ShanneyS_BCA_FinalProject

August 16, 2023

DATA PREPARATION

[1]:
```
import warnings
warnings.filterwarnings('ignore')

#imports and read data
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
import os

df = pd.read_csv("last.csv")
df_raw = df.copy()
```

[2]:
```
#remove unwanted columns (start, end date, casa)
df = df.drop(df.columns[[0,5,9,15,21,27,33,34,35,36,37,38]], axis=1)

#remove some empty value rows
df = df[df.OS_IDR != 0]
df = df[df.PLAFOND_IDR != 0]
df = df[df['AVG_9MTHS_AMT_CR'].notna()]

df = df.reset_index(drop=True)
```

[3]:
```
#new column
df["percent_used"] = df["OS_IDR"]/df["PLAFOND_IDR"]

df["REDFLAG_YELLOW"] = df["REDFLAG_YELLOW"].fillna(0)
df["REDFLAG_RED"] = df["REDFLAG_RED"].fillna(0)
df["REDFLAG_INFORMASI"] = df["REDFLAG_INFORMASI"].fillna(0)
df["total_flags"] = df["REDFLAG_YELLOW"] + df["REDFLAG_RED"]*3 +␣
 ↪df["REDFLAG_INFORMASI"]*5 + df["FLAG_BLACKLIST"]*7
```

[4]:
```
#pd.set_option('display.max_rows', 1000)
```

[5]:
```
#positive is good
df["12to9CR"] = (df["AVG_9MTHS_AMT_CR"] - df["AVG_12MTHS_AMT_CR"])/
 ↪df["AVG_12MTHS_AMT_CR"]
df["9to6CR"] = (df["AVG_6MTHS_AMT_CR"] - df["AVG_9MTHS_AMT_CR"])/
 ↪df["AVG_9MTHS_AMT_CR"]
df["6to3CR"] = (df["AVG_3MTHS_AMT_CR"] - df["AVG_6MTHS_AMT_CR"])/
 ↪df["AVG_6MTHS_AMT_CR"]

df["12to9DB"] = (df["AVG_9MTHS_AMT_DB"] - df["AVG_12MTHS_AMT_DB"])/
 ↪df["AVG_12MTHS_AMT_DB"]
df["9to6DB"] = (df["AVG_6MTHS_AMT_DB"] - df["AVG_9MTHS_AMT_DB"])/
 ↪df["AVG_9MTHS_AMT_DB"]
df["6to3DB"] = (df["AVG_3MTHS_AMT_DB"] - df["AVG_6MTHS_AMT_DB"])/
 ↪df["AVG_6MTHS_AMT_DB"]

df["12to9FCR"] = (df["AVG_9MTHS_FREK_CR"] - df["AVG_12MTHS_FREK_CR"])/
 ↪df["AVG_12MTHS_FREK_CR"]
df["9to6FCR"] = (df["AVG_6MTHS_FREK_CR"] - df["AVG_9MTHS_FREK_CR"])/
 ↪df["AVG_9MTHS_FREK_CR"]
df["6to3FCR"] = (df["AVG_3MTHS_FREK_CR"] - df["AVG_6MTHS_FREK_CR"])/
 ↪df["AVG_6MTHS_FREK_CR"]

df["12to9FDB"] = (df["AVG_9MTHS_FREK_DB"] - df["AVG_12MTHS_FREK_DB"])/
 ↪df["AVG_12MTHS_FREK_DB"]
df["9to6FDB"] = (df["AVG_6MTHS_FREK_DB"] - df["AVG_9MTHS_FREK_DB"])/
 ↪df["AVG_9MTHS_FREK_DB"]
df["6to3FDB"] = (df["AVG_3MTHS_FREK_DB"] - df["AVG_6MTHS_FREK_DB"])/
 ↪df["AVG_6MTHS_FREK_DB"]

from scipy.stats import linregress
import math

df = df.reset_index(drop=True)
df["slopeCR"] = df["12to9CR"]
df["interceptCR"] = df["12to9CR"]
df["slopeDB"] = df["12to9DB"]
df["interceptDB"] = df["12to9DB"]
df["slopeFCR"] = df["12to9FCR"]
df["interceptFCR"] = df["12to9FCR"]
df["slopeFDB"] = df["12to9FDB"]
df["interceptFDB"] = df["12to9FDB"]

#AMT
for i in range(len(df)-1):
    if math.isnan(df["12to9CR"][i]):
        df["slopeCR"][i] = 0
```

```
        df["interceptCR"][i] = 0

for i in range(len(df)-1):
    if math.isnan(df["12to9CR"][i]) or math.isnan(df["9to6CR"][i]) or math.
 ↪isnan(df["6to3CR"][i]):
        if math.isnan(df["12to9CR"][i]):
            df["slopeCR"][i] = 0
            df["interceptCR"][i] = 0
        elif math.isnan(df["9to6CR"][i]):
            df["slopeCR"][i] = 0
            df["interceptCR"][i] = 0
        else:
            x_val = [1,2]
            y_val = [df["12to9CR"][i], df["9to6CR"][i]]
            slope, intercept, r_value, p_value, std_err =␣
 ↪linregress(x_val,y_val)
            df["slopeCR"][i] = slope
    else:
        x_val = [1,2,3]
        y_val = [df["12to9CR"][i], df["9to6CR"][i], df["6to3CR"][i]]
        slope, intercept, r_value, p_value, std_err = linregress(x_val,y_val)
        df["slopeCR"][i] = slope
        df["interceptCR"][i] = intercept

for i in range(len(df)-1):
    if math.isnan(df["12to9DB"][i]):
        df["slopeDB"][i] = 0
        df["interceptDB"][i] = 0

for i in range(len(df)-1):
    if math.isnan(df["12to9DB"][i]) or math.isnan(df["9to6DB"][i]) or math.
 ↪isnan(df["6to3DB"][i]):
        if math.isnan(df["12to9DB"][i]):
            df["slopeDB"][i] = 0
            df["interceptDB"][i] = 0
        elif math.isnan(df["9to6DB"][i]):
            df["slopeDB"][i] = 0
            df["interceptDB"][i] = 0
        else:
            x_val = [1,2]
            y_val = [df["12to9DB"][i], df["9to6DB"][i]]
            slope, intercept, r_value, p_value, std_err =␣
 ↪linregress(x_val,y_val)
            df["slopeDB"][i] = slope
    else:
        x_val = [1,2,3]
        y_val = [df["12to9DB"][i], df["9to6DB"][i], df["6to3DB"][i]]
```

```
        slope, intercept, r_value, p_value, std_err = linregress(x_val,y_val)
        df["slopeDB"][i] = slope
        df["interceptDB"][i] = intercept

#FREK

for i in range(len(df)-1):
    if math.isnan(df["12to9FCR"][i]):
        df["slopeFCR"][i] = 0
        df["interceptFCR"][i] = 0

for i in range(len(df)-1):
    if math.isnan(df["12to9FCR"][i]) or math.isnan(df["9to6FCR"][i]) or math.
 ↪isnan(df["6to3FCR"][i]):
        if math.isnan(df["12to9FCR"][i]):
            df["slopeFCR"][i] = 0
            df["interceptFCR"][i] = 0
        elif math.isnan(df["9to6FCR"][i]):
            df["slopeFCR"][i] = 0
            df["interceptFCR"][i] = 0
        else:
            x_val = [1,2]
            y_val = [df["12to9FCR"][i], df["9to6FCR"][i]]
            slope, intercept, r_value, p_value, std_err =␣
 ↪linregress(x_val,y_val)
            df["slopeFCR"][i] = slope
    else:
        x_val = [1,2,3]
        y_val = [df["12to9FCR"][i], df["9to6FCR"][i], df["6to3FCR"][i]]
        slope, intercept, r_value, p_value, std_err = linregress(x_val,y_val)
        df["slopeFCR"][i] = slope
        df["interceptFCR"][i] = intercept

for i in range(len(df)-1):
    if math.isnan(df["12to9FDB"][i]):
        df["slopeFDB"][i] = 0
        df["interceptFDB"][i] = 0

for i in range(len(df)-1):
    if math.isnan(df["12to9FDB"][i]) or math.isnan(df["9to6FDB"][i]) or math.
 ↪isnan(df["6to3FDB"][i]):
        if math.isnan(df["12to9FDB"][i]):
            df["slopeFDB"][i] = 0
            df["interceptFDB"][i] = 0
        elif math.isnan(df["9to6FDB"][i]):
            df["slopeFDB"][i] = 0
            df["interceptFDB"][i] = 0
```

```
        else:
            x_val = [1,2]
            y_val = [df["12to9FDB"][i], df["9to6FDB"][i]]
            slope, intercept, r_value, p_value, std_err =␣
 ↪linregress(x_val,y_val)
            df["slopeFDB"][i] = slope
    else:
        x_val = [1,2,3]
        y_val = [df["12to9FDB"][i], df["9to6FDB"][i], df["6to3FDB"][i]]
        slope, intercept, r_value, p_value, std_err = linregress(x_val,y_val)
        df["slopeFDB"][i] = slope
        df["interceptFDB"][i] = intercept
```

