Exploratory Data Analysis (EDA): Housing Prices Dataset

Uzair Ahmad

Introduction

In this tutorial, we'll walk through the Exploratory Data Analysis (EDA) process using the `pandas` library on the widely recognized Boston Housing Prices dataset.

Step 1: Setting Up the Environment

First, let's set up the necessary libraries:

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

Step 2: Loading the Dataset

We can load the Boston Housing dataset from the `sklearn` library:

```python
data_url = "http://lib.stat.cmu.edu/datasets/boston"
raw_df = pd.read_csv(data_url, sep="\s+", skiprows=22, header=None)
data = np.hstack([raw_df.values[::2, :], raw_df.values[1::2, :2]])
target = raw_df.values[1::2, 2]
print(data.data.shape)
attributes = ["CRIM","ZN","INDUS", "CHAS", "NOX", "RM", "AGE", "DIS", "RAD"
, "TAX", "PTRATIO" ,"B", "LSTAT"]#, "MEDV"]
df = pd.DataFrame(data, columns=attributes)
df['PRICE'] = target

def assign_age_group(age):
    if age < 25:
        return 'Young'
    elif 25 <= age <= 45:
        return 'Middle-age'
    else:
        return 'Old'

# Apply the function to create the 'age_group' column
df['AGE_GROUP'] = df['AGE'].apply(assign_age_group)
```

Step 3: Initial Data Inspection

Understanding the basic structure and content:

```python
# View the first few rows of the dataset
df.head()
```

```python
# Get information about data types and missing values
df.info()
```

Step 4: Summary Statistics

A statistical summary provides insights about distribution, central tendency, and spread:

```python
df.describe()
```

Step 5: Visualizing the Distribution of Target Variable (`PRICE`)

```python
plt.figure(figsize=(10, 6))
sns.histplot(df['PRICE'], bins=30, kde=True)
plt.title('Distribution of Housing Prices')
plt.xlabel('Price ($1000s)')
plt.ylabel('Frequency')
plt.show()
```

Step 6: Analyzing Relationships

1. **Correlation Matrix**

   To understand linear relationships between variables:

   ```python
   correlation_matrix = df.corr().round(2)
   plt.figure(figsize=(12, 8))
   sns.heatmap(data=correlation_matrix, annot=True, cmap='coolwarm', center=0)
   ```

2. **Scatter Plots**

   Analyze relationships between the most correlated features and the price:

   ```python
   plt.figure(figsize=(20, 5))
   features = ['RM', 'LSTAT']
   for i, col in enumerate(features):
       plt.subplot(1, len(features), i+1)
       x = df[col]
       y = df['PRICE']
       plt.scatter(x, y, marker='o')
       plt.title(col)
       plt.xlabel(col)
       plt.ylabel('PRICE')
   ```

Step 7: Checking for Outliers

Using box plots can help identify outliers:

```python
features = df.columns[:-1]
plt.figure(figsize=(20, 10))
sns.boxplot(data=df[features])
plt.xticks(rotation=45)
plt.title('Box plots to check outliers')
plt.show()
```

Step 8: Normalizing Selected Variables

Normalization is the process of scaling individual samples to have a unit norm. One of the key motivations behind normalization is that algorithms that use gradient descent converge faster with normalized features. Let's see how to normalize features using pandas:

Step: Normalize the 'RM' and 'LSTAT' features:

```python
from sklearn.preprocessing import MinMaxScaler

# Instantiate the scaler
scaler = MinMaxScaler()

# Select columns to normalize
cols_to_normalize = ['RM', 'LSTAT']

# Apply normalization on the selected columns
df[cols_to_normalize] = scaler.fit_transform(df[cols_to_normalize])

# Verify normalization
df[cols_to_normalize].describe()
```

Here, `MinMaxScaler` is used, which scales and translates each feature individually such that it's in the range [0, 1]. However, there are other methods of normalization, like `StandardScaler`, which ensures each feature has mean 0 and variance 1.

