PD2-Jan_Smolen-final

March 23, 2021

1 Jan Smoleń PD2

[37]:
```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
import sklearn
import category_encoders as ce
import sklearn.metrics as metrics
import statistics
```

[38]:
```
df=pd.read_csv("allegro-api-transactions.csv")
```

1.1 Kodowanie zmiennych kategorycznych

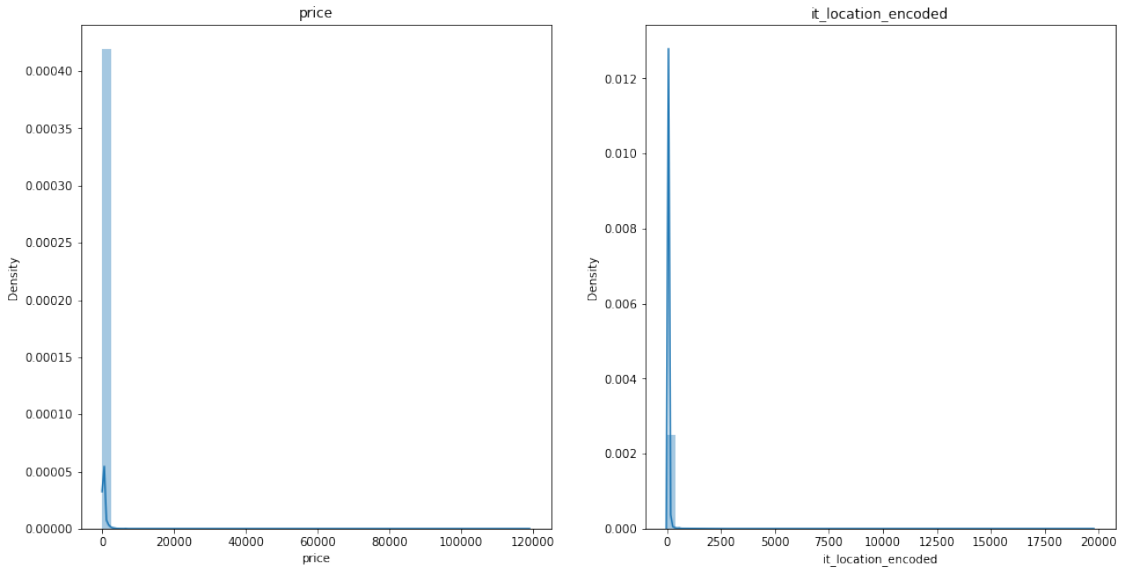
1.1.1 Target Encoding

[55]:
```
encoder=ce.TargetEncoder()
df_target=df.copy()
df_target['it_location_encoded'] = encoder.fit_transform(df['it_location'],␣
 ↪df['price'])
fig, axs= plt.subplots(ncols=2, figsize=(16, 8))
sns.distplot(df_target['price'], ax=axs[0]).set_title("price")
sns.distplot(df_target['it_location_encoded'], ax=axs[1]).
 ↪set_title("it_location_encoded")
```

```
C:\Users\Jan\anaconda3\lib\site-packages\category_encoders\utils.py:21:
FutureWarning: is_categorical is deprecated and will be removed in a future
version.  Use is_categorical_dtype instead
  elif pd.api.types.is_categorical(cols):
C:\Users\Jan\anaconda3\lib\site-packages\seaborn\distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
C:\Users\Jan\anaconda3\lib\site-packages\seaborn\distributions.py:2551:
FutureWarning: `distplot` is a deprecated function and will be removed in a
```

```
future version. Please adapt your code to use either `displot` (a figure-level
function with similar flexibility) or `histplot` (an axes-level function for
histograms).
  warnings.warn(msg, FutureWarning)
```

