Dokumentacja projektu

Wizualizacja danych dotyczących największych koncernów i producent samochodów

Autorzy: Kacper Fornalczyk, Antoni Kowalski, Wojciech Osiak, Michał Zawadzki, Małgorzata Woźniak

Spis treści

1. Cel projektu

Celem projektu było stworzenie wizualizacji przy użyciu oprogramowania do wizualizacji jakim jest PowerBI na temat największych koncernów i producent samochodów z całego świata. Dane te miały dotyczyć informacji mówiących o tym jak owi producenci radzą sobie na giełdzie, jaką pozycję względem konkurencji zajmują, jaki jest ich kapitał, ceny akcji etc. Jednocześnie chcieliśmy w jakiś sposób zobrazować popularność i opinie panujące na temat konkretnych producentów i zestawić je w raz z wynikami giełdowymi.

2. Podstawowe założenia

Naszymi podstawowymi założeniami było:

- stworzenie wizualizacji w oparciu o zróżnicowane formy danych pobierane cyklicznie z różnych źródeł
- opracowanie i postawienie bazy danych do składowania danych
- uzyskanie bezpośredniej łączności z bazą z wnętrza skryptów odpowiedzialnych za pobieranie danych, oraz łączności między bazą danych a oprogramowaniem PowerBI
- Wykorzystanie technologii Apache Airflow do utworzenia potoku przepływu pracy skryptów uzupełniających bazę o nowe dane.
- To, aby wizualizacje były ciekawe i zrozumiałe, przy dające możliwość interaktywności.

3. Korzyści z programu

Podstawowymi korzyściami z naszego programu dla użytkownika są:

- wgląd do danych giełdowych największych producentów samochodów
- możliwość obejrzenia zmienności pewnych wartości odnoszących się do giełdy w pewnym zakresie czasowym
- możliwość zobaczenia pozycji konkretnego producenta na tle konkurencji
- możliwość porównania wyników producentów między sobą
- wgląd do informacji świadczących o opiniach i popularności danej marki producenckiej.
- uzyskanie informacji o kapitale i jego zmienności producentów aut w zadanym przedziale czasu.

4. Opis rozwiązania

4.1. Wprowadzenie

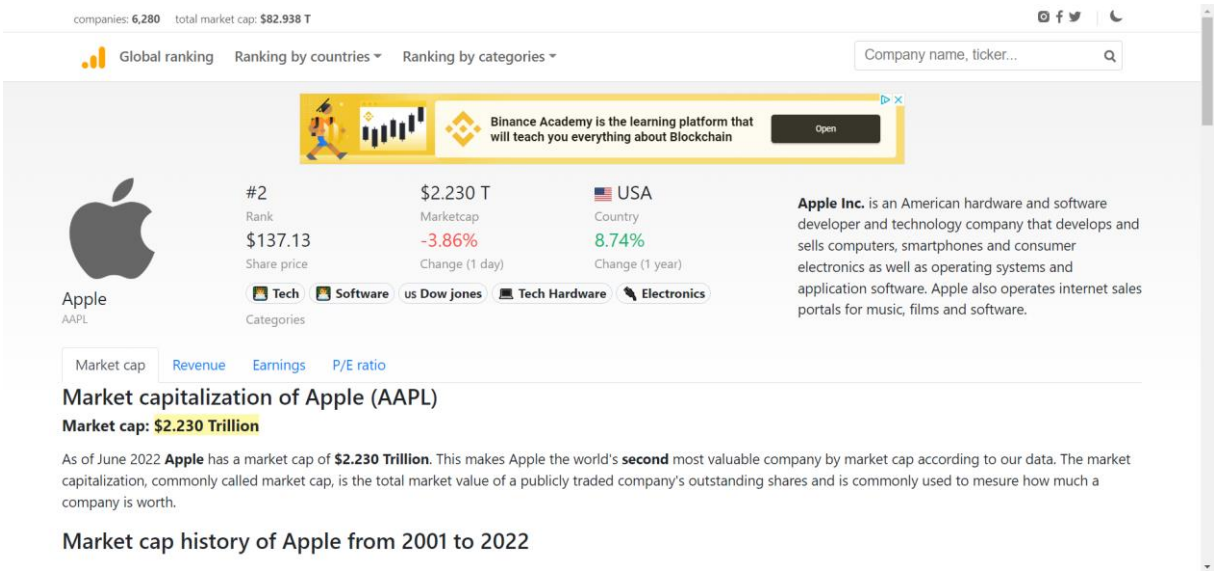
Zgodnie z założeniami nasze rozwiązanie zrealizowane wykorzystuje różnych źródła danych, a wizualizacje stworzone zostały w programie PowerBI. Dane są przechowywane w zaprojektowanej bazie danych SQL Server będącej na zasobach platformy Microsoft Azure, gdzie są wysyłane poprzez napisane w języku python autorskie skrypty, które najpierw pobierają dane, czyszczą je, konwertują do odpowiednich formatów a na koniec umieszczają w bazie. Do cyklicznego uruchamiania skryptów wykorzystana została technologia Apache

Airflow, której środowisko zostało postawiona na utworzonej na zasobach uczelni maszynie wirtualnej z systemem Linux Ubuntu. Poniżej w osobnych sekcjach opisane zostały szczegóły dotyczące konkretnych elementów zbudowanego, przed chwilą opisanego systemu.

4.2. Źródła danych

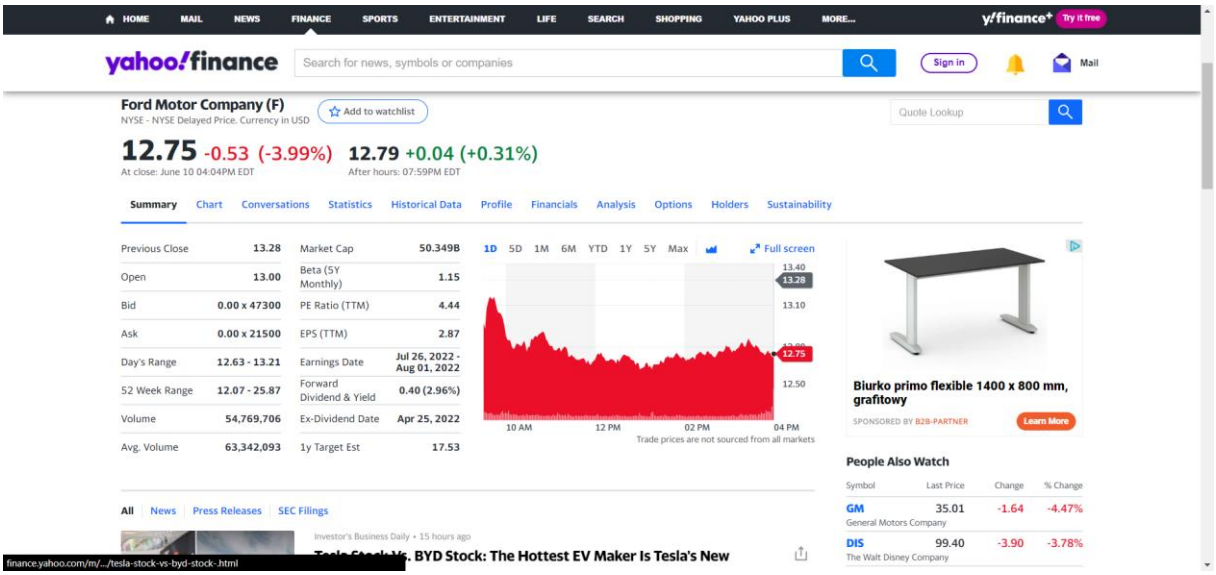
Nasze rozwiązanie czerpie dane w oparciu o 3 różne źródła danych:

1. Serwis internetowy **companiesmarketcap.com**
   Serwis ten dostarcza informacje na temat różnych koncernów, firm, organizacji. Dostarcza między innymi danych na temat cen akcji, kapitału, czym dana firma się zajmuje, jakiej narodowości jest, jej pozycje na świecie i wiele innych. Niektóre dane dotyczące, np.: giełdy, aktualizowane są tutaj co kilkanaście sekund. Strona nie umożliwia łatwego sposobu na pobieranie danych (brak możliwości generowania raportów, brak api) więc nasze rozwiązanie na pozyskanie tych informacji, które zostało opisane później opiera się na web scrapingu. Poniżej zrzut z tego jak wygląda podstrona zawierająca informacje o firmie Apple



   Link do serwisu: https://companiesmarketcap.com/

2. Yahoo!Finance
   Jest to własność medialna będąca częścią Yahoo!. Dostarcza wiadomości finansowe, dane i komentarze, w tym notowania giełdowe, komunikaty prasowe, raporty finansowe i oryginalne treści. Do tego serwisu zostało utworzone specjalne api w języku python pozwalające pobierać wyżej wymienione dane. Poniżej zrzut z podstrony z informacjami o firmie Ford:



Link do serwisu: https://finance.yahoo.com/

3. Twitter
   Ostatnim źródłem danych jest twitter. W języku python zostało stworzone api pozwalające pobierać treści tweetów dla utworzonych z danego konta. Nasz skrypt pobiera za pośrednictwem tego api nowo pojawiające się tweety i próbuje na podstawie ich treści oceniać czy ta tresc jest pozytywna lub negatywna w odniesieniu do firmy.

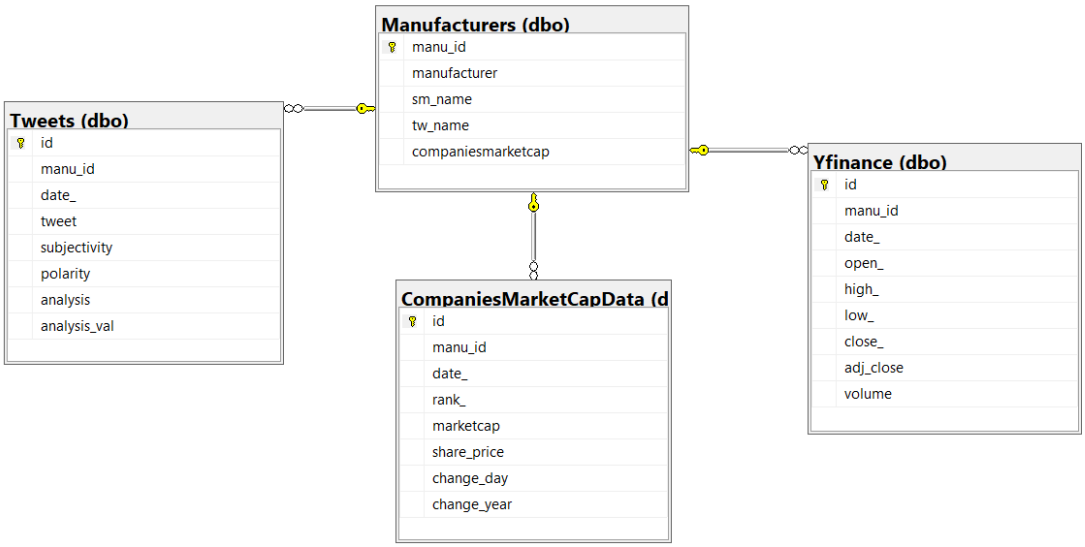
   Link do twittera: https://twitter.com/

4.3. Baza danych

4.3.1. SQL Server

W celu realizacji naszego projektu potrzebowaliśmy bazy danych do której każdy z członków zespołu miałby swobodny dostęp jak i zarówno ten dostęp można było uzyskać z poziomu PowerBI i skryptów w języku python. Jako studenci mając dostęp do zasobów platformy Azure gdzie jedynym rodzajem bazy jaka można utworzyć jest baza danych SQL Server to właśnie na nią padł wybór. Ponadto PowerBI obsługuje łączność z tym rodzajem bazy, oraz istnieją odpowiednie sterowniki umożliwiające łączność poprzez skrypty języka python.

4.3.2. Diagram bazy danych



4.3.3. Tabele

- Manufacturers – jest to tabela łącząca wszystkie pozostałe, przechowująca informacje o producentach. W kolumnach przechowywane są informacje potrzebne do rozróżnienia w skryptach i ich api, dane której firmy powinny być pobierane. Stąd w kolumnach przechowywane są nazwy kont firm na twitterze, jak nazywają się podstrony na platformie companiesmarketcamp, oraz nazwy giełdowe firm co jest wykorzystywane w api od yahoo finance.
- CompaniesMarketCapData – tabela przechowuje dane scrapowane z serwisu o tej samej nazwie. W kolumnach przechowywane sa informacje takie jak: Data pobrania danych, miejsca w rankingu firmy, jej kapitał, cena akcji, zmiana ceny akcji w stosunku do dnia poprzednego i ostatniego roku.
- Yfinance – tabela przechowuje dane pozyskiwane z serwisu yahoo finance. Kolumny zawierają dane takie jak date pobrania danych, ceny otwarcia, zamknięcia dla konkretnego dnia, cenę najwyższą, najniższą, liczbę wolumenów.
- Tweets – tabela przechowuje dane pozyskiwane z Twittera. Kolumny zawierają takie informacje jak, treść tweeta, datę, subiektywność, polaryzację tweeta, oraz wyznaczony semant.

4.4. Opracowane skrypty do gromadzenia danych

4.4.1. Użycie języka python

Do napisania naszych skryptów wykorzystaliśmy język programowania python. Jest on bardzo wygodnym językiem do pracy z danymi, wspieranym przez niezliczone liczby bibliotek dedykowanych właśnie przetwarzaniu danych.