Step 9. Handling Categorical Variables

Often datasets contain categorical variables. These variables are typically stored as text values which represent various traits. Machine Learning models require numerical input, so we need to represent these categories in a meaningful numeric way.

Step: Suppose our dataset has a hypothetical column 'HOUSING_TYPE' (which can be 'APARTMENT', 'BUNGALOW', 'CONDO'). Here's how you'd treat this:

1. **One-Hot Encoding**:

   Convert categorical variable(s) into dummy/indicator variables. For each unique value in the categorical variable, a new column is created. The presence of the category is represented by a 1 or 0.

```python
df_with_dummies = pd.get_dummies(df, columns=['AGE_GROUP'],
drop_first=True)
```

By setting `drop_first=True`, it drops one of the categories to avoid multicollinearity. So, if 'AGE_GROUP' had values 'YOUNG', 'Middle-Age', and 'Old', after one-hot encoding, we'd have two new columns, `AGE_GROUP_YOUNG` and `AGE_GROUP_Middle-Age` where 1 represents the presence of that category.

2. **Label Encoding**:

   Assign each unique category in a categorical variable with a number. It's more compact than one-hot encoding but may not be suitable for all types of algorithms as the numbering can introduce ordinality which might not be present.

   ```python
   from sklearn.preprocessing import LabelEncoder

   le = LabelEncoder()
   df['AGE_GROUP'] = le.fit_transform(df['AGE_GROUP'])
   ```

   Here, each unique string value gets a number. For instance, 'YOUNG' might get 0, 'Middle-Age' might get 1, and so on.

When choosing between One-Hot and Label Encoding, consider the algorithm you plan to use. Algorithms that interpret the magnitude of numbers (like regression) might misinterpret label-encoded data. In such cases, one-hot encoding is safer. On the other hand, tree-based algorithms can handle label-encoded data effectively.

Ah, analyzing the target variable is an essential step in EDA, especially when it comes to regression problems. Let's go over how to analyze the target variable for the housing prices dataset.

Step 10: Analyzing the Target Variable: `PRICE`

To gain insights into the target variable, we should study its distribution and other characteristics. Here are the steps:

1. **Distribution Visualization**:

   Using histograms or KDE plots gives an idea of the distribution of house prices.

   ```python
   plt.figure(figsize=(10, 6))
   sns.histplot(df['PRICE'], bins=30, kde=True)
   plt.title('Distribution of Housing Prices')
   plt.xlabel('Price ($1000s)')
   plt.ylabel('Frequency')
   plt.show()
   ```

   Observation: By visualizing, you might notice if the distribution is skewed. If the distribution has a long tail on the right, it's right-skewed. If it's on the left, it's left-skewed.

2. **Descriptive Statistics**:

It's useful to know the mean, median, minimum, maximum, and other statistics for the target variable.

```
df['PRICE'].describe()
```

Observation: Check for any anomalies. For example, if the max value is unreasonably higher than the 75th percentile, it might indicate outliers.

3. **Box Plot**:

   Box plots allow us to detect outliers in the data.

```
plt.figure(figsize=(6, 4))
sns.boxplot(df['PRICE'])
plt.title('Box plot of Housing Prices')
plt.xlabel('Price ($1000s)')
plt.show()
```

   Observation: If there are any dots (or circles) beyond the whiskers of the box plot, they are potential outliers.

4. **Relationship with Features**:

   Scatter plots of the target against some main features can help in understanding their relationship.

```
plt.scatter(df['RM'], df['PRICE'])
plt.title('Price vs Average Number of Rooms')
plt.xlabel('Average Number of Rooms per Dwelling')
plt.ylabel('Price ($1000s)')
plt.show()
```

   Observation: You might notice a trend. For instance, as the average number of rooms (`RM`) increases, the house price might also increase, indicating a positive correlation.

