

# Gerçek Zamanlı Büyük Veri Analitiği ile Anomali Tespiti: Spark ve Kafka Entegrasyonu

## Büyük Veri Analizine Giriş

Aydın Kasımoğlu

Github Linki:

Erdem Karadeniz

211307011  
211307011@kocaeli.edu.tr

[https://github.com/aydinkasimoglu/big\\_data](https://github.com/aydinkasimoglu/big_data)

221307100  
221307100@kocaeli.edu.tr

### I. ÖZET

Bu proje, büyük veri analitiği ve anomali tespiti üzerine odaklanarak, gerçek zamanlı veri işleme ve makine öğrenimi tekniklerini bir araya getirmektedir. İlk adımda, anomali tespiti için çeşitli veri kümeleri seçilmiş ve ön işleme adımları uygulanmıştır. Bu işlemler eksik verilerin işlenmesi, verinin normalize edilmesi, kategorik değişkenlerin kodlanması ve aykırı değerlerin tespiti gibi süreçleri kapsamaktadır. Veri setinin genel özellikleri grafiklerle görselleştirilerek analiz edilmiştir.

Makine öğrenimi ve derin öğrenme algoritmaları, anomali tespiti için eğitilmiştir. Kullanılan modellerin başarı metrikleri detaylı bir şekilde raporlanmış ve sonuçlar doğruluk (accuracy), duyarlılık (precision), hatırlama (recall) ve F1 skoru gibi ölçütlerle değerlendirilmiştir.

Apache Kafka, projede veri akışı altyapısı olarak kullanılmıştır. Kafka Producer, modele gerçek zamanlı veri sağlarken; Kafka Consumer, anomali tespit sonuçlarını farklı topic'ler altında toplamaktadır. Apache Spark, gerçek zamanlı veri işleme ve anomali tespiti için entegre edilmiştir. Spark Streaming ile işlenen veriler, Kafka topic'lerine aktarılmıştır.

Son olarak, proje detaylı bir raporla belgelenmiş ve GitHub üzerinde kodlarla birlikte paylaşılmıştır. Rapor, veri seti özellikleri, ön işleme adımları, model performansı ve entegrasyon süreçlerini içermektedir.

### Abstract

This project focuses on big data analytics and anomaly detection by integrating real-time data processing and machine learning techniques. Initially, datasets suitable for anomaly detection were selected and underwent preprocessing steps such as handling missing data, normalizing the dataset, encoding categorical variables, and detecting outliers. The general characteristics of the datasets were analyzed using various visualization techniques.

Machine learning and deep learning algorithms were trained to detect anomalies. The performance of the models was thoroughly reported and evaluated using metrics such as accuracy, precision, recall, and F1 score.

Apache Kafka was utilized as the data streaming infrastructure in this project. Kafka Producer provided real-time data to the model, while Kafka Consumer collected anomaly detection results under different topics. Apache Spark was integrated for real-time data processing and anomaly detection. Data processed through Spark Streaming was transferred to Kafka topics.

Finally, the project was documented in detail and shared on GitHub with the accompanying code. The report includes dataset properties, preprocessing steps, model performance, and integration processes.

### II. Veri Ön İşleme ve Görselleştirme

Bu çalışmada finansal işlem anomali veri seti üzerinde işlem tutarlarının dağılımını analiz etmek için Python dilinde veri görselleştirme ve ön işleme yapılmıştır. İlk olarak, eksik veriler dropna yöntemi ile temizlenmiştir. Daha sonra işlem tutarlarının dağılımı, hem lineer ölçekte hem de logaritmik ölçekte histogramlarla görselleştirilmiştir.

- Veri Okuma ve Temizleme:** Veri kümesi, pandas kütüphanesi kullanılarak okunmuş ve eksik değerler (NaN) kaldırılmıştır:

```
import pandas as pd

data = pd.read_csv("financial_anomaly_data.csv")

[44]

Handle null values

data.dropna(inplace=True)

data

[45]

...
```

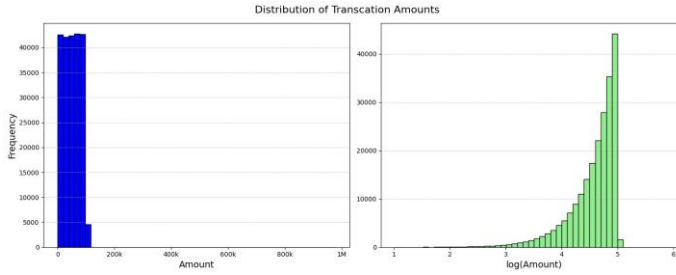
2. **Logaritmik Dönüşüm:** İşlem tutarlarının dağılımını daha iyi analiz edebilmek için logaritmik dönüşüm uygulanmıştır

