preprocessing

October 22, 2024

[1]:
```
# Load necessary libraries
import pandas as pd
import numpy as np
import json
```

1 Data Ingestion

[2]:
```
cards_csv = pd.read_csv('../dataset/cards.csv', sep=";")
prices_csv = pd.read_csv('../dataset/cardPrices.csv', sep=",")

print(f"Cards Dataset length = {len(cards_csv)}")
print(f"Prices Dataset length = {len(prices_csv)} \n")

print(f"Cards Columns Num = {cards_csv.shape[1]}")
print(f"Price Columns Num = {prices_csv.shape[1]}\n")

# number of unique dataset instances by uuid
card_unique = cards_csv['uuid'].nunique()
price_unique = prices_csv['uuid'].nunique()

print(f"Cards length by UUID = {card_unique}")
print(f"Prices length by UUID= {price_unique}\n")

dates = prices_csv['date'].nunique()
date = prices_csv['date'][0]
print(f"Num of unique dates = {dates}")
print(f"Price Date = {date}")

#print(cards_csv.nunique())
#print(prices_csv.nunique())
```

```
Cards Dataset length = 97145
Prices Dataset length = 558079

Cards Columns Num = 25
Price Columns Num = 8
```

```
Cards length by UUID = 97145
Prices length by UUID= 91302

Num of unique dates = 1
Price Date = 2024-09-20
```

[3]:
```python
cards = cards_csv
prices = prices_csv

# Drop columns that are duplicates
cards = cards.drop(columns=['finishes'])
cards = cards.drop(columns=['hasFoil'])
cards = cards.drop(columns=['hasNonFoil'])
cards = cards.drop(columns=['sourceProducts'])

# Drop constant columns
prices = prices[prices['currency'] == 'USD']
prices = prices.drop(columns=['currency'])
prices = prices.drop(columns=['date'])

# Standardize colors and colorIdentity
def unique_colors(value):
    if pd.isna(value):
        return None  # Return None for NaN
    # Split the value, standardize, and get unique characters
    characters = set(', '.join(sorted(value.split(', '))).replace(', ', ''))
    return ''.join(sorted(characters))

# Apply the function to the columns
cards['colors'] = cards['colors'].apply(unique_colors)
cards['colorIdentity'] = cards['colorIdentity'].apply(unique_colors)

# replace NaN values with False
cards.loc[cards['isReprint'].isna(), "isReprint"] = False
# Map True to 1 and False to 0
cards['isReprint'] = cards['isReprint'].astype(int)
```

[4]:
```python
# cards2 is going to be used in secondary
cards2 = cards.copy()

cards2['colors'] = cards2['colors'].fillna('C')
cards2['colorIdentity'] = cards2['colorIdentity'].fillna('C')

cards2['originalType'] = cards2['originalType'].fillna('None')
cards2['power'] = cards2['power'].fillna('None')
cards2['supertypes'] = cards2['supertypes'].fillna('None')
cards2['toughness'] = cards2['toughness'].fillna('None')
```

[5]:
```
# perform inner join based on UUID, then drop null values
primary = pd.merge(prices, cards, on="uuid")
primary = primary.dropna().reset_index(drop=True)
print(f"Primary dataset by uuid (cards with prices) size: {len(primary)}")


secondary = pd.merge(prices, cards2, on="uuid")
secondary = secondary.dropna().reset_index(drop=True)
print(f"Seconday dataset by uuid (cards with prices) size: {len(secondary)}")
```

```
Primary dataset by uuid (cards with prices) size: 17628
Seconday dataset by uuid (cards with prices) size: 269807
```

[6]:
```
# Remove outliers by prices
def removeOutliers(data, col):
    Q3 = np.quantile(data[col], 0.75)
    Q1 = np.quantile(data[col], 0.25)
    IQR = Q3 - Q1

    lower_range = Q1 - 1.5 * IQR
    upper_range = Q3 + 1.5 * IQR
    outlier_free_list = [x for x in data[col] if ((x > lower_range) & (x <␣
 ↪upper_range))]
    print(f"Num outliers: {len(data) - len(outlier_free_list)}")
    filtered_data = data.loc[data[col].isin(outlier_free_list)]
    return filtered_data

# data no outliers
primary = removeOutliers(primary, 'price')
secondary = removeOutliers(secondary, 'price')
```

```
Num outliers: 2414
```

```
Num outliers: 41170
```