5. **Skewness and Kurtosis**:

   Skewness gives information about the direction of skew (departure from horizontal symmetry) and kurtosis can identify the tails' heaviness and peakiness.

```
skewness = df['PRICE'].skew()
kurtosis = df['PRICE'].kurtosis()

print(f"Skewness: {skewness}")
print(f"Kurtosis: {kurtosis}")
```

   Observation: A skewness value > 0 means that there's more weight in the left tail of the distribution. Kurtosis > 0 indicates that the tails are heavier than a normal distribution (more prone to outliers).

Incorporating this analysis gives a robust understanding of the target variable, ensuring you're aware of its properties before moving on to modeling.

Treatment of Skewness and Kurtosis

Understanding the skewness and kurtosis of a distribution provides insights into the shape of the distribution, and depending on the values, there are various treatments you might consider. Let's guide you through the possible treatments based on these observations:

Skewness:

Skewness measures the asymmetry of the probability distribution of a real-valued random variable.

1. **Positive Skewness**:
   - Means when the tail on the right side of the distribution is longer or fatter than the left side.
   - Potential treatments:
     - **Log transformation**: `df['PRICE'] = np.log(df['PRICE'])`. This is often used for right-skewed data. It compresses the data on the higher side, making the distribution more symmetric.
     - **Square root transformation**: `df['PRICE'] = np.sqrt(df['PRICE'])`. This is a milder transformation than the log transformation.
     - **Box-Cox transformation**: This requires `from scipy.stats import boxcox`. It transforms data into a normal shape.
2. **Negative Skewness**:
   - Means when the tail on the left side of the distribution is longer or fatter than the right side.
   - Potential treatments:
     - **Power transformations**: Like squaring the values.
     - **Box-Cox**: This can also be used for negative skewness.

Note: Before applying transformations, ensure that the data does not have negative or zero values, especially for log or square root transformations.

Kurtosis:

Kurtosis measures the "tailedness" of the probability distribution of a real-valued random variable.

1. **High Kurtosis (Leptokurtic)**:
   - Means the distribution has heavier tails and a sharper peak than the normal distribution.
   - It suggests potential presence of outliers.
   - Potential treatments:
     - **Truncation or capping**: Values which are too far from the mean can be set to a maximum threshold.
     - **Winsorization**: Similar to truncation but replaces the extreme values with certain percentiles (e.g., 1st or 99th percentile) rather than completely removing them.
2. **Low Kurtosis (Platykurtic)**:
   - Means the distribution is flat and has a peak lower than the normal distribution.

    - This isn't usually a problem for many statistical techniques. However, the data might not be as informative as one hopes.

General Guiding Comments:

- Always visualize the distribution before and after applying transformations to ensure the transformation had the desired effect.
- Sometimes a combination of transformations is necessary to achieve normality or approximate symmetry.
- Understand the implications of transformations on the model interpretability. For instance, after log transformation, the interpretation of the coefficients (in regression, for instance) changes.
- If using machine learning models, many models (like tree-based models) don't require the target or features to be normally distributed. But for linear models, the assumptions about error normality can be crucial, so paying attention to the distribution of residuals is also key.
- Lastly, it's essential to remember the original business problem and context. If a transformation makes the results uninterpretable or doesn't make sense in the given context, consider other modeling strategies or consult with domain experts.

Step 11: Further Investigations

1. **Feature Engineering**: Deriving new features or transforming existing ones can sometimes improve model performance. For example, using polynomial features for variables that aren't linearly related to the target.
2. **Handling Outliers**: Based on the box plots, one can decide whether to remove outliers or cap them to a specific value.
3. **Normalizing Data**: If features have different scales, consider normalization or standardization.

Step 12: Final Thoughts and Next Steps

After the EDA:

1. Consider splitting the data into training and test sets and building a regression model.
2. Use techniques like cross-validation for model evaluation.
3. Iterate on your EDA and modeling steps as you gain more insights.

Conclusion

EDA is a crucial preliminary step before diving into model building. It helps in understanding underlying patterns, relationships, and potential issues within the dataset. Using `pandas` and visualization libraries like `matplotlib` and `seaborn` makes this process insightful and straightforward.