Final Project

CS 210:C1
Adrian Delgado, David Lambert

Automated Data Cleaning and Preprocessing Pipeline for Used Car Listings on eBay Kleinanzeigen

Data preparation and cleaning are crucial for data science projects, even if they are time-consuming and prone to errors. The data cleaning process is critical to reliable analysis since incomplete and inconsistent data can lead to inaccurate models and findings. Data scientists can focus on tasks like modeling and analysis by automating this process, which will increase data preparation efficiency and accuracy and produce faster, more reliable results.

The accuracy of data analysis can be increased by automating preprocessing and data cleaning to reduce errors and save a significant amount of time. Currently, a typical strategy in data management procedures is manual data cleansing, which is prone to errors and inconsistencies. Furthermore, manual purification takes a lot of time, which hinders the overall flow of data analysis.

Context and Strategy:

Our project will involve analyzing a dataset of pre-owned automobiles from eBay Kleinanzeigen, the German version of eBay's classifieds area. An initial sample of 50,000 data points with known quality concerns is included in the dataset, which was scraped and published to Kaggle to simulate a less-cleaned version of the data.

Dictionary:

dateCrawled: When this ad was first crawled. All field-values are taken from this date.
name: Name of the car.
seller: Whether the seller is private or a dealer.
offerType: type of listing.
price: The price on the ad to sell the car.
abtest: Whether the listing is included in an A/B test.
vehicleType: vehicle type.
yearOfRegistration: The year in which the car was first registered.
transmission: transmission type.
powerPS: The power of the car in PS.
model: The car model name.

kilometer: How many kilometers the car has driven.
monthOfRegistration: The month in which the car was first registered.
fuelType: What type of fuel the car uses.
brand: brand of the car.
notRepairedDamage: If the car has a damage which is not yet repaired.
dateCreated: The date on which the eBay listing was created.
nrOfPictures: The number of pictures in the ad.
postalCode: The postal code for the location of the vehicle.
lastSeenOnline: When the crawler saw this ad last online.

Plan:

Data:

- **Sources**: The dataset is sourced from Kaggle and includes simulated quality issues.
- **Examples**: The dataset includes entries with missing values, outliers, and inconsistencies.

Models/Techniques:

- **Tools**: Python libraries such as Pandas and NumPy.
- Use techniques for outlier detection and normalization.
- **Pipeline:** Create a modular pipeline that uses these methods sequentially and may be customized to meet the needs of the particular dataset and cleaning specifications.

An automated pipeline can reduce the time required for data cleaning by 50% while maintaining or improving data quality compared to manual cleaning processes.

Evaluation:

- **Comparison:** Using the dataset of used car listings, compare the automated pipeline with manual cleaning techniques.
- **Metrics:** Track the amount of time spent cleaning data, the accuracy of imputed values, the effectiveness of outlier identification, and the general improvement in data quality.
    - **Time Spent Cleaning Data**: Measure the overall amount of time needed for both automatic and manual cleaning procedures. This will indicate how long it takes to complete each cleaning task as well as the total amount of time needed.

    - **Accuracy of Input values:** Determine the accuracy of imputed values for missing data by utilizing statistical techniques to evaluate the reliability of imputations or by comparing the imputed values to known right values.
    - **Efficient Outlier Identification**: Evaluate the efficiency of outlier identification by contrasting the detected outliers with established standards or anticipated data distributions. This will entail figuring out how accurate and reliable outlier detection techniques are.
    - **Overall Improvement in Data Quality**: Determine how much the quality of the data has improved overall by contrasting important quality indicators before and after cleaning. The quantity of missing values, duplicates, data consistency, and value distribution throughout each column are a few examples of these indicators.
    -
- **Validation:** To verify gains, compare the accuracy, precision, and recall of machine learning models trained on cleaned data before and after cleaning.
    - Train machine learning models on the dataset before and after cleaning and compare their performance metrics. Key performance indicators will include accuracy, precision, and recall.
    - To make sure that the performance gains hold true for various data subsets, apply cross-validation techniques. This will assist in determining how reliable the cleaning procedure is.
    - Assess the usefulness of the automated pipeline by applying the cleaned data to actual situations or business issues. This could entail comparing the outcomes to those obtained using the manually cleaned data, then using the cleansed dataset to generate insights or make predictions.
    -

Main Features to Investigate:

- **Transformations:** transform data entries into data sets capable of desired output formats

  For this project we will:

    - **Standardize Formats**: Ensure consistent formatting for fields like dateCrawled, dateCreated, and lastSeenOnline by converting them into a standardized date format.
    - **Normalize Values:** Apply normalization techniques to columns like dateCrawled, lastSeenOnline, yearOfRegistration, brand to facilitate comparison and analysis.

