eda_secondary

October 22, 2024

[1]:
```
# Load necessary libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import seaborn as sns
from matplotlib import gridspec
import math
import json
```

0.1 Data Analysis on Secondary

[2]:
```
# set this to either primary or secondary depending on the analysis you want to␣
↪do
dataset = 'secondary'

mappings = json.load(open(f'../dataset/mapping_{dataset}.json'))
data = pd.read_csv(f'../dataset/mapped_{dataset}.csv')
```

[3]:
```
# drop the uuid column as it isn't a quantitative value
data = data.select_dtypes(include=['number'])
print(data.describe())
```

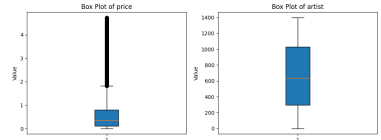
```
              price         artist     cardFinish  colorIdentity  \
count  228637.000000  228637.000000  228637.000000  228637.000000
mean        0.739998     674.922978       1.694048      19.461102
std         1.007018     402.720325       0.476893      10.588725
min         0.010000       0.000000       0.000000       0.000000
25%         0.120000     298.000000       1.000000      16.000000
50%         0.350000     637.000000       2.000000      24.000000
75%         0.800000    1027.000000       2.000000      29.000000
max         4.730000    1398.000000       2.000000      31.000000

              colors     edhrecRank  edhrecSaltiness  gameAvailability  \
count  228637.000000  228637.000000    228637.000000     228637.000000
mean       19.517974   10295.519859         0.293477          0.807083
std        10.532966    7913.381637         0.263940          0.394589
min         0.000000       1.000000         0.010000          0.000000
```

```
25%        16.000000    3332.000000          0.120000          1.000000
50%        24.000000    8549.000000          0.210000          1.000000
75%        29.000000   16490.000000          0.380000          1.000000
max        31.000000   28033.000000          2.990000          1.000000

           isReprint       language  …   originalType          power  \
count  228637.000000  228637.000000  …  228637.000000  228637.000000
mean        0.489632       2.010493  …    1508.449555      14.970066
std         0.499894       0.209510  …     731.337659       5.930958
min         0.000000       0.000000  …       0.000000       0.000000
25%         0.000000       2.000000  …     999.000000      12.000000
50%         0.000000       2.000000  …    1555.000000      16.000000
75%         1.000000       2.000000  …    2180.000000      20.000000
max         1.000000       9.000000  …    2563.000000      20.000000

       priceProvider  providerListing         rarity        setCode  \
count  228637.000000    228637.000000  228637.000000  228637.000000
mean        1.508553         0.141753       2.956630     218.587193
std         1.084295         0.348797       1.506641     159.196328
min         0.000000         0.000000       0.000000       0.000000
25%         1.000000         0.000000       1.000000      67.000000
50%         1.000000         0.000000       3.000000     190.000000
75%         3.000000         0.000000       5.000000     359.000000
max         3.000000         1.000000       5.000000     505.000000

          supertypes      toughness           type          types
count  228637.000000  228637.000000  228637.000000  228637.000000
mean        1.753382      16.177574    1116.682300       4.883291
std         0.664505       5.731757     643.870938       3.792199
min         0.000000       0.000000       0.000000       0.000000
25%         2.000000      13.000000     604.000000       3.000000
50%         2.000000      17.000000    1324.000000       3.000000
75%         2.000000      21.000000    1445.000000       7.000000
max         4.000000      21.000000    2079.000000      13.000000

[8 rows x 25 columns]
```

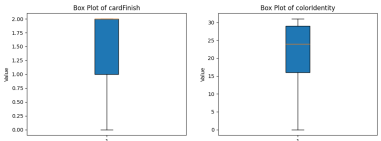
[4]:
```python
# plot function
def plot(data, type, color=plt.cm.viridis(0.2), bins=None, lim=None,␣
 ↪mapping_dict=None):
    """Creates plots for the quantitative DataFrame
    Args:
        - data: dataframe
        - type: type of plot (hist, box)
        - color: plot color
        - bins: number of bins in this hist
        - lim: optional bound for graph
```

```
    """
    # subplot calcuations for size
    num = len(data.columns)
    ncols = 2
    nrows = int(np.ceil(num / ncols))

    # size based on nrows
    plt.figure(figsize=(12, 5*nrows))

    for i, column in enumerate(data.columns):
        plt.subplot(nrows,ncols,i+1)
        if type == "hist":
            plt.hist(data[column], color = color, bins=bins)
            plt.title(f"Histogram of {column}")
            plt.ylabel("Frequency")
            plt.xlabel("Value")
            if lim:
                plt.title(f"Histogram of {column} bounded under {lim}")
                plt.xlim(0,lim)

        elif type == "box":
            if bins == 2 and column != "price":
                sns.boxplot(x = column, y = "price", data = data)
                plt.ylabel("price")
            else:
                plt.boxplot(data[column], whis=lim, patch_artist=True)
                plt.ylabel("Value")
            plt.title(f"Box Plot of {column}")
            if lim:
                plt.ylim(0,lim)
    plt.show()
```

