
	Politechnika Bydgoska im. J. J. Śniadeckich Wydział Telekomunikacji, Informatyki i Elektrotechniki		
<b>Przedmiot</b>	Analiza Regresji i Szeregów Czasowych		
<b>Prowadzący</b>	dr inż. Damian Ledziński		
<b>Temat</b>	Regresja Liniowa, wdrożenie jForex		
<b>Student</b>	Cezary Tytko		
<b>Nr lab.</b>	3 i 4	<b>Data wykonania</b>	23.05.2024
<b>Ocena</b>		<b>Data oddania spr.</b>	

Zgodnie z poleceniem wczytałem dane „california\_housing” i przeprowadziłem eksploracyjną analizę danych z wykorzystaniem narzędzia „ProfileReport”

```

1. housing = fetch_california_housing()
2. housing_df = pd.DataFrame(data=housing.data, columns=housing.feature_names)
3. housing_df['target'] = housing.target
4. profile = ProfileReport(housing_df, title='California Housing Prices - EDA Report', explorative=True)
5. profile.to_file("california_housing_eda_report.html")
6.

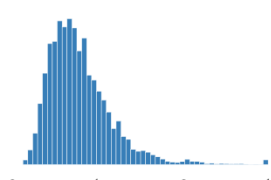
```

## Overview

Overview Alerts 8 Reproduction		Variable types	
Dataset statistics		Numeric 9	
Number of variables	9		
Number of observations	20640		
Missing cells	0		
Missing cells (%)	0.0%		
Duplicate rows	0		
Duplicate rows (%)	0.0%		
Total size in memory	1.4 MiB		
Average record size in memory	72.0 B		

## Variables

Select Columns			
MedInc			
Real number (R)			
HIGH CORRELATION			
Distinct	12928	Minimum	0.4999
Distinct (%)	62.6%	Maximum	15.0001
Missing	0	Zeros	0
Missing (%)	0.0%	Zeros (%)	0.0%
Infinite	0	Negative	0
Infinite (%)	0.0%	Negative (%)	0.0%
Mean	3.870671	Memory size	161.4 KiB



Następnie używając tych danych wytrenowałem 3 modele regresyjne(**LinearRegression**, **RandomForestRegressor** i **MLPRegressor**):

```
1. X_train, X_test, y_train, y_test = train_test_split(housing.data, housing.target,
test_size=0.2, random_state=42)
2. linear_reg = LinearRegression()
3. decision_tree_reg = RandomForestRegressor()
4. neural_network_reg = MLPRegressor(hidden_layer_sizes=(100, 50), activation='relu',
solver='adam', max_iter=1000)
5.
6. linear_reg.fit(X_train, y_train)
7. decision_tree_reg.fit(X_train, y_train)
8. neural_network_reg.fit(X_train, y_train)
9.
10. linear_reg_pred = linear_reg.predict(X_test)
11. decision_tree_reg_pred = decision_tree_reg.predict(X_test)
12. neural_network_reg_pred = neural_network_reg.predict(X_test)
13.
14. linear_reg_mse = mean_squared_error(y_test, linear_reg_pred)
15. decision_tree_reg_mse = mean_squared_error(y_test, decision_tree_reg_pred)
16. neural_network_reg_mse = mean_squared_error(y_test, neural_network_reg_pred)
17.
18. print("Mean Squared Error (MSE) dla regresji liniowej:", linear_reg_mse)
19. print("Mean Squared Error (MSE) dla drzewa decyzyjnego:", decision_tree_reg_mse)
20. print("Mean Squared Error (MSE) dla modelu opartego na sieciach neuronowych:", neural_ne-
twork_reg_mse)
21.
```

