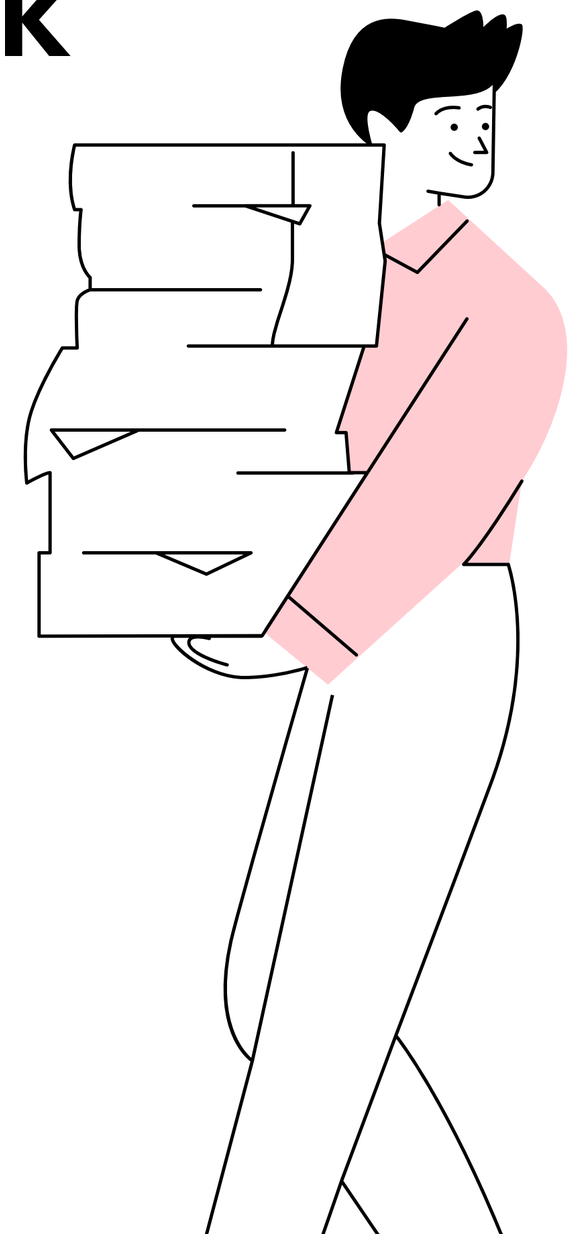
Makine Öğrenmesi ile Stok Optimizasyonu

Furkan GÜRYEL

İçerik



Projenin Konusu ve Amaç

Market reyonu üretimi yapan ve yarı mamullerini satın alan bir fabrikanın geçmiş satın alma, stok ve üretim verilerden yola çıkarak stok durumunu en iyi hale getirecek satın alma tahminlemesi yaptık.