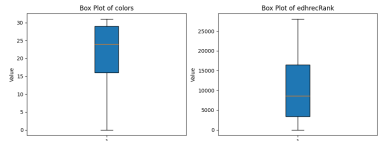
[5]:
```
plot(data, "box")
```


Box Plot of price
Value
1

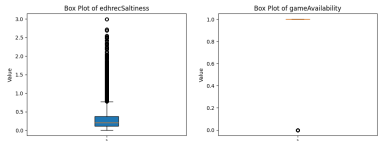
Box Plot of artist
Value
1


Box Plot of cardFinish
Value
1

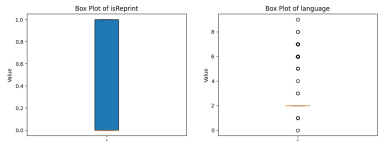
Box Plot of colorIdentity
Value
1


Box Plot of colors
Value
1

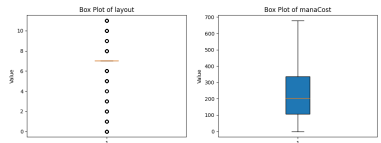
Box Plot of edhrecRank
Value
1


Box Plot of edhrecSaltiness
Value
1

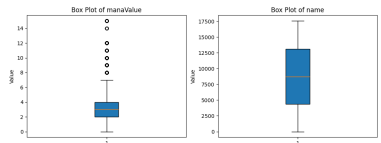
Box Plot of gameAvailability
Value
1


Box Plot of isReprint
Value
1

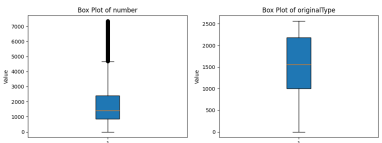
Box Plot of language
Value
1


Box Plot of layout
Value
1

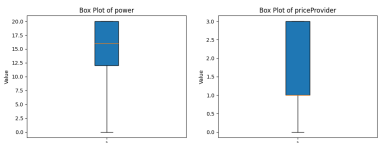
Box Plot of manaCost
Value
1


Box Plot of manaValue
Value
1

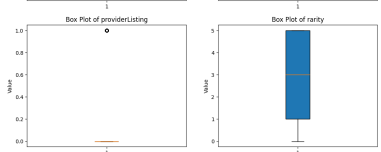
Box Plot of name
Value
1


Box Plot of number
Value
1

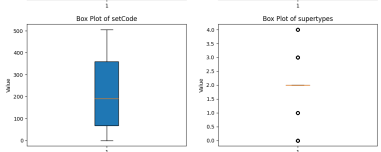
Box Plot of originalType
Value
1


Box Plot of power
Value
1

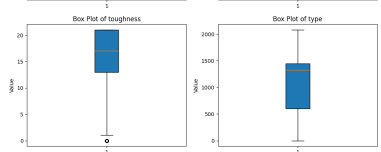
Box Plot of priceProvider
Value
1


Box Plot of providerListing
Value
1

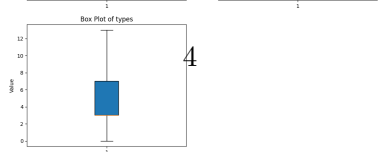
Box Plot of rarity
Value
1


Box Plot of setCode
Value
1

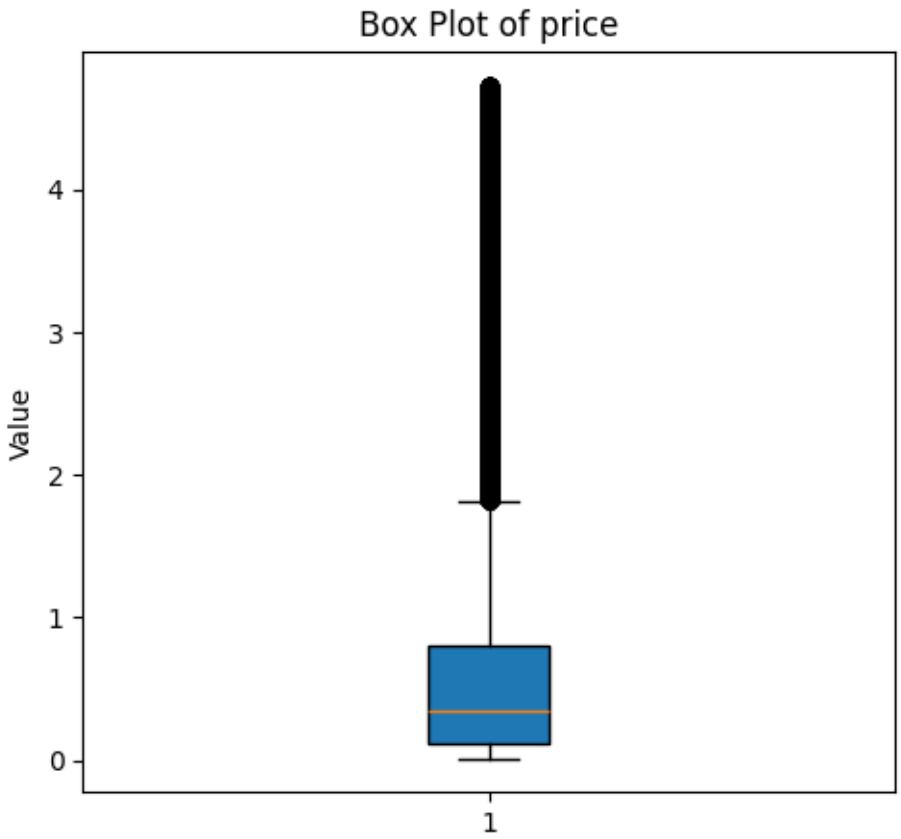
Box Plot of supertypes
Value
1