I sprawdziłem ich skuteczność na trzech metrykach (**MAE**, **RMSE** i **R2**):

```
1. linear_reg_mae = mean_absolute_error(y_test, linear_reg_pred)
2. decision_tree_reg_mae = mean_absolute_error(y_test, decision_tree_reg_pred)
3. neural_network_reg_mae = mean_absolute_error(y_test, neural_network_reg_pred)
4.
5. linear_reg_rmse = np.sqrt(mean_squared_error(y_test, linear_reg_pred))
6. decision_tree_reg_rmse = np.sqrt(mean_squared_error(y_test, decision_tree_reg_pred))
7. neural_network_reg_rmse = np.sqrt(mean_squared_error(y_test, neural_network_reg_pred))
8.
9. linear_reg_r2 = r2_score(y_test, linear_reg_pred)
10. decision_tree_reg_r2 = r2_score(y_test, decision_tree_reg_pred)
11. neural_network_reg_r2 = r2_score(y_test, neural_network_reg_pred)
12.
13. print("Mean Absolute Error (MAE):")
14. print("Regresja liniowa:", linear_reg_mae)
15. print("Drzewo decyzyjne:", decision_tree_reg_mae)
16. print("Sieć neuronowa:", neural_network_reg_mae)
17.
18. print("\nRoot Mean Squared Error (RMSE):")
19. print("Regresja liniowa:", linear_reg_rmse)
20. print("Drzewo decyzyjne:", decision_tree_reg_rmse)
21. print("Sieć neuronowa:", neural_network_reg_rmse)
22.
23. print("\nR^2 score:")
24. print("Regresja liniowa:", linear_reg_r2)
25. print("Drzewo decyzyjne:", decision_tree_reg_r2)
26. print("Sieć neuronowa:", neural_network_reg_r2)
27.
```

```
Mean Absolute Error (MAE):  
Regresja liniowa: 0.5332001304956989  
Drzewo decyzyjne: 0.32685366124031023  
Sieć neuronowa: 0.602894055920684  
  
Root Mean Squared Error (RMSE):  
Regresja liniowa: 0.7455813830127751  
Drzewo decyzyjne: 0.5031943753976059  
Sieć neuronowa: 0.7946062468220908  
  
R^2 score:  
Regresja liniowa: 0.5757877060324521  
Drzewo decyzyjne: 0.8067743859846794  
Sieć neuronowa: 0.5181663908192364
```

Najlepszy okazał się las losowy (opisany jako wariant drzewa decyzyjnego)

# Historyczne dane Forex:

Pobrałem historyczne dane dla pary walutowej EUR/PLN jako świecek godzinowe z okresu 3 lat

EURPLN\_Candlestick\_1\_h\_BID\_01.05.2021-30.04.2024

Dodałem kolumnę z numerem dnia tygodnia i numerem godziny w ciągu dnia (dane przydatne ze względu na zamknięcie rynku w weekend)

```
1. df = pd.read_csv('EURPLN_Candlestick_1_h_BID_01.05.2021-30.04.2024.csv', parse_dates=['Gmt time'],
2.                 date_parser=lambda x: pd.to_datetime(x, format='%d.%m.%Y %H:%M:%S.%f'))
3. # Dodaj kolumnę z dniem tygodnia
4. df['Day of Week'] = df['Gmt time'].dt.dayofweek
5. # Dodaj kolumnę z godziną w ciągu dnia
6. df['Hour of Day'] = df['Gmt time'].dt.hour
7. df = df.drop(columns=['Gmt time'])
8.
```

