In [337]:

```
import pandas as pd
from sklearn.preprocessing import OneHotEncoder
from sklearn.preprocessing import LabelEncoder
import category_encoders as ce
from sklearn.impute import KNNImputer
from copy import deepcopy
import numpy as np
from sklearn.metrics import mean_squared_error
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer
import statistics
from matplotlib import pyplot as plt
import seaborn as sns
```

In [340]:

```
df = pd.read_csv("allegro-api-transactions.csv")
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 420020 entries, 0 to 420019
Data columns (total 14 columns):
 #   Column                  Non-Null Count   Dtype  
---  ------                  --------------   -----  
 0   lp                      420020 non-null  int64  
 1   date                    420020 non-null  object 
 2   item_id                 420020 non-null  int64  
 3   categories              420020 non-null  object 
 4   pay_option_on_delivery  420020 non-null  int64  
 5   pay_option_transfer     420020 non-null  int64  
 6   seller                  420020 non-null  object 
 7   price                   420020 non-null  float64
 8   it_is_allegro_standard  420020 non-null  int64  
 9   it_quantity             420020 non-null  int64  
 10  it_is_brand_zone        420020 non-null  int64  
 11  it_seller_rating        420020 non-null  int64  
 12  it_location             420020 non-null  object 
 13  main_category           420020 non-null  object 
dtypes: float64(1), int64(8), object(5)
memory usage: 44.9+ MB
```

Part 1

Target Encoder

In [43]:

```
encoder1 = ce.TargetEncoder(cols=['it_location'], smoothing=0, return_df=True)

df_transformed = encoder1.fit_transform(df, df['price'])
df_transformed["it_location"].head()
```

Out[43]:

```
0     85.423398
1     85.423398
2     61.990914
3     35.433365
4    117.191956
Name: it_location, dtype: float64
```

One hot encoding może zwrócić wielowymiarową zmieną którą pozwala nam w pewnien sposób zakodować zmienne kategoryczne tak by były łatwiejsze do przetwarzania. Użycie Target encoding pozwala nie tylko na

zakodowanie zmiennych ale także zrobienie to w taki sposób by powiązać je z naszym targetem np zamiana lokacji na średnią cenę towarów z tej lokacji

One hot encoder

In [297]:

```
encoder2 = ce.OneHotEncoder(cols=["main_category"])
df_transformed = encoder2.fit_transform(df, df["price"])
df.head(2)
```

Out[297]:

	lp	date	item_id	categories	pay_option_on_delivery	pay_option_transfer	seller	price	it_is_allegro_sta
0	0	2016-04-03 21:21:08	4753602474	['Komputery', 'Dyski i napędy', 'Nośniki', 'No...	1	1	radzioch666	59.99	
1	1	2016-04-03 15:35:26	4773181874	['Odzież, Obuwie, Dodatki', 'Bielizna damska',...	1	1	InwestycjeNET	4.90	

In [45]:

```
le = LabelEncoder()
integer_encoded = le.fit_transform(df.main_category)
print(integer_encoded)
```

```
[12 18  6 ... 18  5 15]
```

In [46]:

```
onehot_encoder = OneHotEncoder(sparse=False)
integer_encoded = integer_encoded.reshape(len(integer_encoded), 1)
onehot_encoded = onehot_encoder.fit_transform(integer_encoded)
print(onehot_encoded)
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]]
```

In [329]:

```
pd.get_dummies(df["main_category"],drop_first=True).head(2)
```

Out[329]:

	Bilety	Biuro i Reklama	Biżuteria i Zegarki	Delikatesy	Dla Dzieci	Dom i Ogród	Filmy	Fotografia	Gry	Instrumenty	...	Nieruchomości	Odzież, Obuwie, Dodatki	Prz
0	0	0	0	0	0	0	0	0	0	0	...	0	0	
1	0	0	0	0	0	0	0	0	0	0	...	0	1	

2 rows × 26 columns

HashingEncoder

In [314]:

```
encoder3 = ce.HashingEncoder(cols=["main_category"])
df_transformed = encoder3.fit_transform(df, df['price'])
```

In [328]:

```
df_transformed.iloc[:,0:9].head(10)
```

Out[328]:

	col_0	col_1	col_2	col_3	col_4	col_5	col_6	col_7	lp
0	0	0	0	0	0	0	1	0	0
1	0	0	1	0	0	0	0	0	1
2	0	0	0	0	0	0	1	0	2
3	0	0	1	0	0	0	0	0	3
4	0	0	1	0	0	0	0	0	4
5	0	0	1	0	0	0	0	0	5
6	0	0	1	0	0	0	0	0	6
7	0	0	1	0	0	0	0	0	7
8	0	0	0	1	0	0	0	0	8
9	0	0	0	1	0	0	0	0	9

- **Na pierwszy rzut oka wygląda podobnie do One Hot Encoding, ale HashingEncoder zwrocił mniej kolumn. Cieżko wieć domyślić się czym są te kolumny.**

CatBoostEncoder

In [331]:

```
encoder4 = ce.CatBoostEncoder(cols=["main_category"])
df_transformed = encoder4.fit_transform(df, df['price'])
```

In [333]:

```
df_transformed["main_category"].head(20)
```

Out[333]:

```
0     76.811350
1     76.811350
2     76.811350
3     76.811350
4     40.855675
5     33.870450
6     28.150338
7     76.811350
8     76.811350
9     44.405675
10    52.900675
11    68.400675
12    76.811350
13    73.400675
14    58.930450
15    61.597117
16    93.355675
17    24.718270
18    70.200450
19    65.025338
Name: main_category, dtype: float64
```

In [336]:

```
df.main_category.head(20)
```

Out[336]:

```
0                   Komputery
1     Odzież, Obuwie, Dodatki
2                Dom i Ogród
3          Książki i Komiksy
4     Odzież, Obuwie, Dodatki
5     Odzież, Obuwie, Dodatki
6     Odzież, Obuwie, Dodatki
7         Biżuteria i Zegarki
8                   RTV i AGD
9                   RTV i AGD
10        Biżuteria i Zegarki
11                  Komputery
12               Motoryzacja
13               Motoryzacja
14               Motoryzacja
15                  Komputery
16               Dom i Ogród
17    Odzież, Obuwie, Dodatki
18               Dom i Ogród
19               Dom i Ogród
Name: main_category, dtype: object
```

Wygląda podobnie do target encoding. Dla pierwszych 20 wierszy często pojawia się wartość 76.811350, ale kiedy przyjrzymy się jak wcześniej wyglądała kolumna `main_category` to widzimy że ta wartość nie jest przyporządkowana do jednej kategori

Part 2

In [311]:

```
df2 = df[["price", 'it_seller_rating', 'it_quantity']].head(10000)
samp =df2['it_seller_rating'].sample(frac=0.1)
df_nan = df2
df_nan.loc[df_nan.index.isin(samp.index),"it_seller_rating"] = None
df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 3 columns):
 #   Column            Non-Null Count  Dtype  
---  ------            --------------  -----  
 0   price             10000 non-null  float64
 1   it_seller_rating  9000 non-null   float64
 2   it_quantity       10000 non-null  int64  
dtypes: float64(2), int64(1)
memory usage: 234.5 KB
```

Nearest neighbors imputation

In [292]:

```
error1 =[]
for i in range(10):
    np.random.seed(i)
    samp =df2['it_seller_rating'].sample(frac=0.1)
    df_nan = df2
    df_nan.loc[df_nan.index.isin(samp.index),"it_seller_rating"] = None
    df2 = df[["price", 'it_seller_rating', 'it_quantity']].head(10000)
    imputer = KNNImputer(n_neighbors=5)
    df_nni = pd.DataFrame(imputer.fit_transform(df_nan),columns = df_nan.columns)
```

```
    e = np.sqrt(mean_squared_error(df2, df_nni))
    error1.append(e)

statistics.stdev(error1)
```

Out[292]:

```
501.48510756996126
```

Multivariate feature imputation

In [293]:

```
error2 =[]
for i in range(10):
    np.random.seed(i)
    samp =df2['it_seller_rating'].sample(frac=0.1)
    df_nan = df2
    df_nan.loc[df_nan.index.isin(samp.index),"it_seller_rating"] = None
    df2 = df[["price", 'it_seller_rating', 'it_quantity']].head(10000)
    imputer = IterativeImputer()
    imputer.fit(df_nan)
    df_ii = pd.DataFrame(imputer.transform(df_nan),columns = df_nan.columns)
    e = np.sqrt(mean_squared_error(df2, df_ii))
    error2.append(e)

statistics.stdev(error2)
```

Out[293]:

```
570.7481165070046
```

Nearest neighbors imputation - braki w dwóch klumnach

In [294]:

```
error3 =[]
for i in range(10):
    np.random.seed(i)
    samp1 =df2['it_seller_rating'].sample(frac=0.1)
    samp2 =df2['it_seller_rating'].sample(frac=0.1)
    df_nan = df2
    df_nan.loc[df_nan.index.isin(samp1.index),"it_seller_rating"] = None
    df_nan.loc[df_nan.index.isin(samp2.index),"it_quantity"] = None
    df2 = df[["price", 'it_seller_rating', 'it_quantity']].head(10000)
    imputer = KNNImputer(n_neighbors=5)
    df_nni = pd.DataFrame(imputer.fit_transform(df_nan),columns = df_nan.columns)
    e = np.sqrt(mean_squared_error(df2, df_nni))
    error3.append(e)

statistics.stdev(error3)
```

Out[294]:

```
423.9700390739907
```

Multivariate feature imputation - braki w dwóch kolumnach

In [295]:

```
error4 =[]
for i in range(10):
    np.random.seed(i)
    samp1 =df2['it_seller_rating'].sample(frac=0.1)
    samp2 =df2['it_seller_rating'].sample(frac=0.1)
    df_nan = df2
    df_nan.loc[df_nan.index.isin(samp1.index),"it_seller_rating"] = None
    df_nan.loc[df_nan.index.isin(samp2.index),"it_quantity"] = None
```

```
    df2 = df[["price", 'it_seller_rating', 'it_quantity']].head(10000)
    imputer = IterativeImputer()
    imputer.fit(df_nan)
    df_ii = pd.DataFrame(imputer.transform(df_nan),columns = df_nan.columns)
    e = np.sqrt(mean_squared_error(df2, df_ii))
    error4.append(e)

statistics.stdev(error4)
```

Out[295]:

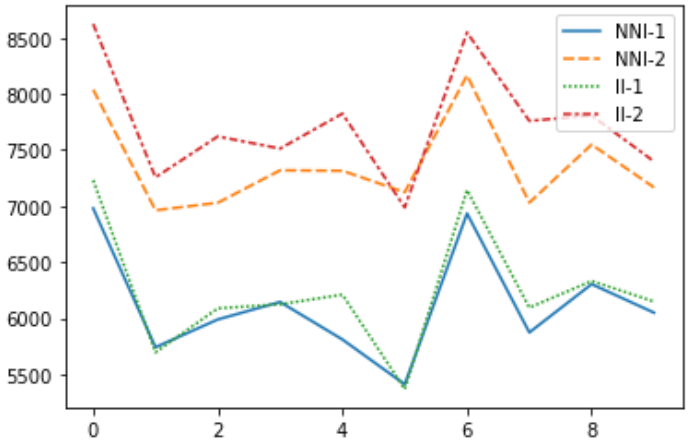
```
519.609113777147
```

In [365]:

```
wyniki = pd.DataFrame([error1,error3,error2,error4])
wyniki = wyniki.transpose()
wyniki.columns =['NNI-1','NNI-2', 'II-1',  'II-2']
sns.lineplot(data=wyniki)
```

Out[365]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x29801d6c070>
```



- **Mniejsze odchylenie kiedy usunęliśmy dane z 2 kolumn**
- **Multivariate działą szybciej niż nni**
- **Podobne wyniki**