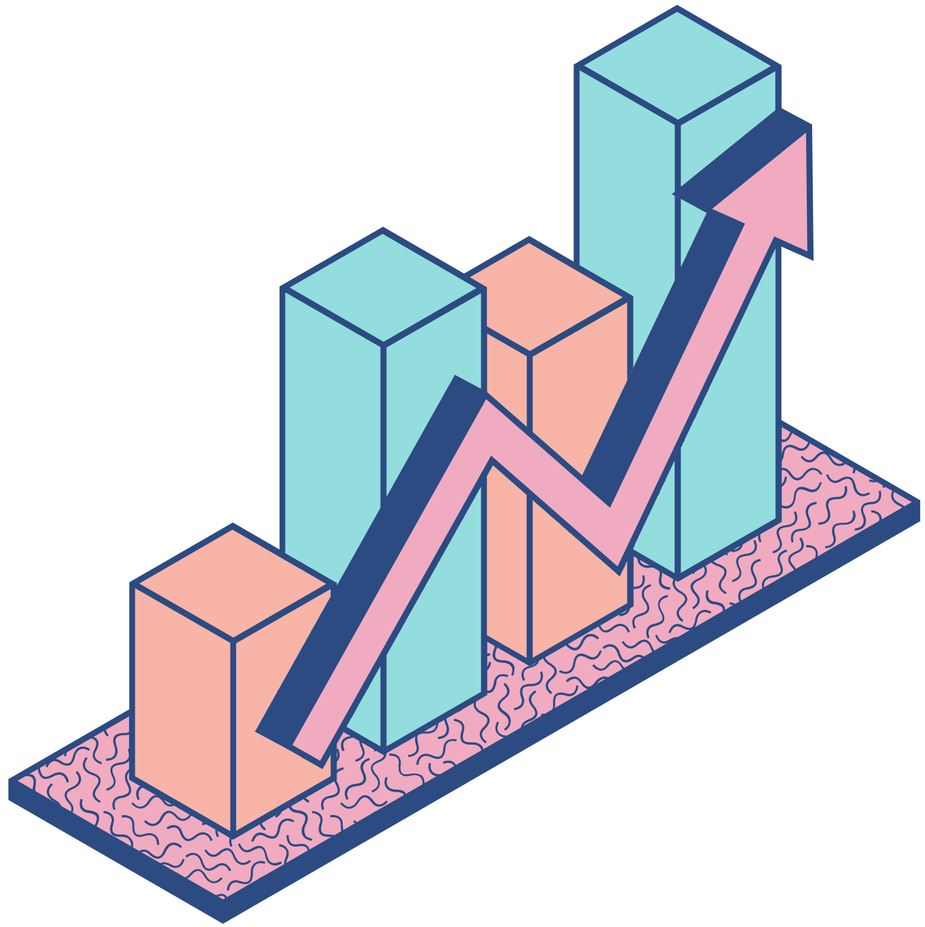
Stok Optimizasyonu Nedir?

Talepleri karşılamak ve aynı zamanda toplam stok maliyetlerini minimize etmek olarak tanımlanabilir. Şirketin lojistik operasyonları, depolama ve üretim süreçleri boyunca oluşacak stokların miktarının ve türlerinin zaman doğrultusunda belirlenmesi stok seviyesi optimizasyonunun başlıca konusudur.

Stok Optimize Edilirken,

- Talep Tahmini
- Stok Düzeyleri
- Tedarik Süreleri
- Maliyetler
- Optimum Sipariş Miktarı
- Trendler ve Değişkenler
- Riskler

Bu faktörleri göz önünde bulundurarak, stok yönetiminde verimliliği artırmak için doğru hesaplamalar yapılabilir ve stratejiler buna göre optimize edilebilir.



Veri Seti

Bir perakende şirketine ait açık kaynak e-satış verilerini üretim planlamaya uyarladık.
(Veri setini görüntülemek için buraya tıklayabilirsiniz.)

Tablo başlıkları ve kullanım şeklimiz:

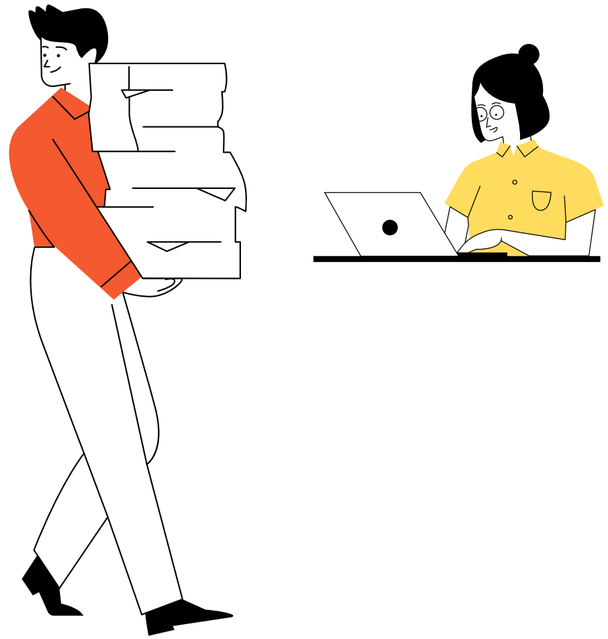
- ItemID: Ürün Kimliği
- year: Yıl
- WeekIdentifier: Hafta
- SalesChannel: Satış Kanalı
- Territory: Bölge
- NewBasePrice: Yeni Taban Fiyatı
- TotalInventory: Toplam Stok
- StoreInventory: Mağaza Stoku
- WarehouseInventory: Depo Stoku
- StockedStorePercentage: Stoklu Mağaza Yüzdesi
- SellingStoresRatio: Satılan Mağaza Oranı
- InboundInventory: Gelen Stok -->Satın Alınan Ürün Adedi
- SalesQuantity: Satış Miktarı --> Üretimde Kullanılan Adet
- ProjectedInventory: Tahmini Stok