Specjalnie przygotowaną funkcją podzieliłem dane na zbiory: badany okres – docelowa wartość, funkcja ta przygotowuje dane w formacie przeznaczonym dla uczenia sieci LSTM, dlatego dodatkowo rozwinąłem macierz do wiersza, aby przekazać dane do prostego modelu drzewa decyzyjnego, aby pokazać działanie „m2cgen”, choć lepszym rozwiązaniem byłoby użycie sieci rekurencyjnej.

```
1. def GetTimeSeriesData(df: pd.DataFrame, target_col: str, point_per_series: int, forward_in-
dex: int, step_next: int=5):
2.     listNew = []
3.     listValue = []
4.     data_range = point_per_series
5.     data_range_index = data_range - 1
6.     full_range = point_per_series + forward_index
7.     full_range_index = full_range - 1
8.     start = 0
9.     print(df.__len__())
10.    for j in range(start, df.__len__() - full_range_index, step_next):
11.        listTemp = []
12.        for k in range(0, data_range):
13.            listTemp.append(df.iloc[k+j][df.columns.difference([target_col]).tolist()])
14.            # listTemp.append(df[j + (k - start)])
15.        listNew.append(listTemp)
16.        listValue.append((df.iloc[full_range_index+j][target_col] - df.iloc[data_range_in-
dex+j][target_col]) >= 0)
17.        # listValue.append((tab[j + stepForward][targetIndex] >= tab[j][targetIn-
des])).astype(int))
18.    return np.array(listNew), np.array(listValue)
19.
```

Dane wcześniej ustandaryzowałem, choć dla drzewa decyzyjnego nie ma to znaczenia (od kolumny volume nie odjąłem średniej, aby zachować wartości 0 kiedy mamy brak handlu na rynku)

```
1.
2. print(df.head())
3. columns_to_standardize = ['Open', 'High', 'Low', 'Close']
4. df_standardized = df.copy()
```

```

5. # Inicjalizuj StandardScaler
6. scaler = StandardScaler()
7.
8. # Standaryzuj wybrane kolumny
9. df_standardized[columns_to_standardize] = scaler.fit_transform(df[columns_to_standardize])
10.
11. # Dla kolumny 'Volume' podziel wartości przez odchylenie standardowe
12. volume_std = df['Volume'].std()
13. df_standardized['Volume'] = df['Volume'] / volume_std
14.
15. print(df_standardized.head())
16.

1. time_series_data, time_series_Value = GetTimeSeriesData(df_standardized, 'Open', 10, 2, 1)
2. print(time_series_data.shape)
3. print(time_series_Value.shape)
4.

```

Pobieramy 10 wierszy jako baza do analizy, kolumną przewidywaną jest „open” dwa wiersze w przód, skok do kolejnej kombinacji następuje co 1 wiersz

```

26304
(26293, 10, 6)
(26293,)

```

```

1. reshaped_array = time_series_data.reshape(time_series_data.shape[0], -1)
2. print(reshaped_array.shape)
3.

```

```

(26293, 60)

```

```

1. model = DecisionTreeRegressor(random_state=42)
2. model.fit(X_train, y_train)
3.
4. # Przewidywanie na zbiorze testowym
5. y_pred = model.predict(X_test)
6.
7. # Ocena modelu
8. mse = mean_squared_error(y_test, y_pred)
9. print(f'Mean Squared Error: {mse}')
10.

```

```

Mean Squared Error: 0.30823350446853015

```

Wynik nie był dla mnie istotny, ponieważ głównym celem było pokazanie działania biblioteki „m2cgen” w innym przypadku skorzystałbym z sieci neuronowych.

```

1. java_code = m2c.export_to_java(model)
2.
3. # Zapisanie kodu Java do pliku
4. with open("LinearRegressionModel.java", "w") as f:
5.     f.write(java_code)
6.

```

```

J LinearRegressionModel.java
1  public class Model {
2      public static double score(double[] input) {
3          double var0;
4          if (input[59] <= 0.00013281732390169054) {
5              if (input[57] <= 18.5) {
6                  if (input[47] <= 0.10094116814434528) {
7                      if (input[1] <= 2.5) {
8                          if (input[49] <= 1.5) {
9                              var0 = 1.0;
10                         } else {
11                             var0 = 0.0;
12                         }
13                     } else {
14                         var0 = 1.0;
15                     }
16                 } else {
17                     var0 = 0.0;
18                 }
19             } else {
20                 if (input[19] <= 5.5) {
21                     if (input[16] <= 2.6488927602767944) {
22                         if (input[55] <= 3.5) {
23                             if (input[29] <= 1.9615308046340942) {
24                                 if (input[35] <= 0.3587728291749954) {
25                                     if (input[47] <= 0.5552820935845375) {
26                                         if (input[6] <= -1.6844459772109985) {
27                                             var0 = 1.0;
28                                         } else {
29                                             if (input[55] <= 1.0) {
30                                                 if (input[9] <= 11.5) {
31                                                     var0 = 1.0;
32                                                 } else {
33                                                     var0 = 0.0;
34                                                 }
35                                             } else {
36                                                 var0 = 1.0;