3. **Histogram Grafikleri:** İşlem tutarlarının hem orijinal değerleri hem de logaritmik dönüşümü için histogramlar oluşturulmuştur:  
İlk grafikte işlem tutarlarının (Amount) frekans dağılımı gösterilmiştir.  
İkinci grafikte logaritmik dönüşüm uygulanmış işlem tutarlarının frekans dağılımı yer almaktadır.  
Histogramlar matplotlib kütüphanesi kullanılarak şu şekilde oluşturulmuştur:

```
import matplotlib.pyplot as plt
import numpy as np

log_amount = np.log10(data['Amount'])

fig, ax = plt.subplots(1, 2, figsize=(15, 6))
ax[0].hist(data['Amount'], bins=50, color='blue', edgecolor='black')
ax[0].set_xlabel("Amount", fontsize=14)
ax[0].set_xticks(ticks=[0, 200_000, 400_000, 600_000, 800_000, 1_000_000],
                  labels=['0', '200k', '400k', '600k', '800k', '1M'])
ax[0].set_ylabel("Frequency", fontsize=14)
ax[0].grid(axis='y', linestyle='--', alpha=0.7)
ax[1].hist(log_amount, bins=50, color='lightgreen', edgecolor='black')
ax[1].set_xlabel("log(Amount)", fontsize=14)
ax[1].grid(axis='y', linestyle='--', alpha=0.7)
fig.suptitle("Distribution of Transaction Amounts", fontsize=16)
fig.tight_layout()
```



(Grafik 1)

#### Grafik Yorumlama:

- İlk grafikte, işlem tutarlarının çoğunlukla düşük değerlerde yoğunlaştığı, yüksek tutarlardaki işlemlerin nadir olduğu gözlemlenmiştir.
- İkinci grafikte, logaritmik dönüşüm kullanılarak işlem tutarlarının dağılımı daha dengeli bir şekilde sunulmuş ve değerlerin daha iyi analiz edilmesi sağlanmıştır.

Bu görselleştirmeler, veri setindeki işlem tutarlarının dengesizliklerini ve olası anomalileri tespit etmek için önemli ipuçları sunmaktadır.

### III. Veri Analizi ve Grafik Yorumlama

Bu bölümde finansal işlem verisi üzerinde yapılan **IQR (Interquartile Range)** yöntemiyle aykırı değer tespiti ve zaman serisi analizi çalışmaları açıklanmaktadır.

#### Aykırı Değer Tespiti (IQR Yöntemi)

IQR yöntemiyle, işlem tutarlarının dağılımında istatistiksel olarak aykırı (anormal) değerler tespit edilmiştir. Bu yöntem şu adımlarla uygulanmıştır:

##### 1. Çeyrekler ve IQR Hesaplama:

- Verinin 1. çeyreği (Q1) ve 3. çeyreği (Q3) hesaplanmıştır.
- IQR (Interquartile Range), Q3 ile Q1 arasındaki fark olarak belirlenmiştir.

```
Q1 = data['Amount'].quantile(0.25)
Q3 = data['Amount'].quantile(0.75)
```

##### 2. Alt ve Üst Sınırların Belirlenmesi:

- Alt sınır:  $Q1 - 1.5 * IQR$
- Üst sınır:  $Q3 + 1.5 * IQR$
- Bu sınırların dışında kalan değerler aykırı olarak işaretlenmiştir.

##### 3. Aykırı Değerlerin Etiketlenmesi:

- Target adlı yeni bir sütun oluşturulmuş ve aykırı değerler 1, normal değerler 0 olarak etiketlenmiştir:

```
data['Target'] = ((data['Amount'] < lower_bound) | (data['Amount'] > upper_bound)).astype(int)
data['Target'].value_counts()

... Target
0    216949
1      11
Name: count, dtype: int64
```

##### 4. Sonuçların Analizi:

- data['Target'].value\_counts() çıktısına göre veri setinde **11 aykırı değer**, **216,949 normal değer** bulunmaktadır.

#### Zaman Serisi Analizi

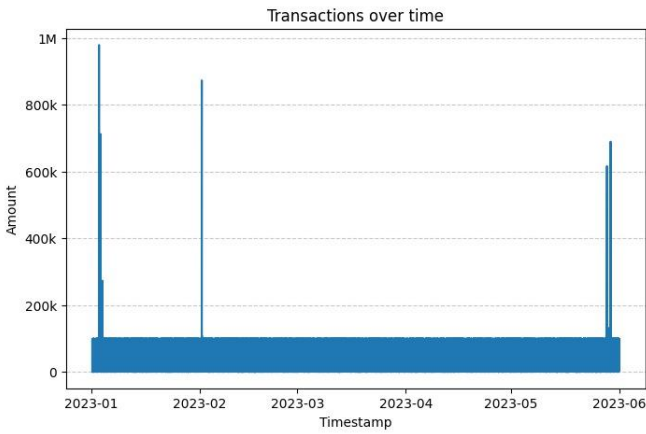
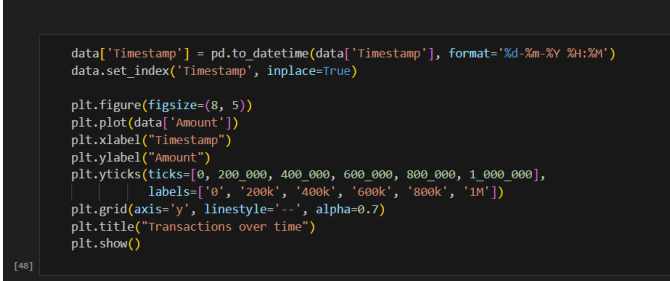
Aykırı değerlerin tespiti sonrası veri, işlem tarihine göre düzenlenmiş ve işlem tutarlarının zamana bağlı değişimi analiz edilmiştir. Bu süreç şu şekilde gerçekleştirilmiştir:

##### 1. Zaman Damgasının Formatlanması:

Timestamp sütunu datetime formatına dönüştürülmüş ve indeks olarak ayarlanmıştır.

