Zachodniopomorski Uniwersytet
Technologiczny w Szczecinie

Praca magisterska

Analiza danych giełdowych przy pomocy narzędzi dostępnych w pakiecie scikit-learn

Łukasz Połoń

Kierunek: Informatyka
Specjalność: Inżynieria oprogramowania

Nr albumu: 24942

Promotor
dr inż. Piotr Błaszyński

Wydział Informatyczny

Szczecin 2018

Oświadczenie autora

Ja, niżej podpisany Łukasz Połoń oświadczam, że praca ta została napisana samodzielnie i wykorzystywała (poza zdobytą na studiach wiedzą) jedynie wyniki prac zamieszczonych w spisie literatury.

.....
(Podpis autora)

Oświadczenie promotora

Oświadczam, że praca spełnia wymogi stawiane pracom magisterskim.

.....
(Podpis promotora)

Spis treści

Streszczenie

Przykładowe streszczenie i test polskich znaków: ąśćżźłóęą

Słowa kluczowe

Python, Scikit-learn, Giełda Papierów Wartościowych, Kivy, Matplotlib, Pandas, Analiza regresji

Abstract

Keywords

Python, Scikit-learn, Stock Market, Kivy, Matplotlib, Pandas, Regression analysis

Wprowadzenie

Giełda Papierów Wartościowych to instytucja, która prowadzi działalność w zakresie organizacji obrotu papierami wartościowymi i instrumentami finansowymi[1]. W praktyce spełnia ona rolę pośrednika finansowego pomiędzy kupującym, a sprzedającym papiery wartościowe. Dane generowane przez giełdę poddawane są ciągłym analizom, w szczególności w celu dostarczenia informacji potrzebnych do właściwego zarządzania kapitałem.

Istnieją dwie główne metody analizy danych giełdowych: analiza fundamentalna i analiza techniczna[2]. Pierwsza z nich polega na analizie faktycznej kondycji finansowej podmiotu, podczas gdy druga ma za zadanie prognozowanie przyszłych wartości wskaźników lub cen na podstawie zebranych danych.

Temat tej pracy podejmuje opisanie i przetestowanie wybranych metod analitycznych dostępnych w pakiecie *Scikit-learn*. Pakiet ten jest biblioteką języka programowania Python umożliwiającą wysokopoziomowe przetwarzanie danych. Udostępnia wiele algorytmów klasyfikacji, regresji oraz uczenia maszynowego, które mogą zostać wykorzystane do przeprowadzania obliczeń między innymi na potrzeby analizy technicznej, której to elementy zostaną tutaj przedstawione.

Rozdział 1

Analiza danych statystycznych

1.1. Rynki finansowe

Giełda jest zbiorem instytucji finansowych, w których odbywa się wymiana papierów wartościowych pomiędzy kupującymi i sprzedającymi[3]. Powinna ona koncentrować popyt i podaż na papiery wartościowe, co prowadzi do kształtowania się powszechnego kursu. Zapewnia ona również bezpieczny i uregulowany przebieg transakcji oraz upowszechnia informacje, które umożliwiają ocenę aktualnej wartości papierów wartościowych.

Papier wartościowy, zgodnie z prawem o obrocie papierami wartościowymi, to *"dokument, mający stwierdzać lub stwierdzający istnienie określonego prawa majątkowego utrwalony w takim brzmieniu i w taki sposób, że może stanowić samodzielny przedmiot obrotu publicznego"*[4] Papierami wartościowymi mogą być między innymi: akcje, obligacje, czeki, bony skarbowe. Akcje dają nabywcy (akcjonariuszowi) prawo do współwłasności w spółce, która je wyemitowała, co przekłada się na bezpośredni udział w wypracowanych zyskach (dywidendy), a także nabycie praw korporacyjnych, umożliwiających decyzyjność w spółce[5].

Obligacje są to papiery dłużne, które poświadczają wierzytelność pomiędzy właścicielem, a dłużnikiem. W ich przypadku dłużnikiem jest emitent, który zobowiązuje się do uregulowania wierzytelności w określonym czasie. Obligacje emituje się w celu pozyskania kapitału, a obligatoriusz, czyli nabywca, otrzymuje prawo do całkowitego zwrotu inwestycji po upłynięciu określonego terminu, a także do otrzymywania stałego dochodu określonego odsetkami[6].

1.2. Ekonometria i metody analizy danych finansowych

Podejmowanie decyzji inwestycyjnych na giełdzie papierów wartościowych nieuchronnie wiąże się z ryzykiem straty zainwestowanego kapitału. Zmniejszenie tego ryzyka jest więc kluczowym działaniem inwestorów, którzy oczekują wymiernych zwrotów z prowadzonych inwestycji. Ekonometria jako nauka zajmuje się dostarczeniem metod i narzędzi potrzebnych do przeprowadzenia niezbędnych analiz rynku,

dzięki którym potencjalny inwestor może ocenić ryzyko inwestycyjne oraz je zminimalizować. Rozwój gospodarki i rynków światowych wymusza niejako wzrost zapotrzebowania na coraz to bardziej zaawansowane i dokładne narzędzia, które wspomagają podejmowanie decyzji. Na decyzje z kolei mają wpływ czynniki, które można podzielić na jakościowe i ilościowe[7].

Czynniki jakościowe, ze względu na swoją naturę i duży wpływ osób dokonujących analizy, nie mogą podlegać bezpośredniej mierzalności, przez co uważane są za subiektywne[7]. Czynniki ilościowe natomiast umożliwiają ocenę na podstawie danych liczbowych, ich wzajemnych powiązań i relacji. Są więc uważane za bardziej obiektywne.

1.2.1. Model ekonometryczny

Ekonometrię finansową można więc dogłębniej zdefiniować jako zastosowanie metod ilościowych do analizy zjawisk na rynku finansowym[7].

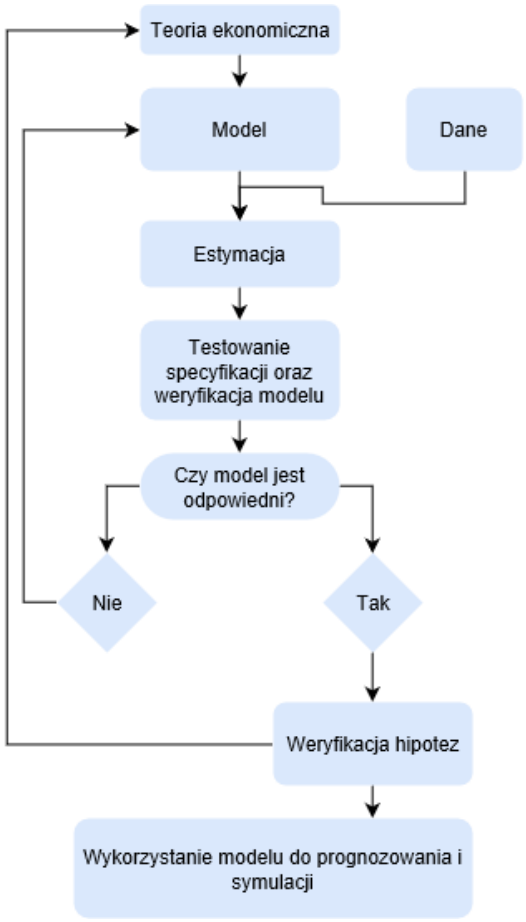
Podstawowym krokiem, który należy postawić podczas analizy ekonometrycznej jest określenie modelu. Model jest uproszczoną reprezentacją rzeczywistego procesu, który podlega naszym badaniom[9]. Powinien więc możliwie dokładnie opisywać badane procesy, z uwzględnieniem właściwych zmiennych. Proste modele z reguły upraszczają rzeczywistość, co w połączeniu z przyjętymi założeniami niekoniecznie będącymi zgodne ze stanem faktycznym pozwalają domniemywać wadliwość modelu. Jednakże dokładność modelu często zależy od dostępności danych. Jedna z koncepcji konstruowania modelu zakłada, że warto rozpoczynać badania od modelu prostego i w miarę poznawania dodatkowych danych, sukcesywnie zwiększać jego złożoność.

Model ekonometryczny składa się z[9]:

- Zbioru równań behawioralnych, czyli opisujących zachowanie
- Opisu możliwych błędów pomiaru lub obserwacji
- Specyfikacji rozkładu prawdopodobieństwa zakłóceń i błędów pomiaru

Model jest częścią algorytmu analizy ekonometrycznej, który został przedstawiony na diagramie 1.1.

Pierwszymi krokami jakie należy wykonać podczas analizy ekonometrycznej są opracowanie teorii, modelu, oraz skompletowanie niezbędnych danych. Następnie model podlega estymacji i weryfikacji, co pozwala na ocenę jego poprawności. Kolejne iteracje tego algorytmu prowadzą do opracowania dokładnego modelu, który w jak najlepszym stopniu opisuje zjawisko, które jest aktualnie badane.



Rysunek 1.1. Schemat kroków w ekonometrycznej analizie modeli ekonomicznych

1.2.2. Analiza techniczna

Analiza techniczna skupia się głównie na bezpośrednich, mierzalnych sposobach oceny aktualnej kondycji rynku. Podejmuje analizę jego aktywności, w tym cen, ilości transakcji w danym okresie czasu, poprzez badanie wzajemnych zależności oraz wzorców.

Oparta jest na trzech zasadach:[8]

- Wszystkie czynniki, które wpływają na rynkową cenę instrumentu finansowego znajdują swoje odzwierciedlenie w cenie tego instrumentu
- Ceny zawsze podlegają określonym trendom
- Historia się powtarza - założenie zakładające, że ceny na rynku zmieniają się i powtarzają zgodnie z określonymi wzorami, które wydarzyły się w przeszłości

Klasyczna analiza techniczna opiera się na teorii Dowa, która jest pierwszą teorią stanowiącą fundament dla analizy tego typu. Założenia teorii Dowa to[10]:

- Średnie giełdowe dyskontują wszystko
- Istnieją trzy kategorie trendu rynkowego: trendy główne, wtórne i mniejsze
- Wolumen potwierdza trend

Główny trend rynkowy jest trendem o największym znaczeniu, trwającym zwykle od roku do kilkunastu lat. Trendy mniejsze, trwające do trzech tygodni, są marginalizowane i utożsamiane z czynnikami losowymi, nie mającymi realnego wpływu na trend główny. Trendy wtórne natomiast korygują trendy główne i trwają od kilku tygodni do trzech miesięcy. Są one szczególnie ważne i utożsamiane z korektą techniczną trendu głównego.

Trend główny dzieli się na trzy fazy. Pierwsza faza, zwana fazą akumulacji[10] oznacza, iż podczas hossy inwestorzy skupują akcje. Faza druga charakteryzuje się przetrzymaniem akcji i niepodejmowaniem działań przez inwestorów, w oczekiwaniu na dalsze informacje. Trzecia faza natomiast związana jest ze sprzedażą akcji, a także z większym zainteresowaniem inwestorów indywidualnych, którzy w tym momencie zaczynają skupować akcje.

W przypadku bessy, która również dzieli się na trzy fazy, faza pierwsza oznacza wyprzedaż akcji przez wtajemniczonych inwestorów, faza druga jest nasileniem tej tendencji, a faza trzecia charakteryzuje się stagnacją.

Z opisu przebiegów trendu głównego podczas hossy i bessy wynika, że związany jest on bezpośrednio z wartościami wolumenu obrotów. Wolumen jest to łączna liczba transakcji przeprowadzonych dla danego papieru wartościowego. Tak więc, jeśli wraz ze wzrostem ceny, wzrasta wolumen obrotów, można potwierdzić występowanie

hossy[10].

Narzędzia analizy technicznej, ponieważ bazują między innymi na teorii Dowa, są najczęściej wykorzystywane do analizy trendów, ich identyfikacji oraz wychwytywania oznak ich odwrócenia.

1.2.3. Analiza fundamentalna

Analiza fundamentalna, w odróżnieniu od analizy technicznej, ma na celu opis danej spółki oraz jej otoczenia. Czynniki brane pod uwagę w tego typu analizie to najczęściej sytuacja gospodarcza kraju, czyli między innymi poziom PKB, inflacja, czy wielkość rynku[10], oraz sytuacja gospodarcza samego przedsiębiorstwa.

Jednym z najstarszych lecz wciąż skutecznych sposobów mierzenia pozycji danego przedsiębiorstwa na rynku jest macierz BCG (Boston Consulting Group). Pozwala ona na przypisanie przedsiębiorstwa do jednej z czterech kategorii:
- Gwiazdy
- Obiecujące
- Dojne krowy
- Psy

Kategoria pierwsza, czyli gwiazdy, to przedsiębiorstwa z największym udziałem sprzedaży w odniesieniu do całego rynku, a także o najwyższej pozycji konkurencyjnej. Przedsiębiorstwa kategorii drugiej charakteryzują się szybkim wzrostem sprzedaży, lecz równolegle intensywnie inwestujące, przez co są pozbawione zapasów kapitału. Dojne krowy są stosunkowo podobne do przedsiębiorstw kategorii drugiej, lecz w ich przypadku intensywność inwestycyjna jest minimalna. Są zazwyczaj stabilne, a ich udział w rynku pozostaje na stałym poziomie. Kategoria czwarta określa podmioty nieobiecujące, które posiadają niewielki udział w rynku, wraz z brakiem perspektyw do rozwoju.

1.3. Statystyczne metody analizy danych

1.3.1. Klasyfikacja

Klasyfikacja jest jednym z najbardziej podstawowych metod analizy danych statystycznych. Jej głównym zadaniem jest przyporządkowanie klas do obiektów z danego zbioru danych. Zagadnienia uczenia maszynowego dzielą klasyfikację na nadzorowaną i klasyfikację bez nadzoru[13].

Algorytmy klasyfikacji nadzorowanej charakteryzują się dostępnością testowego zbioru danych z przypisanymi klasami, który spełnia rolę wzorca dla pozostałych danych. Algorytm klasyfikacji bez nadzoru pozbawiony jest tego typu informacji, często również brak jest informacji jakie klasy ma tworzyć dany zbiór. Zadaniem takiego algorytmu jest więc wydzielenie i powiązanie ze sobą danych w taki sposób, aby stworzyć klasy i przyporządkować do nich odpowiednie dane.

Istnieje wiele algorytmów klasyfikacji, których zastosowanie praktyczne uzależnione jest od czynników takich jak szybkość działania, zużycie pamięci, łatwość interpre-

Rysunek 1.2. Klasyfikacja nienadzorowana

tacji, oraz oczywiście trafność predykcji[12].
Wśród nich możemy wyróżnić:
- Naiwny klasyfikator Bayes'a
- Drzewa decyzyjne
- Algorytm najbliższego sąsiada
- Maszyna wektorów nośnych (SVM)
- Liniowa analiza dyskryminacyjna

1.3.1.1. Naiwny klasyfikator Bayes'a

W 1763 roku Thomas Bayes przestawił w swojej pracy twierdzenie teorii prawdopodobieństwa, które warunkuje prawdopodobieństwa dwóch zdarzeń warunkujących się na wzajem.
Brzmi ono [14]: *Dla dowolnej hipotezy* $h \in H$ *oraz zbioru danych* D *zachodzi równość*

$$Pr(h \mid D) = \frac{Pr(h)Pr(D \mid h)}{Pr(D)} \tag{1.1}$$

Prawdopodobieństwo *Pr(h)* jest prawdopodobieństwem *a priori* co oznacza, że przy jego określaniu nie były brane pod uwagę dane, które mogły mieć wpływ na jego wartość. Poprzez ich uwzględnienie określone zostaje prawdopodobieństwo *a posteriori* $Pr(h \mid D)$. Wzór Bayesa wyraża zatem związek między tymi dwoma prawdopodobieństwami, natomiast do jego wyrażenia używane są jeszcze prawdopodobieństwo zaobserwowania danych *Pr(D)*, oraz prawdopodobieństwo zaobserwowania tych danych przy założeniu poprawności hipotezy $Pr(D \mid h)$.

W klasyfikatorze Bayes'a wybór decyzja o wyborze odpowiedniej hipotezy h ze zbioru hipotez H nie jest podejmowana na podstawie dokładności czy złożoności, lecz prawdopodobieństwa. Wybór dokonywany jest na podstawie dwóch sposobów, dzięki którym można uznać hipotezę za najlepszą[14]:
- Zasada maksymalnej zgodności
- Zasada maksymalnego prawdopodobieństwa *a posteriori*

Pierwsza z nich jest nazywana zasadą ML (*Maximum Likelihood*) i mówi, że najlepszą jest hipoteza $h_{ML} \in H$, która maksymalizuje warunkowe prawdopodobieństwo danych treningowych

$$h_{ML} = arg \overset{h \in H}{max}\ Pr(T \mid h) \tag{1.2}$$

Kolejna, nazywana zasadą MAP (*Maximum a posteriori*) wyjaśnia, że należy wybrać taką hipotezę $h_{MAP} \in H$, która posiada maksymalne prawdopodobieństwo *a posteriori*:

$$h_{MAP} = arg \overset{h \in H}{max}\ Pr(h \mid T) \tag{1.3}$$

1.3.1.2. Drzewa decyzyjne

Algorytmy drzew decyzyjnych są jednymi z najszerzej stosowanych metod analizy danych za pomocą uczenia maszynowego. Polegają na sekwencyjnym podziale danego zbioru danych na dwa rozłączne podzbiory w taki sposób, aby oba podzbiory były możliwie jednorodne[15]. Działanie drzewa decyzyjnego ilustruje Rysunek 1.3.

Wierzchołki drzewa oznaczone literą t są nazywane węzłami i stanowią podzbiory zbioru danych. Węzły oznaczone okręgami są węzłami wewnętrznymi, natomiast kwadratami - węzłami zewnętrznymi. Dla każdego węzła określona jest funkcja podziału, która każdemu elementowi do niego należącymi przypisuje jedną z wartości : *Prawdę* lub *Fałsz*.
Klasyfikatory zbudowane na podstawie drzewa decyzyjnego mają więc postać[15]:

$$d_T(x) = \sum_{t \in T} ind(t) I(x \in t) \tag{1.4}$$

Zaletą algorytmów drzew decyzyjnych jest niewątpliwie możliwość bezproblemowego wykorzystania zarówno jakościowych jak i ilościowych do klasyfikacji. Algorytmy te wykazują również odporność na sytuacje braku części zmiennych, a także na obserwacje odstające[15].

1.3.2. Analiza regresji

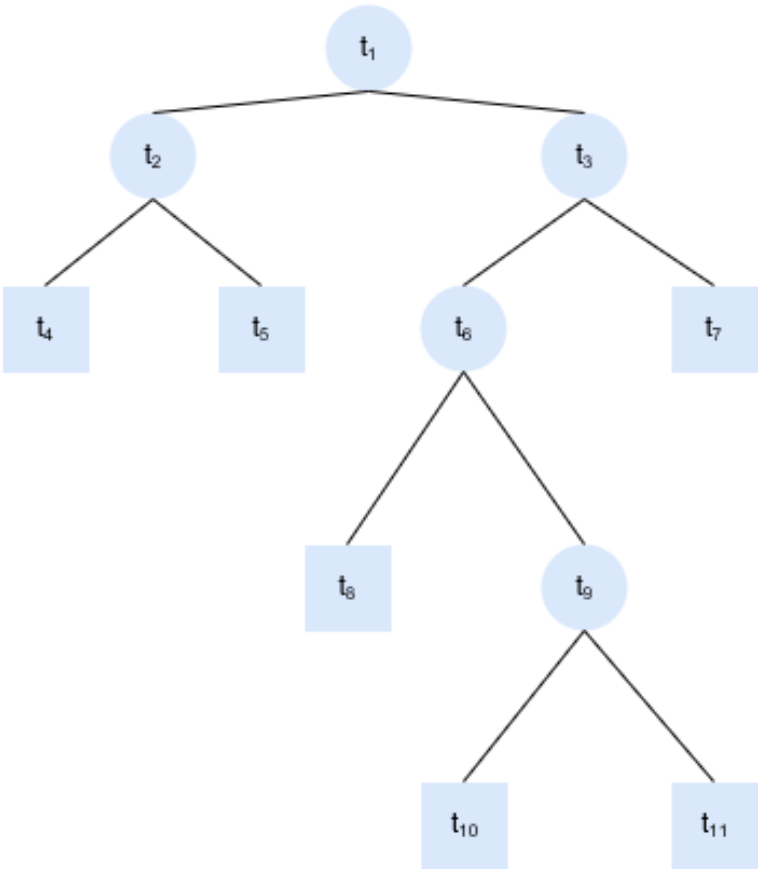
Jedną z metod umożliwiających predykcję wartości na podstawie szeregu innych jest analiza regresji. Polega ona na odszukiwaniu i opisie związków między zmiennymi, co prowadzi do stworzenia modelu, czyli równania regresji, które pozwala na analizę struktury zależności i umożliwia prognozowanie.

Model regresji liniowej jest to funkcja liniowa zmiennych objaśniających i składnika losowego[16]. Ma postać:

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + ... + \beta_k X_k + \xi \tag{1.5}$$

Symbole β_k odpowiadają współczynnikom regresji cząstkowej, które są najważniejszą częścią równania. Informują o zmianie zmiennej Y podczas zmiany zmiennej mu odpowiadającej przy założeniu, że pozostałe zmienne nie uległy zmianie. Wyraz wolny regresji jest wyrażony symbolem β_0 i zazwyczaj nie podlega interpretacji[16]. Składnik losowy oznaczany jest przez ξ, a zmienne Y i X nazywane są odpowiednio zmienną objaśnianą (zależną) i zmiennymi objaśniającymi (niezależnymi).

Zasadniczą rolę podczas analizy regresji odgrywa estymacja parametrów modelu. Ma ona na celu znalezienie wartości ocen parametrów na podstawie danych z



Rysunek 1.3. Drzewo decyzyjne

próby[16]. Tak więc dzięki estymacji można uzyskać takie wartości ocen, które sprawiają iż model regresji jak najlepiej pasuje do danych. W przypadku regresji liniowej z jedną zmienną objaśniającą, gdzie równanie regresji przyjmuje postać prostej na wykresie, uzyskujemy jak najlepsze dopasowanie tej prostej do punktów na wykresie rozrzutu danych.

Do przeprowadzenia estymacji parametrów modelu regresji liniowej najczęściej stosowana jest Metoda Najmniejszych Kwadratów. Pozwala ona na znalezienie ocen parametrów o najmniejszej wartości kwadratów odchyleń pomiędzy rzeczywistymi a teoretycznymi wartościami zmiennej objaśnianej[16]. Oznaczanie wartości dopasowanej $\hat{y}_i$ za pomocą wzoru regresji liniowej jest przewidywaniem tej zmiennej na podstawie modelu. Zmienna ta jest różna od rzeczywistej wartości y_i, a różnica pomiędzy nimi nazywana jest *resztą*, którą można zdefiniować jako[17]:

$$e_i = y_i - x_{1i}b_1 - x_{2i}b_2 - ... - x_{ki}b_k = y_i - \hat{y}_i \tag{1.6}$$

Dopasowanie modelu jest tym lepsze, im mniejsza jest różnica pomiędzy wartościami dopasowanymi i rzeczywistymi. Na wykresach jest to przedstawione jako wartość bezwzględna odległości punktu od prostej regresji.

Rozdział 2

Zastosowanie języka programowania Python w obliczeniach analitycznych

Python jest językiem programowania wysokiego poziomu, charakteryzujący się przede wszystkim wysoką klarownością i czytelnością kodu. Jest to język interpretowany, co w odróżnieniu od języków kompilowanych pozwala na bardzo szybkie tworzenie i testowanie kodu. Wadą tego rozwiązania jest niestety spadek wydajności oraz zwiększone zużycie pamięci i procesora, jednak zastosowania praktyczne Pythona zazwyczaj pozwalają na poniesienie tego typu kosztów.

Python został stworzony w 1989 roku przez Guido van Rossum, a do dzisiaj rozwijany jest jako projekt Open Source i zarządzany przez organizację non-profit Python Software Foundation. Jego specyficzna struktura oraz cechy takie jak dynamiczne typowanie, automatyczne zarządzanie pamięcią, przenośność, czy duża czytelność i prostota kodu, umożliwiają bardzo szybkie wytwarzanie i utrzymywanie aplikacji.

Biblioteka standardowa języka Python zawiera wiele użytecznych modułów i gotowych rozwiązań, które wspomagają szybką i efektywną implementację kodu. Ponadto dostępny jest *Python Package Index* (PyPI) - zbiór paczek zewnętrznych, tworzonych przez niezależnych programistów, dystrybuowanych na licencjach Open Source. Dzięki takiej mnogości pakietów i modułów język Python może być wykorzystywany w wielu projektach, łącząc różne technologie i dziedziny informatyki. Jednym z przykładów wykorzystania tego języka jest tworzenie aplikacji internetowych za pomocą frameworku Django. Łączy on ze sobą różne technologie wykorzystywane przy tworzeniu serwisów internetowych, zapewniając bardzo dobry mechanizm back-endowy oraz wygodne środowisko.