```

Jak widać wszystko zadziałało i otrzymaliśmy wynik w postaci drzewa decyzyjnego zaimplementowanego w java, należy zauważyć że wyniki są w postaci danych po standaryzacji, przy wdrażaniu takiego modelu należy to uwzględnić i proces odwrócić

## Wdrażane modelu na JForex:

Jest to stosunkowo proste zadanie jeżeli podejdziemy do tego rozsądnie, najpierw należy się zastanowić jakie dane możemy wyłuskać w trakcie działania strategii, następnie dla określonych danych, pobrać ich wartości historyczne i wytrenować na nich model,. Za pomocą biblioteki m2cgen możemy przenieść model do języka java, następnie zaimportować model na do strategii, wykonywać na jego postawie predykcji, a na podstawie tych predykcji podejmować działania.

Dla pokazania samej implementacji utworzyłem prostu model drzewa decyzyjnego, biorący wartości z 5 ostatnich godzin ceny zamknięcia, który próbuje przewidzieć czy za godzinę będzie wzrost czy spadek, jeżeli ma być wzrost to kupujemy, a jeżeli model przewiduje spadek to zamykamy:

```
1. import com.dukascopy.api.*;
2. import java.util.*;
3.
4. public class MyStrategy implements IStrategy {
5.     private IEngine engine;
6.     private IConsole console;
7.     private IHistory history;
8.     private IIndicators indicators;
9.     private IContext context;
10.    @Configurable("selectedInstrument:")
11.    public Instrument instrument = Instrument.EURPLN;
12.    private IOrder order;
13.    private static final int WINDOW_SIZE = 5;
14.
15.    // Wygenerowany model drzewa decyzyjnego
16.    private Model model;
17.
18.    @Override
19.    public void onStart(IContext context) throws JFException {
20.        this.context = context;
21.        this.engine = context.getEngine();
22.        this.console = context.getConsole();
23.        this.history = context.getHistory();
24.        this.indicators = context.getIndicators();
25.        this.model = new Model();
26.        console.getOut().println("Strategy started.");
27.    }
28.
29.    @Override
30.    public void onBar(Instrument instrument, Period period, IBar askBar, IBar bidBar) throws JFException {
31.        if (!instrument.equals(this.instrument) || !period.equals(Period.ONE_HOUR)) {
32.            console.getOut().println("Return");
33.            return;
34.        }
35.
36.        // Pobierz ostatnie 6 cen zamknięcia
37.        List<Double> closePrices = new ArrayList<>();
38.        for (int i = 0; WINDOW_SIZE + 1 > i; i++) {
39.            IBar bar = history.getBar(instrument, Period.ONE_HOUR, OfferSide.BID, i);
40.            closePrices.add(bar.getClose());
```

```

41.     }
42.     Collections.reverse(closePrices);
43.
44.     // Oblicz różnice
45.     double[] diffs = new double[WINDOW_SIZE];
46.     for (int i = 0; i < WINDOW_SIZE; i++) {
47.         diffs[i] = closePrices.get(i + 1) - closePrices.get(i);
48.     }
49.
50.     // Przewiduj zmianę na następną godzinę
51.     double prediction = model.predict(diffs);
52.
53.     // Jeśli przewidywana zmiana jest dodatnia, otwórz transakcję kupna
54.     if (prediction > 0) {
55.         if (order == null || order.getState() == IOrder.State.CLOSED) {
56.             order = engine.submitOrder("Order_" + System.currentTimeMillis(), instrument,
IEngine.OrderCommand.BUY, 0.1);
57.             console.getOut().println("Opened BUY order.");
58.         }
59.     }
60.
61.     // Zamknij transakcję, jeśli wartość zacznie spadać
62.     if (order != null && order.getState() == IOrder.State.FILLED) {
63.         if (prediction < 0) {
64.             order.close();
65.             console.getOut().println("Closed order due to falling price.");
66.         }
67.     }
68. }
69.
70.
71.
72. @Override
73. public void onTick(Instrument instrument, ITick tick) throws JFException {
74.     // Nie potrzebujemy implementacji dla onTick w tej strategii
75. }
76.
77. @Override
78. public void onMessage(IMessage message) {
79.     // Można tutaj obsłużyć różne komunikaty
80. }
81.
82. @Override
83. public void onAccount(IAccount account) {
84.     // Można tutaj obsłużyć informacje o koncie
85. }
86.
87. @Override
88. public void onStop() throws JFException {
89.     if (order != null && order.getState() == IOrder.State.FILLED) {
90.         order.close();
91.     }
92.     console.getOut().println("Strategy stopped.");
93. }
94. public class Model {
95.     public double predict(double[] input) {
96.         double var0;
97.         if (input[3] <= 0.06116499751806259) {
98. ...
99.

