Problem definition
The analysis aims to determine changes in warehouse stock levels at individual distribution points. The business goal of such analysis is to predict the amount of goods that will flow in and to counteract surpluses in the warehouse.

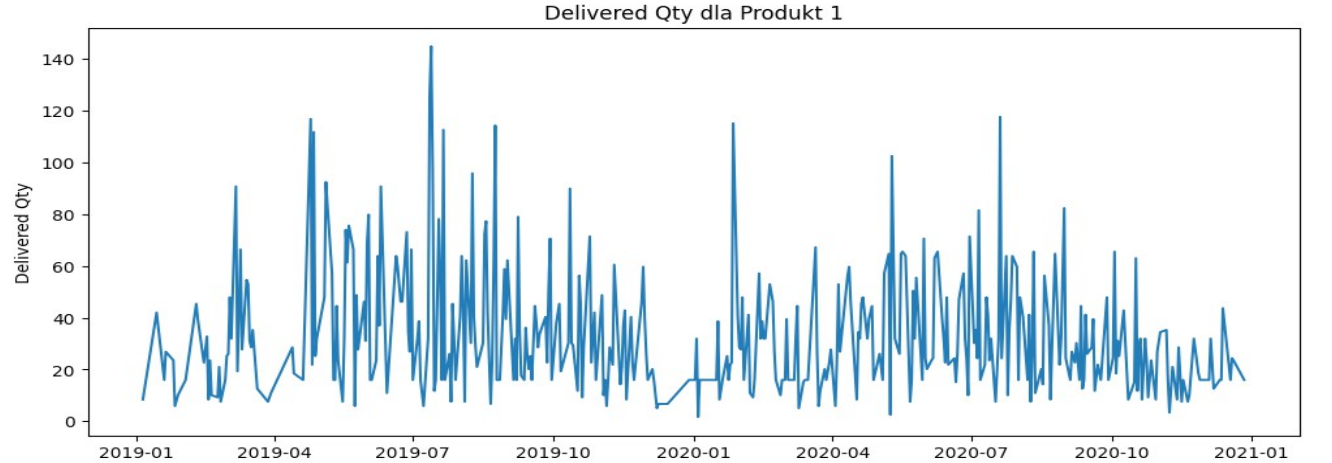
Data
Initial stock 0 at all distributors,
No information on the storage area of individual warehouses.
No information on emptying warehouses.

Exploratory data analysis (EDA)
Examination of the structure and quality of data.
Checking for gaps, outliers, types of variables.
Descriptive statistics (mean, median, variance).
Initial detection of correlations between variables.
Data cleaning
Correction of errors, checking for duplicates and NaN values, transforming data to appropriate formats.

Data visualization.
Distribution charts, box plots, histograms.
Scatter plots, correlation heatmaps.
Time plots
Visualization of anomalies, checking for outliers.
2027 outliers were detected in all tested products. This indicates that there may be large jumps in stock levels and space problems. Such outliers also affect the stability of the model.
Unique values in the 'Product' column: 13
Unique values in the 'Loading_place' column: 7
Unique values in the 'Department_number' column: 2158

Time plots were made for each of the 13 products.

```
produkty = df_grouped['Produkt'].unique()

for produkt in produkty:
    df_temp = df_grouped[df_grouped['Produkt'] == produkt]
    plt.figure(figsize=(12,5))
    plt.plot(df_temp['Data'], df_temp['Delivered Qty'])
    plt.title(f'Delivered Qty dla {produkt}')
    plt.xlabel('Data')
    plt.ylabel('Delivered Qty')
    plt.show()
```



Conclusions: There are abrupt fluctuations in the list of products in the warehouses. Some products have little data (occur rarely). This can have a negative impact on the overall evaluation of the model. Data engineering.
The column "branch number" has been removed. This feature cannot be treated as a numeric variable because the number of branch 1 has no other value than the number of branch 2. Performing one hot encoding on 2158 unique values is cumbersome.

Based on the column "Date", the following columns were created: Year_Week, Weekend, Workday, Month, WeekofYear, Holiday, SeasonOfYear. These data, which are cyclical, were transformed using trigonometric functions. Categorical data were transformed using one hot encoding.

ANOVA test (analysis of variance) for each product separately. In this way, it was checked whether the average quantity of delivered goods (Delivered Qty) differs significantly between working day, weekend, month, holiday and seasons.

Conclusions. There are no statistically significant differences between average stock levels and working day, weekend, holiday for all 13 products, for some products there is a statistically significant difference between average stock levels and month and for all products there is a statistically significant difference between season and average stock levels. Therefore, the column season_year will be included in the prediction.

```
produkty = df['Produkt'].unique()

for produkt in produkty:
    df_temp = df[df['Produkt'] == produkt]


    groups = [group['Delivered Qty'].values for name, group in df_temp.groupby('Pora_roku') if len(group) > 1]
    seasons = [name for name, group in df_temp.groupby('Pora_roku') if len(group) > 1]

    if len(groups) < 2:
        print(f"{produkt}: Za mało grup do testu ANOVA")
        continue

    f_stat, p_value = stats.f_oneway(*groups)

    print(f"{produkt}:")
    print(f"  Test ANOVA (pory roku): F={f_stat:.3f}, p={p_value:.5f}")
    if p_value < 0.05:
        print("  -> Istotna różnica między porami roku (p < 0.05)\n")
    else:
        print("  -> Brak istotnej różnicy między porami roku (p >= 0.05)\n")
```

```
Produkt 21:
  Test ANOVA (pory roku): F=3.148, p=0.02438
  -> Istotna różnica między porami roku (p < 0.05)

Produkt 25:
  Test ANOVA (pory roku): F=9.252, p=0.00000
  -> Istotna różnica między porami roku (p < 0.05)

Produkt 27:
  Test ANOVA (pory roku): F=34.485, p=0.00000
  -> Istotna różnica między porami roku (p < 0.05)

Produkt 15:
  Test ANOVA (pory roku): F=68.960, p=0.00000
  -> Istotna różnica między porami roku (p < 0.05)
```

XGB model

This model was initially chosen because:
it copes well with outliers, which occur here
it is not necessary to assume stationarity of time series as in ARIMA, it is necessary to encode temporal features via lags.
After encoding one-hot categorical features, it learns nonlinear dependencies well
1. Grouping data weekly + aggregating (summing Delivered Qty weekly for each product and loading location. Also calculating the average weekly distance for each product and loading location. The average distance can affect the variability of deliveries.

2. Sorting by time, data must be chronological to calculate lags
3. Calculating the weekly change in the delivered quantity Delivered_Delta - this is the target for prediction. Instead of predicting the value of the delivery, the change is predicted relative to the previous week. This gives better quality of prediction and allows for better detection of anomalies.

4. Creating lagged features. These are features that show how a given value looked in the past:
Delivered_Delta_lag1: delivery change 1 week ago
Delivered_Delta_lag2: 2 weeks ago, etc.
5. Cleaning up gaps after lag. First 3 weeks for each product/location have NaN in lags.

6. Encoding categorical variables. Transforming text variables (Product, Item, Season) into numbers.

7. Data preparation and splitting into features and target. X: input features (laged variables + categories) y: target for prediction, i.e. Delivered_Delta

8. Time validation (TimeSeriesSplit). Splitting data according to time - the model trains on the past and tests on the future. There is no mixing of data from different periods (this would be data leakage).

9. Training the XGBoost model + metrics

```python
# Usuwanie wierszy z brakami po lagach
df_grouped = df_grouped.dropna()

# Kodowanie kategorii (Produkt, Miejsce_zaladunku, Pora_roku)
df_grouped = pd.get_dummies(df_grouped, columns=['Produkt', 'Miejsce_zaladunku', 'Pora_roku'], drop_first=True)

# Przygotowanie cech i targetu
exclude_cols = ['Rok_Tydzien', 'Delivered Qty', 'Delivered_Delta']
feature_cols = [col for col in df_grouped.columns if col not in exclude_cols]

X = df_grouped[feature_cols]
y = df_grouped['Delivered_Delta']

# TimeSeriesSplit
tscv = TimeSeriesSplit(n_splits=5)

r2_list = []

for fold, (train_idx, val_idx) in enumerate(tscv.split(X)):
    X_train, X_val = X.iloc[train_idx], X.iloc[val_idx]
    y_train, y_val = y.iloc[train_idx], y.iloc[val_idx]

    model = XGBRegressor(objective='reg:squarederror', n_estimators=200, random_state=42)
    model.fit(X_train, y_train)

    preds = model.predict(X_val)
    r2 = r2_score(y_val, preds)

    print(f"Fold {fold+1}:  R2 = {r2:.3f}")
    r2_list.append(r2)

print(f"Average R2: {sum(r2_list)/len(r2_list):.3f}")
```

```
Fold 1:  R2 = 0.661
Fold 2:  R2 = 0.703
Fold 3:  R2 = 0.428
Fold 4:  R2 = 0.846
Fold 5:  R2 = 0.690
Average R2: 0.666
```

LSTM.