Veri Toplama ve Hazırlama

- Fabrikaya ait ilgili verilerin toplanması,
- Eksik veya bozuk verileri tespit edilmesi ve düzeltilmesi
- Verileri uygun bir formata dönüştürülmesi ve modele girebilecekleri şekilde hazırlanması işlemleri gerçekleştirildi.

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import LocalOutlierFactor
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, mean_absolute_error
```

```
df = pd.read_csv("SampleSales.csv")
```



Veri İnceleme

Dataframe dönüştürdüğümüz dosyanın

- ilk 5 satırını, tüm sütunlarını,
- verinin istatistiksel olarak ifadesini,
- veri tipini,
- boş değer girilen hücre olup olmadığını

inceleyerek modelimize uygun olup olmadığını tespit ederek modelimize uygun hale getirdik.

```
def veri_inceleme(df):
    print("\nDataframe'in ilk 5 tanesi")
    print(df.head())
    print("\n\nDataframe'in columnları")
    print(df.columns)
    print("\n\nDataframe hakkında istatistiksel bilgi")
    print(df.describe())
    print("/Column Sayısı:")
    print(df.shape)
    print("\n\nDataframe hakkındaki bilgileri verir")
    print(df.info())
    print("\n\nDataframe'deki boş girilen hücreleri gösterir")
    print(df.isnull().sum())
veri_inceleme(df)
```

Veri Ayıklama

Modelimizde kullanacağımız, tahminleme yaparken gereken sütunları seçtik:

- ItemID: Ürün isimlerinin yer aldığı kolon
- WeekIdentifier: Ürünün satın alındığı ya da üretimde kullanıldığı hafta
- Year: Ürünün satın alındığı ya da üretimde kullanıldığı tarih
- TotalInventory: Toplam stoklanan ürün adedi
- SalesQuantity: Üretimde kullanılan adet
- InboundInventory: Ürün için eklenen stok
- ProjectedInventory: Ürünün stok tahmini

```
df=df[['ItemID',"year","WeekIdentifier","NewBasePrice",
      "TotalInventory","SalesQuantity","InboundInventory","ProjectedInventory"]]
```

Kategorik Veriyi Sayısal Veriye Dönüştürme

- Verinin türünü inceledikten sonra girilen kategorik veriyi integera çevirerek makinenin anlayacağı hale getirdik.

```
def data_converter(df):
    label_encoder = LabelEncoder()
    for column in df.columns:
        if df[column].dtype == 'object':
            df[column] = label_encoder.fit_transform(df[col
data_converter(df)
print(df)
```

Korelasyon ile Veriler Arası Bağlantıyı Bulma

- Veriler arasındaki ilişkiyi inceleyerek hangi kolonların modelimiz için gerekli hangilerine ihtiyaç olup olmadığının tespitini yaptık.

```python
def korelasyon_inceleme(df):
    # Korelasyon matrisini hesaplama
    correlation_matrix = df.corr()
    highest_correlations = correlation_matrix.unstack().sort_values(ascending=False)
    highest_correlations = highest_correlations[(highest_correlations > 0) & (highest_correlations < 1)]
    print("En Yüksek Korelasyonlar:")
    print(highest_correlations)

    # Korelasyon matrisini görüntüleme
    print("\nKorelasyon Matrisi:")
    print(correlation_matrix, "\n\n")
    correlation_matrix['ProjectedInventory'].sort_values(ascending=False)
korelasyon_inceleme(df)
print(df)
```

Aykırı Değerlerin Düzeltilmesi

.

- Girilen aykırı değerlerin (örneğin hafta bilgisine negatif değer girilmesi gibi) tespiti yapıldıktan sonra bu değerler yerine kolona ait medyan değerini atadık.

```python
def outlier_inceleme(df):
    # Local Outlier Factor (LOF) modelini oluşturma
    lof_model = LocalOutlierFactor(n_neighbors=20, contamination=0.1)

    # LOF modelini veri setine uygulama
    outlier_scores = lof_model.fit_predict(df)

    # Aykırı değerlerin indekslerini alma
    outliers_index = df.index[outlier_scores == -1]

    # Aykırı değerleri medyan değer ile değiştirme
    df_corrected = df.copy()
    for column in df.columns:
        mean_value = df[column].median()
        df_corrected.loc[outliers_index, column] = mean_value

    # Aykırı değerleri düzeltilmiş veriyi görüntüleme
    print("Aykırı Değerler Düzeltildi:")
    print(df_corrected.loc[outliers_index])
    return df_corrected
outlier_inceleme(df)
```

Standartlaştırma

- Daha iyi sonuç alabilmek için elimizdeki verileri 0-1 aralığına sıkıştırarak makinenin bunu daha kolay anlamasını sağladık.

```
def scaler_uygulama(df):
    # Standartlaştırma işlemi için StandardScaler kullanma
    scaler = StandardScaler()

    num_columns = ['ItemID',"year","WeekIdentifier","NewBasePrice",
       "TotalInventory","SalesQuantity","InboundInventory","ProjectedInventory"]
    scaled_features = scaler.fit_transform(df[num_columns])

    # Standartlaştırılmış veriyi DataFrame'e dönüştürme
    df_scaled = pd.DataFrame(scaled_features, columns=['ItemID_scaled',"year_scaled","WeekIdentifier_scaled","NewBasePrice_scaled",
       "TotalInventory_scaled","SalesQuantity_scaled","InboundInventory_scaled","ProjectedInventory_scaled"])

    # Standartlaştırılmış veriyi görüntüleme
    print("Standartlaştırılmış Veri:")
    print(df_scaled.head())
    df = pd.concat([df, df_scaled], axis=1)
    df = df.drop(num_columns, axis=1)
    return df
scaler_uygulama(df)
```

Lineer Regresyon ile ML

```python
# Lineer regresyon modeli oluşturma ve eğitme
from sklearn.linear_model import LinearRegression


def linearRegression(df):
    X = df.drop(columns=['ProjectedInventory'])
    y = df["ProjectedInventory"]

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    model = LinearRegression()
    model.fit(X_train, y_train)

    # Modelin doğruluk oranlarını yazdırma
    train_accuracy = model.score(X_train, y_train)
    test_accuracy = model.score(X_test, y_test)

    print("\nTest seti doğruluk skoru:", train_accuracy)
linearRegression(df)
```

✓ 0.0s

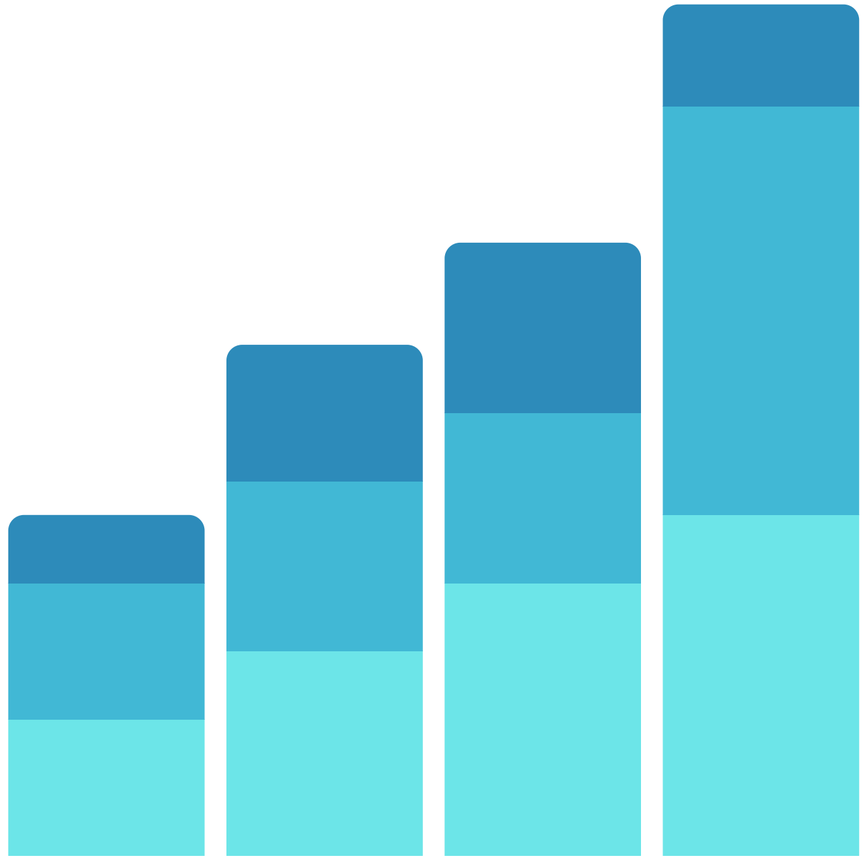
```
Test seti doğruluk skoru: 0.6928836202047786
```

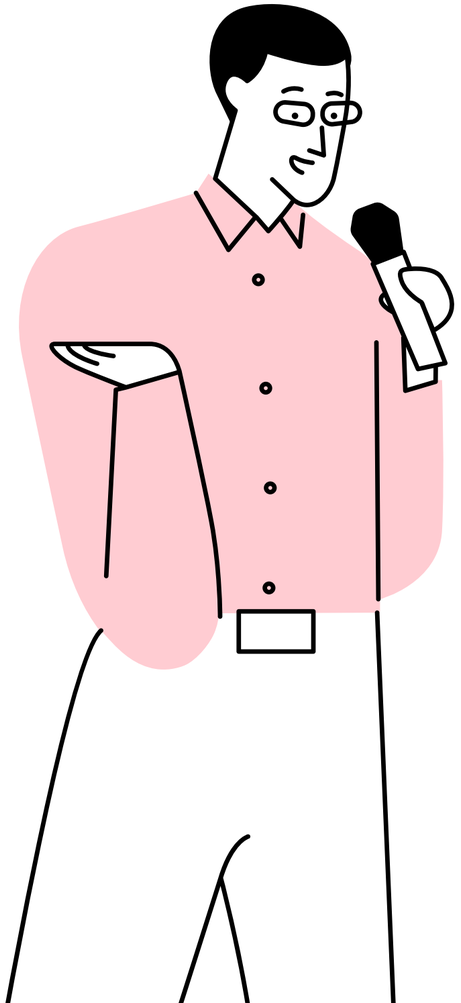
Zaman Serisi ARIMA ile Çözüm

```python
from statsmodels.tsa.arima.model import ARIMA
import pmdarima as pm

# Tarih sütunlarını datetime tipine dönüştürme
df['Date'] = pd.to_datetime(df['year'].astype(str) + '-W' + df['WeekIdentifier'].astype(str) + '-
# Tarih sütununu index olarak ayarlama
df.set_index('Date', inplace=True)

# Her ürün için ayrı modelleme
def forecast_Stock(item_id,prediction_date):
    try:
        product_df = df[df['ItemID'] == item_id]
        desired_date = pd.to_datetime(prediction_date)
        last_date = df.index[-1]
        steps_to_desired_date = (desired_date - last_date).days
        # Seçilen ürün için ARIMA modeli oluşturma
        model = ARIMA(product_df['SalesQuantity'], order=(5, 1, 0))
        model_fit = model.fit()

        # Tahmin yapma
        forecast_date = pd.to_datetime(prediction_date)
        forecast = model_fit.forecast(steps=steps_to_desired_date)
        exact_day_prediction = forecast[-1]
        print("Exact prediction",round(exact_day_prediction))

        #Hata fonksiyonu
        mse = mean_squared_error(df['SalesQuantity'].tail(steps_to_desired_date), forecast)
        print("Hata değeri",mse)

    except Exception as e:
        print(f"ARIMA model hatası: {e}")
```

Çıktı ve hata değeri

```
Exact prediction 5
Hata değeri 22.861738467236155
```





TEŞEKKÜRLER