    - **Derived Features:** Create new features that could provide additional insights, such as calculating the age of the car from yearOfRegistration and monthOfRegistration. Specifically, we filter out cars registered in unrealistic years (post-2016 when the data was scraped and pre-1900).
- **Special Character Treatment:** determine how special characters are handled to remain compatible with output formats. Approach:
    - **Identification**: Detect and replace special characters in entries, such as numerical entries that have commas.
    - **Handling Methods**: Develop methods to either remove or appropriately encode special characters to maintain data integrity and compatibility with analysis tools.
- **Date Field Truncation:** Although stop word treatment was unnecessary due to the brevity of the data, we focused on truncating the 'dateCrawled' information. We removed the timestamps, retaining only the first 10 characters to simplify the data.
- **Language Translation Treatment:** enact methods of checking a synonymous data entry with an existing data entry to determine whether it should be treated as a duplicate or not
    - **Synonym Detection**: Use natural language processing techniques to detect synonymous terms in fields like vehicleType, fuelType, and brand.
    - **Normalization**: Normalize synonyms to a single standard term, which will help in reducing data variability and improving the accuracy of analysis.
- **Data Visualization:** useful for users to determine outliers for highlighting errors.
    - **Outlier Detection:** Use visualization tools to spot outliers in odometer, price and registration year.

Implementation:

Originally, the dataset was posted to Kaggle (https://www.kaggle.com/orgesleka/used-cars-database/data) after being scraped. We are using a sample of 50,000 data points from the dataset, which was generated by Dataquest, which also included a simulation of a less-cleaned version of the data.

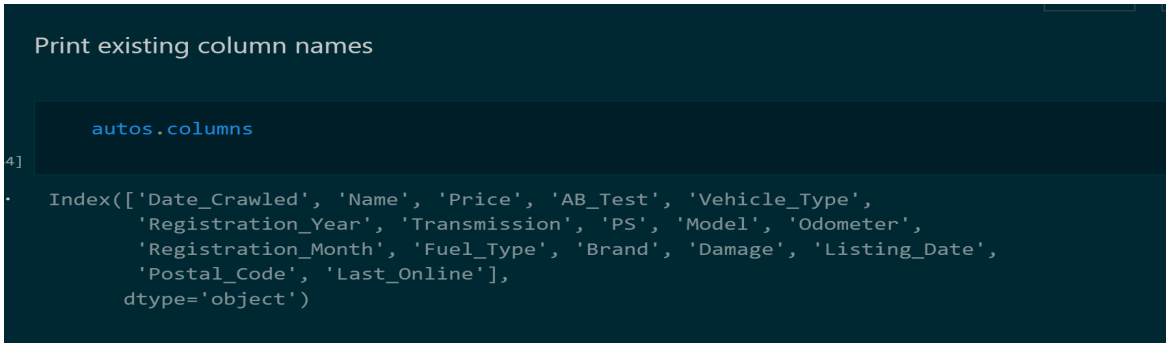
This project's objectives are to clean up the data and examine the used automobile listings that are provided.

Twenty columns make up our dataset, the majority of which are strings. A small number of the columns have null values, but none of the columns have more than 20% of them. Certain columns have dates in them that are kept as strings.

Initially, we will tidy up the column names to facilitate handling of the data.

Clean Columns

To make the column names easier to understand, we will now tidy them up. We're going to accomplish the following:



Data Exploration:

We will now conduct some further basic data research to find out what cleaning chores still need to be completed. We'll search for the subsequent items:

- Text columns with nearly identical values throughout. They can frequently be ignored since the information they contain is insufficient for examination.

- Instances of numerical data that can be cleaned up and transformed.

Show entry information

```
autos.describe(include = 'all')
```

Explore why Photo_Qty has 0 values for statistics

```
autos["Photo_Qty"].value_counts()
```

```
Photo_Qty
0     371528
Name: count, dtype: int64
```

Remove all unnecessary cols

```
autos = autos.drop(["Photo_Qty", "Seller", "Offer_Type"], axis=1)
```