LSTM model will receive:

Sequential input: weekly sales history.

Static input: data independent of sequence in a given week (e.g. distance, season, etc.).

1. Create time features: holiday, month, season.

2. Trigonometric encoding of cyclical features.

3. Transform categorical features into numeric ones – season, get dummies

4. Chronological sorting, create Year_week column for aggregation
5. Weekly aggregation because the model has to predict at the weekly level.

6. Pivot sales to products (sequence), so there is one row = one week with columns as products.

7. Preparation of static data: 'distance', 'month_sin', 'month_cos', 'day_of_the_year_sin',

'day_of_the_year_cos', 'whether_holiday', 'season_of_autumn', 'season_of_summer', 'season_of_the_year_spring'. Pivot of static features. This is the input to the second branch of the model. In this version of the LSTM model, a simplification was applied, passing time features outside the sequence, as "static" auxiliary features. Adding these features as separate vectors for each time step would require changing the input to:
X_seq_train.shape = (samples, time_steps, features)
where features = number of products + number of time variables
8. Merging and sorting chronologically.
9. Splitting into train and test chronologically to avoid data leakage. 10. Scaling only on training to avoid data leakage.
11. Creating LSTM sequences. The model analyzes the past window 4, so 4 weeks ago.

```
# Parametry
seq_len = X_seq_train.shape[1]       # liczba kroków czasowych (window)
num_products = X_seq_train.shape[2]  # liczba produktów (wymiar cech sekwencyjnych)
num_static = X_stat_train.shape[1]   # liczba cech statycznych

# 1. Definicja modelu

# Wejście sekwencyjne
input_seq = Input(shape=(seq_len, num_products), name='input_sequence')
x = LSTM(64, return_sequences=False)(input_seq)
x = Dropout(0.2)(x)
x = Dense(32, activation='relu')(x)

# Wejście statyczne
input_stat = Input(shape=(num_static,), name='input_static')
y = Dense(16, activation='relu')(input_stat)
y = Dropout(0.2)(y)

# Połączenie obu wejść
combined = Concatenate()([x, y])
z = Dense(64, activation='relu')(combined)
z = Dense(num_products, activation='linear')(z)  # regresja - predykcja wartości sprzedaży na każdy produkt

# Model
model = Model(inputs=[input_seq, input_stat], outputs=z)
```

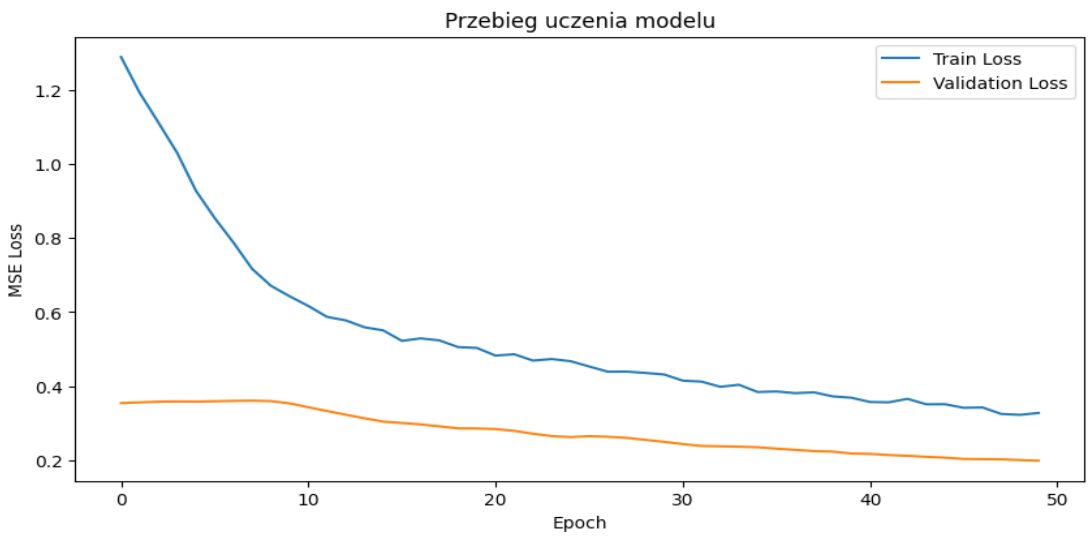

```
# Parametry
seq_len = X_seq_train.shape[1]       # liczba kroków czasowych (window)
num_products = X_seq_train.shape[2]  # liczba produktów (wymiar cech sekwencyjnych)
num_static = X_stat_train.shape[1]   # liczba cech statycznych

# 1. Definicja modelu

# Wejście sekwencyjne
input_seq = Input(shape=(seq_len, num_products), name='input_sequence')
x = LSTM(64, return_sequences=False)(input_seq)
x = Dropout(0.2)(x)
x = Dense(32, activation='relu')(x)

# Wejście statyczne
input_stat = Input(shape=(num_static,), name='input_static')
y = Dense(16, activation='relu')(input_stat)
y = Dropout(0.2)(y)

# Połączenie obu wejść
combined = Concatenate()([x, y])
z = Dense(64, activation='relu')(combined)
z = Dense(num_products, activation='linear')(z)  # regresja - predykcja wartości sprzedaży na każdy produkt

# Model
model = Model(inputs=[input_seq, input_stat], outputs=z)
```


```
model.compile(optimizer='adam', loss='mse')

model.summary()
```

12. Model training, dropout regularization 0.2. Sequential input (input_seq): LSTM analyzes the last window of weeks of sales of all products. Static input (input_stat): Data independent of the sequence, such as distance or seasons. Both inputs are processed separately, then combined, which allows the model to take into account different types of information. The output layer is linear, because of regression.
13. Evaluation: MSE and R2 on all products simultaneously and also separately.

    Conclusions: Some products have incomplete data and that is why R2 is so weak. The model needs some refinement, but for a start the R2 result of - 0.65 for product 3 is quite good. Additional features are needed for the model to be able to learn in a larger context.
    The model may have problems with generalization, has a constant number of outputs corresponding to training products and will not predict well other points and products than those on which it was trained.

```
import matplotlib.pyplot as plt

plt.figure(figsize=(10,5))
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('MSE Loss')
plt.title('Przebieg uczenia modelu')
plt.legend()
plt.show()
```



```
Mean Squared Error na zbiorze testowym: 0.2161
R^2 na zbiorze testowym: 0.4094
```

```
[164] num_products = y_test.shape[1]
      r2_per_product = []

      for i in range(num_products):
          r2_p = r2_score(y_test[:, i], y_pred[:, i])
          r2_per_product.append(r2_p)
          print(f'R^2 dla produktu {i}: {r2_p:.4f}')
```

```
R^2 dla produktu 0: 0.6090
R^2 dla produktu 1: -34359230090260351225749118648320.0000
R^2 dla produktu 2: 0.3744
R^2 dla produktu 3: 0.6598
R^2 dla produktu 4: -74051235564423612161054893670 4.0000
R^2 dla produktu 5: -36546135129146544430525332848 64.0000
R^2 dla produktu 6: 0.2087
R^2 dla produktu 7: -11693094265528151343864595087360.0000
R^2 dla produktu 8: 0.1025
R^2 dla produktu 9: 0.1523
R^2 dla produktu 10: 0.2658
R^2 dla produktu 11: 0.5808
R^2 dla produktu 12: 0.0078
```

```
produkt_idx = 3

plt.figure(figsize=(10,5))
plt.plot(y_test[:, produkt_idx], label='Rzeczywiste wartości')
plt.plot(y_pred[:, produkt_idx], label='Predykcje modelu')
plt.xlabel('Próbka (tydzień)')
plt.ylabel('Delivered Qty')
plt.title(f'Porównanie predykcji i wartości rzeczywistych - Produkt {produkt_idx}')
plt.legend()
plt.show()
```