Box Plot of toughness
Value
1

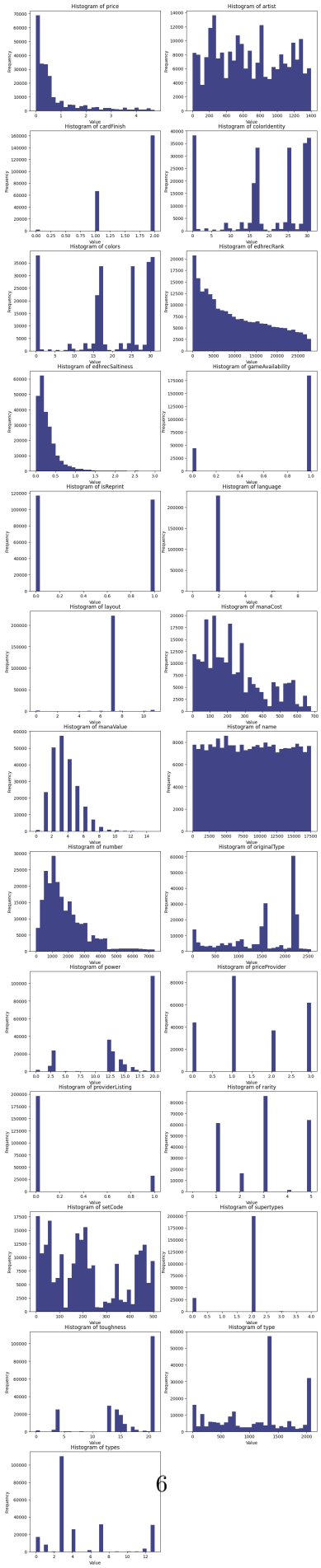
Box Plot of type
Value
1


Box Plot of types
Value
1

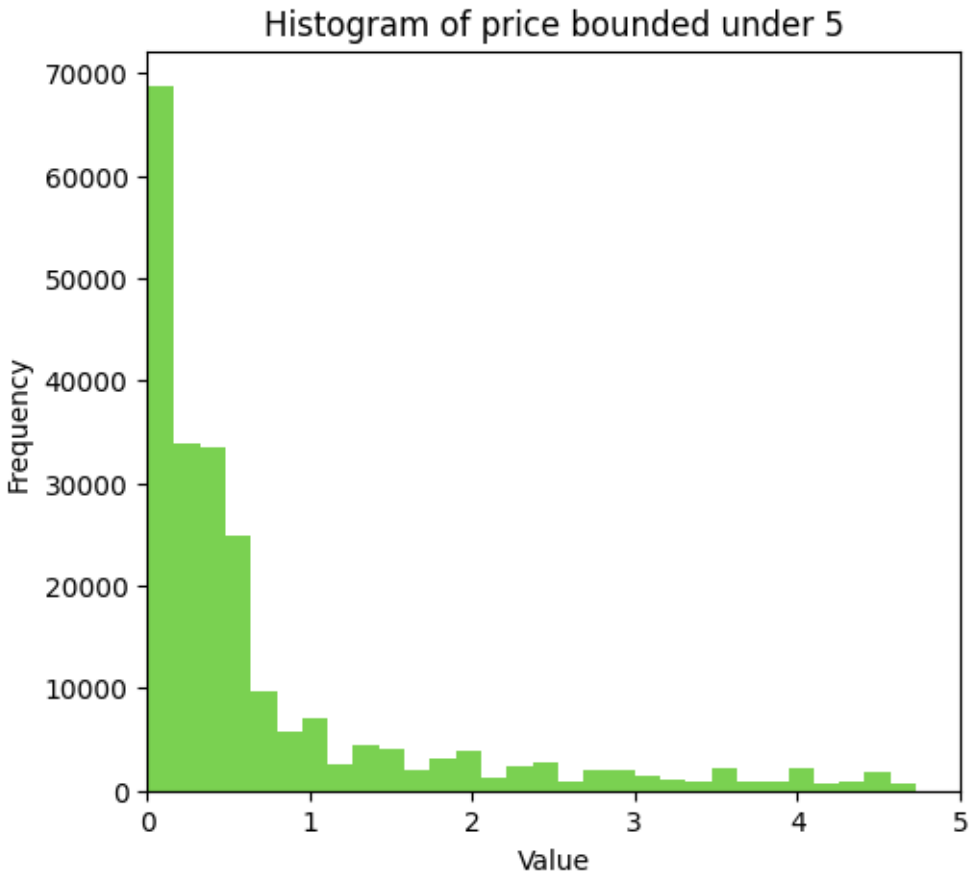
[6]:
```
plot(data[["price"]], "box")
```



[7]:
```
plot(data, "hist", bins=30)
```


Histogram of price
Histogram of artist
Histogram of cardFinish
Histogram of coloridentity
Histogram of colors
Histogram of edhrecRank
Histogram of edhrecSaltiness
Histogram of gameAvailability
Histogram of isReprint
Histogram of language
Histogram of layout
Histogram of manaCost
Histogram of manaValue
Histogram of name
Histogram of number
Histogram of originalType
Histogram of power
Histogram of priceProvider
Histogram of providerListing
Histogram of rarity
Histogram of setCode
Histogram of supertypes
Histogram of toughness
Histogram of type
Histogram of types
Frequency
Value

[8]:
```
plot(data[["price"]], "hist", plt.cm.viridis(0.8), bins=30, lim=5)
print(data['price'].describe())
```

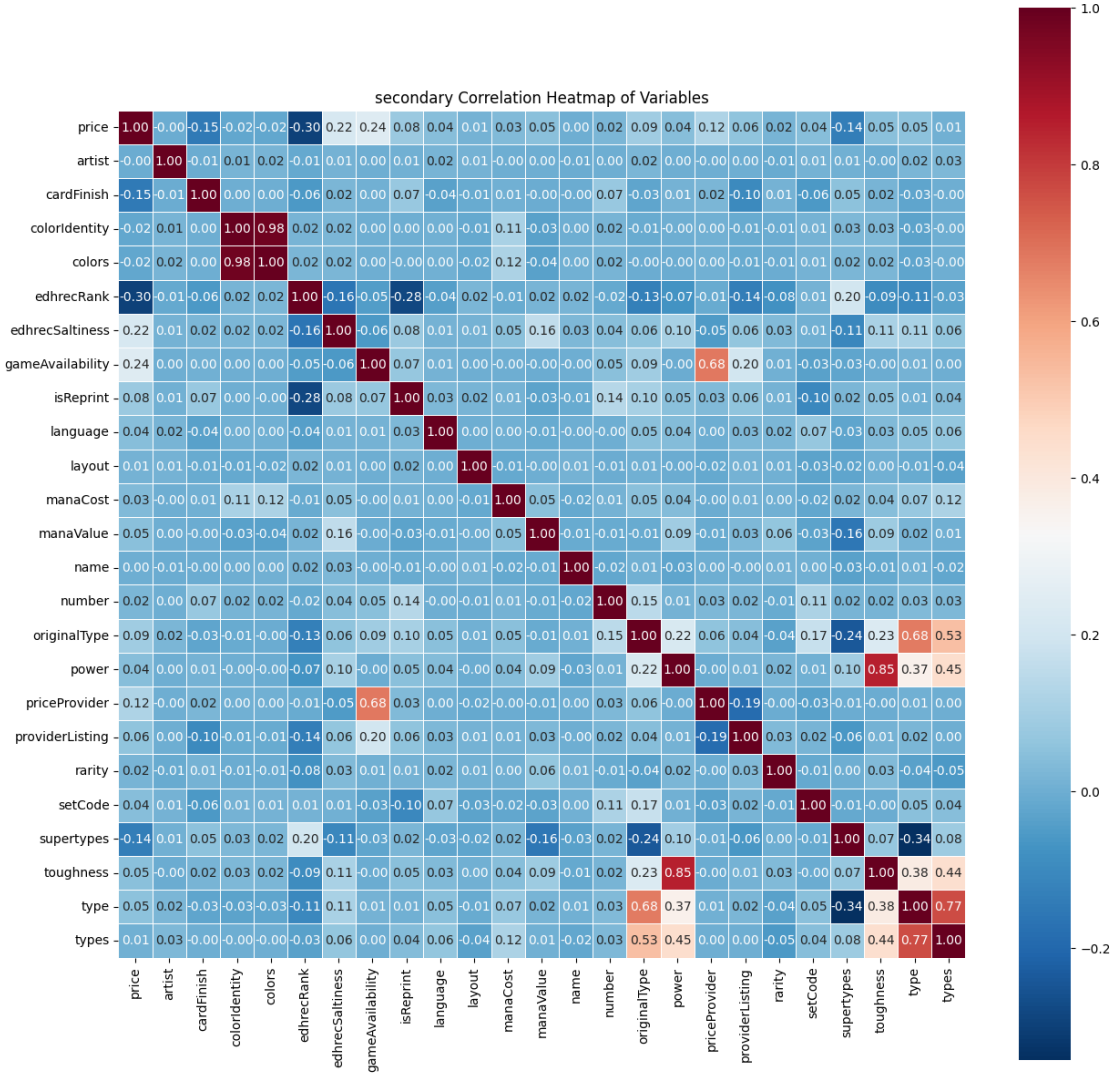



```
count    228637.000000
mean          0.739998
std           1.007018
min           0.010000
25%           0.120000
50%           0.350000
75%           0.800000
max           4.730000
Name: price, dtype: float64
```

[9]:
```
# Correlations
def corr(data, dataset):
    """Plot correlation between variables in a dataframe
    Args:
```

```
        - data: dataframe
    """
    correlation_matrix = data.corr()

    # get price to be the first variable
    ordered = ['price'] + [col for col in correlation_matrix.columns if col !=␣
↪'price']
    ordered_correlation_matrix = correlation_matrix.loc[ordered, ordered]

    plt.figure(figsize=(15,15))
    sns.heatmap(ordered_correlation_matrix, annot=True, fmt='.2f',␣
↪cmap='RdBu_r', square=True, linewidths=0.5)
    plt.title(f"{dataset} Correlation Heatmap of Variables")
    plt.savefig(f'../dataset/graphs/Correlation_{dataset}.png', format='png')
    plt.show()
```

[10]:
```
corr(data, dataset)
```



[11]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

df = data.copy()
# Define a function to calculate VIF
def calculate_vif(df):
    # Adding a constant for intercept
    X = add_constant(df)
    vif_data = pd.DataFrame()
    vif_data['feature'] = X.columns
    vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.
 ↪shape[1])]
    return vif_data
```

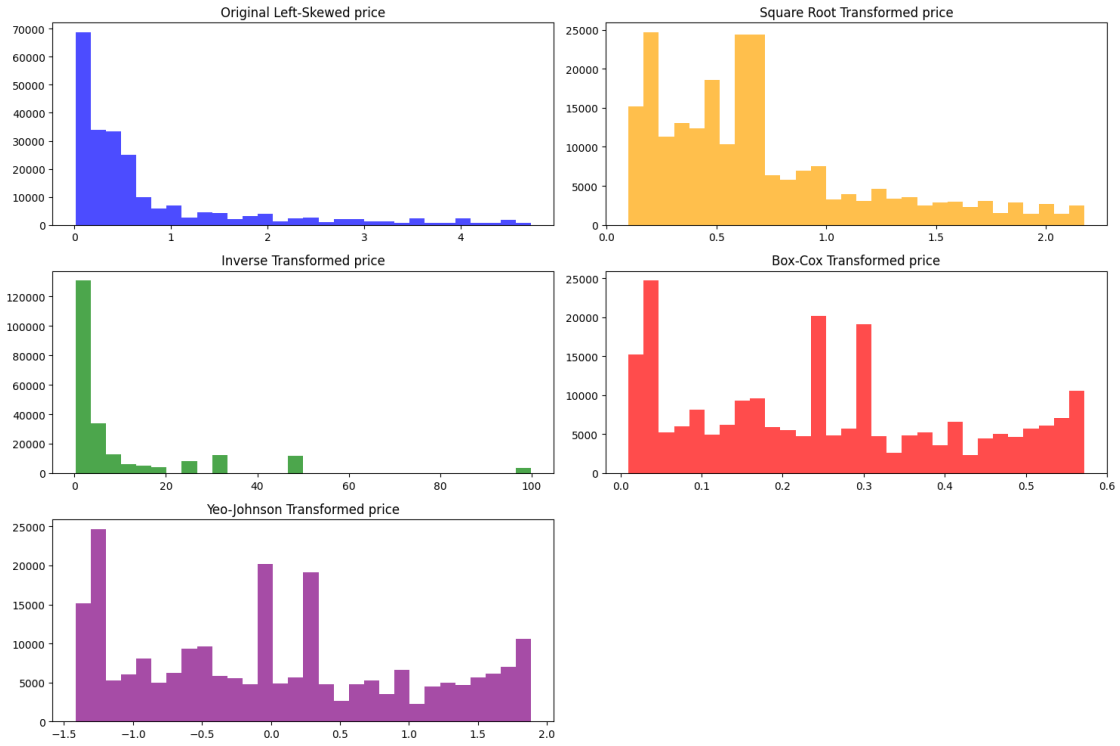
```
# Calculate VIF
vif_results = calculate_vif(df.drop(columns='price'))  # Drop the dependent␣
 ↪variable
print("Variance Inflation Factor (VIF) results:")

# Identify features with high VIF
high_vif = vif_results[vif_results['VIF'] > 10]  # or 5 based on context
print("\nFeatures with high VIF (> 10):")
print(high_vif)
```

```
Variance Inflation Factor (VIF) results:

Features with high VIF (> 10):
         feature         VIF
0          const  243.812157
3  colorIdentity   26.922122
4         colors   26.931672
```

[12]:
```
## transform data for normalcy
df = data.copy()


import scipy.stats as stats
from sklearn.preprocessing import PowerTransformer


def transforms (df, col):
    df['square_root_transformed'] = np.sqrt(df[col])
    df['inverse_transformed'] = 1 / (df[col] + 1e-5)  # Adding a small value to␣
 ↪avoid division by zero
    df['boxcox_transformed'], _ = stats.boxcox(df[col] + 1)  # Adding 1 to␣
 ↪handle zeros
    pt = PowerTransformer(method='yeo-johnson')
    df['yeo_transformed'] = pt.fit_transform(df[[col]])

    skewness = {
        'Original': df[col].skew(),
        'Square Root': df['square_root_transformed'].skew(),
        'Inverse': df['inverse_transformed'].skew(),
        'Box-Cox': df['boxcox_transformed'].skew(),
        'Yeo-Johnson': df['yeo_transformed'].skew(),
    }

    for transform, skew in skewness.items():
        print(f'{transform} skewness: {skew}')

