Advertising Impact on Sales Prediction

Project Overview

This project aims to build a machine learning model that predicts sales based on the budget allocation across different advertising channels: TV, Radio, and Newspapers. The dataset contains the advertisement spend in each of these media and the corresponding sales output. By analyzing this data, we can identify relationships between the advertisement spend on each medium and its impact on sales, which can help optimize the allocation of advertising budgets.

We will use Python for data handling, visualization, and modeling. The project will involve several key steps, including data cleaning, exploration, preprocessing, model building, evaluation, and interpretation of the results. The primary machine learning algorithm used is the **Random Forest Regressor**, although other algorithms could also be tested.

Table of Contents

---

1. Importing Necessary Libraries

```
python
Copy code
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import PolynomialFeatures
from sklearn.ensemble import RandomForestRegressor
```

Explanation:

- **pandas**: Used for data manipulation and analysis. It provides data structures like DataFrames to handle tabular data efficiently.
- **numpy**: A fundamental package for numerical computations in Python.
- **matplotlib** and **seaborn**: Visualization libraries that help in creating plots and charts for EDA (Exploratory Data Analysis) and model evaluation.
- **scikit-learn**: A machine learning library in Python. We import several modules from this library:
    - **train_test_split**: Splits the dataset into training and testing sets.
    - **LinearRegression**: A simple linear regression model.
    - **mean_squared_error** and **r2_score**: Performance metrics to evaluate the model.
    - **StandardScaler**: Used to scale features to a standard range.
    - **Pipeline**: Helps streamline the machine learning workflow by combining steps.
    - **SimpleImputer**: For handling missing data.
    - **RandomForestRegressor**: The main regression model used to predict sales.

---

2. Loading the Dataset

```
python
Copy code
data = pd.read_csv(r"D:\# DATA SCIENCE\# PROJECTS\- PROJECTS
INTERNSHIPS\CODEALPHA - DATA SCIENCE\Sales Prediction using Python
Project\Advertising.csv")
print(data.head())
```

Explanation:

- **pd.read_csv()**: This function reads the dataset from the CSV file into a pandas DataFrame. The dataset contains advertisement spending for TV, Radio, and Newspapers, along with the resulting Sales.
- **data.head()**: Displays the first five rows of the dataset to get a quick overview of the data.

---

3. Data Cleaning and Exploration

Dataset Information:

```
python
Copy code
print("\nDataset Info:")
print(data.info())
```

- **data.info()**: Provides a concise summary of the dataset, including the number of non-null entries, column names, data types, and memory usage. This is helpful to check for any potential issues, such as missing values or incorrect data types.

Descriptive Statistics:

```
python
Copy code
print("\nDescriptive Statistics:")
print(data.describe())
```

- **data.describe()**: Gives a statistical summary of the dataset, showing the mean, standard deviation, min, and max values for each numerical feature. This is useful to understand the distribution of the data and detect any anomalies.

Checking for Missing Values:

```
python
Copy code
print("\nMissing values in each column:\n", data.isnull().sum())
```

- **data.isnull().sum()**: Checks for missing values in each column. This step ensures that the data is complete before proceeding with model training. Missing data can significantly impact the performance of machine learning models.

---

4. Handling Missing Values

If any missing values are detected, they can be handled in one of two ways:

1. **Dropping rows with missing values:**

```
python
Copy code
data_cleaned = data.dropna()
```

2. **Filling missing values with the mean (or median):**

```
python
Copy code
# data_cleaned = data.fillna(data.mean())
```

Explanation:

- **dropna()**: Removes any rows that contain missing values.
- **fillna()**: Replaces missing values with the mean of the column. In some cases, the median might be a better option, especially if the data is skewed.

---

5. Exploratory Data Analysis (EDA)

Correlation Matrix:

```
python
Copy code
plt.figure(figsize=(8, 6))
sns.heatmap(data_cleaned.corr(), annot=True, cmap='coolwarm', fmt='.2f')
plt.title('Correlation Matrix')
plt.show()
```

- **sns.heatmap()**: Visualizes the correlation between numerical variables in the dataset. This helps in identifying relationships between features and the target variable (Sales). High correlation values indicate strong relationships.
- **annot=True**: Displays the correlation values on the heatmap.
- **cmap='coolwarm'**: The color palette used for the heatmap.

From the correlation matrix, we can determine how much each feature (TV, Radio, Newspaper) correlates with the target variable (Sales). Features with higher correlations are more likely to impact the prediction.

---

6. Feature Selection and Scaling

```
python
Copy code
X = data[['TV', 'Radio', 'Newspaper']]
y = data['Sales']
```

Explanation:

- **X**: The independent variables (features) which include TV, Radio, and Newspaper advertisement budgets.
- **y**: The dependent variable (target), which is Sales.

Scaling the Features:

```
python
Copy code
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
X_scaled_df = pd.DataFrame(X_scaled, columns=X.columns)
```

- **StandardScaler()**: Standardizes the features by removing the mean and scaling to unit variance. This ensures that all features are on the same scale, which is particularly important for algorithms like Random Forest.
- **fit_transform()**: Fits the scaler to the data and then transforms it, standardizing the features.

---

7. Splitting the Data into Training and Testing Sets

```
python
Copy code
```

```
X_train, X_test, y_train, y_test = train_test_split(X_scaled_df, y,
test_size=0.2, random_state=42)
```

Explanation:

- **train_test_split()**: Splits the data into training and testing sets. In this case, 80% of the data is used for training the model, and 20% is reserved for testing.
    - **X_train, y_train**: The training set used to train the model.
    - **X_test, y_test**: The testing set used to evaluate the model's performance.
    - **test_size=0.2**: Allocates 20% of the data for testing.
    - **random_state=42**: Ensures reproducibility by using a fixed seed.

---

8. Model Training

Initializing the Random Forest Regressor:

```
python
Copy code
model = RandomForestRegressor()
model.fit(X_train, y_train)
```

Explanation:

- **RandomForestRegressor()**: The Random Forest algorithm is an ensemble learning method based on decision trees. It aggregates the predictions of multiple decision trees to make more accurate predictions and reduce overfitting.
- **model.fit()**: Fits the model to the training data (X_train and y_train). The Random Forest learns patterns in the data during this step.

---

9. Making Predictions

```
python
Copy code
y_pred = model.predict(X_test)
```

Explanation:

- **model.predict()**: Uses the trained Random Forest model to predict the sales values for the test data (X_test). The predicted values are stored in the variable `y_pred`.

---

10. Evaluating the Model

Evaluation Metrics:

```
python
Copy code
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
print(f"Root Mean Squared Error (RMSE): {rmse}")
print(f"R-squared (R2): {r2}")
```

Explanation:

- **r2_score()**: Calculates the R-squared value, which is a measure of how well the model's predictions match the actual values. An R-squared value close to 1 indicates that the model explains a large portion of the variance in the target variable.
- **mean_squared_error()**: Computes the mean squared error (MSE), which is the average squared difference between the predicted and actual values. The lower the MSE, the better the model.
- **np.sqrt()**: Takes the square root of the MSE to obtain the root mean squared error (RMSE). RMSE provides a more interpretable metric by putting the error in the same units as the target variable (Sales).

---

11. Visualizing the Results

1. Scatter Plot: Actual vs. Predicted Sales

```
python
Copy code
plt.figure(figsize=(10, 6))
plt.scatter(y_test, y_pred, edgecolors=(0, 0, 0), label='Predictions')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'k--',
lw=2, label='Ideal Prediction')
plt.xlabel('Actual Sales')
plt.ylabel('Predicted Sales')
plt.title('Actual vs Predicted Sales')
plt.legend()
plt.show()
```

Explanation:

- **plt.scatter()**: Plots the actual sales values against the predicted sales values. A perfect model would have all points lying on the diagonal line (where actual equals predicted).
- **plt.plot()**: Plots the ideal prediction line (diagonal) for reference. Any deviation from this line indicates the error in predictions.
- **edgecolors**: Specifies the color of the edges of the points.
- **'k--'**: Plots a dashed black line representing the ideal prediction.

2. Residual Plot:

```
python
Copy code
residuals = y_test - y_pred
plt.figure(figsize=(10, 6))
plt.scatter(y_pred, residuals)
plt.axhline(0, color='red', linestyle='--')
plt.xlabel('Predicted Sales')
plt.ylabel('Residuals')
plt.title('Residuals vs Predicted Sales')
plt.show()
```

Explanation:

- **Residuals**: The difference between the actual and predicted values. The residual plot helps to assess the distribution of errors.
- A good model will have residuals scattered randomly around 0, with no clear pattern.

3. Feature Importance Plot:

```
python
Copy code
importances = model.feature_importances_
indices = np.argsort(importances)[::-1]

plt.figure(figsize=(10, 6))
plt.title('Feature Importances')
plt.barh(range(X_scaled_df.shape[1]), importances[indices], align='center')
plt.yticks(range(X_scaled_df.shape[1]),
np.array(X_scaled_df.columns)[indices])
plt.xlabel('Importance')
plt.show()
```

Explanation:

- **model.feature_importances_**: Returns the importance of each feature in making predictions. In a Random Forest, feature importance is determined by how much each feature contributes to reducing the prediction error.
- **np.argsort()**: Sorts the feature importances in descending order, so the most important features are displayed at the top of the plot.

4. Learning Curves:

```
python
Copy code
from sklearn.model_selection import learning_curve

train_sizes, train_scores, test_scores = learning_curve(
    model, X_scaled_df, y, cv=5, scoring='r2',
    train_sizes=np.linspace(0.1, 1.0, 10)
)

train_scores_mean = train_scores.mean(axis=1)
test_scores_mean = test_scores.mean(axis=1)

plt.figure(figsize=(10, 6))
plt.plot(train_sizes, train_scores_mean, 'o-', color='r', label='Training
score')
plt.plot(train_sizes, test_scores_mean, 'o-', color='g', label='Cross-
validation score')
plt.title('Learning Curves')
plt.xlabel('Training Size')
plt.ylabel('R² Score')
plt.legend(loc='best')
plt.grid()
plt.show()
```

Explanation:

- **learning_curve()**: Generates the learning curves for the model by plotting the training score and cross-validation score against the training size.
  - **train_sizes**: The sizes of the training subsets used for generating the learning curve.
  - **train_scores**: The training scores (R-squared values) for different training sizes.
  - **test_scores**: The cross-validation scores for different training sizes.
- **plt.plot()**: Plots the learning curves, showing how the model's performance changes as the amount of training data increases.

Conclusion

In this project, we successfully built a machine learning model to predict sales based on advertisement spending in three different media: TV, Radio, and Newspaper. The project involved several steps:

1. **Data Cleaning and Exploration**: We handled missing values, examined the dataset, and created visualizations to understand the relationships between variables.
2. **Feature Selection and Scaling**: We standardized the features and split the dataset into training and testing sets.
3. **Model Training**: We trained a Random Forest Regressor on the training data and made predictions on the test data.
4. **Model Evaluation**: We evaluated the model using RMSE and R-squared metrics and generated visualizations like scatter plots, residual plots, and learning curves to assess the model's performance.
5. **Feature Importance**: We visualized the importance of each feature in making predictions, helping us understand which media has the most impact on sales.

The model performed well, providing accurate predictions and valuable insights into how different advertisement channels contribute to sales.