## 2. Zaman Serisi Grafiğinin Çizimi:

İşlem tutarlarının zamana göre değişimi bir çizgi grafikte görselleştirilmiştir (Görsel 2) :



(Görsel 2)

### Grafik Yorumlama

Grafikte, işlem tutarlarının zamana göre dağılımı görülmektedir. Çoğu işlem düşük tutarlarda yoğunlaşırken, bazı zamanlarda yüksek işlem tutarları gözlemlenmiştir. Bu yüksek değerler, IQR yöntemiyle aykırı olarak tespit edilen 11 işlem arasında yer alabilir.

- **Düşük İşlem Tutarları:** Çoğu işlem 200,000 birimin altında yoğunlaşmaktadır. Bu durum normal işlemleri temsil etmektedir.
- **Yüksek İşlem Tutarları:** Grafikte, özellikle belirli zaman aralıklarında (ör. Ocak, Şubat ve Haziran aylarında) büyük işlemler dikkat çekmektedir. Bu değerler potansiyel olarak aykırı (anormal) olarak etiketlenmiştir.

Bu analiz, veri setindeki işlem tutarlarının dağılımını anlamaya ve anormal işlem tespitine katkı sağlamaktadır.

## IV. Boxplot ile Veri Analizi

Bu bölümde, işlem tutarlarının dağılımını görselleştirmek için **boxplot** (kutu grafiği) kullanılarak gerçekleştirilen analiz açıklanmıştır.



Bu kod, **Amount** sütunundaki verilerin istatistiksel dağılımını analiz etmek için bir kutu grafiği üretmektedir.

### Kutu Grafiği Açıklaması

Kutu grafiği, bir veri setindeki merkezi eğilim ve dağılımı görselleştirmek için kullanılan bir yöntemdir. Aşağıda, grafiğin unsurları açıklanmıştır:

#### 1. Kutu (Box):

Kutunun sol ve sağ sınırları, 1. çeyrek (Q1) ve 3. çeyrek (Q3) değerlerini temsil eder. Kutunun içindeki çizgi, **medyan** değeri gösterir. Bu, veri setindeki ortanca değerdir.

#### 2. İçerik:

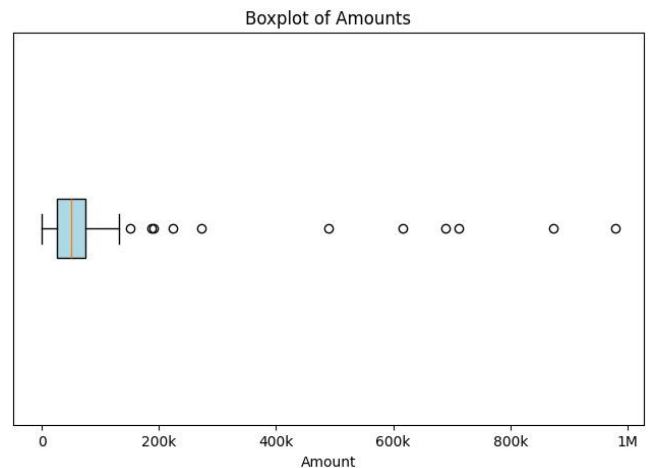
Kutunun boyutu, çeyrekler arası aralığı (**IQR**) gösterir ve verilerin büyük çoğunluğunun bu aralıkta olduğunu ifade eder.

#### 3. Whisker (Kuyruk):

Whisker'lar, aykırı olmayan en düşük ve en yüksek değerleri temsil eder.

#### 4. Aykırı Değerler (Outliers):

Kutunun ve whisker'ların dışında yer alan noktalar, aykırı değerlerdir. IQR yöntemine göre aykırı olarak tespit edilen işlem tutarları, grafikte bu noktalarda gözlemlenebilir.



(Görsel 3)

### Grafik Yorumlama

Grafikten çıkarılan bulgular şunlardır:

#### 1. Dağılımın Yoğunluğu:

Verilerin büyük bir kısmı düşük işlem tutarları (0 - 200,000 birim aralığında) içerisinde yoğunlaşmıştır. Kutu (IQR) bu aralıkta yer almakta ve grafiğin çoğunu kaplamaktadır.

#### 2. Medyan Değeri:

Verilerin ortanca değeri (medyan) düşük bir işlem tutarına işaret etmektedir ve kutu içinde bir çizgi ile gösterilmektedir.

#### 3. Aykırı Değerler:

Grafikte kutunun ve whisker'ların çok uzağında yer alan noktalar, aykırı (anormal) değerlerdir. Bu değerler daha önce IQR yöntemine göre tespit edilen **11 aykırı değer** ile uyum göstermektedir. Özellikle 800,000 ve 1,000,000 gibi yüksek işlem tutarları, grafikte net bir şekilde aykırı olarak öne çıkmaktadır.