Ze względu na wyżej wymienione cechy Python znalazł również zastosowanie w analityce i analizie danych, włączając w to analizę danych statystycznych i giełdowych, a także we wspomaganiu obliczeń matematycznych.

2.1. Cechy charakterystyczne języka Python

Podstawową charakterystyczną cechą języka Python jest fakt, iż nie jest on kompilowany lecz interpretowany, czyli tłumaczony do wykonywalnego kodu maszynowego

lub kodu pośredniego. Dzięki użyciu interpretera w konsoli systemowej można bezpośrednio wykonywać kod Pythona w czasie rzeczywistym.

```
Python 2.7.12 (default, Nov 20 2017, 18:23:56)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from sklearn import datasets
>>>
>>> iris = datasets.load_iris()
>>> digits = datasets.load_digits()
>>> digits.data
array([[  0.,   0.,   5., ...,   0.,   0.,   0.],
       [  0.,   0.,   0., ...,  10.,   0.,   0.],
       [  0.,   0.,   0., ...,  16.,   9.,   0.],
       ...,
       [  0.,   0.,   1., ...,   6.,   0.,   0.],
       [  0.,   0.,   2., ...,  12.,   0.,   0.],
       [  0.,   0.,  10., ...,  12.,   1.,   0.]])
>>>
>>>
```

Pozwala to na bardzo szybkie testowanie niewielkich fragmentów kodu, użycia bibliotek, a także przeprowadzanie testowych obliczeń. Jest to również doskonałe narzędzie do sprawdzania i dostosowywania środowiska, w szczególności gdy użyte zostaje symulowane środowisko - program *virtualenv*, który instaluje wybraną wersję interpretera we wskazanym katalogu i umożliwia instalowanie bibliotek niezależnie od tych, które zainstalowane są w systemie.

Kolejną wartą uwagi cechą języka Python jest jego składnia. W odróżnieniu od języków takich jak na przykład Java czy C++, w Pythonie zastosowano tak zwane dynamiczne typowanie. Oznacza to, że podczas definiowania zmiennych nie określa się ich typu. Jest to możliwe, ponieważ w języku Python każdy element, na przykład funkcja, klasa czy też struktura danych jest obiektem. Obiekt ten ma z góry zdefiniowany typ, więc przypisanie jego referencji do konkretnej zmiennej pomaga go w ten sposób określić.

```
  Python 2.7.12 (default, Nov 20 2017, 18:23:56)
  [GCC 5.4.0 20160609] on linux2
  Type "help", "copyright", "credits" or "license" for more information.
  >>> variable_one = 44
  >>> type(variable_one)
  <type 'int'>
  >>>
  >>> variable_one = 'text'
  >>> type(variable_one)
  <type 'str'>
  >>>
```

Jedną z najbardziej użytecznych cech Pythona jest zastosowanie elementów programowania funkcyjnego. Elementami takimi są przykładowo wyrażenia *lambda*, oraz *list comprehension* i *dict comprehension*.

Wyrażenia *lambda* pozwalają na stworzenie i przypisanie do zmiennej krótkiej funkcji, która jest w stanie przyjmować argumenty oraz zwracać wartości. Znajduje to zastosowanie w przypadkach, które wymagają wielokrotnego wykorzystania danego fragmentu kodu, a użycie ich skraca znacznie ilość wypisanych poleceń. Pozwala to uniknąć tworzenia wielu krótkich funkcji lub metod poza obecnie wykorzystywaną przestrzenią, co często wpływa pozytywnie przede wszystkim na czytelność kodu.

Wyrażenia *list comprehension* oraz *dict comprehension* wykorzystywane są do szybkiego tworzenia odpowiednio list oraz słowników. W swojej konstrukcji zawierają pętlę *For*, która iteruje po wskazanej strukturze, na przykład liście, zwracając w każdym kroku jeden jej element. Element ten może być sprawdzony warunkiem wbudowanym w strukturę, oraz następnie zmieniony i wbudowany w nową listę lub słownik.

```
1  Python 2.7.12 (default, Nov 20 2017, 18:23:56)
2  [GCC 5.4.0 20160609] on linux2
3  Type "help", "copyright", "credits" or "license" for more information.
4  >>> test_lambda = lambda x: x+5
5  >>> test_lambda(10)
6  15
7  >>>
8  >>> test_list = [1, 2, 3, 4, 5]
9  >>>
10 >>> list_comprehension = [x+5 for x in test_list]
11 >>> list_comprehension
12 [6, 7, 8, 9, 10]
13 >>>
14 >>> dict_comprehension = {x: x+1 for x in test_list}
15 >>> dict_comprehension
16 {1: 2, 2: 3, 3: 4, 4: 5, 5: 6}
17 >>>
```

Naturalnie, natura i składnia języka Python jest o wiele bardziej różnorodna, a przedstawione przykłady odzwierciedlają jedynie namiastkę jego możliwości. Należałoby wspomnieć tutaj między innymi o posługiwaniu się choćby wbudowanymi strukturami danych, wykorzystaniu programowania orientowanego obiektowo oraz typowych dla niego elementach. Niemniej jednak, biorąc pod uwagę temat niniejszej pracy, którym jest przedstawienie możliwości biblioteki *Scikit-learn*, wyżej wymienione podstawy uzupełnione późniejszymi wyjaśnieniami powinny wystarczyć aby w pełni zrozumieć naturę problemu.

2.2. Python w obliczeniach analitycznych

Język programowania Python jest bardzo dobrym narzędziem wspomagającym obliczenia analityczne. Cechy tego języka zapewniają skoncentrowanie się na bezpośrednim podejściu do problemu tworzenia algorytmów i modeli, minimalizując czas projektowania od podstaw skomplikowanych algorytmów pomocniczych. Jednak największą zaletą tego języka jest dostępność wielu bibliotek z gotowymi rozwiązaniami, które mogą zostać wykorzystane do sprawnej implementacji modeli analitycznych. Podstawowymi bibliotekami wspomagającymi przeprowadzanie obliczeń matematycznych są *NumPy* i *SciPy*.

Pierwsza z nich dostarcza przede wszystkim obiekty wielowymiarowych list oraz szereg metod i funkcji umożliwiających szybką manipulację, przetwarzanie i sortowanie. Zawiera także zestaw metod pozwalających na przeprowadzanie podstawowych działań statystycznych i matematycznych[19]. Stosowana jest w wielu innych bibliotekach analitycznych, na przykład w pakiecie *Scikit-learn*.

Podstawową różnicą pomiędzy obiektami *array* z pakietu *NumPy*, a wbudowanymi listami języka Python jest fakt, iż podczas tworzenia obiektu ustala się stały rozmiar struktury, a każde zwiększenie tego rozmiaru powoduje powstanie nowego obiektu i usunięcie poprzedniego.

```
1  >>> python_list = [1, 2, 3, 4]
2  >>> before_append = id(python_list)
3  >>> python_list.append(5)
4  >>> python_list
5  [1, 2, 3, 4, 5]
6  >>> after_append = id(python_list)
7  >>> print(before_append, after_append)
8  (140635439293360, 140635439293360)
9  >>> print(before_append == after_append)
10 True
11 >>>
12 >>>
13 >>> import numpy as np
14 >>> np_array = np.zeros(shape=(1, 4))
15 >>> np_array
16 array([[ 0.,  0.,  0.,  0.]])
17 >>> before_resize = id(np_array)
18 >>> np_array = np.resize(np_array, (1, 5))
19 >>> np_array
20 array([[ 0.,  0.,  0.,  0.,  0.]])
21 >>> after_resize = id(np_array)
22 >>> print(before_resize, after_resize)
23 (139910114654128, 139910001813664)
24 >>> print(before_resize == after_resize)
25 False
26 >>>
```

Powyższa cecha obiektów biblioteki *NumPy* oznacza, że wykonywanie operacji na takich obiektach powinno być bardziej skuteczne pod względem czasu ich przeprowadzania. Jednakże wielokrotne przebudowywanie struktury obiektu wiąże się z bardzo dużym zużyciem pamięci, dlatego polecane jest stosowanie konwersji i tworzenie obiektów dopiero w momencie, kiedy dane są skompletowane i gotowe do przetwarzania[19].

Biblioteka *SciPy* zbudowana jest na podstawie biblioteki *NumPy* i rozszerza ją o wiele algorytmów analizy danych. Elementami składowymi tej biblioteki są między innymi[19]:
- **cluster** - algorytmy klastrowania
- **linalg** - algebra liniowa
- **signal** - przetwarzanie sygnałów
- **stats** - funkcje i algorytmy statystyczne

Funkcjonalność biblioteki jest bardzo szeroka, dzięki czemu znajduje ona zastosowanie w wielu projektach, a także jest ona częścią składową innych bibliotek analitycznych języka Python. Przykładową metodą należącą do biblioteki *stats* jest *linregress*, która umożliwia przeprowadzenie regresji liniowej dla wskazanych danych.

```
>>> from scipy import stats
>>> import numpy as np
>>>
>>> data_x = np.random.random_integers(1, 99, 10)
>>> data_y = np.random.random_integers(1, 99, 10)
>>> data_x
array([72, 45, 69, 52, 93, 14, 80, 14, 13,  5])
>>> data_y
array([37, 90, 19,  7, 95, 89, 88, 94, 81, 19])
>>>
>>> slope, intercept, r_value,
    p_value, std_err = stats.linregress(data_x, data_y)
>>> print(slope, intercept, r_value, p_value, std_err)
(-0.033350664784966184, 63.424125380672955,
 -0.029541780200591877, 0.93543372355182242, 0.39896357644710517)
>>>
```

2.3. Pakiet Scikit-learn

2.3.1. Cel i przeznaczenie pakietu

Biblioteka *Scikit-learn* zawiera zestaw zaawansowanych narzędzi stosujących uczenie maszynowe do analizy danych w języku Python. Dystrybuowana jest na licencji BSD, która pozwala na modyfikowanie i rozprowadzanie kodu źródłowego, a nawet na włączanie go do produktów komercyjnych pod warunkiem umieszczenia w dokumentacji odpowiednich adnotacji dotyczących autorów. Dzięki temu zaliczana jest do wolnego oprogramowania, które rozwijane jest przez społeczność kontrybutorów. Większa część kodu stworzona jest bezpośrednio w języku Python, lecz niektóre elementy takie jak na przykład implementacje SVM oraz modeli liniowych oparte są na

bibliotekach języka C++, odpowiednio LibSVM oraz LibLinear[21].

Podstawowym założeniem twórców biblioteki jest priorytetyzacja utrzymywania jakości i czytelności kodu, ponad implementację bardzo wielu funkcji[21]. Dodatkowo rozwijana jest wysokiej jakości kompleksowa dokumentacja, co razem stanowi bardzo dobrą bazę do rozwijania całego projektu przez wielu niezależnych deweloperów i wydawania stabilnych wersji produktu. Scikit-learn bazuje na trzech bibliotekach języka Python: *NumPy*, *SciPy* i *Matplotlib*. Stanowią one wymagania systemowe, niezbędne do poprawnego działania pakietu.

W pakiecie *Scikit-learn* algorytmy podzielone są na algorytmy uczenia z nadzorem oraz algorytmy uczenia bez nadzoru[22]. Pierwsze z nich opierają się o podział danych na uczące i testowe, gdzie dane uczące zawierają przykładowe oczekiwane wartości na podstawie których budowany jest model. W zestawie algorytmów uczenia nadzorowanego znaleźć można między innymi[22]:
- Ogólne modele liniowe
- Liniową i kwadratową analizę dyskryminacyjną
- Regresję grzbietową (KRR)
- Maszynę wektorów nośnych (SVM)
- Algorytm k najbliższych sąsiadów
- Proces Gaussa
- Naiwny klasyfikator Bayesa
- Drzewa decyzyjne

Algorytmy uczenia bez nadzoru w pakiecie *Scikit-learn*, dla których dane uczące nie posiadają żadnych wartości odniesienia, możemy natomiast podzielić między innymi na:
- Klasteryzację
- Estymację kowariancji
- Nieliniową redukcję przestrzenną

W niniejszej pracy zastosowane zostały algorytmy uczenia z nadzorem.

2.3.2. Ogólne modele liniowe

W pakiecie *Scikit-learn* przedstawione zostały algorytmy regresji, w których oczekiwane wartości docelowe są liniową kombinacją wartości wejściowych. Podstawę stanowi równanie regresji:

$$\hat{y} = \omega_0 + \omega_1 X_1 + \omega_2 X_2 + ... + \omega_p X_k \tag{2.1}$$

Wektor $\omega = (\omega_1, ..., \omega_p)$ jest utożsamiany z parametrem *coef_*, a wyraz wolny ω_0 z parametrem *intercept_* [22].

Regresja liniowa w pakiecie *Sciki-learn* dostępna jest poprzez obiekt **LinearRegression**. Parametry, jakie przyjmuje ten obiekt w momencie inicjalizacji to[22]:
- *fit_intercept*: boolean, optional, default True
- *normalize*: boolean, optional, default False
- *copy_X*: boolean, optional, default True

- *n_jobs*: int, optional, default 1

Parametr *fit_intercept* przyjmuje wartości typu boolean, a jego domyślna wartość wynosi *True*. W takim wypadku przed procesem dopasowania modelu obliczany jest punkt przecięcia z osią y dla modelu. Jeśli przed procesem dopasowywania modelu dokonano normalizacji danych, parametr może zostać ustawiony wartością *False*.

Parametr *normalize* jest flagą uwzględniającą lub pomijającą wstępną normalizację danych. Przyjmuje wartości typu boolean i domyślnie jest ustawony na *False* i jest ignorowany, gdy parametr *fit_intercept* jest ustawiony wartością *False*. Jeśli przed dokonaniem dopasowania modelu nie została przeprowadzona normalizacja danych, ustawienie tego parametru na *True* spowoduje ich przetworzenie.

Parametr *copy_X* jest odpowiedzialny za ustawienie flagi kopiowania, bądź nadpisywania wartości X. Domyślnie ustawiony jest na wartość *True*, więc w tym przypadku podane dane X są kopiowane.

Parametr *n_jobs* określa ilość procesorów, które będą użyte do przetwarzania danych. Przyjmuje wartości typu integer, a domyślnie ustawiony jest wartością 1. Jeśli podana zostanie wartość -1, wykorzystane zostaną wszystkie dostępne procesory. Znajduje on praktyczne zastosowanie przy przetwarzaniu dużych zbiorów danych, skracając czas wykonania obliczeń.

```
>>> from sklearn.linear_model import LinearRegression
>>> LinearRegression()
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,
                 normalize=False)
>>>
```

Obiekt udostępnia następujące metody:

- *fit*: dopasowanie modelu liniowego
- *get_params*: pobranie parametrów estymacji
- *predict*: predykcja na podstawie modelu liniowego
- *score*: współczynnik determinacji R^2
- *set_params*: ustawienie parametrów estymacji

Metoda *fit()* jest odpowiedzialna za dopasowanie modelu dla podanych danych. Jako argumenty przyjmuje dane testowe X, które powinny być strukturą definiowaną typem *numpy.ndarray*. Przykładowo:

```
>>> X = np.asarray([x for x in range(5)])
>>> X
array([0, 1, 2, 3, 4])
>>> X.reshape(-1, 1)
array([[0],
       [1],
       [2],
       [3],
       [4]])
```

```
10 >>>
```

Wartości y, jako kolejny argument przyjmowany przez funkcję, powinny być zgodne typem z wartościami X, lecz ich postać powinna być macierzą jednowymiarową. Ważną informacją jest fakt, iż struktura X oraz struktura y powinny mieć dokładnie ten sam rozmiar. W innym przypadku niemożliwe jest dokonanie dopasowania, a metoda zwraca błąd *ValueError*. W przypadku danych giełdowych wartości X utożsamiane są przez przekształcone wartości dat kolejnych próbek danych testowych, natomiast wartości y to odpowiadające im ceny akcji lub wielkości wolumenu.

Kolejną metodą klasy *LinearRegression* jest *predict()*. Umożliwia ona dokonanie obliczeń predykcji dla podanych danych testowych X, na podstawie dopasowanego przy pomocy metody *fit()* modelu. Przyjmowane dane testowe powinny być zgodne zarówno typem, jak i formatem struktury z danymi testowymi X użytymi do przeprowadzenia dopasowania modelu metodą *fit()*.

```
1  >>> import random
2  >>> import numpy as np
3  >>> from sklearn.linear_model import LinearRegression
4  >>>
5  >>> X = np.asarray(range(1, 10))
6  >>> X.reshape(-1, 1)
7  >>> y = np.asarray(sorted([random.uniform(1.0, 2.0)
8                             for x in range(10)]))
9  >>> print(y)
10 array([ 1.03622622,  1.08102786,  1.15758493,  1.30854005,
11         1.43797429,  1.47473032,  1.5770967 ,  1.64707456,
12         1.7256157 ,  1.8449893 ])
13 >>> X_test = np.asarray(11, 15)
14 >>> X_test = X_test.reshape(-1, 1)
15 >>>
16 >>> linear = LinearRegression()
17 >>> linear.fit(X, y)
18 LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1,
19                  normalize=False)
20 >>> linear.predict(X_test)
21 array([ 2.02318035,  2.11457948,  2.20597861,  2.29737775])
22 >>> linear.predict(X)
23 array([ 1.0177899 ,  1.10918903,  1.20058816,  1.2919873 ,
24         1.38338643,  1.47478556,  1.56618469,  1.65758382,
25         1.74898295,  1.84038209])
26 >>>
```

2.3.3. Nieliniowe modele regresji

W pakiecie *Scikit-learn* regresja grzbietowa reprezentowana jest przez klasę *KernelRidge*. Jest ona w istocie połączeniem algorytmu regresji grzbietowej z tak zwany-

mi funkcjami jądra (kernel functions). Najważniejszymi parametrami przyjmowanymi przez tą klasę są:

- *alpha*: float, array
- *kernel*: string, default *linear*
- *gamma*: float, default None

Parametr *kernel* może przyjmować wartości nazw funkcji jądra, na przykład: rbf, laplacian, polynomial lub linear. W niniejszej pracy we wszystkich nieliniowych modelach regresji zastosowano funkcję *rbf*.

Parametry *alpha* i *gamma* dobierane są dzięki klasie *sklearn.model_selection.GridSearchCV*, która przyjmuje listy parametrów wraz z oryginalnym obiektem regresji i iterując po ich iloczynie kartezjańskim dobiera taką kombinację, która zwraca najlepsze dopasowanie metody *fit*.

```
>>> krr = GridSearchCV(KernelRidge(kernel='rbf'),
                               param_grid={
                        "alpha": [1e0, 0.1, 1e-2, 1e-3],
                        "gamma": np.logspace(-2, 2, 5)
                        })
>>> >>> krr
GridSearchCV(cv=None, error_score='raise',
             estimator=KernelRidge(alpha=1, coef0=1, degree=3,
                                   gamma=None, kernel='rbf',
                                   kernel_params=None),
             fit_params={}, iid=True, n_jobs=1,
             param_grid={'alpha': [1.0, 0.1, 0.01, 0.001],
                         'gamma': array([  1.00000e-02,
                                           1.00000e-01,
                                           1.00000e+00,
                                           1.00000e+01,
                                           1.00000e+02])},
             pre_dispatch='2*n_jobs', refit=True,
             return_train_score=True, scoring=None, verbose=0)
>>>
```

Pozostałymi, zastosowanymi w pracy metodami regresji nieliniowej są:

- Regresja Wektorów nośnych - *sklearn.svm.SVR* - przyjmująca parametry C i *epsilon*
- Regresja procesu Gaussa - *sklearn.gaussian_process.GaussianProcessRegressor* - przyjmująca prarmetr *alpha*

Do przeprowadzenia procesu normalizacji danych może być użyta klasa *StandardScaler* pakietu *Scikit-learn*.

```
>>> sc_x = StandardScaler()
>>> sc_y = StandardScaler()

>>> sort_dates = sc_x.fit_transform(dates_delta)
>>> sort_values = sc_y.fit_transform(sort_values)

>>> sort_dates = sc_x.inverse_transform(sort_dates)
```

```
>>> sort_values = sc_y.inverse_transform(sort_values)
>>>
```

Metoda *fit_transform* utworzonego obiektu klasy *StandardScaler* umożliwia przeprowadzenie normalizacji, natomiast metoda *inverse_transform* umożliwia powrót do oryginalnych wartości.

Rozdział 3

Przedstawienie aplikacji

Na potrzeby tej pracy została napisana aplikacja w języku programowania Python w wersji 2.7, która ma za zadanie przedstawić część możliwości pakietu *Scikit-learn* analizując dane giełdowe. Zastosowany został graficzny interfejs użytkownika (GUI), dzięki któremu jest możliwa szybka zmiana wybranych opcji, oraz dokonywanie wielu analiz bez potrzeby ponownego uruchamiania aplikacji. Przeznaczeniem aplikacji jest analiza danych, tak więc generowane wyniki zapisywane są we wskazanym przez użytkownika katalogu, a także wyświetlane bezpośrednio w aplikacji.

3.1. Podstawowe założenia

Aplikacja została napisana w modelu orientowanym obiektowo. Starano się zachować wyraźny podział na część biznesową, oraz interfejs użytkownika. W związku z tym struktura katalogów w projekcie wygląda następująco:

```
/
└── /mdata
    ├── /client
    ├── /drivers
    │   ├── /config
    │   └── /databases
    └── /gui
        ├── /buttons
        ├── /items
        ├── /layouts
        └── /screens
```

W katalogu */client* umieszczony został kod odpowiadający za pobieranie danych giełdowych, przeprowadzanie analizy regresji, oraz generowanie wykresów. Katalog */drivers* zawiera mechanizmy zapisu i odczytu głównego pliku konfiguracyjnego aplikacji, oraz mechanizm bazodanowy napisany przy pomocy pakietu *SqlAlchemy*. Ostatni katalog */gui* zawiera wszystkie elementy graficznego interfejsu użytkownika, w podziale na przyciski, układy, okna oraz pozostałe elementy.

Przeprowadzono analizę kodu dla katalogu */mdata* programem *pylint*.
Według tego raportu kod aplikacji uzyskał ocenę 7.28/10 i zawiera:

- 43 zaimplementowane klasy
- 199 zaimplementowanych metod
- 2 funkcje pozaklasowe
- 1707 linii kodu

3.1.1. Dane giełdowe

Ze względu na ogromne możliwości pakietu *Scikit-learn*, a także na ograniczenia dotyczące samej konstrukcji aplikacji, poczynione zostały pewne założenia definiujące działanie aplikacji.
Główne założenia związane z pobieraniem danych giełdowych:
- Dane giełdowe pobierane są na podstawie wcześniej przygotowanej bazy firm giełdy amerykańskiej
- Pobieranie danych giełdowych odbywa się za pomocą API *Yahoo Finance*
- Rodzaj danych jest determinowany jedynie przez użyte API, a ich zakres wybierany jest poprzez podanie daty początkowej i końcowej

Zastosowane w pracy API *Yahoo Finance* umożliwia pobieranie danych jedynie w jednej postaci, bez możliwości wyboru lub dodania dodatkowych elementów. Każde pobranie danych w tym samym zakresie powoduje otrzymanie dokładnie takich samych rezultatów, co kwalifikuje to API do udziału w analizie. Do bezpośredniego pobierania danych zastosowano funkcję *DataReader* pakietu *pandas_datareader*. Umożliwia ona prostą implementację API w języku Python oraz zwracanie danych w pożądanym formacie *pandas.DataFrame*.
Dane zwracane przez API dzielą się na:
- *Open*: cena początkowa (otwarcia)
- *Close*: cena końcowa (zamknięcia)
- *High*: najwyższa wartość ceny
- *Low*: najniższa wartość ceny
- *Adj Close*: skorygowana wartość ceny końcowej
- *Volume*: wartość wolumenu

Przykładowe zastosowanie funkcji *DataReader* do pobierania danych giełdowych za pomocą API *Yahoo Finance*:

```
>>> from pandas_datareader.data import DataReader
>>> source = 'yahoo'
>>> company = 'AAPL'
>>> start = '2017-12-01'
>>> end = '2017-12-02'
>>> DataReader(company, source, start, end)
                Open        High         Low \
Date
2017-12-01  170.429993  172.139999  168.440002
2017-12-02  169.949997  171.669998  168.500000

                Close   Adj Close     Volume
Date
2017-12-01  171.850006  171.850006   41527200
2017-12-02  171.050003  171.050003   39759300
```

```
16 >>>
```

3.1.2. Analiza danych giełdowych

Główne założenia związane z analizą danych:
- Do analizy można wykorzystać dane jednego, wskazanego przez użytkownika rodzaju (Open, Close, High, Low)
- Analiza danych może być przeprowadzona jedną z czterech zaimplementowanych metod regresji z pakietu *Scikit-learn*
- Podział danych na zbiór danych uczących i zbiór danych testowych następuje poprzez wybranie wartości procentowej ilości danych uczących w oknie opcji aplikacji

Ograniczenie rodzaju analizowanych danych umożliwia zawężenie czynników, które mogą wpłynąć na wyniki badań, co prowadzi do poprawy wiarygodności otrzymanych rezultatów. Z drugiej jednak strony, może to prowadzić do otrzymania wyników różniących się od tych, które byłyby otrzymywane z pełnego zbioru danych.