    # Visualize the transformed data
```

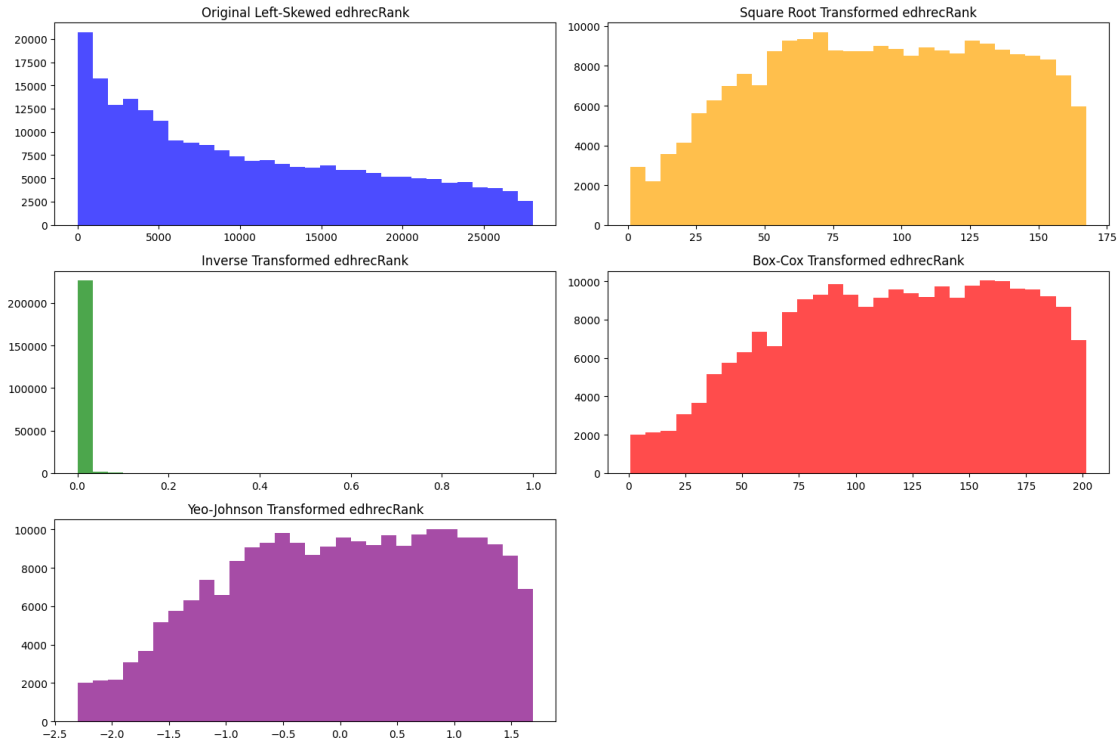
```
    plt.figure(figsize=(15, 10))

    plt.subplot(3, 2, 1)
    plt.hist(df[col], bins=30, alpha=0.7, color='blue')
    plt.title(f'Original Left-Skewed {col}')

    plt.subplot(3, 2, 2)
    plt.hist(df['square_root_transformed'], bins=30, alpha=0.7, color='orange')
    plt.title(f'Square Root Transformed {col}')

    plt.subplot(3, 2, 3)
    plt.hist(df['inverse_transformed'], bins=30, alpha=0.7, color='green')
    plt.title(f'Inverse Transformed {col}')

    plt.subplot(3, 2, 4)
    plt.hist(df['boxcox_transformed'], bins=30, alpha=0.7, color='red')
    plt.title(f'Box-Cox Transformed {col}')

    plt.subplot(3, 2, 5)
    plt.hist(df['yeo_transformed'], bins=30, alpha=0.7, color='purple')
    plt.title(f'Yeo-Johnson Transformed {col}')

    plt.tight_layout()
    plt.show()

    return df

price = transforms(df,'price')
```

```
Original skewness: 2.0777525736345335
Square Root skewness: 1.1214983436053272
Inverse skewness: 3.0061944433522036
Box-Cox skewness: 0.31754573903010114
Yeo-Johnson skewness: 0.3175457509434511
```

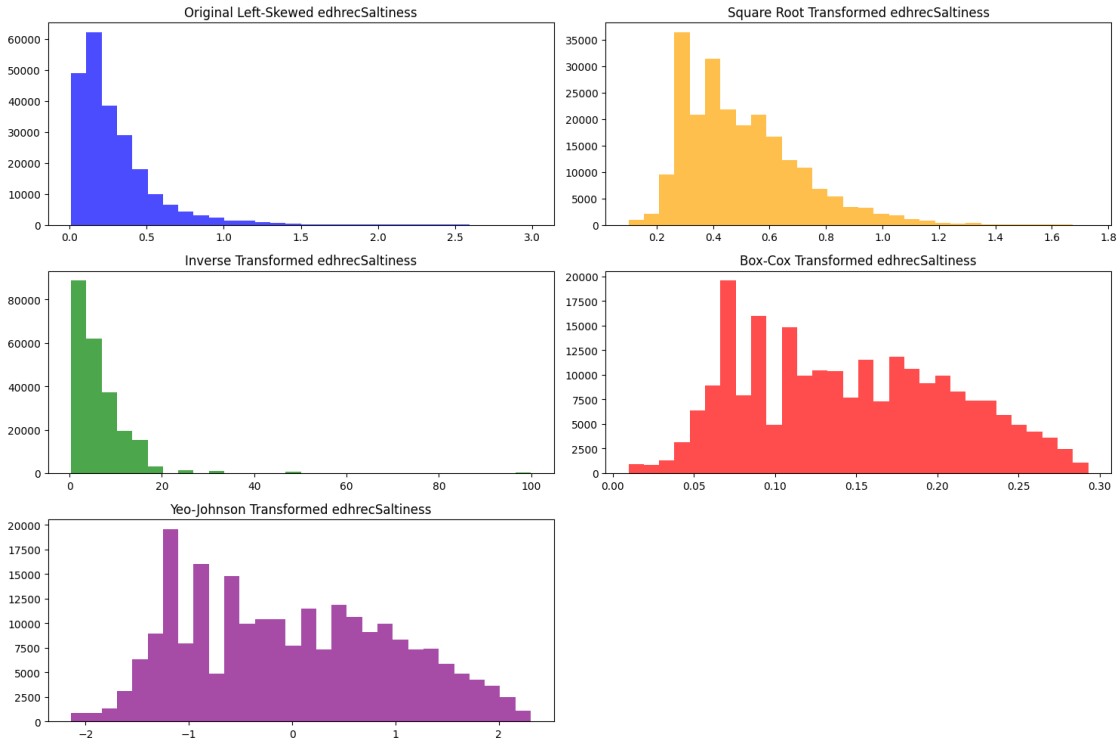


[13]:
```
transform = data.copy()
transform['price'] = price['boxcox_transformed']

edr = transforms(df,'edhrecRank')
transform['price'] = edr['boxcox_transformed']

eds = transforms(df,'edhrecSaltiness')
transform['price'] = eds['boxcox_transformed']
```