[55]:
```
Text(0.5, 1.0, 'it_location_encoded')
```



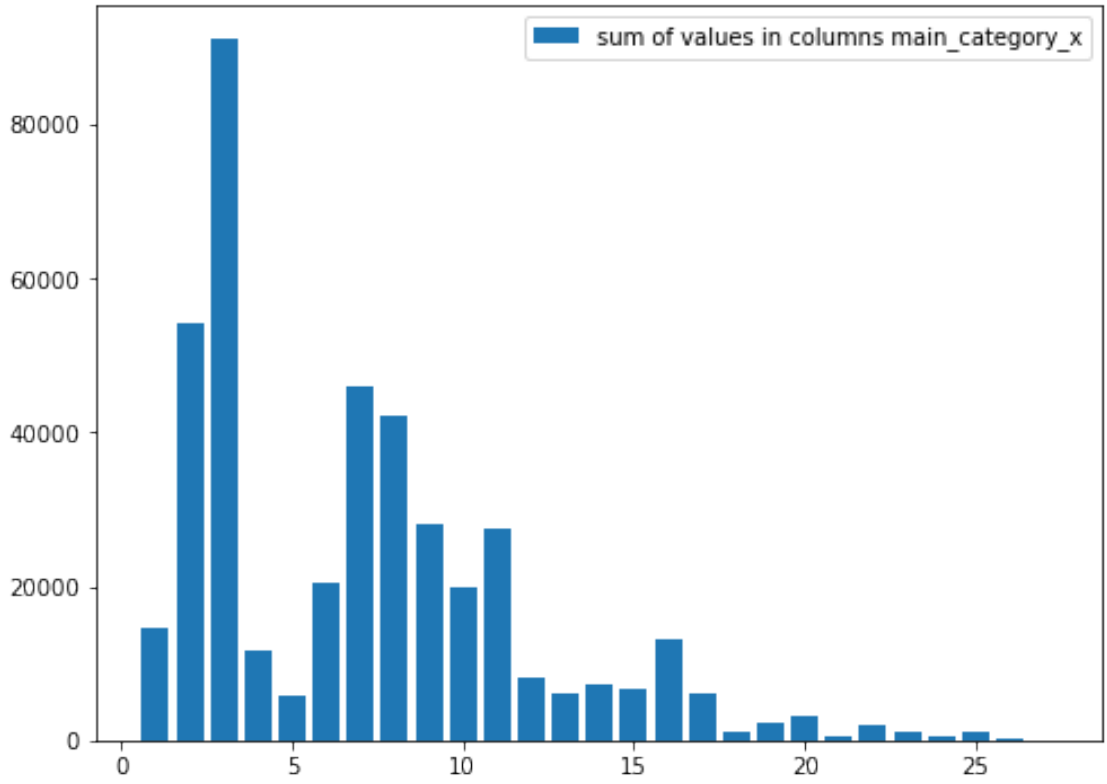
[ ]:
```
#df_target.head()  podgląd ramki danych źle sie konwertuje do PDFa
```

1.1.2 One Hot Encoding

[69]:
```
encoder=ce.OneHotEncoder()
enc_df = pd.DataFrame(encoder.fit_transform(df[['main_category']]))
enc_df=df.join(enc_df).drop("main_category", axis=1)
arr=[]
r1=[]
for i in range(1, 28):
    r1.append(i)
    tmp="main_category_"+str(i)
    arr.append(sum(enc_df[tmp]))
plt.figure(figsize=(8,6))
plt.bar(r1, arr, label="sum of values in columns main_category_x")
plt.legend()
```

```
C:\Users\Jan\anaconda3\lib\site-packages\category_encoders\utils.py:21:
FutureWarning: is_categorical is deprecated and will be removed in a future
version.  Use is_categorical_dtype instead
  elif pd.api.types.is_categorical(cols):
```

[69]: <matplotlib.legend.Legend at 0x1c13e7ab040>



[ ]:
```
#enc_df.head() podgląd ramki danych źle sie konwertuje do PDFa
```

1.1.3 Target Encoding vs One Hot Encoding

Dużą zaletą target encoding jest możliwość zakodowania zmiennych kategorycznych bez potrzeby dodawania wielu dodatkowych kolumn. Wadą za to jest fakt, że używając target encoding możemy narazić się na overfitting - jest ono bardzo zależne od naszych danych.