Do przeprowadzenia badań wybrano i wykorzystano jedną metodę analizy regresji liniowej, oraz trzy metody analizy regresji nieliniowych dostępne w pakiecie *Scikit-learn*. Są to:
- Regresja liniowa
- Regresja Grzbietowa
- Regresja Wektorów Nośnych (SVR)
- Regresja Procesu Gaussa (GPR)

Przeprowadzone analizy skupiają się na róznicach pomiędzy zdolnościami predykcyjnymi wyżej wymienionych metod regresji oraz ich zdolnościach do opisywania trendów dla danego zakresu danych. Opisana została także różnica w dokładności metod regresji w zależności od proporcji ilości danych uczących i testowych.

3.2. Zastosowane pakiety języka Python

3.2.1. Kivy

Kivy jest biblioteką języka programowania Python umożliwiającą tworzenie graficznego interfejsu użytkownika przeznaczonego na wiele platform, takich jak Windows, Linux, iOS czy Android. Jego główną zaletą jest możliwość oddzielenia warstwy biznesowej aplikacji od warstwy prezentacji, co wpływa zarówno na czytelność, skalowalność i ogólną jakość kodu.

Są możliwe dwa warianty budowania aplikacji za pomocą tego pakietu. Pierwszy umożliwia tworzenie plików w języku *Kivy Design Language*, które definiują wygląd aplikacji. Pliki te są następnie importowane do kodu w języku Python, a na ich podstawie generowane jest GUI.

Drugi wariant polega na użyciu odpowiednich klas udostępnianych przez pakiet bezpośrednio kodzie języka Python. Ze względu na lepszą organizację kodu, w niniej-

szej pracy zastosowano wariant drugi.

Podstawowym podziałem okien w pakiecie *Kivy* są okna i układy. Okna pozwalają na tworzenie wielu niezależnych od siebie kart i zakładek. Żeby umożliwić wstawienie jakiegokolwiek elementu funkcjonalnego do danego okna, należy najpierw "nałożyć na niego" jeden lub kilka układów. W taki sposób można dzielić odpowiednie okna na części, na przykład tworząc "menu".

W niniejszej pracy zastosowano wzorzec polegający na tworzeniu i umieszczaniu każdego elementu GUI w osobnej klasie, oraz osobnym pliku *.py*. Umożliwia to większą kontrolę nad poszczególnymi elementami, a także zwiększa skalowalność.
Przykładowy kod prezentujący implementację przycisku wyboru okna analizy regresji:

```
1  from kivy.uix.button import Button
2
3
4  class RegressionButton(Button):
5      def __init__(self, base, **kwargs):
6          self.base = base
7          self.text = 'Regression\n  Analysis'
8          super(RegressionButton, self).__init__(**kwargs)
9
10     def on_press(self):
11         if not self.base.screen_manager.current\
12                 == self.base.screens['regression'].name:
13             self.base.screen_manager.switch_to(
14                             self.base.screens['regression'])
```

3.2.2. Pandas i Matplotlib

Biblioteka *Pandas* udostępnia bardzo rozbudowane narzędzia do przechowywania i analizy danych. W niniejszej pracy wykorzystano należącą do niej klasę *DataFrame* w celu przechowywania danych giełdowych pobranych za pomocą funkcji *DataReader*, oraz przygotowywania tych danych do wygenerowania wykresów.

Większość elementów biblioteki *Pandas* wykorzystuje możliwości pakietów takich jak *NumPy* i *SciPy* jako bazę, tak więc są one w pełni kompatybilne z obiektami tychże pakietów. Głównym elementem wykorzystanym w pracy jest obiekt *DataFrame*, który jest kontenerem zarówno dla oryginalnych pobranych danych giełdowych, jak i dla wyników analizy regresji.
Przykładowo, po dokonanej analizy regresji budowany jest obiekt:

```
1  df_all_results = pd.DataFrame(
2              index=all_dates,
3              data={chosen_data_type: all_values,
4                    'Regression': sort_pred})
```

Przyjmuje on jako argumenty:

- *index*: obiekt typu *numpy.ndarray* zawierający listę dat dla osi X wykresu
- *data*: słownik zawierający oryginalne dane, oraz wyniki predykcji, także typu *numpy.ndarray*

Biblioteka *Pandas* jest także w pełni kompatybilna z biblioteką *Matplotlib*, która umożliwia generowanie wykresów na podstawie podanych danych. W pracy zastosowano metodę *plot()* udostępnianą przez obiekt *DataFrame*, która zwraca gotowy obiekt klasy *matplotlib.axes.Axes*. Obiekt ten, uzupełniony o niezbędne elementy i parametry, jest gotowy do wyświetlenia lub zapisania wykresu na dysku, poprzez "wyciągnięcie" figury metodą *get_figure()* i zastosowania na niej metody *savefig()*.

Przykładowy kod generowania wykresu wyników analizy regresji, z dodaniem pionowej linii oznaczającej podział danych na uczące i testowe:

```
1  df_reg_results = self.reg_data['df_all_results']
2
3  self.regression_plot = df_reg_results.plot(title=plot_title,
4                                              grid=True)
5  self.regression_plot.axvline(x=self.reg_data['train_test_v_date'],
6                               color='red')
7  self.regression_plot.set_ylabel('Price')
8  self.regression_plot.set_xlabel('Date')
9
10 self.regression_fig = self.regression_plot.get_figure()
11
12 self.regression_fig.savefig('{dir}\{file}'\
13                              .format(dir=self._get_tempdir(),
14                                      file=self.diagram_names['reg']))
```

3.2.3. Scikit-learn

Biblioteka *Scikit-learn* i jej możliwości jest głównym tematem niniejszej pracy. W aplikacji zaimplementowano cztery rodzaje analizy regresji: regresję liniową, grzbietową, wektorów nośnych i procesu Gaussa.

Przygotowanie danych do analizy dla wszystkich metod jest identyczne:

```
1  data = deepcopy(self.data)
2  split_rate = self.get_split_rate()
3  chosen_data_type = self.get_data_type()
4  all_data_types = self.regression_data_types
5  data_to_drop = [data_t for data_t in all_data_types
6                  if data_t != chosen_data_type]
7
8  sort = data.drop(data_to_drop, axis=1)
9  sort_values = np.concatenate(sort.values)
10 sort_dates = sort.index.values
11
12 original_dates, dates_delta = self.change_dates(sort_dates)
```

```
13  dates_delta = np.reshape(dates_delta, (len(dates_delta), 1))
14
15  sc_x = StandardScaler()
16  sc_y = StandardScaler()
17
18  sort_dates = sc_x.fit_transform(dates_delta)
19  sort_values = sc_y.fit_transform(sort_values)
20
21  sort_dates_train, sort_dates_test = self.split_data(sort_dates,
22                                                      split_rate)
23  sort_values_train, sort_values_test = self.split_data(sort_values,
24                                                        split_rate)
```

W pierwszej kolejności pobrane dane zawierające wszystkie kolumny z cenami walut wolumenem są redukowane do jedjen wybranej wcześniej kolumny. Tak przygotowane dane dzielone są na ceny i daty i przechowywane w obiektach typu *numpy.ndarray*, odpowiednio: *sort_values* i *sort_dates*. Daty konwertowane są na wartości typu *Integer* przedstawiające odległość danej daty od pierwszej podanej na potrzeby późniejszej analizy regresji. Przekształcana jest również macierz dat z jednowymiarowej na dwuwymiarową.

Kolejnym krokiem jest użycie obiektu klasy *sklearn.preprocessing.StandardScaler* w celu przeprowadzenia normalizacji danych. Obiekt ten udostępnia metody *fit_transform()* oraz *inverse_transform()*, które pozwalają na odpowiednio: przeprowadzenie normalizacji danych oraz powrót do oryginalnych wartości.

Tak przygotowane dane są następnie dzielone na zbiory uczące i testowe za pomocą osobnej metody.

Obiekt klasy *sklearn.linear_model.LinearRegression* został użyty do przeprowadzenia regresji liniowej:

```
1  linear_reg = LinearRegression(fit_intercept=False)
2  linear_reg.fit(sort_dates_train, sort_values_train)
3  sort_pred = linear_reg.predict(sort_dates)
```

Pierwszym krokiem jest stworzenie obiektu *linear_reg*, podając jako argument *fit_intercept=False*. Tak przygotowany obiekt udostępnia metodę *fit()*, której argumentami są zbiory danych uczących, odpowiednio dat i cen akcji. Metoda ta przygotowuje model do predykcji, którą wykonuje się za pomocą kolejnej metody *predict()*, jako argumenty podając zbiór wszystkich dat, zarówno testowych jak i uczących. Pozwala to na stworzenie prostej regresji dla całego zbioru danych.

Klasa *sklearn.kernel_ridge.KernelRidge* pozwala na przeprowadzenie analizy regresji grzbietowej która, zaliczając się do regresji nieliniowych, została wzbogacona o obiekt klasy *sklearn.model_selection.GridSearchCV*:

```
1  krr = GridSearchCV(
```

```
2          KernelRidge(kernel='rbf'),
3                      param_grid={"alpha": [1e0, 0.1, 1e-2, 1e-3],
4                                  "gamma": np.logspace(-2, 2, 5)})
5
6  krr.fit(sort_dates_train, sort_values_train)
7  sort_pred = krr.predict(sort_dates)
```

Obiekt *KernelRidge* przyjmuje parametry *alpha* i *gamma*, których dopasowanie jest niezmierne istotne przy przeprowadzaniu poprawnej analizy. Znalezienie poprawnego dopasowania parametrów jest znacznie uproszczone w przypadku wykorzystania obiektu *GridSearchCV*, który dla podanego obiektu regresji i parametrów określonych w *param_grid* automatycznie dopasowuje najlepsze parametry. Tworzy on iloczyn kartezjański wszystkich podanych argumentów, i podczas działania kolejnego kroku, czyli metody *fit()* wybiera taki zbiór, który jest najlepiej dopasowany.

W przypadku pozostałych metod regresji, czyli Regresji Wektorów Nośnych i Regresji Procesu Gaussa zastosowano ten sam mechanizm dobierania parametrów obiektu. Pierwsza z nich udostępniona jest przez obiekt klasy *sklearn.svm.SVR*, a druga przez obiekt klasy *sklearn.gaussian_process.GaussianProcessRegressor*.

3.3. Opis funkcjonalności aplikacji

Aplikacja została napisana w języku programowania Python w środowisku Windows 10 Home. Do poprawnego uruchomienia wymaga zainstalowanego interpretera języka Python w wersji 2.7, oraz zainstalowanych bibliotek zewnętrznych, z których najważniejsze to:

- Matplotlib, Pandas, Numpy i SciPy
- Scikit-learn
- Kivy
- Pandas_datareader
- Configparser
- SqlAlchemy

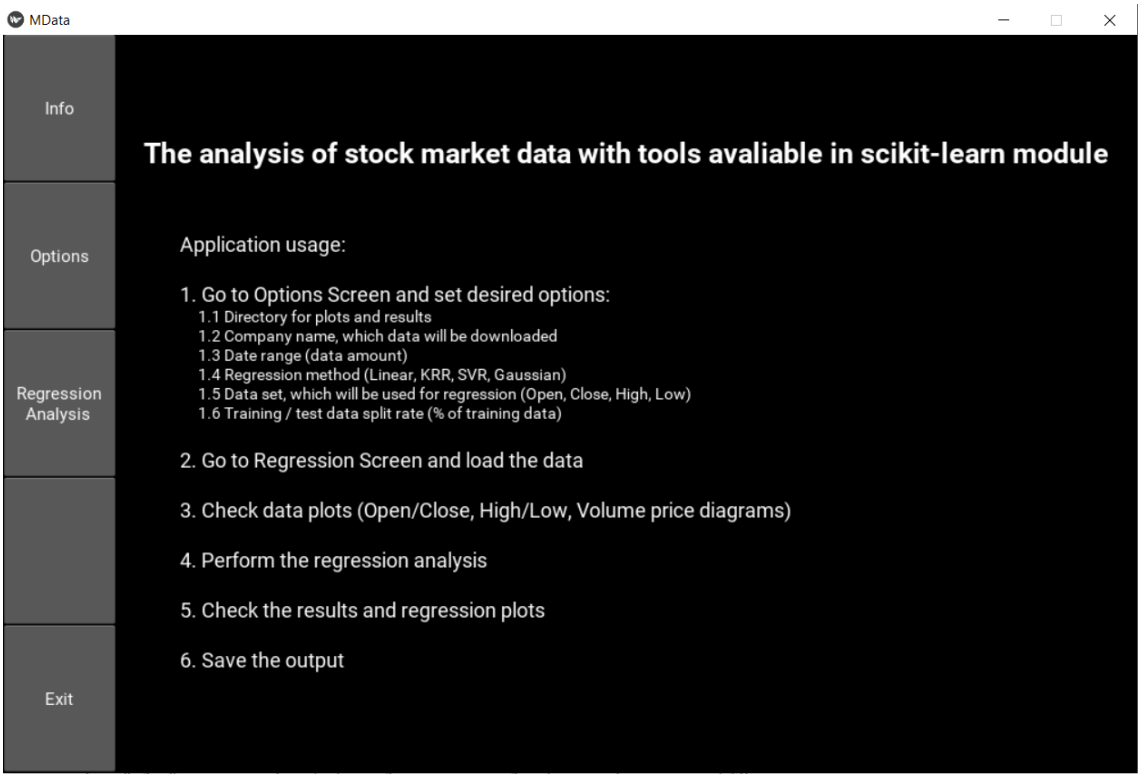
Wszystkie biblioteki użyte w pracy dostępne są na licencjach open-source i są gotowe do pobrania poprzez interfejs *PyPa - Python Package Index*.

Aplikacja podzielona jest na trzy okna:

- Okno instrukcji
- Okno opcji
- Okno analizy regresji

Po uruchomieniu aplikacji użytkownikowi jako pierwsze przedstawiane jest okno instrukcji, które zawiera schemat kroków potrzebnych do poprawnego przeprowadzenia analizy.

Z lewej strony dostępny jest pasek menu, który zawiera przyciski: *Info*, *Options*, *Regression analysis* oraz *Exit*. Każdy z nich pozwala na wyświetlenie innego okna aplikacji, z wyjątkiem przycisku *Exit* który wywołuje okno z zapytaniem o zakończenie pracy programu.



Rysunek 3.1. Aplikacja: okno instrukcji

3.3.1. Okno opcji

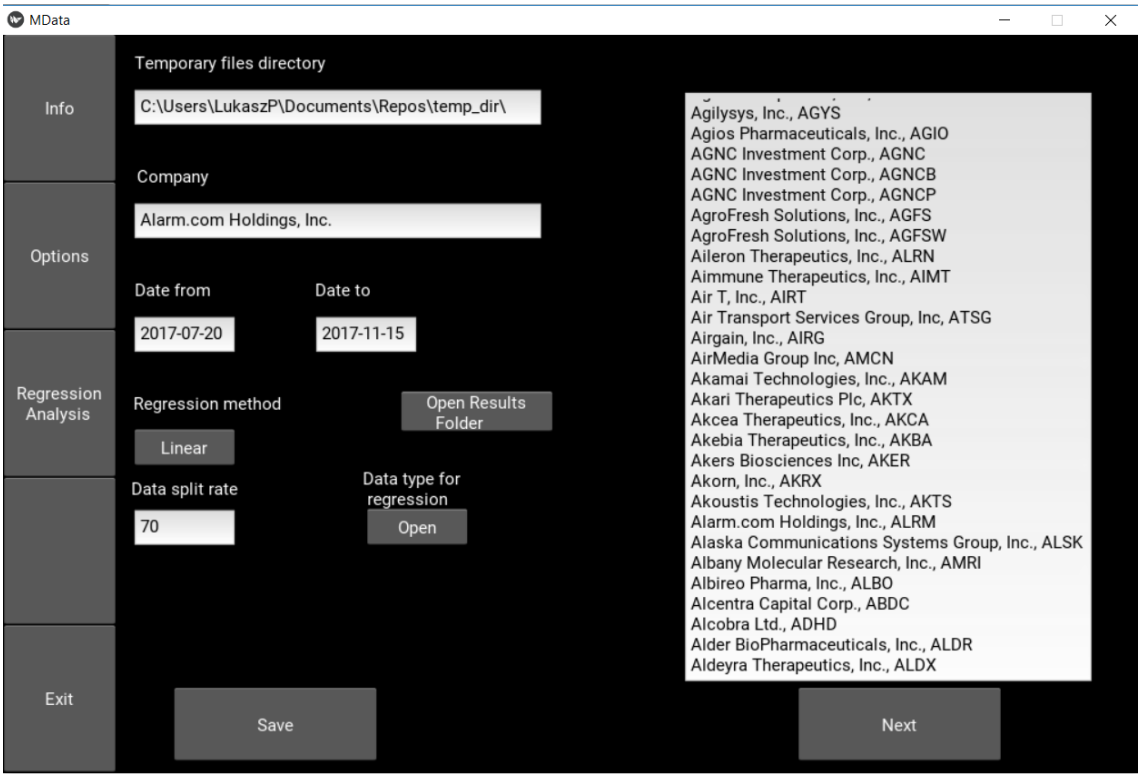
W oknie opcji użytkownik może zmienić lub ustawić parametry, które będą używane w późniejszej analizie regresji.

Parametry możliwe do zmiany to:

- *Temporary files directory*: ścieżka do folderu, do którego mają być zapisywane wyniki
- *Company*: nazwa firmy, dla której mają być pobrane dane
- *Date from*: data od której mają być pobrane dane, w formacie yyyy-mm-dd
- *Date to*: data do której mają być pobierane dane, w formacie yyyy-mm-dd
- *Regression method*: lista typu dropdown z możliwością wyboru metody regresji
- *Data split rate*: wartość procentowa ilości danych uczących względem danych testowych
- *Data type for regression*: lista typu dropdown umożliwiająca wybór typu danych do analizy

Do zapisania konfiguracji służy przycisk *Save* który, ze względu na walidację każdego pola lub listy, po wciśnięciu poinformuje użytkownika o powodzeniu lub niepowodzeniu operacji. W przypadku niepowodzenia użytkownikowi zostaje wskazane pole, które zostało źle wypełnione.

Lista firm, dla których jest możliwość pobrania danych znajduje się w zablokowanym polu tekstowym z prawej strony okna. Jako, iż istnieje możliwość kopiowania tekstu z tego pola, aby wypełnić parametr *Company* należy skopiować dokładną



Rysunek 3.2. Aplikacja: okno opcji

nazwę firmy i wkleić ją we właściwe miejsce. Przycisk *Next* pozwala natomiast na wyświetlenie kolejnej części listy firm, która jest posortowana alfabetycznie.

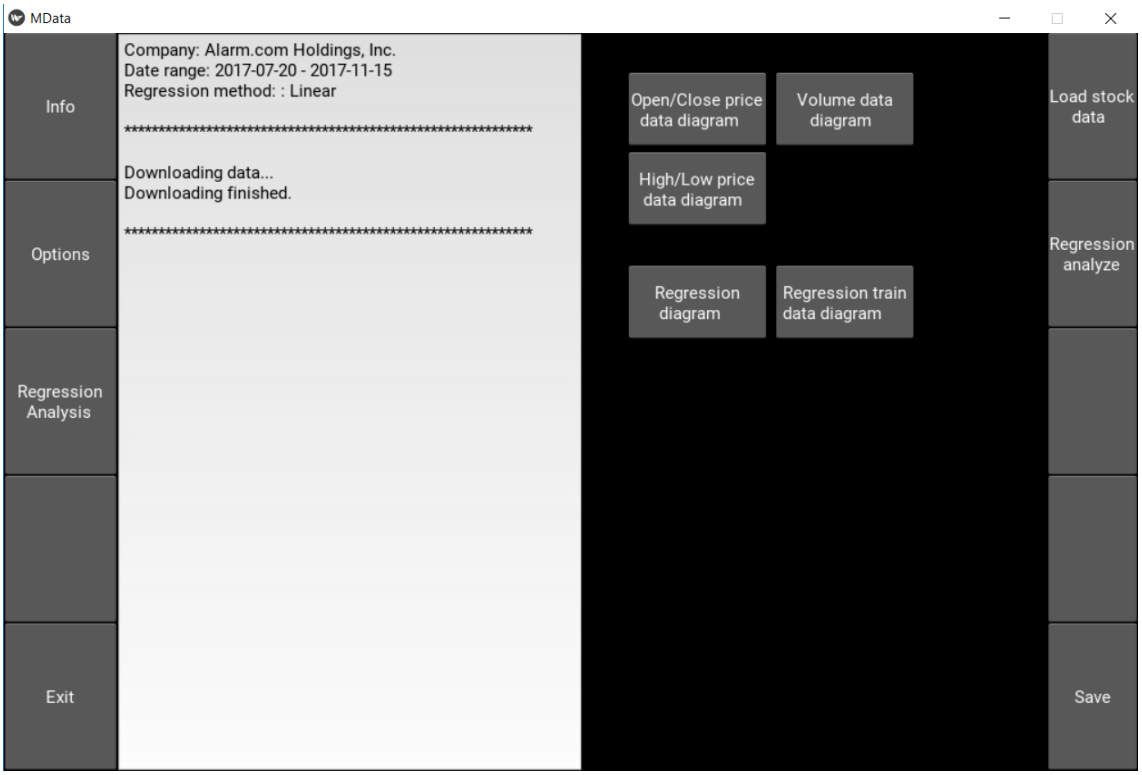
3.3.2. Okno analizy regresji

W oknie analizy regresji użytkownik może przeprowadzić operacje: pobrania danch giełdowych i przeprowadzenia analizy, zgodnie z parametrami ustawionymi w oknie opcji.

Z prawej strony okna znajduje się pasek menu z przyciskami: *Load stock data*, *Regression analyze* oraz *Save*. Z lewej strony znajduje się natomiast zablokowane pole tekstowe, w którym pojawiają się wyniki i informacje. W centralnej części obecne są przyciski, które są odpowiedzialne za wyświetlanie odpowiednich wykresów:

- *Open/Close price data diagram*
- *High/Low price data diagram*
- *Volume data diagram*
- *Regression diagram*
- *Regression train data diagram*

Pierwsze trzy wykresy przedstawiają niemodyfikowane dane giełdowe pobrane za pomocą przycisku *Load stock data* i tylko po jego wciśnięciu będą dostępne. Ostatnie dwa wykresy udostępnione zostają po przeprowadzeniu analizy regresji i przedstawiają odpowiednio: rezultat analizy regresji dla całości danych oraz rezultat zawężony jedynie do danych testowych.



Rysunek 3.3. Aplikacja: okno analizy regresji (pobrane dane giełdowe)

3.4. diagramy UML

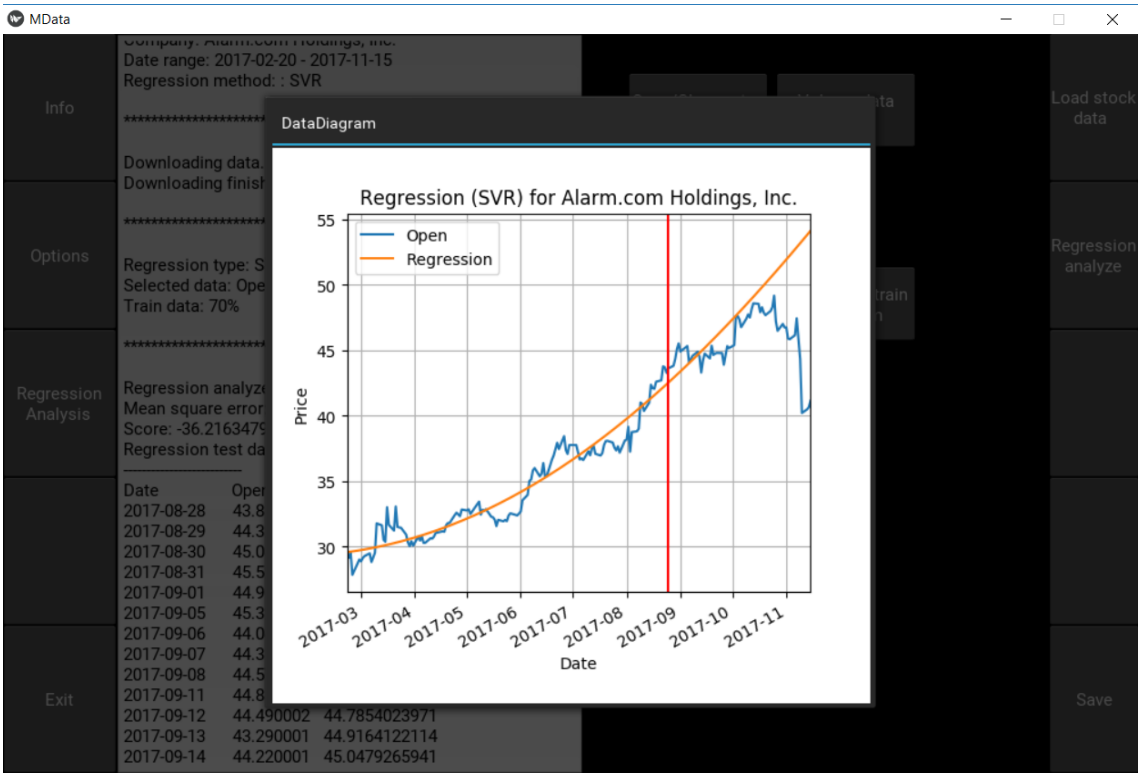
Ze względu na dużą ilość klas, w szczególności należących do graficznego interfejsu użytkownika, przedstawienie pełnego diagramu klasowego UML nie jest możliwe. Wyróżnić można natomiast trzy główne funkcje aplikacji, które przedstawione na diagramie UML mogą posłużyć jako dobre odzwierciedlenie działania aplikacji.

3.4.1. Zapis parametrów w oknie opcji

Pierwszą główną funkcją aplikacji jest zapis wybranych przez użytkownika parametrów do pliku konfiguracyjnego.

Podstawową klasą jest *OptionsScreenLayout*, która dziedziczy po klasie *FloatLayout* należącej do biblioteki *Kivy*, a także implementuje klasy odpowiedzialne za zarządzenie konfiguracją (*ConfigManagement*) oraz za przycisk zapisu konfiguracji (*SaveButton.*)

Pozostałe klasy, w zależności od przeznaczenia, dziedziczą po pochodzących z pakietu *Kivy* klasach *TextInput* praz *Button*. Dwie klasy odpowiedzialne za implementację list rozwijanych dodatkowo implementują klasę *DropDown*.



Rysunek 3.4. Aplikacja: Wykres rezultatu analizy regresji

3.4.2. Pobieranie danych giełdowych

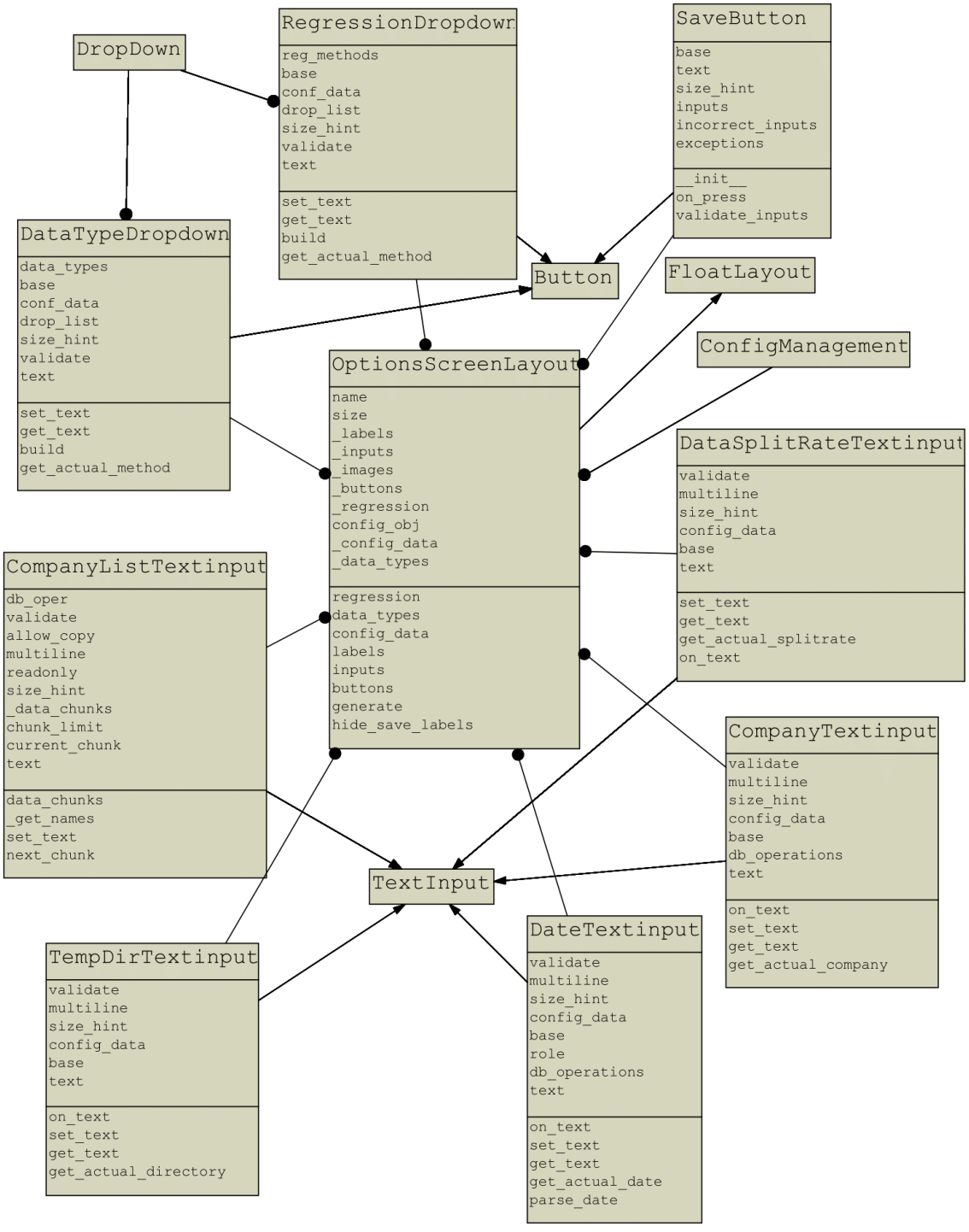
Kolejną funkcją aplikacji przedstawioną na schemacie UML jest mechanizm pobierania danych giełdowych.

Przycisk odpowiedzialny za akcję, *LoadDataButton*, dziedziczy po klasie *Button* biblioteki *Kivy*, oraz implementuje klasy zarządzania konfiguracją oraz pobierania danch (*DataDownload*). Klasa odpowiadająca za pobieranie danych implementuje natomiast bazodanową klasę *Operations*, która wraz z *CompanyInfo* oraz *DbDriver* umożliwiają dostęp do bazy danych nazw i znaczników giełdowych firm.

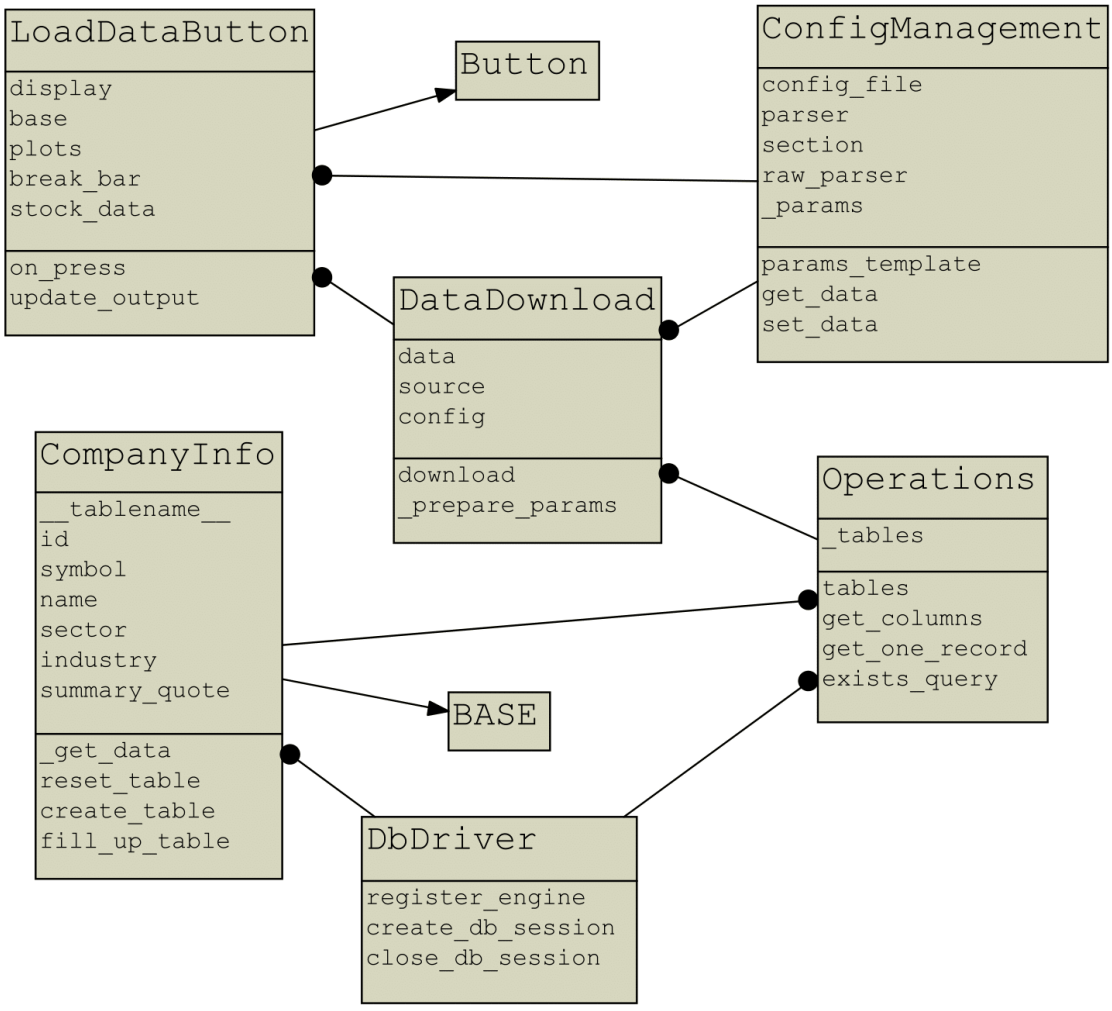
3.4.3. Analiza regresji

Ostatnią przedstawioną funkcją aplikacji jest przeprowadzanie analizy regresji.

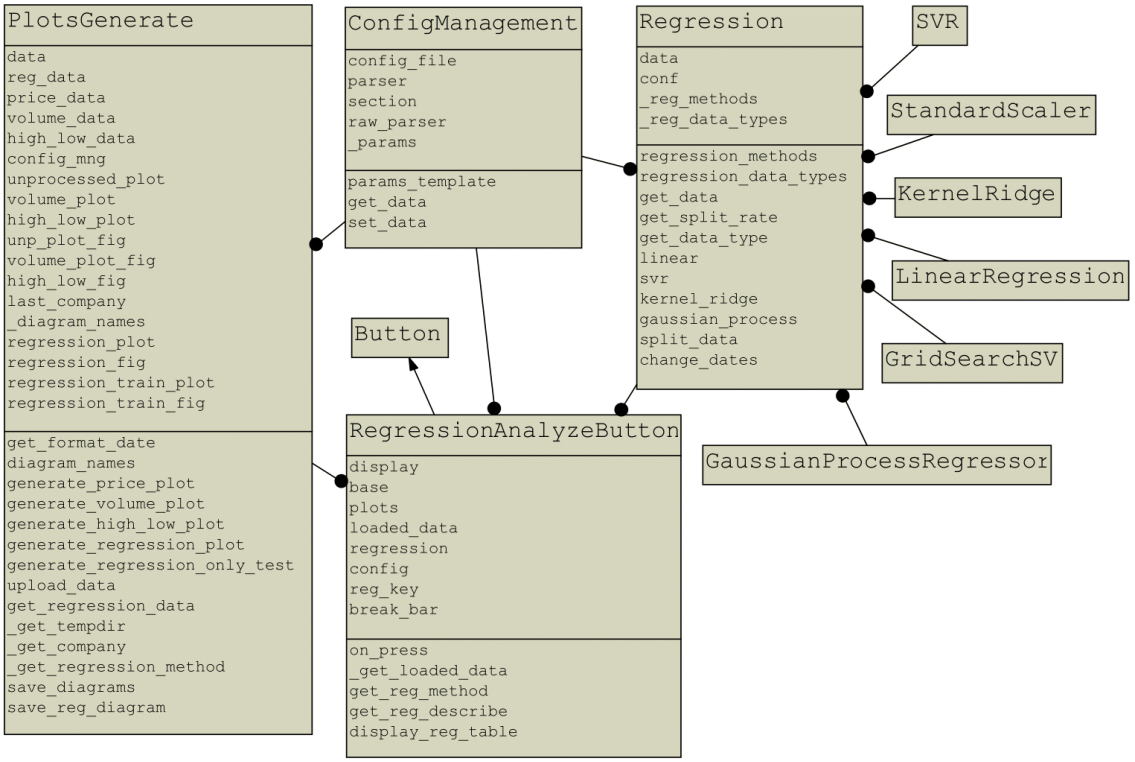
Przycisk *RegressionAnalyzeButton* implementuje zarówno klasy *PlotsGenerate*, *ConfigManagement* oraz *Regression*. Klasa *Regression* natomiast, odpowiedzialna za przeprowadzenie właściwej analizy i zwrócenie wyników, implementuje szereg klas pochodzących z pakietu *Scikit-learn*.



Rysunek 3.5. Diagram UML: zapis parametrów w oknie opcji



Rysunek 3.6. Diagram UML: pobierane danych giełdowych



Rysunek 3.7. Diagram UML: analiza regresji

Rozdział 4

Testy aplikacji

4.1. Cel przeprowadzonych analiz

Wykorzystując napisaną na potrzeby niniejszej pracy aplikację przeprowadzono testy, które zmierzają do porównania wybranych modeli regresji dostępnych w pakiecie *Scikit-learn*.

Testy przeprowadzone zostały na dwóch zbiorach danych: cen giełdowych firmy Microsoft w zakresie od 2017-01-01 do 2017-10-30, oraz cen giełdowych firmy Intel w zakresie od 2017-01-01 do 2017-04-30. W dalszej części zbiory te będą nazywane odpowiednio: zbiór szeroki i zbiór wąski. Dane wykorzystane do testów były cenami otwarcia.

Każdy przeprowadzony test uwzględnia wyliczenie dwóch wartości: średniego błędu kwadratowego oraz wyniku predykcji.

Średni błąd kwadratowy liczony jest poprzez wykorzystanie funkcji *sklearn.metrics.mean_squared* z pakietu *Scikit-learn*, a jego wartość w wypadku predykcji idealnej wynosi zero. Liczony jest na podstawie wartości predykcji dla danych testowych. Wynik predykcji jest natomiast liczony poprzez wywołanie metody *score()* obieku danego modelu regresji, a jego wartość zmierza do osiągnięcia 1.0 w przypadku idealnym. Liczony jest na podstawie wyników predykcji zarówno dla zbioru uczącego jak i testowego.

Przeprowadzono osobne testy dla każdej z trzech wartości procentowych ilości danych uczących w stosunku do ilości danych testowych: 20%, 50% oraz 80%.

Wybrane modele regresji to:
- Regresja Liniowa
- Regresja Grzbietowa (KRR)
- Regresja Wektorów Nośnych (SVR)
- Regresja Procesu Gaussa (GPR)

Reasumując, dla każdej z metod regresji wykonano sześć testów.

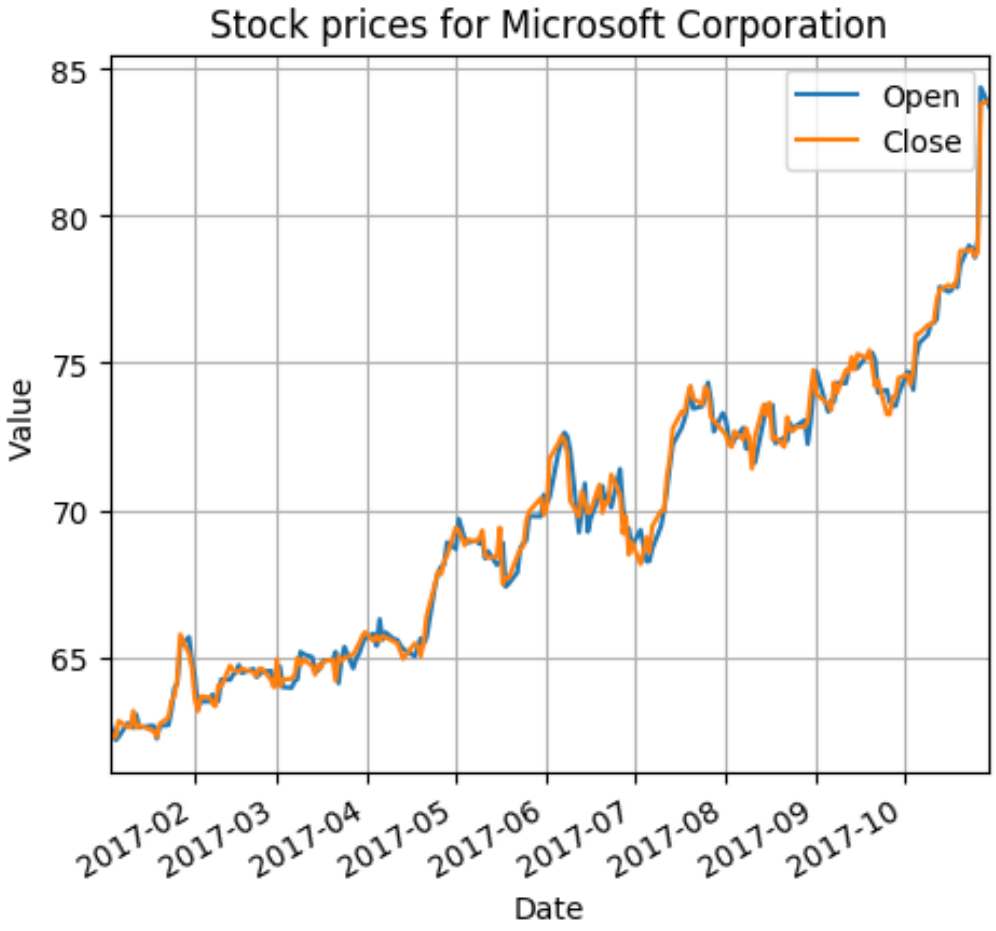
Celem testów jest porównanie modeli regresji dostępnych w pakiecie *Scikit-learn* i wyciągnięcie wniosków dotyczących:

- dokładności predykcji modeli w zależności od ilości danych uczących oraz całkowitej ilości danych
- zdolności modeli do reprezentacji trendu
- wpływu zmiany ilości danych uczących na dopasowanie modeli
- wpływu całkowitej ilości danych na dopasowanie modeli

4.2. Testy zbioru danych: Microsoft

4.2.1. Informacje ogólne

Zakres danych użytych do testów wynosi 209 próbek, w przedziale dat od 2017-01-01 do 2017-10-30 z krokiem wynoszącym jeden dzień.



Rysunek 4.1. Wykres cen otwarcia i zamknięcia firmy Microsoft

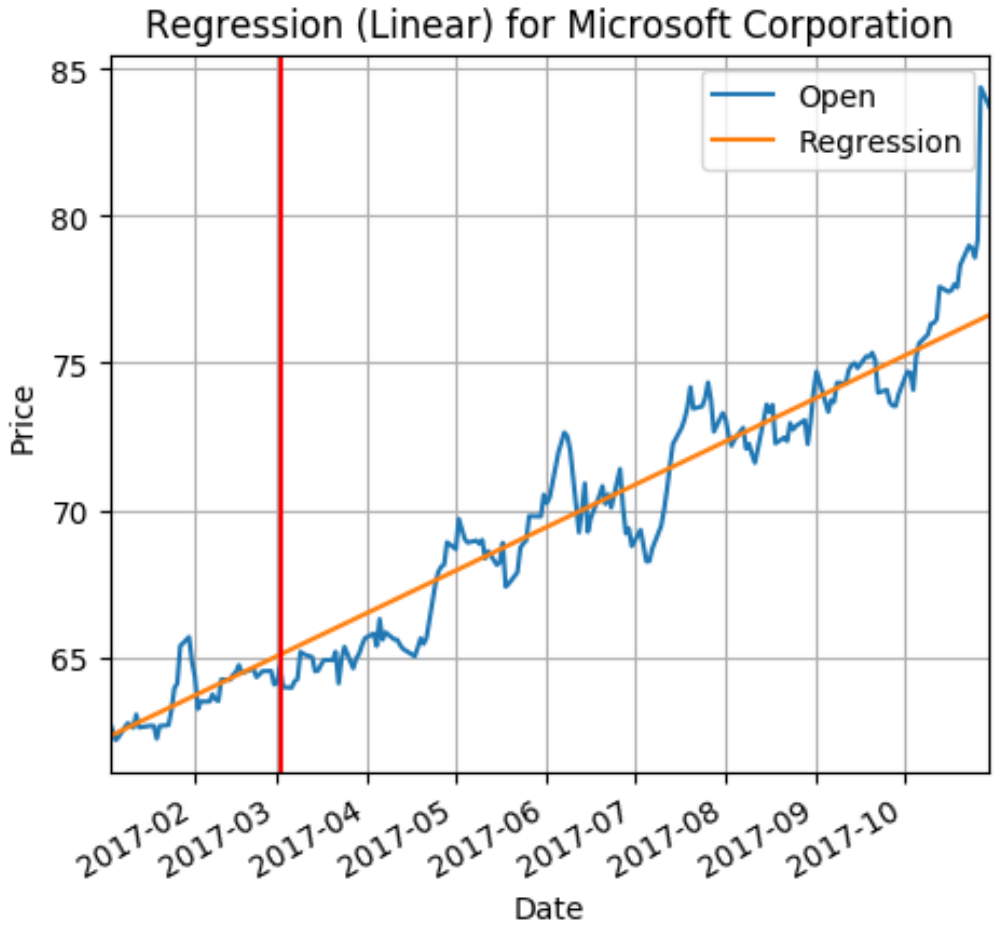
Na rysunku 4.1 przedstawiono wykres zmian cen otwarcia i zamknięcia dla podanego zakresu dat.

Ilość próbek danych uczących i testowych wynosi odpowiednio:
- 41/168 dla wartości 20% danych uczących
- 104/105 dla wartości 50% danych uczących
- 167/42 dla wartości 80% danych uczących

Na przedstawionych wykresach zaznaczona została linia podziału danych uczących i testowych i jest ona reprezentowana przez czerwoną pionową prostą.

4.2.2. Regresja liniowa

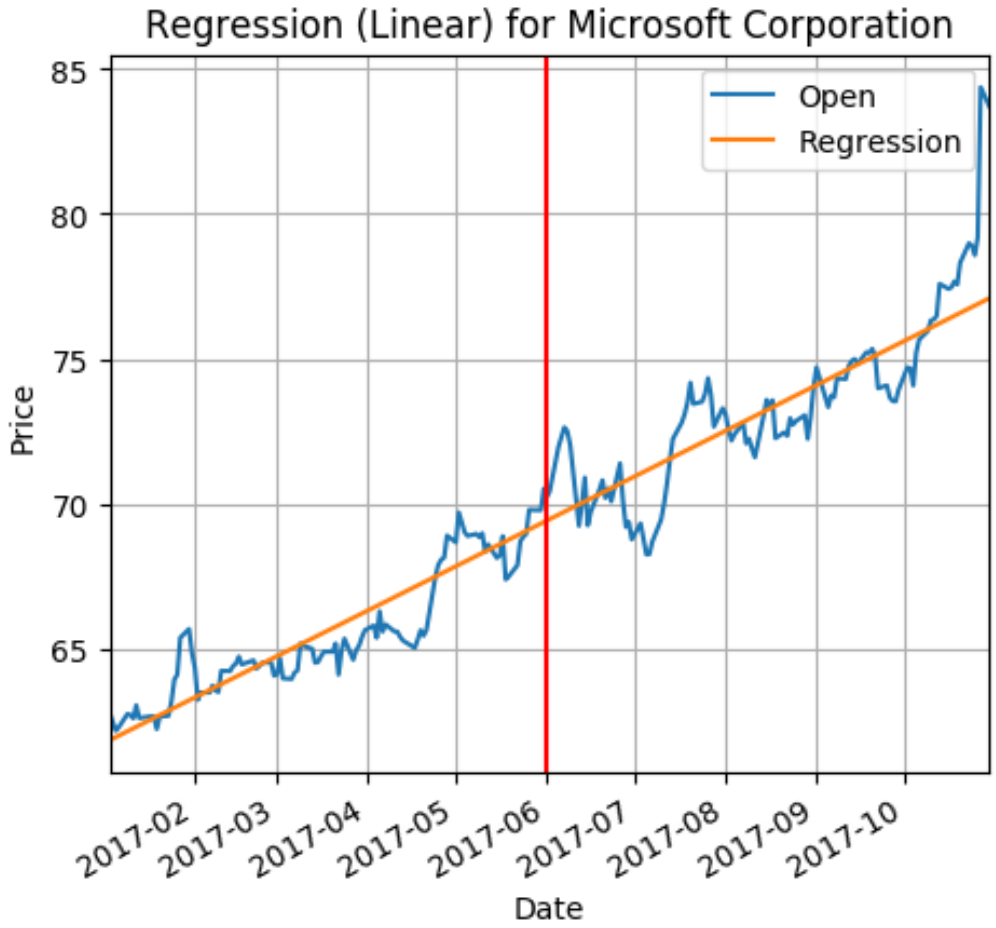
Wykres regresji liniowej dla podanego zbioru danych, przy proporcji 20% danych uczących przedstawiony jest na rysunku 4.2.