4.4.2. Scrapowanie HTML z serwisu companiesmarketcap.com

Do pobierania danych z serwisu companiesmarketcap.com został napisany skrypt, który opiera się na scrapowaniu html'a. Wykorzystywana jest do tego dedykowana pobieraniu elementów stron pythonowa biblioteka o nazwie BeautifulSoup

```
import requests as req
from bs4 import BeautifulSoup
import re
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import pyodbc

server = 'ipzazuresql.database.windows.net'
database = 'ipz2db'
username = 'osuch'
password = 'IPZ_2022_sem_letni'
conn = pyodbc.connect(
'DRIVER={ODBC Driver 17 for SQL Server};SERVER=' + server + ';DATABASE=' + database + ';UID=' + username + ';PWD=' + password)
cursor = conn.cursor()

car_makers = pd.read_sql('SELECT * FROM Manufacturers', conn)

# wyznaczenie akltualnej daty
today = datetime.now().date()

# funkcja usuwajaca tagi html
def striphtml(data):
    p = re.compile(r'<.*?>')
    return p.sub('', data)
```

Na początku skrypt łączy się z bazą danych aby pobrać z niej informacje o przechowywanych w niej producentach samochodów. Wyznaczana też jest od razu obecna data, która będzie stanowiła element rekordu dodawanego do bazy danych. Zadeklarowana została także funkcja która będzie czyściła pobrane elementy drzewa DOM ze znaczników html za pomocą wyrażeń regularnych.

```
marketcap = list()
share_price = list()
change_day = list()
change_year = list()

lists = [rank, marketcap, share_price, change_day, change_year]
for i in range(len(car_makers)):
    # tworze link do podstrony o danym automakerze na serwisie companiemarketcap.com
    # nazwy odpowiednich podstron znajdują się w kolumnie "companiesmarketcap"
    link = "https://companiesmarketcap.com/" + str(car_makers["companiesmarketcap"][i]) + "/marketcap/"
    webpage = req.get(link)
    soup = BeautifulSoup(webpage.content, "html.parser")  # "lxml", "html5lib"
    # pobieram wiersze z tej strony gdzie sa dane: marketcap, share price, Changen, rank
    values = soup.find_all(attrs={'class': 'line1'})
    headers = soup.find_all(attrs={'class': 'line2'})
    # wyswietlam te dane (domyslnie dodawane bedzie do dataframe)
    # print(car_makers["manufacturer"][i])
    ids.append(i + 1)
    date.append(today)

    del values[2]
    del values[5]
    del headers[2]
    del headers[5]

    for j in range(len(values)):
        val = striphtml(str(values[i]))
        # head = striphtml(str(headers[i]))
        # print(head, ":\t", val)
        lists[i].append(val)
```

Następnie deklarowane są listy, które będą odpowiadać kolumną tabeli w bazie danych. Następnie w pętli dla każdego producenta jest tworzony link (na podstawie danych pobranych na początku skryptu z odpowiedniej tabeli) do podstrony w serwisie companiesmarketcap zawierającej informacje o konkretnym producencie. Następnie wykorzystując obiekt biblioteki BeautifulSoup pobierane sa odpowiednie elementy z podstron. Później usuwane są niepotrzebne fragmenty, a pozostałe sa czyszczone z tagów html i pakowane do list.

```
# tworzenie dataframu z list
data = {'manu_id': ids, 'date': date, 'rank_': rank, 'marketcap': marketcap, 'share_price': share_price,
    'change_day': change_day, 'change_year': change_year}
df = pd.DataFrame(data)

# formatowanie kolumn
df['rank_'] = df['rank_'].apply(lambda val: val[1:])
df['marketcap'] = df['marketcap'].apply(lambda val: val[1:])
df['share_price'] = df['share_price'].apply(lambda val: val[1:])
df['change_day'] = df['change_day'].apply(lambda val: val[:-1])
df['change_year'] = df['change_year'].apply(lambda val: val[:-1])
df.loc[df['change_year'] == "N/", "change_year"] = ''

for index, row in df.iterrows():
    print(row)
    try:
        marketcap = row.marketcap
        if marketcap[-1] == 'B':
            marketcap = float(marketcap[0:-2]) * 10 ** 9
        elif marketcap[-1] == 'T':
            marketcap = float(marketcap[0:-2]) * 10 ** 12

        cursor.execute(
            "INSERT INTO CompaniesMarketCapData (manu_id, date_, rank_, marketcap, share_price, change_day, change_year) values(?,?,
            row.manu_id, row.date, row.rank_, marketcap, row.share_price, row.change_day, row.change_year)
    except ValueError:
        print(row)
        continue
    conn.commit()
cursor.close()
```

Później z list tworzony jest DataFrame będący formą tabeli danych, jest to funkcjonalność biblioteki pandas. Kolumny są końcowo odpowiednio formatowane, pozbywamy się prostymi operacjami niechcianych treści, ostatecznie w pętli rekordy są dodawane do bazy danych.

4.4.3. Pobieranie danch z Yahoo Finance

W tym skrypcie wykorzystywane jest api o nazwie yfinance.

```
import yfinance as yf
import pandas as pd
import numpy as np
from datetime import datetime, timedelta
import pyodbc

day_number = datetime.today().weekday()

if day_number == 6 or day_number == 0: # w te dni giełda jest zamknięta
    exit(0)

# Połączenie z bazą danych
server = 'ipzazuresql.database.windows.net'
database = 'ipz2db'
username = 'osuch'
password = 'IPZ_2022_sem_letni'
conn = pyodbc.connect(
'DRIVER={ODBC Driver 17 for SQL Server};SERVER=' + server + ';DATABASE=' + database + ';UID=' + username + ';PWD=' + password)
cursor = conn.cursor()

# Pobranie danych o producentach
car_makers = pd.read_sql('SELECT * FROM Manufacturers', conn)
#print(car_makers)
```

Początkowo identycznie jak poprzednio następuje łączność z baza danych i pobranie danych o producentach oraz wyznaczenie aktualnej daty.

```python
# pobieranie danych przez API
for i, sm_name in enumerate(car_makers["sm_name"]):
    #print(i + 1)
    sm_data = yf.download(sm_name,
                          period='1d',  # z obecnego dnia
                          progress=False,
                          interval='1d')  # sumarczyne dla całego dnia
    # dodawanie kolumny z id producenta
    id = np.ones((len(sm_data), 1), dtype=int) * (i + 1)
    sm_data.insert(0, "manu_id", id)
    # print(sm_data)
    if i == 0:
        data = sm_data
    else:
        data = data.append(sm_data)

# print(data)

for index, row in data.iterrows():
    print(row)
    try:
        date = str(index)[0:10]
        cursor.execute(
            "INSERT INTO Yfinance (manu_id, date_, open_, high_, low_, close_, adj_close, volume) values(?,?,?,?,?,?,?,?)",
            int(row.manu_id), date, row.Open, row.High, row.Low, row.Close, row['Adj Close'], int(row.Volume))
        i += 1
    except ValueError:
        print(row)
        continue
    conn.commit()
cursor.close()
```

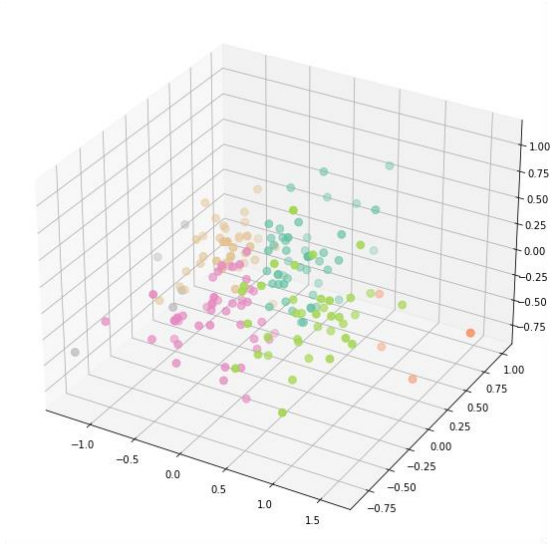
Następnie w pętli dla każdego producenta poprzez api pobierane są uśrednione wyniki giełdowe z ostatnich 24 godzin (jako że skrypt odpalany jest cyklicznie o północy, więc są to uśrednione dane dla całego poprzedniego dnia). Do danych dodawane jest id producenta oraz wyznaczona wcześniej obecna data. Tak utworzone rekordy trafia później ponownie w pętli do bazy danych.

