



# **SALES PREDICT PROJECT**

ML TABANLI SATIŞ TAHMİNİ VE MÜŞTERİ  
SEGMENTASYONU API PROJESİ

# **Pair 6**

## **Ekibimiz**



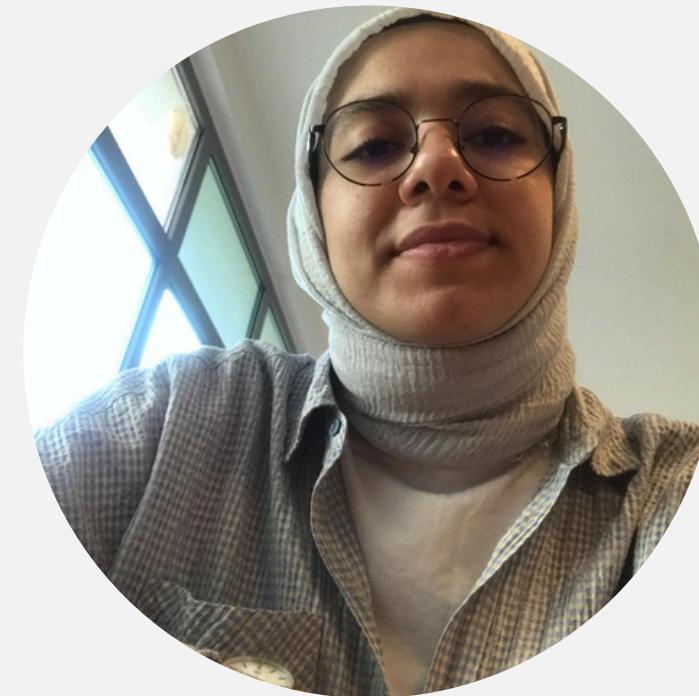
**BERNA UZUNOĞLU**



**KÜBRA UYAR**



**SEVDA UÇUM**



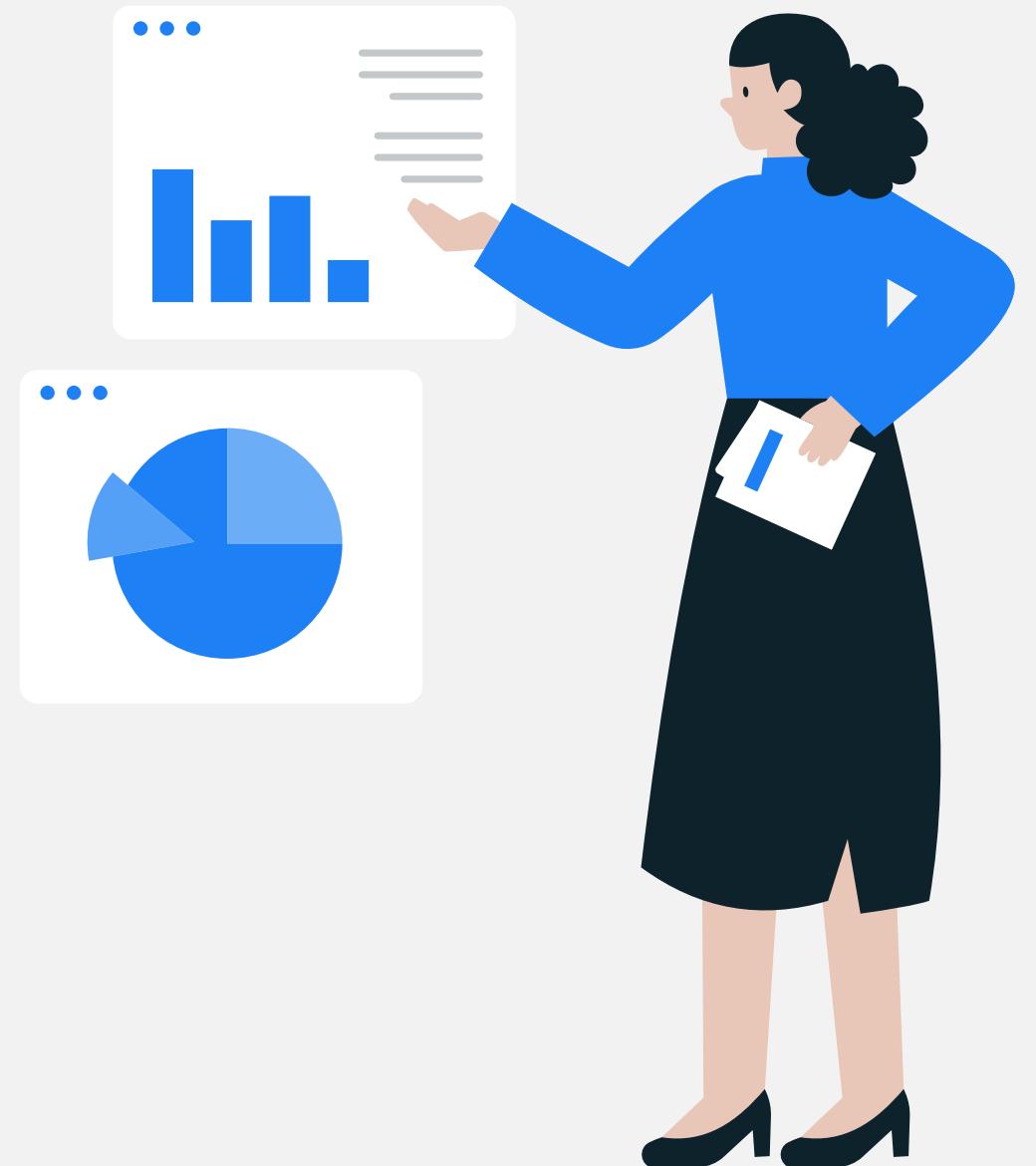
**ŞEVVAL MIKÇI**



**Z. MELİKE İŞİK**

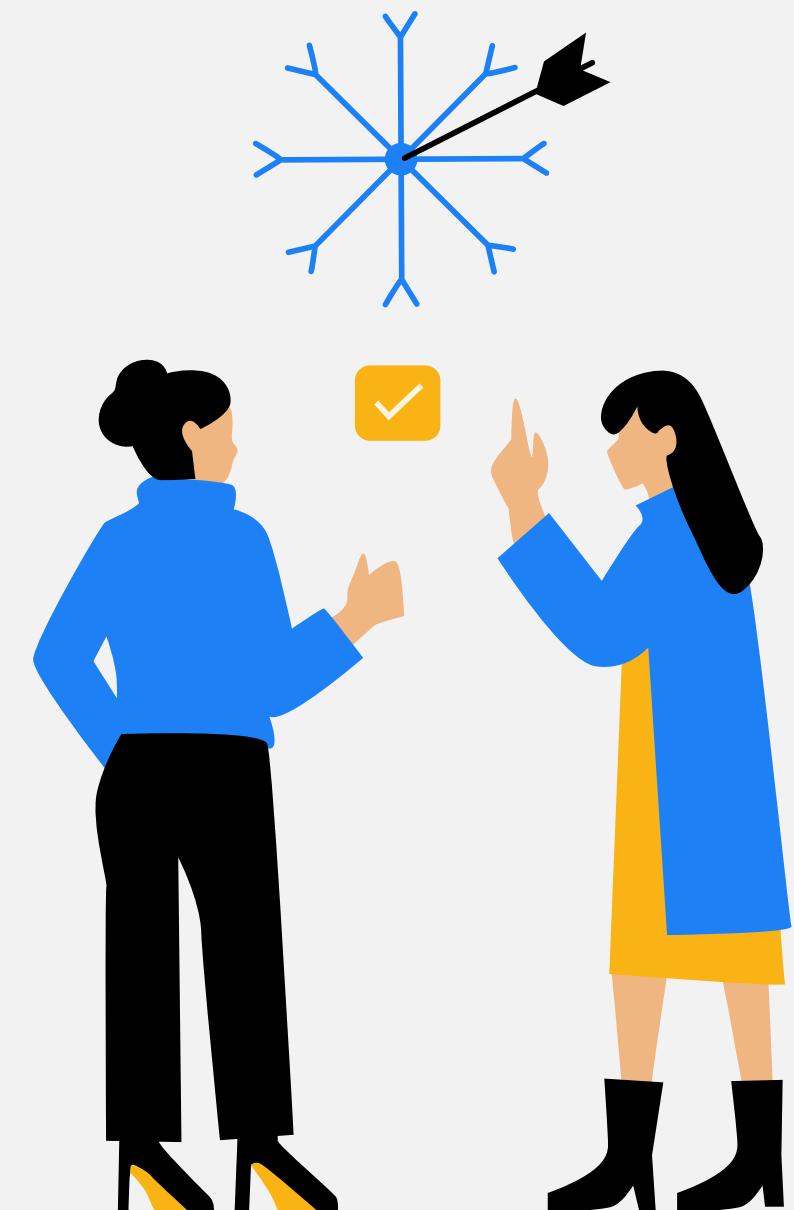
# Sales Predict Project

Makine öğrenmesi temelli bu proje, belirli ürünler için gelecekteki satış miktarlarını tahmin etmekte ve müşteri segmentlerini sınıflandırmaktadır. FastAPI kullanılarak geliştirilen bu RESTful API, ürün/satış verileri üzerinde analiz yapma ve tahmin üretme imkânı sunar.

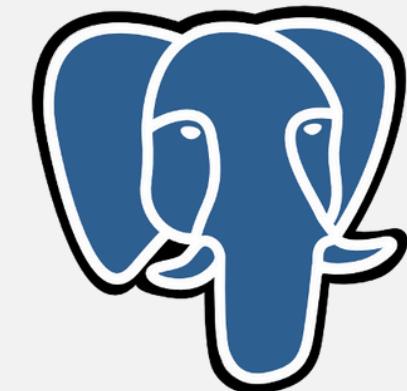


# Projenin Hedefleri

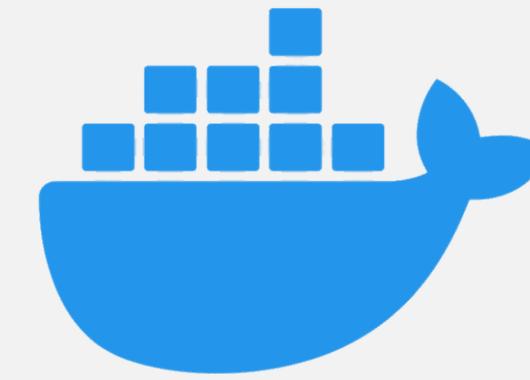
- Ürün bazlı satışları tahmin etmek
- Müşterileri davranışlarına göre segmente etmek
- Tüm bu işlemleri FastAPI tabanlı bir REST API ile sunmak
- Geliştirilen modeli Docker konteyneri içinde taşınabilir hale getirmek



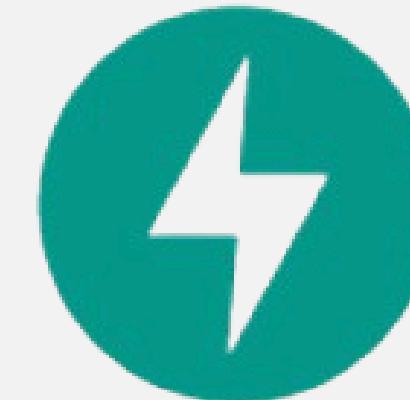
## Kullanılan Teknolojiler



PostgreSQL



docker®

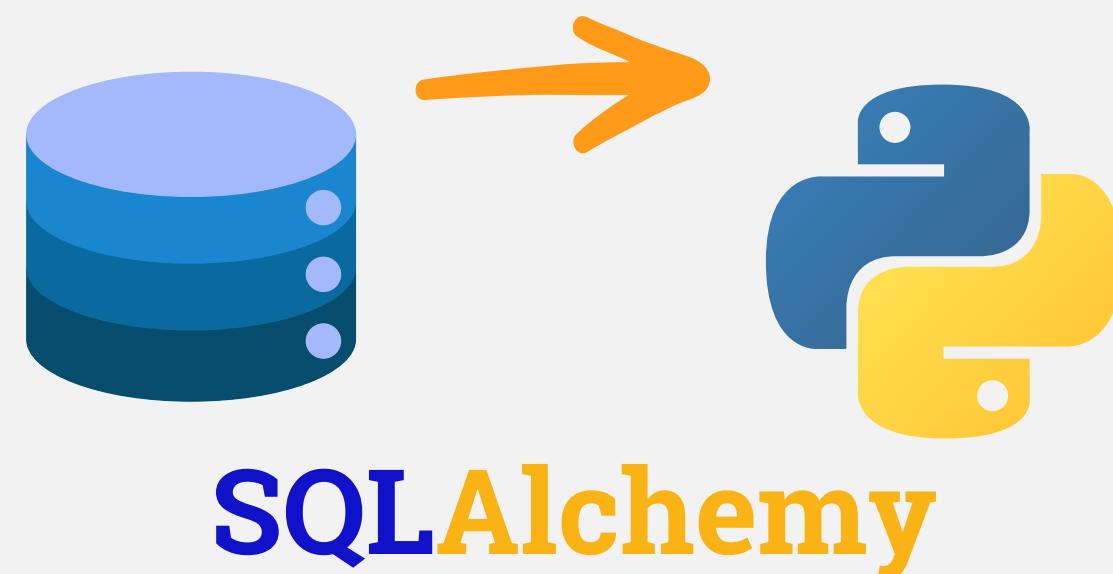


FastAPI



| Geleceği Yazanlar

# Veritabanı Bağlantısı ve Yönetimi: SQLAlchemy ile ORM Kullanımı



- Python ile veritabanı arasında köprü kuruldu.
- Veritabanı işlemleri Python sınıfları ve nesneleri aracılığıyla yönetildi
- Kodun okunabilirliği ve bakım kolaylığı sağlandı.

# SQLAlchemy ile Uygulama Entegrasyonu

**PostgreSQL ile Bağlantı Yönetimi:**

.env ve Config sınıfı ile yönetildi

**Model Yapısı ve Organize Edilmesi:**

models klasöründe her tablo için ayrı model dosyaları tanımlandı

**SessionLocal ile Güvenli Erişim:**

SessionLocal() ile FastAPI endpoint'lerinde verilere güvenli erişim sağlandı.

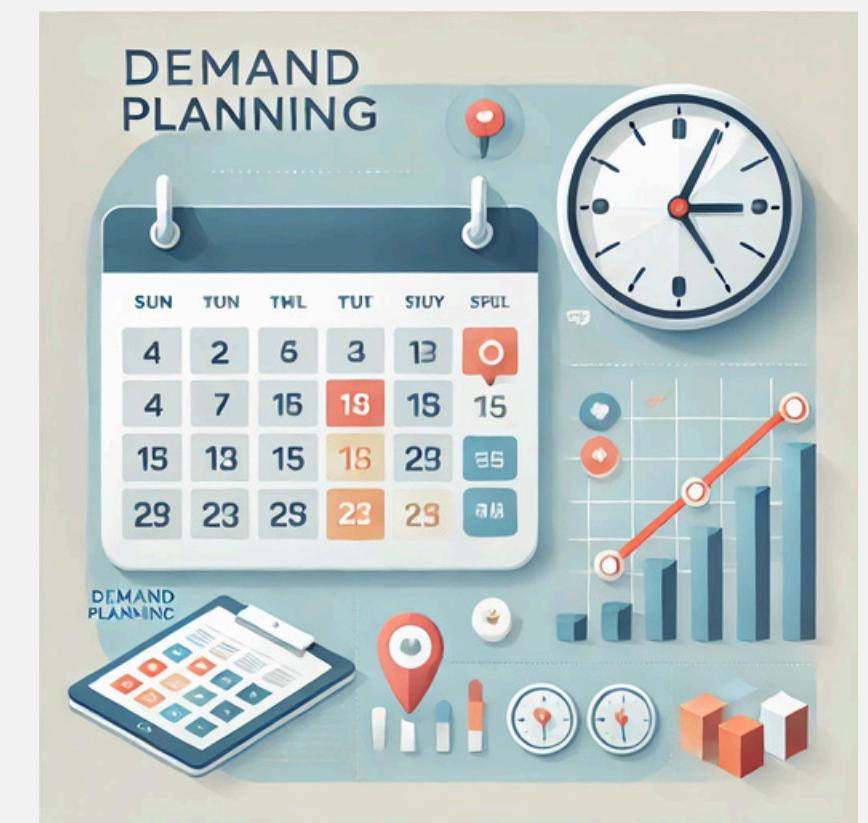


# Ürün Bazlı Satış Miktarı Tahmini

## İş Problemi Tanımı :

Ürün bazında satış tahmini neden gerekli?

- Stok yönetimi
- Talep planlaması
- Satış artışı gibi





## Veri Kaynağı

### Kullanılan Tablolar :

**Orders - Order\_Details - Products**

- order\_date
- product\_id
- quantity
- total\_sales



**sales\_forecasting\_data.csv**



# Özellik Mühendisliği - Feature Engineering

## Neden Özellik Mühendisliğine Başvurduk ?



**Problem:** "Model geçmiş satışlardan öğrenemiyordu"

- **Çözüm:** "**lag\_1 – lag\_14**: Geçmiş günlerin satış verileri"



**Problem:** "Mevsimsel dalgalanmalar bilinmiyordu"

- **Çözüm:** "**month, dayofweek, dayofyear**"



**Problem:** "Eğitim ve alışkanlıklar görünmüyordu"

- **Çözüm:** "**moving\_avg\_7, exp\_moving\_avg\_7**"

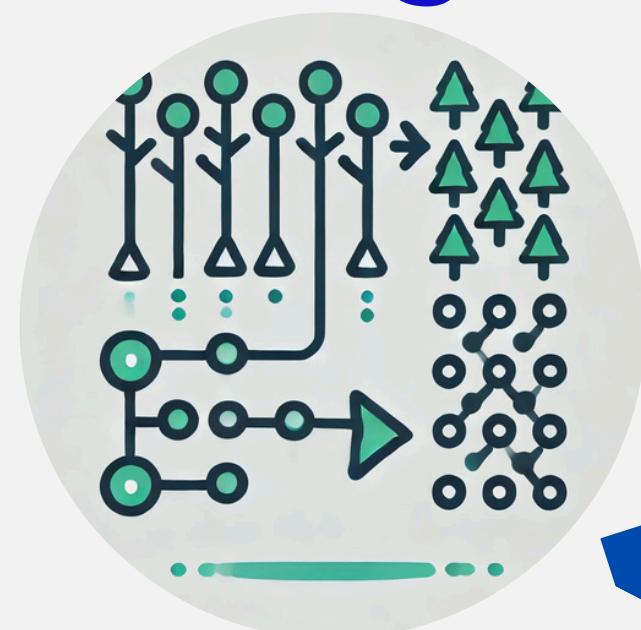


**Problem:** "Ürünün genel satış trendi bilinmiyordu"

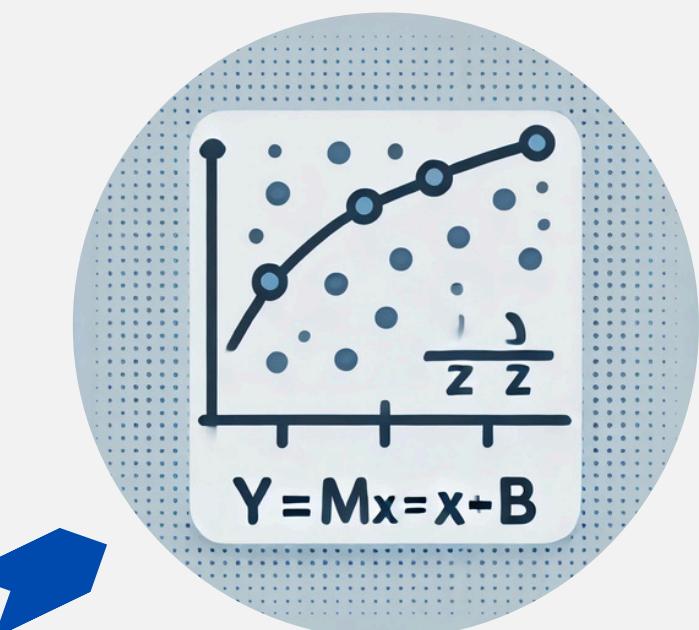
- **Çözüm:** "**cumulative\_sales, avg\_sales\_per\_day**"



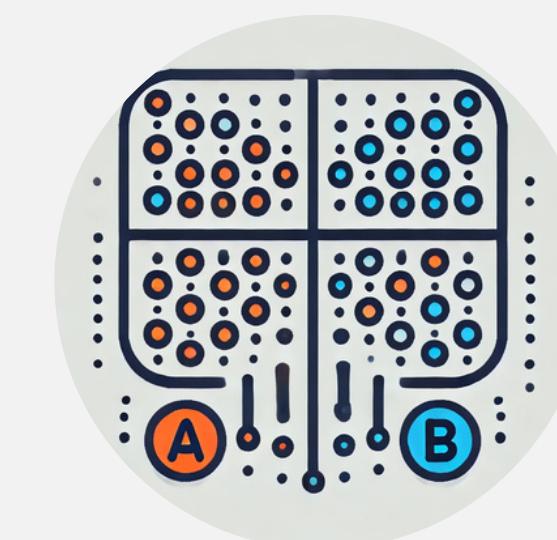
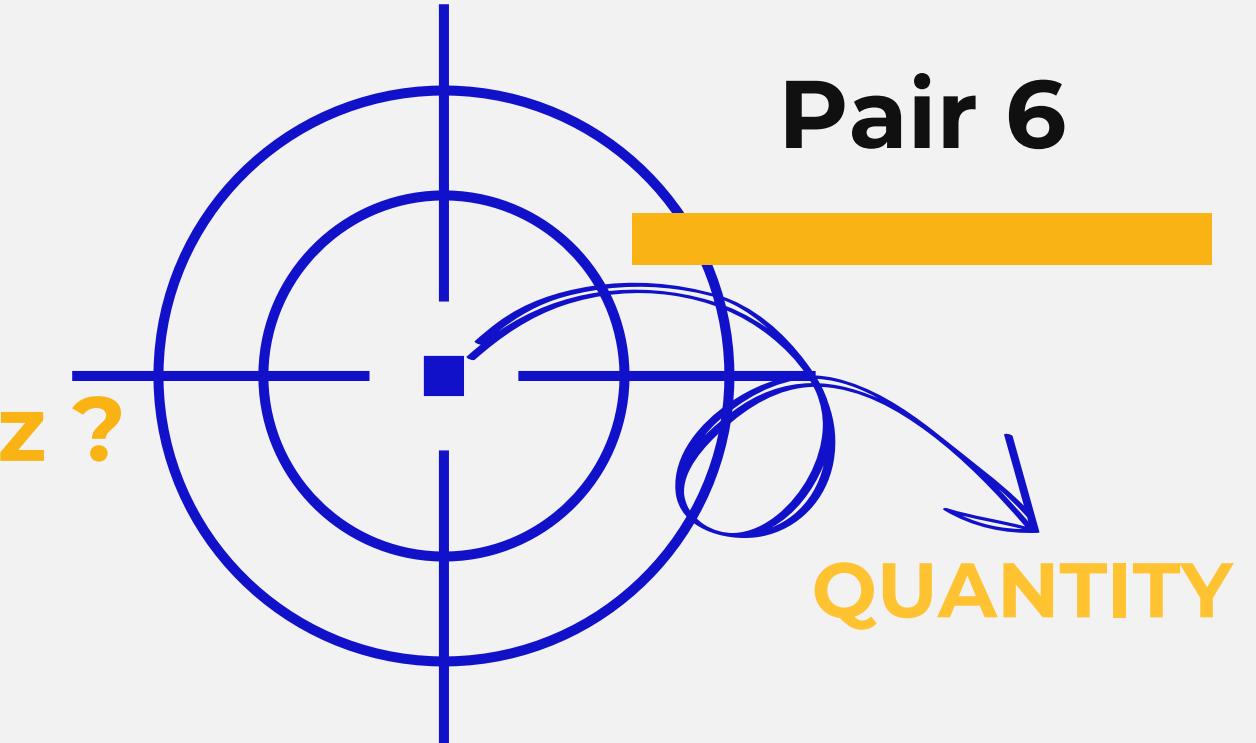
## Model Eğitimi: Hangi Modeli Neden Seçmeliyiz ?



RandomForestRegressor()



LinearRegression()



Sınıflandırma  
Problemleri



XGBRegressor()

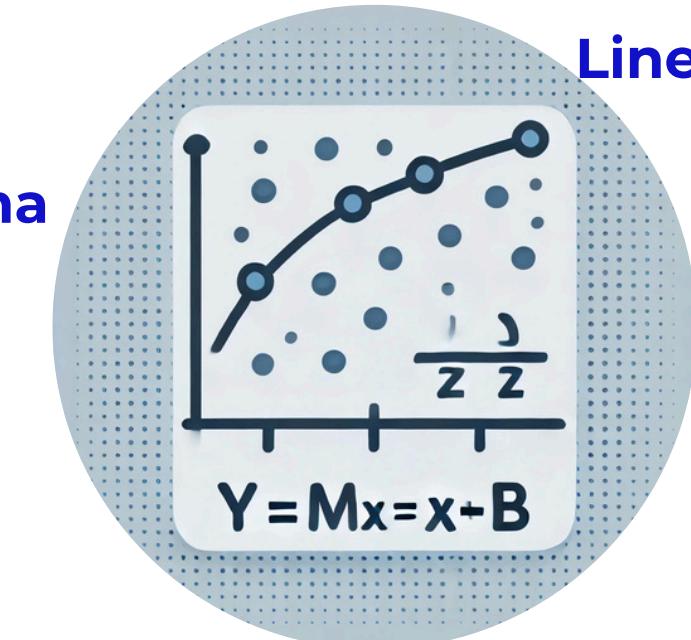
Regresyon  
Problemleri

## Modelin Kullanımı : Pipeline Yapısı



```
pipeline = Pipeline([
    ("feature_engineering", FunctionTransformer(create_features,
        kw_args={"ts_data": ts_data},
        validate=False)),
    ("model", LinearRegression())
])
```

pipeline ile taşıma



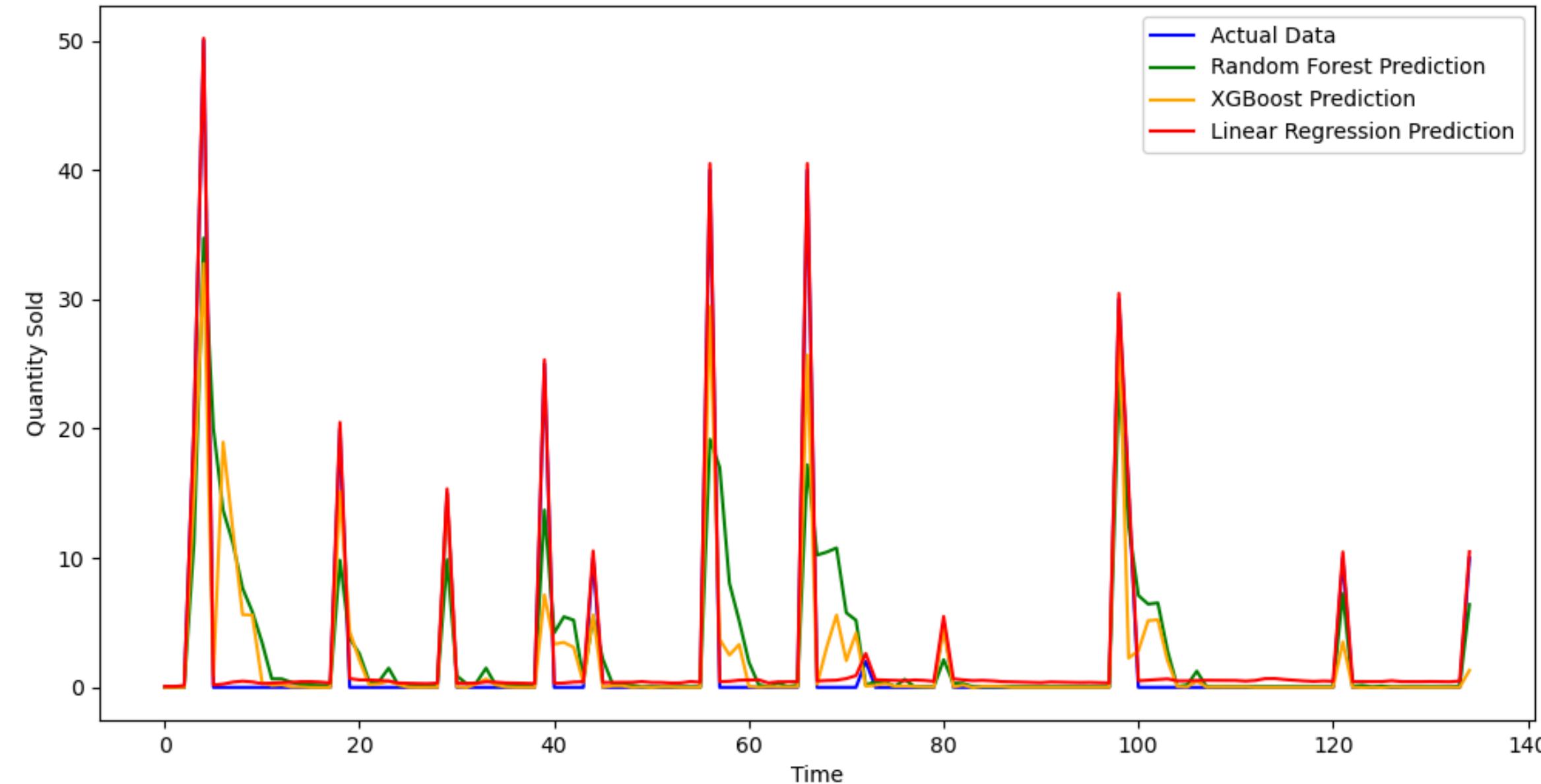
joblib



sales\_pipeline\_model.pkl

# Kullanılan Modellerin Gerçek Veriye Göre Yorumlanması

Sales Forecast for Product 11 - Model Comparison



## Model Performansı: Model Değerlendirme Metrikleri

### Model Çıktı Değerleri :

Random Forest **RMSE:** 5.068

XGBoost **RMSE:** 3.917

**Linear Regression RMSE:** 0.460

Random Forest Train **RMSE:** 1.246

XGBoost Train **RMSE:** 0.00077909

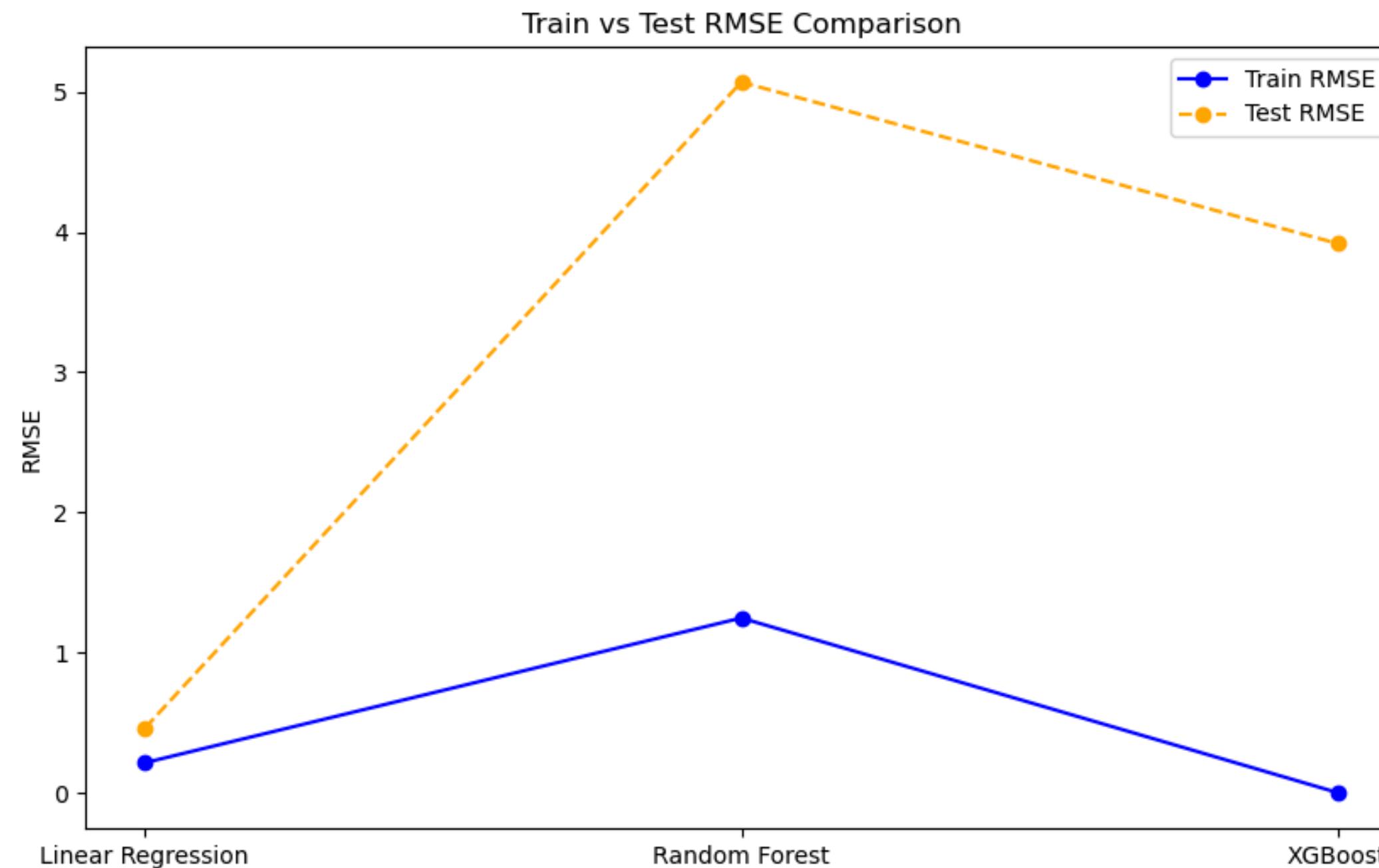
**Linear Regression Train RMSE:** 0.213

Random Forest Cross-Validation **RMSE:** 3.233

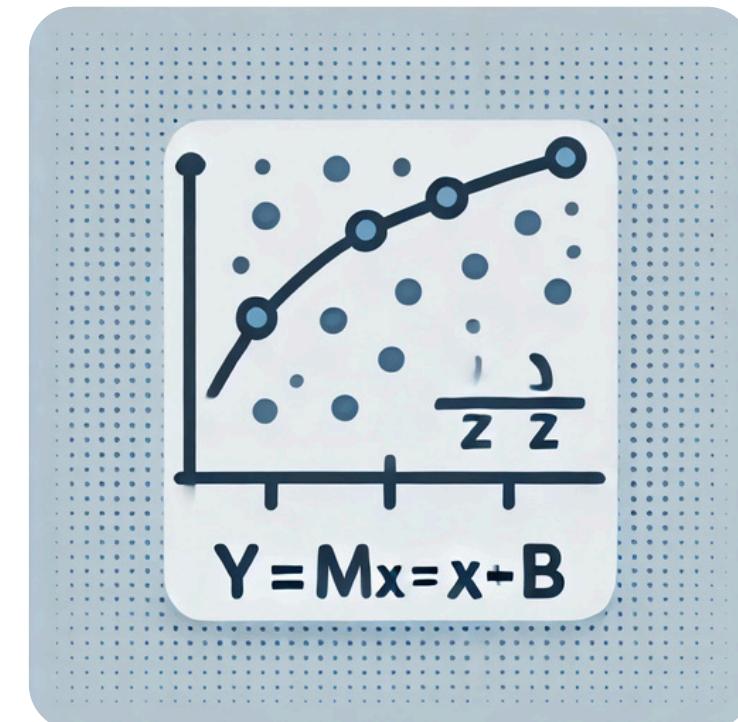
**Linear Regression Cross-Validation RMSE:** 0.6855



## Modellerde Aşırı Öğrenme Kontrolü



**SONUÇ Ve KARAR**



**Linear Regression**

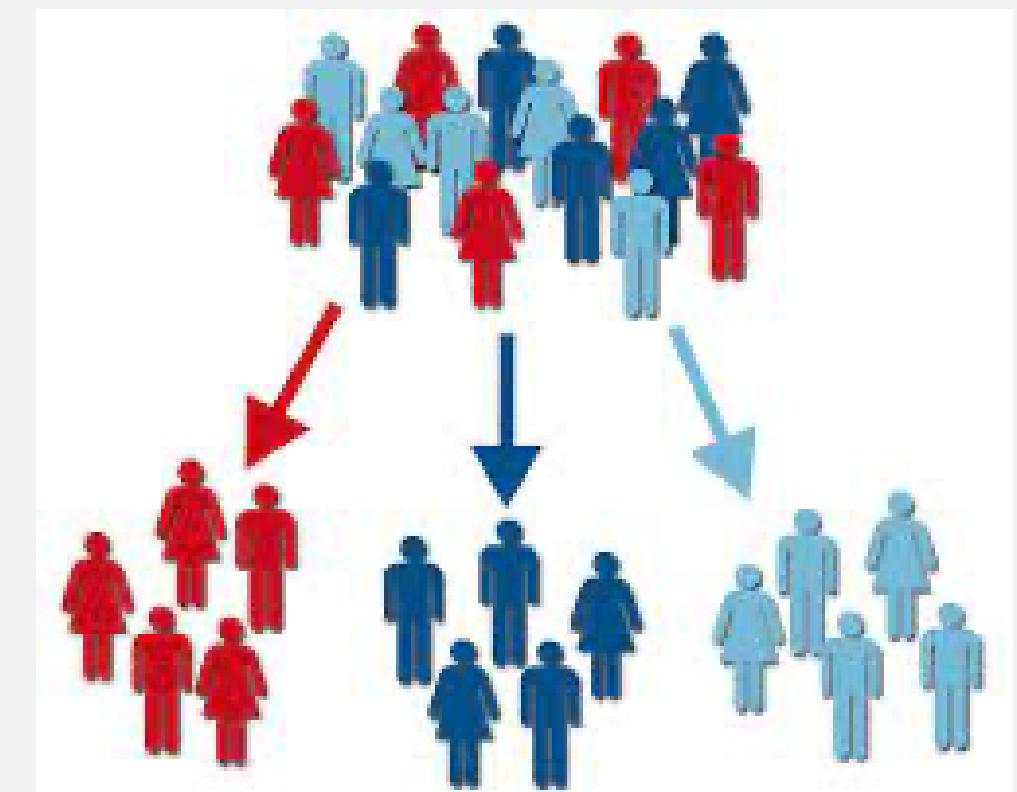
## Müşteri Segmentasyonu

Bu çalışmada müşterilerin satın alma davranışları incelenerek segmentlere ayrılması ve makine öğrenmesi ile ürün bazlı satış tahmin modeli geliştirilmiştir.

### Problem tanımı:

- Bu çalışma, müşterileri segmentlere ayırmak ve yeni müşteriyi doğru segmente atamak için yapılmıştır.
- Hedef: Segmentasyon + Tahminleme
- Müşteri sayısı arttıkça kişiselleştirilmiş yaklaşımalar zorlaşıyor.
- Ürün satış tahminleri, kaynak planlaması ve kampanya başarısı için kritik.

**Hedef:** Müşteri segmentasyonu ile davranış kümeleri oluşturmak ve her segmentin ürün satın alma eğilimini tahmin etmek.



## Veri Seti ve Özellikler

Kaynak: Northwind veritabanı

### Değişkenler:

Kolon Adı	Veri Tipi	Açıklama
customer_id	string	Müşteri kimliği
total_spent	float	Müşterinin toplam harcaması
num_orders	int	Toplam sipariş sayısı
avg_order_value	float	Ortalama sipariş tutarı
num_products	int	Toplam satın alınan ürün sayısı
recency	float	Son siparişten bu yana geçen gün sayısı
segment_id	int	Segment sınıf ID'si (etiket)

#### 🎯 Hedef Değişken:

**segment\_id: Müşterinin ait olduğu segment sınıfı (int)**



### Yöntem

**Kullanılan yöntem:** KMeans / KNN

**Normalizasyon:** MinMaxScaler

**Belirlenen segment sayısı:** 4 (KNN)

Elbow Yöntemi ile K değeri seçimi

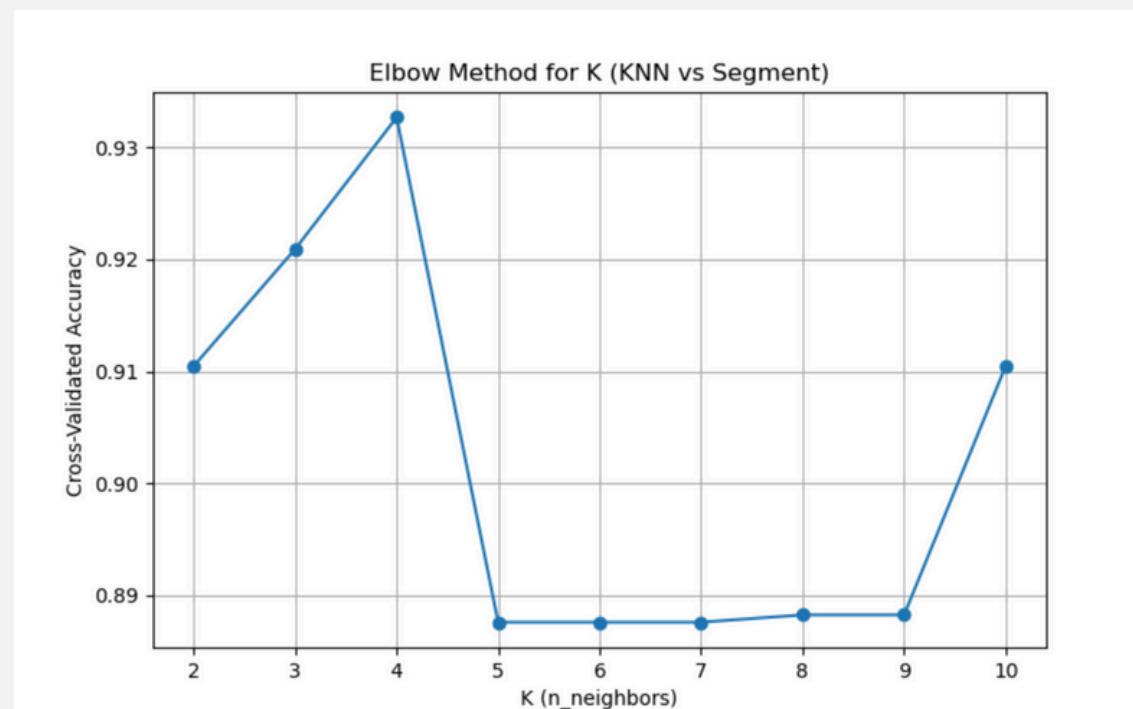
### Segment örnekleri:

Segment 0: "Unengaged",

Segment 1: "Potential Loyalists",

Segment 2: "Champions",

Segment 3: "Regulars"



Veri Akışı – Tahmin Süreci

API: POST /predict-segment

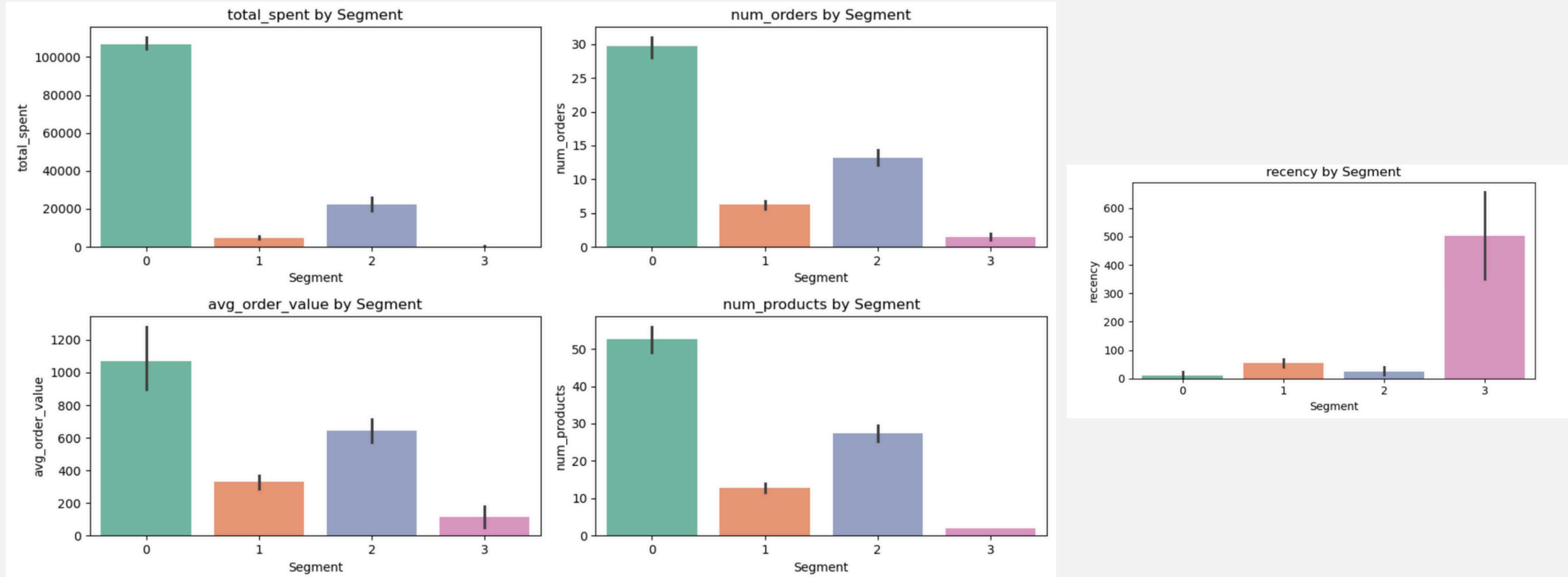
Customer Feature Extraction

Segmentasyon Modeli (KMeans/KNN)

Segment Tahmini: segment\_id & segment\_name

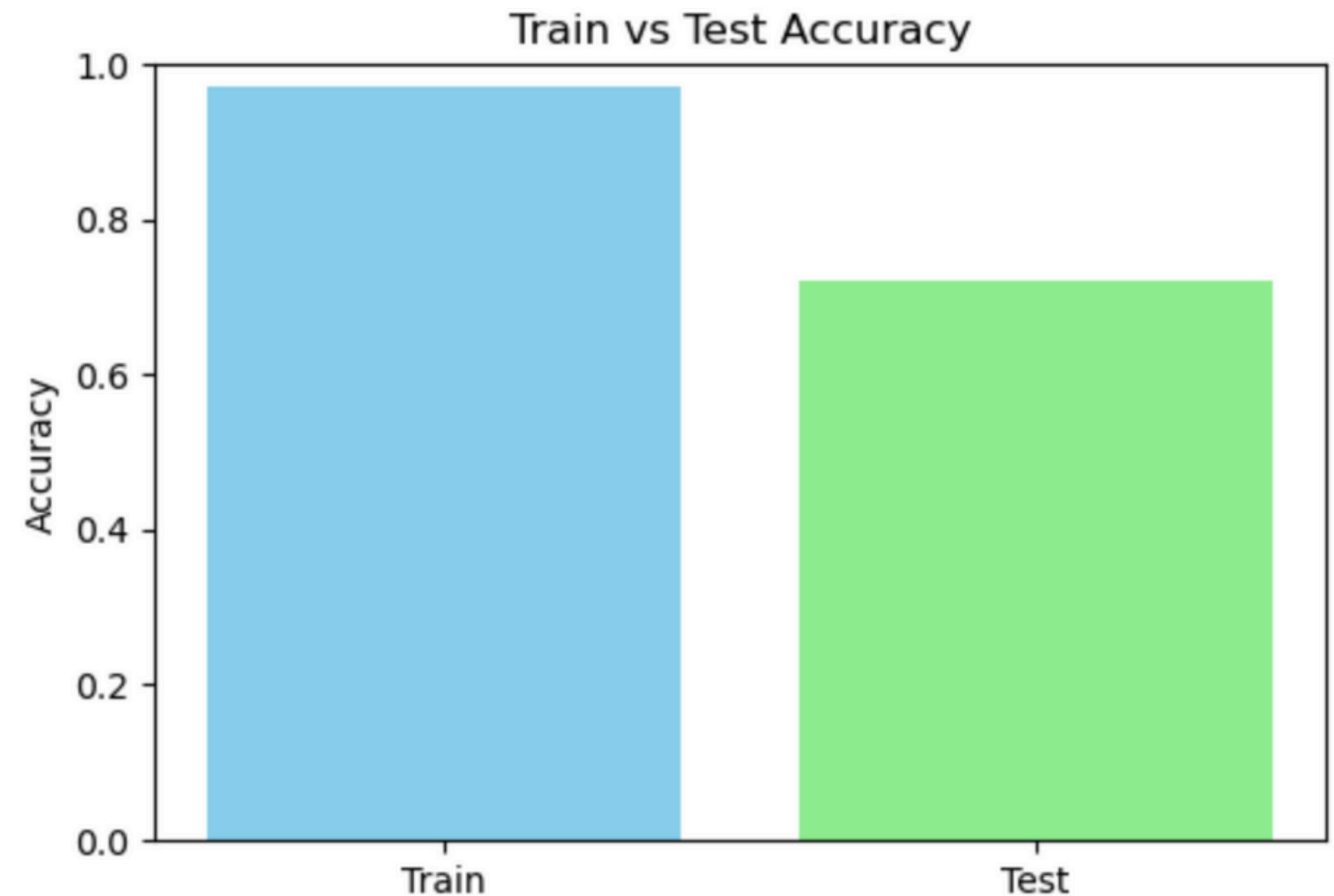
API Response

## Segmentlerin Karşılaştırılması



## Segment Tahmini Modeli (KNN) ve Sonuçları

- KNeighborsClassifier modeli:
- Hedef değişken = segment
- ✅ En iyi K değeri: 4
- Train-test split: 80-20
- ✅ Train Accuracy: 0.9718
- ✅ Test Accuracy: 0.7222
- ✅ Model Doğruluk Skoru: 0.7222



# KNN Model Performansı, Sonuç ve Öneriler

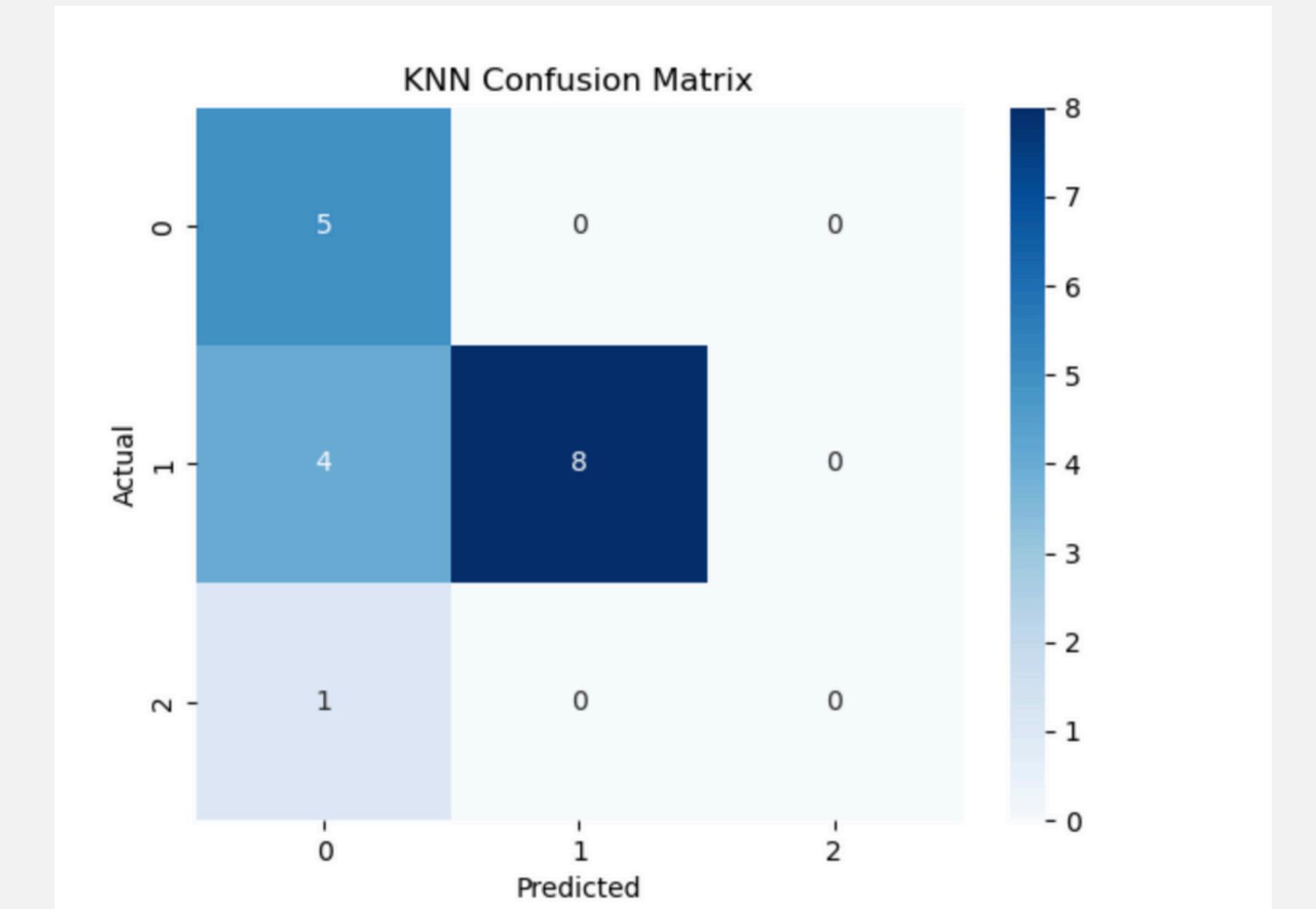
## Confusion Matrix:

🔍 Sınıflandırma Raporu: modelin hangi segmentleri ne kadar doğru tahmin ettiğini detaylı şekilde gösterir.

	precision	recall	f1-score	support
1	0.50	1.00	0.67	5
2	1.00	0.67	0.80	12
3	0.00	0.00	0.00	1

## Öneriler:

- Segment bazlı kampanya önerileri yapılabılır. Model API ile servise alınabilir
- Gelecekte: Zaman serisi, ürün önerisi, RFM analizleri



## KMeans Model Performansı ve Sonuç

### Inertia: 3.99

Verilerin kümelere olan uzaklıklarının toplamıdır. Daha düşük inertia, daha sıkı kümelenmiş segmentler anlamına gelir.

Bu değer, modelin segmentleri ne kadar iyi ayırdığını gösterir (ama tek başına yeterli değildir).

### Silhouette Score: 0.447

- Her bir müşterinin kendi segmentine olan benzerliği ile diğer segmentlerden farkını ölçer.
- 0 ile 1 arasında değer alır.
  - 0.44, orta düzeyde başarılı bir segmentasyon olduğunu gösteriyor.
  - Kümeler arasında belirgin bir ayrıım var, ancak daha iyi ayrıım için iyileştirme yapılabilir.

## FastAPI Nedir?

FastAPI, Python diliyle yazılmış, modern ve çok hızlı bir web API framework'üdür.

API: Uygulamalar veya sistemler arasında veri alışverişini sağlayan arayüz.

## Ne İçin Kullanılır?

⌚ Hızlı ve Aynı Anda Birden Fazla İstek Karşılama: FastAPI, aynı anda birçok kullanıcı isteğini bekletmeden işleyebilir.

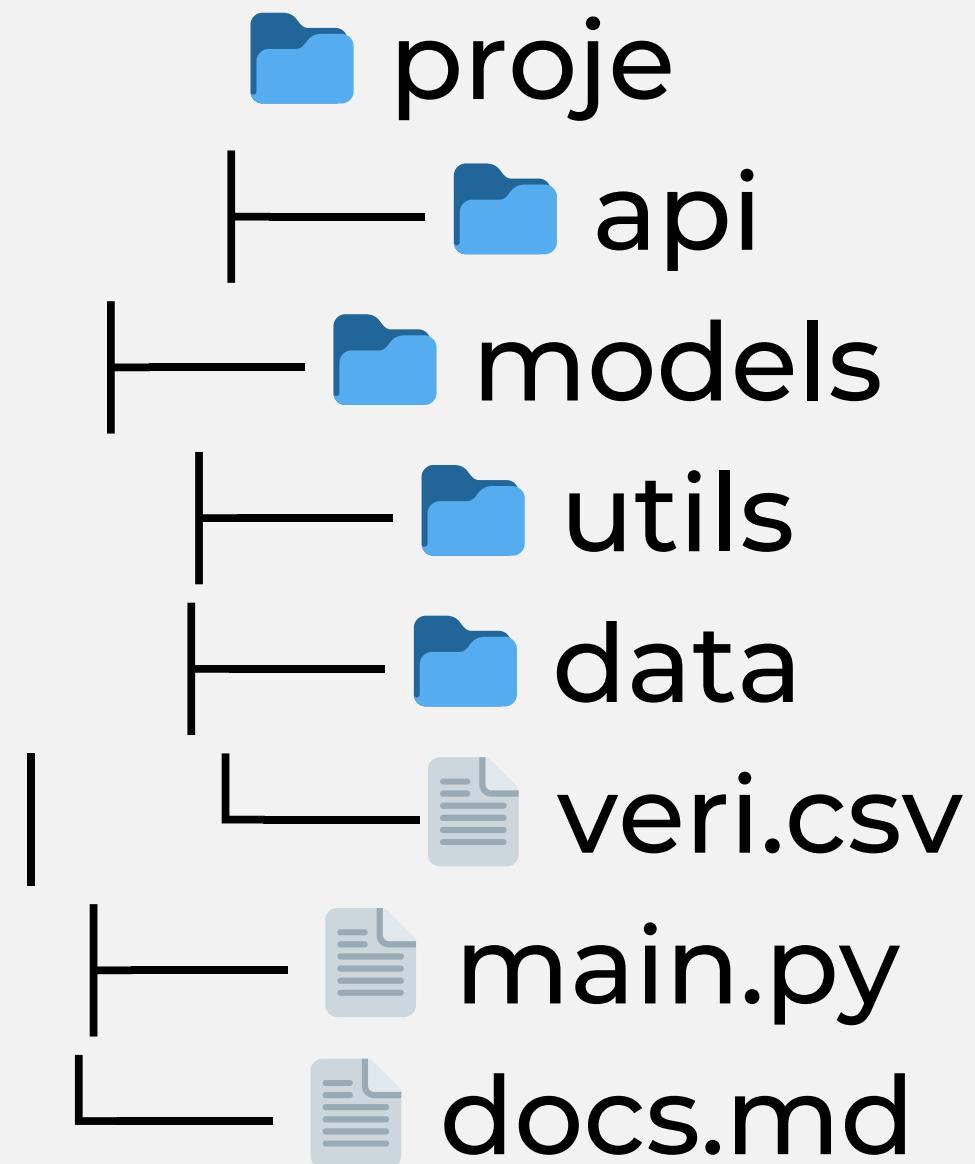
🧠 Otomatik Tip Kontrolü ve Validasyon (Pydantic ile): Gelen veriler otomatik olarak kontrol edilir.

📄 Otomatik API Dokümantasyonu: Yazdığın koddan otomatik olarak Swagger UI (/docs) ve ReDoc (/redoc) sayfaları oluşur.



## Proje Klasör Yapısı

- api/ → endpoint router dosyaları
- models/ → eğitilen model ve feature engineering
- utils/ → yardımcı fonksiyonlar (örneğin load\_model)
- data/ → veri dosyaları (CSV)
- main.py → uygulamanın başlatıldığı yer
- docs.md → API dokümantasyonu



## Endpoint Nedir

- Endpoint, bir API'nin dış dünyaya sunduğu belirli bir adres (URL) ve işlemidir.
- Yani kullanıcıların ya da uygulamaların, API üzerinden veri almak, veri göndermek ya da bir işlem yapmak için eriştiği noktalardır.

Endpoint, API'nin hangi işlemleri sunduğunu gösteren URL + işlem (HTTP metodu) kombinasyonudur.

GET /products	Ürün listesini getir
POST /predict	Satış tahmini yap

# Dokümantasyon

Yazdığınız her API otomatik olarak belgelendirilir.

- /docs adresinden Swagger UI ile test yapılabilir.
- /redoc adresinden sade ReDoc arayüzüyle belgeler görüntülenebilir.

The screenshot shows the Swagger UI interface for the 'ML Based Sales Prediction API (1.0.0)'. The left sidebar lists several endpoints: Health Check, Prediction, Products, Sales Summary, Segmentation, and Retrain. The 'Health Check' endpoint is expanded, showing its detailed documentation. The main content area displays the 'Health Check' endpoint's details, including its description ('A FastAPI project for predicting sales using trained models.'), a 'Download OpenAPI specification' button, and a 'Responses' section. The 'Responses' section includes a green bar indicating a successful response (200).

## ML Based Sales Prediction API 1.0.0 OAS 3.1

/openapi.json

A FastAPI project for predicting sales using trained models.

### Health Check

GET /health/ Health Check

### Prediction

POST /predict/ Predict

### Products

GET /product/ List Products

### Sales Summary

## Veri Doğrulama (Validasyon)

FastAPI, Pydantic kütüphanesi sayesinde kullanıcıdan gelen verilerin doğru formatta olup olmadığını otomatik olarak kontrol eder.

Yanlış veri girildiğinde anlaşılır hata mesajları döner.

**Prediction**

**POST** /predict/ Predict

**Parameters**

No parameters

**Request body** required

application/json

```
{  
    "product_id": 1000,  
    "year": 2024,  
    "month": 5,  
    "day": 1  
}
```

Error: Bad Request

ted

Response body

```
{  
    "detail": {  
        "error_code": 1001,  
        "error_message": "ProductId bulunamadı."  
    }  
}
```



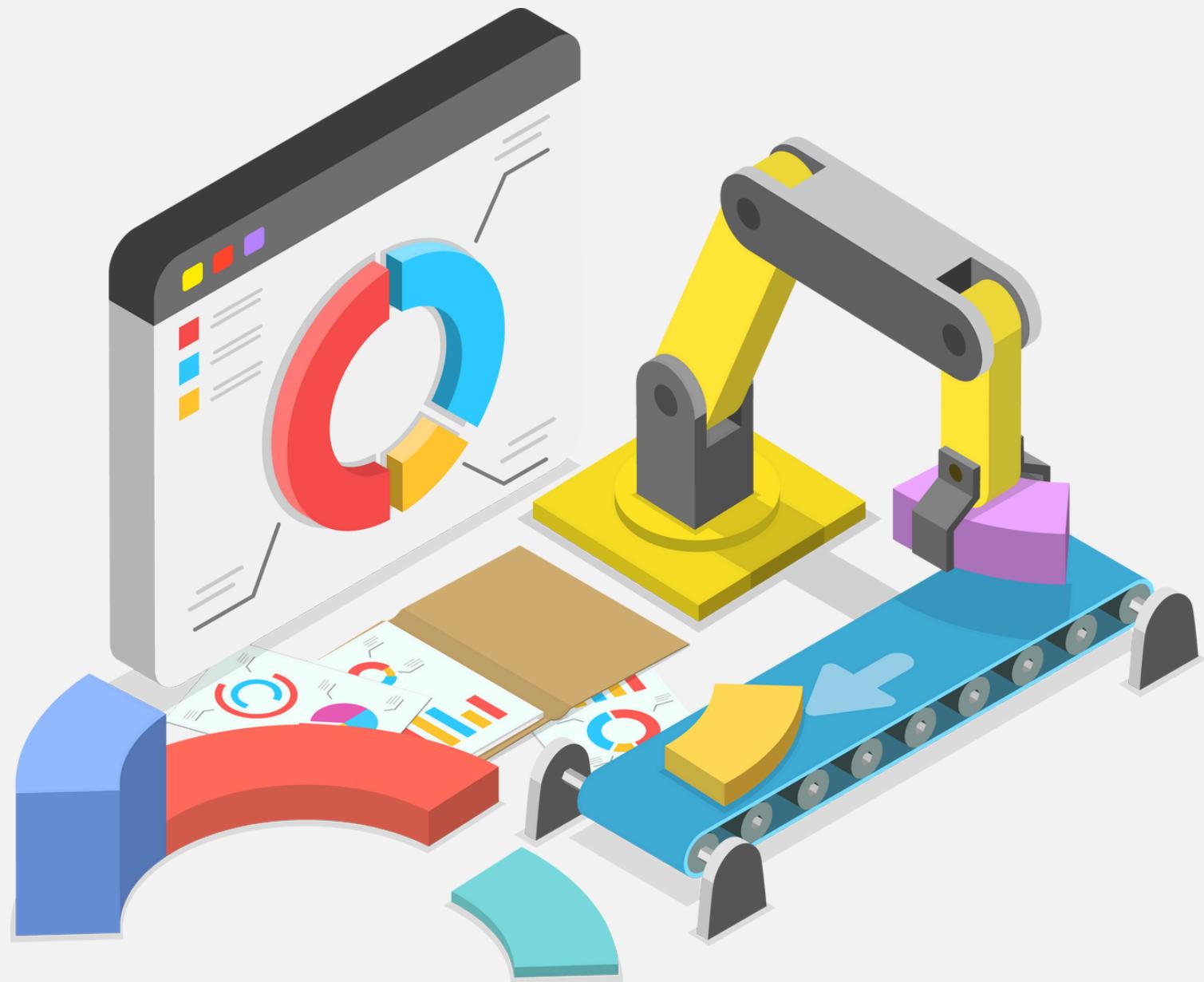
Download



| Geleceği Yazanlar

## Test ve Dağıtım Sürecinin Amacı

- Amaç uygulamanın doğru çalıştığını doğrulamak ve kullanıcıların erişebileceği bir ortamda çalıştmak.
- **Kapsam:** API'nin test edilmesi, hata yönetimi, bağımlılıkların yönetimi ve Docker ile konteynerleştirme.



## Test Süreci

### API Endpoint'lerinin Test Edilmesi

- API endpoint'lerini test etmek için Postman veya Swagger kullanıyoruz:
  - Swagger arayüzü üzerinden **/predict** gibi endpoint'lere örnek istekler gönderiyoruz.
  - Postman ile JSON formatında veri göndererek API'nin doğru yanıt verip vermediğini kontrol ediyoruz.

## Örnek Talepler Gönderilmesi

- **/predict** endpoint'ine şu formatta bir POST isteği gönderiyoruz:

```
{  
  "product_id": 50,  
  "year": 2025,  
  "month": 4,  
  "day": 7  
}  
  
{  
  "product_id": 50,  
  "predicted_quantity":  
    2.28278516151554486  
}
```

## Hata Yönetimi ve Validasyon

- API'nin güvenilir çalışması için hata yönetimi ve validasyon mekanizmaları ekledik.

```
{  
  detail': {  
    "error_code": 1001,  
    "error_message":  
      "ProductId bulunamadi.  
  }  
}
```

- Örneğin, geçersiz bir product\_id gönderildiğinde anlamlı bir hata mesajı döner.
- API'de ürün ID'si doğrulaması, sistem bütünlüğünü korumak ve kullanıcı deneyimini optimize etmek açısından çok önemlidir.

## Bağımliliklerin Yönetimi

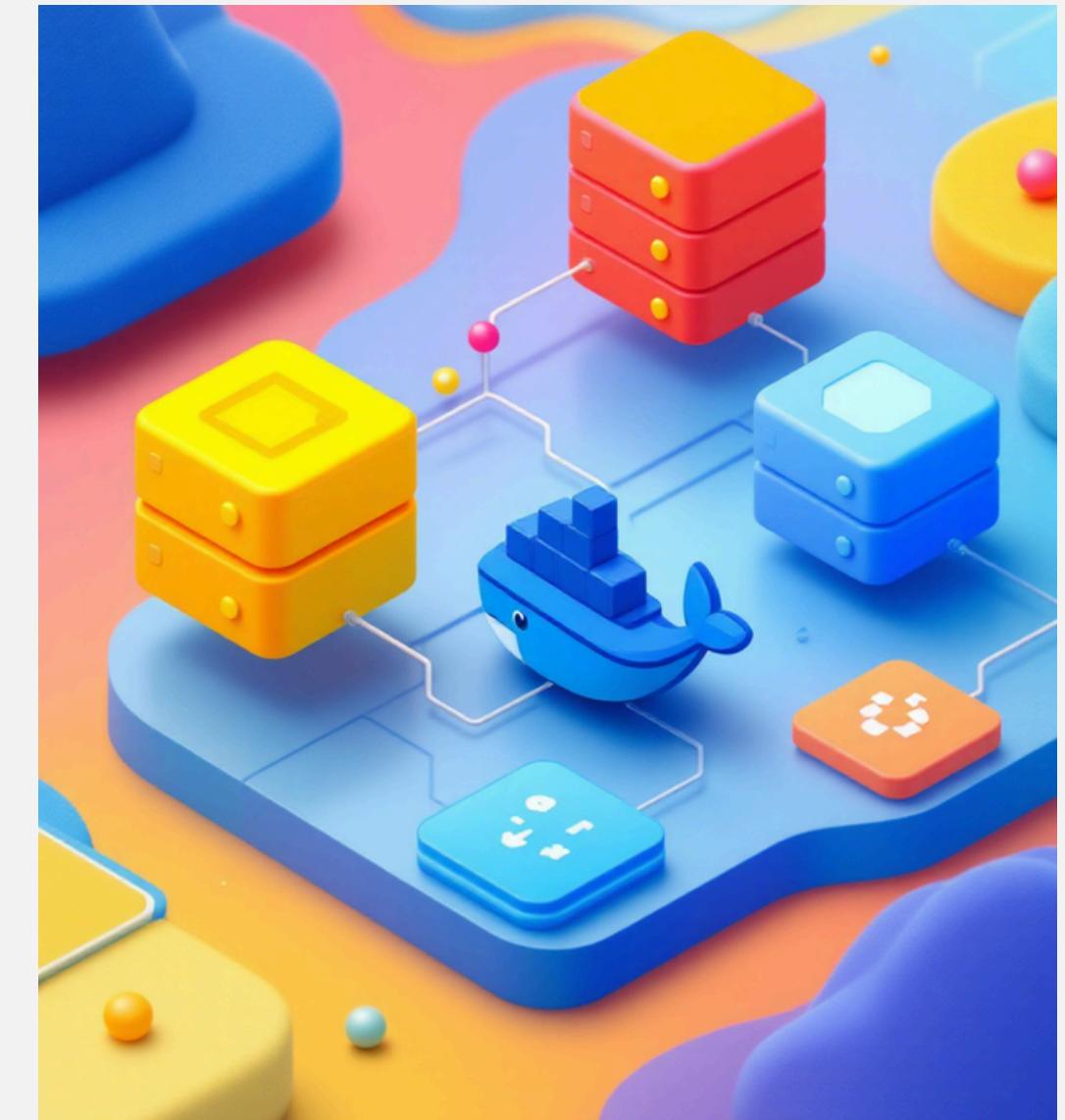
- Projenin bağımlılıklarını **requirements.txt** dosyasında tanımladık.
- Bu dosya, projeyi başka bir ortamda çalıştırılmak için gerekli tüm Python paketlerini içerir.

```
pip install -r requirements.txt
```

python-dotenv== 0.21.0  
sqlalchemy== 2.0.37  
psycopg2  
pandas==2.2.3  
numpy==1.26.4  
scikit-learn==1.6.1  
xgboost==1.7.6  
reicorn  
pydantic==2.10.3  
matplotlib==3.10.0  
joblib==1.4.2  
fastapi==0.112.2  
seaborn==0.13.2

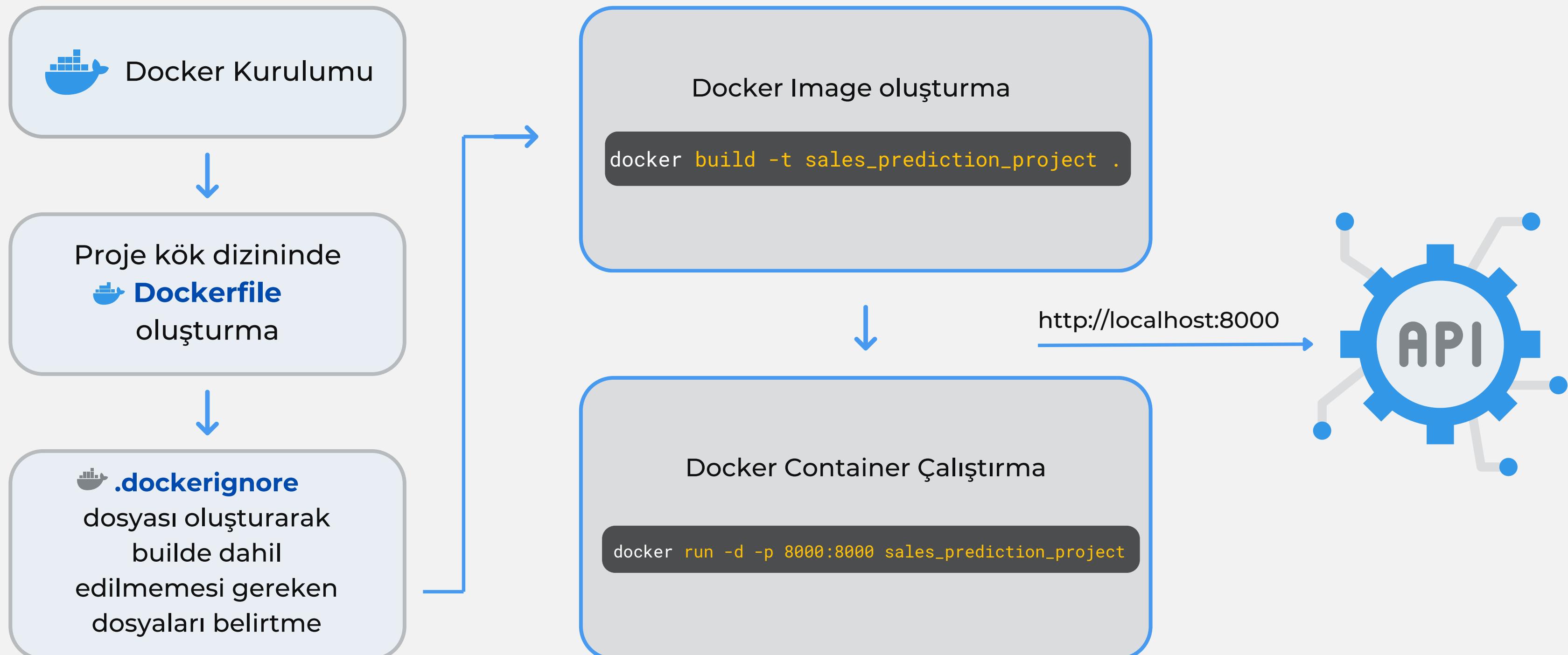
# Docker Nedir ve Neden Kullanıyoruz?

- Docker, uygulamaları ve bağımlılıklarını bir arada çalıştırmak için kullanılan bir konteyner platformudur.
- Projemizde Docker kullanarak:
  - Uygulamayı her ortamda aynı şekilde çalıştırabiliyoruz.
  - Bağımlılık sorunlarını ortadan kaldırıyoruz.
  - Hızlı ve kolay bir şekilde dağıtım yapabiliyoruz.



# Docker ile Konteynerleştirme

Pair 6





| Geleceği Yazanlar

**TEŞEKKÜRLER**