Rysunek 4.2. Wykres regresji liniowej dla 20% danych uczących, Microsoft

Widoczny jest tu trend wzrostowy, który określony na podstawie danych uczących, kontynuowany jest także dla danych testowych. Należy jednak zauważyć, że podany zbiór danych nie zawiera gwałtownych wzrostów i spadków cen w szczególności w części testowej, dzięki czemu regresja z powodzeniem przewiduje jego kontynuację.

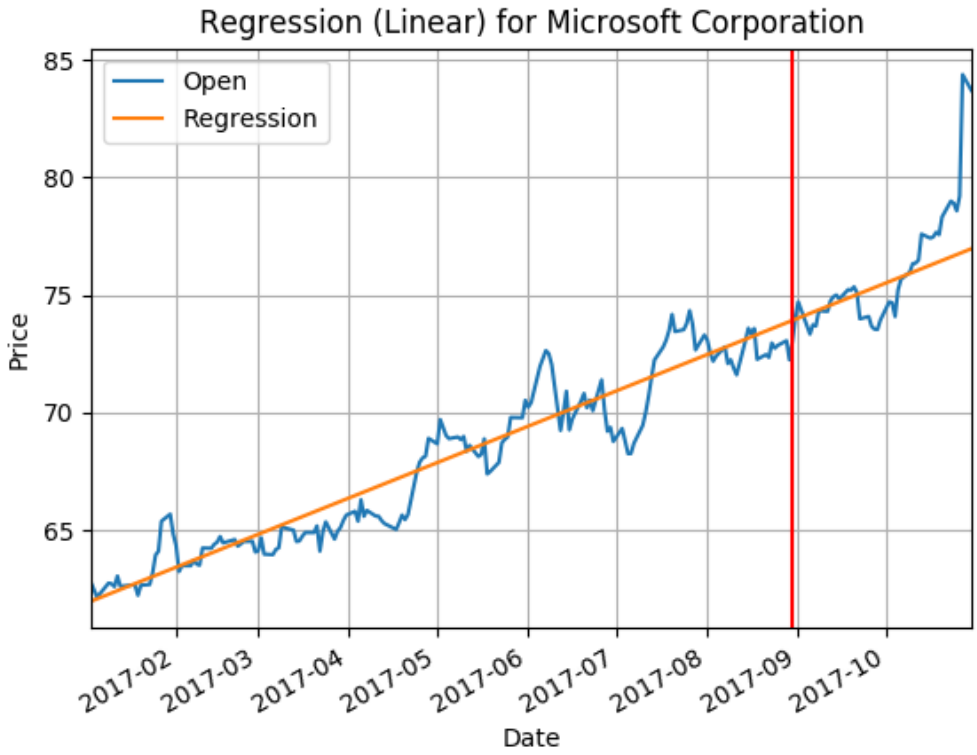
Na rysunku 4.3 przedstawiono wykres regresji liniowej dla 50% proporcji danych uczących.



Rysunek 4.3. Wykres regresji liniowej dla 50% danych uczących, Microsoft

Porównując wykresy 4.2 oraz 4.3 zauważalne jest niewielkie przesunięcie prostej regresji w kierunku niższej ceny, choć mimo tego można stwierdzić, że dla danego zbioru danych zwiększenie wartości podziału danych na testowe i uczące nie miało wpływu na wynik.

Rysunek 4.4 przedstawia wykres regresji liniowej dla 80% proporcji danych uczących.



Rysunek 4.4. Wykres regresji liniowej dla 80% danych uczących, Microsoft

W porównaniu do regresji liniowej przeprowadzonej dla 20% i 50% danych uczących, wyniki regresji przeprowadzonej dla 80% danych uczących nie różnią się wiele od pozostałych. Zauważalne jest jedynie delikatne przesunięcie prostej regresji w kierunku cen niższych, co może być spowodowane przez zmiany kierunku mniejszych trendów obecnych na wykresie.

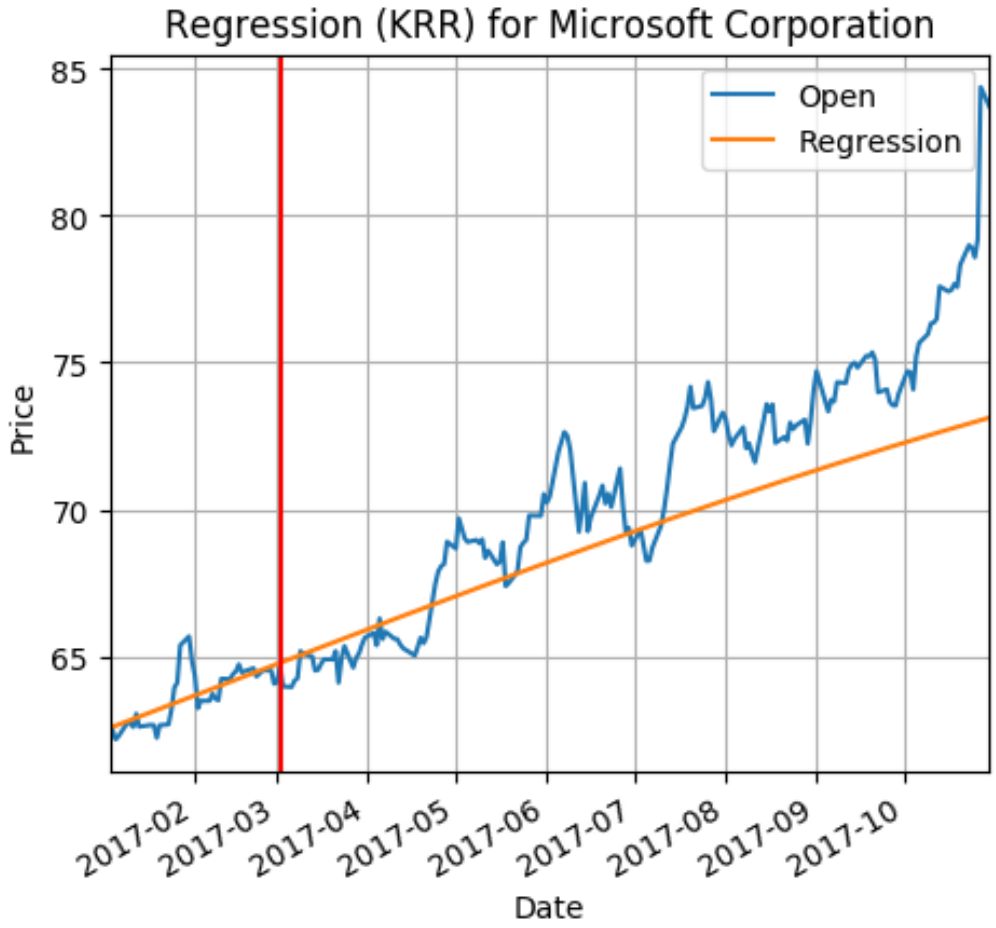
4.2.3. Regresja Grzbietowa

Rysunki 4.5, 4.6 i 4.7 przedstawiają wyniki regresji grzbietowej dla 20%, 50% i 80% użytych danych uczących.

Krzywa regresji widoczna na powyższym rysunku nie posiada dużych odchyleń, co upodabnia ją do prostej regresji liniowej. Odchylenie jest zauważalne dopiero przy końcowej części danych testowych. Fakt ten może być spowodowany zbyt małą liczbą danych uczących użytych do przeprowadzenia analizy.

W porównaniu do rysunku 4.5, powyższy wykres przedstawia wynik bardziej dokładny i dopasowany. Zaznaczony jest wyraźny trend wzrostowy z dopasowanymi krzywiznami, które trafnie pokrywają dużą część danych testowych.

Wykres przedstawiający wyniki regresji grzbietowej dla największej ilości zastoso-



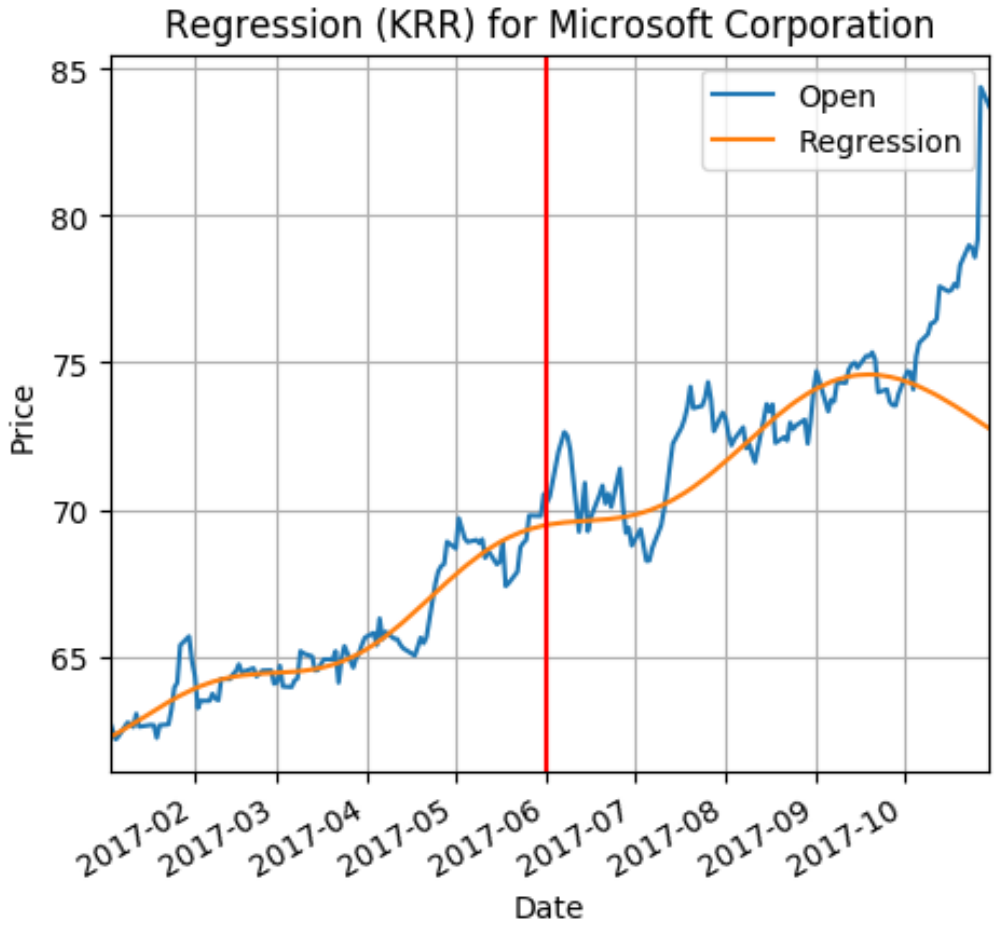
Rysunek 4.5. Wykres regresji grzbietowej dla 20% danych uczących, Microsoft

wanych danych testowych przedstawia krzywą regresji zbliżoną kształtem do prostej. Może to być spowodowane zastosowaniem zbyt dużej ilości danych uczących, które znajdują się w wyraźnym wzrostowym trendzie o niewielkich odchyleniach.

4.2.4. Regresja Wektorów Nośnych

Rysunki 4.8, 4.9 i 4.10 przedstawiają wykresy wyników regresji wektorów nośnych dla wskazanego zbioru danych, o podziale danych uczących względem danych testowych odpowiednio: 20%, 50% i 80%.

W powyższym przypadku regresja wektorów nośnych wykazuje bardzo niewielkie zdolności predykcyjne, a także niewielkie dopasowanie modelu. Krzywa regresji po osiągnięciu granicy danych uczących, znacząco odchyla się w kierunku dolnym, coraz

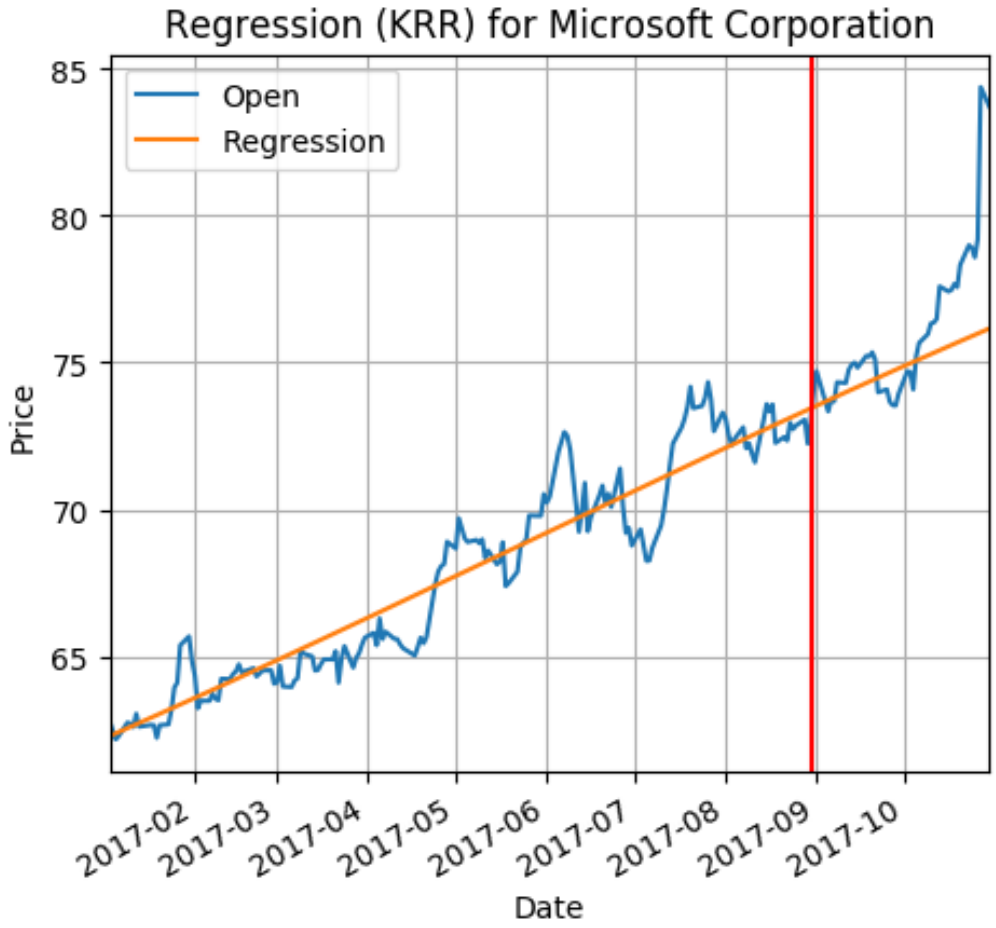


Rysunek 4.6. Wykres regresji grzbietowej dla 50% danych uczących, Microsoft

bardziej zwiększając błąd predykcji.

W odróżnieniu od krzywej regresji na rysunku 4.8, w powyższym przypadku krzywa ta poprawnie przewiduje trend wzrostowy. Jednakże odchylenie krzywej w stronę górną jest tak silne, że wraz ze wzrostem wartości na osi X wykresu, spada dopasowanie danych.

Na wykresie 4.10 można zaobserwować krzywą regresji wektorów nośnych, która przyjmuje postać prostej. Zachowanie to jest zbieżne z zaobserwowanym na wykresach regresji grzbietowej, co może prowadzić do umocnienia wniosku o użytej zbyt dużej ilości danych uczących przy fakcie, iż dane te reprezentują silny trend wzrostowy.



Rysunek 4.7. Wykres regresji grzbietowej dla 80% danych uczących, Microsoft

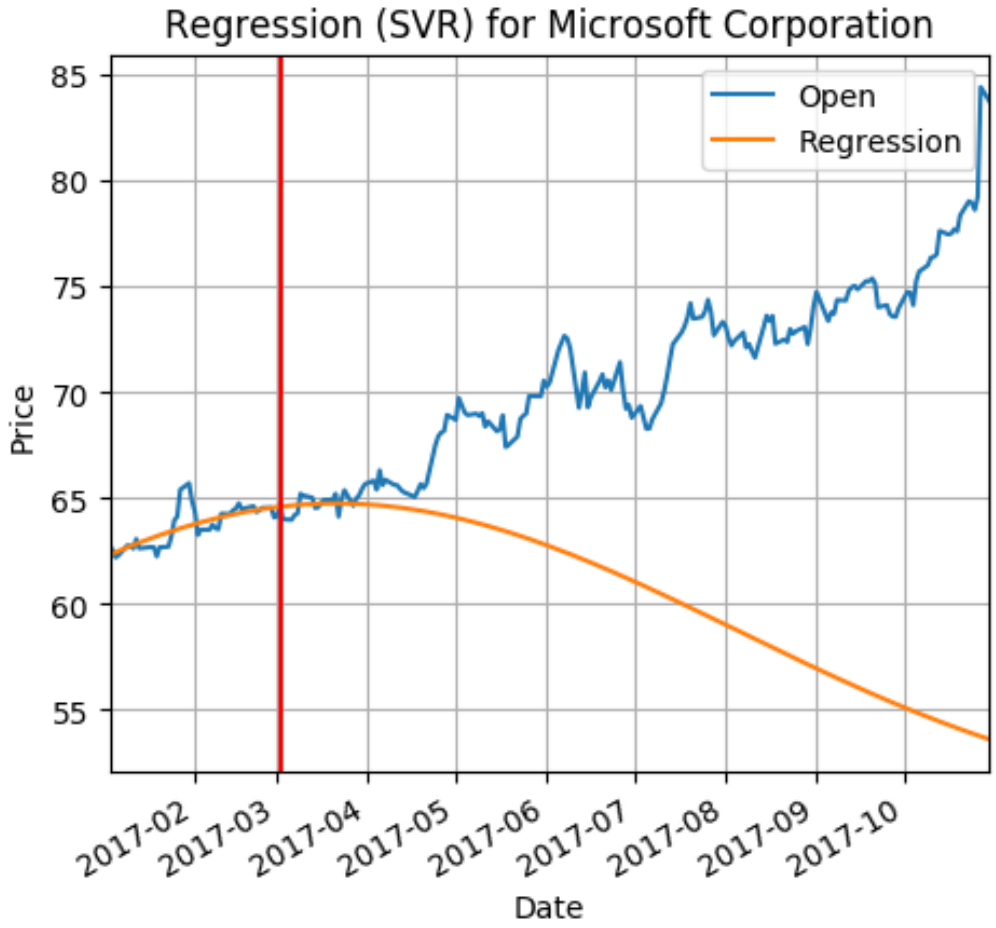
4.2.5. Regresja Procesu Gaussa

Rysunki 4.11, 4.12, 4.13 przedstawiają wykresy wyników analiz regresji procesu Gaussa dla odpowiednio: 20%, 50% i 80% zastosowanych danych uczących.

Na powyższym wykresie możemy zaobserwować zachowanie się krzywej regresji, które jest bardzo podobne do tej z rysunku 4.8. Spowodowane to może być zbyt małą ilością użytych danych uczących, które charakteryzują się niewielką zmiennością.

Wykres na rysunku 4.12 powtarza tendencję z wykresu na rysunku 4.11. Trend wzrostowy jest poprawnie przewidywany jedynie dla początkowych wartości danych testowych, a następnie występuje odchylenie krzywej regresji w dół.

Krzywa regresji na powyższym wykresie w odróżnieniu od pozostałych nieliniowych metod regresji, nie wykazuje tendencji do zbliżania się kształtem do prostej.



Rysunek 4.8. Wykres regresji wektorów nośnych dla 20% danych uczących, Microsoft

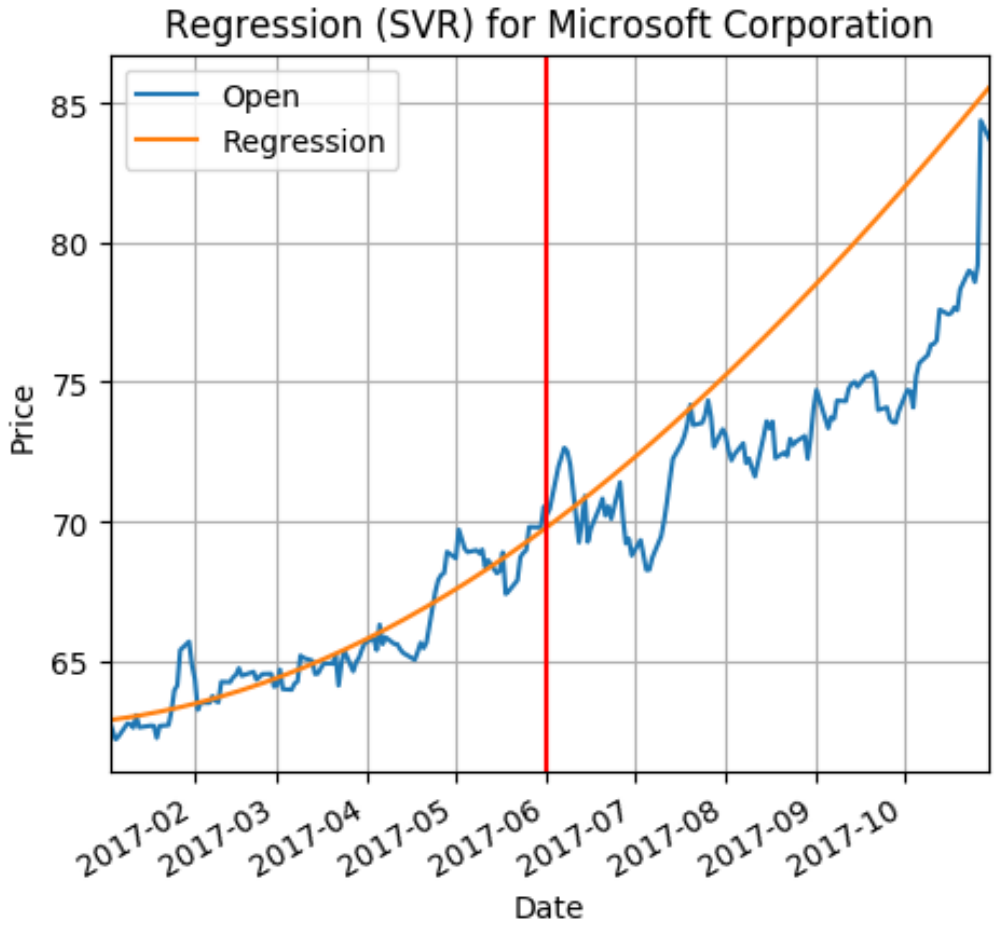
Jednak predykcja dla zbioru testowego, oraz trend jaki wskazuje krzywa, gorzej odzwierciedlają rzeczywiste dane.

4.2.6. Podsumowanie

Podczas dokonanych analiz zebrano dane odnośnie wartości średniego błędu kwadratowego każdej z nich, a także wyniku predykcji. Przedstawiono je na wykresach znajdujących się na rysunkach 4.14 i 4.15.

Z powyższych wykresów wynika, iż średni błąd kwadratowy rośnie wraz ze wzrostem ilości danych uczących zastosowanych w analizie. Wynik predycji natomiast odznacza się niewielkim wzrostem wraz ze zwiększaną ilością danych uczących.

Dla 20% zastosowanych danych uczących najmniejszy błąd osiągnęła regresja wek-

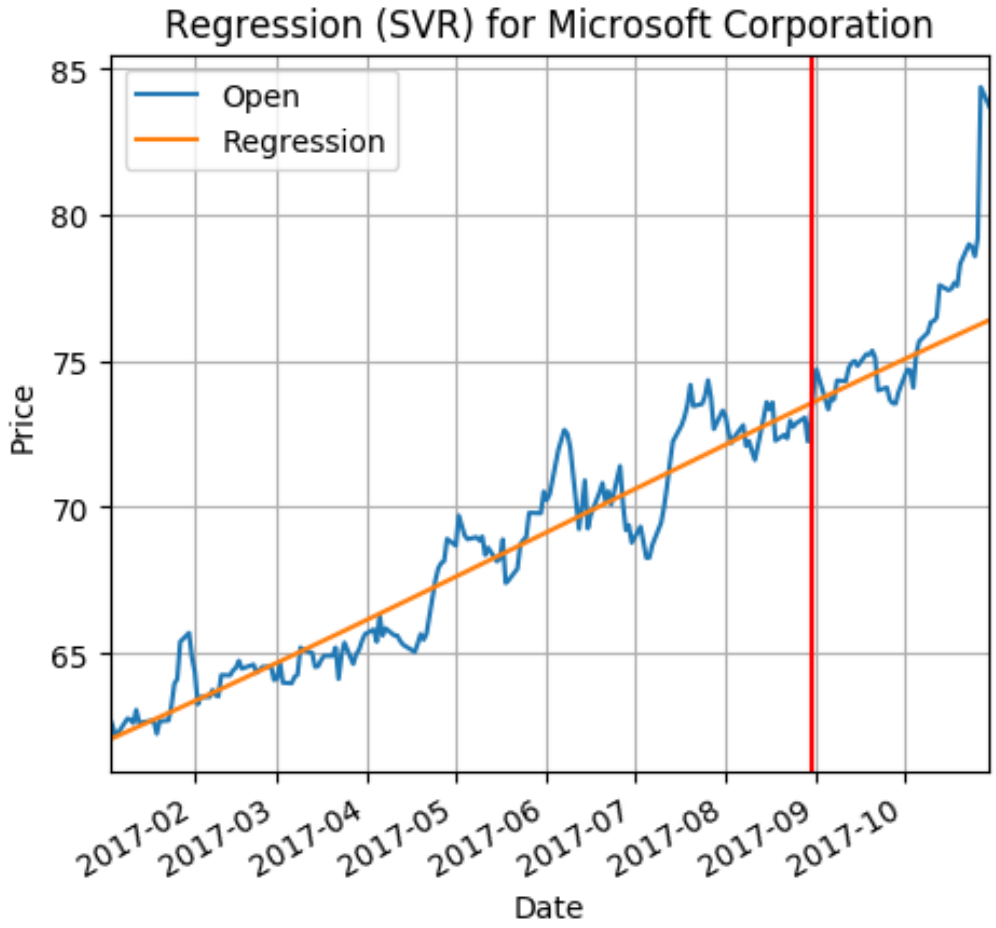


Rysunek 4.9. Wykres regresji wektorów nośnych dla 50% danych uczących, Microsoft

torów nośnych, zaś największy - liniowa. W przypadku wyniku predykcji jest natomiast odwrotnie: najlepszą wartość osiągnęła regresja liniowa, a najgorszą regresja wektorów nośnych.

Dla 50% zastosowanych danych uczących najmniejszym błędem kwadratowym charakteryzuje się regresja procesu Gaussa, a najwyższym regresja wektorów nośnych. Powtórzona zostaje zależność zidentyfikowana w przypadku 20% danych uczących: wynik predykcji oceniany najlepiej należy do regresji wektorów nośnych, natomiast najgorzej do regrtesji procesu Gaussa

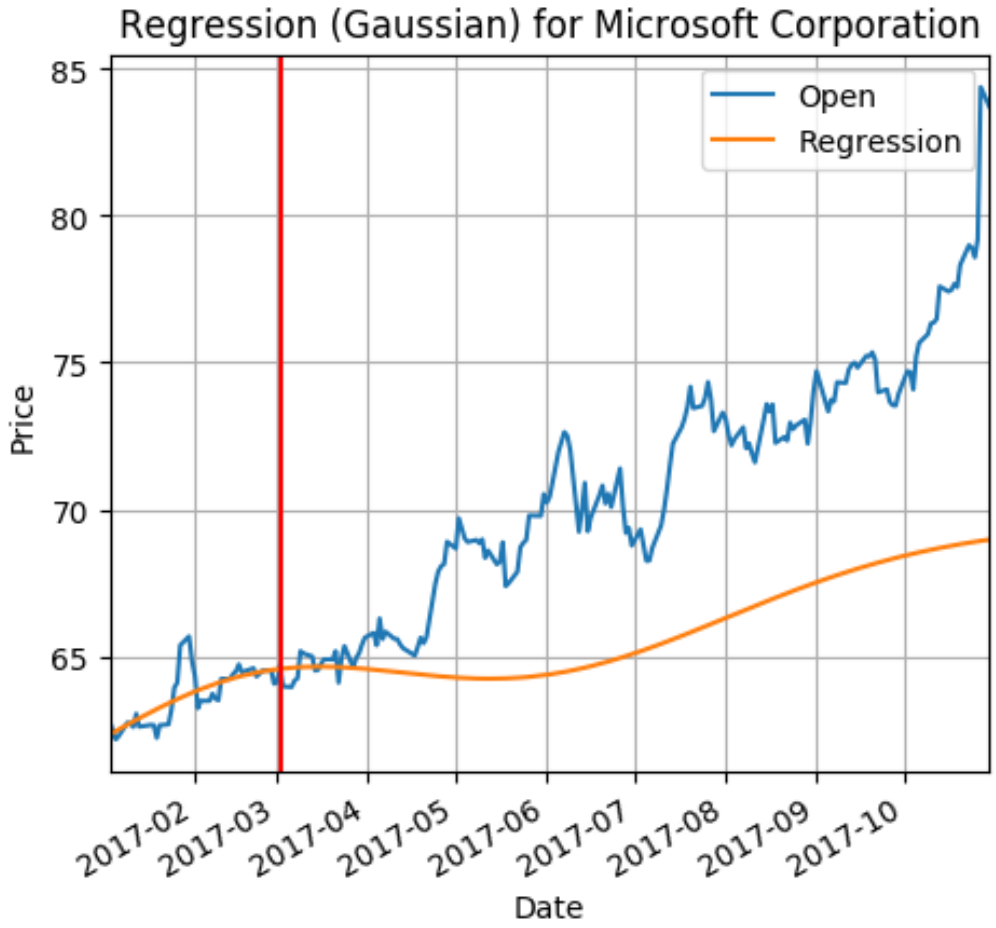
W przypadku 80% zastosowanych danych testowych regresja procesu Gaussa osiągnęła najniższy wynik średniego błędu kwadratowego. Pozostałe modele regresji charakteryzują się zbliżonym wynikiem. Wynik predykcji dla każdego z modeli jest w tym przypadku bardzo zbliżony.



Rysunek 4.10. Wykres regresji wektorów nośnych dla 80% danych uczących, Microsoft

Na podstawie zgromadzonych wykresów, a także powyższych danych, można określić następujące wnioski:

- trafne określenie trendu dla danych testowych zaobserwowano w przypadkach regresji liniowej, wektorów nośnych (80% danych uczących) oraz grzbietowej (20% i 80% danych uczących)
- krzywe regresji, które nie identyfikowały poprawnie trendu dla danych testowych należą do regresji wektorów nośnych (20% danych testowych) oraz procesu Gaussa
- w przypadku regresji grzbietowej (50% danych uczących), zaobserwowano dobre dopasowanie i potwierdzenie trendu dla około 80% danych testowych, po czym następowała zmiana nachylenia krzywej



Rysunek 4.11. Wykres regresji procesu Gaussa dla 20% danych uczących, Microsoft

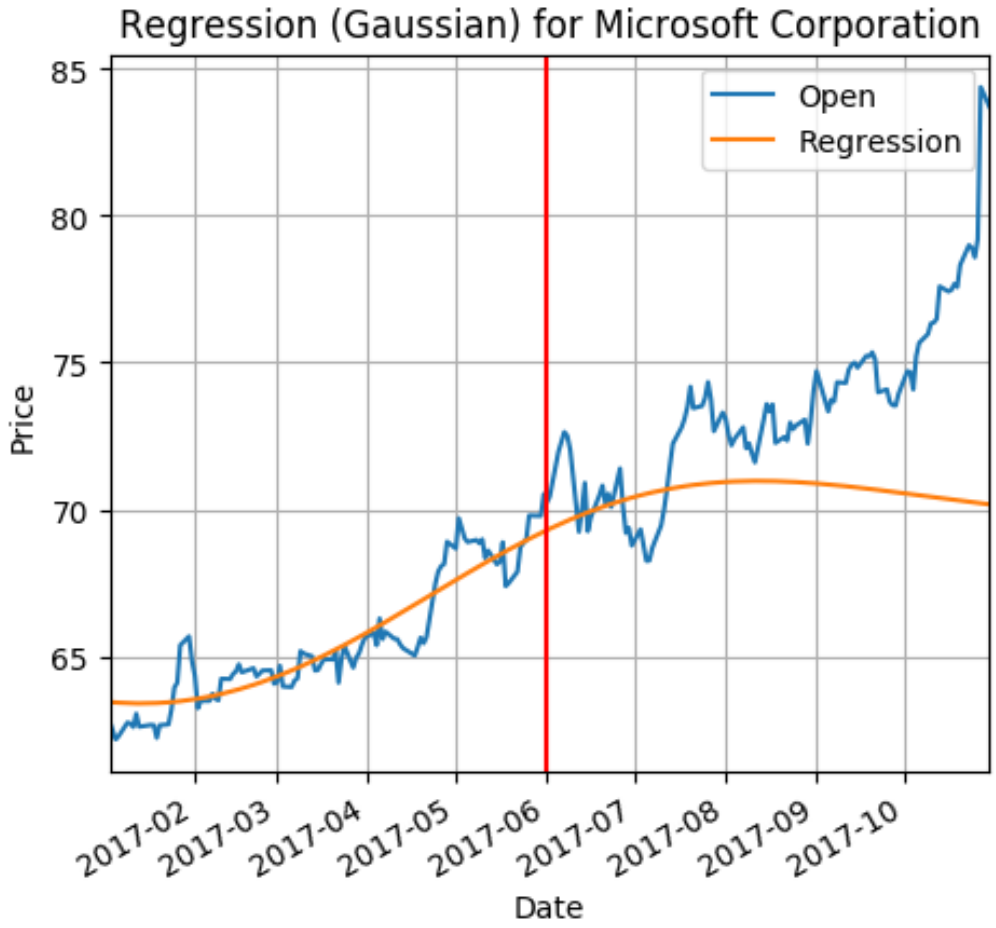
4.3. Testy zbioru danych: Intel

4.3.1. Informacje ogólne

Zakres danych użytych do testów wynosi 81 próbek, w przedziale dat od 2017-01-01 do 2017-14-30 z krokiem wynoszącym jeden dzień.
Zakres ten charakteryzuje się mniejszą ilością danych o większej zmienności, niż zakres poprzedni.

Na rysunku 4.16 przedstawiono wykres zmian cen otwarcia i zamknięcia dla podanego zakresu dat.

Ilość próbek danych uczących i testowych wynosi odpowiednio:
- 16/65 dla wartości 20% danych uczących
- 40/41 dla wartości 50% danych uczących
- 64/17 dla wartości 80% danych uczących



Rysunek 4.12. Wykres regresji procesu Gaussa dla 50% danych uczących, Microsoft

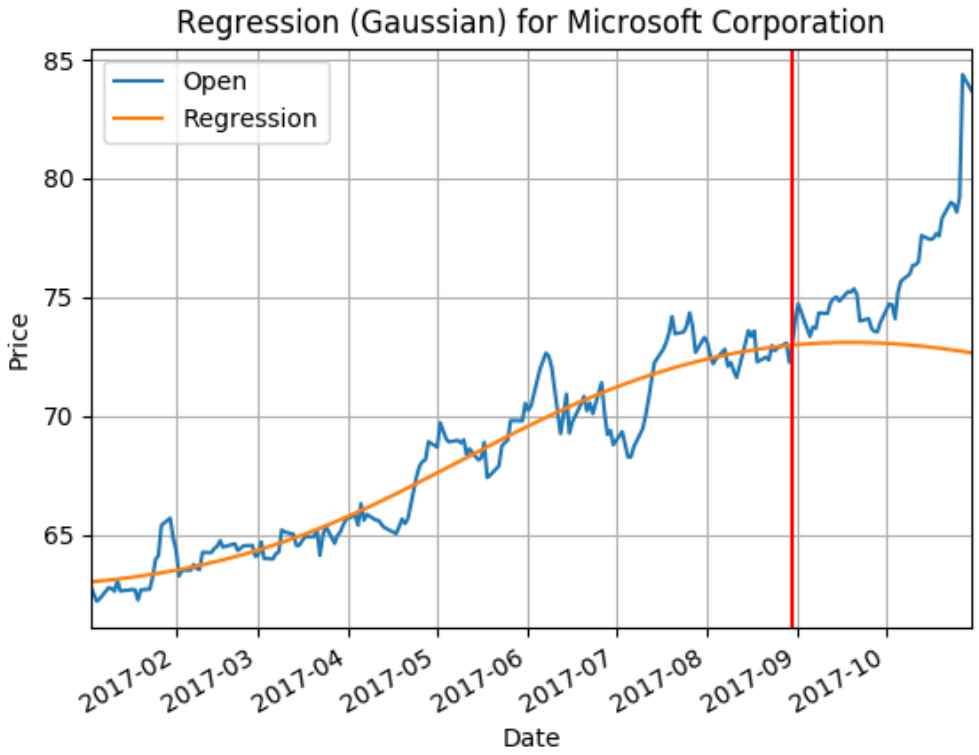
4.3.2. Regresja liniowa

Na wykresach 4.17, 4.18 oraz 4.19 przedstawiono wyniko regresji liniowej o udziale danych uczących odpowiednio: 20%, 50% i 80%.

Z powyższego wykresu wynika, iż poprzez predykcję poprawnie zidentyfikowano trend spadkowy.

W porównaniu do wykresu na rysunku 4.18, na powyższym wykresie zaobserwować można prostą regresji identyfikującą trend spadkowy, lecz o mniejszym nachyleniu niż poprzednio. Może być to spowodowane lepszym dopasowaniem modelu, lub też zwiększoną liczbą danych uczących.

Porównując wszystkie trzy powyższe analizy regresji liniowej, można zaobserwować zależność spadku nachylenia prostej regresji od ilości zastosowanych danych uczących.



Rysunek 4.13. Wykres regresji procesu Gaussa dla 80% danych uczących, Microsoft

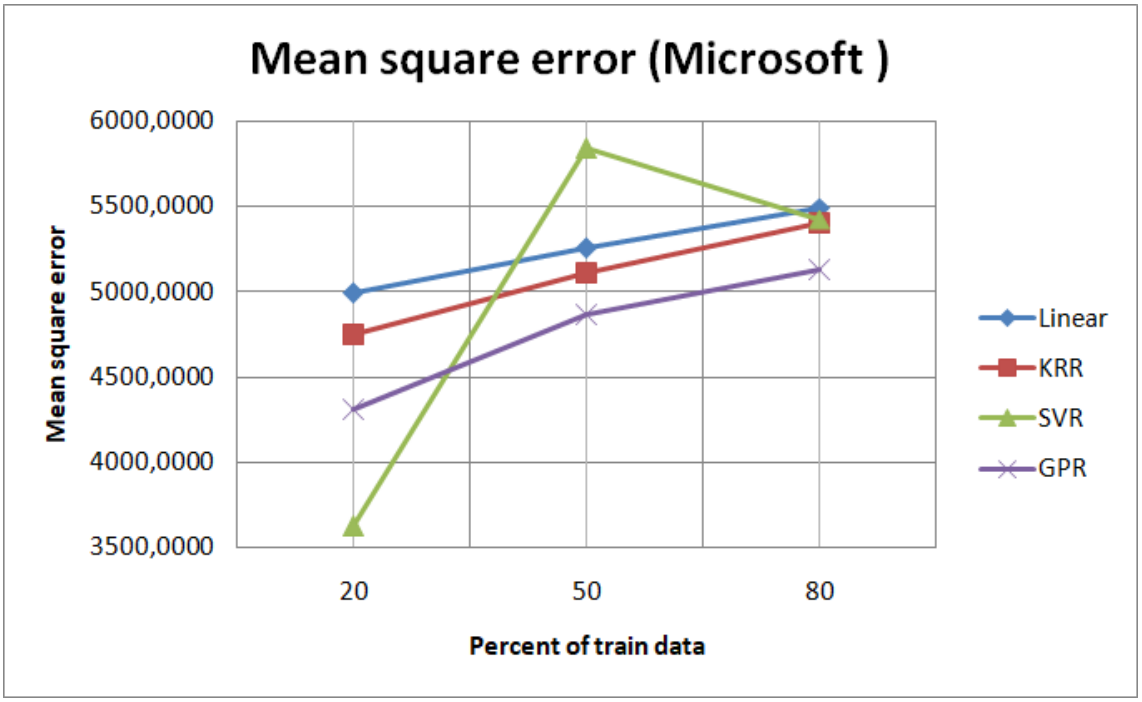
4.3.3. Regresja Grzbietowa

Na rysunkach 4.20, 4.21 i 4.22 przedstawiono wyniko analizy regresji grzbietowej, przy zastosowaniu 20%, 50% i 80% danych uczących.

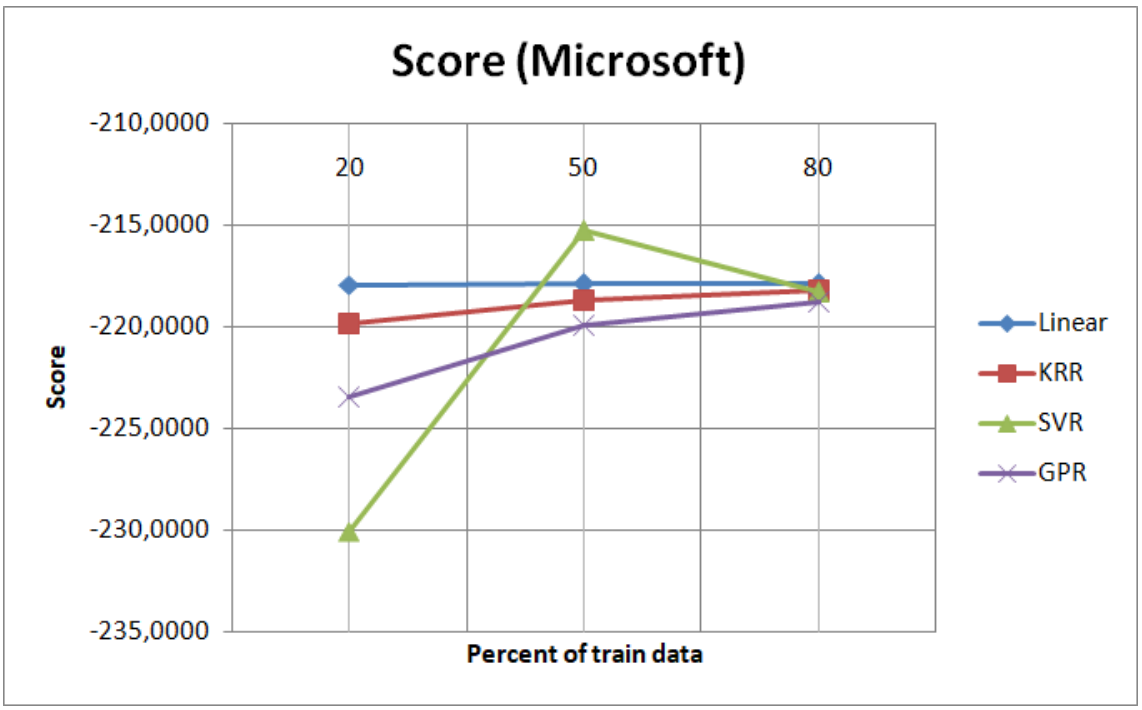
Powyższy wykres wykazuje błędną predykcję i rozpoznanie trendu, które mogą być skutkiem zbyt małej ilości danych uczących. Poza tym, dane zakres danych testowych obejmuje jedynie mniejszy trend wzrostowy, przez co utrudniona lub niemożliwa jest identyfikacja szerszego trendu spadkowego.

Wykres 4.21 przedstawia krzywą regresji, która charakteryzuje się dobrym dopasowaniem w części danych uczących, lecz w części danych testowych staje się ona niemal linią prostą. Może być to ponownie spowodowane nietrafnie dobranym zakresem danych uczących, ponieważ można w nim zaobserwować zarówno duży wzrost i duży spadek, co w połączeniu z małą ilością próbek danych może czynić zbiór nierepreprezentacyjnym.

Spośród trzech przedstawionych analiz regresji grzbietowej, wykres na rysunku 4.22 zdecydowanie przedstawia krzywą regresji najlepiej przedstawiającą trend. Należy jednak zauważyć, iż fragment przedstawiający dane testowe jest relatywnie nie-

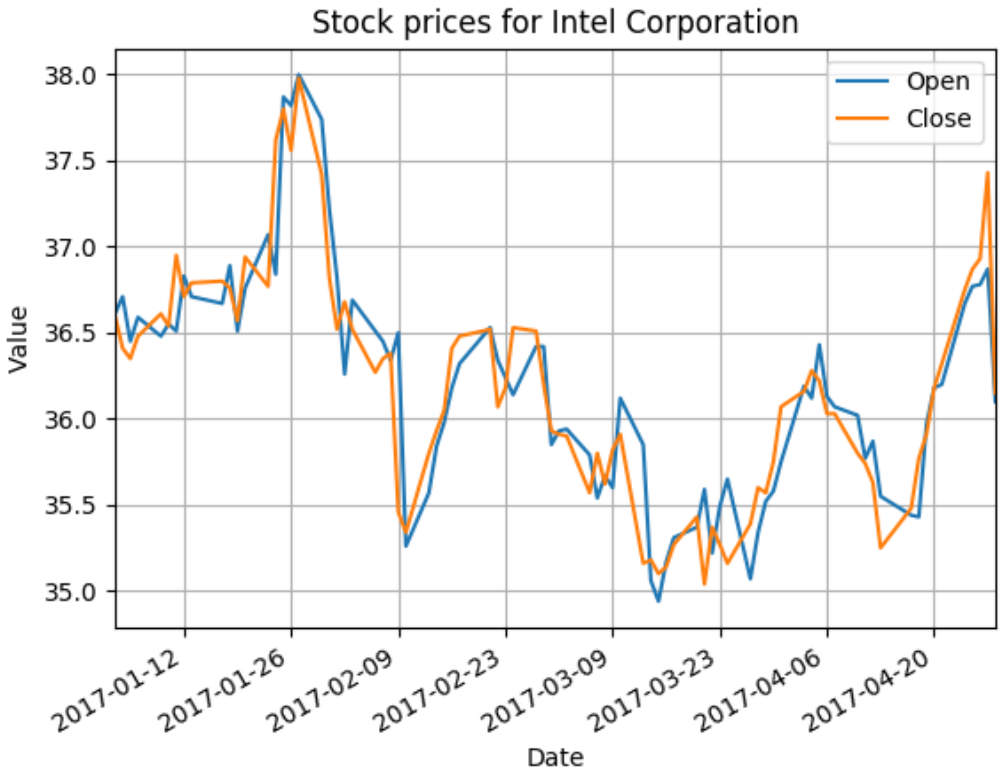


Rysunek 4.14. Wykres zmian wartości średniego błędu kwadratowego, Microsoft



Rysunek 4.15. Wykres zmian wartości wyników dopasowania modelu, Microsoft

wielki, a przewidywane wartości nie pokrywają się w żadnym punkcie z wartościami testowymi.



Rysunek 4.16. Wykres cen otwarcia i zamknięcia firmy Intel

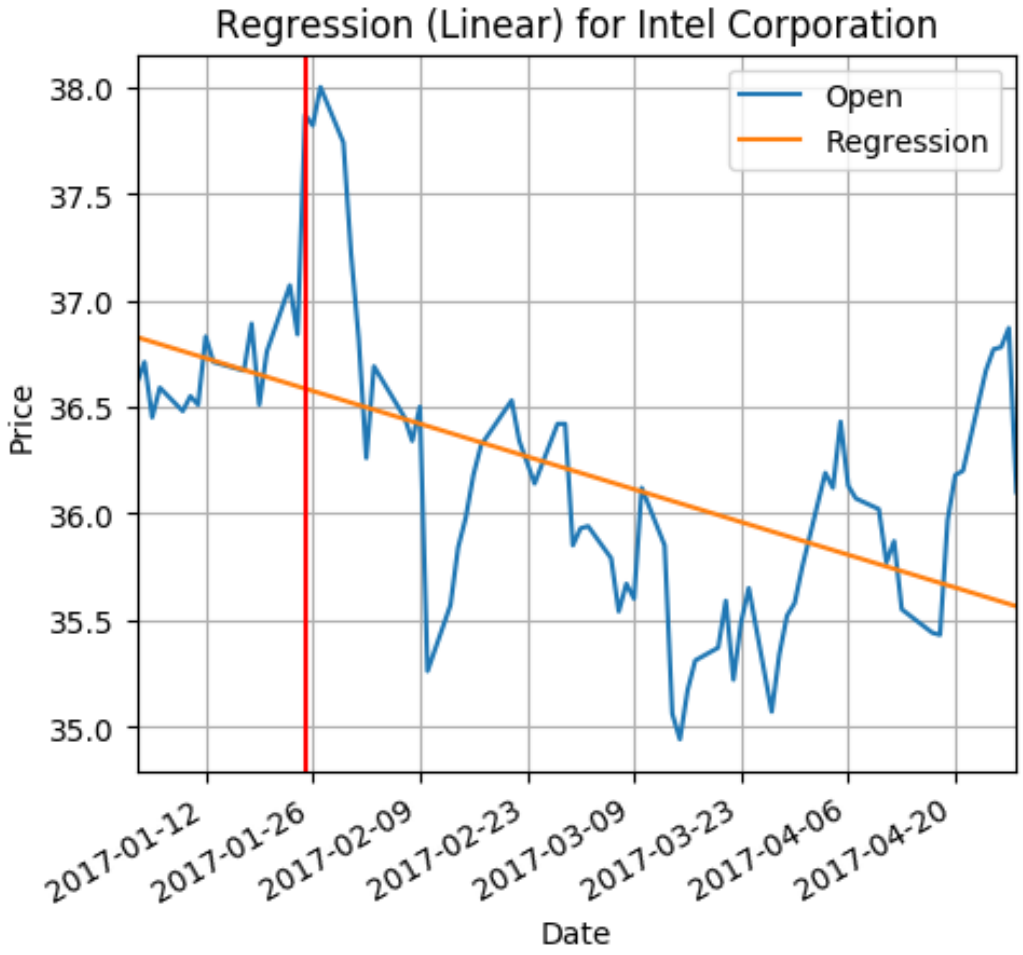
4.3.4. Regresja Wektorów Nośnych

Rysunki 4.23, 4.24 i 4.25 przedstawiają wykresy analizy regresji wektorów nośnych dla proporcji danych uczących wynoszących odpowiednio: 20%, 50% i 80%.