### Sonuç

Bu analiz, işlem tutarlarının istatistiksel özetini ve aykırı değerlerin grafikte görselleştirilmesini sağlamıştır. Grafikteki aykırı değerler, işlem tutarlarının normal aralığın çok üzerinde olduğunu ve anormal işlem olarak ele alınabileceğini göstermektedir. Bu durum, veri temizleme, risk analizi veya dolandırıcılık tespiti gibi işlemler için önemli bir adım oluşturur.

## V. Verilerin Standardizasyonu ve Özellik Çıkartımı

### 1. Miktar Sütununun Standardizasyonu

### Standardizing values

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

data[['Amount']] = scaler.fit_transform(data[['Amount']])

data
```

[50]

...

### Açıklama:

- **StandardScaler:** Veriyi standardize etmek için kullanılan bir araçtır. Bu yöntem, veri setindeki her bir değeri **ortalama 0** ve **standart sapma 1** olacak şekilde yeniden ölçekler.
- **Amaç:**
  - Farklı özellikler (ör. işlem tutarı) arasında ölçek farklılıklarını ortadan kaldırmak.
  - Özellikle makine öğrenimi algoritmalarıyla çalışırken, standardizasyon veri doğruluğunu artırır.
- **Sonuç:**
  - Amount sütunundaki işlem tutarları, standart normal dağılıma uygun hale getirilmiştir. Bu, işlem tutarlarını daha anlamlı bir şekilde analiz etmeye ve modellemeye olanak tanır.

### 2. Kategorik Sütunların Etiketlenmesi (Label Encoding)

### Handle Categorical Variables

```
from sklearn.preprocessing import LabelEncoder

# Initialize LabelEncoder
le_merchant = LabelEncoder()
le_transaction_type = LabelEncoder()
le_location = LabelEncoder()

# Apply label encoding
data['Merchant'] = le_merchant.fit_transform(data['Merchant'])
data['TransactionType'] = le_transaction_type.fit_transform(data['TransactionType'])
data['Location'] = le_location.fit_transform(data['Location'])

data
```

### Açıklama:

- **Label Encoding:** Kategorik verileri sayısal değerlere dönüştürmek için kullanılan bir tekniktir.
- **Amaç:**
  - Merchant, TransactionType ve Location gibi kategorik değişkenlerin makine öğrenimi modellerinde işlenebilmesi için sayısal formatta ifade edilmesi.
  - Örneğin, Merchant sütunundaki farklı mağaza isimleri sırasıyla 0, 1, 2 gibi sayılarla temsil edilir.
- **Sonuç:**
  - Kategorik veriler, makine öğrenimi algoritmalarında kullanılabilir hale getirilmiştir.

### 3. Tarihsel Özelliklerin Çıkarılması

```
...
# Extract features
data['Hour'] = data.index.hour
data['DayOfWeek'] = data.index.weekday
data['Month'] = data.index.month
data['DayOfMonth'] = data.index.day
```

#### Açıklama:

- Bu kod, Timestamp sütunundan yeni zaman bazlı özellikler türetir:
  - Hour:** İşlemin yapıldığı saat (0-23 arasında).
  - DayOfWeek:** Haftanın günü (0 = Pazartesi, 6 = Pazar).
  - Month:** İşlemin yapıldığı ay.
  - DayOfMonth:** Ayın günü.
- Amaç:**
  - Zaman bazlı analizler ve desenlerin daha kolay tespit edilebilmesi.
  - Örneğin, hangi saat aralığında anormal işlemlerin gerçekleştiği belirlenebilir.
- Sonuç:**
  - Zaman damgasından türetilen bu özellikler, işlem sıklığı ve dolandırıcılık tespiti gibi analizlerde kullanılabilir.

#### 4. Gereksiz Sütunların Çıkarılması

```
data.drop(['TransactionID', 'AccountID'], axis=1, inplace=True)
data
```

#### Açıklama:

- TransactionID** ve **AccountID** gibi sütunlar modelleme veya analiz için gerekli değildir, çünkü bu sütunlar genellikle benzersiz kimlik bilgileri içerir ve öğrenme algoritmaları üzerinde etkisizdir.
- Amaç:**
  - Veri setini daha temiz ve anlamlı hale getirmek.
- Sonuç:**
  - İşleme ve analiz sürecinde kullanılmayacak sütunlar kaldırılmıştır.

#### 5. Ön İşlenmiş Verilerin Kaydedilmesi

```
[53] data.to_csv("preprocessed_data.csv")
```

#### Açıklama:

- Ön işleme adımlarından geçen veri seti, `preprocessed_data.csv` adlı bir dosyaya kaydedilmiştir.
- Amaç:**
  - Veriyi tekrar kullanmak veya başka analizlerde kullanabilmek için kalıcı olarak saklamak.
- Sonuç:**
  - İşlenmiş veri seti, kolay erişim ve paylaşım için CSV formatında kaydedilmiştir.

#### Genel Yorum

Bu veri ön işleme adımları, veri setini makine öğrenimi modelleriyle analiz yapmaya uygun hale getirmek için önemlidir. Yapılan işlemler sayesinde:

- Veriler standart bir forma getirilmiş ve farklı ölçeklendirme problemleri giderilmiştir.
- Kategorik değişkenler sayısal hale getirilerek modelleme için uygun hale getirilmiştir.
- Zaman damgasından türetilen özellikler, işlem alışkanlıklarını ve olası anormallikleri analiz etmek için yeni açılımlar sağlamıştır.
- Gereksiz sütunların kaldırılmasıyla veri seti daha yalın ve etkili bir hale getirilmiştir.

Sonuç olarak, bu işlemler veri hazırlama sürecinin temel adımlarını temsil eder ve daha karmaşık analizler veya modelleme süreçleri için bir temel oluşturur.

### VI. Makine Öğrenimi Modelinin Eğitimi: Random Forest Classifier Kullanımı

Bu bölümde, veriler üzerinde makine öğrenimi algoritmasını uygulamak için bir model geliştirilmiştir. Aşağıdaki adımlar izlenmiştir:

#### 1. Gerekli Kütüphanelerin Yüklenmesi

İlk olarak, Python programlama dili için gerekli kütüphaneler (scikit-learn, numpy, matplotlib) `%pip install` komutu ile yüklenmiştir. Bu kütüphaneler, model oluşturma, veri analizi ve görselleştirme gibi işlemler için gereklidir.

#### 2. Veri Setinin Yüklenmesi

Ön işleme aşamasında oluşturulan `preprocessed_data.csv` dosyası, pandas kütüphanesi kullanılarak yüklenmiştir.

Zaman bilgisi (Timestamp) analize dahil edilmemiştir ve drop() metodu ile veri setinden çıkarılmıştır. Bu işlem, modelin gereksiz bilgilerle karmaşıklaşmasını önlemeyi amaçlamaktadır.

```
import pandas as pd

data = pd.read_csv("preprocessed_data.csv")
data.drop(columns=['Timestamp'], inplace=True)
data
```

### 3. Hedef Değişkenin Ayrılması ve Veri Bölünmesi

Veri setinde, hedef değişken (Target) ile bağımsız değişkenler (X) ayrıştırılmıştır. Veriler, eğitim ve test setlerine bölünmüştür:

- **Eğitim Seti (%70):** Modelin öğrenme süreci için kullanılan veri.
- **Test Seti (%30):** Modelin performansını değerlendirmek için ayrılan veri.

train\_test\_split() fonksiyonu ile veri, dengeli bir şekilde stratifiye edilmiştir, yani sınıflar arasındaki dağılım korunmuştur.

### 4. Random Forest Classifier Modelinin Eğitilmesi

Veri setine uygun olarak, RandomForestClassifier algoritması seçilmiştir. Bu algoritma, karar ağaçlarının bir topluluğundan oluşur ve anomali tespitinde etkili sonuçlar verir.

- **n\_estimators:** 100 karar ağacı kullanılmıştır.
- **class\_weight:** Veriler arasında dengesizlik olması durumunda sınıf ağırlıkları ayarlanmıştır.
- **random\_state:** Modelin tutarlı sonuçlar üretmesi için rastgelelik sabitlenmiştir.

Model, fit() metodu kullanılarak eğitim seti (X\_train, y\_train) üzerinde eğitilmiştir.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split

X = data.drop(columns=['Target'])
y = data['Target']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42, stratify=y)

model = RandomForestClassifier(n_estimators=100, random_state=42, class_weight='balanced')
model.fit(X_train, y_train)
```

### 5. Tahminlerin Üretilmesi

Test veri kümesi (X\_test) kullanılarak, model tahminlerde bulunmuştur. predict() metodu, bağımsız değişkenleri kullanarak tahmin edilen hedef değişkenleri (y\_pred) oluşturmuştur.

### 6. Model Performansının Değerlendirilmesi

Modelin performansı, classification\_report fonksiyonu kullanılarak analiz edilmiştir. Bu raporda aşağıdaki metrikler hesaplanmıştır:

- **Precision (Kesinlik):** Modelin pozitif sınıfları doğru bir şekilde tahmin etme oranı.
- **Recall (Duyarlılık):** Gerçek pozitiflerin, model tarafından doğru bir şekilde tespit edilme oranı.
- **F1-Score:** Precision ve Recall'un harmonik ortalaması.
- **Support:** Her bir sınıfın veri setindeki örnek sayısı.

Ayrıca, confusion\_matrix fonksiyonu ile modelin sınıflandırma doğruluğu bir matris biçiminde görselleştirilebilir.

```
from sklearn.metrics import classification_report, confusion_matrix

y_pred = model.predict(X_test)
report = classification_report(y_test, y_pred)

print(report)
```

	precision	recall	f1-score	support
0	1.00	1.00	1.00	65085
1	1.00	1.00	1.00	3
accuracy			1.00	65088
macro avg	1.00	1.00	1.00	65088
weighted avg	1.00	1.00	1.00	65088

### 7. Sonuçların İncelenmesi

Aşağıdaki sonuçlar elde edilmiştir:

- **Class 0 (Normal Veriler):** Precision, Recall ve F1-Score değerleri 1.00'dır, yani model bu sınıfta tamamen doğru tahmin yapmıştır.
- **Class 1 (Anomaliler):** Precision, Recall ve F1-Score değerleri 1.00'dır. Bu, modelin anormal verileri de eksiksiz şekilde tespit ettiğini göstermektedir.
- **Accuracy:** Genel doğruluk oranı %100'dür.
- **Macro Avg:** Ortalama performans değerleri, sınıf dağılımını dikkate alır.
- **Weighted Avg:** Sınıf destek oranlarına göre ağırlıklı ortalama değerlerdir.

### VII. Makine Öğrenimi Modelinin Performans Değerlendirmesi: Confusion Matrix Analizi

Makine öğrenimi modelinin performansını görselleştirmek için **Confusion Matrix** (Karışıklık Matrisi) kullanılmıştır. Confusion Matrix, sınıflandırma modelinin başarımını



değerlendirmek için kullanılan etkili bir araçtır ve modelin hangi sınıfta doğru veya yanlış tahminler yaptığını açıkça gösterir.

### Kullanılan Kodlar

Aşağıdaki kod parçacığı, Confusion Matrix'in Seaborn ve Matplotlib kullanılarak görselleştirilmesini sağlar:

#### 1. Confusion Matrix'in Hesaplanması

confusion\_matrix fonksiyonu ile gerçek etiketler (y\_test) ve tahmin edilen etiketler (y\_pred) karşılaştırılarak matristeki değerler elde edilmiştir.

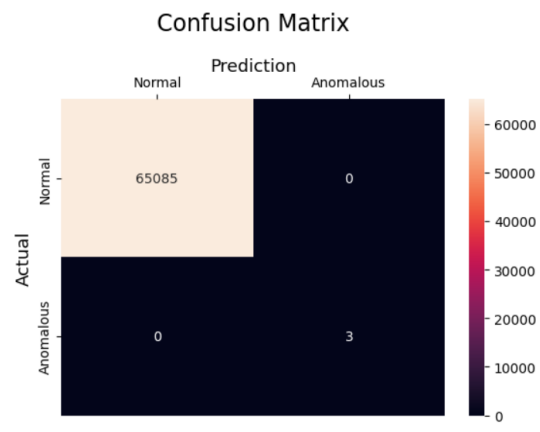
#### 2. Görselleştirme

- Seaborn Heatmap:** Matrisin ısı haritası olarak görselleştirilmesi sağlanmıştır.
- Etiketler:** X eksen (Tahmin) ve Y eksen (Gerçek) etiketlenerek, Normal ve Anomalous sınıfları belirtilmiştir.
- Matris Değerleri:** Hücrelere modelin tahmin ettiği sayıların anotasyonu eklenmiştir.

```
import seaborn as sns
import matplotlib.pyplot as plt

conf_matrix = confusion_matrix(y_test, y_pred)

sns.heatmap(conf_matrix, annot=True, fmt='g', xticklabels=['Normal', 'Anomalous'], yticklabels=['Normal', 'Anomalous'])
plt.xlabel('Actual', fontsize=11)
plt.ylabel('Prediction', fontsize=11)
plt.title('Confusion Matrix', fontsize=12, pad=20)
plt.grid(True, linestyle='dotted', color='gray')
plt.show()
```



### Confusion Matrix Görselleştirme Sonuçları

Confusion Matrix grafiği aşağıdaki çıktıyı göstermektedir:

- Normal (65085):** Tüm normal örnekler doğru bir şekilde tahmin edilmiştir.
- Anomalous (3):** Anomalous sınıfındaki tüm örnekler doğru bir şekilde sınıflandırılmıştır.
- Yanlış Tahmin:** Yanlış pozitif veya yanlış negatif tahmin bulunmamaktadır.

Grafik, modelin tüm örnekleri doğru bir şekilde sınıflandırdığını ve hatasız çalıştığını açıkça göstermektedir.

### VII. Makine Öğrenimi Modelinin Performans Değerlendirmesi: Azınlık Sınıf Analizi ve Yeni Veri Tahminleri

#### 1. Model Tahminlerinin DataFrame Olarak Saklanması

```
results = pd.DataFrame({
    'Actual': y_test,
    'Predicted': y_pred
})

# Optionally, include features from the test set for more context
results_with_features = X_test.copy()
results_with_features['Actual'] = y_test.values
results_with_features['Predicted'] = y_pred

# Display the first few rows
results_with_features.head()
```

- Amaç:** Test setindeki gerçek değerler (Actual) ile modelin tahmin ettiği değerler (Predicted) bir DataFrame formatında saklanır.
- Gelişmiş Analiz:** Test setinin özellikleri (örneğin, işlem bilgileri) eklenerek tahminlerin daha ayrıntılı bir şekilde analiz edilmesi sağlanır.

#### 2. Azınlık (Minority) Sınıf Tahminlerinin İncelenmesi

```
minority_class_predictions = results_with_features[results_with_features['Actual'] == 1]
minority_class_predictions
```

- Amaç:** Gerçek değerleri 1 (anomalous) olan örnekler, modelin bu sınıftaki performansını daha iyi incelemek için filtrelenmiştir.
- Sonuç:** Bu analiz, azınlık sınıfındaki tahminlerin doğru olup olmadığını ve hangi özelliklere sahip olduklarını incelemek için kullanılır.

#### 3. Yeni Anomali Veri Üzerinde Tahminler

```

> anomalous_data = pd.DataFrame({
    'Amount': [23.1221, 56.4752],
    'Merchant': [0, 8],
    'TransactionType': [2, 1],
    'Location': [5, 6],
    'Hour': [3, 23],
    'DayOfWeek': [6, 0],
    'Month': [12, 1],
    'DayOfMonth': [25, 1]
})

anomalous_preds = model.predict(anomalous_data)

anomalous_data['Predicted'] = anomalous_preds

anomalous_data
[8]
...

```

- **Amaç:** Yeni oluşturulan anomalous verilere model uygulanarak tahminlerin değerlendirilmesi sağlanır.
- **Özellikler:** Her veri noktası işlem miktarı, işlem tipi, konum, saat, gün, ay gibi özelliklere sahiptir.
- **Sonuç:** Modelin yeni anomali verileri doğru sınıflandırıp sınıflandıramadığı test edilir.

#### 4. Yeni Normal Veri Üzerinde Tahminler

```

> normal_data = pd.DataFrame({
    'Amount': [1.523, 0.52313],
    'Merchant': [7, 6],
    'TransactionType': [1, 2],
    'Location': [2, 2],
    'Hour': [13, 13],
    'DayOfWeek': [1, 3],
    'Month': [3, 3],
    'DayOfMonth': [12, 1]
})

normal_preds = model.predict(normal_data)

normal_data['Predicted'] = normal_preds

normal_data
[9]
...

```

- **Amaç:** Yeni oluşturulan normal verilere model uygulanarak tahminlerin değerlendirilmesi sağlanır.
- **Sonuç:** Modelin yeni normal verileri doğru bir şekilde sınıflandırıp sınıflandıramadığı test edilir.

#### Sonuç

Bu kodlar, modelin hem mevcut test seti hem de yeni oluşturulan veriler üzerindeki performansını detaylı bir şekilde analiz etmiştir. Azınlık sınıflarına özel analiz yapılmış ve modelin bu sınıfları sınıflandırma yeteneği incelenmiştir. Ayrıca, hem normal hem de anomali sınıflar için yeni verilerle modelin genelleme kapasitesi test edilmiştir.

### VIII. Kafka Entegrasyonu

#### 1. Gerekli Kütüphanelerin İçte Aktarılması

```

1 from kafka import KafkaProducer
2 import pandas as pd
3 import json

```

**Amaç:** Kafka kullanımı için KafkaProducer sınıfı ve veri manipülasyonu için pandas kütüphanesi yüklenmiştir. Ayrıca mesajları JSON formatına dönüştürmek için json modülü kullanılmıştır.

#### 2. Veri Setinin Yüklenmesi ve Hazırlanması

```

5 data = pd.read_csv("preprocessed_data.csv")
6
7 data.drop(columns=["Timestamp"], inplace=True)
8
9 data["kafka_message"] = data.apply(lambda row: json.dumps(row.to_dict()), axis=1)
10

```

**Amaç:** Veri, preprocessed\_data.csv dosyasından yüklenir. İşlemler için gereksiz olan Timestamp sütunu veri setinden kaldırılır. Her satır, JSON formatına dönüştürülerek kafka\_message sütununda saklanır.

#### 3. Kafka Producer Nesnesinin Oluşturulması

**Amaç:** Kafka'ya mesaj göndermek için bir KafkaProducer nesnesi oluşturulur.

#### 4. Mesajların Kafka'ya Gönderilmesi

```

14 try:
15     for index, row in data.iterrows():
16         producer.send("financialData", row["kafka_message"].encode())
17         print(f"Sent message to financialData: {row['kafka_message']}")
18     except Exception as e:
19         print(f"Error sending message: {e}")

```

**Amaç:** financialData adlı Kafka konusuna (topic) her satırdaki mesaj gönderilir. Gönderim sırasında herhangi bir hata olursa bu, Exception yakalanarak ekrana yazdırılır.

#### Sonuç

Bu kod, finansal verilerin gerçek zamanlı işlenmesi için Kafka kullanarak bir mesajlaşma sistemi oluşturur. Ön işlenmiş veriler Kafka konusuna (financialData) mesaj olarak gönderilir. Bu, büyük ölçekli veri akışlarında sıkça kullanılan bir yaklaşım olup, mesajların asenkron olarak gönderilmesini sağlar.

### IX. Spark Entegrasyonu

Bu kısım, Spark ve Kafka kullanarak dolandırıcılık tespiti gerçekleştiren bir sistem oluşturur. Sistem, ön işlenmiş veriler üzerinde bir rastgele orman (Random Forest) modeli eğitir, model performansını değerlendirir ve ardından Kafka'dan gerçek zamanlı veri akışıyla gelen mesajları işleyerek dolandırıcılık tahmini yapar.

#### 1. SparkSession Oluşturma

```

8 spark = SparkSession\
9     .builder\
10    .appName("FraudDetection")\
11    .config("spark.jars.packages", "org.apache.spark:spark-sql-kafka-0-10_2.12:3.5.4")\
12    .getOrCreate()
13

```



**Amaç:** Spark oturumunu başlatarak Spark SQL ve Kafka entegrasyonunu yapılandırır.

## 2. Veri Setinin Yüklenmesi ve Hazırlanması

**Amaç:** CSV formatındaki veri yüklenir, gereksiz sütunlar (Timestamp) kaldırılır ve hedef sütun (Target) etiketlenir.

### Özelliklerin Vektörleştirilmesi

```
20 assembler = VectorAssembler(inputCols=feature_columns, outputCol="fea
21 data = assembler.transform(data).withColumnRenamed("Target", "label")
22
23 (train_data, test_data) = data.randomSplit([0.8, 0.2], seed=42)
```

**Amaç:** Model eğitimi için özellik sütunları (features) birleştirilir ve veri seti eğitim ve test verileri olarak ayrılır.

## 3. Random Forest Modeli Eğitimi ve Değerlendirme

```
25 rf = RandomForestClassifier(numTrees=100)
26 model = rf.fit(train_data)
27
28 # Make predictions and evaluate the model
29 predictions = model.transform(test_data)
30 roc_auc = BinaryClassificationEvaluator().evaluate(predictions)
31
```

**Amaç:** Rastgele orman sınıflandırıcısı ile model eğitilir ve test verilerindeki tahminler yapılır. Modelin ROC AUC değeri hesaplanarak performansı değerlendirilir.

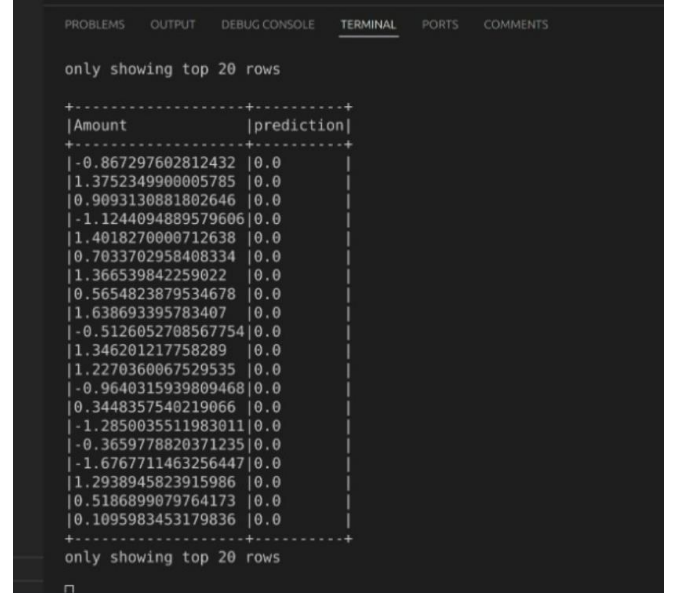
## 4. Kafka'dan Gerçek Zamanlı Veri Akışı ve Tahminler

```
87 streaming_df = (
88     spark.readStream.format("kafka")
89     .option("kafka.bootstrap.servers", "localhost:9092")
90     .option("subscribe", "financialData")
91     .load()
92 )
93
94 # Parse the JSON messages
95 parsed_df = (
96     streaming_df.selectExpr("CAST(value AS STRING)")
97     .select(from_json(col("value"), schema.alias("data")))
98     .select("data.*")
99 )
100
101 # Process the data (e.g., predictions)
102 def process_batch(batch_df, epoch_id):
103     # Assemble features
104     features_df = assembler.transform(batch_df).withColumnRenamed("Target", "label")
105
106     # Predict using the loaded model
107     predictions = model.transform(features_df)
108     predictions.select("Amount", "prediction").show(truncate=False)
109
110
111 # Start the streaming query
112 query = parsed_df.writeStream.foreachBatch(process_batch).start()
113 query.awaitTermination()
```

**Amaç:** Kafka'dan financialData konusundan mesajlar alınır. Mesajlar JSON formatında ayrıştırılır ve sütunlara dönüştürülür. Her veri grubu (batch) için özellikler dönüştürülür, model ile tahmin yapılır ve sonuçlar gösterilir. Veri akışı

sürekli işlenir ve tahminler gerçek zamanlı olarak yapılır.

## Yapılan İşlemlerin Terminal Ekranındaki Görüntüsü



## X. Kaynakça

L. with Nas, "Data Preprocessing Steps for Machine Learning in Python (Part 1)," Women in Technology, Oct. 29, 2023. <https://medium.com/womenintechology/data-preprocessing-steps-for-machine-learning-in-python-part-1-18009c6f1153>

B. Priya C, "How to Detect Outliers in Machine Learning – 4 Methods for Outlier Detection," freeCodeCamp.org, Jul. 05, 2022. <https://www.freecodecamp.org/news/how-to-detect-outliers-in-machine-learning/>

A. Shafi, "Sklearn Random Forest Classifiers in Python Tutorial," www.datacamp.com, Feb. 2023. <https://www.datacamp.com/tutorial/random-forests-classifier-python>

