OLS_RF_primary

October 22, 2024

1 Primary Dataset - Linear Regression and Price Analysis using Random Forest

[61]:
```
# 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
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import MinMaxScaler

sns.set()
```

[62]:
```
# set this to either primary or secondary depending on the analysis you want to␣
↪do
dataset = 'primary'

data = pd.read_csv(f'../dataset/mapped_{dataset}.csv', sep=",")
transform = pd.read_csv(f'../dataset/transform_{dataset}.csv', sep=",")
data = data.drop(columns=['uuid'])
```

[63]:
```
import statsmodels.api as sm

X = data.loc[:, ~data.columns.isin(['price'])]
y = data["price"]

model = sm.OLS(y, X)     # Describe model

result = model.fit()       # Fit model

print(result.summary())
```

```
                            OLS Regression Results
==============================================================================
=======
```

```
Dep. Variable:                  price   R-squared (uncentered):
0.498
Model:                            OLS   Adj. R-squared (uncentered):
0.497
Method:                 Least Squares   F-statistic:
654.9
Date:                Tue, 22 Oct 2024   Prob (F-statistic):
0.00
Time:                        03:17:47   Log-Likelihood:
-27745.
No. Observations:               15214   AIC:
5.554e+04
Df Residuals:                   15191   BIC:
5.571e+04
Df Model:                          23
Covariance Type:            nonrobust
================================================================================
====
                       coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----
artist              -0.0001   9.52e-05     -1.390      0.164      -0.000
5.43e-05
cardFinish          -0.3984      0.022    -17.710      0.000      -0.443
-0.354
colorIdentity        0.0150      0.006      2.652      0.008       0.004
0.026
colors              -0.0137      0.006     -2.406      0.016      -0.025
-0.003
edhrecRank       -4.789e-05   2.71e-06    -17.658      0.000   -5.32e-05
-4.26e-05
edhrecSaltiness      0.8681      0.043     20.391      0.000       0.785
0.952
gameAvailability     1.3893      0.049     28.612      0.000       1.294
1.485
isReprint           -0.1448      0.027     -5.333      0.000      -0.198
-0.092
layout               0.2547      0.029      8.863      0.000       0.198
0.311
manaCost             0.0003      0.000      1.657      0.098   -4.67e-05
0.001
manaValue            0.0621      0.010      6.258      0.000       0.043
0.082
name             -1.338e-05   3.18e-05     -0.420      0.674   -7.58e-05
4.9e-05
number            8.447e-05   6.24e-05      1.354      0.176   -3.78e-05
0.000
```

```
originalType        -0.0010      0.000     -6.922      0.000      -0.001
-0.001
power               -0.0038      0.007     -0.553      0.580      -0.017
0.010
priceProvider       -0.1241      0.017     -7.138      0.000      -0.158
-0.090
providerListing     -0.4133      0.038    -10.768      0.000      -0.489
-0.338
rarity              -0.4203      0.013    -31.138      0.000      -0.447
-0.394
setCode              0.0012      0.000      4.969      0.000       0.001
0.002
supertypes           0.2266      0.055      4.153      0.000       0.120
0.334
toughness           -0.0193      0.009     -2.244      0.025      -0.036
-0.002
type                 0.0006      0.000      3.581      0.000       0.000
0.001
types                0.4023      0.056      7.231      0.000       0.293
0.511
==============================================================================
Omnibus:                     4828.653   Durbin-Watson:                   1.191
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            13067.893
Skew:                           1.723   Prob(JB):                         0.00
Kurtosis:                       5.957   Cond. No.                     3.99e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[3] The condition number is large, 3.99e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[64]:

```python
Xt = transform.loc[:, ~transform.columns.isin(['price'])]
yt = transform["price"]

modelt = sm.OLS(yt, Xt)    # Describe model

resultt = modelt.fit()       # Fit model

print(resultt.summary())
```

```
                                 OLS Regression Results
================================================================================
=======
Dep. Variable:                  price   R-squared (uncentered):
```

```
0.975
Model:                            OLS   Adj. R-squared (uncentered):
0.975
Method:                 Least Squares   F-statistic:
2.700e+04
Date:                Tue, 22 Oct 2024   Prob (F-statistic):
0.00
Time:                        03:17:47   Log-Likelihood:
29964.
No. Observations:               15214   AIC:
-5.988e+04
Df Residuals:                   15192   BIC:
-5.972e+04
Df Model:                          22
Covariance Type:            nonrobust
================================================================================
====
                       coef    std err          t      P>|t|      [0.025
0.975]
--------------------------------------------------------------------------------
----
artist           -2.501e-07   2.15e-06     -0.117      0.907   -4.46e-06
3.96e-06
cardFinish           0.0012      0.001      2.277      0.023       0.000
0.002
colors              -0.0002   2.91e-05     -6.594      0.000      -0.000
-0.000
edhrecRank       -1.212e-06    6.1e-08    -19.867      0.000   -1.33e-06
-1.09e-06
edhrecSaltiness      0.2169      0.001    226.413      0.000       0.215
0.219
gameAvailability     0.0019      0.001      1.764      0.078      -0.000
0.004
isReprint            0.0062      0.001     10.103      0.000       0.005
0.007
layout               0.0204      0.001     31.549      0.000       0.019
0.022
manaCost         -8.661e-06   3.47e-06     -2.497      0.013   -1.55e-05
-1.86e-06
manaValue            0.0015      0.000      6.520      0.000       0.001
0.002
name              5.836e-06   7.17e-07      8.141      0.000    4.43e-06
7.24e-06
number            3.712e-06   1.41e-06      2.641      0.008    9.57e-07
6.47e-06
originalType     -2.431e-05    3.1e-06     -7.833      0.000   -3.04e-05
-1.82e-05
power                0.0016      0.000     10.388      0.000       0.001
```

```
0.002
priceProvider        0.0007      0.000      1.710      0.087   -9.81e-05
0.001
providerListing      0.0019      0.001      2.231      0.026       0.000
0.004
rarity              -0.0050      0.000    -16.368      0.000      -0.006
-0.004
setCode           2.626e-05   5.44e-06      4.826      0.000    1.56e-05
3.69e-05
supertypes           0.0055      0.001      4.467      0.000       0.003
0.008
toughness           -0.0010      0.000     -5.226      0.000      -0.001
-0.001
type              2.442e-06   3.64e-06      0.671      0.502   -4.69e-06
9.57e-06
types                0.0174      0.001     13.924      0.000       0.015
0.020
==============================================================================
Omnibus:                     6070.317   Durbin-Watson:                   0.420
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            46003.618
Skew:                          -1.731   Prob(JB):                         0.00
Kurtosis:                      10.784   Cond. No.                     3.99e+04
==============================================================================

Notes:
[1] R² is computed without centering (uncentered) since the model does not
contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly
specified.
[3] The condition number is large, 3.99e+04. This might indicate that there are
strong multicollinearity or other numerical problems.
```

[65]:
```python
from sklearn.metrics import mean_squared_error

y_pred = result.predict(X)
mse = mean_squared_error(y, y_pred)

fstat, pvalue = sm.stats.linear_rainbow(result)
print(f' f stat: {fstat} | p value: {pvalue} | mse: {mse}')
```

```
 f stat: 1.4061864622192486 | p value: 5.707235837274266e-50 | mse:
2.2464624841702174
```

[66]:
```python
y_predt = resultt.predict(Xt)
mset = mean_squared_error(yt, y_predt)

fstatt, pvaluet = sm.stats.linear_rainbow(resultt)
print(f' f stat: {fstatt} | p value: {pvaluet} | mse: {mset}')
```

f stat: 1.309542223236456 | p value: 4.318233436136021e-32 | mse:
0.0011398375904127526

---

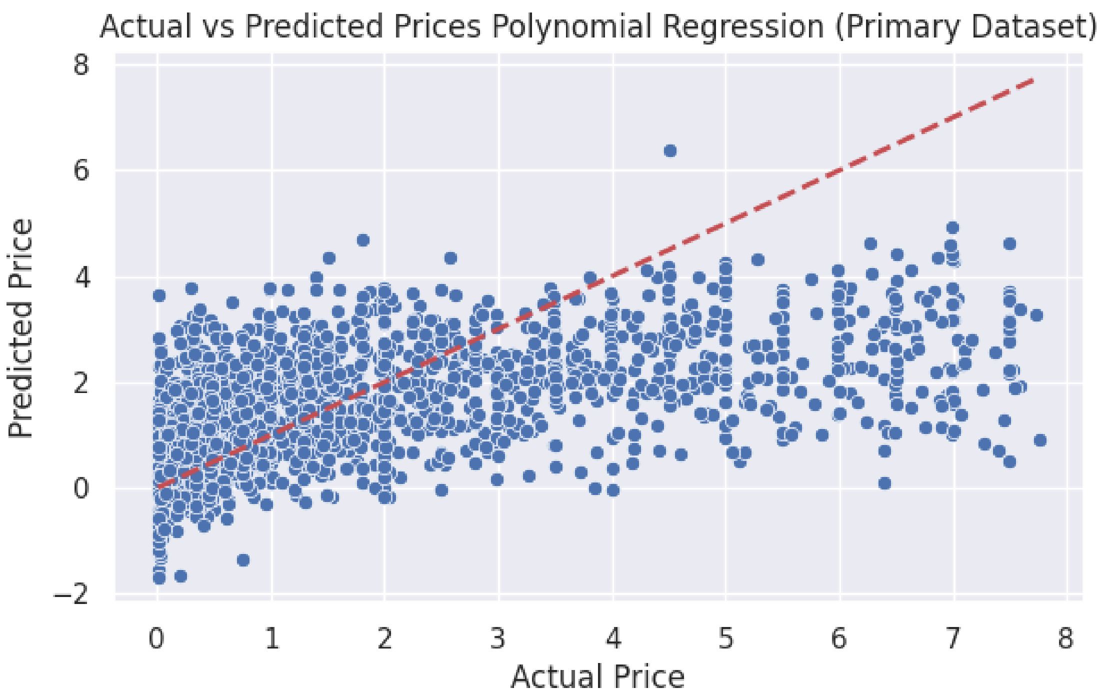
2 Polynomial Regression Prediction

[67]:
```
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score

# Create linear regression model and use it based on selected feature and target
def prediction(feature, target):
    X_train, X_test, y_train, y_test = train_test_split(feature, target,␣
 ↪test_size=0.2, random_state=42)

    poly = PolynomialFeatures(degree=2)
    X_train_poly = poly.fit_transform(X_train)
    X_test_poly = poly.transform(X_test)

    model = LinearRegression().fit(X_train_poly, y_train)

    y_pred = model.predict(X_test_poly)

    # Calculate metrics
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    return y_test, y_pred, mse, r2
```

[75]:
```
import statsmodels.api as sm
# Price Prediction based on Rank

X = data.loc[:, ~data.columns.isin(['price'])]
y = data["price"]

# Polinomial Linear Regression
y_test, y_pred, mse, r2 = prediction(X, y)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Ploting results

plt.figure(figsize=(7,4))
sns.scatterplot(x=y_test, y=y_pred)
```

```
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--',␣
↪lw=2)
plt.title(f'Actual vs Predicted Prices Polynomial Regression (Primary Dataset)')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.show()
```

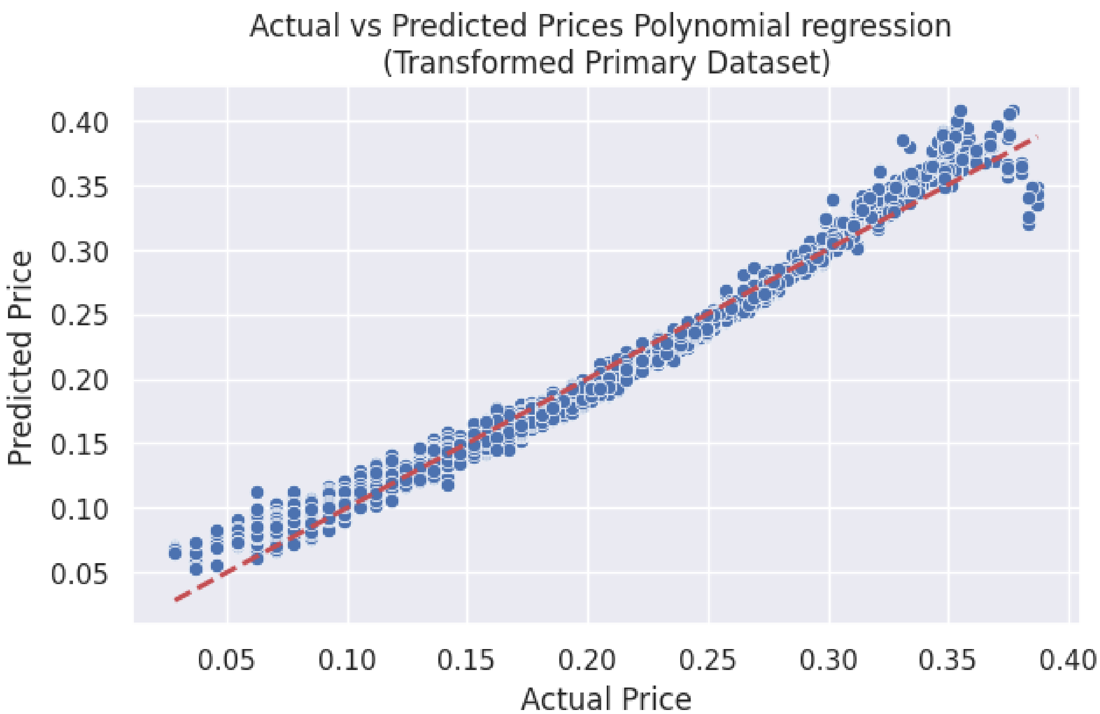
```
Mean Squared Error: 1.91355490272543
R-squared: 0.35104788760622274
```



[76]:
```
Xt = transform.loc[:, ~transform.columns.isin(['price'])]
yt = transform["price"]

# Polinomial Linear Regression
y_testt, y_predt, mset, r2t = prediction(Xt, yt)

print(f'Transformed Mean Squared Error: {mset}')
print(f'Transformed R-squared: {r2t}')

# Ploting results

plt.figure(figsize=(7,4))
sns.scatterplot(x=y_testt, y=y_predt)
```

```
plt.plot([y_testt.min(), y_testt.max()], [y_testt.min(), y_testt.max()], 'r--',␣
↪lw=2)
plt.title(f'Actual vs Predicted Prices Polynomial regression \n(Transformed␣
↪Primary Dataset)')
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.show()
```

```
Transformed Mean Squared Error: 0.00015645371327158383
Transformed R-squared: 0.9758433486702031
```



---

3 Random Forest implementation

[70]:
```
# Random Forest implementation

# Modelling
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score,␣
↪recall_score, ConfusionMatrixDisplay
from sklearn.model_selection import RandomizedSearchCV, train_test_split
from scipy.stats import randint
```

```
from sklearn.preprocessing import StandardScaler
from sklearn import preprocessing
from sklearn import utils

# Tree Visualisation
from sklearn.tree import export_graphviz
from IPython.display import Image
import graphviz
```

[71]:

```
# define features
X = data.loc[:, ~data.columns.isin(['price'])]
#convert y values to categorical values
y = data["price"]


# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)


scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)


rf = RandomForestRegressor(n_estimators=100, random_state=42).
 ↪fit(X_train_scaled, y_train)
```

[72]:

```
# define features
Xt = transform.loc[:, ~transform.columns.isin(['price'])]
#convert y values to categorical values
yt = transform["price"]


# Split the data into training and test sets
X_traint, X_testt, y_traint, y_testt = train_test_split(Xt, yt, test_size=0.2)


scalert = StandardScaler()
X_train_scaledt = scalert.fit_transform(X_traint)
X_test_scaledt = scalert.transform(X_testt)


rft = RandomForestRegressor(n_estimators=100, random_state=42).
 ↪fit(X_train_scaledt, y_traint)
```

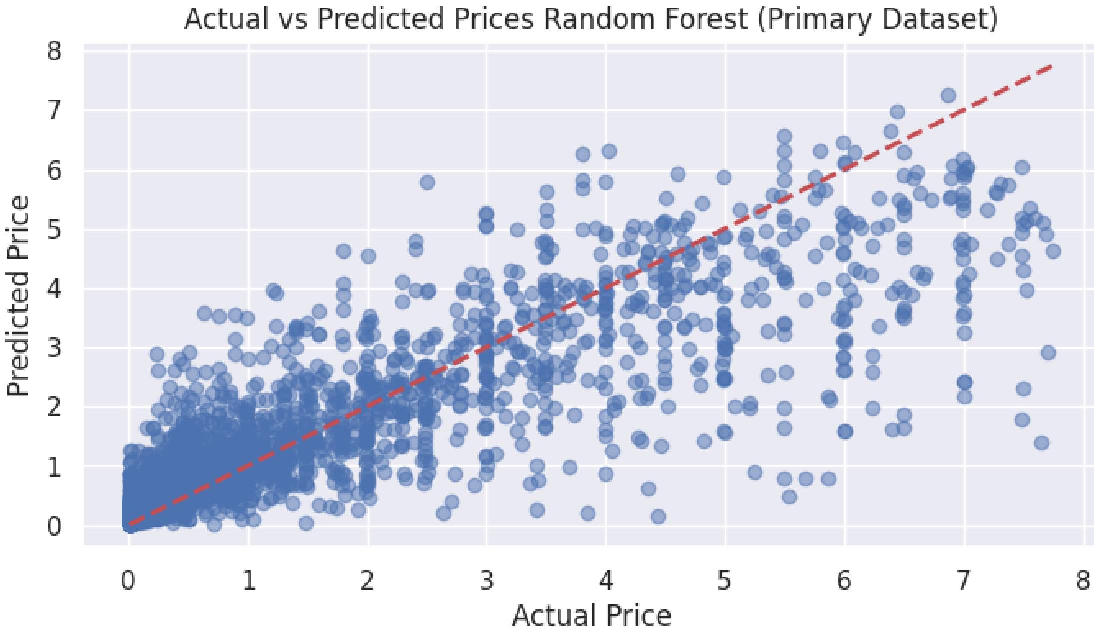
[73]:

```
# Make predictions
y_pred = rf.predict(X_test_scaled)
```

```python
# Calculate metrics
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f'Mean Squared Error: {mse}')
print(f'R-squared: {r2}')

# Feature importance
feature_importance = pd.DataFrame({'feature': X.columns, 'importance': rf.
 ↪feature_importances_})
feature_importance = feature_importance.sort_values('importance',␣
 ↪ascending=False)
print(feature_importance)


# Visualize actual vs predicted prices
plt.figure(figsize=(8, 4))
plt.scatter(y_test, y_pred, alpha=0.5)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--',␣
 ↪lw=2)
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual vs Predicted Prices Random Forest (Primary Dataset)')
plt.show()
```

```
Mean Squared Error: 0.7753397847880381
R-squared: 0.7410351685732852
             feature  importance
4         edhrecRank    0.151957
17            rarity    0.095249
12            number    0.074681
5    edhrecSaltiness    0.073812
1         cardFinish    0.060270
15     priceProvider    0.058717
13      originalType    0.058204
0             artist    0.051208
11              name    0.050712
18           setCode    0.048277
9           manaCost    0.044138
16   providerListing    0.039142
21              type    0.036199
6   gameAvailability    0.033390
10         manaValue    0.024110
20         toughness    0.022553
2      colorIdentity    0.020475
3             colors    0.018325
```

```
14          power    0.017025
7       isReprint    0.013510
8          layout    0.004747
22          types    0.002648
19     supertypes    0.000652
```

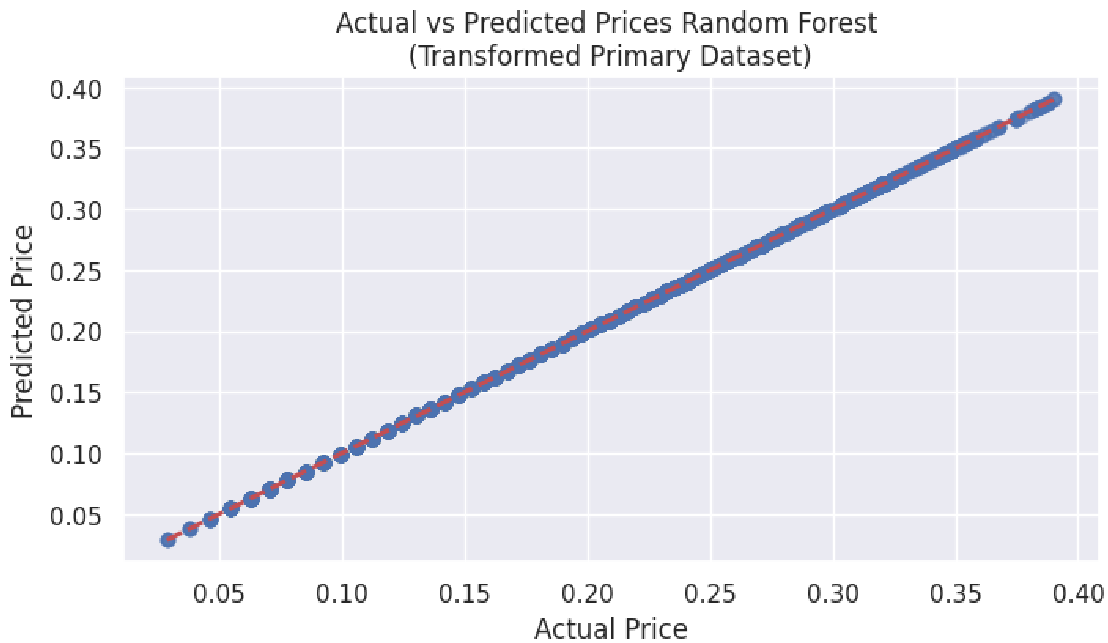


[74]:

```
# Make predictions
y_predt = rft.predict(X_test_scaledt)

# Calculate metrics
mset = mean_squared_error(y_testt, y_predt)
r2t = r2_score(y_testt, y_predt)

print(f'Mean Squared Error: {mset}')
print(f'R-squared: {r2t}')

# Feature importance
feature_importancet = pd.DataFrame({'feature': Xt.columns, 'importance': rft.
 ↪feature_importances_})
feature_importancet = feature_importancet.sort_values('importance',␣
 ↪ascending=False)
print(feature_importancet)


# Visualize actual vs predicted prices
plt.figure(figsize=(8, 4))
```

```
plt.scatter(y_testt, y_predt, alpha=0.5)
plt.plot([y_testt.min(), y_testt.max()], [y_testt.min(), y_testt.max()], 'r--',␣
↪lw=2)
plt.xlabel('Actual Price')
plt.ylabel('Predicted Price')
plt.title('Actual vs Predicted Prices Random Forest \n(Transformed Primary␣
↪Dataset)')
plt.show()
```

```
Mean Squared Error: 8.185580384175704e-10
R-squared: 0.9999998765046534
              feature    importance
4    edhrecSaltiness  9.999861e-01
8           manaCost  2.262619e-06
20              type  2.228246e-06
19         toughness  1.838576e-06
9          manaValue  1.199122e-06
3         edhrecRank  1.137642e-06
13             power  1.082453e-06
2             colors  1.034501e-06
16            rarity  8.476498e-07
10              name  7.212146e-07
12      originalType  4.918205e-07
17           setCode  4.003563e-07
0             artist  3.253511e-07
11            number  1.614810e-07
21             types  1.266552e-07
1         cardFinish  4.615628e-09
5   gameAvailability  4.198553e-09
14     priceProvider  3.143585e-09
6          isReprint  1.757209e-09
15   providerListing  1.022150e-09
7             layout  7.211011e-17
18        supertypes  3.301014e-18
```


Actual vs Predicted Prices Random Forest
(Transformed Primary Dataset)
Predicted Price
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.05
0.10
0.15
0.20
0.25
0.30
0.35
0.40
Actual Price