```
Original skewness: 0.516758773844703
Square Root skewness: -0.10542944288730692
Inverse skewness: 27.761915554956353
Box-Cox skewness: -0.21367313858641496
Yeo-Johnson skewness: -0.21367313802988314
```




```
Original skewness: 2.551911988642195
Square Root skewness: 1.1043078139230864
Inverse skewness: 5.92291343922691
Box-Cox skewness: 0.21533214511501866
Yeo-Johnson skewness: 0.21533218317695288
```



[14]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
from statsmodels.tools.tools import add_constant

df = data.copy()
# Define a function to calculate VIF
def calculate_vif(df):
    # Adding a constant for intercept
    X = add_constant(df)
    vif_data = pd.DataFrame()
    vif_data['feature'] = X.columns
    vif_data['VIF'] = [variance_inflation_factor(X.values, i) for i in range(X.
 ↪shape[1])]
    return vif_data

# Calculate VIF
vif_results = calculate_vif(df.drop(columns='price'))  # Drop the dependent␣
 ↪variable
print("Variance Inflation Factor (VIF) results:")

# Identify features with high VIF
high_vif = vif_results[vif_results['VIF'] > 10]  # or 5 based on context
print("\nFeatures with high VIF (> 10):")
print(high_vif)
```

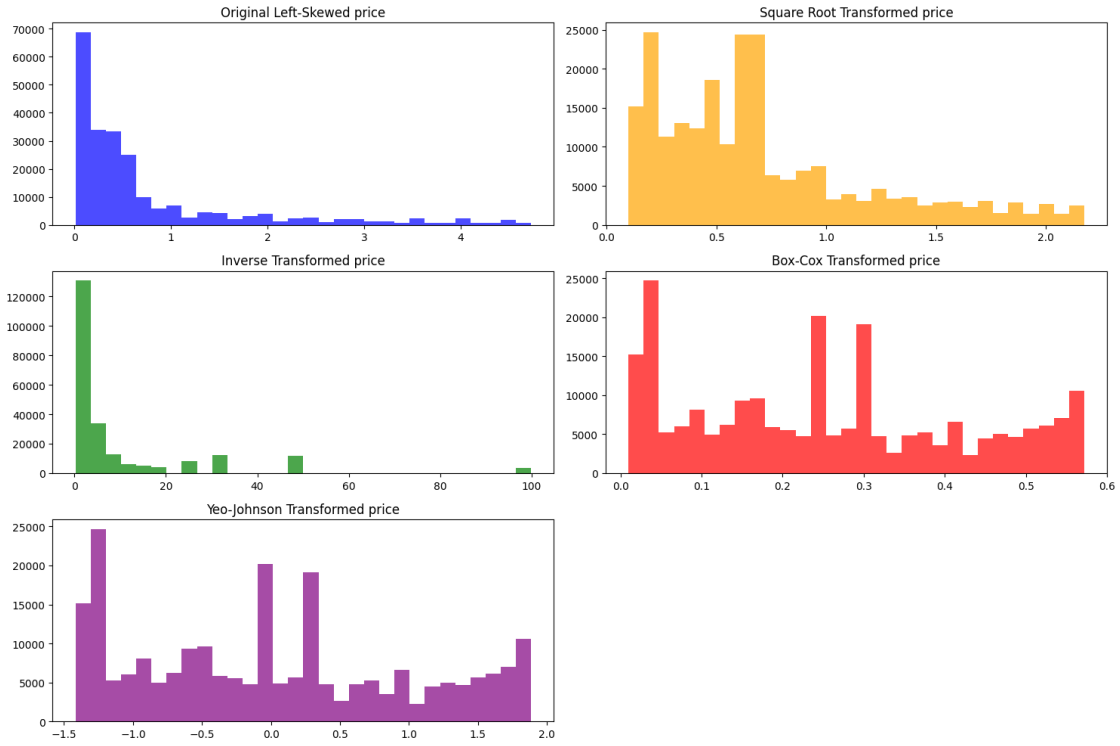
```
Variance Inflation Factor (VIF) results:

Features with high VIF (> 10):
         feature         VIF
0          const  243.812157
3  colorIdentity   26.922122
4         colors   26.931672
```

[15]:
```
# drop primary colorIdentity
data = data.drop(columns='colorIdentity')
```

[16]:
```
## transform data for normalcy
df = data.copy()


import scipy.stats as stats
from sklearn.preprocessing import PowerTransformer


def transforms (df, col):
    df['square_root_transformed'] = np.sqrt(df[col])
    df['inverse_transformed'] = 1 / (df[col] + 1e-5)  # Adding a small value to␣
 ↪avoid division by zero
    df['boxcox_transformed'], _ = stats.boxcox(df[col] + 1)  # Adding 1 to␣
 ↪handle zeros
    pt = PowerTransformer(method='yeo-johnson')
    df['yeo_transformed'] = pt.fit_transform(df[[col]])

    skewness = {
        'Original': df[col].skew(),
        'Square Root': df['square_root_transformed'].skew(),
        'Inverse': df['inverse_transformed'].skew(),
        'Box-Cox': df['boxcox_transformed'].skew(),
        'Yeo-Johnson': df['yeo_transformed'].skew(),
    }

    for transform, skew in skewness.items():
        print(f'{transform} skewness: {skew}')

    # Visualize the transformed data
    plt.figure(figsize=(15, 10))

    plt.subplot(3, 2, 1)
    plt.hist(df[col], bins=30, alpha=0.7, color='blue')
    plt.title(f'Original Left-Skewed {col}')

    plt.subplot(3, 2, 2)
```