```

**Wnioski:** Model się nie sprawdził, ale nie to było celem, celem było pokazanie jak wdrożyć taki model, kluczową kwestią jest to że wytrenowany model musi działać na tym samym instrumencie i parametrach, w przeciwnym wypadku wyniki można uznać za losowe, nawet jeśli byłby pozytywne.



MyStrategy strategy report for EUR/PLN instrument(s) from 2024-02-29 22:00:00 to 2024-05-31 20:59:59

Account Currency	USD
Initial Equity	50'000
Final Equity	49'674,81
Turnover	1'298'509,41
Commission Fees	24.13

## Parameters

instrument	EUR/PLN
------------	---------

## Instrument EUR/PLN

First tick time	2024-02-29 22:00:00
First BID	4.31402
First ASK	4.31747
Last tick time	2024-05-31 20:59:59
Last BID	4.26849
Last ASK	4.27951
Positions total	6
Closed positions	6
Orders total	6
Amount bought	0.60
Amount sold	0.60
Turnover	1'298'509,41
Commission Fees	24.13

## Opened orders:

Label	Amount	Direction	Open price	Profit/Loss at the end	Profit/Loss at the end in pips	Open date	Comment
-------	--------	-----------	------------	------------------------	--------------------------------	-----------	---------

## Closed orders:

Label	Amount	Direction	Open price	Close price	Profit/Loss	Profit/Loss in pips	Open date	Close date	Comment
Order_1718293708581	0,100	BUY	4.3179	4.31344	-112.00	-44.6	2024-02-29 23:00:00	2024-03-01 02:00:00	
Order_1718293708628	0,100	BUY	4.31572	4.31587	3.75	1.5	2024-03-01 03:00:00	2024-03-01 10:00:00	
Order_1718293708757	0,100	BUY	4.32091	4.31841	-62.66	-25.0	2024-03-01 11:00:00	2024-03-01 13:00:00	
Order_1718293708814	0,100	BUY	4.32051	4.31931	-30.03	-12.0	2024-03-01 14:00:00	2024-03-01 15:00:00	
Order_1718293708868	0,100	BUY	4.31782	4.31687	-23.88	-9.5	2024-03-01 16:00:00	2024-03-01 19:00:00	
Order_1718293708950	0,100	BUY	4.3178026	4.31477	-76.24	-30.3	2024-03-01 20:00:00	2024-03-02 00:00:00	