Powyższy wykres przedstawia podobną zależność, co wykres analizy regresji grzbietowej dla tych samych ilości danych uczących. Podtrzymany zostaje więc wniosek, iż niepoprawna identyfikacja trendu spowodowana może być złym dobraniem danych uczących, reprezentujących jedynie mniejszy trend wzrostowy.

Wykres na rysunku 4.24 wykazuje poprawne dopasowanie modelu regresji w części danych uczących, lecz w części danych testowych błędnie identyfikuje trend, a krzywa regresji przybiera postać prostej. Może to być spowodowane błędnym dobraniem parametrów obiektu *SVR()*.

Podobnie jak w przypadku wykresu 4.22 regresji grzbietowej, także i w powyższym przypadku dopasowanie modelu i identyfikacja trendu spadkowego są zauważalne.



Rysunek 4.17. Wykres regresji liniowej dla 20% danych uczących, Intel

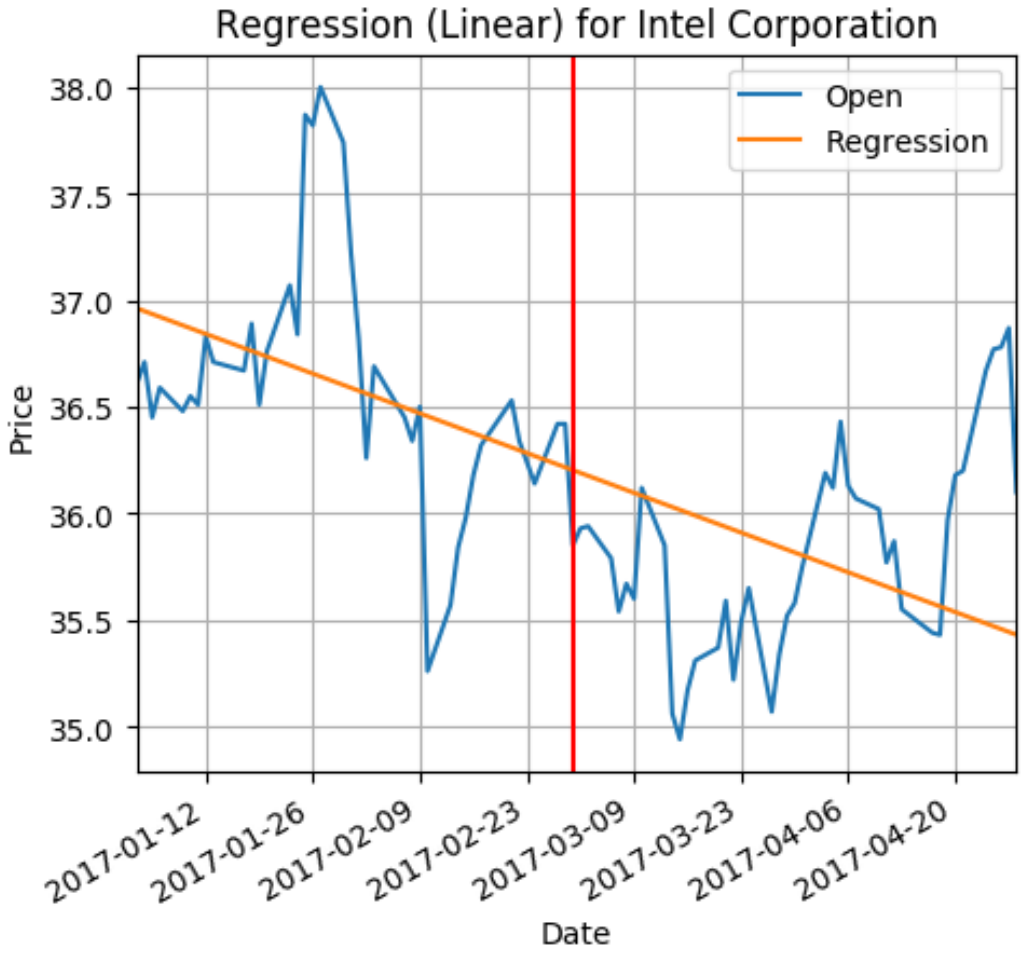
4.3.5. Regresja Procesu Gaussa

Na rysunkach 4.26, 4.27 i 4.28 przedstawiono wykresy analizy regresji procesu Gaussa dla następujących ilości danych testowych: 20%, 50% i 80%.

W powyższym przypadku krzywa regresji poprawnie identyfikuje trend spadkowy w części testowej, jednakże niewielkie nachylenie krzywej nie oddaje do końca charakteru spadkowego danych.

Na rysunku 4.27 można zaobserwować krzywą regresji poprawnie opisującą i przewidującą przebieg trendu. W części danych testowych zauważalny jest nie tylko delikatny spadek, lecz również wzrost dla końcowych wartości danych, co zgadza się z wartościami testowymi.

W porównaniu do poprzednich wyników regresji procesu Gaussa, powyższy wy-



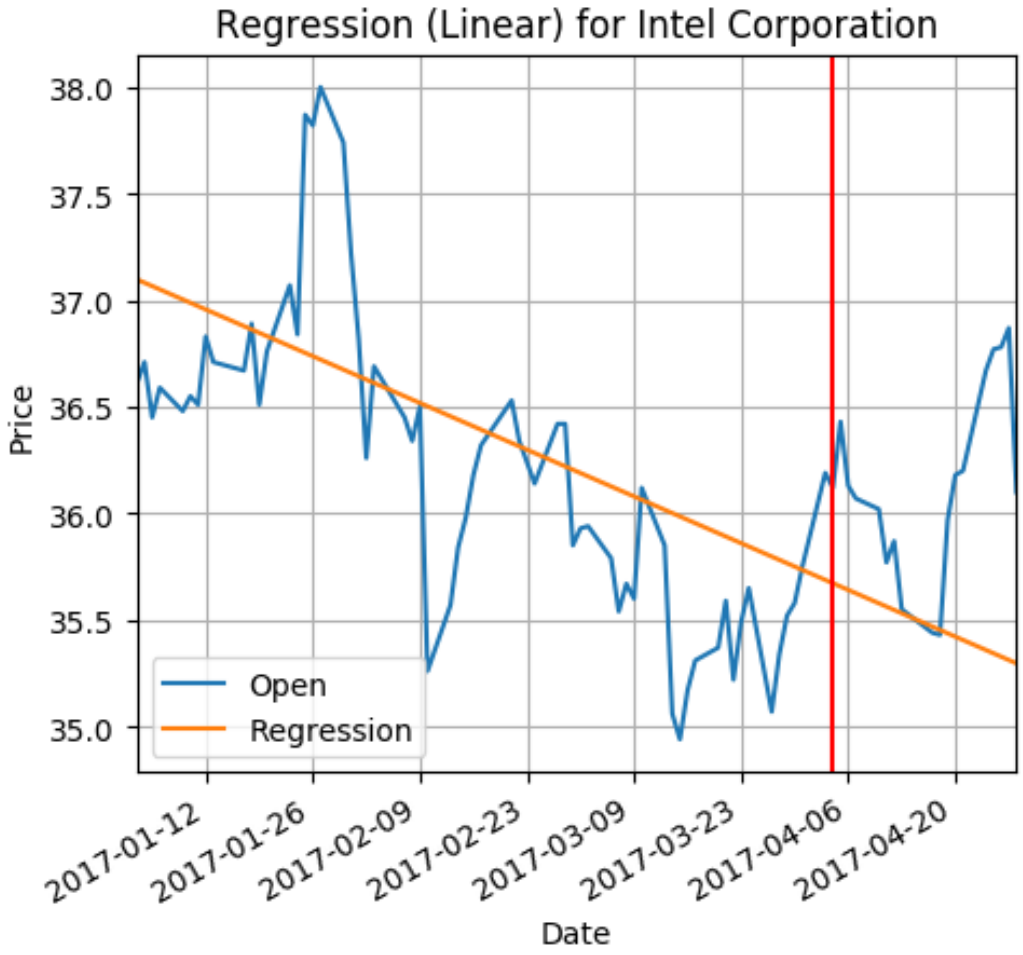
Rysunek 4.18. Wykres regresji liniowej dla 50% danych uczących, Intel

kres potwierdza wniosek, iż wraz ze wzrostem ilości danych testowych dla tego zbioru danych, rośnie dopasowanie modelu oraz jakość predykcji.

4.3.6. Podsumowanie

Dane dotyczące średniego błędu kwadratowego oraz wyników predykcji modeli regresji są przedstawione na wykresach 4.29 oraz 4.30.

Z powyższych danych wynika, iż dla ilości danych uczących wynoszącej 20% najwyższy średni błąd kwadratowy uzyskały modele regresji wektorów nośnych oraz grzbietowa, najniższy zaś model regresji liniowej. Dla ilości danych uczących wynoszącej 50% najwyższa wartość błędu odnosi się do modelu regresji wektorów nośnych, a najniższa do modelu regresji liniowej. Dla 80% danych uczących natomiast, wszystkie modele regresji osiągnęły podobną, niską wartość, z wyjątkiem modelu regresji procesu



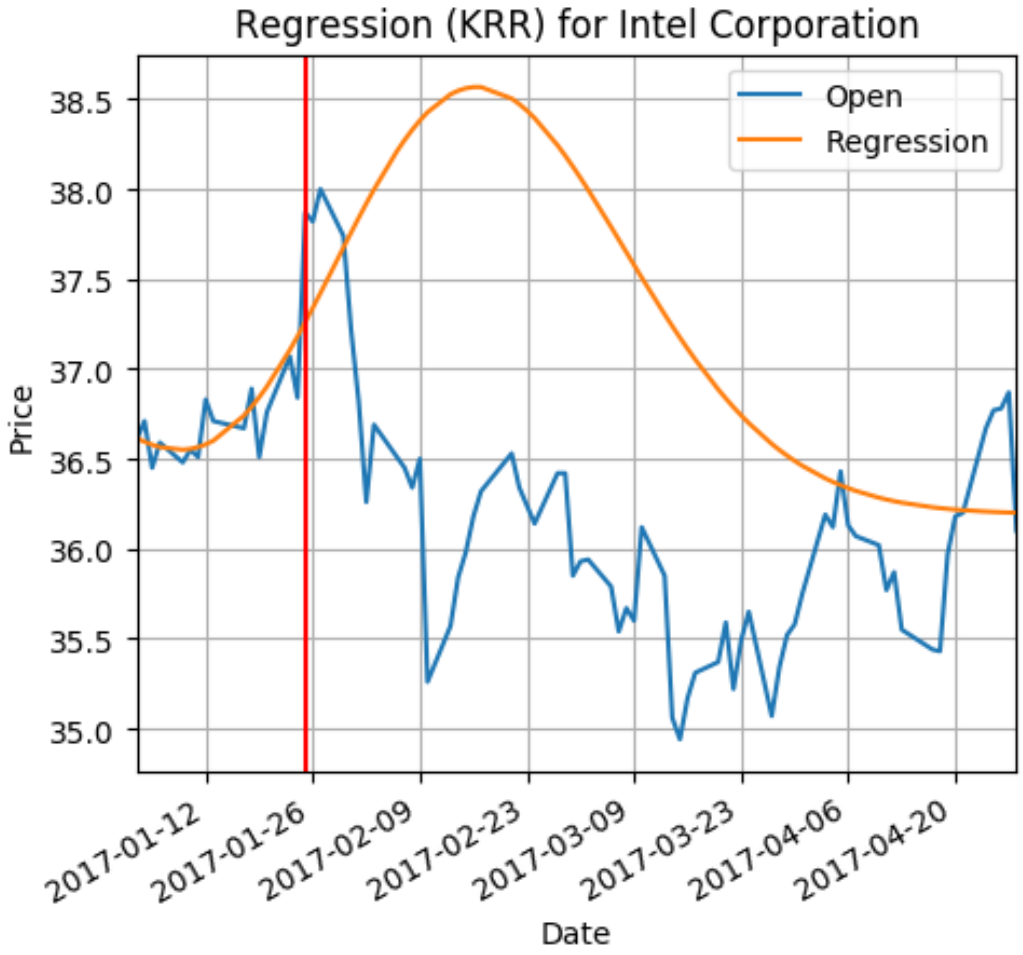
Rysunek 4.19. Wykres regresji liniowej dla 80% danych uczących, Intel

Gaussa, który osiągnął wartość najwyższą.

W odróżnieniu od poprzedniego testowanego zakresu danych, w tym przypadku zaobserwować można tendencję spadku wartości średniego kwadratu błędów wraz ze wzrostem procentowej ilości użytych danych uczących.

Z wykresu 4.30 wynika, iż wraz ze wzrostem ilości użytych danych uczących spada wartość dopasowania każdego modelu, z wyjątkiem modelu liniowego, który przedstawia za każdym razem podobną wartość. Może być to spowodowane faktem, iż wartość ta jest liczona jedynie dla danych testowych, więc przy zmniejszaniu ich ilości obserwujemy spadek dopasowania.

Analiza wykresów pozwala wnioskować, iż trafne określenie trendu można dostrzec na wykresach regresji procesu Gaussa, liniowej, grzbietowej (80% danych uczących) oraz wektorów nośnych (80% danych uczących). Natomiast niedokładna predykcja

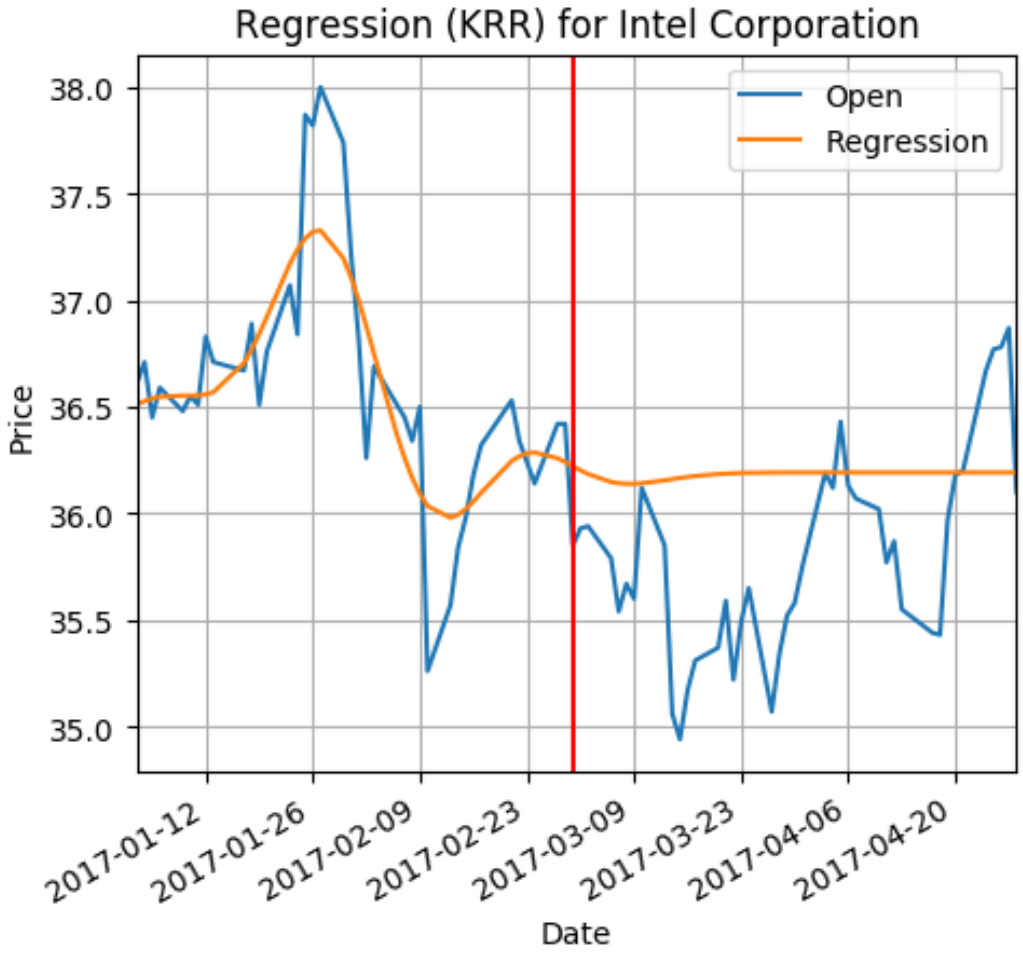


Rysunek 4.20. Wykres regresji grzbietowej dla 20% danych uczących, Intel

trendu widoczna jest na wykresach regresji grzbietowej (20% i 50% danych uczących) oraz wektorów nośnych (20% i 50% danych uczących).

4.4. Wnioski

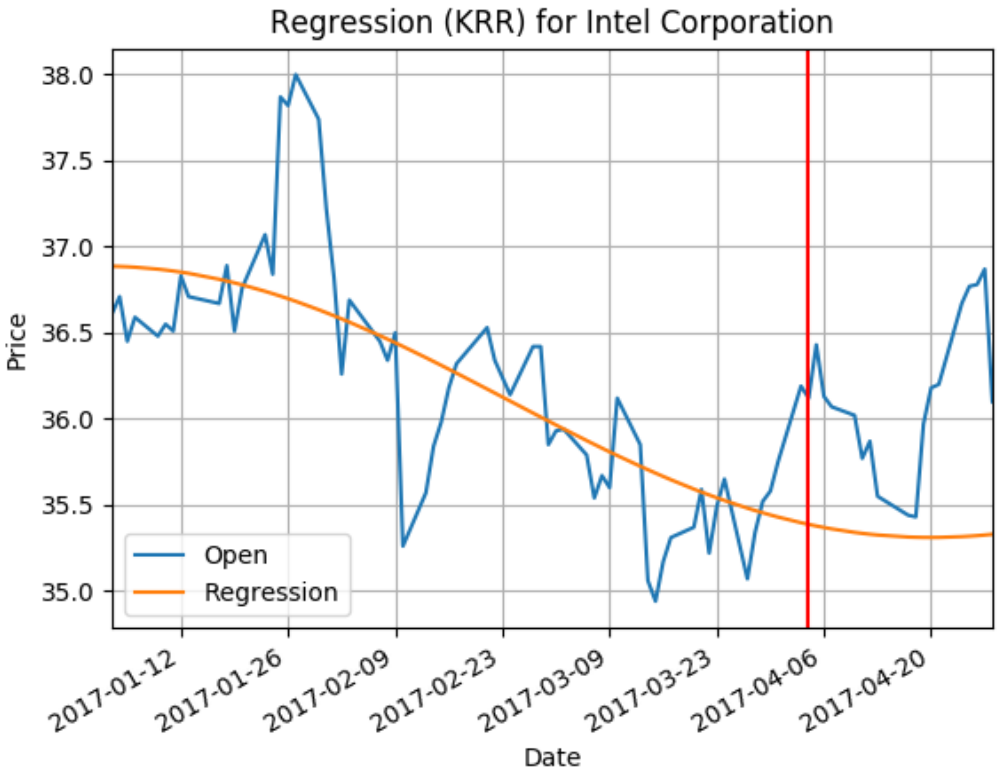
1. Dokładność predykcji modelu regresji liniowej zmienia się nieznacznie w zależności od zmiany proporcji ilości danych uczących względem danych testowych.
2. Przedstawione modele regresji nieliniowych wykazują dużą wrażliwość na wysoką zmienność danych poddawanych analizie, przez co w celu osiągnięcia bardziej dokładnych wyników predykcji niezbędne jest znalezienie właściwych zakresów parametrów podawanych jako argumenty do obiektów tych regresji.
3. Wartości średniego kwadratu błędów oraz wyników predykcji (dopasowania modelu) wykazują zależność od całkowitej ilości analizowanych danych. Wraz ze wzro-



Rysunek 4.21. Wykres regresji grzbietowej dla 50% danych uczących, Intel

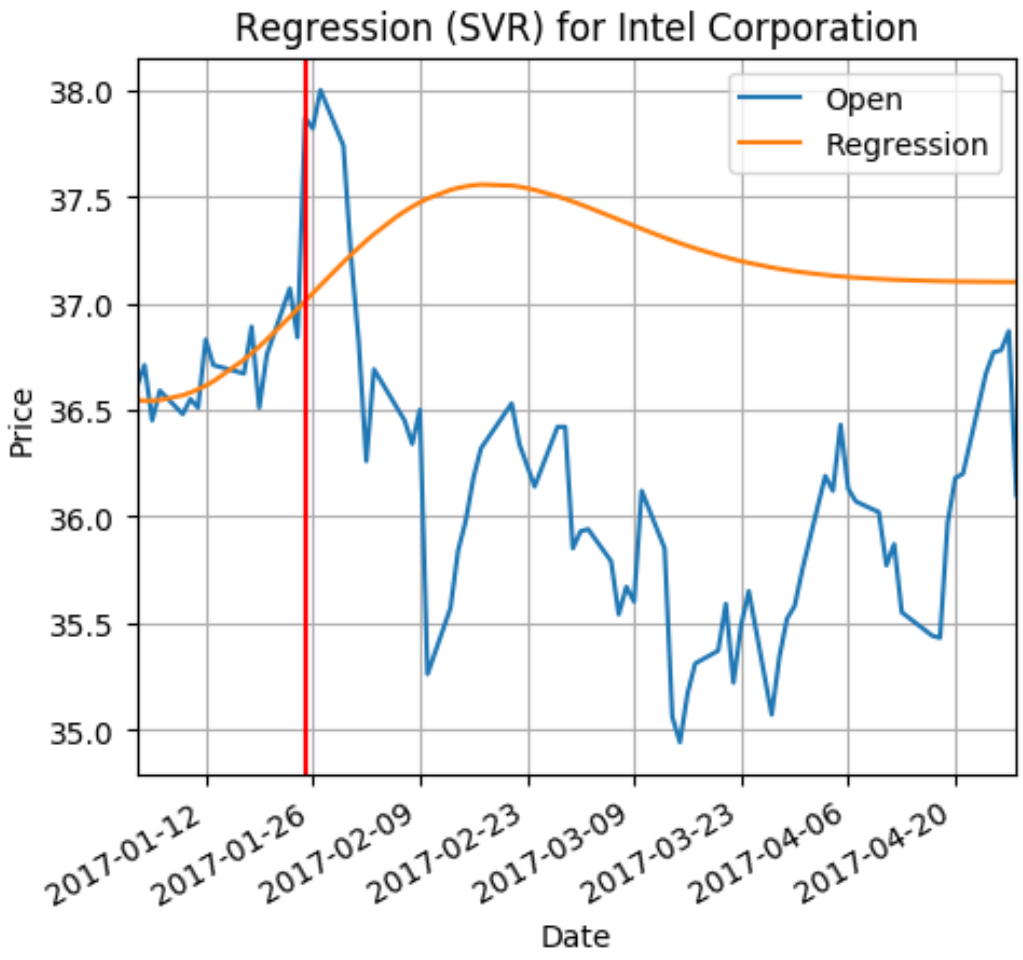
stem ich ilości zwiększają się wartości błędów, a wyniki predykcji ulegają polepszeniu.