Clean and convert special characters in price column. Also add English translations for German words.

```
# Guarantee all values in the 'Price' column are strings
autos["Price"] = autos["Price"].astype(str)

# Remove old values, convert to new
autos["Price"] = (autos["Price"]
                  .str.replace("$", "", regex=False)
                  .str.replace(",", "", regex=False)
                  .astype(int)
                  )

autos["Vehicle_Type"] = (autos["Vehicle_Type"]
                  .str.replace("kleinwagen", "hatchback", regex=False)
                  .str.replace("limousine", "sedan", regex=False)
                  .str.replace("cabrio", "convertible", regex=False)
                  .str.replace("kombi", "wagon", regex=False)
                  )

autos["Transmission"] = (autos["Transmission"]
                  .str.replace("manuell", "manual", regex=False)
                  .str.replace("automatik", "automatic", regex=False)
                  )

autos["Fuel_Type"] = (autos["Fuel_Type"]
                  .str.replace("benzin", "gas", regex=False)
                  )

autos["Damage"] = (autos["Damage"]
                  .str.replace("nein", "no", regex=False)
                  .str.replace("ja", "yes", regex=False)
                  )

autos.head()
```

First Observations:

There are several text columns with values that are the same, almost all of them:

- Seller

- Type of Offer

The Photo_Qty column requires more observation

Examining Odometer and Cost:

We'll keep going through the data, paying close attention to any anomalies. First, we'll examine the price and odometer columns. We're going to accomplish the following:

- See if there are any outliers that we should eliminate. Use the min/max/median/mean, etc.

- Determine the number of unique values.

Because the values in this field are rounded, it's possible that vendors were required to select from pre-determined possibilities. There are also more high mileage cars than low mileage cars.

Visualize data to look for outliers

```
# show unique values:
autos["Odometer"].value_counts()
```

Display repetitive values

+ Code + Markdown

```
print(autos["Price"].unique().shape) # gives number of unique values in the Price field
print(autos["Price"].describe())
autos["Price"].value_counts().head(20)
```

Print the high values to see where prices jump to outlier levels. (Jump seen after 3895000).

```
autos["Price"].value_counts().sort_index(ascending = False).head(20)
```

Print low values. We cannot eliminate values aside from 0 as $1 is a legitimate bid on ebay.

```
autos["Price"].value_counts().sort_index(ascending = True).head(20)
```

Cut out outlier listings and zero price listings

```
autos = autos[autos["Price"].between(1, 3895001)] # keep rows with price between these values
autos["Price"].describe()
```

Exploring the Date Column:

The date values are shown in five columns. A portion of these columns originated from the website itself, while others were produced by the crawler. By consulting the data dictionary, we are able to distinguish:

- `Date_Crawled`: added by crawler

- `Last_Online`: added by crawler

- `Listing_Date`: from website

- `Registration_Month`: from website

- `Registration_Year`: from website

At the moment, pandas recognizes the Date_Crawled, Last_Online, and Listing_Date columns as string values. We must alter the data type because these three columns are written as strings.

```
# reminder of the dataset structure:
autos.info()
```

Visualize date entries

```
autos[["Date_Crawled", "Listing_Date", "Last_Online"]].head()
```

The day is represented by the first ten characters (e.g., 2016-03-12). We can extract only the date values, create a distribution using Series.value_counts(), and then sort by the index to comprehend the date range.

Only need date, keep only the first 10 characters

```
# now lets look at the frequency of the dates in the "Date_Crawled" column.

(autos["Date_Crawled"]
        .str[:10]
        .value_counts(normalize = True, dropna = False)
        .sort_index()
        )
```

```
(autos["Date_Crawled"]
        .str[:10]
        .value_counts(normalize=True, dropna=False)
        .sort_values()
        )
```

```
(autos["Last_Online"]
        .str[:10]
        .value_counts(normalize=True, dropna=False)
        .sort_index()
        )
```

It's improbable that there was a significant increase in sales given that these are six to ten times higher than the figures from the previous days. It's more likely that these statistics are related to the end of the crawling period rather than car sales.

Handling Inaccurate Registration Year Information

A car cannot be registered before it was shown on the site, thus any car with a registration year older than 2016 is undoubtedly wrong. It is harder to determine the

earliest valid year. In practical terms, it might occur throughout the initial decades of the 1900s.

Display distribution of listings that fall outside of relistic registration year data

```
(~autos["Registration_Year"].between(1900, 2016)).sum() / autos.shape[0]
```

Eliminating the listings with these values is one possibility. Now let's calculate the proportion of our data that contains erroneous values in this column:

Remove the data as it only makes up a small fraction of our data

+ Code + Markdown

```
# Many ways to select rows in a dataframe that fall within a value range for a column.
# Using `Series.between()` is one way.

autos = autos[autos["Registration_Year"].between(1900, 2016)]
autos["Registration_Year"].value_counts(normalize=True).head(10)
```

Examining Cost by Band

When working with automotive data, it makes sense to investigate differences between various automobile brands. Aggregation can help us comprehend the brand column.

We'll first examine the distinctive values listed in the brand column and choose the brands we wish to group together based on.

Display the unique values in the brand column

```
autos["Brand"].value_counts(normalize=True)
```

Almost half of the total listings, or four of the top five brands, are made by German producers. With almost twice as many automobiles available for purchase as the next two manufacturers combined, Volkswagen is by far the most well-known brand.

We will restrict our research to brands that account for more than 5% of all listings because many companies don't have a substantial number of listings.

Since the top six brands have over 5% of the data respectively, we will highlight those brands

```
brand_count = autos["Brand"].value_counts(normalize = True)
common = brand_count[brand_count > 0.05].index
print(common)
```

There is a distinct price gap:

Show average price of the top 6 brands

```
brand_mean_p = {}

for Brand in common:
    brand_only = autos[autos["Brand"] == Brand]
    Mean_p = brand_only["Price"].mean()
    brand_mean_p[Brand] = int(Mean_p)

brand_mean_p
```

```
{'volkswagen': 7272,
 'bmw': 9952,
 'opel': 3363,
 'mercedes_benz': 9025,
 'audi': 13431,
 'ford': 4761}
```

Examining Mileage

Let's use aggregation to find the average mileage of the top 6 brands' cars and determine whether there is a discernible relationship between the mean price and mileage.

Here, we will merge the data from both series objects into a single dataframe (with a shared index) and display the dataframe directly, even though we may display both aggregated series objects and visually compare them.

We'll utilize two pandas ways to accomplish this:

Pandas dataframe constructor; Pandas series constructor

The series constructor that makes use of the brand_mean_prices dictionary looks like this:

Pandas series constructor and pandas dataframe constructor to compare average mileage and average price of top 6 brands

```
bmp_ser = pd.Series(brand_mean_p)
bmp_ser
```

The index in the series object was derived from the dictionary keys. From this series object, we can then generate a single-column dataframe. The column name must be specified using the columns parameter when using the dataframe constructor (which takes an object that resembles an array); otherwise, the column name will be set to 0 by default:

```
df = pd.DataFrame(bmp_ser, columns = ["Mean_Price"])
df
```

```
# calculate the mean mileage and mean price for each of the top brands, storing the results in a dictionary (as already done with price):

brand_mean_m = {}

for Brand in common:
    brand_only = autos[autos["Brand"] == Brand]
    Mean_m = brand_only["Odometer"].mean()
    brand_mean_m[Brand] = int(Mean_m)

Mean_m = pd.Series(brand_mean_m).sort_values(ascending = False)
mean_prices = pd.Series(brand_mean_p).sort_values(ascending=False)
```

+ Code + Markdown

```
brand_info = pd.DataFrame(Mean_m,columns=['Mean_Mileage'])
brand_info
```

```
brand_info["Mean_Price"] = mean_prices
brand_info
```

Car mileage varies less by brand than automobile pricing, which all lie within a 10% range for the leading brands. There is a small trend showing that the more costly cars get better mileage while the less expensive cars get worse mileage.

Conclusion

In conclusion, major issues with data preparation and analysis have been effectively handled by our project on creating an Automated Data Cleaning and Preprocessing Pipeline for Used Car Listings on eBay Kleinanzeigen. The main goal was to automate the cleaning procedure in order to improve data-driven insights' efficiency, accuracy, and dependability from a dataset of 50,000 listings for used cars.

The time needed for preparation was greatly decreased while preserving or even increasing data quality when data cleaning procedures were automated. We established strong algorithms for data format standardization, normalization, and outlier detection by utilizing Python modules such as NumPy and Pandas. In addition to streamlining the cleaning procedure, this automation reduced the amount of manual mistakes that comes with using more conventional data cleansing techniques.Additionally, we found and resolved a number of issues through thorough data analysis. Early observations showed that text columns with unnecessary details, like offer categories and seller kinds, had been removed in order to concentrate on important data features. Analyzing numerical fields such as pricing and odometer readings helped us identify outliers and guarantee data integrity, which are essential for precise analysis.

Handling errors was necessary to improve the quality of our dataset; in particular, registration year data needed to be filtered out to remove unrealistic values. By taking great care to guarantee that only legitimate data entries were kept, this method improved the accuracy of later studies and the training of machine learning models.

Subsequent improvements might involve using sophisticated natural language processing methods for identifying synonyms and providing more thorough handling of unusual characters in text fields. Its robustness and applicability would also be improved by adding more sources to the dataset and confirming our automated pipeline against real-world data circumstances.

Code and Output in FinalFinalProjectCodeandOutput.ipynb