[7]:
```
def map_categorical(data):
    numerical_data = pd.DataFrame()
    categorical_data = pd.DataFrame()
    mapping_dict = {}

    # columns to encode
    to_encode = ['rarity', 'artist', 'cardFinish', 'supertypes', 'type',
                 'gameAvailability', 'priceProvider', 'setCode', 'types',
                 'colors', 'colorIdentity', 'layout', 'language',
                 'power', 'toughness', 'manaCost', 'name', 'number',
                 'originalType']
    # non encoded columns uuid (54855)
```

```
    # first map binary data
    for col in data.columns:
        if data[col].dtype == 'float64' or data[col].dtype == 'int64':
            numerical_data[col] = data[col]
        elif data[col].dtype == 'object' or isinstance(data[col].dtype, pd.
 ↪CategoricalDtype):
            unique_values = data[col].unique()
            if len(unique_values) == 2:
                # creating mapping to use later
                mapping = {unique_values[0]: 0, unique_values[1]: 1}
                mapping_dict[col] = mapping

                # Replace the values in the binary DataFrame
                numerical_data[col] = data[col].map(mapping)
            else:
                categorical_data[col] = data[col]
        else:
            categorical_data[col] = data[col]

    # map other categorical columns
    for col in categorical_data.columns:
        if col in to_encode:
            numerical_data[col] = categorical_data[col].astype('category').cat.
 ↪codes
            mapping_dict[col] = {category: code for category, code in␣
 ↪zip(categorical_data[col].astype('category').cat.categories,␣
 ↪range(len(categorical_data[col].astype('category').cat.categories)))}
            categorical_data = categorical_data.drop(columns=[col])

        if col == 'uuid':
            numerical_data[col] = categorical_data[col]

    return numerical_data, categorical_data, mapping_dict
```

[8]:
```
numerical_primary, categorical_primary, mapping_primary =␣
 ↪map_categorical(primary)
numerical_secondary, categorical_secondary, mapping_secondary =␣
 ↪map_categorical(secondary)

print(f"Primary Columns mapped example: {mapping_primary['rarity']}")
print(f"Secondary columns mapped example: {mapping_secondary['rarity']}\n")

print(f"Primary Categorical data excluded: {categorical_primary.columns.
 ↪tolist()}")
print(f"Secondary Categorical data excluded: {categorical_secondary.columns.
 ↪tolist()}")
```

```
Primary Columns mapped example: {'common': 0, 'mythic': 1, 'rare': 2, 'special':
3, 'uncommon': 4}
Secondary columns mapped example: {'bonus': 0, 'common': 1, 'mythic': 2, 'rare':
3, 'special': 4, 'uncommon': 5}

Primary Categorical data excluded: ['uuid']
Secondary Categorical data excluded: ['uuid']
```

[9]:
```
# sort dataframes
order = ['price'] + sorted([col for col in numerical_primary.columns if col !=␣
 ↪'price'])
numerical_primary = numerical_primary[order]
numerical_secondary = numerical_secondary[order]
numerical_primary = numerical_primary.drop(columns=['language'])

# save out mapped data + uuid to same file
numerical_primary.to_csv('../dataset/mapped_primary.csv', index = False)
numerical_secondary.to_csv('../dataset/mapped_secondary.csv', index = False)

json.dump(mapping_primary, open('../dataset/mapping_primary.json', 'w'))
json.dump(mapping_secondary, open('../dataset/mapping_secondary.json', 'w'))
```

[10]:
```
print(f"Primary Dataset length = {len(numerical_primary)}")
print(f"Secondary Dataset length = {len(numerical_secondary)} \n")

print(f"Primary Columns Num = {numerical_primary.shape[1]}")
print(f"Secondary Columns Num = {numerical_secondary.shape[1]}\n")

# number of unique dataset instances by uuid
primary_unique = numerical_primary['uuid'].nunique()
secondary_unique = numerical_secondary['uuid'].nunique()

print(f"Primary length by UUID = {primary_unique}")
print(f"Secondary length by UUID= {secondary_unique}\n")
```

```
Primary Dataset length = 15214
Secondary Dataset length = 228637

Primary Columns Num = 25
Secondary Columns Num = 26

Primary length by UUID = 3123
Secondary length by UUID= 50208
```

[11]:
```
print(f"Primary Columns Num = {numerical_primary.columns.tolist()}")
print(f"Secondary Columns Num = {numerical_secondary.columns.tolist()}")
```

```
Primary Columns Num = ['price', 'artist', 'cardFinish', 'colorIdentity',
```

```
'colors', 'edhrecRank', 'edhrecSaltiness', 'gameAvailability', 'isReprint',
'layout', 'manaCost', 'manaValue', 'name', 'number', 'originalType', 'power',
'priceProvider', 'providerListing', 'rarity', 'setCode', 'supertypes',
'toughness', 'type', 'types', 'uuid']
Secondary Columns Num = ['price', 'artist', 'cardFinish', 'colorIdentity',
'colors', 'edhrecRank', 'edhrecSaltiness', 'gameAvailability', 'isReprint',
'language', 'layout', 'manaCost', 'manaValue', 'name', 'number', 'originalType',
'power', 'priceProvider', 'providerListing', 'rarity', 'setCode', 'supertypes',
'toughness', 'type', 'types', 'uuid']
```