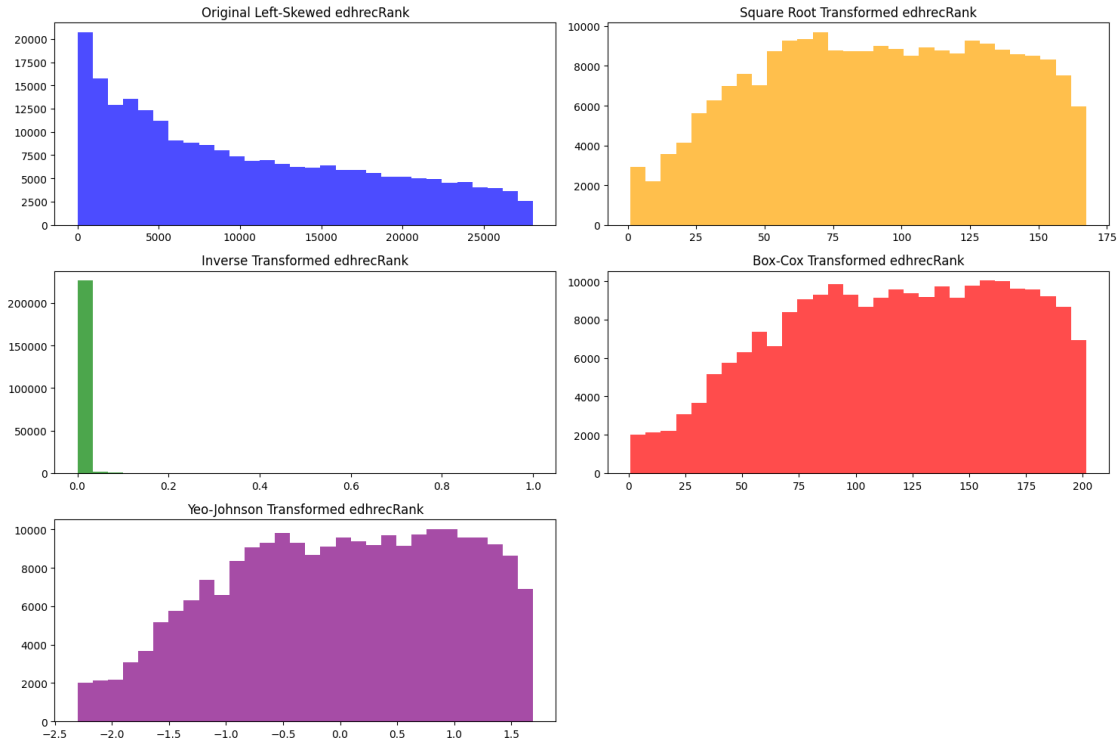
```
    plt.hist(df['square_root_transformed'], bins=30, alpha=0.7, color='orange')
    plt.title(f'Square Root Transformed {col}')

    plt.subplot(3, 2, 3)
    plt.hist(df['inverse_transformed'], bins=30, alpha=0.7, color='green')
    plt.title(f'Inverse Transformed {col}')

    plt.subplot(3, 2, 4)
    plt.hist(df['boxcox_transformed'], bins=30, alpha=0.7, color='red')
    plt.title(f'Box-Cox Transformed {col}')

    plt.subplot(3, 2, 5)
    plt.hist(df['yeo_transformed'], bins=30, alpha=0.7, color='purple')
    plt.title(f'Yeo-Johnson Transformed {col}')

    plt.tight_layout()
    plt.show()

    return df

price = transforms(df,'price')
```

```
Original skewness: 2.0777525736345335
Square Root skewness: 1.1214983436053272
Inverse skewness: 3.0061944433522036
Box-Cox skewness: 0.31754573903010114
Yeo-Johnson skewness: 0.3175457509434511
```

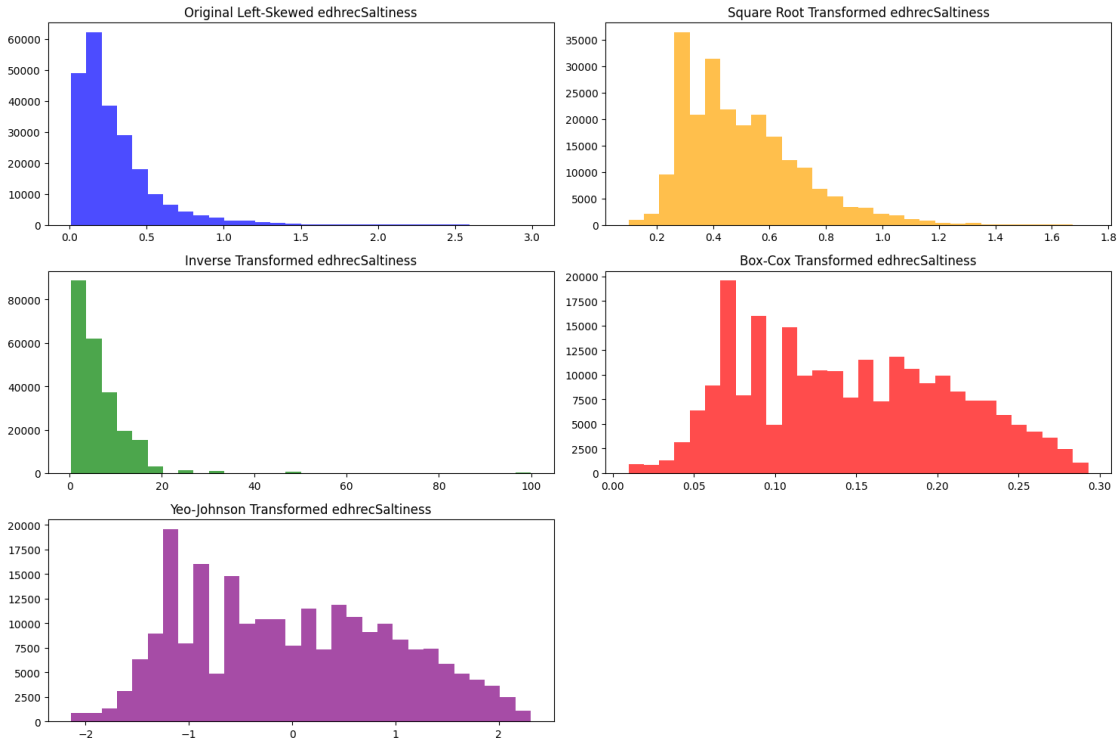


[17]:
```
transform = data.copy()
transform['price'] = price['boxcox_transformed']

edr = transforms(df,'edhrecRank')
transform['price'] = edr['boxcox_transformed']

eds = transforms(df,'edhrecSaltiness')
transform['price'] = eds['boxcox_transformed']
```

```
Original skewness: 0.516758773844703
Square Root skewness: -0.10542944288730692
Inverse skewness: 27.761915554956353
Box-Cox skewness: -0.21367313858641496
Yeo-Johnson skewness: -0.21367313802988314
```




```
Original skewness: 2.551911988642195
Square Root skewness: 1.1043078139230864
Inverse skewness: 5.92291343922691
Box-Cox skewness: 0.21533214511501866
Yeo-Johnson skewness: 0.21533218317695288
```



[18]:
```
# Initialize a results list
results = []

# Calculate t-scores and p-values for each independent variable
for col in df.columns:
    if col != 'price':
        t_stat, p_value = stats.ttest_ind(df[col], df['price'],␣
 ↪equal_var=False)  # Welch's t-test
        results.append({'Variable': col, 'T-score': t_stat, 'P-value': p_value})

# Create a results DataFrame
results_df = pd.DataFrame(results)

print(f"Secondary Dataset Results: \n {results_df.to_string(index=False)}")
```

```
Secondary Dataset Results:
          Variable     T-score       P-value
            artist  800.471980  0.000000e+00
        cardFinish  409.419464  0.000000e+00
            colors  848.585308  0.000000e+00
       edhrecRank  622.054171  0.000000e+00
    edhrecSaltiness -205.092953  0.000000e+00
   gameAvailability   29.658054 5.119910e-193
          isReprint -106.482666  0.000000e+00
```

```
                language  590.618660  0.000000e+00
                  layout 2345.247354  0.000000e+00
                manaCost  668.759183  0.000000e+00
               manaValue  650.367138  0.000000e+00
                    name  823.969217  0.000000e+00
                  number  656.399831  0.000000e+00
            originalType  985.763186  0.000000e+00
                   power 1131.055230  0.000000e+00
           priceProvider  248.340464  0.000000e+00
         providerListing -268.418148  0.000000e+00
                  rarity  584.873268  0.000000e+00
                 setCode  654.309904  0.000000e+00
              supertypes  401.622635  0.000000e+00
               toughness 1268.420404  0.000000e+00
                    type  828.734928  0.000000e+00
                   types  504.929862  0.000000e+00
square_root_transformed -111.424613  0.000000e+00
    inverse_transformed  421.673324  0.000000e+00
     boxcox_transformed -281.461977  0.000000e+00
        yeo_transformed -249.324211  0.000000e+00
```

[19]:
```
# Initialize a results list
results = []

# Calculate t-scores and p-values for each independent variable
for col in transform.columns:
    if col != 'price':
        t_stat, p_value = stats.ttest_ind(df[col], df['price'],␣
 ↪equal_var=False)  # Welch's t-test
        results.append({'Variable': col, 'T-score': t_stat, 'P-value': p_value})

# Create a results DataFrame
results_df = pd.DataFrame(results)

print(f"Primary Dataset Results: \n{results_df.to_string(index=False)}")
```

```
Primary Dataset Results:
        Variable     T-score       P-value
          artist  800.471980  0.000000e+00
      cardFinish  409.419464  0.000000e+00
          colors  848.585308  0.000000e+00
      edhrecRank  622.054171  0.000000e+00
 edhrecSaltiness -205.092953  0.000000e+00
gameAvailability   29.658054 5.119910e-193
       isReprint -106.482666  0.000000e+00
        language  590.618660  0.000000e+00
          layout 2345.247354  0.000000e+00
        manaCost  668.759183  0.000000e+00
```

```
          manaValue  650.367138  0.000000e+00
               name  823.969217  0.000000e+00
             number  656.399831  0.000000e+00
       originalType  985.763186  0.000000e+00
              power 1131.055230  0.000000e+00
      priceProvider  248.340464  0.000000e+00
    providerListing -268.418148  0.000000e+00
             rarity  584.873268  0.000000e+00
            setCode  654.309904  0.000000e+00
         supertypes  401.622635  0.000000e+00
          toughness 1268.420404  0.000000e+00
               type  828.734928  0.000000e+00
              types  504.929862  0.000000e+00
```

[20]:
```
print(transform)

transform.to_csv('../dataset/transform_secondary.csv', index = False)
```

```
           price  artist  cardFinish  colors  edhrecRank  edhrecSaltiness  \
0       0.209563     813           2      29     10512.0             0.44
1       0.220675      56           2      25      6705.0             0.50
2       0.156677     574           2      17     25046.0             0.25
3       0.182993     337           2       0     23391.0             0.33
4       0.152848    1106           2      16      1337.0             0.24
…            …     …           …      …          …                …
228632  0.094143     487           1      17       128.0             0.12
228633  0.283761    1181           2      29       347.0             1.58
228634  0.215367     596           2      17       602.0             0.47
228635  0.215367     596           2      17       602.0             0.47
228636  0.215367     596           2      17       602.0             0.47

        gameAvailability  isReprint  language  layout  …  originalType  \
0                      0          1         2       7  …          2180
1                      0          1         2       7  …          2180
2                      0          1         2       7  …          2180
3                      0          1         2       7  …          2180
4                      0          1         2       7  …          2180
…                    …          …         …       …  …             …
228632                 1          1         4       7  …          2180
228633                 1          1         2       7  …          2180
228634                 1          1         5       7  …          2180
228635                 1          1         5       7  …          2180
228636                 1          1         5       7  …          2180

        power  priceProvider  providerListing  rarity  setCode  supertypes  \
0          20              0                0       3      388           2
1          20              0                0       3      388           2
2          13              0                0       3      388           2
```

```
3            3                0                  0        3       388             2
4           20                0                  0        5       388             2
…        …                 …                  …        …        …              …
228632       3                1                  1        3       263             2
228633      17                1                  1        3       412             2
228634      20                1                  1        3       441             2
228635      20                1                  0        3       441             2
228636      20                2                  0        3       441             2

        toughness  type  types
0              21  2075     13
1              21  1385      7
2              14   372      3
3               4  1258      3
4              21     0      0
…           …    …    …
228632          4   438      3
228633         19   781      3
228634         21  1324      4
228635         21  1324      4
228636         21  1324      4

[228637 rows x 24 columns]
```