4.4.4. Tweety

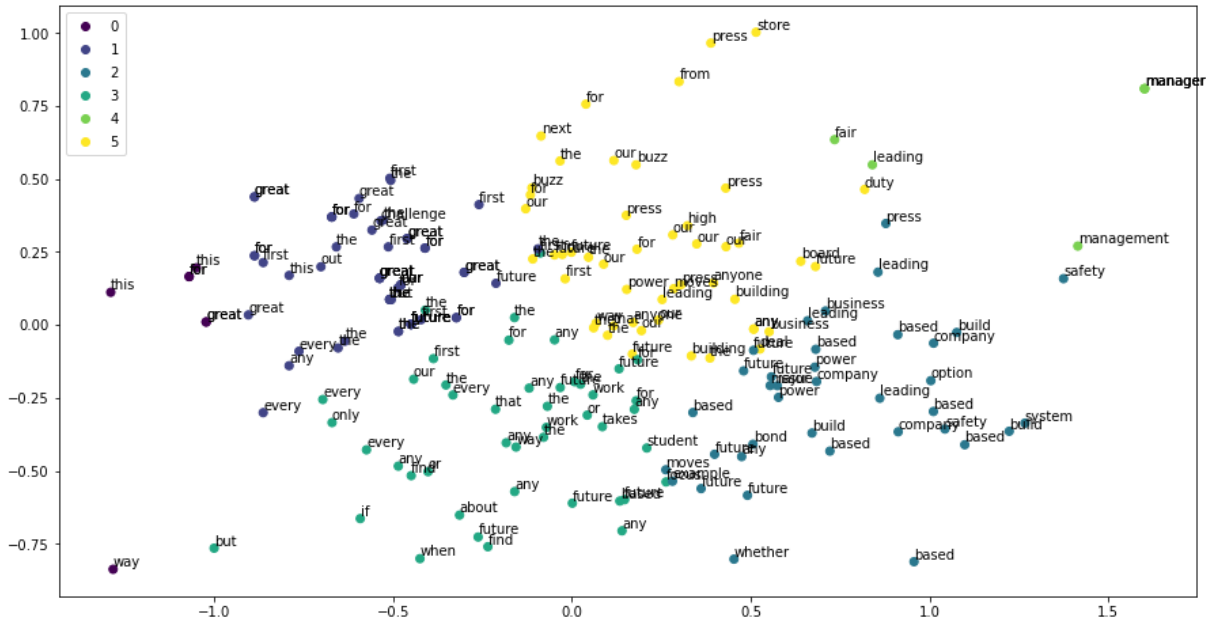
Działanie skryptu polega na kolejno: pobraniu tweetów z ich metadanymi z konta użytkownika twittera, przetwarzaniu tekstu tweetów, analizie semantycznej, wyznaczeniu wektorów dla tweetów przy pomocy techniki „word embedding”, klasteryzacji wektorów i usunięciu klastra najbardziej odbiegającego od wszystkich. Tweety są pobierane z konta użytkownika przy pomocy biblioteki tweepy i danych autoryzacyjnych do postaci dataframe z biblioteki pandas. W tweetach pozostają tylko słowa zaczynające i kończące się literami, usuwane są linki, słowa przestankowe, hashtagi i nazwy kont użytkowników. W celu dokonania analizy semantycznej, tekst jest poddawany tokenizacji na tokeny. Celem otrzymania bardziej wartościowych wyników tokeny są poddawane lemmatyzacji, a nie stemmingowi w procesie przypisywania im części mowy, do których należą(„POS tagging”) w postaci już otagowanej poddawane są lemmatyzacji. Powstała lemma, czyli rdzenie słów są podawane na wejścia funkcji z biblioteki TextBlob, która zwraca pozytywność i subiektywność lemmy tweetów. Każde słowo w lemmie dla danego tweeta, ma wyznaczany wektor słowa, przy pomocy gotowego modelu w postaci kluczy wektorów słów z biblioteki gensim – budowa własnego modelu na tego rodzaju danych, podobnych i w małej ilości dawała wyniki słabej jakości. Z racji na to, że lemma tweetów składa się z małej ilości słów, wektor tweeta jest obliczany jako średnia wektorów słów go tworzących, a nie z wykorzystaniem wag opartych o częstości wystąpień słów jak w technice

TF-IDF, popularnej dla dłuższych dokumentów. Otrzymane wektory dla tweetów były poddane klasteryzacji z wykorzystaniem nienadzorowanej techniki uczenia maszynowego – algorytmu k-średnich, dla 6 klastrów. Obliczono centroidy dla poszczególnych klastrów i jeden środkowy na podstawie wszystkich, usunięto tweety należące do klastra znajdującego się najdalej z wykorzystaniem metryki odległości cosinusowej. W celu wizualizacji wyników, wektory tweetów zostały poddane technikom redukcji wymiarowości PCA i t-SNE, w przypadku PCA otrzymano najbardziej odległe od siebie klastry co jest widoczne na obrazkach poniżej:

Wizualizacja dla 3 głównych składowych:



Wizualizacja dla 2 głównych składowych otagowana najbliższymi podobnymi słowami.



4.5. Apache Airflow

4.5.1. Rola

Rolą tej technologii w niniejszym projekcie było budowanie przepływu pracy naszych skryptów i cykliczne ich wyzwalanie. Jako iż oprogramowanie te było postawione na maszynie wirtualnej, mogło one być uruchomione cały czas i nadzorować pracę skryptów, informować o wynikach i ewentualnych niepowodzeniach, oraz wyzwalać skrypty z zadaną częstotliwością. W ten sposób nowe dane z zadanym interwałem same trafiały do bazy danych bez potrzeby sprawowania ciągłego nadzoru nad przebiegiem tego procesu.

4.5.2. Jak to działa?

Pierwszym krokiem było utworzenie maszyny wirtualnej z systemem Linux, gdzie cały system airflow mógłby funkcjonować. Kiedy maszyna już działała, a środowisko airflow było już zainstalowane, wystarczyło w odpowiedni sposób utworzyć DAGi, czyli skrypty pythonowe, które wykorzystują specjalnie zdefiniowane operatory do zarządzania przepływem pracy. Jeden DAG to jeden potok/przepływ. DAG może wyglądać następująco:

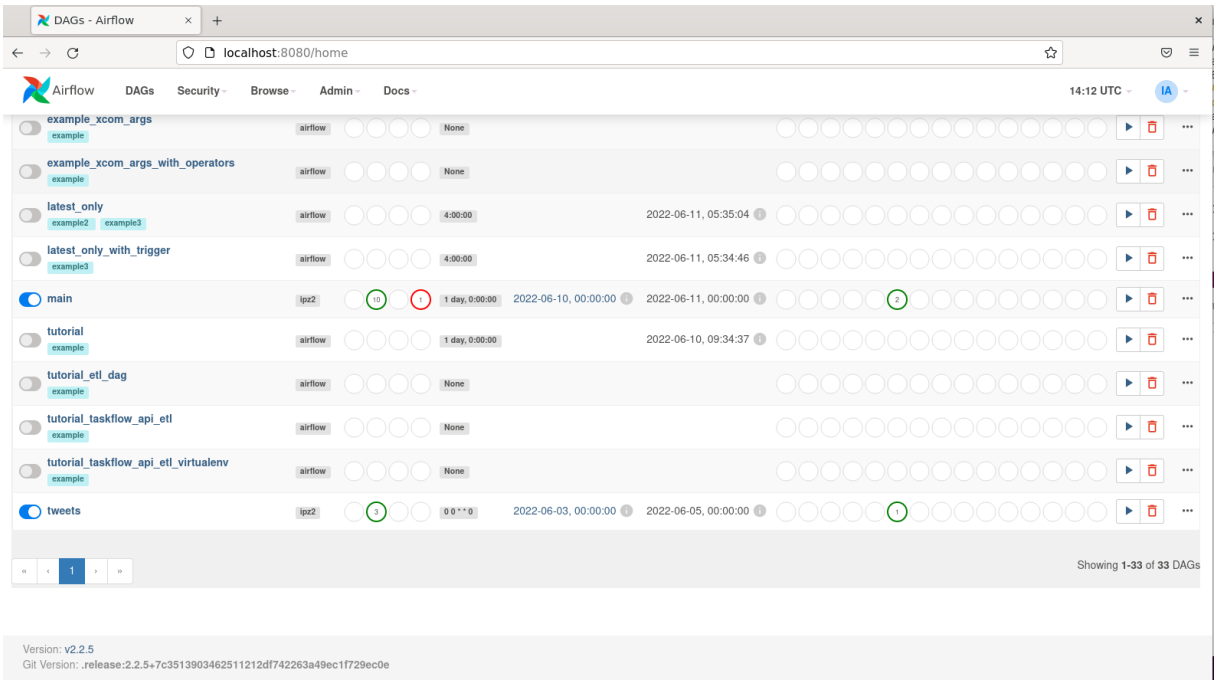
```
default_args = {
    'owner': 'ipz2',
    'depends_on_past': False,
    'start_date': datetime(2022, 6, 2),
    'email': ['fk46509@zut.edu.pl'],
    'email_on_failure': False,
    'email_on_retry': False,
    'retries': 1,
    'retry_delay': timedelta(minutes=2),
    'schedule_interval': "@daily",
}

dag = DAG(
    'main',
    catchup=False,
    default_args=default_args,
    description="glowne drzewo",
)

scraping_html_data = PythonOperator(
    task_id='task1',
    python_callable=companiesmarketcap_scraping,
    dag=dag
)

yahoo = PythonOperator(
task_id='task2',
    python_callable=finance_update,
    dag=dag
)

scraping_html_data >> yahoo
```

DAG posiada swoją nazwę, opis oraz liczne argumenty. Najważniejsze z nich to data rozpoczęcia mówiąca kiedy przepływ ma zostać uruchomiony, kto jest właścicielem DAGa, co jaką jednostkę czasu przepływ ma być uruchamiany (tutaj jest to określane za pomocą stringu „@daily”) oraz ilość prób ponownego uruchomienia w przypadku pojawienia się błędu. Dla jednego DAGa tworzone są uruchamiane w ramach jego zadania. Jako ze nasze skrypty są w języku python używane są do ich określenia operatory PythonOperator. Obiekt ten przyjmuje unikatowe id zadania, funkcje która ma być wywołana (sa ta opakowane w funkcje konkretne skrypty opisane i przedstawione w poprzedniej sekcji), oraz informację do

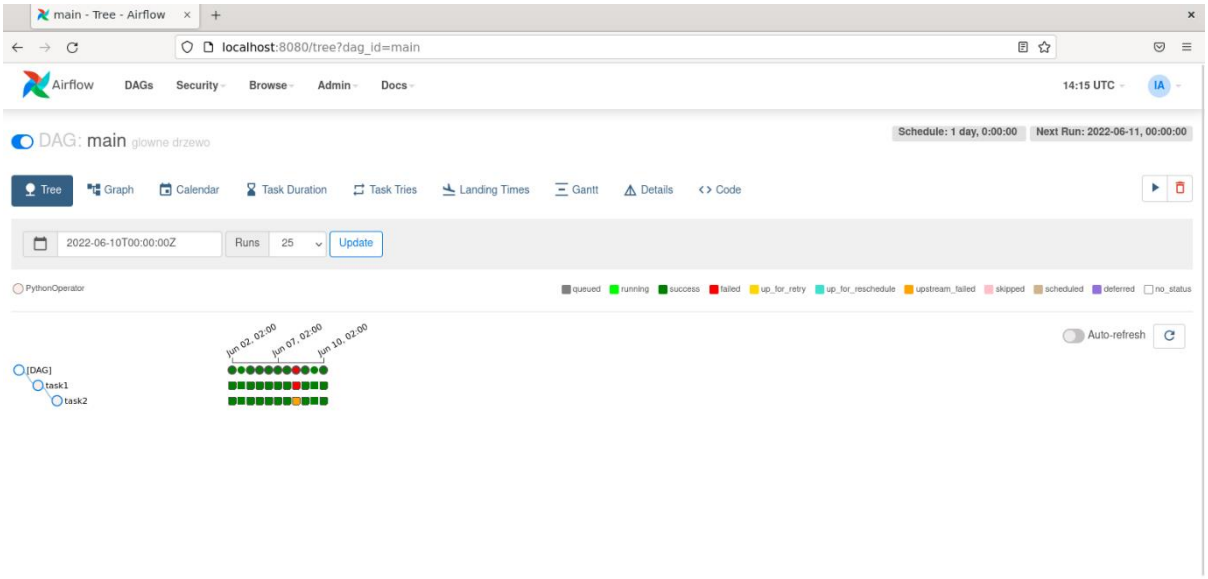
jakiego DAGa utworzone zadanie ma zostać przypisane. Na załączonym przykładzie, mamy dwa takie zadania, jednym jest pobieranie danych z serwisu comaniesmarketcap, drugim z serwisu yahoo finance. Na koniec widzimy zdefiniowany przepływ. Operator >> świadczy o tym, ze najpierw uruchamiame ma być przez airflow zadanie z lewej strony operatora, a następnie te po prawej stronie.

Nasz system posiada tylko dwa takie DAGi, jeden opisany wyżej, uruchamiany raz dziennie dla danych z giełdy. Drugi osobny, uruchamiany raz na tydzień dla tweetów (ze względu ze danych tutaj jest o wiele mniej, nowe tweety pojawiają się rzadko i nie ma sensu uruchamiać skryptu z duża częstotliwością).

Aby airflow mógł wykonywać swoją pracę, należy uruchomić cały zasób, a następnie odpowiednimi komendami w terminalu ustanowić serwis webowy na lokalnym adresie do zarządzania dagami, oraz uruchomić planistę, nadzorującego kiedy, w jakiej kolejności przepływy mają być wyzwalane. Po wykonaniu powyższych czynności możemy zarządzać systemem airflow z przeglądarki a interfejs prezentuje się tak:



Widzimy tutaj wszystkie dagi (wraz z tymi stanowiącymi elementy tutorialu), możemy je uruchamiać, wyłączać, wyzwalać, zarządzać i wiele innych. Po wejściu w konkretnego DAGa możemy zobaczyć następujący ekran:



Tutaj możemy zobaczyć historię przepływów, czy zadania konczyly się powodzeniem, kiedy cos poszlo nie tak, możemy zobaczyć graf, kod dagów, logi inne. Jak widać jest to obszerne narzędzie dostarczające masę funkcjonalności. W naszym projekcie wykorzystujemy wręcz tylko namiastkę jego potencjału, właściwie do realizacji tak prostych zadań jak cykliczne wyzwalanie skryptów mogliśmy wybrać prostsze i lżejsze narzędzie.

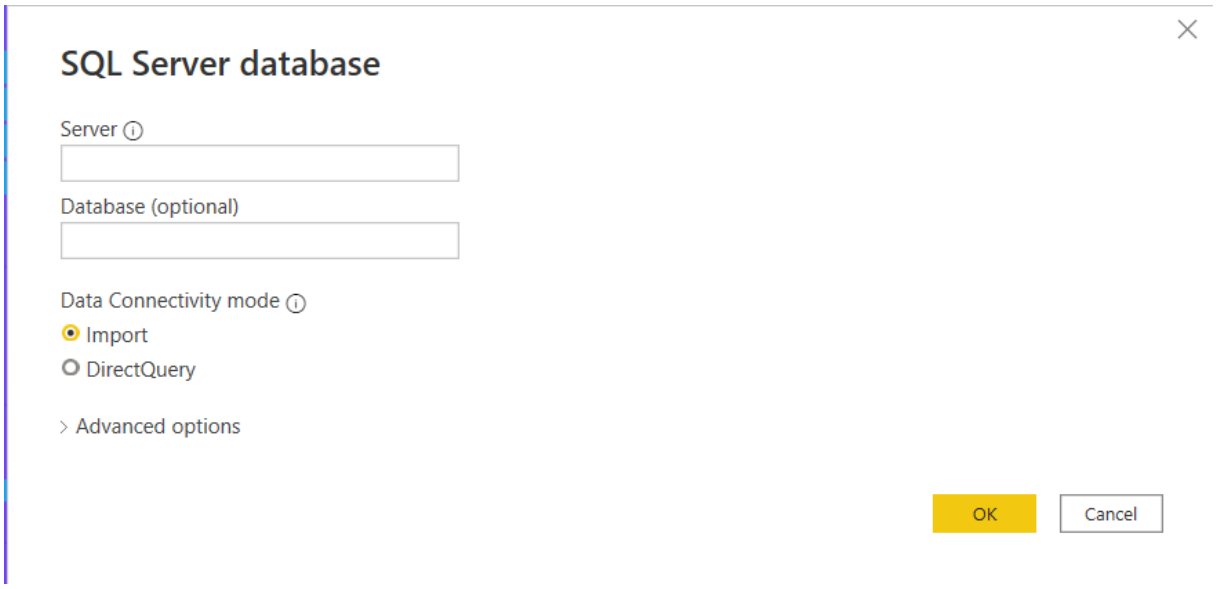
4.6. PowerBI

4.6.1. Dlaczego PowerBi

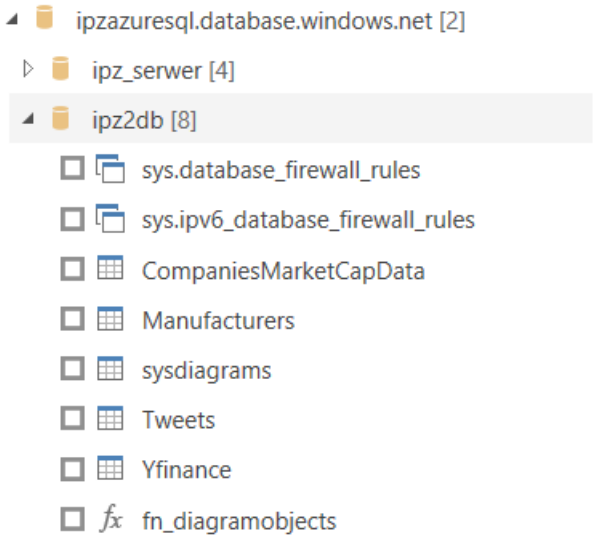
Zdecydowaliśmy się na wybór Power BI jako narzędzia do wizualizacji naszych danych ze względu na duże możliwości tego środowiska oraz jego znajomość, co zdecydowanie wpłynęło korzystnie na szybkość pracy oraz jej efektywność.

4.6.2. Połączenie PowerBi do bazy danych

Do połączenia się z bazą danych z poziomu Power BI została wykorzystana wbudowana opcja.

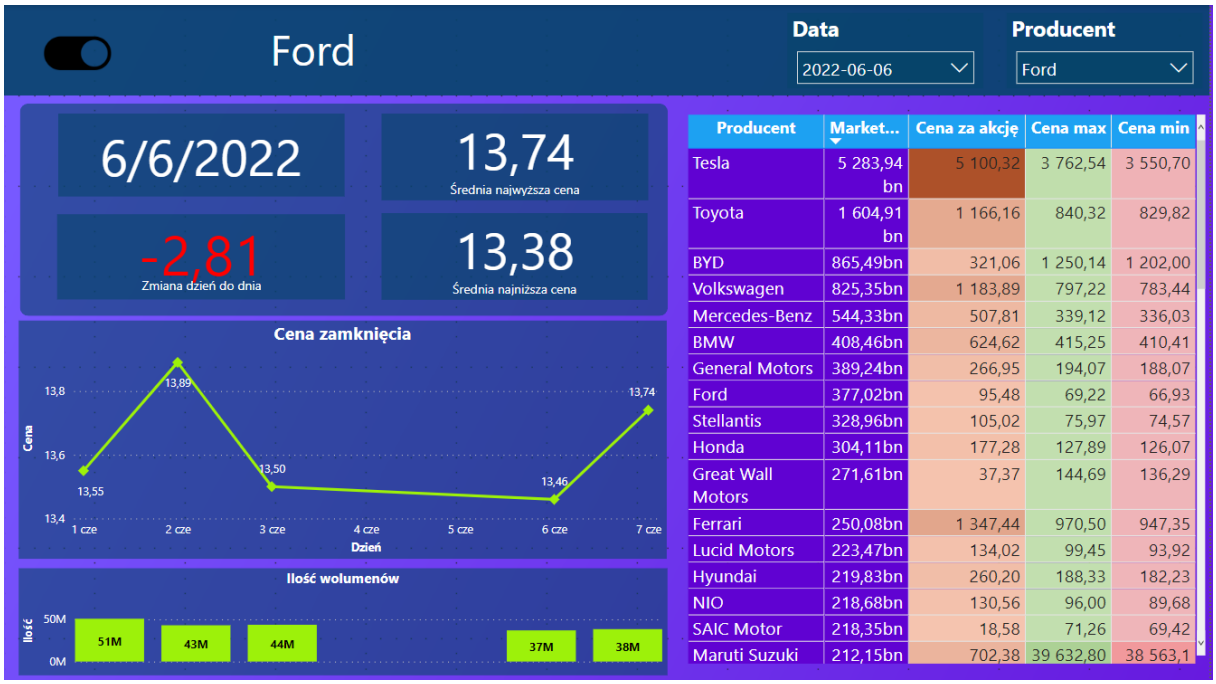


Po podaniu adresu bazy oraz loginu i hasła wybraliśmy tabele, które zostały przez nas stworzone w celu zaimportowania ich do programu.



4.6.3. Dashboard z danymi giełdowymi

W prawym górnym rogu mamy do wyboru datę oraz producenta. Zmiana tych parametrów wpływa na elementy, które umieszczone są po lewej stronie strony. Wygląd tabeli oraz dane prezentowane w niej nie ulegają zmianie, są stałe.



Producent	Market...	Cena za akcję	Cena max	Cena min
Tesla	5 283,94 bn	5 100,32	3 762,54	3 550,70
Toyota	1 604,91 bn	1 166,16	840,32	829,82
BYD	865,49bn	321,06	1 250,14	1 202,00
Volkswagen	825,35bn	1 183,89	797,22	783,44
Mercedes-Benz	544,33bn	507,81	339,12	336,03
BMW	408,46bn	624,62	415,25	410,41
General Motors	389,24bn	266,95	194,07	188,07
Ford	377,02bn	95,48	69,22	66,93
Stellantis	328,96bn	105,02	75,97	74,57
Honda	304,11bn	177,28	127,89	126,07
Great Wall Motors	271,61bn	37,37	144,69	136,29
Ferrari	250,08bn	1 347,44	970,50	947,35
Lucid Motors	223,47bn	134,02	99,45	93,92
Hyundai	219,83bn	260,20	188,33	182,23
NIO	218,68bn	130,56	96,00	89,68
SAIC Motor	218,35bn	18,58	71,26	69,42
Maruti Suzuki	212,15bn	702,38	39 632,80	38 563,1

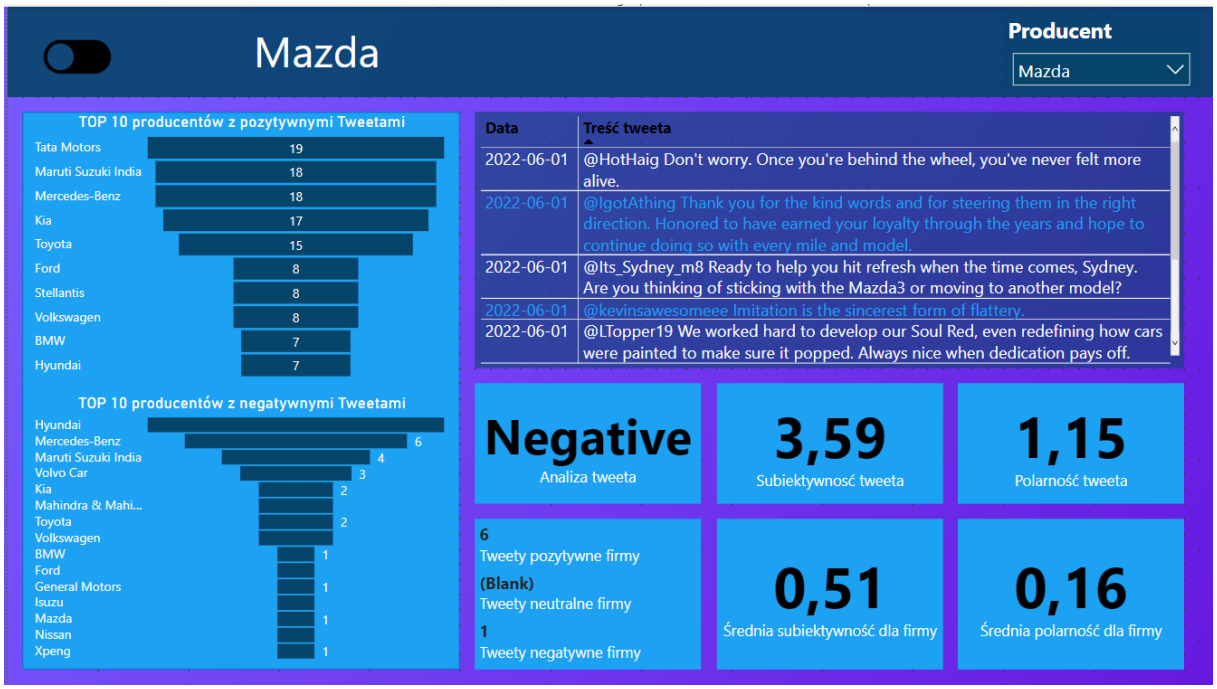
W tym miejscu możemy podejrzeć maksymalną oraz minimalną średnią cenę za akcję dla wybranego przez nas producenta. Jest też pokazana zmiana cen w stosunku dzień wybrany do dnia poprzedniego (jeśli dzień poprzedni nie wypadał w trakcie weekendu).

Niżej mamy dwa wykresy, które prezentują cenę zamknięcia akcji dla danej firmy oraz ilość wolumenów, które były w obiegu w danym dniu.

W tabeli mamy dane które przedstawiają nam nazwę producenta oraz jego całkowitą wartość rynkową, cenę za jedną akcję oraz maksymalną i minimalną cenę.

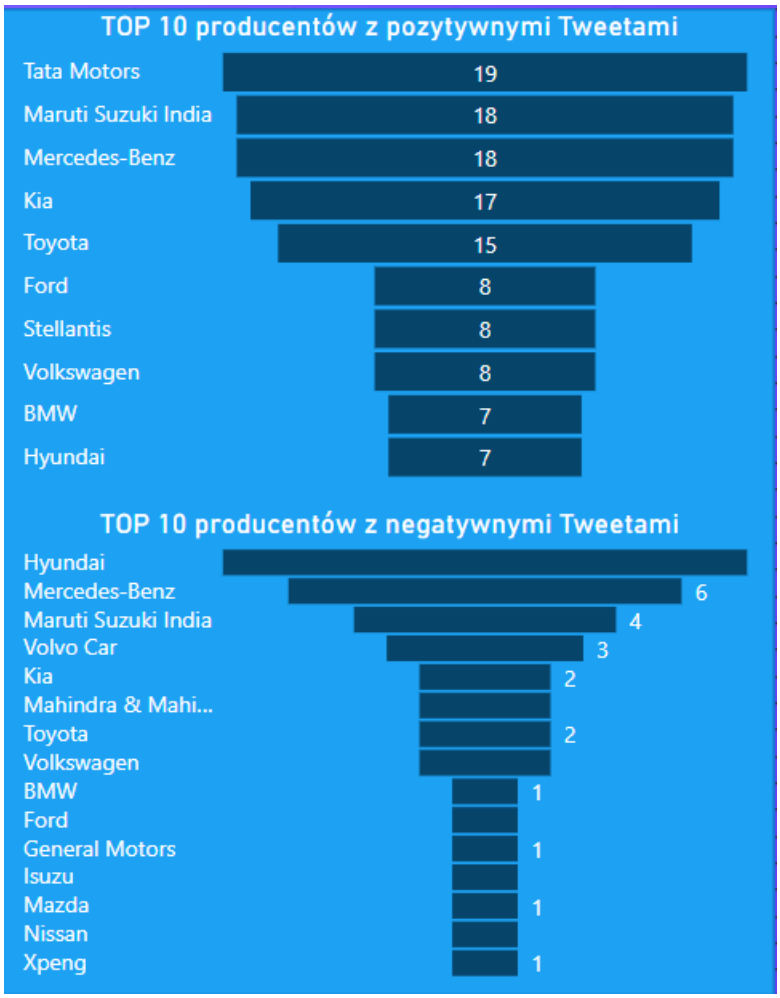
Producent	Market...	Cena za akcję	Cena max	Cena min
Tesla	5 283,94 bn	5 100,32	3 762,54	3 550,70
Toyota	1 604,91 bn	1 166,16	840,32	829,82
BYD	865,49bn	321,06	1 250,14	1 202,00
Volkswagen	825,35bn	1 183,89	797,22	783,44
Mercedes-Benz	544,33bn	507,81	339,12	336,03
BMW	408,46bn	624,62	415,25	410,41
General Motors	389,24bn	266,95	194,07	188,07
Ford	377,02bn	95,48	69,22	66,93
Stellantis	328,96bn	105,02	75,97	74,57
Honda	304,11bn	177,28	127,89	126,07
Great Wall Motors	271,61bn	37,37	144,69	136,29
Ferrari	250,08bn	1 347,44	970,50	947,35
Lucid Motors	223,47bn	134,02	99,45	93,92
Hyundai	219,83bn	260,20	188,33	182,23
NIO	218,68bn	130,56	96,00	89,68
SAIC Motor	218,35bn	18,58	71,26	69,42
Maruti Suzuki	212,15bn	702,38	39 632,80	38 563,1

Używając przełącznika zlokalizowanego w lewym górnym rogu możemy przełączyć się między stronami z danymi giełdowymi, a danymi o tweetach. Na tej stronie możemy wybrać producenta, aby zobaczyć tweety z jego oficjalnego konta. W tabeli mamy podaną datę oraz treść samego tweeta. Po wybraniu konkretnego tweeta zobaczymy jego analizę, wartość subiektywną oraz polarność.



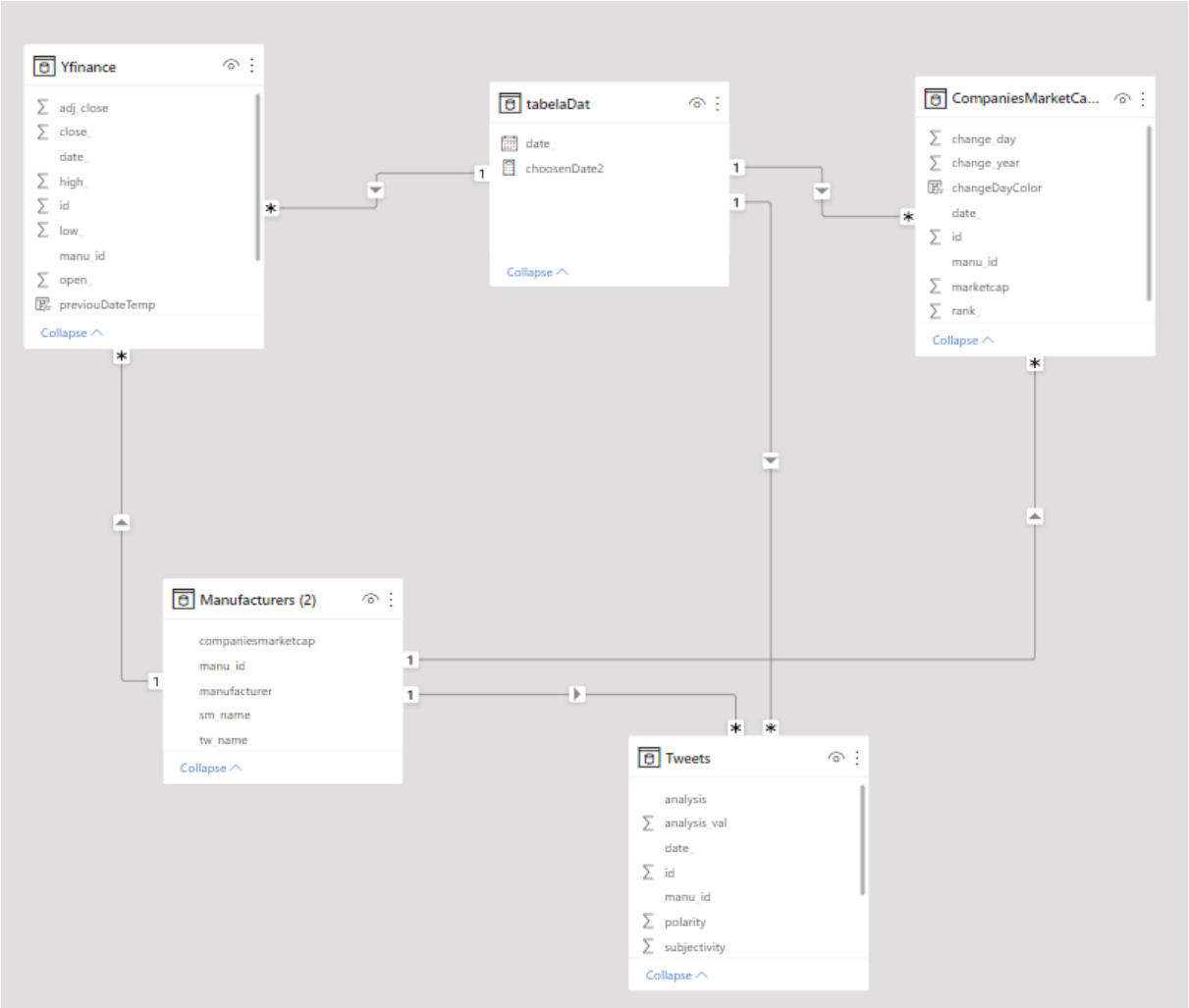
Niżej mamy zestawienie z ilościami pozytywnych, negatywnych oraz neutralnych tweetów dla danego producenta oraz wyliczaną średnią subiektywność oraz polarność dla jego wszystkich tweetów.

Po lewej stronie mamy zestawienie, którzy producenci mają najwięcej pozytywnych i negatywnych tweetów.



4.6.4. Model danych w Power BI

W naszym modelu wszystkie tabele z bazy danych połączone są po polu daty, jest to nasz główny klucz. Dodatkowo, utworzyliśmy specjalną tabelę dat, która ujednolica nasze wszystkie daty. Ułatwia też tworzenie wizualizacji.



4.6.5. Język DAX

W raporcie nie używamy skomplikowanych miar napisanych w języku DAX, jednakże kilka z nich się pojawia. Są to na przykład miary zliczające ilości tweetów, już wcześniej wspomniana tabela dat czy po prostu miara, która pokazuje wybraną datę.

```
tabelaDat = SUMMARIZE(Yfinance, Yfinance[date_])
```

```
countAnalysisNegative = CALCULATE(COUNT(Tweets[analysis]), Tweets[analysis] = "Negative")
```

```
choosenDate2 = LEFT(SELECTEDVALUE(tabelaDat[date_]), 14)
```

5. Podsumowanie

Jako zespołowi udało nam się utworzyć produkt, który spełnia nałożone sobie założenia. W między czasie udało nam się poznać nowe technologie, poszerzyć swoją wiedze w zakresie

pracy z danymi i ich przetwarzania, nabyć nowe umiejętności, dowiedzieć więcej o samej giełdzie. Temat projektu bardzo nam się spodobał i zachęcamy do korzystania z naszej wizualizacji.

Link do repozytorium, gdzie znajdują się opisane skrypty, oraz plik z wizualizacjami który może zostać otwarty w środowisku PowerBI: https://github.com/zm46704/IPZ-2021-22