4. Dobre właściwości określania trendu dla obydwu testowanych zakresów danych wykazały: regresja liniowa, wektorów nośnych (przy 80% danych uczących) oraz grzbietowa (przy 80% danych uczących).
5. Regresja procesu Gaussa w przypadku dużego zbioru danych nie wykazała dobrych zdolności predykcyjnych, natomiast w przypadku małego zbioru można ją uznać za najlepszą metodę regresji dla podanych warunków.
6. Jakość predykcji za pomocą testowanych metod w dużym stopniu zależy od struktury danych użytych do analizy. Jednocześnie najbardziej odporną na ten czynnik metodą okazała się regresja liniowa.
7. Wraz ze wzrostem ilości użytych do analizy danych uczących, w większości przypadków wzrasta trafność predykcji lub trafność określenia trendu.
8. Pakiet *Scikit-learn* i udostępniane przez niego metody regresji są przydatne w

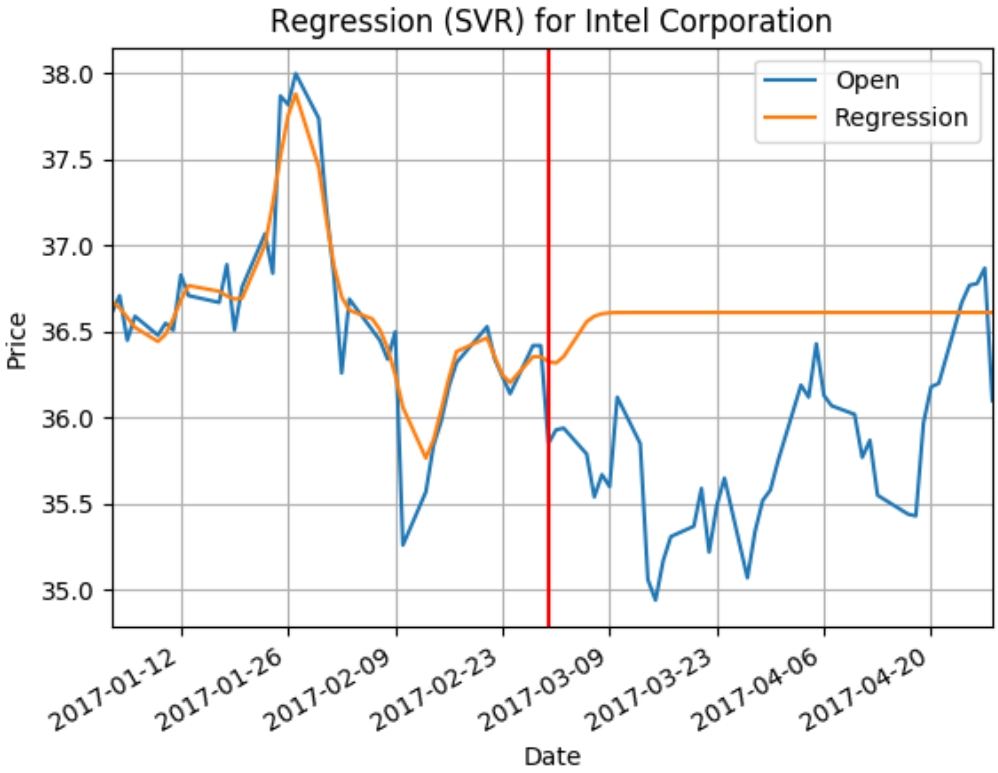


Rysunek 4.22. Wykres regresji grzbietowej dla 80% danych uczących, Intel

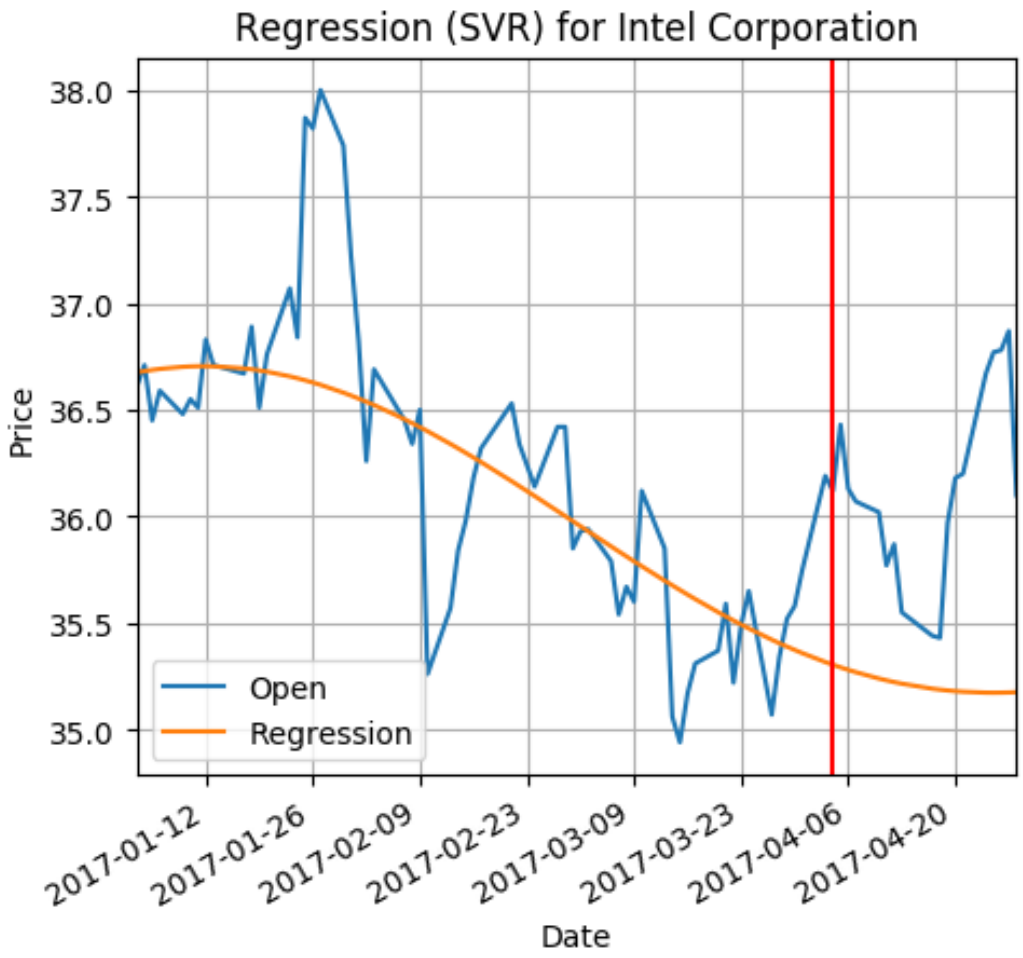
analizie danych giełdowych. Ich dokładność można zwiększyć odpowiednio parametryzując tworzone obiekty modeli regresji, oraz poprawnie dobierając dane.



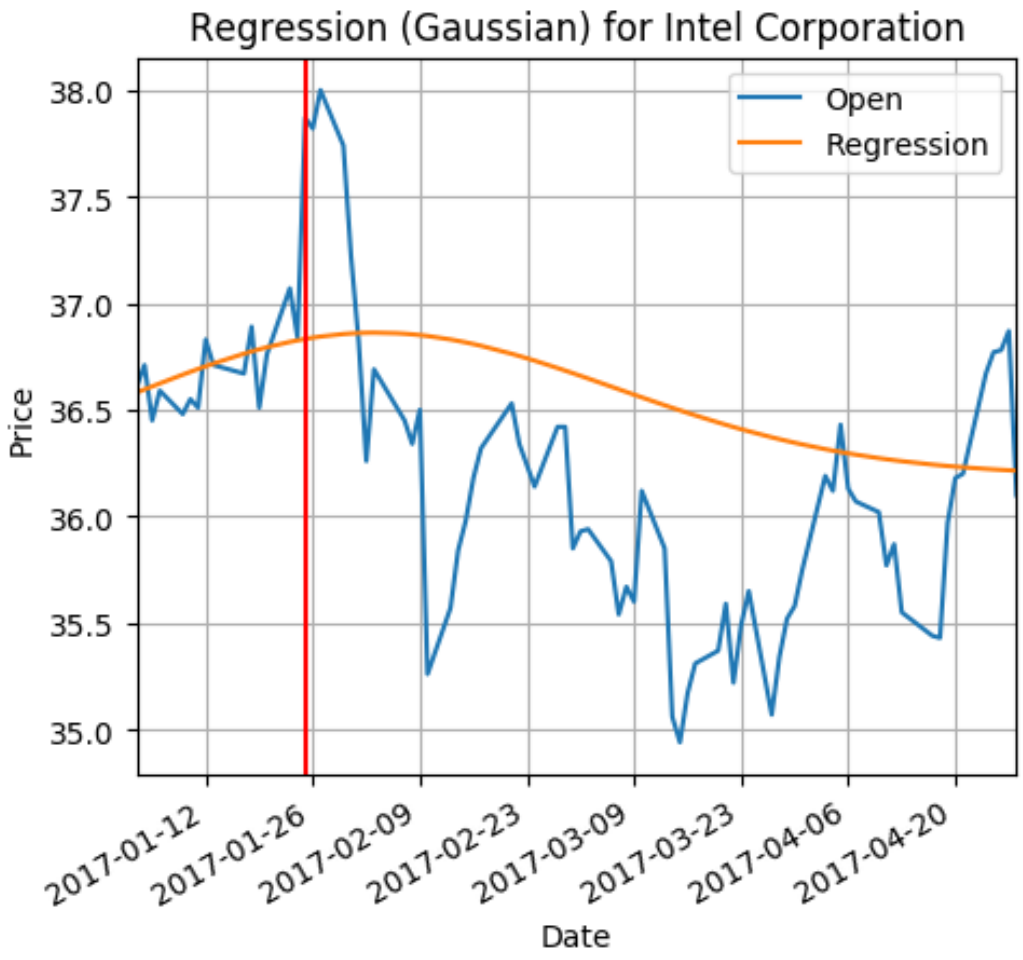
Rysunek 4.23. Wykres regresji wektorów nośnych dla 20% danych uczących, Intel



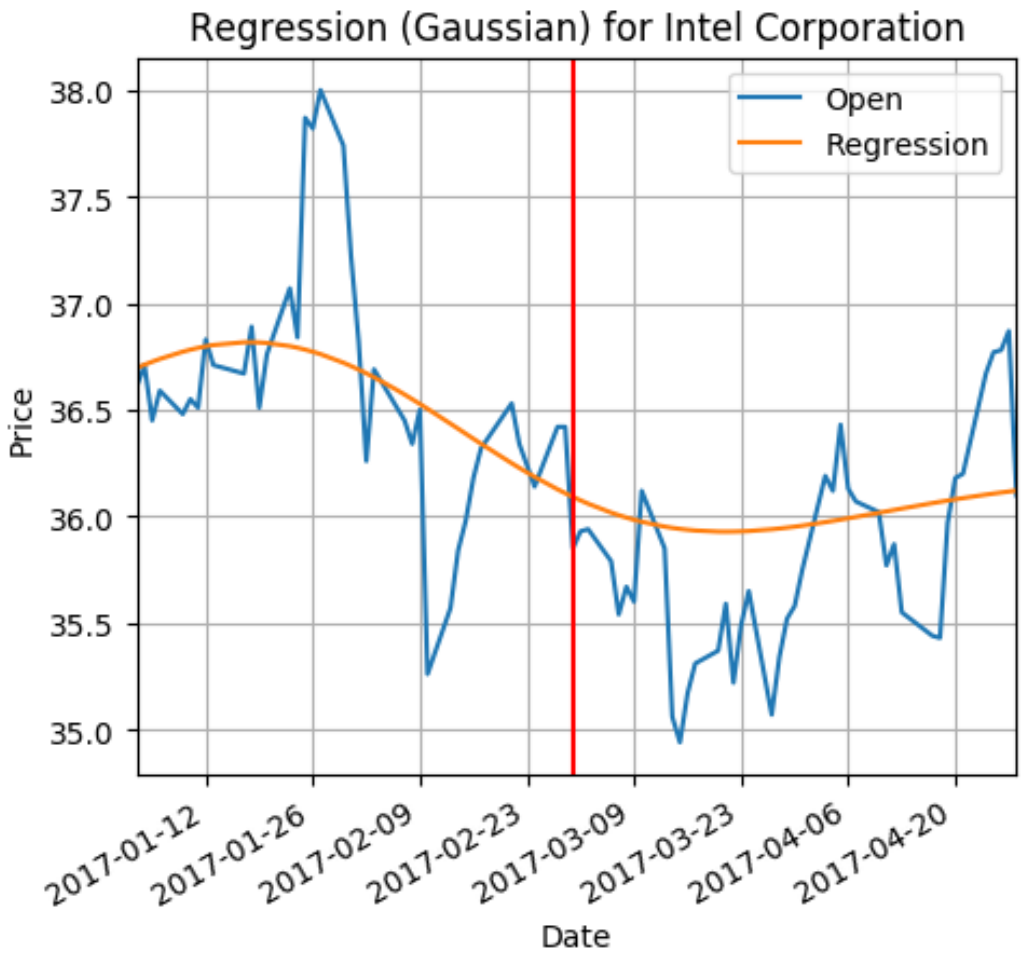
Rysunek 4.24. Wykres regresji wektorów nośnych dla 50% danych uczących, Intel



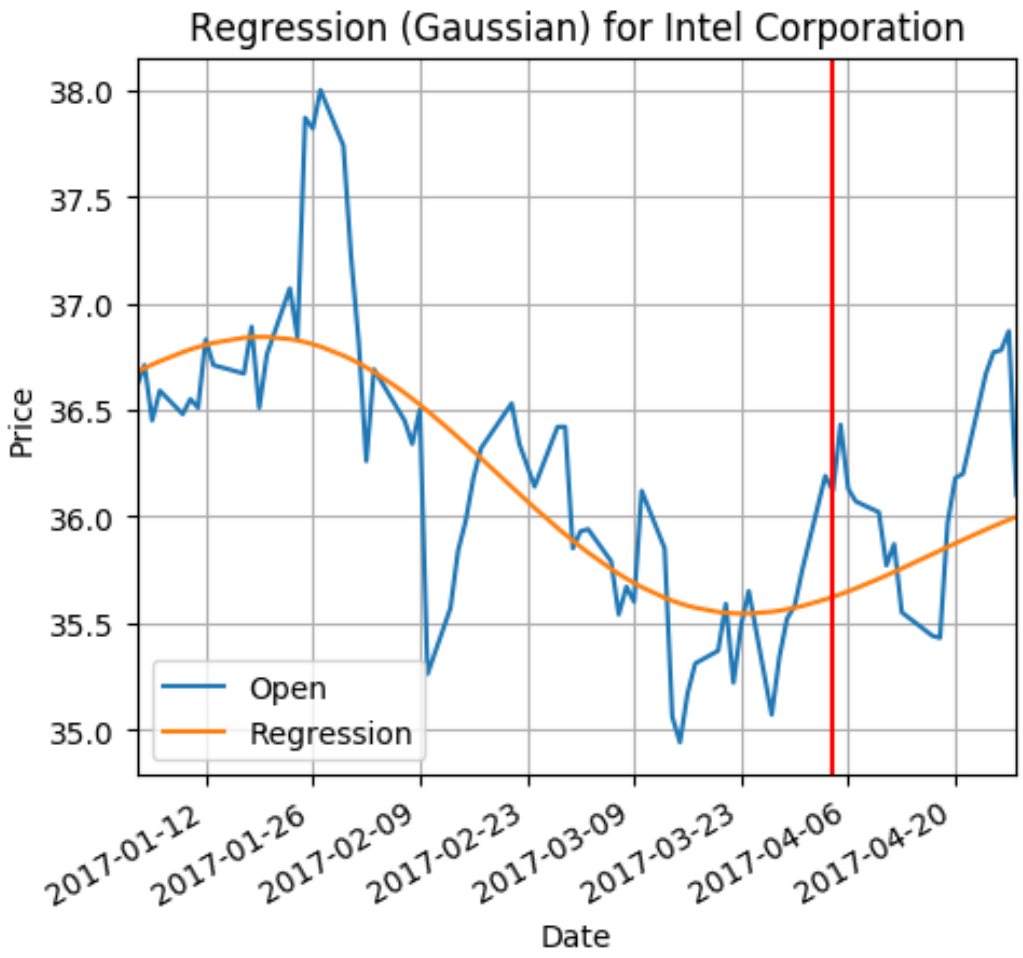
Rysunek 4.25. Wykres regresji wektorów nośnych dla 80% danych uczących, Intel



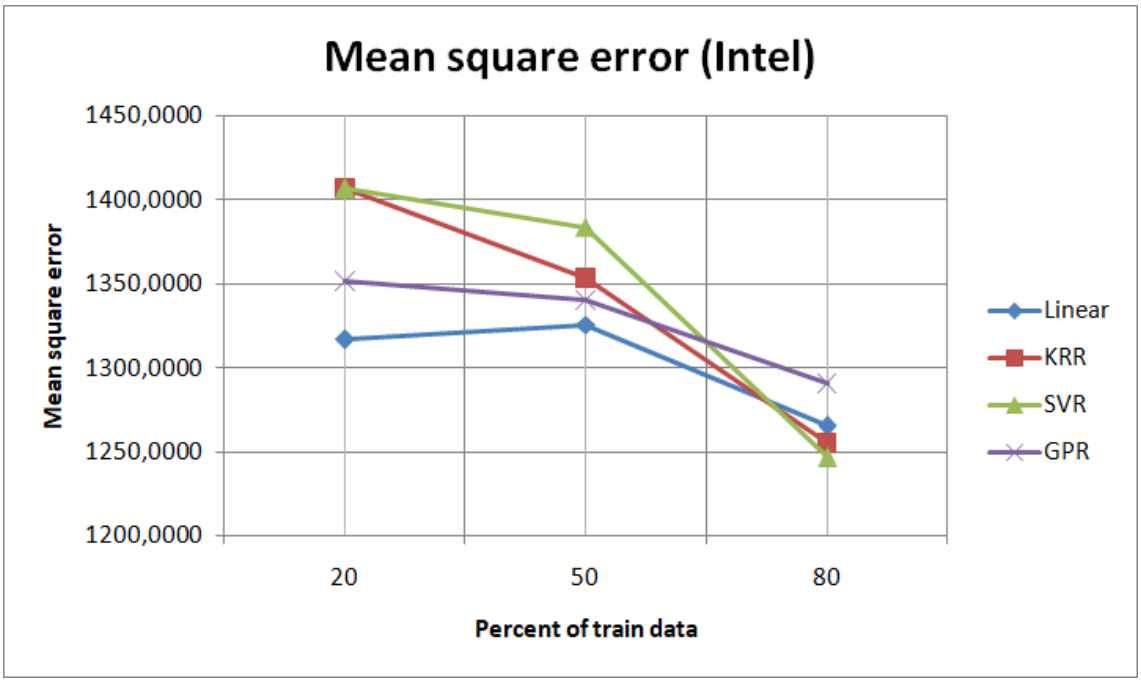
Rysunek 4.26. Wykres regresji procesu Gaussa dla 20% danych uczących, Intel



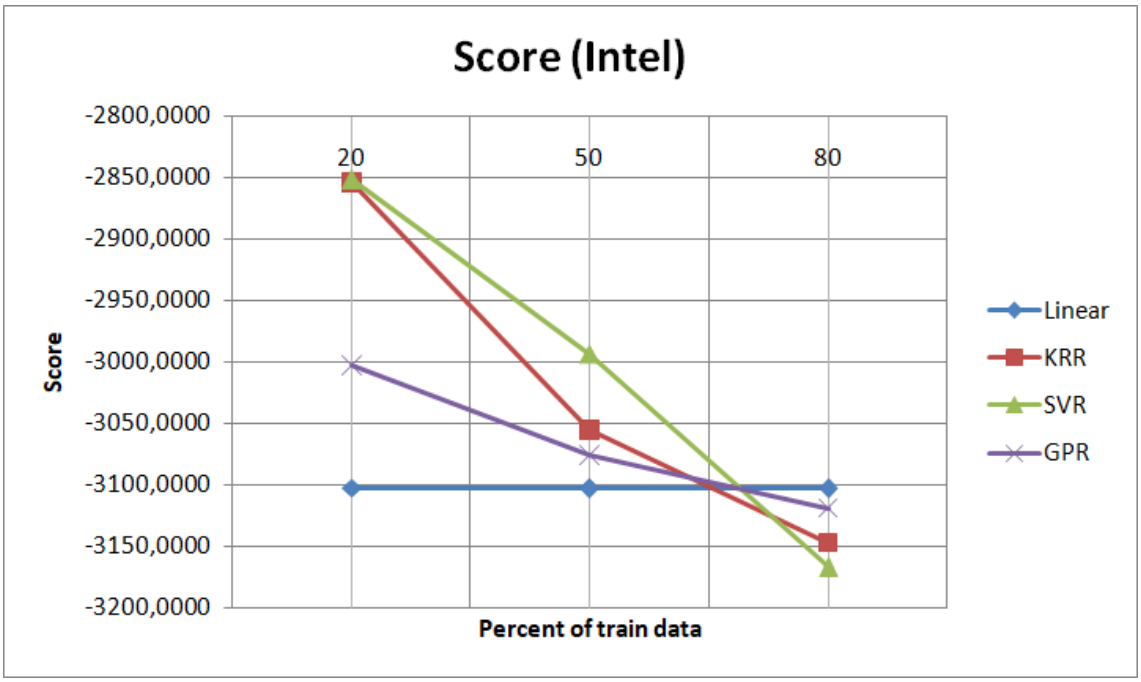
Rysunek 4.27. Wykres regresji procesu Gaussa dla 50% danych uczących, Intel



Rysunek 4.28. Wykres regresji procesu Gaussa dla 80% danych uczących, Intel



Rysunek 4.29. Wykres zmian wartości średniego błędu kwadratowego, Intel



Rysunek 4.30. Wykres zmian wartości wyników dopasowania modelu, Intel

Rozdział 5

Wnioski

Bibliografia

[1] Tobiasz Maliński, *Giełda Papierów Wartościowych Dla Bystrzaków*, Helion 2016.

[2] Justin Kuepper, *Basics of Technical Analysis*, 19 Kwiecień 2017. https://www.investopedia.com/university/technical/

[3] Investopedia, *Stock Market*, 20 Listopad 2017. https://www.investopedia.com/terms/s/stockmarket.asp

[4] Prawo o publicznym obrocie papierami wartościowymi i funduszach powierniczych *Ustawa z dnia 22 marca 1991r.*, Art 2.

[5] Roman Ciepiela, Piotr Pytlik, Magda Wiernusz, *Encyklopedia Zarządzania* 25 Październik 2016. https://mfiles.pl/pl/index.php/Akcje

[6] Roman Ciepiela, Szymon Kułakowski, Sabina Blok, *Encyklopedia Zarządzania*, 13 Lipiec 2017 https://mfiles.pl/pl/index.php/Obligacje

[7] Małgorzata Łuniewska *Ekonometria Finansowa: Analiza rynku kapitałowego*, Warszawa 2008 Wydawnictwo Naukowe PWN

[8] Stockopedia *Technical Analysis (Part 1): History, Theory and Philosophy* Kwiecień 2017 https://www.stockopedia.com/content/technical-analysis-part-1-history-theory-and-philosophy-17959

[9] G.S. Maddala *Ekonometria* Warszawa 2006 Wydawnictwo Naukowe PWN

[10] Mariusz Czekała *Analiza fundamentalna i techniczna* Wrocław 1997 Wydawnictwo Akademii Ekonomicznej im. Oskara Langego we Wrocławiu

[11] Jacek Koronacki, Jan Ćwik *Statystyczne Systemy Uczące Się* Warszawa 2005 Wydawnictwa Naukowo-Techniczne

[12] Emil Lundkvist *Decision Tree Classication and Forecasting of Pricing Time Series Data* Stockholm 2014 Master's Degree Project

[13] Alex Smola *Introduction to Machine Learning* 2008 Cambridge University Press

[14] Paweł Cichosz *Systemy Uczące Się* Warszawa 2000 Wydawnictwo Naukowo-Techniczne Warszawa

[15] Mirosław Krzyśko *Systemy uczące się* Warszawa 2008 Wydawnicatwa Naukowo-Techniczne

[16] Stanisław M. Kot *Statystyka* Warszawa 2011 Difin SA

[17] Siegmund Brandt *Analiza Danych* Warszawa 1998 Wydawnictwo Naukowe PWN

[18] Fabrizio Romano *Learning Python* 2015 Packt Publishing
[19] NumPy Community *NumPy User Guide* Release 1.13.0
[20] SciPy.org *SciPy Documentation* https://docs.scipy.org/doc/scipy/reference/
[21] Fabian Pedregosa *Scikit-learn: Machine Learning in Python* Journal of Machine Learning Research 12 (2011
[22] scikit-learn.org *Sciki-learn Documentation* http://scikit-learn.org/stable/documentation.html
[23] Gavin Hackeling *Mastering Machine Learning with scikit-learn* Packt Publishing

Spis rysunków

Spis tabel