|  |  |  |
| --- | --- | --- |
| Indeks | Adam Bednarski 260338 |  |
| Grupa | E | |

W miarę możliwości/potrzeby proszę o dodatkowy komentarz do kodu – w szczególności, warto zaznaczyć zrealizowane dodatkowe elementy, czy skomentować przyjętą interpretację pewnych elementów składowych.

Celem zadań jest zapoznanie się z możliwościami rozszerzenia funkcjonalności Power BI Desktop o własne skrypty przygotowane w języku Python – zarówno w warstwie dostępu do danych, warstwie przetwarzania danych oraz warstwie wizualizacji danych.

Korzystając z danych z zewnętrznego API (NBP) chcemy uzupełnić model z poprzedniej listy zadań o możliwość wyrażenia wartości sprzedaży (w firmie AW) w PLN (wartości sprzedaży wyznaczone są w walucie USD, a chcielibyśmy je przeliczyć na PLN – interesują nas jedynie kwoty zamówień).

Wszystkie zadania zrealizuj w ramach jednego pliku .pbix.

# Zadanie 1

Jako rozwiązanie zadania prześlij kod źródłowy przygotowanej funkcji oraz zrzut ekranu przedstawiający wykres zmiany kursu walut.

## Funkcja

def import\_data(currency, dates):

    start, end = dates

    url = f'http://api.nbp.pl/api/exchangerates/rates/a/{currency}/{start}/{end}/'

    response\_api = requests.get(url)

    data = response\_api.text.encode().decode('utf-8-sig')

    parse\_json = json.loads(data)

    df = pd.DataFrame(parse\_json)

    return df

def clean\_data(df):

    df['rates'].apply(pd.Series)

    df = pd.concat([df, df['rates'].apply(pd.Series)], axis=1)

    df.drop(columns=['rates', 'table'], inplace=True)

    return df

currency = 'usd'

date\_range = ('2022-01-03', '2023-01-03')

data = import\_data(currency, date\_range)

rates = clean\_data(data)

## Wykres

Obraz zawierający wykres

Opis wygenerowany automatycznie

## Konkluzje dot. zadaniA 1

W podpunkcie dotyczącym pobrania danych z API zdecydowaliśmy się zaimplementować funkcje w następujący sposób:

1. Zamiast przekazywać funkcji wartość X odpowiadającą za liczbę ostatnich dni, my zdecydowaliśmy się na przekazywanie zakresu dat (data początkowa i data końcowa) dla których mają się pobrać dane. Powodem jest ograniczenie API na liczbę maksymalnie zwracanych dni, więc według nas może bardziej elastycznym podejściem będzie umożliwienie wprowadzenia dowolnego zakresu dat (nie rozwiązuje to problemu limitu API ale pozwala użytkownikowi na więcej możliwości ekstrakcji danych.
2. Dane od razu konwertujemy na typ DataFrame z biblioteki Pandas, jako że PowerBI dobrze radzi sobie z takim typem danych. Uprzednio występuje dekodowanie danych tekstowych z formatu utf-8-sig, gdyż domyślnie API pobrało nam je właśnie w takiej formie, a na potrzeby następnych kroków (patrz pkt 3) dane te musiały być zdekodowane.
3. Jeszcze z poziomu skryptu Pythona „czyścimy dane”, zamiast robić to później w PowerBI. Kolumnę z wartościami no, effectiveDate oraz mid rozbijamy na 3 kolumny (chyba oczywisty zabieg). Następnie usuwamy zbędne atrybuty (table - mający tę samą wartość dla każdego rekordu oraz rates – pozostałość po wcześniejszej separacji kolumny).
4. Na wykresie nieczytelne są wartości z osi X, gdyż jako okres do wizualizacji wybraliśmy pełen rok od 01.03.2022 do 01.03.2023 (nie zaczynamy od 1 stycznia gdyż do 3 stycznia baza nie posiada danych o kursie waluty).

# Zadanie 2

Jako rozwiązanie zadania prześlij kod zapytania zawierającego wszystkie daty oraz skrypt Python.

## KOD – POWER QUERY – Zapytania

1. **dates:**

let

Source = Python.Execute("import pandas as pd#(lf)#(lf)dates = pd.date\_range(start='2022-01-03', end='2023-01-03')#(lf)dates = pd.to\_datetime(dates, format='%Y-%d-%m')#(lf)df = pd.DataFrame(dates)"),

df = Source{[Name="df"]}[Value],

#"Changed Type" = Table.TransformColumnTypes(df,{{"0", type text}}),

#"Renamed Columns" = Table.RenameColumns(#"Changed Type",{{"0", "effectiveDate"}})

in

#"Renamed Columns"

1. **fixed\_rates:**

let

Source = Table.FromRows(Json.Document(Binary.Decompress(Binary.FromText("i44FAA==", BinaryEncoding.Base64), Compression.Deflate)), let \_t = ((type nullable text) meta [Serialized.Text = true]) in type table [Column1 = \_t]),

#"Changed Type" = Table.TransformColumnTypes(Source,{{"Column1", type text}}),

#"Run Python script" = Python.Execute("# Python:#(lf)import pandas as pd#(lf)df3 = pd.merge(df1, df2, how = 'right', on = ['effectiveDate'])#(lf)df3 = df3.sort\_values(by=['effectiveDate'])#(lf)df3 = df3.fillna(method='ffill')",[df1=rates, df2=dates]),

df3 = #"Run Python script"{[Name="df3"]}[Value],

#"Sorted Rows" = Table.Sort(df3,{{"effectiveDate", Order.Ascending}})

in

#"Sorted Rows"

## Kod – PYTHON

dates = pd.date\_range(start='2022-01-03', end='2023-01-03')

dates = pd.to\_datetime(dates, format='%Y-%d-%m')

dates = pd.DataFrame(dates)

dates = dates.astype(str)

df3 = pd.merge(rates, dates, how='right', left\_on=['effectiveDate'], right\_on=0)

df3.fillna(method='ffill')

## Konkluzje dot. zadaniA 2

Z polecenia w zadaniu 2 zrozumieliśmy, że cała realizacja tego zadania ma być zaimplementowana w skrypcie pythonowym, który później użyjemy w PowerBI. Tak więc pod nagłówkiem KOD – POWERQUERY-ZAPYTANIA umieściliśmy kod z PowerBI z każdego kroku jaki wykonaliśmy dla utworzenia dwóch zapytań (odpowiednio kod pod numerem 1 to kod do utworzenia zapytania z samymi datami, a kod pod numerem 2 to kod do utworzenia zapytania z uzupełnionymi danymi).

Konkluzje dotyczące skryptu z Pythona: utworzyliśmy listę dat z zadanego zakresu (takiego samego jak w zadaniu 1) i zapisaliśmy ją jako DataFrame z typem danych tekstowym (wymagane do złączenia tabel w dalszej części zadania). Tabele złączyliśmy prawostronnie względem tabeli z datami, aby dane w trzeciej tabeli miały każdy dzień z zadanego zakresu dat (łączyliśmy tabelę po atrybucie z datą), a następnie używając funkcji „ffill” uzupełniliśmy brakujące dane dla wartości Null (nie tylko kursy walut ale też nazwa waluty).

Konkluzje dotyczące zapytania z PowerQuery: aby zrealizować te zadanie trzeba było przeprowadzić szereg kroków:

1. Dla tabeli z danymi pobranymi z API:
   1. typ danych dla effectiveDate musi być tekstowy (aby móc łączyć tabele – pandas.merge nie łączył nam tabel gdy daty były w formacie innym niż string)
   2. dane posortowane względem effectiveDate aby poprawnie zadziałała funkcja *fillna* w dalszej części zadania
2. Dla tabeli z samymi datami:
3. typ oraz nazwa kolumny musi się zgadzać z kolumną w tabeli z danymi pobranymi z API

# Zadanie 3

Jako rozwiązanie zadania prześlij kod źródłowy Python oraz zrzut ekranu gotowego raportu (wizualizacje).

## KOD – PYTHON

## Raport – zrzut ekranu

## Konkluzje dot. zadaniA 3

### PODSUMOWANIE – Konkluzje do całej listy zadań