[6]:
```
#AMT CR
#for null rows in 9to6, 6to3 and 3to1, use slope to predict
for i in range(len(df)-1):
    if math.isnan(df["6to3CR"][i]):
        df["6to3CR"][i] = (df["slopeCR"][i])*3 + df["interceptCR"][i]
        if math.isnan(df["9to6CR"][i]):
            df["9to6CR"][i] = (df["slopeCR"][i])*2 + df["interceptCR"][i]
            if math.isnan(df["12to9CR"][i]):
                df["12to9CR"][i] = (df["slopeCR"][i]) + df["interceptCR"][i]

#for null rows in avg6, avg3 and avg1, use above to calculate
for i in range(len(df)-1):
    if math.isnan(df["AVG_3MTHS_AMT_CR"][i]):
        if math.isnan(df["AVG_3MTHS_AMT_CR"][i]):
            df["AVG_6MTHS_AMT_CR"][i] = df["AVG_9MTHS_AMT_CR"][i] +␣
 ↪(df["AVG_9MTHS_AMT_CR"][i])*(df["9to6CR"][i])
        df["AVG_3MTHS_AMT_CR"][i] = df["AVG_6MTHS_AMT_CR"][i] +␣
 ↪(df["AVG_6MTHS_AMT_CR"][i])*(df["6to3CR"][i])

#create new column 12to3
df["12to3CR"] = (df["AVG_12MTHS_AMT_CR"] - df["AVG_3MTHS_AMT_CR"])/
 ↪df["AVG_12MTHS_AMT_CR"]
df["12to3CR"] = df["12to3CR"].fillna(0)

#AMT DB
for i in range(len(df)-1):
    if math.isnan(df["6to3DB"][i]):
        df["6to3DB"][i] = (df["slopeDB"][i])*3 + df["interceptDB"][i]
        if math.isnan(df["9to6DB"][i]):
            df["9to6DB"][i] = (df["slopeDB"][i])*2 + df["interceptDB"][i]
            if math.isnan(df["12to9DB"][i]):
                df["12to9DB"][i] = (df["slopeDB"][i]) + df["interceptDB"][i]

#for null rows in avg6, avg3 and avg1, use above to calculate
```

```
for i in range(len(df)-1):
    if math.isnan(df["AVG_3MTHS_AMT_DB"][i]):
        if math.isnan(df["AVG_3MTHS_AMT_DB"][i]):
            df["AVG_6MTHS_AMT_DB"][i] = df["AVG_9MTHS_AMT_DB"][i] +␣
 ↪(df["AVG_9MTHS_AMT_DB"][i])*(df["9to6DB"][i])
        df["AVG_3MTHS_AMT_DB"][i] = df["AVG_6MTHS_AMT_DB"][i] +␣
 ↪(df["AVG_6MTHS_AMT_DB"][i])*(df["6to3DB"][i])

#create new column 12to3
df["12to3DB"] = (df["AVG_12MTHS_AMT_DB"] - df["AVG_3MTHS_AMT_DB"])/
 ↪df["AVG_12MTHS_AMT_DB"]
df["12to3DB"] = df["12to3DB"].fillna(0)
```

[7]:
```
#FREK CR
#for null rows in 9to6, 6to3 and 3to1, use slope to predict
for i in range(len(df)-1):
    if math.isnan(df["6to3FCR"][i]):
        df["6to3FCR"][i] = (df["slopeFCR"][i])*3 + df["interceptFCR"][i]
        if math.isnan(df["9to6FCR"][i]):
            df["9to6FCR"][i] = (df["slopeFCR"][i])*2 + df["interceptFCR"][i]
            if math.isnan(df["12to9FCR"][i]):
                df["12to9FCR"][i] = (df["slopeFCR"][i]) + df["interceptFCR"][i]

#for null rows in avg6, avg3 and avg1, use above to calculate
for i in range(len(df)-1):
    if math.isnan(df["AVG_3MTHS_FREK_CR"][i]):
        if math.isnan(df["AVG_3MTHS_FREK_CR"][i]):
            df["AVG_6MTHS_FREK_CR"][i] = df["AVG_9MTHS_FREK_CR"][i] +␣
 ↪(df["AVG_9MTHS_FREK_CR"][i])*(df["9to6FCR"][i])
        df["AVG_3MTHS_FREK_CR"][i] = df["AVG_6MTHS_FREK_CR"][i] +␣
 ↪(df["AVG_6MTHS_FREK_CR"][i])*(df["6to3FCR"][i])

#create new column 12to3
df["12to3FCR"] = (df["AVG_12MTHS_FREK_CR"] - df["AVG_3MTHS_FREK_CR"])/
 ↪df["AVG_12MTHS_FREK_CR"]
df["12to3FCR"] = df["12to3FCR"].fillna(0)

#FREK DB
for i in range(len(df)-1):
    if math.isnan(df["6to3FDB"][i]):
        df["6to3FDB"][i] = (df["slopeFDB"][i])*3 + df["interceptFDB"][i]
        if math.isnan(df["9to6FDB"][i]):
            df["9to6FDB"][i] = (df["slopeFDB"][i])*2 + df["interceptFDB"][i]
            if math.isnan(df["12to9DB"][i]):
                df["12to9FDB"][i] = (df["slopeFDB"][i]) + df["interceptFDB"][i]
df["12to9FDB"] = df["12to9FDB"].fillna(0)
```

```
#for null rows in avg6, avg3 and avg1, use above to calculate
for i in range(len(df)-1):
    if math.isnan(df["AVG_3MTHS_FREK_DB"][i]):
        if math.isnan(df["AVG_3MTHS_FREK_DB"][i]):
            df["AVG_6MTHS_FREK_DB"][i] = df["AVG_9MTHS_FREK_DB"][i] +␣
 ↪(df["AVG_9MTHS_FREK_DB"][i])*(df["9to6FDB"][i])
        df["AVG_3MTHS_FREK_DB"][i] = df["AVG_6MTHS_FREK_DB"][i] +␣
 ↪(df["AVG_6MTHS_FREK_DB"][i])*(df["6to3FDB"][i])

#create new column 12to3
df["12to3FDB"] = (df["AVG_12MTHS_FREK_DB"] - df["AVG_3MTHS_FREK_DB"])/
 ↪df["AVG_12MTHS_FREK_DB"]
df["12to3FDB"] = df["12to3FDB"].fillna(0)
```

[8]:
```
#identify correlation
#retain one representative column from the correlated group
correlation_matrix = df.corr().abs()
repetitive_columns = set()
correlated_columns = []

for column in correlation_matrix.columns:
    correlated_columns_list = correlation_matrix.index[
        (correlation_matrix[column] >= 0.9) & (correlation_matrix[column] < 1)].
 ↪tolist()
    if correlated_columns_list:
        repetitive_columns.update(correlated_columns_list)
        correlated_columns.extend([(column, col, correlation_matrix.loc[column,␣
 ↪col]) for col in correlated_columns_list])

representative_columns = []
for group in correlated_columns:
    representative_columns.append(min(group[:-1], key=lambda col: df[col].
 ↪nunique()))

selected_columns = list(set(df.columns) - repetitive_columns)
print(selected_columns)
print(repetitive_columns)
print("pairs")
for pair in correlated_columns:
    col1, col2, correlation_coefficient = pair
    print(f"{col1} - {col2}: {correlation_coefficient}")
```

```
['REDFLAG_YELLOW', 'CUST_TYPE_CD', 'HARI_TUNGGAKAN', 'interceptFDB', 'slopeCR',
'DEBTOR_CATEGORY', '12to3FCR', 'interceptDB', 'OS_IDR', 'FLAG_RESTRU',
'12to9CR', '12to9DB', '12to3DB', 'interceptFCR', '12to3FDB', 'slopeFCR',
'6to3CR', '12to9FDB', '9to6FDB', '9to6FCR', '6to3FCR', 'REDFLAG_RED', '9to6CR',
'FLAG_BLACKLIST', 'percent_used', 'slopeDB', '12to3CR', 'interceptCR',
```

```
'12to9FCR', 'PLAFOND_IDR', '9to6DB', 'FLAG_DEFERRED', 'COLLECT_KEY', '6to3DB',
'ID_DEB']
{'AVG_6MTHS_DPK', 'AVG_3MTHS_AMT_CR', 'AVG_6MTHS_AMT_CR', 'AVG_6MTHS_FREK_CR',
'AVG_3MTHS_FREK_DB', 'REDFLAG_INFORMASI', 'AVG_6MTHS_FREK_DB',
'AVG_9MTHS_AMT_DB', 'AVG_3MTHS_FREK_CR', 'AVG_12MTHS_FREK_DB', 'total_flags',
'AVG_9MTHS_FREK_DB', 'AVG_6MTHS_AMT_DB', 'AVG_12MTHS_DPK', 'AVG_9MTHS_DPK',
'AVG_9MTHS_AMT_CR', 'AVG_3MTHS_AMT_DB', 'slopeFDB', 'AVG_3MTHS_DPK', '6to3FDB',
'AVG_9MTHS_FREK_CR', 'AVG_12MTHS_AMT_DB', 'AVG_12MTHS_FREK_CR',
'AVG_12MTHS_AMT_CR'}
pairs
AVG_12MTHS_DPK - AVG_9MTHS_DPK: 0.99657961637972
AVG_12MTHS_DPK - AVG_6MTHS_DPK: 0.9855975326095443
AVG_12MTHS_DPK - AVG_3MTHS_DPK: 0.9676517005857695
AVG_12MTHS_AMT_DB - AVG_9MTHS_AMT_DB: 0.9886186951949024
AVG_12MTHS_AMT_DB - AVG_6MTHS_AMT_DB: 0.9677321751818605
AVG_12MTHS_AMT_DB - AVG_3MTHS_AMT_DB: 0.9436324473030611
AVG_12MTHS_AMT_CR - AVG_9MTHS_AMT_CR: 0.9891423945816435
AVG_12MTHS_AMT_CR - AVG_6MTHS_AMT_CR: 0.9707048028591465
AVG_12MTHS_AMT_CR - AVG_3MTHS_AMT_CR: 0.9420538157685553
AVG_12MTHS_FREK_DB - AVG_9MTHS_FREK_DB: 0.9992612227435349
AVG_12MTHS_FREK_DB - AVG_6MTHS_FREK_DB: 0.983640541052276
AVG_12MTHS_FREK_DB - AVG_3MTHS_FREK_DB: 0.9157709417834073
AVG_12MTHS_FREK_CR - AVG_9MTHS_FREK_CR: 0.9982762566530344
AVG_12MTHS_FREK_CR - AVG_6MTHS_FREK_CR: 0.9937467527814176
AVG_12MTHS_FREK_CR - AVG_3MTHS_FREK_CR: 0.9888689591709829
AVG_9MTHS_DPK - AVG_12MTHS_DPK: 0.99657961637972
AVG_9MTHS_DPK - AVG_6MTHS_DPK: 0.9935909933097706
AVG_9MTHS_DPK - AVG_3MTHS_DPK: 0.9772597679355698
AVG_9MTHS_AMT_DB - AVG_12MTHS_AMT_DB: 0.9886186951949024
AVG_9MTHS_AMT_DB - AVG_6MTHS_AMT_DB: 0.981166652193324
AVG_9MTHS_AMT_DB - AVG_3MTHS_AMT_DB: 0.9371674762289948
AVG_9MTHS_AMT_CR - AVG_12MTHS_AMT_CR: 0.9891423945816435
AVG_9MTHS_AMT_CR - AVG_6MTHS_AMT_CR: 0.9867319831215624
AVG_9MTHS_AMT_CR - AVG_3MTHS_AMT_CR: 0.9398525086798611
AVG_9MTHS_FREK_DB - AVG_12MTHS_FREK_DB: 0.9992612227435349
AVG_9MTHS_FREK_DB - AVG_6MTHS_FREK_DB: 0.9884790301201123
AVG_9MTHS_FREK_DB - AVG_3MTHS_FREK_DB: 0.9275474951449867
AVG_9MTHS_FREK_CR - AVG_12MTHS_FREK_CR: 0.9982762566530344
AVG_9MTHS_FREK_CR - AVG_6MTHS_FREK_CR: 0.9977799781141005
AVG_9MTHS_FREK_CR - AVG_3MTHS_FREK_CR: 0.9941282693918777
AVG_6MTHS_DPK - AVG_12MTHS_DPK: 0.9855975326095443
AVG_6MTHS_DPK - AVG_9MTHS_DPK: 0.9935909933097706
AVG_6MTHS_DPK - AVG_3MTHS_DPK: 0.9903369927562047
AVG_6MTHS_AMT_DB - AVG_12MTHS_AMT_DB: 0.9677321751818605
AVG_6MTHS_AMT_DB - AVG_9MTHS_AMT_DB: 0.981166652193324
AVG_6MTHS_AMT_DB - AVG_3MTHS_AMT_DB: 0.9494289477677564
AVG_6MTHS_AMT_CR - AVG_12MTHS_AMT_CR: 0.9707048028591465
AVG_6MTHS_AMT_CR - AVG_9MTHS_AMT_CR: 0.9867319831215624
```

```
AVG_6MTHS_AMT_CR - AVG_3MTHS_AMT_CR: 0.9475009267475655
AVG_6MTHS_FREK_DB - AVG_12MTHS_FREK_DB: 0.983640541052276
AVG_6MTHS_FREK_DB - AVG_9MTHS_FREK_DB: 0.9884790301201123
AVG_6MTHS_FREK_DB - AVG_3MTHS_FREK_DB: 0.9707120025852226
AVG_6MTHS_FREK_CR - AVG_12MTHS_FREK_CR: 0.9937467527814176
AVG_6MTHS_FREK_CR - AVG_9MTHS_FREK_CR: 0.9977799781141005
AVG_6MTHS_FREK_CR - AVG_3MTHS_FREK_CR: 0.9984936905192534
AVG_3MTHS_DPK - AVG_12MTHS_DPK: 0.9676517005857695
AVG_3MTHS_DPK - AVG_9MTHS_DPK: 0.9772597679355698
AVG_3MTHS_DPK - AVG_6MTHS_DPK: 0.9903369927562047
AVG_3MTHS_AMT_DB - AVG_12MTHS_AMT_DB: 0.9436324473030611
AVG_3MTHS_AMT_DB - AVG_9MTHS_AMT_DB: 0.9371674762289948
AVG_3MTHS_AMT_DB - AVG_6MTHS_AMT_DB: 0.9494289477677564
AVG_3MTHS_AMT_CR - AVG_12MTHS_AMT_CR: 0.9420538157685553
AVG_3MTHS_AMT_CR - AVG_9MTHS_AMT_CR: 0.9398525086798611
AVG_3MTHS_AMT_CR - AVG_6MTHS_AMT_CR: 0.9475009267475655
AVG_3MTHS_FREK_DB - AVG_12MTHS_FREK_DB: 0.9157709417834073
AVG_3MTHS_FREK_DB - AVG_9MTHS_FREK_DB: 0.9275474951449867
AVG_3MTHS_FREK_DB - AVG_6MTHS_FREK_DB: 0.9707120025852226
AVG_3MTHS_FREK_CR - AVG_12MTHS_FREK_CR: 0.9888689591709829
AVG_3MTHS_FREK_CR - AVG_9MTHS_FREK_CR: 0.9941282693918777
AVG_3MTHS_FREK_CR - AVG_6MTHS_FREK_CR: 0.9984936905192534
REDFLAG_INFORMASI - total_flags: 0.9542378188220114
total_flags - REDFLAG_INFORMASI: 0.9542378188220114
6to3FDB - slopeFDB: 0.9009604608862583
slopeFDB - 6to3FDB: 0.9009604608862583
```

[9]:
```
df_removeALL = df.drop(df.
↪columns[[7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,35,36,37,38,39,40,41,42,4
↪axis=1)
```

[10]:
```
#df_removeALL.isnull().any()
#df_removeALL.to_csv(r'final_df.csv')
```

[11]:
```
df_removeALL.isnull().any()
```

[11]:
```
ID_DEB              False
DEBTOR_CATEGORY     False
PLAFOND_IDR         False
OS_IDR              False
HARI_TUNGGAKAN       True
CUST_TYPE_CD        False
COLLECT_KEY         False
FLAG_RESTRU         False
FLAG_DEFERRED       False
REDFLAG_YELLOW      False
REDFLAG_RED         False
```

```
REDFLAG_INFORMASI    False
FLAG_BLACKLIST       False
percent_used         False
total_flags          False
slopeCR              False
interceptCR          False
slopeDB              False
interceptDB          False
slopeFCR             False
interceptFCR         False
slopeFDB             False
interceptFDB         False
12to3CR              False
12to3DB              False
12to3FCR             False
12to3FDB             False
dtype: bool
```

[12]:
```
df_npl = df_removeALL[df_removeALL['COLLECT_KEY'] >= 3]
df_pl = df_removeALL[df_removeALL['COLLECT_KEY'] < 3]
df_pl = df_pl.drop(df.columns[[4]], axis=1)
```

[13]:
```
df_npl_I = df_npl[df_npl['CUST_TYPE_CD'] == "I"]
df_npl_O = df_npl[df_npl['CUST_TYPE_CD'] == "O"]
df_pl_I = df_pl[df_pl['CUST_TYPE_CD'] == "I"]
df_pl_O = df_pl[df_pl['CUST_TYPE_CD'] == "O"]
```

ANALYSIS

[14]:
```
#remove unwanted columns (start and end date)
#df_pl = df_pl.drop(df.columns[[4]], axis=1)
print(df_removeALL.groupby(["COLLECT_KEY"]).agg(len)["percent_used"])
```

```
COLLECT_KEY
1    35948
2      626
4      127
5      654
Name: percent_used, dtype: int64
```

[15]:
```
#df_pl['total_flags'].value_counts().sort_index()
#df_pl_I['total_flags'].value_counts().sort_index()
#df_pl_O['total_flags'].value_counts().sort_index()
```

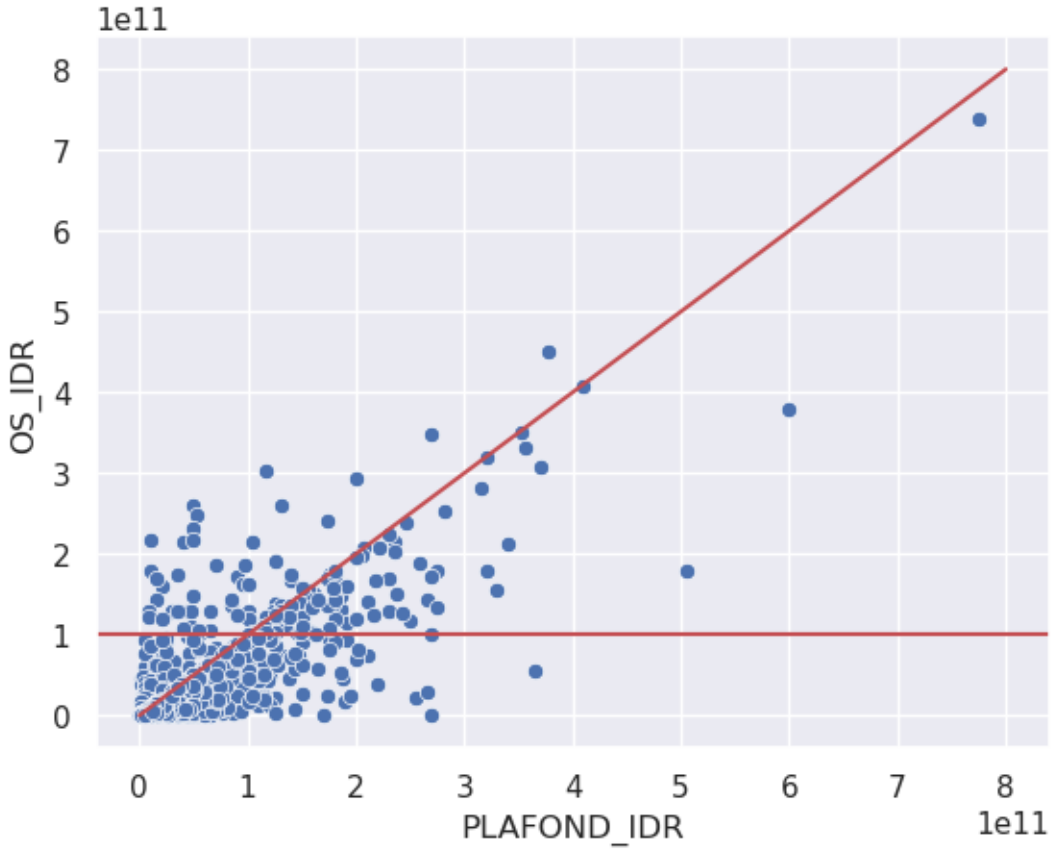
[16]:
```
#fig, ax1 = plt.subplots()
#df_npl['total_flags'].value_counts().sort_index().plot(ax=ax1, kind='bar')
#fig, ax2 = plt.subplots()
#df_npl_I['total_flags'].value_counts().sort_index().plot(ax=ax2, kind='bar')
```

```
#fig, ax3 = plt.subplots()
#df_npl_0['total_flags'].value_counts().sort_index().plot(ax=ax3, kind='bar')
```

[17]:
```
sns.set(style="darkgrid")
my_pal = {1: "r", 2: "y", 4:"g", 5:"b"}

#sns.boxplot(
#    x = 'COLLECT_KEY',
#    y = 'REDFLAG_INFORMASI',
#    data = df_removeALL,
#    palette = my_pal
    #ax = axes[0,0]
#)

#sns.boxplot(
#    x = 'COLLECT_KEY',
#    y = 'REDFLAG_RED',
#    data = df_removeALL,
#    palette = my_pal
     #ax = axes[0,0]
#)

#sns.boxplot(
#    x = 'COLLECT_KEY',
#    y = 'REDFLAG_YELLOW',
#    data = df_removeALL,
#    palette = my_pal
    #ax = axes[0,0]
#)

#sns.boxplot(
#    x = 'COLLECT_KEY',
#    y = 'REDFLAG_YELLOW',
#    data = df_removeALL,
#    palette = my_pal
    #ax = axes[0,0]
#)

#sns.boxplot(
#    x = 'COLLECT_KEY',
#    y = 'total_flags',
#    data = df_removeALL,
#    palette = my_pal
    #ax = axes[0,0]
#)
```

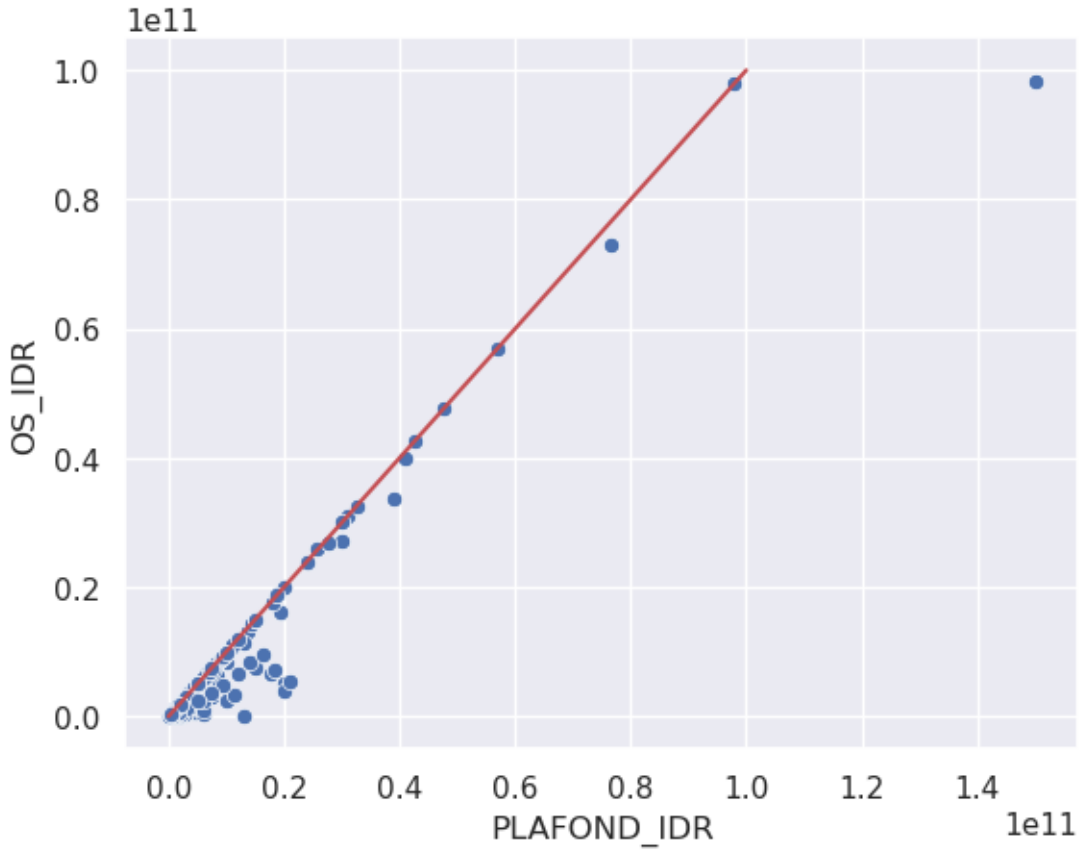
[18]:
```
#x = np
#y = df_removeALL['PLAFOND_IDR']

#plt.scatter(x, y)
#plt.axvline(x=180, color='r')
plt.axhline(y=100000000000, color='r')
#plt.title('hari tunggakan all days')
#plt.show()

#num = 20
#x = df_removeALL['PLAFOND_IDR']
#y = df_removeALL['PLAFOND_IDR']
#labels = np.random.choice(['a', 'b', 'c'], num)
#df = pd.DataFrame(dict(x=x, y=y, label=labels))

sns.scatterplot(x='PLAFOND_IDR', y='OS_IDR', data = df_pl)
plt.plot([10000000, 800000000000], [10000000, 800000000000], color='r')
```

[18]: [<matplotlib.lines.Line2D at 0x7fcd9966db90>]



[19]:
```
sns.scatterplot(x='PLAFOND_IDR', y='OS_IDR', data = df_npl)
plt.plot([10000000, 100000000000], [10000000, 100000000000], color='r')
```

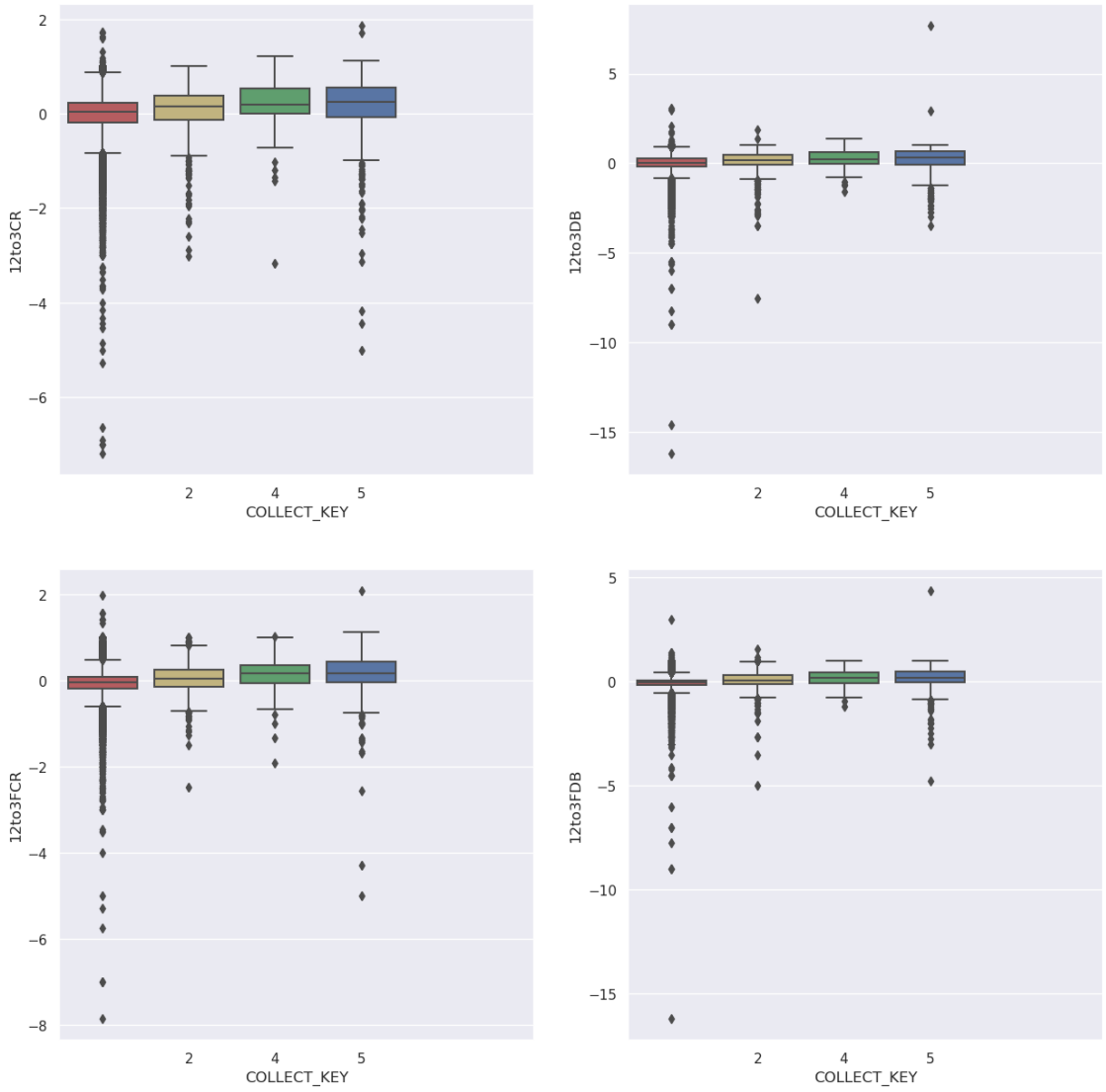
[19]: [<matplotlib.lines.Line2D at 0x7fcd99617590>]



[20]:
```
###
fig, axes = plt.subplots(2,2, figsize=(15,15))
#fig.subtitle('box plot for each cluster')
sns.set(style="darkgrid")
my_pal = {1: "r", 2: "y", 4:"g", 5:"b"}
sns.boxplot(
    x = 'COLLECT_KEY',
    y = '12to3CR',
    data = df_removeALL,
    palette = my_pal,
    ax = axes[0,0]
).set(xticks=list(range(1,6)))

sns.boxplot(
    x = 'COLLECT_KEY',
```

```
    y = '12to3DB',
    data = df_removeALL,
    palette = my_pal,
    ax = axes[0,1]
).set(xticks=list(range(1,6)))

sns.boxplot(
    x = 'COLLECT_KEY',
    y = '12to3FCR',
    data = df_removeALL,
    palette = my_pal,
    ax = axes[1,0]
).set(xticks=list(range(1,6)))

sns.boxplot(
    x = 'COLLECT_KEY',
    y = '12to3FDB',
    data = df_removeALL,
    palette = my_pal,
    ax = axes[1,1]
).set(xticks=list(range(1,6)))
```

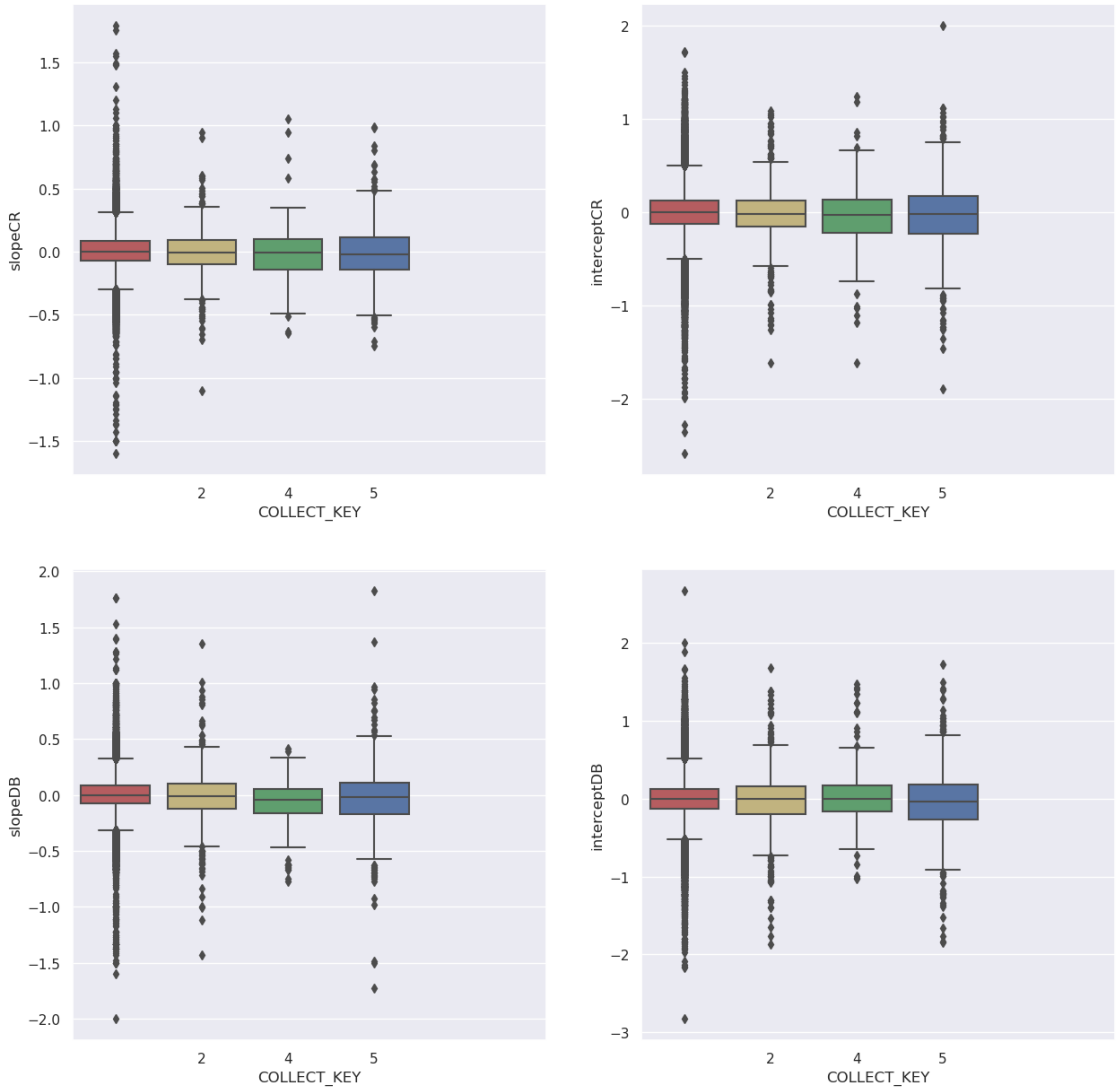
[20]:
```
[[<matplotlib.axis.XTick at 0x7fcd91066d90>,
  <matplotlib.axis.XTick at 0x7fcd91095b10>,
  <matplotlib.axis.XTick at 0x7fcd90f9bcd0>,
  <matplotlib.axis.XTick at 0x7fcd90fd4b90>,
  <matplotlib.axis.XTick at 0x7fcd90ef9010>]]
```



[21]:
```
fig, axes = plt.subplots(2,2, figsize=(15,15))
#fig.subtitle('box plot for each cluster')
sns.set(style="darkgrid")
my_pal = {1: "r", 2: "y", 4:"g", 5:"b"}
sns.boxplot(
    x = 'COLLECT_KEY',
    y = 'slopeCR',
    data = df_removeALL,
    palette = my_pal,
    ax = axes[0,0]
).set(xticks=list(range(1,6)))

sns.boxplot(
```

```
    x = 'COLLECT_KEY',
    y = 'interceptCR',
    data = df_removeALL,
    palette = my_pal,
    ax = axes[0,1]
).set(xticks=list(range(1,6)))

sns.boxplot(
    x = 'COLLECT_KEY',
    y = 'slopeDB',
    data = df_removeALL,
    palette = my_pal,
    ax = axes[1,0]
).set(xticks=list(range(1,6)))

sns.boxplot(
    x = 'COLLECT_KEY',
    y = 'interceptDB',
    data = df_removeALL,
    palette = my_pal,
    ax = axes[1,1]
).set(xticks=list(range(1,6)))
```

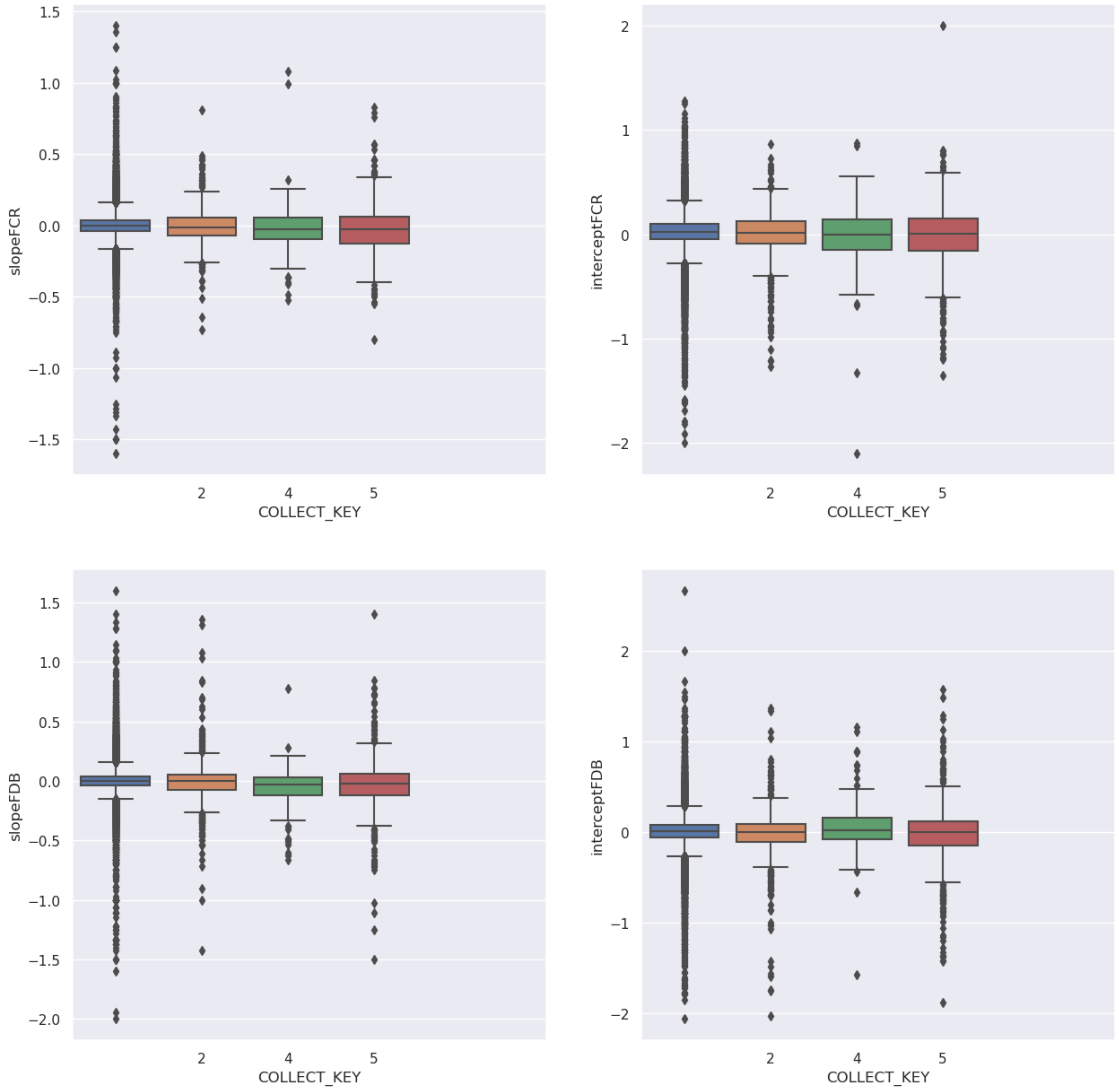
[21]:
```
[[<matplotlib.axis.XTick at 0x7fcd90ff1890>,
  <matplotlib.axis.XTick at 0x7fcd90508210>,
  <matplotlib.axis.XTick at 0x7fcd90411550>,
  <matplotlib.axis.XTick at 0x7fcd90428350>,
  <matplotlib.axis.XTick at 0x7fcd905d1a90>]]
```



[22]:
```
fig, axes = plt.subplots(2,2, figsize=(15,15))
#fig.subtitle('box plot for each cluster')

sns.boxplot(
    x = 'COLLECT_KEY',
    y = 'slopeFCR',
    data = df_removeALL,
    ax = axes[0,0]
).set(xticks=list(range(1,6)))

sns.boxplot(
    x = 'COLLECT_KEY',
    y = 'interceptFCR',
    data = df_removeALL,
```

```
    ax = axes[0,1]
).set(xticks=list(range(1,6)))

sns.boxplot(
    x = 'COLLECT_KEY',
    y = 'slopeFDB',
    data = df_removeALL,
    ax = axes[1,0]
).set(xticks=list(range(1,6)))

sns.boxplot(
    x = 'COLLECT_KEY',
    y = 'interceptFDB',
    data = df_removeALL,
    ax = axes[1,1]
).set(xticks=list(range(1,6)))
```

[22]:
```
[[<matplotlib.axis.XTick at 0x7fcd904eed90>,
  <matplotlib.axis.XTick at 0x7fcd9037e290>,
  <matplotlib.axis.XTick at 0x7fcd90e47e90>,
  <matplotlib.axis.XTick at 0x7fcd902b8b90>,
  <matplotlib.axis.XTick at 0x7fcd9027c590>]]
```



LOGISTIC REGRESSION TO PREDICT KEY

[23]:
```
df_ml = df_removeALL.drop(df_removeALL.
  ↪columns[[0,1,2,3,5,7,8,10,12,15,16,17,18,19,20,21,22]], axis=1)
```

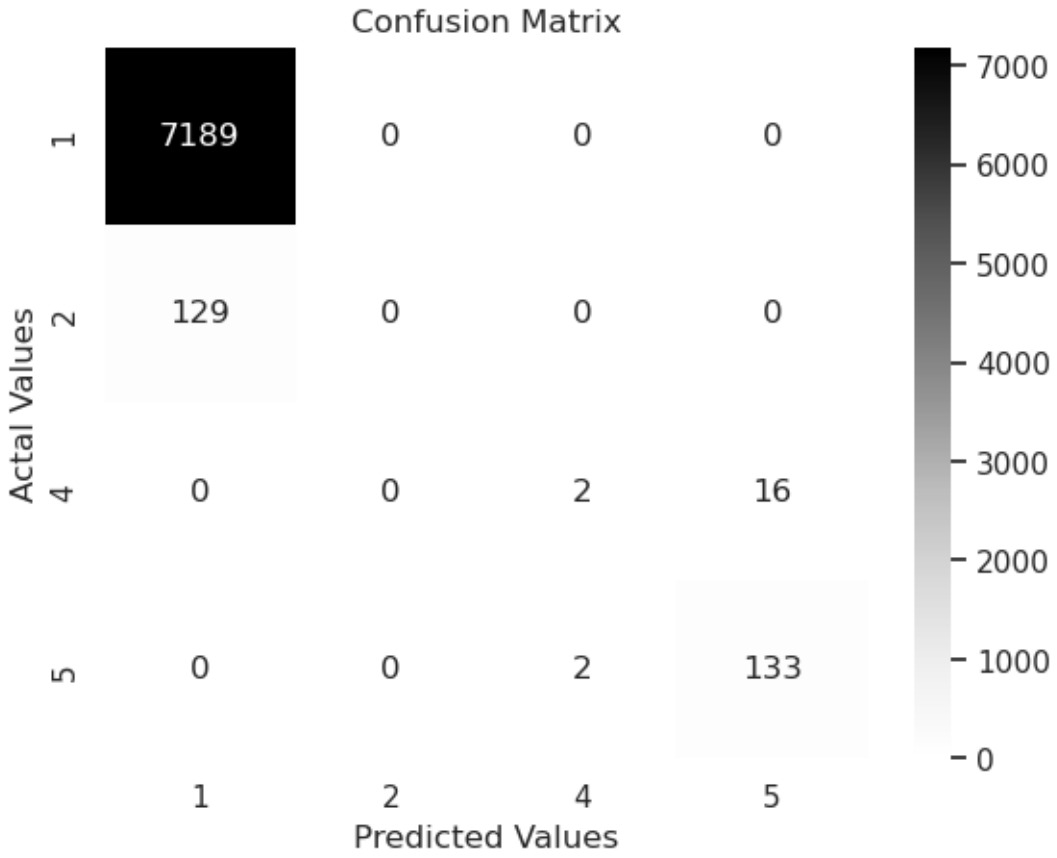
[24]:
```
df_ml["HARI_TUNGGAKAN"] = df_ml["HARI_TUNGGAKAN"].fillna(0)

from sklearn.model_selection import train_test_split
train, test = train_test_split(df_ml, test_size=0.2)

x_train = train.drop('COLLECT_KEY', axis=1)
y_train = train['COLLECT_KEY']

x_test = test.drop('COLLECT_KEY', axis=1)
```

```
y_test = test['COLLECT_KEY']
```

[25]:
```
from sklearn import linear_model

clf = linear_model.LogisticRegression(solver='saga', max_iter=300)
clf.fit(x_train, y_train)
pred = clf.predict(x_test)
```

[26]:
```
result = pd.DataFrame(pred, columns=['pred'])
result['actual'] = y_test.reset_index(drop=True)
```

[27]:
```
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(result['actual'], result['pred'], labels = [1,2,4,5])
cm_df = pd.DataFrame(cm, index = [1,2,4,5], columns = [1,2,4,5])

sns.heatmap(cm_df, annot=True, cmap="Greys", fmt='g')
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



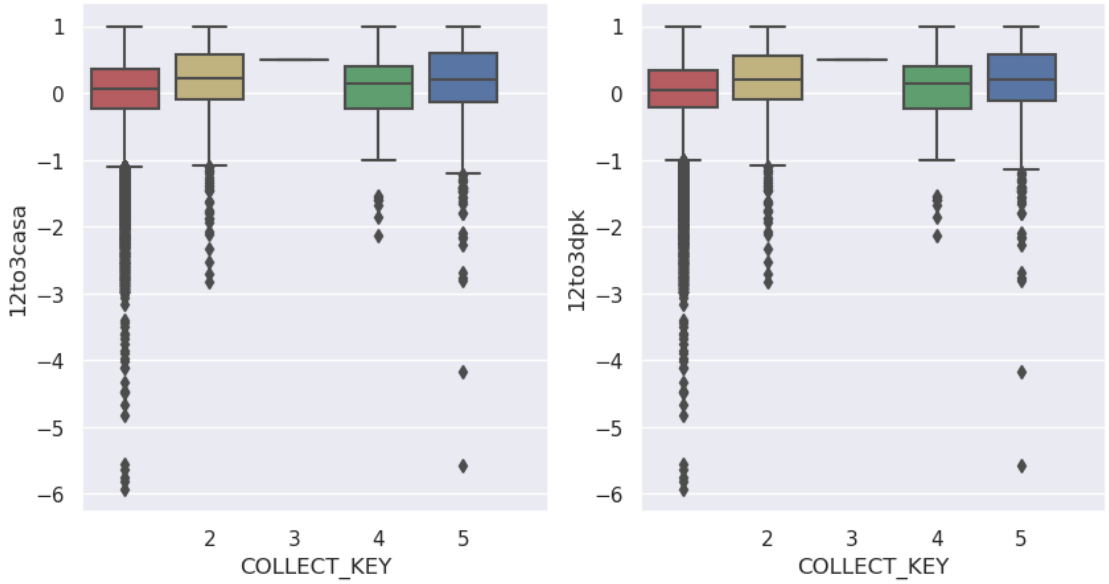
[28]:
```
df_cd = df_raw[['COLLECT_KEY', 'AVG_12MTHS_CASA',␣
 ↪'AVG_12MTHS_DPK','AVG_9MTHS_CASA', 'AVG_9MTHS_DPK', 'AVG_6MTHS_CASA',␣
 ↪'AVG_6MTHS_DPK', 'AVG_3MTHS_CASA', 'AVG_3MTHS_DPK']]
```

[29]:
```
df_cd = df_cd[df_cd['AVG_12MTHS_CASA'].notna()]
df_cd = df_cd[df_cd['AVG_12MTHS_DPK'].notna()]
df_cd = df_cd[df_cd['AVG_9MTHS_CASA'].notna()]
df_cd = df_cd[df_cd['AVG_9MTHS_DPK'].notna()]
df_cd = df_cd[df_cd['AVG_6MTHS_CASA'].notna()]
df_cd = df_cd[df_cd['AVG_6MTHS_DPK'].notna()]
df_cd = df_cd[df_cd['AVG_3MTHS_CASA'].notna()]
df_cd = df_cd[df_cd['AVG_3MTHS_DPK'].notna()]
df_cd["12to3casa"] = (df_cd["AVG_12MTHS_CASA"] - df_cd["AVG_3MTHS_CASA"])/
 ↪df_cd["AVG_12MTHS_CASA"]
df_cd["12to3dpk"] = (df_cd["AVG_12MTHS_DPK"] - df_cd["AVG_3MTHS_DPK"])/
 ↪df_cd["AVG_12MTHS_DPK"]
```

[30]:
```
fig, axes = plt.subplots(1,2, figsize=(10,5))
#fig.subtitle('box plot for each cluster')


sns.set(style="darkgrid")
my_pal = {1: "r", 2: "y", 3: "grey", 4:"g", 5:"b"}

sns.boxplot(
    x = 'COLLECT_KEY',
    y = '12to3casa',
    data = df_cd,
    palette = my_pal,
    ax = axes[0]
).set(xticks=list(range(1,6)))

sns.boxplot(
    x = 'COLLECT_KEY',
    y = '12to3dpk',
    data = df_cd,
    palette = my_pal,
    ax = axes[1]
).set(xticks=list(range(1,6)))
```

[30]:
```
[[<matplotlib.axis.XTick at 0x7fcd8a46cbd0>,
  <matplotlib.axis.XTick at 0x7fcd8a4a9f90>,
  <matplotlib.axis.XTick at 0x7fcd8a412bd0>,
  <matplotlib.axis.XTick at 0x7fcd8a280a90>,
```

```
<matplotlib.axis.XTick at 0x7fcd8a282450>]]
```



[31]:
```
df_1 = df_removeALL[df_removeALL['COLLECT_KEY'] == 1]
df_2 = df_removeALL[df_removeALL['COLLECT_KEY'] == 2]
df_3 = df_removeALL[df_removeALL['COLLECT_KEY'] == 3]
df_4 = df_removeALL[df_removeALL['COLLECT_KEY'] == 4]
df_5 = df_removeALL[df_removeALL['COLLECT_KEY'] == 5]
df_5[["12to3CR", "12to3DB", "12to3FCR", "12to3FDB"]].describe()
```

[31]:

	12to3CR	12to3DB	12to3FCR	12to3FDB
count	654.000000	654.000000	654.000000	654.000000
mean	0.138670	0.208409	0.151356	0.185612
std	0.690041	0.749481	0.512084	0.595897
min	-5.000000	-3.469467	-5.000000	-4.760000
25%	-0.070857	-0.092470	-0.049405	-0.043226
50%	0.244202	0.300917	0.168448	0.185108
75%	0.552754	0.687958	0.437500	0.505422
max	1.862868	7.683775	2.080000	4.375000

[ ]: