

**BURSA TEKNİK ÜNİVERSİTESİ**  
**MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ**



**Makine Öğrenmesi Temelli Gerçek Zamanlı  
Akıllı Oda Sistemi**

**LİSANS BİTİRME ÇALIŞMASI**

**Selçuk ŞAN**

**Bilgisayar Mühendisliği Bölümü**

**HAZİRAN, 2023**

**BURSA TEKNİK ÜNİVERSİTESİ**  
**MÜHENDİSLİK VE DOĞA BİLİMLERİ FAKÜLTESİ**



**MAKİNE ÖĞRENMESİ TEMELLİ GERÇEK ZAMANLI  
AKILLI ODA SİSTEMİ**

**LİSANS BİTİRME ÇALIŞMASI**

**Selçuk ŞAN**  
**18360859049**

**Bilgisayar Mühendisliği Bölümü**

**Danışman: Dr. Öğr. Üyesi Volkan ALTUNTAŞ**

**HAZİRAN, 2023**

BTÜ, Mühendislik ve Doğa Bilimleri Fakültesi Bilgisayar Mühendisliği Bölümü'nün 18360859049 numaralı öğrencisi Selçuk ŞAN, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "MAKİNE ÖĞRENMESİ TEMELLİ GERÇEK ZAMANLI AKILLI ODA SİSTEMİ" başlıklı bitirme çalışmasını aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

**Danışmanı :** **Dr. Öğr. Üyesi Volkan ALTUNTAŞ** .....  
Bursa Teknik Üniversitesi

**Jüri Üyeleri :** **Dr. Öğr. Üyesi Adı SOYADI** .....  
Bursa Teknik Üniversitesi

**Öğr. Gör. Dr. Adı SOYADI** .....  
Bursa Teknik Üniversitesi


**Savunma Tarihi :** 15 HAZİRAN 2023

**BM Bölüm Başkanı : Prof. Dr. Tugay Tugay Bilgin** .....  
Bursa Teknik Üniversitesi ...../...../.....

## İNTİHAL BEYANI

Bu bitirme alışmasında görsel, işitsel ve yazılı biçimde sunulan tüm bilgi ve sonuçların akademik ve etik kurallara uyularak tarafımdan elde edildiğini, bitirme alışması içinde yer alan ancak bu alışmaya özgü olmayan tüm sonuç ve bilgileri bitirme alışmasında kaynak göstererek belgelediğimi, aksinin ortaya ıkması durumunda her türlü yasal sonucu kabul ettiğimi beyan ederim.

Öğrencinin Adı Soyadı: Seluk ŞAN

İmzası : 

## **ÖNSÖZ**

Bitirme projesi geliştirme aşamasında çalışmamı takip eden değerli hocam Dr. Öğr. Üyesi Volkan ALTUNTAŞ'a, lisans hayatım boyunca desteklerini esirgemeyen Bölüm Başkanımız Prof. Dr. Turgay Tugay Bilgin'e ve diğer tüm saygı değer hocalarıma değerli katkıları için teşekkürlerimi sunuyorum.

Haziran 2023

Selçuk Şan

## İÇİNDEKİLER

### Sayfa

ÖNSÖZ .....	v
İÇİNDEKİLER .....	vi
KISALTMALAR .....	viii
ŞEKİL LİSTESİ.....	ix
ÖZET .....	x
SUMMARY .....	xi
1. GİRİŞ .....	12
1.1 Tezin Amacı .....	12
1.2 Literatür Taraması .....	13
1.3 Hipotez .....	14
2. KULLANILAN TEKNOLOJİLER.....	16
2.1 Docker .....	16
2.1.1 Sanallaştırma .....	16
2.1.2 Konteyner teknolojisi .....	17
2.1.3 Docker imajları .....	17
2.1.4 Docker konteynerleri.....	18
2.1.5 Docker compose.....	19
2.2 Python.....	19
2.3 Spark.....	20
2.3.1 Spark streaming.....	20
2.3.2 Spark sql.....	21
2.3.3 Mllib.....	21
2.3.4 Pyspark.....	22
2.4 Kafka .....	22
2.4.1 Zookeeper.....	23
2.5 Elasticsearch.....	23
2.6 Kibana .....	23
2.7 Slack.....	24
3. METODOLOJİ .....	25
3.1 Proje Geliştirme Süreci .....	25
3.2 Proje Taslağının Oluşturulması .....	26
3.3 Araçların Kurulumu ve Yönetimi .....	27
3.3.1 Docker-compose.yml dosyası .....	27
3.4 Veri Kaynağı .....	32
3.5 Veri Ön İşleme .....	33
3.5.1 Aykırı değişken analizi .....	33
3.5.2 Encoding işlemleri .....	35
3.6 Veri Görselleştirme .....	36
3.7 Veri Doğrulama (Validation) .....	37
3.8 Makine Öğrenmesi .....	38

3.8.1 Modelin eğitilmesi .....	39
3.8.1.1 Logistic Regression .....	39
3.8.2 Model performansı değerlendirme .....	40
3.9 Gerçek Zamanlı Analiz .....	41
3.9.1 Kafka-python ile iot sensör simülasyonu .....	41
3.9.2 Akan verinin elasticsearch'te depolanması .....	42
3.9.3 Spark streaming ile gerçek zamanlı işleme .....	43
3.9.4 Spark streaming ile gerçek zamanlı tahminleme .....	45
3.9.4.1 Tahmin sonucuna göre alarm gönderilmesi .....	45
<b>4. UYGULAMA ÇIKTILARI .....</b>	<b>48</b>
4.1 Iot Simülasyonu .....	48
4.2 Gerçek Zamanlı Görselleştirme .....	49
4.3 Alarm Gönderimi .....	51
<b>5. SONUÇ .....</b>	<b>52</b>
5.1 Çalışmanın Uygulama Alanı .....	52
<b>6. KAYNAKLAR .....</b>	<b>54</b>
<b>ÖZGEÇMİŞ .....</b>	<b>55</b>

## **KISALTMALAR**

<b>API</b>	: Application Programming Interface
<b>AUC</b>	: Area under the ROC Curve
<b>CSV</b>	: Comma-separated values
<b>DF</b>	: Dataframe
<b>IOT</b>	: Internet of things
<b>JDBC</b>	: Java™ EE Database Connectivity
<b>JSON</b>	: JavaScript Object Notation
<b>LR</b>	: Linear Regression
<b>ML</b>	: Machine Learning
<b>NB</b>	: Naive Baies
<b>ORC</b>	: Optimized Row Columnar
<b>PY</b>	: Python
<b>RF</b>	: Random Forrest
<b>ROC</b>	: Receiver operating characteristic
<b>SQL</b>	: Structured Query Language
<b>VM</b>	: Virtual Machine
<b>YAML</b>	: Yet another markup language



## ŞEKİL LİSTESİ

### Sayfa

Şekil 3.1 : İş-Zaman Planı .....	25
Şekil 3.2 : Proje Taslağı .....	26
Şekil 3.3 : Elasticsearch servisi .....	28
Şekil 3.4 : Kibana servisi .....	29
Şekil 3.5 : Spark servisi.....	30
Şekil 3.6 : Kafka ve Zookeeper servisleri .....	31
Şekil 3.7 : Docker bitirme_network .....	32
Şekil 3.8 : Docker Desktop'daki servislerin ekran görüntüsü .....	32
Şekil 3.9 : outlier_thresholds fonksiyonu .....	34
Şekil 3.10 : remove_outlier fonksiyonu .....	34
Şekil 3.11 : String Indexer işlemi.....	35
Şekil 3.12 : One Hot Encoding işlemi.....	35
Şekil 3.13 : Vector Assembler .....	36
Şekil 3.14 : Humidity sütunu için histogram grafiğı .....	36
Şekil 3.15 : Humidity değışkeni için aykırı değer grafiğı .....	37
Şekil 3.16 : Sayısal değışkenlerin birbiri ile korelasyonunu gösteren ısı haritası.....	37
Şekil 3.17 : Cross Validation işlemi.....	38
Şekil 3.18 : Model eğitim işlemi .....	39
Şekil 3.19 : Logistic Regression Modeli .....	40
Şekil 3.20 : Model Performans değeriendirme .....	40
Şekil 3.21 : Sensör simülasyon fonksiyonu .....	42
Şekil 3.22 : Elasticsearch'e veri yazan ToElastic sınıfı.....	43
Şekil 3.23 : Spark Session oluşturma fonksiyonu.....	43
Şekil 3.24 : Gerçek zamanlı veri işleme fonksiyonu.....	44
Şekil 3.25 : Gerçek zamanlı tahminleme fonksiyonu .....	45
Şekil 3.26 : Tahmin sonuçlarını ayrı kafka topiclerine gönderen fonksiyon .....	46
Şekil 3.27 : Gerçek zamanlı alarm oluşturan ToAlert sınıfı .....	47
Şekil 4.1 : producer.sh örnek çıktısı.....	48
Şekil 4.2 : Kümülatif toplam kayıt sayısını gösteren çizgi grafiğı.....	49
Şekil 4.3 : Odalara göre veri dağılımını gösteren pasta grafiğı.....	49
Şekil 4.4 : Anlık ortalama co2 değeriini gösteren çizgi grafiğı .....	50
Şekil 4.5 : Anlık ortalama humidity değeriini gösteren çizgi grafiğı.....	50
Şekil 4.6 : Anlık ortalama ışık değeriini gösteren çizgi grafiğı .....	50
Şekil 4.7 : Anlık ortalama sıcaklık değeriini gösteren çizgi grafiğı.....	51
Şekil 4.8 : Örnek Slack bildirimi.....	51

# MAKİNE ÖĞRENMESİ TEMELLİ GERÇEK ZAMANLI AKILLI ODA SİSTEMİ

## ÖZET

Bu tez, gerçek zamanlı makine öğrenmesi tabanlı akıllı oda sistemi projesini ele almaktadır. Projede, odaların ortam şartlarının gerçek zamanlı tahminleme ile kontrol edilmesi, o esnada odada birinin veya birilerinin bulunup bulunmadığının tahmininin yapılması ve sonucun mail gibi yollarla kullanıcıya iletilmesi hedeflenmektedir.

Projenin ilk adımında, internet üzerinden elde edilen; co2, nem, sıcaklık, hareket ve ışık sensörlerinden toplanmış verileri barındıran veri seti analiz edilmiş, ön işleme, keşifçi veri analizi ve temizleme işlemleri gerçekleştirilmiştir. Ardından, Spark'ın makine öğrenmesi kütüphanesi olan MLlib hakkında literatür araştırması yapılmış ve devamında veri seti için uygun bir makine öğrenmesi modeli seçilmiştir. Son olarak da model eğitimi gerçekleştirilerek optimum parametreler belirlenmiştir.

Projenin bir diğer önemli bileşeni, gerçek zamanlı bir şekilde verilerin işlenmesi, analizinin yapılması ve görselleştirilmesidir. Bu amaçla, Kafka, Elasticsearch ve Kibana gibi açık kaynak kodlu teknolojiler kullanılmıştır. Python ile simüle edilen gerçek zamanlı veriler Spark Streaming ile yakalanarak, makine öğrenmesi modeli uygulanmıştır. Model çıktıları ayrı ayrı Kafka Topic'lerine gönderilerek bir alarm mekanizması oluşturulmuştur. Sonuç pozitif olduğunda, yani odada bir hareket gözlemlendiğinde bir Slack kanalına bildirim yollanmaktadır.

Projenin sonucunda, gerçek zamanlı verilerin kullanılarak, oda koşullarının optimize edilmesi ve enerji tasarrufu sağlanması başarıyla gösterilmiştir. Ayrıca, çalışma süresince paralel programlama teknikleri de kullanılarak uygulama performansının artırılması hedeflenmiştir.

Bu çalışma, gerçek zamanlı makine öğrenmesi uygulamaları yapacak araştırmacılar için uygun bir yol haritası sunmaktadır. Projenin başarılı sonuçları, benzer uygulamaların geliştirilmesi ve yaygınlaştırılması için de bir örnek teşkil etmektedir.

**Anahtar kelimeler:** Gerçek Zamanlı, Makine Öğrenmesi, PySpark, Kafka, Kibana, Elasticsearch

## **MACHINE LEARNING BASED REAL-TIME SMART ROOM SYSTEM**

### **SUMMARY**

This thesis deals with machine learning based real-time smart room system project. In the project, it is aimed to control the ambient conditions of the rooms with real-time estimation, to estimate whether there is someone in the room at that time, and to send the result to the user via e-mail.

In the first step of the project, the data set containing data collected from co2, humidity, temperature, motion and light sensors obtained over the internet was analyzed, preprocessing, exploratory data analysis and cleaning processes were carried out. Then, a literature search was made about MLib, Spark's machine learning library, and then a suitable machine learning model was selected for the data set. Finally, the optimum parameters were determined by performing model training.

Another important component of the project is the processing, analysis and visualization of data in real time. For this purpose, open-source technologies such as Kafka, Elasticsearch and Kibana were used. Real-time data simulated with Python was captured with Spark Streaming and machine learning model was applied. An alert mechanism was created by sending the model outputs to Kafka Topics separately. A notification is sent to a Slack channel when the result is positive, that is, when a movement is observed in the room.

As a result of the project, optimizing room conditions and saving energy have been successfully demonstrated by using real-time data. In addition, it is aimed to increase the application performance by using parallel programming techniques during the study.

This study provides a suitable roadmap for researchers who will make real-time machine learning applications. The successful results of the project also set an example for the development and dissemination of similar practices.

**Keywords:** Realtime, Machine Learning, PySpark, Kafka, Kibana, Elasticsearch

## **1. GİRİŞ**

Gerçek Zamanlı Makine öğrenmesi, verilerin gerçek zamanlı olarak akış halinde olduğu ve bu verilerin analiz edilerek anlık kararların alındığı çalışmalardır. Bu yöntem, özellikle büyük veri setleriyle uğraşan ve hızlı, doğru kararlar almak isteyen işletmeler için oldukça önemlidir.

Gerçek Zamanlı Makine öğrenmesi, sürekli olarak yeni verilerin geldiği ve bu verilerin hızlı bir şekilde işlenmesi gereken durumlarda kullanılır. Bu veriler, genellikle sensörler, cihazlar ve ağlardan gelen akış verileri, sosyal medya paylaşımları, web tıklamaları gibi kaynaklardan oluşur. Bu verilerin işlenmesi, hızlı ve anlık kararlar almak için oldukça önemlidir. Bu nedenle, birçok endüstri için kritik öneme sahiptir.

Gerçek Zamanlı Makine öğrenmesi uygulamaları arasında müşteri davranışı analizi, trafiğin yönetimi, finansal piyasa analizi, endüstriyel kontrol, tıbbi teşhis, güvenlik tehditlerinin tespiti, araç takibi gibi alanlar bulunmaktadır. Bu uygulamaların çoğu, verilerin anlık olarak işlenmesi ve kararların hızlı bir şekilde alınması gerektiğinden, streaming data ile makine öğrenmesi yöntemleri kullanmaktadır.

Sonuç olarak, bu yöntem, hızlı ve doğru kararlar almak isteyen işletmeler için oldukça önemlidir ve birçok endüstri için kritik öneme sahiptir.

### **1.1 Tezin Amacı**

Bu tezin amacı, gerçek zamanlı makine öğrenmesi tabanlı akıllı oda sistemi projesini ele alarak, odaların ortam şartlarının kontrol edilmesi, o esnada odada birinin veya birilerinin bulunup bulunmadığının tahmin edilmesi ve sonucun mail gibi yollarla kullanıcıya iletilmesini hedeflemektedir. Bunun yanı sıra, gerçek zamanlı verilerin kullanılarak, oda koşullarının optimize edilmesi ve enerji tasarrufu sağlanması amaçlanmaktadır.

## 1.2 Literatür Taraması

M. E. Bilgin, H. H. Kilinc and A. H. Zaim, "An Anomaly Detection Study for the Smart Home Environment," (2022), evdeki 7 farklı sensör verisindeki anormallikleri ve olağandışı durumları tespit etmeye odaklanılmıştır. Bunun için, denetimsiz ve denetimli makine öğrenimi algoritmalarının bir kombinasyonu ile oluşturulmuş bir model kullanılmıştır. Sensör verileri, denetimsiz algoritmalarından biri olan Isolation Forest kullanılarak etiketlenmiş ve ardından veriler, Karar Ağacı, Random Forest ve XGBoost sınıflandırma algoritmaları gibi denetimli algoritmalarla eğitilmiştir. Anormallik kararları %99'dan fazla doğrulukla yapılmıştır.

Abdul Rehman Javed, Mohammad S. Khan, "Automated cognitive health assessment in smart homes using machine learning, (2021), çalışmada, akıllı evlerde yapılan faaliyetlerin tanınması ve analiz edilmesi amaçlanmıştır. Basit Günlük Yaşam Faaliyetleri ile Karmaşık İç İç Faaliyetlerin sınıflandırma performansının geliştirilmesi üzerine odaklanılmıştır. Mevcut yaklaşımlarla karşılaştırıldığında en iyi doğruluk oranını %96,02 ve %99,6 olarak elde edilmiştir. Bunun yanı sıra, bunama sorunu olan kişilerin hangi faaliyetlerde zorlandığını analiz edilmiştir. Ek olarak zaman bazlı özellik analizleri sağlayarak, özellikle engelli kişilerin etkin bir şekilde tespit edilmesine yardımcı olup olmadığı tahmin edilmiştir. Bu çalışma, bireylerin ev ortamlarında fonksiyonel sağlık analizini yapmak için önemli bir adımdır. Gelecekte, gerçek zamanlı olarak insan faaliyetlerini değerlendirecek otomatik bir çerçeve oluşturulması planlanmaktadır ve bu çerçeve, gerçek zamanlı olarak akıllı evde gerçekleştirilen faaliyetlere otomatik olarak üretilen puanlar atayacaktır.

John Jaihar, Neehal Lingayat (2020), çalışma, ev otomasyon sistemlerinin yapısını ve kullanımını açıklamaktadır. Bu sistemler, ev içindeki farklı cihazların merkezi bir sunucu üzerinden kontrol edilmesine ve yönetilmesine imkan vermektedir. Bu sistemlere makine öğrenimi algoritmaları ve nesne algılama teknolojileri de entegre edilerek, kullanıcının duygu durumuna göre ortam koşullarını otomatik olarak ayarlamak mümkündür. Bu şekilde, ev otomasyonu daha akıllı bir karar verme sistemine dönüştürülebilir ve kullanıcılara daha fazla konfor, sağlık, güvenlik ve enerji tasarrufu sağlanabilir. Önerilen sistem, Raspberry Pi gibi tek kartlı bir bilgisayar üzerinden farklı cihazlar ve sensörlerle bağlantı kurarak çalışmaktadır. Ayrıca, kullanıcının yüz ifadesini algılamak için kameradan yararlanmaktadır.

M. K. I. Shafi, M. R. Sultan, S. M. M. Rahman and M. M. Hoque, "IoT Based Smart Home: A Machine Learning Approach," (2021), çalışmada, IoT tabanlı bir akıllı ev sistemi incelenmiş ve makine öğrenimi teknikleri kullanılarak otomatik ve etkili bir şekilde IoT cihazlarının kontrol edilebilmesi için yeni bir yaklaşım önerilmiştir. Sistem, insan varlığı sayısı, sıcaklık, nem ve aydınlık gibi çevresel değişkenleri kullanarak yapay verilerin yanı sıra gerçek zamanlı sensör verilerinin bir bölümünü toplayarak eğitilir. Karar Ağacı algoritması, önerilen kontrol modellerinin verilerini sınıflandırmak için uygulanır ve modellerin performans değerlendirmesi çapraz doğrulama teknikleri kullanılarak ölçülmüştür.

N. Morresi, S. Casaccia, L. Scalise and G. M. Revel, "Machine learning algorithms for the activity monitoring of elders by home sensor network," (2022), çalışmada, yaşlı insanların aktivitelerini takip etmek için hareket ve ışık sensörleri ile oluşturulan bir ev sensör ağı kullanılmıştır. Toplanan veriler, 4 saatlik 6 zaman dilimine bölünmüş ve unsupervised machine learning algoritmaları kullanılarak farklı aktivasyon kalıplarına göre kümeleme yapılmıştır. Kümeleme sonuçları, House 4 ve House 5'in ilk zaman diliminde en iyi sonuçları vermiştir. Daha sonra, DT algoritması kullanılarak, her evin farklı zaman dilimlerindeki kümeleme metodolojisi anlaşılmasına çalışılmıştır. Sonuç olarak, bu çalışma yaşlı insanların aktivitelerini takip etmek için kullanılan sensör ağlarının ve makine öğrenmesi algoritmalarının potansiyelini göstermektedir.

### **1.3 Hipotez**

IoT sensörlerinin kullanımı, birçok endüstride otomasyon sistemleri üzerinde çalışırken daha doğru sonuçlar elde etmek için potansiyel faydalar sağlayabilir. Bu potansiyel faydalar, verilerin gerçek zamanlı olarak toplanması ve işlenmesi sayesinde elde edilir. Bu süreç, endüstriyel üretimde kullanılan makinelerin performansını ölçmek, hava kalitesini izlemek, enerji tasarrufu yapmak ve daha birçok amaç için kullanılabilir.

Bu proje, Elasticsearch, Kibana, Kafka ve Spark gibi açık kaynaklı araçları birleştirerek binlerce satır verinin analiz edilmesini ve hızlı bir şekilde işlenmesini mümkün kılarak daha iyi sonuçlar elde edilmesine yardımcı olur.

Bununla birlikte gerek zamanlı olarak odadaki insan varlıđının bir makine ğrenimi modeli kullanılarak tespit edilmesi, otomasyon sistemlerinde daha dođru sonuçlar retmeye yardımcı olacak ve verimli bir alıřma sađlayacaktır.

## **2. KULLANILAN TEKNOLOJİLER**

Bu proje, gerçek zamanlı makine öğrenmesi dayalı bir akıllı oda sistemi oluşturma üzerine odaklanmaktadır. Projede, odaların ortam şartları gerçek zamanlı olarak tahmin edilerek kontrol edilmekte ve odada biri olup olmadığı tahmin edilmektedir. Son olarak kullanıcılara sonuçlar çeşitli yöntemlerle iletilebilmektedir.

Projenin ilk aşamasında, internet üzerinden elde edilen verilerin analizi yapılmıştır. Daha sonra, PySpark MLib kütüphanesi kullanılarak, uygun makine öğrenmesi modeli seçilmiştir. Model, optimum parametreler belirlenerek eğitilmiştir.

Projenin bir diğer önemli bileşeni, gerçek zamanlı verilerin işlenmesi, analizi ve görselleştirilmesidir. Burada, Kafka, Elasticsearch ve Kibana araçları tercih edilmiştir. Gerçek zamanlı verilerin yakalanmasında Spark Streaming kullanılmıştır. Model çıktılarının bildirim olarak gönderilmesinde de yine Kafka tercih edilmiştir. Projenin geliştirilmesi tamamen python dilinde gerçekleştirilmiştir.

Projenin uygulanmasında, araçların kurulumu ve yönetimi için Docker kullanılmıştır. Docker, araçların kolayca kurulumu ve konfigürasyonu sağlayarak, proje geliştirme sürecini hızlandırmıştır. Ayrıca Docker sayesinde, proje ortamının farklı bilgisayarlarda tutarlı bir şekilde çalışması sağlanmıştır.

### **2.1 Docker**

Docker, uygulama ve servislerin farklı ortamlarda ve cihazlarda sorunsuz bir şekilde çalıştırılmasını sağlayan bir sanallaştırma platformudur. Docker, uygulamaların tüm bağımlılıklarını kapsayan hafif, taşınabilir ve ölçeklenebilir sanal konteynerler oluşturur. Bu konteynerler, uygulamaların farklı işletim sistemleri veya donanım yapılandırmaları üzerinde sorunsuz bir şekilde çalıştırılmasını sağlar ve yazılım dağıtım sürecini kolaylaştırır. Docker, açık kaynaklı bir projedir ve Linux tabanlı işletim sistemleri, Windows ve MacOS gibi diğer platformlarla da uyumludur.

#### **2.1.1 Sanallaştırma**

Sanallaştırma, bir bilgisayarın kaynaklarını (donanım, yazılım, ağ kaynakları vb.) bir sanal ortamda bölüştürerek birden fazla işletim sistemi veya uygulama çalıştırmayı mümkün kılan bir teknolojidir. Sanallaştırma, farklı işletim sistemleri, uygulama versiyonları veya farklı ağ yapıları gibi farklı ortamları tek bir fiziksel sunucuda



çalıştırmak için kullanılabilir. Bu sayede donanım kaynakları daha verimli kullanılır, yönetim kolaylaşır ve sistemlerin birbirleriyle etkileşimi azaltılır. Sanallaştırma teknolojileri arasında öne çıkanlar arasında Docker, VMWare, Hyper-V ve VirtualBox yer almaktadır.

### **2.1.2 Konteyner teknolojisi**

Konteyner teknolojisi, uygulamaları sanallaştırmak ve her uygulamanın kendi ortamını izole etmek için kullanılan bir teknolojidir. Konteynerlar, uygulamaları çalıştırmak için gereken tüm kaynakları içinde barındırır ve farklı konteynerlar arasında kaynak paylaşımı yapabilirler. Bu sayede uygulama performansı artar ve farklı uygulamalar arasında kaynak tüketimi daha verimli hale gelir.

Docker, konteyner teknolojisi için bir platformdur. Docker, uygulamaları ve tüm bağımlılıklarını tek bir konteyner içinde paketleyerek taşınabilir ve bağımsız hale getirir. Böylece uygulamalar, herhangi bir ortamda çalıştırılabilir hale gelir. Docker, uygulamaların hızlı ve güvenli bir şekilde dağıtılmasını, çalıştırılmasını ve yönetilmesini sağlar.

Sanallaştırma ve konteyner teknolojileri, yazılım uygulamalarını çalıştırmak için kullanılan iki farklı yaklaşımdır. Sanallaştırma teknolojileri, bir ana bilgisayar üzerinde birden fazla sanal makine oluşturarak her bir sanal makineye ayrı bir işletim sistemi yükler. Bu sayede her uygulama için ayrı bir sanal makine oluşturulur ve uygulama, kendi sanal makinesinde çalıştırılır. Konteyner teknolojisi ise, uygulamaların her birinin kendi ortamında çalışmasını sağlayan daha hafif bir sanallaştırma tekniğidir.

Docker, bir konteyner yönetim sistemidir ve uygulamaların farklı ortamlarda çalıştırılmasını kolaylaştıran bir araçtır. Docker, bir imaj olarak adlandırılan önceden yapılandırılmış bir ortamı kullanarak, uygulamanın gereksinim duyduğu tüm bileşenleri içeren bir konteyner oluşturur. Bu sayede uygulama, gerekli tüm ortam bileşenleriyle birlikte, farklı sistemlerde de çalıştırılabilir hale gelir.

### **2.1.3 Docker imajları**

Docker imajları, Docker konteynerlerinin çalıştırılması için gerekli olan dosyalar, bağımlılıklar ve konfigürasyonları içeren önceden hazırlanmış paketlerdir. Bir Docker imajı, bir uygulamanın tüm bileşenlerini, çalışma zamanı, kütüphaneleri ve diğer

bağımlılıklarını içerebilir. Bu sayede, bir Docker imajı oluşturarak, uygulama farklı ortamlarda sorunsuz bir şekilde çalıştırılabilir.

Docker imajları, Dockerfile adı verilen özel bir dosya kullanılarak oluşturulurlar. Dockerfile, imajın içerisindeki paketleri, bağımlılıkları, çalıştırılacak komutları ve diğer konfigürasyonları tanımlayan bir betik dosyasıdır. Dockerfile'da tanımlanan tüm işlemler, imaj oluşturulduğunda adım adım gerçekleştirilir ve sonuçta bir imaj oluşur. Bu imaj daha sonra Docker Hub gibi bir yerel ya da uzak imaj deposuna kaydedilebilir ve paylaşılabilir.

#### **2.1.4 Docker konteynerleri**

Docker konteynerleri, uygulamaları ve hizmetleri izole edilmiş ortamlarda çalıştırmak için kullanılan bir sanallaştırma teknolojisidir. Docker, uygulamaları bir konteyner içinde paketlemek ve bu konteynerleri farklı işletim sistemleri ve platformlar üzerinde tutarlı bir şekilde çalıştırmak için bir dizi araç ve hizmet sağlar.

Docker konteynerleri, bir uygulamanın tüm gereksinimlerini içeren bir yazılım birimi olarak düşünülebilir. Bu gereksinimler, uygulama kodunu, çalışma zamanı, sistem araçlarını, kütüphaneleri ve diğer bağımlılıkları içerebilir. Konteynerler, bu gereksinimleri kendilerine özgü bir dosya sistemine ve kaynaklara sahip bir ortamda paketler.

Docker, konteynerlerin dağıtımını ve yönetimini kolaylaştıran bir dizi araç sunar. Bir Docker konteyneri oluşturmak için, bir Docker imajı kullanılır. İmajlar, konteynerin başlangıç noktasıdır ve konteynerin çalışma zamanında kullanılacak tüm bileşenleri içerir. Bir imaj oluşturulduktan sonra, bu imajdan birden fazla konteyner oluşturulabilir.

Docker konteynerleri, yüksek düzeyde taşınabilirlik sunar. Bir konteyner, kendi içinde çalışması gereken tüm gereksinimleri ile birlikte taşınabilir bir birimdir. Bu, bir konteynerin farklı işletim sistemleri, bulut platformları veya geliştirme ortamları arasında kolayca taşınabilmesini sağlar.

Docker konteynerleri, uygulama dağıtımını ve ölçeklendirmesi için popüler bir seçenek haline gelmiştir. Konteynerlerin hafif olması ve hızlı başlatılabilmesi, uygulamaların hızlı bir şekilde dağıtılmasını ve ölçeklendirilmesini sağlar. Ayrıca, birden çok

konteynerin birlikte çalışmasını ve kaynakları etkin bir şekilde paylaşmasını sağlayan orkestrasyon araçlarıyla (örneğin Kubernetes) birlikte kullanılabilirler.

Docker konteynerleri, yazılım geliştirme sürecinde de faydalar sağlar. Geliştiriciler, uygulamaları yerel ortamlarında çalıştırırken bile izole edilmiş bir ortamda çalışmalarını sağlar. Ayrıca, farklı geliştirme ortamlarında çalışan ekip üyeleri arasında tutarlılık sağlamak için kullanılabilir.

Sonuç olarak, Docker konteynerleri, uygulamaları taşınabilir, hızlı ve izole edilmiş bir şekilde çalıştırmak için kullanılan bir sanallaştırma teknolojisidir.

### **2.1.5 Docker compose**

Docker Compose, birden fazla Docker konteynerini bir arada çalıştırmak için kullanılan bir araçtır. Compose, YAML dosyaları aracılığıyla Docker konteynerlerinin yapılandırmasını tanımlamanıza olanak tanır. Bu dosyalarda, konteynerlerin nasıl oluşturulacağı, çalıştırılacağı, birbirleriyle nasıl etkileşime girecekleri, hangi ağlar ve depolama aygıtları kullanılacağı gibi konular belirtilir. Docker Compose kullanılarak birden fazla konteyner aynı ortamda çalıştırılabilir.

Docker Compose, karmaşık mikro servis uygulamaların kolayca yönetilmesini sağlar. Her bir mikro servis, ayrı bir Docker konteynerinde çalıştırılabilir ve bu konteynerler Docker Compose ile kolayca bir araya getirilebilir. Bu sayede, mikro servis mimarisi kullanılarak geliştirilen uygulamaların geliştirilmesi, test edilmesi ve dağıtımı kolaylaşır.

## **2.2 Python**

Python, Guido van Rossum tarafından geliştirilen, açık kaynak kodlu, nesne yönelimli bir programlama dilidir. Basit ve anlaşılır sözdizimi, güçlü bir standart kütüphanesi ve çeşitli platformlar arasında taşınabilirliği sayesinde popüler bir dildir. Python, birçok farklı alanda kullanılabilir; web geliştirme, veri bilimi, yapay zeka, otomasyon, oyun geliştirme ve daha pek çok alanda kullanılabilir.

Python, yüksek seviyeli bir programlama dilidir, yani kodlama sürecinde programcıların çok az sayıda ayrıntıya dikkat etmesine izin verir. Örneğin, hafıza yönetimi veya işletim sistemi kaynaklarına erişim konularında endişelenmek gerekmez.

Python, açık kaynak kodlu bir dildir, yani kullanıcılar tarafından ücretsiz olarak indirilebilir, kullanılabilir ve değiştirilebilir. Ayrıca, Python topluluğu oldukça aktif bir topluluktur ve yeni özelliklerin ve geliştirmelerin düzenli olarak yayınlandığı bir dil olmasıyla bilinir.

Pandas ve NumPy gibi veri bilimi için kullanılan kütüphaneler mevcuttur. Pandas, Python için açık kaynaklı bir veri analizi kütüphanesidir. Pandas, özellikle büyük ve karmaşık veri setlerinin işlenmesinde kolaylık sağlar. Veri işleme, temizleme, analiz ve modelleme için birçok fonksiyon içermektedir. Bu kütüphane projenin geliştirme sürecinde oldukça fazla kullanılmaktadır.

Sonuç olarak, Python, güçlü bir programlama dilidir ve birçok alanda kullanılabilir. Basit ve anlaşılır sözdizimi, açık kaynak kodlu yapısı ve çeşitli kütüphaneleri sayesinde popüler bir dil olarak öne çıkmaktadır.

## **2.3 Spark**

Apache Spark, büyük veri işleme ve analiz için kullanılan bir açık kaynaklı veri işleme motorudur. Spark, hızlı ve ölçeklenebilir bir yapıya sahip olması nedeniyle, veri bilimi ve büyük veri uygulamalarında popüler bir araçtır. Spark, dağıtık veri işleme işlemlerini optimize etmek için in-memory veri işleme teknolojisini kullanır. Spark, veri işleme için birden fazla API sağlar, bunlar arasında Spark SQL, Spark Streaming, Spark MLlib (makine öğrenmesi) ve GraphX (grafik işleme) bulunur. Ayrıca, Spark'ın Java, Scala, Python ve R gibi farklı programlama dilleriyle kullanılabilmesi de avantajıdır. Spark'ın yüksek hız ve performansı, özellikle büyük veri işleme ve analiz işlemlerinde kullanılması için tercih edilmesinin nedenlerindendir.

### **2.3.1 Spark streaming**

Spark Streaming, Apache Spark'ın bir parçasıdır ve gerçek zamanlı veri işleme için tasarlanmıştır. Geleneksel olarak, veri işleme işlemleri sadece birkaç saatte bir yapılırken, Spark Streaming gerçek zamanlı verileri sürekli olarak işleyebilir.

Spark Streaming, küçük toplu işlemler (batch) üzerinde çalışır. Bu küçük toplu işlemler, akış halindeki verileri bir araya getirir ve bunları birlikte işler. Spark Streaming, yüksek bir işlem oranı sağlar ve hızlı veri işleme işlemleri yapabilir.

Spark Streaming, birçok farklı kaynaktan veri girdisi alabilir. Örneğin, Kafka, Flume ve Amazon S3 gibi çeşitli veri kaynaklarından veri akışı sağlayabilir. Bu veriler daha sonra Spark Streaming tarafından işlenir ve sonuçlar gerçek zamanlı olarak üretilir.

Spark Streaming, çevik ve ölçeklenebilir bir veri işleme çözümüdür ve büyük veri işleme işlemleri için popüler bir seçimdir.

### **2.3.2 Spark sql**

Spark SQL, Apache Spark'ın bir bileşenidir ve verilerin yapılandırılmış ve yarı yapılandırılmış veri kaynaklarından işlenmesine ve sorgulanmasına izin verir.

Spark SQL, SQL sorgularını çalıştırmak, DataFrame ve Dataset API'lerini kullanmak ve veri kaynakları arasında SQL benzeri bir dil olan DataFrame API'sini kullanarak veri transferi yapmak gibi birçok özellik sunar. Spark SQL, birden fazla veri kaynağına erişim sağlar, özellikle de yapılandırılmış verileri işlemek için kullanılan Parquet, ORC, JSON, CSV ve JDBC veri kaynaklarını destekler.

Spark SQL, Spark'ın en popüler bileşenlerinden biridir. Spark'ın genişletilebilirliği, hızı ve performansından faydalanarak büyük ölçekli veri işleme problemlerini çözmek için kullanılan birçok endüstriyel ve akademik uygulamalarda kullanılmaktadır.

### **2.3.3 Mllib**

Spark MLlib (Machine Learning library) açık kaynaklı bir makine öğrenmesi kütüphanesidir ve Apache Spark tarafından desteklenmektedir. Bu kütüphane, büyük veri setleri üzerindeki karmaşık veri işleme ve analiz işlemlerini gerçekleştirmek için kullanılabilir.

Spark MLlib, çeşitli makine öğrenmesi algoritmalarını içermektedir. Bunlar arasında sınıflandırma, regresyon, kümeleme, boyut indirgeme, özellik çıkarma, işlem gücü optimizasyonu ve veri ön işleme algoritmaları bulunmaktadır. Bu algoritmaların çoğu, Spark'ın özellikle paralel hesaplama ve dağıtık veri işleme yetenekleri sayesinde büyük ölçekli veri setlerinde uygulanabilir. Ayrıca, Spark SQL, Spark Streaming ve GraphX gibi diğer Spark bileşenleriyle entegre çalışabilir ve büyük ölçekli veri işleme ve analiz projelerinde kullanılabilecek birçok araç sağlar.

### **2.3.4 Pyspark**

PySpark, Apache Spark'ın Python programlama dili için bir arayüzüdür. Apache Spark, büyük veri işleme uygulamaları için açık kaynaklı bir veri işleme çerçevesidir ve veri işleme işlemlerini büyük ölçüde hızlandırır.

PySpark, Python geliştiricilerinin Apache Spark'ın özelliklerini kullanmalarına olanak tanır. Python'da yazılan kodlar, PySpark aracılığıyla Apache Spark üzerinde çalıştırılabilir. PySpark, Python için Spark'ın dağıtılmış veri işleme modeli üzerinde yüksek seviyede bir API sağlar.

PySpark'ın birçok avantajı vardır. Python, Apache Spark'ın daha düşük seviyeli Java ve Scala API'larına kıyasla daha kolay okunur ve yazılır. Ayrıca, PySpark, Python topluluğunda çok yaygın olan birçok kütüphanenin kullanımına da olanak tanır. Bunlar arasında Pandas, NumPy ve SciPy gibi veri bilimi kütüphaneleri yer alır.

PySpark ayrıca, Spark SQL, Spark Streaming, Spark MLLib ve GraphX gibi Apache Spark'ın diğer bileşenleri için de Python API'leri sağlar. Bu, Python geliştiricilerinin Spark'ın farklı bileşenlerini kullanarak büyük veri işleme işlemleri gerçekleştirmelerini kolaylaştırır.

### **2.4 Kafka**

Apache Kafka, yüksek ölçekli, dağıtılmış, açık kaynak kodlu bir mesajlaşma sistemidir. Kafka, verilerin farklı uygulamalar arasında güvenilir, yüksek performanslı ve ölçeklenebilir bir şekilde akışını sağlar.

Kafka, birçok farklı uygulama senaryosuna uygun şekilde kullanılabilir. Örneğin, uygulamalar arasında mesajlaşma, loglama, akış işleme, ölçeklenebilir veri depolama ve işleme, gerçek zamanlı veri beslemeleri gibi birçok senaryoda kullanılabilir.

Kafka, mesajları birçok farklı kaynaktan alarak, bunları birçok farklı hedefe gönderir. Bu işlem, kaynakların ve hedeflerin bağımsız olarak ölçeklendirilebildiği ve değiştirilebildiği bir şekilde gerçekleştirilir. Kafka, yüksek performans ve yüksek ölçeklenebilirlik sağlamak için çok sayıda paralel işlemi destekler ve birçok farklı protokolü kullanarak mesajların alınması ve gönderilmesini sağlar.

### **2.4.1 Zookeeper**

ZooKeeper, açık kaynaklı bir koordinasyon servisi. Büyük ölçekli dağıtık sistemlerde kullanılan bir koordinasyon aracıdır. ZooKeeper, birbirinden bağımsız birden fazla sunucudan oluşan bir kümeden oluşur. Bu sunucular arasında veri senkronizasyonu yaparlar ve bu sayede sistemdeki diğer bileşenler arasında koordinasyon sağlarlar.

ZooKeeper, genellikle Apache Kafka ve gibi büyük ölçekli dağıtık sistemler ile birlikte kullanılır. Kafka, bir mesajlaşma sistemidir ve ZooKeeper, Kafka kümelerinin konfigürasyon, izleme ve senkronizasyonunda kullanılır.

### **2.5 Elasticsearch**

Elasticsearch, açık kaynaklı bir arama ve analiz motorudur. Büyük ölçekli verileri gerçek zamanlı olarak depolayabilir, arayabilir ve analiz edebilir. Elasticsearch, veri depolama ve arama işlemlerini hızlandırmak için Apache Lucene arama kütüphanesi üzerine inşa edilmiştir. Elasticsearch, birçok farklı veri kaynağından veri alabilir ve bu verileri gerçek zamanlı olarak analiz edebilir.

Elasticsearch, çeşitli endüstrilerde kullanılan birçok farklı kullanım senaryosu sunar. Bunlar arasında log yönetimi, metin arama, güvenlik bilgilerinin analizi, iş zekası ve daha birçokları bulunmaktadır. Elasticsearch, yüksek performans, ölçeklenebilirlik, gerçek zamanlı veri analizi ve kolay kullanım gibi özellikleriyle popüler bir arama ve analiz motorudur.

Elasticsearch, JSON tabanlı bir API ile iletişim kurar ve RESTful arayüzü sayesinde birçok farklı programlama dilinde yazılmış uygulamalar tarafından kullanılabilir. Ayrıca, Kibana gibi görselleştirme araçları da Elasticsearch ile entegre edilebilir, bu sayede verilerin görselleştirilmesi ve analiz edilmesi kolaylaşır.

### **2.6 Kibana**

Kibana, Elasticsearch ile birlikte kullanılan açık kaynaklı bir veri görselleştirme aracıdır. Kullanıcıların Elasticsearch verilerini görselleştirmelerine, analiz etmelerine ve araştırmalarına olanak tanır. Kibana, Elasticsearch ile birlikte çalışarak gerçek zamanlı veri analizi yapmak için kullanılabilir. Kibana ile kullanıcılar, verileri grafikler, tablolar, haritalar ve özel görsel öğeler kullanarak görselleştirebilirler.

## **2.7 Slack**

Slack, bir takımın ya da bir şirketin çalışanlarının bir arada çalışabilmesi için tasarlanmış, mesajlaşma, dosya paylaşımı, işbirliği vb. amaçlar için kullanılmakta olan bir platformdur. Bu platform üzerinden mesajlar, dosyalar, notlar, görevler, takvim etkinlikleri ve diğer iş süreçleri paylaşılabilir ve yönetilebilir. Slack, aynı zamanda diğer iş uygulamaları ve hizmetleriyle entegrasyonlar sağlar ve kullanıcıların bu hizmetleri doğrudan Slack üzerinden kullanmalarına olanak tanır.

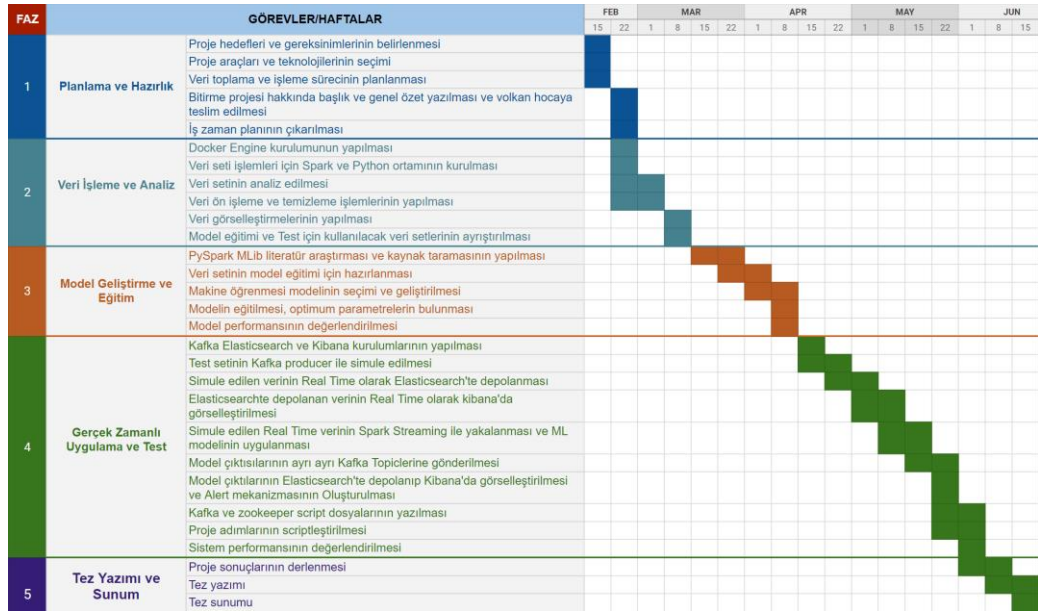


### 3. METODOLOJİ

Bu projenin metodolojisi; araçların kurulumu ve yönetimi, gerçek zamanlı verilerin analiz edilmesi, işlenmesi, makine öğrenmesi modelinin eğitilmesi gibi bir dizi adımı içerir. Bu adımların her biri, projenin başarılı bir şekilde tamamlanması için önemlidir.

#### 3.1 Proje Geliştirme Süreci

Çalışma planı hazırlamak ve bu plana sadık kalmak projenin başarılı bir şekilde ilerlemesi açısından oldukça önemlidir. Projeye başlamadan önce hazırlanan, Şekil 3.1'deki İş-Zaman planı, bu konuda zaman yönetimi ve isterlerin netleştirilmesi konusunda çok büyük faydalar sağlamıştır.



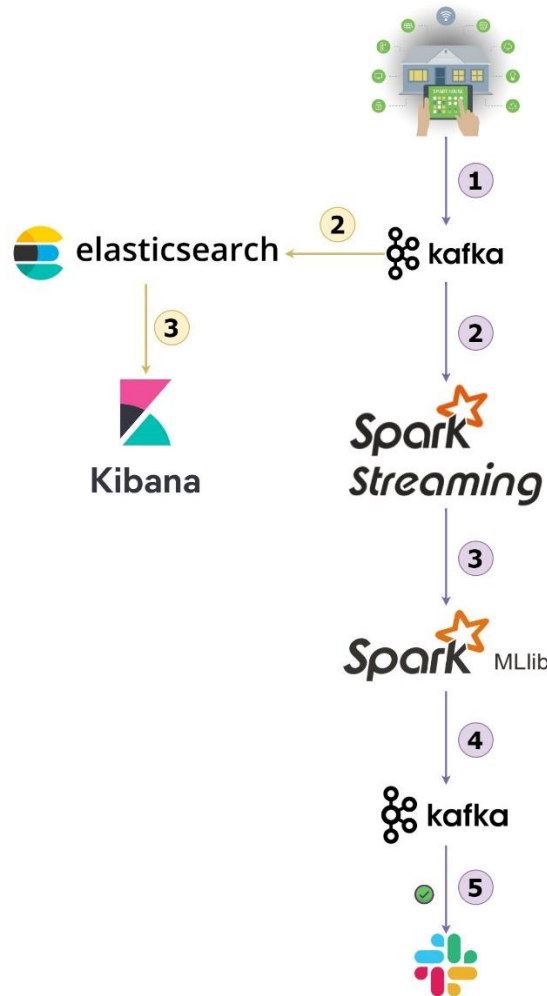
Şekil 3.1 : İş-Zaman Planı

Ayrıca, teknoloji kullanımıyla ilgili olası problemleri öngörebilmek ve buna göre önlemler almak, projenin zamanında tamamlanması ve başarılı bir şekilde sonuçlanması için önemli unsurlardandır. Bu projede yaşanan çeşitli versiyon uyumsuzluğu sorunları, önceden öngörülerek VirtualBox yerine Docker kullanılarak aşılmıştır. Bu gibi durumlarda hızlı bir şekilde çözüm üretebilmek, proje sürecini olumlu anlamada doğrudan etkilemiştir.

### 3.2 Proje Taslağının Oluşturulması

Projenin ilk aşamasında, Kafka ve Python araçlarını kullanarak IoT simülasyonu ile veriler üretilir. Kafka consumer'ı ile veriler yakalanır. Bu veriler gerçek zamanlı olarak Elasticsearch'te depolanır. Depolanan veriler, Kibana aracılığıyla gerçek zamanlı olarak görselleştirilir. Kafka consumer ile yakalanan bu veriler Spark Streaming ile ön işlemeye sokulur. Ön işlemeyen çıkan veriler, önceden eğitilmiş modele gerçek zamanlı olarak gönderilir. Modelden çıkan sonuçlar, ilgili Kafka topic'ine yönlendirilir ve eğer sonuç pozitif ise Slack kanalına bir uyarı bildirimi iletilir.

Sonuç olarak, gerçek zamanlı veri işleme, analiz ve görselleştirme işlemleri, Şekil 3.2'de de gösterilen açık kaynaklı teknolojiler kullanılarak uygulanır.



Şekil 3.2 : Proje Taslağı

### 3.3 Araçların Kurulumu ve Yönetimi

Bu projede, çeşitli teknolojilerin kullanılması gerekmektedir. Bu teknolojilerin kurulumu ve yönetimi için Docker kullanılmaktadır. Docker, uygulamaları herhangi bir ortamda hızlı ve kolay bir şekilde çalıştırmak için kullanılan bir containerization teknolojisidir. Bu, uygulamaların bilgisayar kaynaklarından izole edilmesini ve daha güvenli bir şekilde çalışmasını sağlar.

Bunun yanı sıra, proje gereksinimleri dikkate alındığında, konteynerizasyon teknolojisi sanallaştırmaya göre çok daha hızlı ve performanslı bir yöntemdir. Sanallaştırma, tamamen ayrı bir işletim sistemi kurulumu gerektirirken, Docker konteynerizasyon, çok daha az kaynak tüketimiyle çalışabilir. Bu sebeple bir sanallaştırma teknolojisi olan VirtualBox yerine Docker tercih edilmiştir. Ayrıca Docker, sanal makine kurulumu için gerekli olan uzun yüklemeleri ve yapılandırmaları ortadan kaldırır. Böylece, uygulama geliştirme sürecini hızlandırır ve uygulama ortamını farklı bilgisayarlarda daha kolay bir şekilde oluşturur.

Projenin yönetimi için ise Docker Compose kullanılmaktadır. Docker Compose, birden fazla Docker konteynerini koordine etmek için kullanılan bir araçtır. Bu, uygulamanın tüm bileşenlerinin tek bir komutla çalıştırılmasını ve yönetilmesini sağlar. Ayrıca Docker Compose, birden fazla container'ın yapılandırmasını tek bir dosyada tanımlayarak, uygulamanın daha kolay bir şekilde yönetilmesini sağlar.

Docker ve Docker Compose kullanılması ile uygulamanın kurulumu ve yönetimi oldukça kolay hale gelmektedir. Bu sayede uygulamanın geliştirme süreci daha hızlanmış ve daha güvenli hale gelmiştir.

#### 3.3.1 Docker-compose.yml dosyası

Docker-compose.yml dosyası, bir akıllı oda sistemi oluşturmak için kullanılan teknolojilerin containerization yöntemiyle kurulumunu ve yönetimini kolaylaştırmak için kullanılmıştır. Docker Compose, birden fazla container'ı tek bir ortamda çalıştırmayı sağlar ve bu dosya, ElasticSearch, Kibana, Spark, Zookeeper ve Kafka container'larını içerir.

Docker Compose dosyası YAML formatında yazılır ve yapılandırma bilgilerini içerir. Bu dosya, projenin çalıştırılması için gereken tüm konteynerleri oluşturmak ve

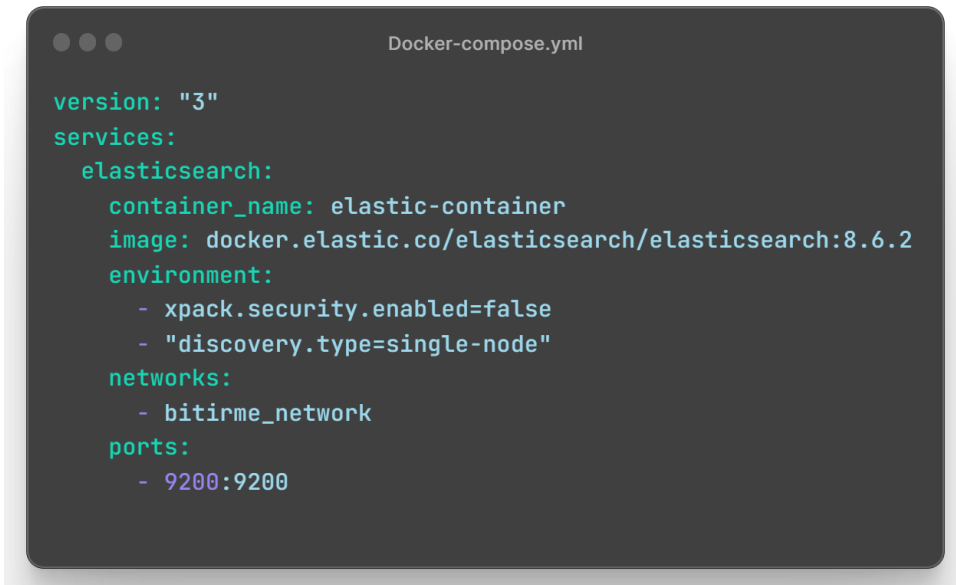
konfigüre etmek için kullanılır. Dosyanın adı genellikle "docker-compose.yml" olarak belirtilir.

Dosya, "version" anahtar kelimesiyle başlar, bu anahtar kelimesi Docker Compose sürümünü belirtir. Dosyanın geri kalanı, "services" anahtar kelimesi altında tanımlanan her konteyner için ayrı ayrı yapılandırma bilgilerini içerir. Her konteyner, bir "container\_name" ve bir "image" ile başlar. "container\_name", konteynerin ismini belirlerken, "image" ise konteynerin hangi Docker imajını kullanacağını belirler.

Ayrıca, her konteynerin hangi ağlarda bulunacağı, hangi portlara açık olacağı ve diğer çeşitli konfigürasyon ayarları "environment" ve "ports" anahtar kelimeleri altında belirtilir. Son olarak, Docker Compose dosyası, "networks" anahtar kelimesi altında belirtilen ağlara erişmek için kullanılacak ağ yapılandırmalarını içerir.

Bu dosya, proje için gerekli tüm konteynerlerin oluşturulması, konfigüre edilmesi ve birbirleriyle iletişim kurması için gerekli yapılandırma bilgilerini içerir.

Dosyanın başlangıcında, Compose dosyasının sürümü belirtilir. Bu dosya, sürüm 3 formatını kullanır. Dosya, "services" anahtar kelimesiyle başlayan bir blok içerir. Bu blok, kurulumu yapılacak servislerin listesini içerir. Şekil 3.3'deki örnekte, Elasticsearch adlı bir servis tanımlanmıştır. "container\_name" parametresi, oluşturulacak Docker konteynerinin ismini belirtir.



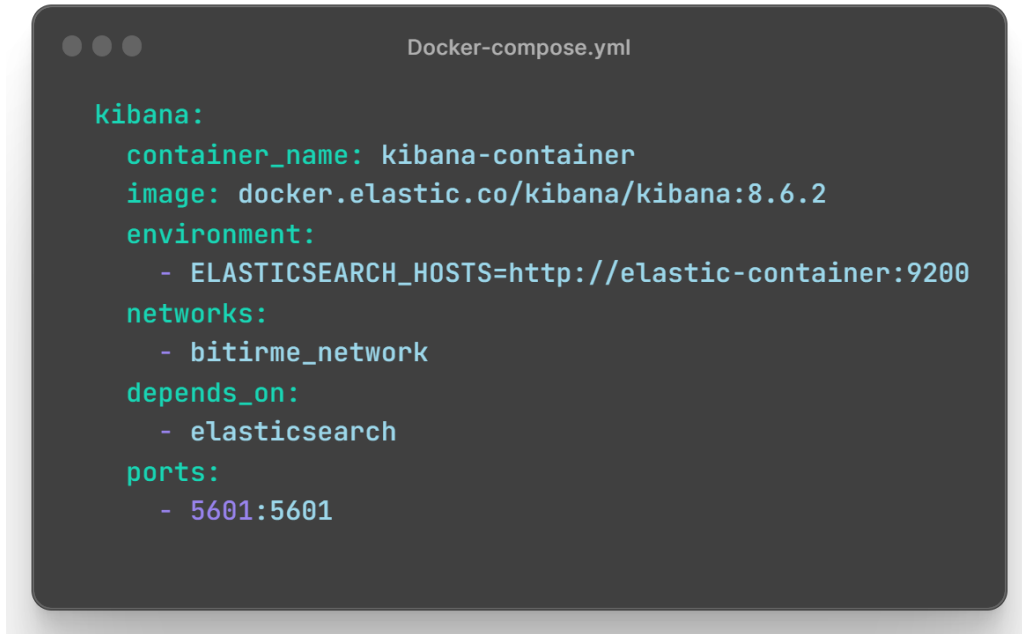
```
version: "3"
services:
  elasticsearch:
    container_name: elastic-container
    image: docker.elastic.co/elasticsearch/elasticsearch:8.6.2
    environment:
      - xpack.security.enabled=false
      - "discovery.type=single-node"
    networks:
      - bitirme_network
    ports:
      - 9200:9200
```

**Şekil 3.3 :** Elasticsearch servisi

"image" parametresi, kullanılacak Docker imajının adını belirtir. Burada, "docker.elastic.co/elasticsearch/elasticsearch:8.6.2" imajı kullanılacaktır.

"environment" parametresi, konteyner içinde kullanılacak ortam değişkenlerini belirtir. Bu örnekte, "xpack.security.enabled" ve "discovery.type" değişkenleri belirtilmiştir. "networks" parametresi, konteynerin hangi ağlara bağlanacağını belirtir. Bu örnekte, "bitirme\_network" adlı bir ağa bağlanacaktır. "ports" parametresi, konteynerin hangi portlardan erişilebileceğini belirtir. Bu örnekte, 9200 numaralı port dış dünyaya açık hale getirilmiştir.

Şekil 3.4’de, Elasticsearch ile birlikte çalışacak olan Kibana servisi oluşturulmaktadır. "kibana" adı verilen bu servis, "kibana-container" adında bir Docker konteyneri olarak ayarlanmaktadır.



**Şekil 3.4 : Kibana servisi**

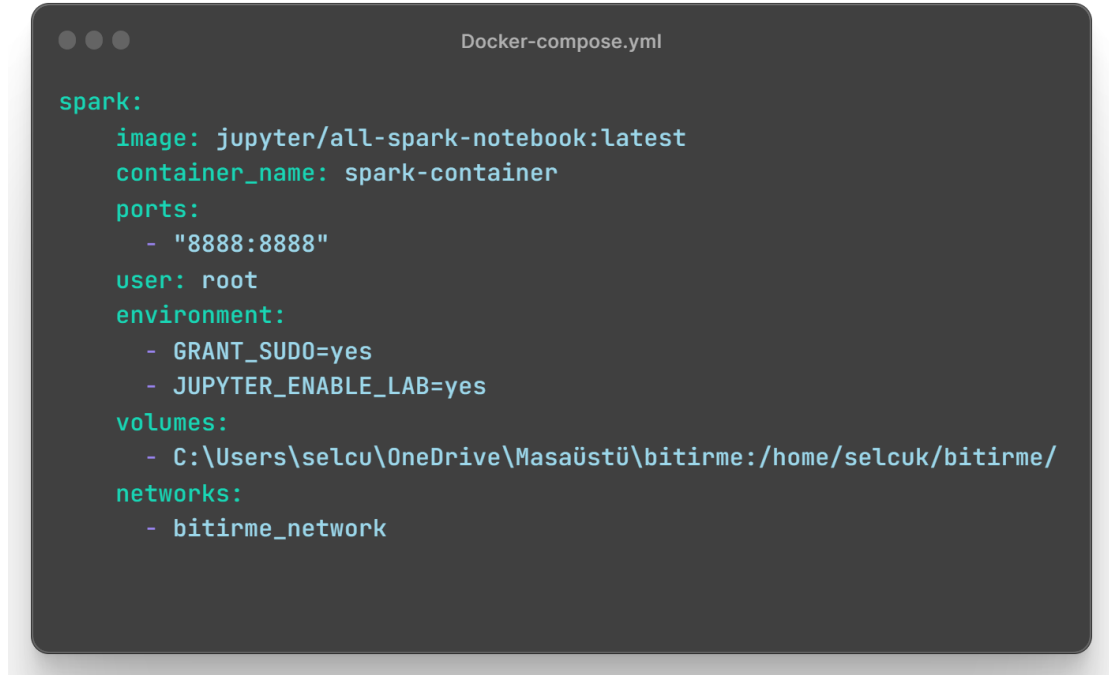
Kibana servisi, "docker.elastic.co/kibana/kibana:8.6.2" imajından yaratılır ve bu imaj, Kibana'nın en son sürümünü içermektedir.

Bu servis, "ELASTICSEARCH\_HOSTS" ortam değişkeni aracılığıyla Elasticsearch servisinin konumunu öğrenmektedir. Bu nedenle, Kibana'nın Elasticsearch ile iletişim kurabilmesi için Elasticsearch URL'si belirtilir.

Aynı şekilde Elasticsearch servisinde olduğu gibi, Kibana servisi de "bitirme\_network" ağına bağlanır ve Kibana servisi Elasticsearch servisine bağımlıdır. Bu, "elasticsearch" servisi çalışmadan "kibana" servisinin çalışmayacağı anlamında

gelmektedir. Kibana servisi "5601" portu üzerinden dışarıya açılır ve web arayüzüne erişim sağlanır.

Şekil 3.5’de "spark" adında bir servis tanımlanmaktadır. Bu servis, "jupyter/all-spark-notebook" adındaki Docker imajını kullanarak oluşturulur. Servis, "spark-container" adında bir Docker konteyneri olarak çalıştırılmakta ve 8888 numaralı portu host makineye yönlendirilerek erişim sağlanmaktadır.



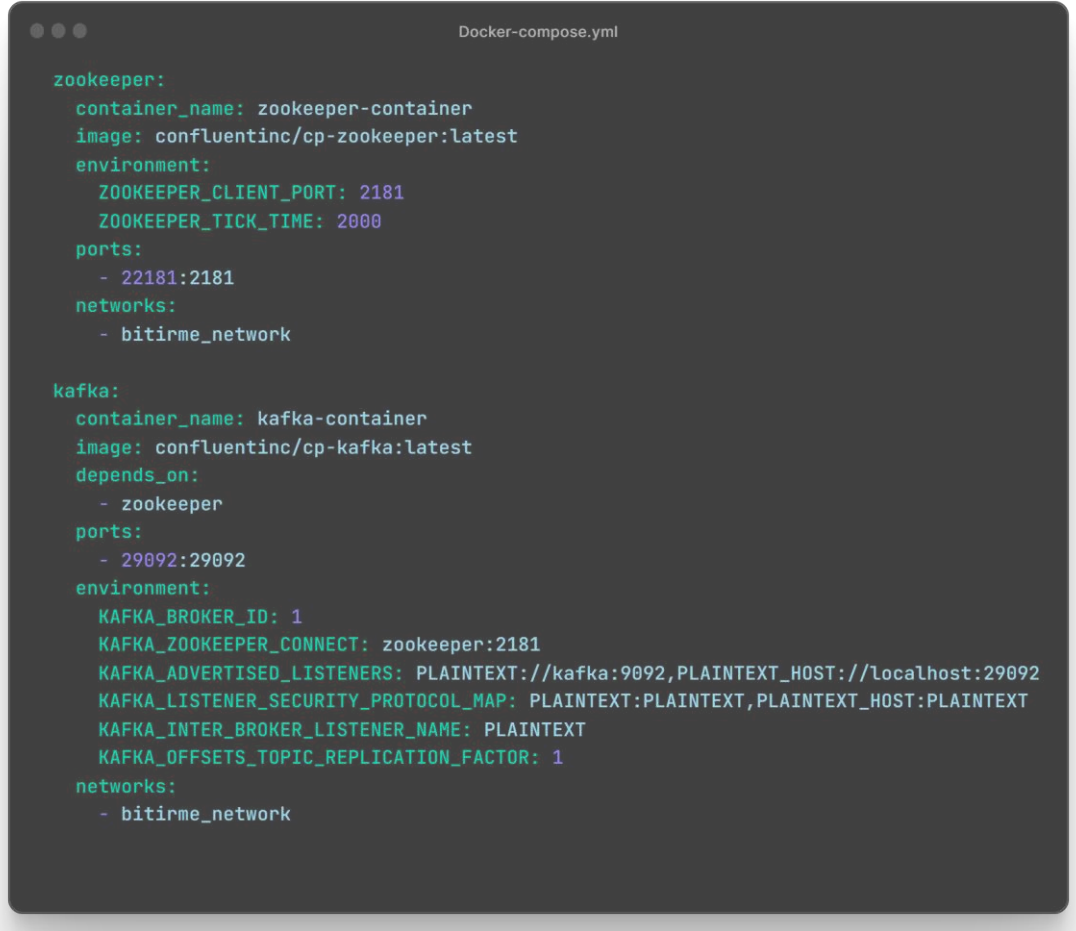
**Şekil 3.5 : Spark servisi**

Docker konteynerinin root kullanıcısı tarafından çalıştırılması için "user" anahtar kelimesi kullanılıp, "GRANT\_SUDO" ve "JUPYTER\_ENABLE\_LAB" gibi çevre değişkenleri belirlenmiştir.

Ayrıca, "C:\Users\selcu\OneDrive\Masaüstü\bitirme" dizini spark konteyneri içinde "/home/selcuk/bitirme/" konumuna bağlanmıştır. Local bilgisayar dizinindeki her geliştirme spark konteynerine de yansır, bu Docker'ın volume özelliği sayesinde sağlanmaktadır.

Son olarak servis, "bitirme\_network" adındaki Docker ağına bağlanıp diğer konteynerler ile iletişim kurabilmektedir.

Şekil 3.6’da ise "zookeeper" ve "kafka" adlı iki farklı konteyner oluşturulmaktadır.



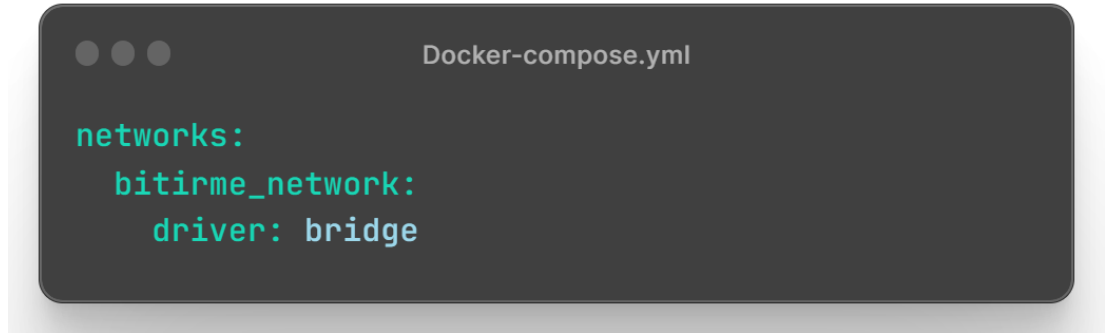
**Şekil 3.6 :** Kafka ve Zookeeper servisleri

"Zookeeper" konteyneri, "confluentinc/cp-zookeeper:latest" adlı Docker imajını kullanır ve "bitirme\_network" adlı bir Docker ağına bağlanır. Bu konteyner, "ZOOKEEPER\_CLIENT\_PORT" ve "ZOOKEEPER\_TICK\_TIME" adlı iki farklı ortam değişkeni ayarlar ve "22181" numaralı port'u kullanarak "2181" numaralı port'a yönlendirir.

"Kafka" konteyneri ise, "confluentinc/cp-kafka:latest" adlı Docker imajını kullanır ve o da "bitirme\_network" adlı Docker ağına bağlanır. Bu konteyner, "zookeeper" konteynerinin başlamasını bekler ("depends\_on") ve "29092" numaralı port'u kullanarak "29092" numaralı port'a yönlendirir. "Kafka" konteyneri ek olarak "KAFKA\_ZOOKEEPER\_CONNECT", "KAFKA\_ADVERTISED\_LISTENERS", "KAFKA\_LISTENER\_SECURITY\_PROTOCOL\_MAP", "KAFKA\_INTER\_BROKER\_LISTENER\_NAME", "KAFKA\_BROKER\_ID" ve "KAFKA\_OFFSETS\_TOPIC\_REPLICATION\_FACTOR" adlı farklı ortam değişkenleri ayarlar.

Bu değişkenler, "Kafka" konteynerinin "zookeeper" konteyneriyle iletişim kurulabilmesi ve doğru şekilde yapılandırılabilmesi için gerekmektedir.

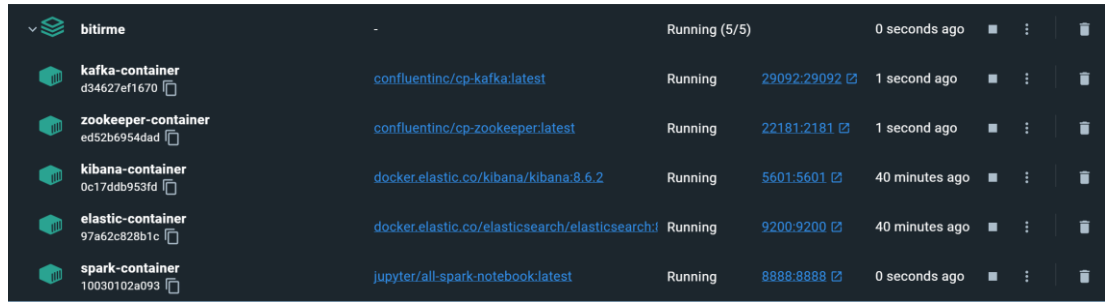
Şekil 3.7'deki kod bloğu ise "bitirme" adlı bir Docker ağı tanımlamak için kullanılmaktadır.



Şekil 3.7 : Docker bitirme\_network

"driver: bridge" satırı, Docker'ın varsayılan ağ sürücüsü olan bridge sürücüsünün kullanılacağını belirtir. Bridge sürücüsü, bir Docker ağı oluşturulduğunda, Docker'ın varsayılan ağında bir köprü oluşturur ve ağa bağlı olan konteynerlere IP adresleri atar. Bu sayede konteynerler arasında iletişim kurulabilir.

Şekil 3.8'de docker-compose.yml dosyasında yer alan servislerin çalıştığını gösteren docker desktoptan alınmış ekran görüntü yer almaktadır.



Şekil 3.8 : Docker Desktop'daki servislerin ekran görüntüsü

Docker-compose dosyasının çalıştırılıp servislerin ayağa kaldırılması için, yml dosyasının bulunduğu dizine gidip "docker-compose up" komutunun çalıştırılması yeterlidir.

### 3.4 Veri Kaynağı

Veri seti, UC Berkeley'deki Sutardja Dai Hall'un (SDH) 4 katındaki 51 odada kullanılan 255 sensör zaman serisinden toplanmıştır.



Her odada 5 tür ölçüm bulunmaktadır:

- CO2 konsantrasyonu
- Nem
- Sıcaklık
- Işık
- PIR: hareket sensörü

### **3.5 Veri Ön İşleme**

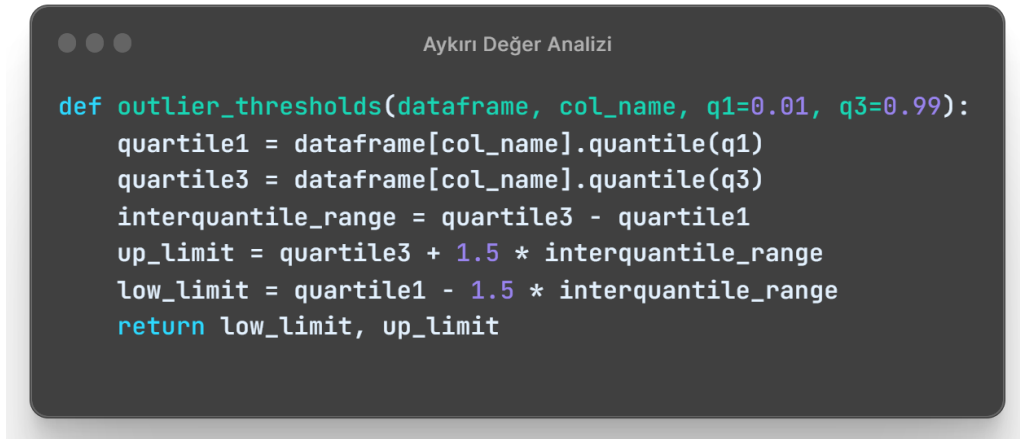
Veri ön işleme, makine öğrenmesi veya diğer veri analizi tekniklerinin uygulanabilmesi için verinin hazırlanması sürecidir. Veri ön işleme, veri analizi için kullanılacak verilerin doğru, tutarlı ve anlamlı olmasını sağlamak için veri temizleme, dönüştürme ve ölçeklendirme gibi çeşitli teknikleri içerir.

Veri ön işleme aynı zamanda veri setindeki özellikleri çıkararak veya yeni özellikler oluşturarak da veri setini zenginleştirebilir. Bu özellikler, makine öğrenmesi algoritmaları için daha anlamlı veriler sağlayabilir ve sonuçların doğruluğunu artırabilir.

Veri ön işleme, veri analizi veya makine öğrenmesi projelerinin önemli bir adımıdır ve doğru bir şekilde uygulandığında veri setlerinin kalitesini artırabilir ve sonuçların daha güvenilir hale gelmesini sağlayabilir.

#### **3.5.1 Aykırı değişken analizi**

Şekil 3.9'daki Python fonksiyonu, bir dataframe ve sütun adı (col\_name) parametrelerini alır. Fonksiyon, bu sütunun aykırı değerlerini tanımlamak için kullanılan üst ve alt sınır eşiklerini hesaplar.



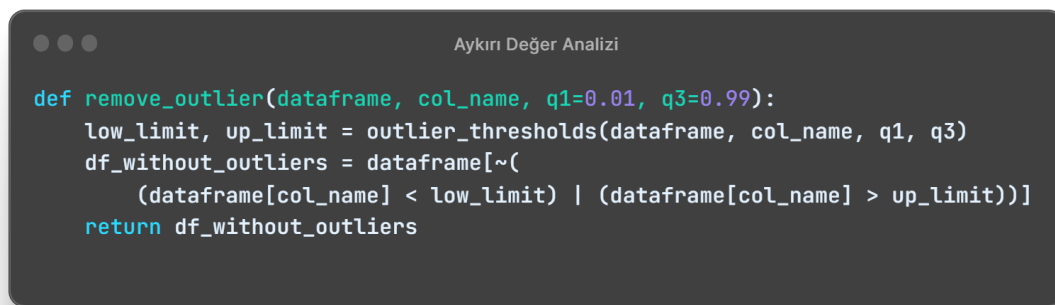
**Şekil 3.9 :** outlier\_thresholds fonksiyonu

Fonksiyon, önce veri çerçevesindeki belirtilen sütunun 1. ve 99. persentillerini (quartile1 ve quartile3) hesaplar. Ardından, bu sütunun interquartile range'ini (quartile3 - quartile1) bulur.

Aykırı değerleri tanımlamak için, üst ve alt sınırlar hesaplanır. Üst sınır, quartile3 değerine 1.5 interquartile range eklenmesiyle, Alt sınır ise quartile1 değerinden 1.5 interquartile range çıkarılmasıyla hesaplanır.

Son olarak, fonksiyon alt ve üst sınırları döndürür. Bu eşikler, veri setindeki aykırı değerleri tanımlamak için kullanılabilir. Eğer bir veri noktası alt veya üst sınırın dışında kalıyorsa, bu değer aykırı bir değer olarak kabul edilir.

Şekil 3.10'daki Python fonksiyonu, bir dataframe ve sütun adı (col\_name) parametrelerini alır.



**Şekil 3.10 :** remove\_outlier fonksiyonu

Fonksiyon, bu sütunun aykırı değerlerini belirlemek için öncelikle Şekil 4.9'daki "outlier\_thresholds" adlı fonksiyonu kullanır. Ardından, veri çerçevesinden aykırı değerleri çıkararak veri setindeki aykırı değerleri temizler.

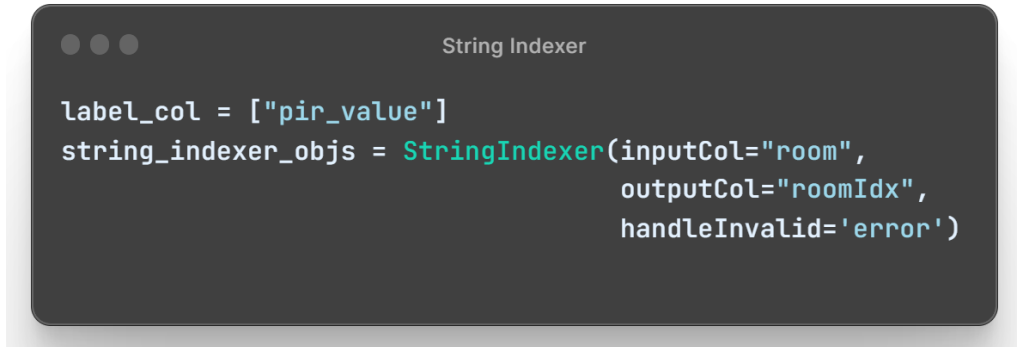
"outlier\_thresholds" fonksiyonu tarafından hesaplanan alt ve üst sınır eşikleri (low\_limit ve up\_limit), veri çerçevesindeki sütundaki her bir değerin bu sınırların içinde olup olmadığını kontrol etmek için kullanılır.

Fonksiyon, "~" işaretiyle negatif işlem kullanarak, alt veya üst sınırların dışında kalan tüm satırları filtreler. Yani, aykırı değerleri içeren satırlar hariç tutulur ve geriye kalan satırlar yeni bir veri çerçevesi olarak döndürülür.

Son olarak, fonksiyon temizlenmiş veri çerçevesini döndürür. Bu işlem, veri setindeki aykırı değerleri kaldırmak ve veri analizi ve modelleme işlemlerinde daha güvenilir sonuçlar elde etmek için kullanılabilir.

### 3.5.2 Encoding işlemleri

Şekil 3.11'deki bu işlem, "room" sütunundaki farklı kategori değerlerini sayılara dönüştürerek, bu sütundaki verilerin sayısal olarak kullanılabileceği bir formata dönüştürür.

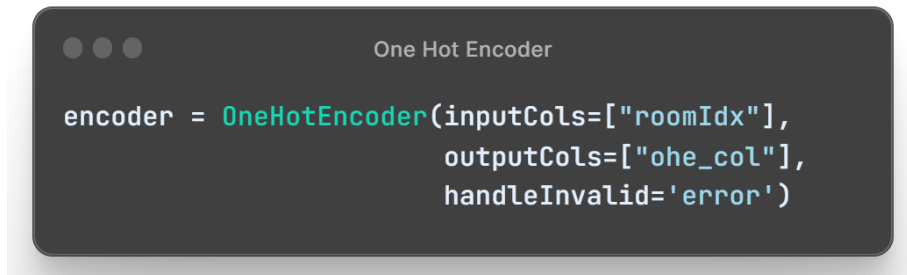


```
label_col = ["pir_value"]
string_indexer_objs = StringIndexer(inputCol="room",
                                     outputCol="roomIdx",
                                     handleInvalid='error')
```

Şekil 3.11 : String Indexer işlemi

Bu dönüşüm, makine öğrenimi modellerinin eğitimi ve kullanımı için gereklidir, çünkü genellikle bu modeller sayısal verilerle çalışırlar.

Şekil 3.12'deki bu kod ise, PySpark kütüphanesi kullanılarak "roomIdx" sütununu One-Hot Encoding işlemine tabi tutmak için kullanılır.



```
encoder = OneHotEncoder(inputCols=["roomIdx"],
                        outputCols=["ohe_col"],
                        handleInvalid='error')
```

Şekil 3.12 : One Hot Encoding işlemi

One-Hot Encoding, bir kategorik sütundaki farklı kategori değerlerini ikili bir vektör şeklinde kodlama işlemidir. Bu işlem sonucunda, her bir farklı kategori değeri, birer sütun olarak temsil edilir ve her satırda yalnızca bir sütunda 1 diğerlerinde 0 değeri bulunur. Bu işlem, makine öğrenimi modellerinde kullanılmak üzere veri setinin sayısal bir formata dönüştürülmesini sağlar.

VectorAssembler sınıfı, makine öğrenimi modellerinde kullanılmak üzere verileri birleştirmek için kullanılır. Bu sınıf, bir vektör olarak birleştirilmiş verileri temsil etmek için kullanılır. Şekil 3.13'deki bu işlem, bir veri setinin makine öğrenimi modeline beslenebilmesi için gerekli sayısal formata dönüştürülmesini sağlar.

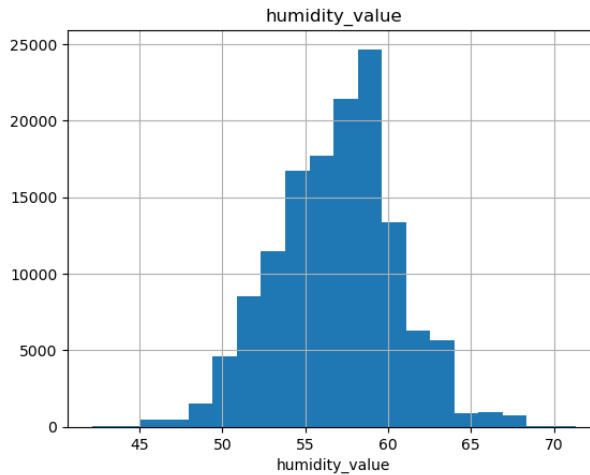
```
Vector Assembler

assembler = VectorAssembler(inputCols=['co2_value', "temp_value", "light_value",
                                         "humidity_value", 'ohe_col'],
                             outputCol='features',
                             handleInvalid='skip')
```

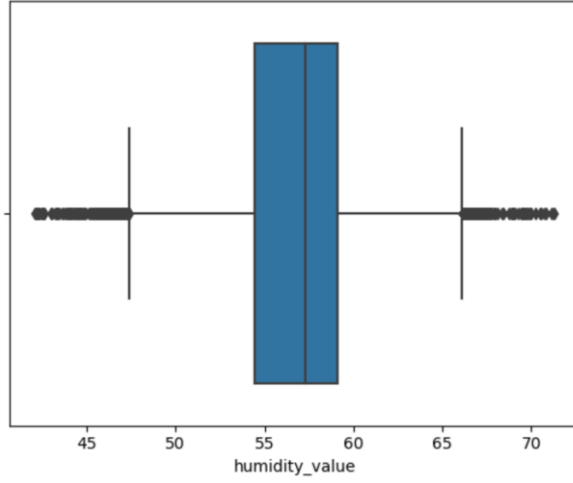
Şekil 3.13 : Vector Assembler

### 3.6 Veri Görselleştirme

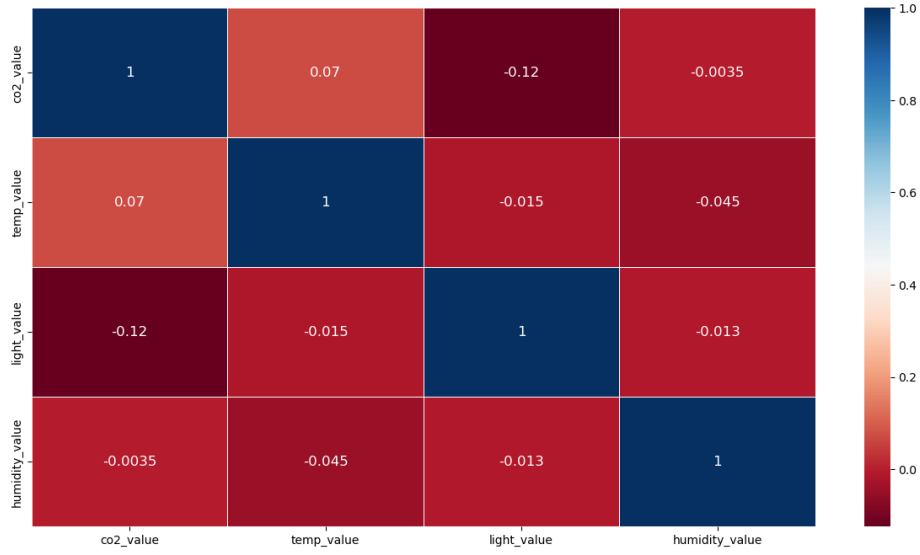
Veri görselleştirme, verileri grafikler, tablolar, haritalar, histogramlar ve diğer görsel öğeler gibi farklı yöntemler kullanarak görsel bir şekilde temsil etme işlemidir. Bu yöntem sayesinde, verilerin daha anlaşılır hale getirilmesi, veri analizinde kullanıcıların daha kolay bir şekilde bilgi edinmesi sağlanır. Ayrıca, veri görselleştirme, işletmelerin müşterileri ve pazarlar hakkında daha iyi bilgi edinmelerine yardımcı olabilir.



Şekil 3.14 : Humidite sütunu için histogram grafiği



**Şekil 3.15 :** Humidity değışkeni için aykırı değeri grafiğı

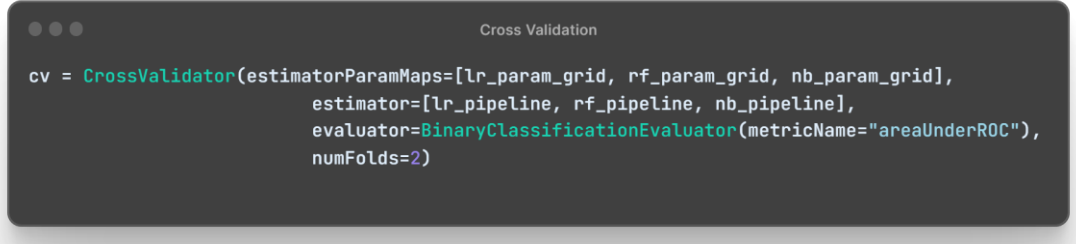


**Şekil 3.16 :** Sayısal değışkenlerin birbiri ile korelasyonunu gösteren ısı haritası

### 3.7 Veri Doğrulama (Validation)

Cross Validation, bir veri setini birden fazla eğitim ve test kümesi olarak ayırarak makine öğrenimi modellerini test etmek ve performanslarını ölçmek için kullanılır. "estimatorParamMaps" ve "estimator" parametreleri, test edilecek modellerin farklı hiperparametre değeriğerlerinin kombinasyonlarını içeren bir sözlükler listesi ve bu modellerin kendilerini içeren bir liste olarak belirtilir. "evaluator" parametresi, modellerin performansını değeriğerlendirmek için kullanılan bir ölçüttür. "numFolds" parametresi ise çapraz doğrulama için kullanılan kat sayısını belirtir.

Şekil 3.17'deki kod bloğı, PySpark kütüphanesi kullanılarak çapraz doğrulama işlemini gerçekleştirmek için CrossValidator sınıfını kullanır.



```
cv = CrossValidator(estimatorParamMaps=[lr_param_grid, rf_param_grid, nb_param_grid],
                    estimator=[lr_pipeline, rf_pipeline, nb_pipeline],
                    evaluator=BinaryClassificationEvaluator(metricName="areaUnderROC"),
                    numFolds=2)
```

**Şekil 3.17 :** Cross Validation işlemi

Burada, "lr\_pipeline", "rf\_pipeline" ve "nb\_pipeline" modelleri kullanılır. Bu modeller, sırasıyla Lojistik Regresyon, Rastgele Orman ve Naive Bayes sınıflandırıcılarıdır. Bu modeller, PySpark MLlib kütüphanesinde önceden tanımlanmış sınıflardır.

"evaluator" parametresi, sonuçları değerlendirmek için kullanılan bir ölçütü belirtir. Bu kod bloğunda, "BinaryClassificationEvaluator" sınıfı kullanılır ve "areaUnderROC" metriği belirlenir. Bu metrik, sınıflandırma performansını ölçmek için kullanılır.

### 3.8 Makine Öğrenmesi

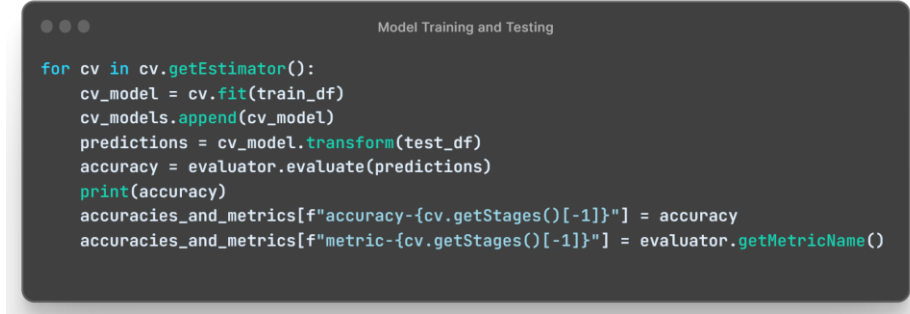
Makine öğrenmesi, yapay zeka alanında kullanılan ve bilgisayarların öğrenme yeteneğini geliştirmek için kullanılan bir yöntemdir. Bu yöntem, bilgisayarların deneyimlerinden öğrenmesine ve karar vermelerine olanak tanır. Bu nedenle, insan müdahalesine ihtiyaç duymadan, daha hızlı ve doğru sonuçlar elde edilebilir.

Makine öğrenmesi yöntemleri, genellikle denetimli öğrenme, denetimsiz öğrenme ve pekiştirmeli öğrenme olarak sınıflandırılabilir. Denetimli öğrenme, belirli bir sonuca ulaşmak için önceden belirlenmiş verilerin kullanımını gerektirir. Denetimsiz öğrenme, verilerin yapısal olarak önceden belirlenmediği durumlarda kullanılır ve kümelenme ve boyut azaltma gibi teknikleri içerir. Pekiştirmeli öğrenme ise, bir ajanın belirli bir çevreye karşı davranışını optimize etmesini amaçlayan bir yöntemdir.

Sonuç olarak, makine öğrenmesi, bilgisayarların kendi kendine öğrenmesi ve karar vermeleri için kullanılan bir yöntemdir. Büyük veri kümelerinin işlenmesi ve analiz edilmesi ile birlikte, birçok endüstride kullanılabilir ve sonuçları hızlı ve doğru olabilir.

### 3.8.1 Modelin eğitilmesi

Şekil 3.18'deki bu işlem, CrossValidator nesnesinin her bir tahmin modeli üzerinde döngü oluşturur. Ardından her bir modeli, train\_df veri kümesine uygun hale getirir.



```
Model Training and Testing

for cv in cv.getEstimator():
    cv_model = cv.fit(train_df)
    cv_models.append(cv_model)
    predictions = cv_model.transform(test_df)
    accuracy = evaluator.evaluate(predictions)
    print(accuracy)
    accuracies_and_metrics[f"accuracy-{cv.getStages()[-1]}"] = accuracy
    accuracies_and_metrics[f"metric-{cv.getStages()[-1]}"] = evaluator.getMetricName()
```

Şekil 3.18 : Model eğitim işlemi

Daha sonra, test\_df veri kümesinde bu modeli kullanarak tahminler oluşturur. Tahminlerin doğruluğunu hesaplar ve "accuracy" olarak adlandırılan bir değişkene atar. Ayrıca, accuracies\_and\_metrics adlı bir sözlük oluşturarak, her bir model için doğruluğu ve kullanılan metriği kaydeder. En son olarak, her bir modelin doğruluğunu ve kullanılan metriği yazdırmaktadır.

#### 3.8.1.1 Logistic Regression

Şekil 3.19'daki bu kod, bir lojistik regresyon modeli için bir makine öğrenimi pipeline'ı oluşturmayı ve çapraz doğrulama (cross-validation) kullanarak modelin hiperparametrelerini ayarlamayı sağlar. İlk olarak, LogisticRegression sınıfından bir lojistik regresyon modeli nesnesi oluşturulur. Bu model, ikili sınıflandırma problemleri için kullanılır. Daha sonra, bir Pipeline nesnesi olan lr\_final\_pipeline oluşturulur. Bu pipeline, veri işleme adımlarını ve lojistik regresyon modelini bir araya getirir. pipeline adımları sırasıyla şunlardır:

1. **string\_indexer\_objs**: Kategorik özellikleri sayısal değerlere dönüştürmek için kullanılan bir dizi **StringIndexer** nesnesi.
2. **encoder**: Kategorik özelliklerin sayısal değerlere dönüştürülmesini sağlayan bir **OneHotEncoder** nesnesi.
3. **assembler**: Özellik sütunlarını birleştirerek bir vektör özelliği oluşturan bir **VectorAssembler** nesnesi.
4. **logistic\_regression\_final**: Önceden oluşturulan lojistik regresyon modeli.

```
Logistic Regression

logistic_regression_final = LogisticRegression()
lr_final_pipeline = Pipeline().setStages([string_indexer_objs, encoder, assembler,
logistic_regression_final])
lr_final_param_grid = ParamGridBuilder().addGrid(logistic_regression_final.regParam, [0.01,
0.1, 1]).build()
cv_final = CrossValidator(estimatorParamMaps=lr_final_param_grid,
estimator=lr_final_pipeline,
evaluator=BinaryClassificationEvaluator(metricName="areaUnderROC"),
numFolds=5)
```

Şekil 3.19 : Logistic Regression Modeli

Daha sonra, ParamGridBuilder kullanılarak lojistik regresyon modelinin hiperparametrelerini ayarlamak için bir parametre ızgarası (lr\_final\_param\_grid) oluşturulur. Bu grid, regParam hiperparametresinin farklı değerlerini içerir. regParam, lojistik regresyon modelinin düzenleme parametresidir ve aşırı uydurmayı kontrol etmek için kullanılır.

Son olarak, CrossValidator nesnesi olan cv\_final oluşturulur. Bu nesne, çapraz doğrulama kullanarak modelin performansını değerlendirmeyi sağlar. estimatorParamMaps parametresi, lojistik regresyon modelinin hiperparametrelerinin farklı kombinasyonlarını içeren bir parametre grid'ini alır. estimator parametresi, kullanılacak olan pipeline'ı belirtir. evaluator parametresi, modelin performansını ölçmek için kullanılan bir değerlendirici nesnesini belirtir. numFolds parametresi ise çapraz doğrulama katlarının sayısını belirtir.

### 3.8.2 Model performansı değerlendirme

Şekil 3.20'deki bu kodda ise, labelCol parametresi "label" olarak ayarlanmış bir BinaryClassificationEvaluator nesnesi oluşturulur. final\_predictions veri kümesi üzerinde çalışan BinaryClassificationEvaluator'nın evaluate() metodunu çağırarak bir doğruluk puanı hesaplar ve bunu accuracy değişkenine atar.

```
Performance Evaluation

evaluator = BinaryClassificationEvaluator(labelCol="label")
accuracy = evaluator.evaluate(final_predictions)
evaluator.getMetricName()
```

Şekil 3.20 : Model Performans değerlendirme



**getMetricName()** metodunu çağırmak ise **BinaryClassificationEvaluator**'nın kullandığı performans metriklerinin adını döndürür. Bu örnekte, metrik adı "areaUnderROC" olarak ayarlandığı için **getMetricName()** metodu "areaUnderROC" döndürecektir. AUC değeri 0 ile 1 arasında değişir ve 1'e ne kadar yakınsa, model o kadar iyidir. Bu durumda, modelin performansı 0.9572868 gibi oldukça yüksek bir değerdir, bu nedenle modelin oldukça iyi performans gösterdiği söylenebilir.

### 3.9 Gerçek Zamanlı Analiz

Gerçek zamanlı analiz, büyük veri kaynaklarından gelen sürekli akan verilerin gerçek zamanlı olarak işlenmesi, analiz edilmesi ve tahminlemeler yapılması için kullanılan bir tekniktir. Bu teknik, geleneksel makine öğrenmesi yöntemlerinden farklıdır, çünkü verilerin birikmesini veya depolanmasını beklemek yerine, verilerin anlık olarak işlenmesi, analiz edilmesi ve tahminlenmesi için gerekmektedir.

Bu uygulama, birçok farklı uygulama alanında kullanılabilir. Örneğin, finansal piyasalar, hava durumu veya iot sensörler ile tarım gibi alanlarda kullanılabilir.

Gerçek zamanlı analiz, verilerin sürekli olarak değiştiği ve güncellendiği uygulama alanları için çok önemlidir. Bu teknik, veri analizinde daha hızlı, daha doğru ve daha güncel sonuçlar elde etmek için kullanılır.

#### 3.9.1 Kafka-python ile iot sensör simülasyonu

Şekil 3.21'de **df\_to\_kafka** adlı bir metot tanımlanır ve bu metot, DataFrame'deki verileri Kafka'ya göndermek için bir üretici oluşturur. Bu üretici, **self.topic** adlı Kafka topic'ine gönderim yapar.

```
Kafka Producer

def df_to_kafka(self):
    sayac = 0
    df_size = len(self.df) * self.repeat
    total_time = self.row_sleep_time * df_size
    for i in range(0, self.repeat):
        for index, row in self.df.iterrows():
            if self.key_index == 1000:
                self.producer.send(self.topic, key=str(index).encode(),
value=row[-1].encode())
            else:
                self.producer.send(self.topic, key=str(row[self.key_index]).encode(),
value=row[-1].encode())
            self.producer.flush()
            time.sleep(self.row_sleep_time)
            sayac = sayac + 1
            remaining_per = 100 - (100 * (sayac / df_size))
            remaining_time_secs = (total_time - (self.row_sleep_time * sayac))
            remaining_time_mins = remaining_time_secs / 60
        if sayac >= df_size:
            break
    self.producer.close()
```

**Şekil 3.21 : Sensör simülasyon fonksiyonu**

Metot, bir döngü içinde çalışır ve DataFrame'deki her bir satırı bir Kafka mesajına dönüştürür. Sonuç olarak, bu kod, bir Pandas DataFrame'deki verileri gerçek zamanlı olarak Kafka'ya göndermek için bir Kafka producer oluşturur ve DataFrame'deki her bir satırı bir Kafka mesajına dönüştürerek Kafka'ya gönderir.

### 3.9.2 Akan verinin elasticsearch'te depolanması

Şekil 3.22'de kafka'ya yazılmış veriler okunur ve gerçek zamanlı bir şekilde Elasticsearch'e kaydedilir. Okunan veriler virgülle ayrılmış bir dize olarak gelir ve bu dize ayrıştırılarak JSON formatına dönüştürülür.

```
ElasticSearch

class ToElastic(object):
    def __init__(self):
        self.es = Elasticsearch(ELASTIC["SERVER"])
        self.consumer = create_consumer()

    def write_to_elastic(self):
        for msg in self.consumer:
            message = msg.value
            timestamp = msg.timestamp / 1000
            date = datetime.fromtimestamp(timestamp, timezone(
                timedelta(hours=3))).strftime('%Y-%m-%dT%H:%M:%S.%f%z')
            string = message.decode("ascii").split(",")
            json_string = {
                "co2_value": float(string[0]),
                "temp_value": float(string[1]),
                "light_value": float(string[2]),
                "humidity_value": float(string[3]),
                "time": date,
                "room": str(string[4])
            }
            json_string = json.dumps(json_string)
            resp = self.es.index(
                index=ELASTIC["INDEX_INPUT"], body=json_string)
```

Şekil 3.22 : Elasticsearch'e veri yazan ToElastic sınıfı

Daha sonra, Elasticsearch REST API'sini kullanarak JSON verisi Elasticsearch endeksine yazılır. Bu, Kibana ile gerçekleştirilecek olan gerçek zamanlı görselleştirme işlemi için büyük öneme sahiptir.

### 3.9.3 Spark streaming ile gerçek zamanlı işleme

Şekil 3.23'de **get\_spark\_session** adlı bir metot tanımlanır ve bu metot, isteğe bağlı olarak **session\_params** adlı bir sözlük parametresi alır. Ancak, bu parametre metot içinde kullanılmamaktadır.

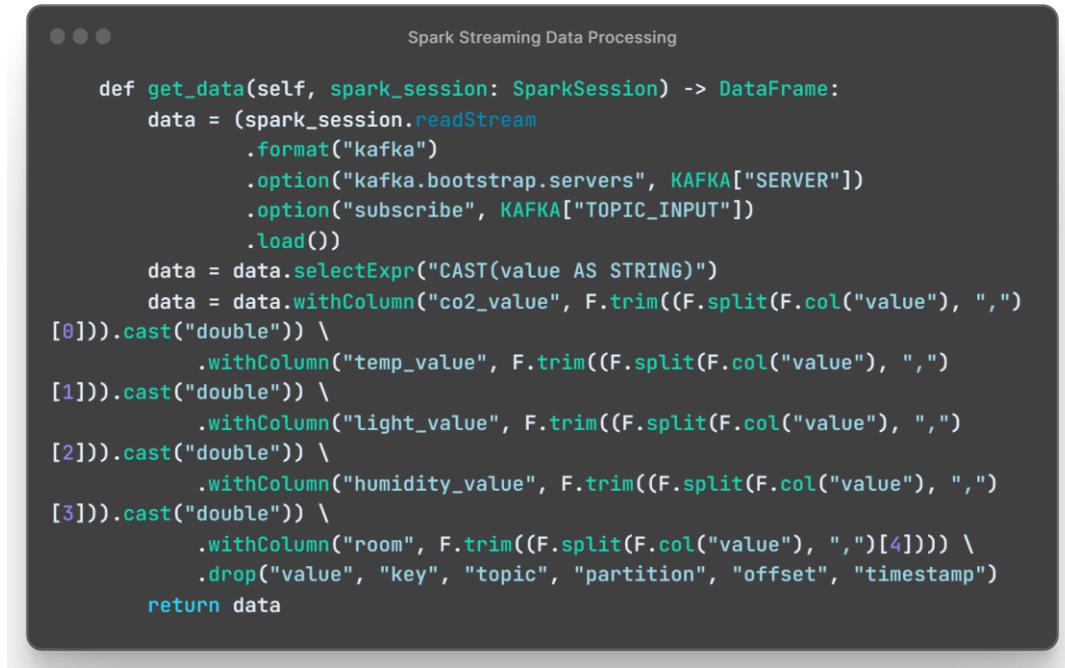
```
Spark streaming

def get_spark_session(self, session_params: dict = {}) -> SparkSession:
    spark = (SparkSession.builder
        .appName("kafka_streaming")
        .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-
10_2.12:3.1.1")
        .getOrCreate())
    return spark
```

Şekil 3.23 : Spark Session oluşturma fonksiyonu

Kod içersinde, **SparkSession.builder** kullanılarak "kafka\_streaming" adlı bir **SparkSession** nesnesi oluşturulur ve "spark.jars.packages" adlı bir yapılandırma ayarı ile Kafka ile etkileşim sağlamak için gerekli olan "spark-sql-kafka-0-10\_2.12:3.1.1" adlı paket yüklenir. Sonuç olarak, bu kod, Apache Spark'ı kullanarak Kafka veri akışı işleme için gerekli olan bir SparkSession nesnesi oluşturur ve bu nesneyi geri döndürür.

Şekil 3.24'de tanımlanan bu metot ise, **spark\_session** adlı bir SparkSession nesnesi alır ve Kafka'dan verileri okumak için bir **DataFrame** oluşturur.



```
def get_data(self, spark_session: SparkSession) -> DataFrame:
    data = (spark_session.readStream
            .format("kafka")
            .option("kafka.bootstrap.servers", KAFKA["SERVER"])
            .option("subscribe", KAFKA["TOPIC_INPUT"])
            .load())
    data = data.selectExpr("CAST(value AS STRING)")
    data = data.withColumn("co2_value", F.trim((F.split(F.col("value"), ",")
[0])).cast("double")) \
                .withColumn("temp_value", F.trim((F.split(F.col("value"), ",")
[1])).cast("double")) \
                .withColumn("light_value", F.trim((F.split(F.col("value"), ",")
[2])).cast("double")) \
                .withColumn("humidity_value", F.trim((F.split(F.col("value"), ",")
[3])).cast("double")) \
                .withColumn("room", F.trim((F.split(F.col("value"), ",")[4]))) \
                .drop("value", "key", "topic", "partition", "offset", "timestamp")
    return data
```

Şekil 3.24 : Gerçek zamanlı veri işleme fonksiyonu

Bu **DataFrame**, **KAFKA** adlı bir sözlükte tanımlanan Kafka ayarlarını kullanarak **KAFKA["TOPIC\_INPUT"]** adlı bir Kafka topic'inden gerçek zamanlı veri almaktadır.

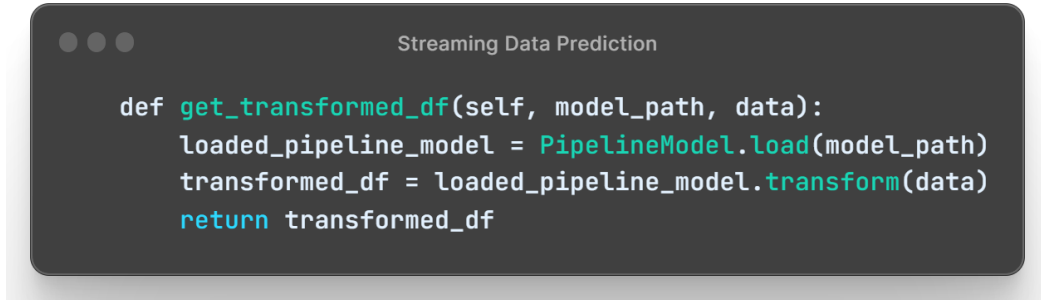
Gelen veriler, ilk olarak **data.selectExpr("CAST(value AS STRING)")** ifadesiyle **value** sütununu **string** olarak dönüştürür.

Daha sonra, verileri ayrıştırmak için **withColumn** ifadeleri kullanılarak birkaç yeni sütun eklenir. Bu yeni sütunlar, verilerdeki CO2, sıcaklık, ışık ve nem değerlerini içerir. Ayrıca, verilerdeki timestamp ve oda bilgisi de sütun olarak eklenir. Son olarak, gereksiz sütunlar (**value**, **key**, **topic**, **partition**, **offset**, **timestamp**) silinir ve oluşturulan **DataFrame** döndürülür.

Sonuç olarak, input verilerini okumak için Kafka kullanarak bir **DataFrame** oluşturulur. Bu **DataFrame** içindeki veriler ayrıştırılır ve uygun sütunlar eklenir.

### 3.9.4 Spark streaming ile gerçek zamanlı tahminleme

Şekil 3.25'deki bu metodun amacı, verileri önceden eğitilmiş bir ML modeli kullanarak tahminleme yapmaktır. ML Pipeline modeli, veri ön işleme, özellik mühendisliği, model eğitimi ve model tahmini gibi adımlar içerir.



```
def get_transformed_df(self, model_path, data):
    loaded_pipeline_model = PipelineModel.load(model_path)
    transformed_df = loaded_pipeline_model.transform(data)
    return transformed_df
```

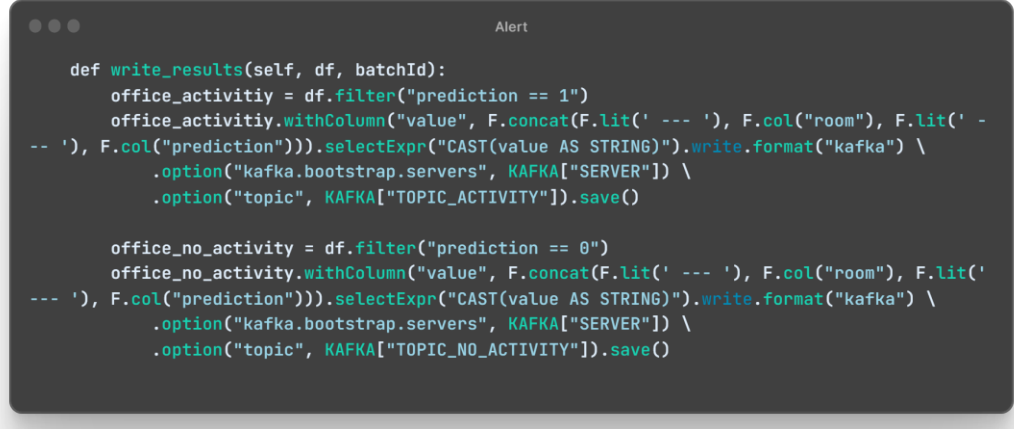
Şekil 3.25 : Gerçek zamanlı tahminleme fonksiyonu

**get\_transformed\_df** adlı metod iki parametre almaktadır: **model\_path** ve **data**. **model\_path** önceden eğitilmiş bir ML Pipeline modelinin kaydedildiği dizin yolunu, **data** ise bir Spark DataFrame'i içermektedir.

Metot, **PipelineModel.load** yöntemi kullanarak **model\_path** dizin yolundaki önceden eğitilmiş ML Pipeline modelini yükler. Daha sonra, **transform** yöntemi kullanarak **data** DataFrame'ini önceden eğitilmiş Pipeline modeli kullanarak dönüştürür, yani dataframe'deki verilerin her biri ile tahminleme gerçekleştirilir. Dönüştürülmüş DataFrame **transformed\_df** değişkenine atanır ve **transformed\_df** geri döndürülür.

#### 3.9.4.1 Tahmin sonucuna göre alarm gönderilmesi

Şekil 3.26'deki bu metod, model tahminlemelerinin Kafka'ya yazılmasında kullanılır. "prediction" sütunu değeri 1 ve 0 olan satırlar ayrı ayrı olarak "office\_activity" ve "office\_no\_activity" adlı değişkenlere filtrelenir. Her bir satırın "room" ve "prediction" sütunlarının birleştirilmesiyle bir "value" sütunu oluşturulur. Bu "value" sütunu "CAST" kullanılarak string'e dönüştürülür ve "KAFKA[SERVER]" değişkeninde tanımlanan Kafka sunucusuna ve "KAFKA[TOPIC\_ACTIVITY]" değişkeninde tanımlanan kafka topic'ine yazılır.



```
def write_results(self, df, batchId):
    office_activitiy = df.filter("prediction == 1")
    office_activitiy.withColumn("value", F.concat(F.lit(' --- '), F.col("room"), F.lit(' -
-- '), F.col("prediction"))).selectExpr("CAST(value AS STRING)").write.format("kafka") \
        .option("kafka.bootstrap.servers", KAFKA["SERVER"]) \
        .option("topic", KAFKA["TOPIC_ACTIVITY"]).save()

    office_no_activity = df.filter("prediction == 0")
    office_no_activity.withColumn("value", F.concat(F.lit(' --- '), F.col("room"), F.lit('
--- '), F.col("prediction"))).selectExpr("CAST(value AS STRING)").write.format("kafka") \
        .option("kafka.bootstrap.servers", KAFKA["SERVER"]) \
        .option("topic", KAFKA["TOPIC_NO_ACTIVITY"]).save()
```

**Şekil 3.26 :** Tahmin sonuçlarını ayrı kafka topiclerine gönderen fonksiyon

Bu işlemler gerçek zamanlı olarak gerçekleştirilir ve bir alarm mekanizması oluşturulmasını sağlar.

Şekil 3.27'daki **ToAlert** sınıfı ise, Kafka mesajlarını tüketen ve Elasticsearch veritabanına yazan bir **alert()** metodu içermektedir. Bu sınıftan bir nesne oluşturulduğunda **consumer** adlı bir Kafka tüketicisi ilklendirilir ve **alert()** metodu bu tüketicisi ile Kafka mesajlarını tüketir.

Mesajlar, **str** tipine dönüştürülür ve "---" işaretine göre bölünür. Bu bölünmüş mesajlar bir Python sözlüğüne (dictionary), bu sözlük de **json.dumps()** metodu kullanılarak bir JSON dizesine dönüştürülür. Son olarak, Elasticsearch **index()** yöntemi kullanılarak JSON dizesi veritabanına yazılır.

```
Alert

class ToAlert(object):
    def __init__(self) -> None:
        self.es = Elasticsearch(ELASTIC["SERVER"])
        self.consumer = create_consumer()

    def alert(self):
        while True:
            messages = self.consumer.poll()
            for topic_partition, message_list in messages.items():
                for message in message_list:
                    timestamp = message.timestamp / 1000
                    date = datetime.fromtimestamp(timestamp, timezone(
                        timedelta(hours=3))).strftime('%Y-%m-%dT%H:%M:%S.%f%z')
                    string = str(message.value.decode('utf-8')).split("---")
                    string = {
                        'date': date,
                        'room': string[1].strip(),
                        'alert-type': string[2].strip()
                    }
                    json_string = json.dumps(string)
                    self.es.index(
                        index=ELASTIC["INDEX_ALERT"], body=json_string)
                    if topic_partition.topic == KAFKA["TOPIC_ACTIVITY"]:
                        send_slack_message(json_string)
                    elif topic_partition.topic == KAFKA["TOPIC_NO_ACTIVITY"]:
                        continue
```

Şekil 3.27 : Gerçek zamanlı alarm oluşturan ToAlert sınıfı

Eğer gelen Kafka mesajı **KAFKA["TOPIC\_ACTIVITY"]** adlı topicten geldiyse, **send\_slack\_message()** fonksiyonu kullanılarak bir Slack kanalına bildirim gönderilir. Eğer mesaj **KAFKA["TOPIC\_NO\_ACTIVITY"]** adlı topicten geldiyse, hiçbir işlem yapılmaz ve bir sonraki mesaja geçilir.

## 4. UYGULAMA ÇIKTILARI

### 4.1 Iot Simülasyonu

Şekil 4.1'deki bu ekran görüntüsü, Kafka producer'ın çalıştırıldığı bir bash script'in çıktısıdır.

Script, bir CSV dosyasındaki (test veri seti) verileri okuyarak her bir satırı belirtilen bir Kafka topic'ine göndermektedir.

```
(base) root@10030102a093:/home/selcuk/bitirme/scripts# bash producer.sh
input: /home/selcuk/bitirme/test_df/data-simulation.csv
sep: ,
kafka_sep: ,
row_sleep_time: 1.0
repeat: 1
shuffle: False
self.excluded_cols: ['pir_value']
columns_to_write ['co2_value', 'temp_value', 'light_value', 'humidity_value', 'room']
topic: bitirme-input-1
key_index: 1000
bootstrap_servers: ['kafka-container:9092']
0 - 465.0,22.8,165.0,52.4,644
1/27049 processed, % 100.00 will be completed in 450.80 mins.
1 - 495.0,23.94,97.0,45.34,413
2/27049 processed, % 99.99 will be completed in 450.78 mins.
2 - 426.0,24.81,2208.0,49.1,668
3/27049 processed, % 99.99 will be completed in 450.77 mins.
3 - 439.0,23.11,4.0,53.51,558
4/27049 processed, % 99.99 will be completed in 450.75 mins.
4 - 410.0,23.56,4.0,51.25,448
5/27049 processed, % 99.98 will be completed in 450.73 mins.
5 - 395.0,23.46,3.0,46.03,452
6/27049 processed, % 99.98 will be completed in 450.72 mins.
6 - 367.0,22.84,194.0,52.87,421
7/27049 processed, % 99.97 will be completed in 450.70 mins.
7 - 368.0,23.5,3.0,51.35,454
8/27049 processed, % 99.97 will be completed in 450.68 mins.
8 - 349.0,22.3,199.0,52.02,721
9/27049 processed, % 99.97 will be completed in 450.67 mins.
9 - 554.0,25.26,100.0,46.49,656B
```

Şekil 4.1 : producer.sh örnek çıktısı

- input: Kafka'ya gönderilecek verinin kaydedildiği csv dosyasının yolu
- sep: csv dosyasındaki sütunları ayıran karakter
- kafka\_sep: Kafka mesajı içinde kullanılacak sütunları ayıran karakter
- row\_sleep\_time: her bir mesaj arasındaki süre
- repeat: aynı veriyi birden fazla kez göndermek istenirse kullanılacak sayı
- shuffle: verilerin rastgele sıralanıp sıralanmayacağı
- excluded\_cols: Kafka'ya gönderilmeyecek sütunlar

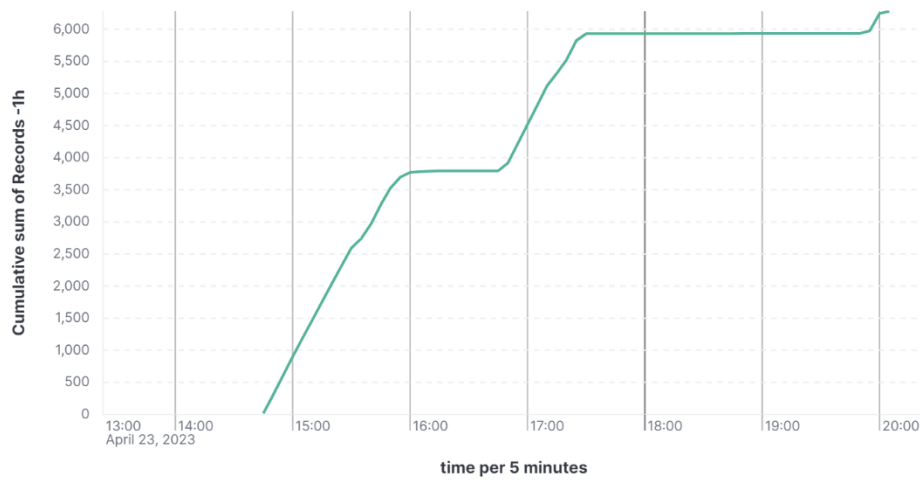


- `columns_to_write`: Kafka'ya gönderilecek sütunlar
- `topic`: verilerin gönderileceği Kafka topic adı
- `key_index`: Kafka mesajının key'inde kullanılacak sütunun index numarası
- `bootstrap_servers`: Kafka broker'larının adresleri

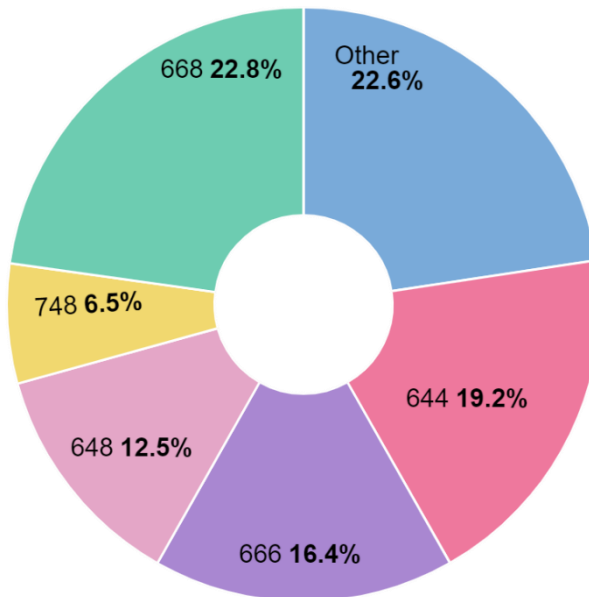
Kaç veri satırının işlendiği ve kaçının geri kaldığı da yüzdelik olarak belirtilmektedir.

Son olarak, toplam işlem süresi tahmini de verilmektedir. Bu tahmin, işlem hızına ve işlenecek toplam veri sayısına göre hesaplanmaktadır.

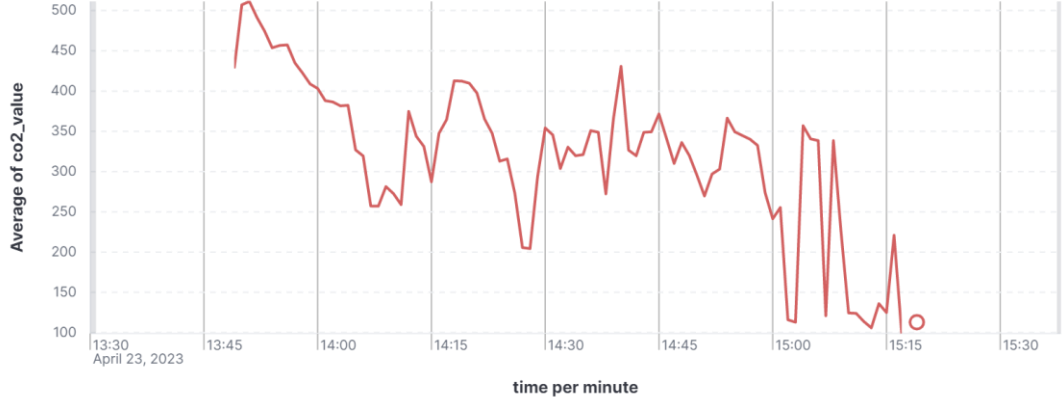
## 4.2 Gerçek Zamanlı Görselleştirme



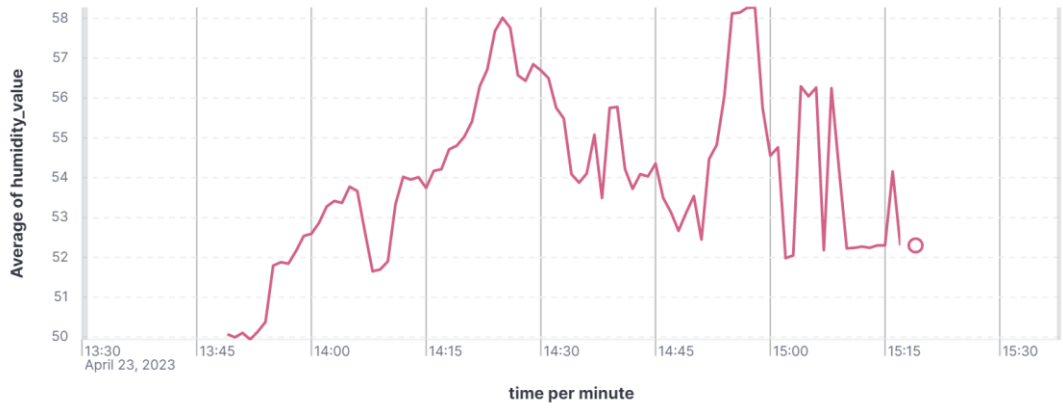
**Şekil 4.2 :** Kümülatif toplam kayıt sayısını gösteren çizgi grafiği



**Şekil 4.3 :** Odalara göre veri dağılımını gösteren pasta grafiği



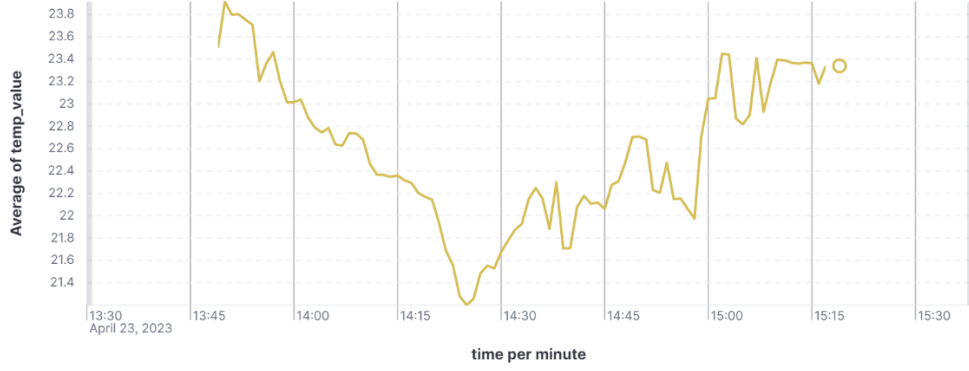
**Şekil 4.4 :** Anlık ortalama co2 değerini gösteren çizgi grafiği



**Şekil 4.5 :** Anlık ortalama humidity değerini gösteren çizgi grafiği



**Şekil 4.6 :** Anlık ortalama ışık değerini gösteren çizgi grafiği

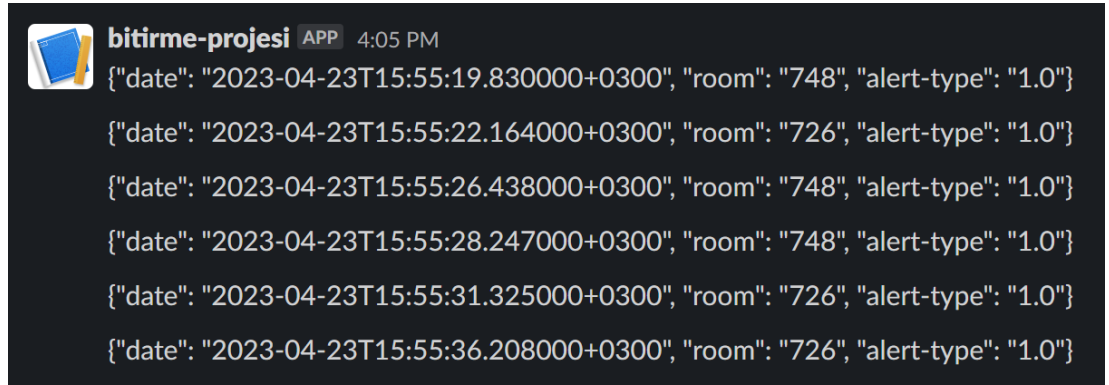


**Şekil 4.7 :** Anlık ortalama sıcaklık değerini gösteren çizgi grafiği

### 4.3 Alarm Gönderimi

Slack bildirimleri, kullanıcıların verileri hızlıca fark etmelerini ve gerektiği durumlarda acil önlemler almalarını sağlayacak şekilde tasarlanmıştır. Bu sayede kullanıcılar, sensörlerden gelen verileri anlık olarak takip edebilir ve olası sorunlara hızlıca müdahale edebilirler.

Şekil 4.8'deki bu örnek çıktı, bildirimi oluşturan verileri içermektedir. Bu veriler, odanın numarası, uyarı tipi ve tarih saat bilgisini içerir.



**Şekil 4.8 :** Örnek Slack bildirimi

## 5. SONUÇ

Sonuç olarak bu proje, IoT sensörlerinden toplanmış gerçek verilerin gerçek zamanlı işlenmesine ve oda koşullarını kontrol etmek için bir makine öğrenimi modeli kullanılmasına dayalı olarak yenilikçi bir yaklaşım benimsemektedir.

Projede kullanılan Docker, Elasticsearch, Kibana, Kafka ve Spark gibi açık kaynaklı araçların birleştirilmesi, birden fazla veri kaynağının analiz edilmesini ve hızlı bir şekilde işlenmesini mümkün kılarak daha iyi sonuçlar elde edilmesine yardımcı olur.

Makine öğrenimi algoritmalarının kullanımı ise, odadaki insan varlığının tespit edilmesi gibi ilgili birçok uygulama için potansiyel sağlar. Ayrıca, projenin ilerleyen aşamalarda enerji tasarrufu gibi birçok potansiyel fayda sağlaması da beklenmektedir.

### 5.1 Çalışmanın Uygulama Alanı

Bu çalışmanın uygulama alanı oldukça geniştir. Öncelikle, IoT teknolojisinin gelişmesiyle birlikte, çeşitli endüstriyel, ticari ve ev tipi cihazlar artık internete bağlanarak veri toplayabilmektedirler. Bu veriler, bir otomasyon sistemi ile analiz edilerek daha doğru sonuçlar elde edilebilir ve bu sayede, enerji tasarrufu, maliyet azaltımı, daha verimli işletmeler ve daha konforlu yaşam alanları gibi birçok alanda faydalar sağlanabilir.

Örneğin, bu projeyle bir ofis ortamında sensörler aracılığıyla toplanan veriler kullanılarak, insan varlığı tespit edilebilir ve bu sayede aydınlatma, ısıtma ve soğutma sistemleri otomatik olarak kontrol edilebilir. Böylece, enerji tasarrufu sağlanıp ve çalışanların konforu artırılabilir. Ek olarak, endüstriyel tesislerde de sensörler aracılığıyla toplanan veriler sayesinde, makine arızaları önceden tespit edilebilir ve önleyici bakım yapılabilir. Bu da üretim süreçlerinin daha verimli hale getirilmesine yardımcı olur.

Bu projenin kullanım alanları sadece endüstriyel uygulamalarla sınırlı kalmamakla birlikte, ev otomasyon sistemleri için de bu teknolojiye yararlanılabilir. Örneğin, akıllı termostatların, sensörler aracılığıyla evde kimse bulunmadığı zamanlarda ısıtma ya da soğutma sistemlerinin kapatılması pek çok yönden fayda sağlayacaktır.

Sonu olarak, IoT sensörlerinin kullanımı ve bu verilerin doğru şekilde analiz edilmesi, birçok alanda faydalar sağlar ve gelecekte de bu alanda daha fazla yenilik ve gelişme beklenmektedir.

## 6. KAYNAKLAR

**A. Geron.**(2019), Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd ed. O'Reilly Media,

**A. K. M. Nasim, M. I. Hossain, and M. A. Matin.** "Internet of Things (IoT) based Smart Homes: A Review," International Journal of Computer Applications, vol. 167, no. 8, pp. 1-6, 2017.

**Apache Kafka official documentation.** Erişim: 23 Mayıs 2023, <https://kafka.apache.org/documentation/>

**Apache Spark documentation.** Erişim: 23 Mayıs 2023, <https://spark.apache.org/docs/latest/>

**B. Radovanovic, S. Milenkovic, and I. Radovanovic.** (2017), "A Review of Internet of Things for Smart Home: Challenges and Solutions," Journal of Cleaner Production, vol. 140, pp. 1454-1464.

**Elasticsearch official documentation.** Erişim: 23 Mayıs 2023, <https://www.elastic.co/guide/index.html>

**Kibana official documentation.** Erişim: 23 Mayıs 2023, <https://www.elastic.co/guide/en/kibana/current/index.html>

**Docker Compose official documentation.** Erişim: 23 Mayıs 2023, <https://docs.docker.com/compose/>

**M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica.** (2010), "Spark: Cluster Computing with Working Sets," HotCloud, vol. 10, pp. 10-10

**S. Rashidi, R. G. Clegg, and N. Mohd-Nasir.** (2016), "A Review of Machine Learning Techniques for Smart Homes," Energy and Buildings, vol. 130, pp. 687-707.

**S. S. Hasan, M. R. Hasan, and M. J. Islam.** (2020), "IoT-Based Smart Home Automation: A Systematic Review," IEEE Internet of Things Journal, vol. 7, no. 5, pp. 3816-3830.

**Şirin E., Github Repository (2022),** Erişim: 23 Mayıs 2023, [https://github.com/erkansirin78/data-generator/blob/master/dataframe\\_to\\_kafka.py](https://github.com/erkansirin78/data-generator/blob/master/dataframe_to_kafka.py)

## ÖZGEÇMİŞ

TARANMIŞ  
VESİKALIK  
FOTOĞRAF

**Ad-Soyad** : Selçuk Şan  
**Doğum Tarihi ve Yeri** : 27/10/2000, Adana  
**E-posta** : selcuk1330@gmail.com

## BİTİRME ÇALIŞMASINDAN TÜRETİLEN MAKALE, BİLDİRİ VEYA SUNUMLAR:

- .....
- .....
- .....