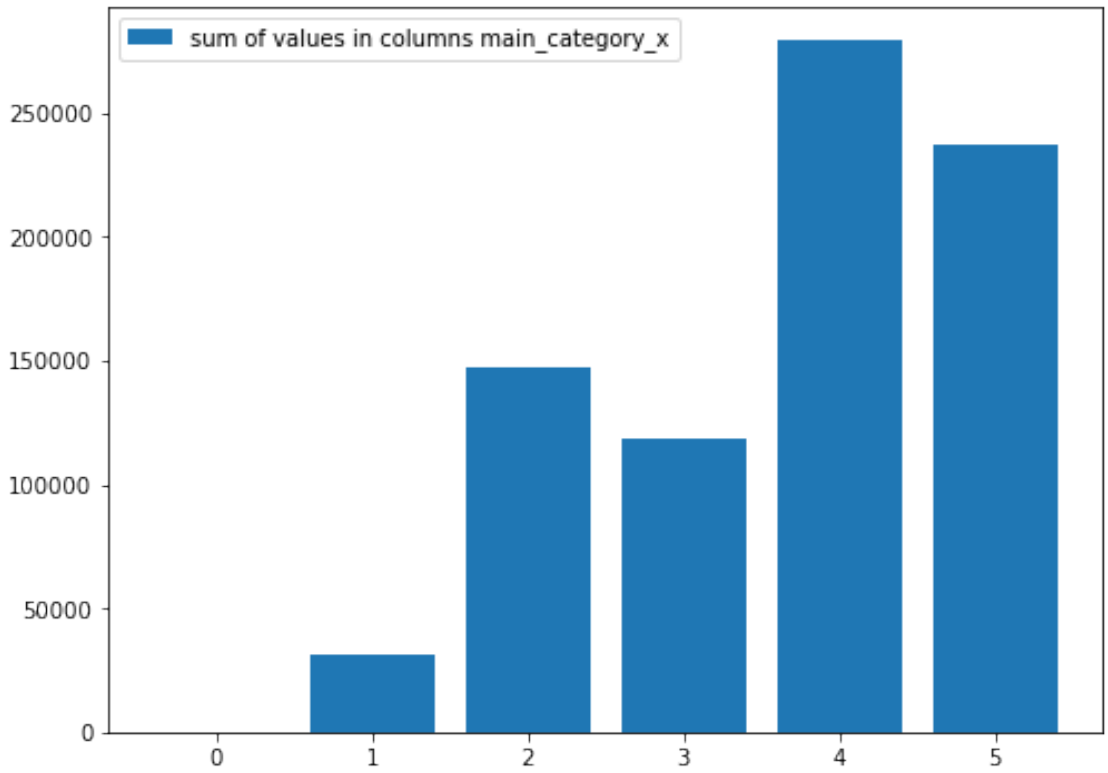
1.1.4 Binary Encoding

[70]:
```
encoder=ce.BinaryEncoder()
enc_df = pd.DataFrame(encoder.fit_transform(df[['main_category']]))
enc_df=df.join(enc_df).drop("main_category", axis=1)
arr=[]
r1=[]
for i in range(6):
    r1.append(i)
    tmp="main_category_"+str(i)
    arr.append(sum(enc_df[tmp]))
plt.figure(figsize=(8,6))
```

```
plt.bar(r1, arr, label="sum of values in columns main_category_x")
plt.legend()
```

```
C:\Users\Jan\anaconda3\lib\site-packages\category_encoders\utils.py:21:
FutureWarning: is_categorical is deprecated and will be removed in a future
version.  Use is_categorical_dtype instead
  elif pd.api.types.is_categorical(cols):
```

[70]: `<matplotlib.legend.Legend at 0x1c13e972fd0>`



[ ]:
```
#enc_df.head() podgląd ramki danych źle sie konwertuje do PDFa
```

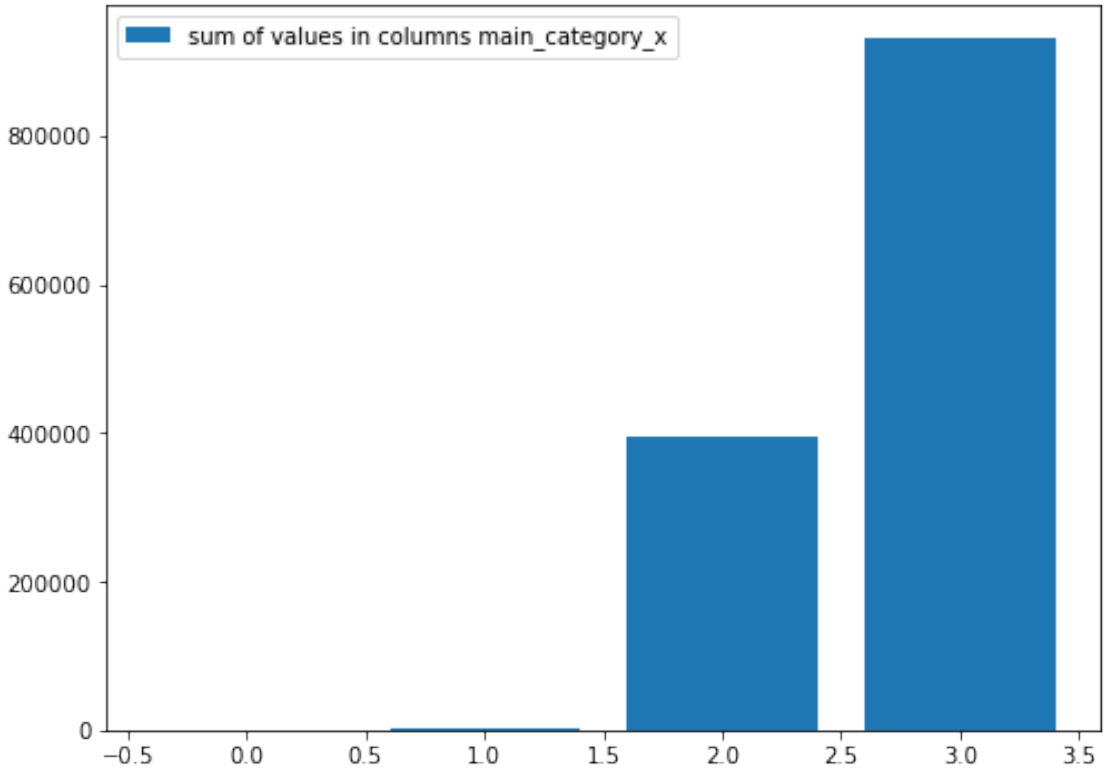
1.1.5 Base N Encoding

[71]:
```
encoder=ce.BaseNEncoder(base=5)
enc_df = pd.DataFrame(encoder.fit_transform(df[['main_category']]))
enc_df=df.join(enc_df).drop("main_category", axis=1)
arr=[]
r1=[]
for i in range(4):
    r1.append(i)
    tmp="main_category_"+str(i)
```

```
    arr.append(sum(enc_df[tmp]))
plt.figure(figsize=(8,6))
plt.bar(r1, arr, label="sum of values in columns main_category_x")
plt.legend()
```

```
C:\Users\Jan\anaconda3\lib\site-packages\category_encoders\utils.py:21:
FutureWarning: is_categorical is deprecated and will be removed in a future
version.  Use is_categorical_dtype instead
  elif pd.api.types.is_categorical(cols):
```

[71]: <matplotlib.legend.Legend at 0x1c13eb84640>



[ ]:
```
#enc_df.head() podgląd ramki danych źle sie konwertuje do PDFa
```

Base N Encoding jest zatem uogólnieniem Binary Encoding na kodowanie w systemie o dowolnej podstawie. Ich przewagą nad one hot encoding jest fakt, że możemy dodać mniejszą ilość kolumn. Wadą Base N Encoding jest natomiast to, że wartości w kolumnach mogą przyjmować wiele wartości, co może komplikować dalsze działanie.

1.2 Uzupełnianie brakujących danych

1.2.1 Nearest Neighbors Imputation

[43]:
```
import numpy as np
from sklearn.impute import KNNImputer
df2=df[['price', 'it_seller_rating', 'it_quantity']].head(1000)
arr1=[]
arr2=[]
imputer = KNNImputer(n_neighbors=2, weights="uniform")
for i in range(10):
    df1=df2.copy()
    df1.loc[df1['it_seller_rating'].sample(frac=0.1, random_state=i).index,␣
 ↪"it_seller_rating"] = np.nan
    df1=pd.DataFrame(imputer.fit_transform(df1), columns=df1.columns)
    arr1.append(metrics.mean_squared_error(df2["it_seller_rating"],␣
 ↪df1["it_seller_rating"], squared=False))
    df1=df2.copy()
    df1.loc[df1['it_seller_rating'].sample(frac=0.1, random_state=i).index,␣
 ↪"it_seller_rating"] = np.nan
    df1.loc[df1['it_quantity'].sample(frac=0.1, random_state=i).index,␣
 ↪"it_quantity"] = np.nan
    df1=pd.DataFrame(imputer.fit_transform(df1), columns=df1.columns)
    arr2.append(metrics.mean_squared_error(df2["it_seller_rating"],␣
 ↪df1["it_seller_rating"], squared=False))
```

[44]:
```
statistics.stdev(arr1)
```

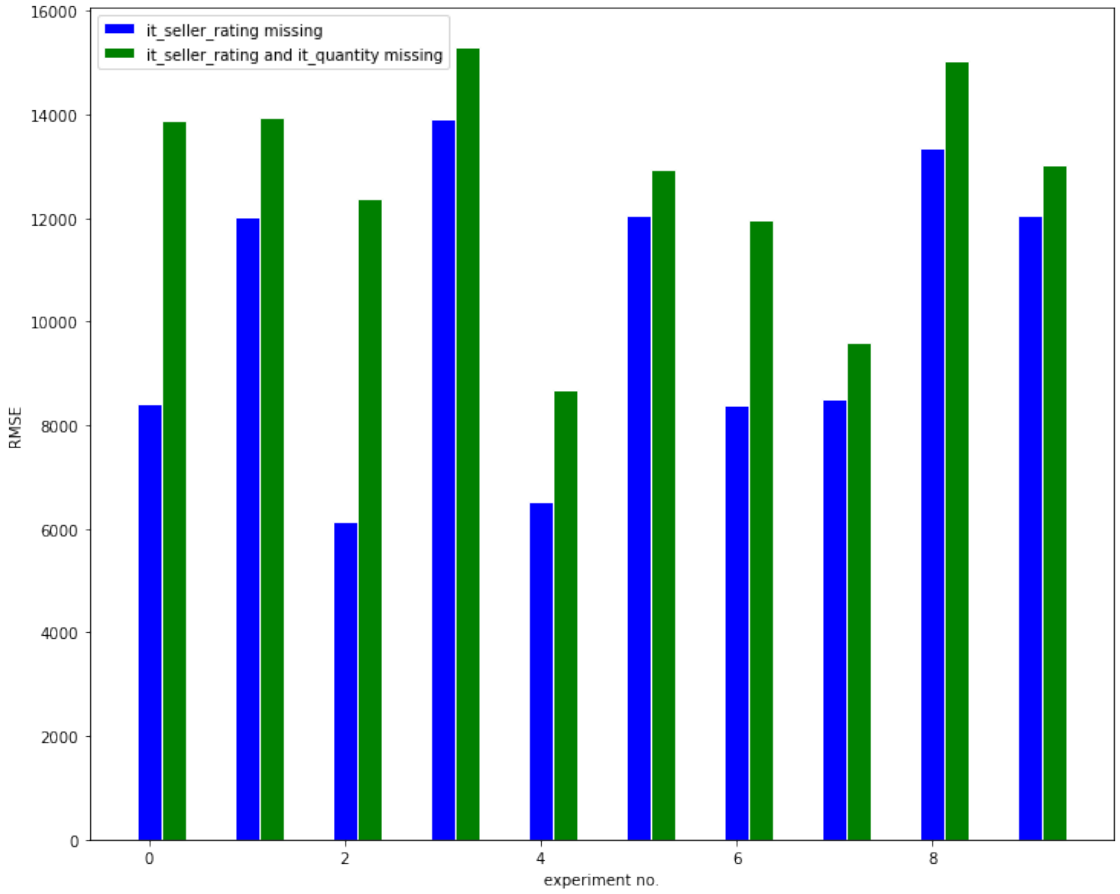
[44]:
```
2854.564534983827
```

[45]:
```
statistics.stdev(arr2)
```

[45]:
```
2148.511208925456
```

[46]:
```
plt.figure(figsize=(12,10))
barWidth = 0.25
r1 = np.arange(len(arr1))
r2 = [x + barWidth for x in r1]
plt.bar(r1, arr1, color='blue', width=barWidth, edgecolor='white',␣
 ↪label='it_seller_rating missing')
plt.bar(r2, arr2, color='green', width=barWidth, edgecolor='white',␣
 ↪label='it_seller_rating and it_quantity missing')
plt.legend()
plt.ylabel("RMSE")
plt.xlabel("experiment no.")
```

[46]:
```
Text(0.5, 0, 'experiment no.')
```



Zgodnie z oczekiwaniami, usunięcie wartości w innej kolumnie wpywa negatywnie na dokładność naszego przybliżenia - naszemu algorytmowi brakuje danych żeby znaleźć najbliższych sąsiadów, których używamy do obliczania brakujących wartości.

[ ]: