_results(projected_data, cluster_labels, species):
    # Plot the K-Means Clustering output
    fig, axes = plt.subplots(1, 2, figsize=(12, 5))
    fig.suptitle('K-Means Clustering and Principal Component Analysis')

    axes[0].scatter(projected_data.iloc[:, 0], projected_data.iloc[:, 1], c=cluster_labels)
    axes[0].set_xlabel('PC1')
    axes[0].set_ylabel('PC2')
    axes[0].set_title('K-Means Clustering')

    # Plot the actual species
    species_colors = {'Iris-setosa': 'blue', 'Iris-versicolor': 'red', 'Iris-virginica': 'green'}
    species_labels = species.map(species_colors)
    axes[1].scatter(projected_data.iloc[:, 0], projected_data.iloc[:, 1], c=species_labels)
    axes[1].set_xlabel('PC1')
    axes[1].set_ylabel('PC2')
    axes[1].set_title('Actual Species')

    plt.show()
```

```
# Preprocess the data
data, species = preprocess_data('/content/Iris.csv')

# Perform K-Means Clustering
k = 3
cluster_labels = kmeans_clustering(data, k)

# Perform Principal Component Analysis
projected_data, eigenvalues = pca(data)

# Plot the results
plot_results(projected_data, cluster_labels, species)

# Print the eigenvalues for the first three eigenvectors
print('Eigenvalues:', eigenvalues)
```

Results: