İstanbul Medeniyet Üniversitesi
Bilgisayar Mühendisliği Bölümü

Örüntü Tanıma Projesi

Grup No: 1

İsim 1: Ayşenur Yörür

İsim 2: Kaan Berk Duman

Veri Kümeler : C2 ve R1

k-Fold : 2,5

Algoritmalar : Random Forest ve Xgboost (Sınıflandırma); Ridge Regression ve LSTM (regresyon)

Teslim Tarihi : 11/01/2025

Ocak 2025

1. GİRİŞ

1.1 Veri Kümelerinin Tanıtımı

1.1.1. Classification

Colab Linkleri:

https://colab.research.google.com/drive/1JxFkU5GYWvkpEBktrDLfhucxGCqUCFXM?usp=sharing

https://colab.research.google.com/drive/1z2iW3Br4HUsmGiYAXT227NQM81Q56_rZ?usp=sharing

Veri setimizin adı Dry Bean Dataset'tir. Veri setimizde 7 farklı kayıtlı bean türüne ait 13.611 tane tanelerin görüntüleri yüksek çözünürlüklü bir kamera ile çekilmiştir. Toplamda 16 özellik, 12 boyut ve 4 şekil formu bu tanelerden elde edilmiştir.

Veri Türü: Çok Değişkenli

Öznitelik Türü: Categorical Integer Real

Biçim Türü: Matris

Veri kümeniz eksik değerler içeriyor mu? Hayır

Örnek Sayısı (veri kümenizdeki kayıtlar): 13611

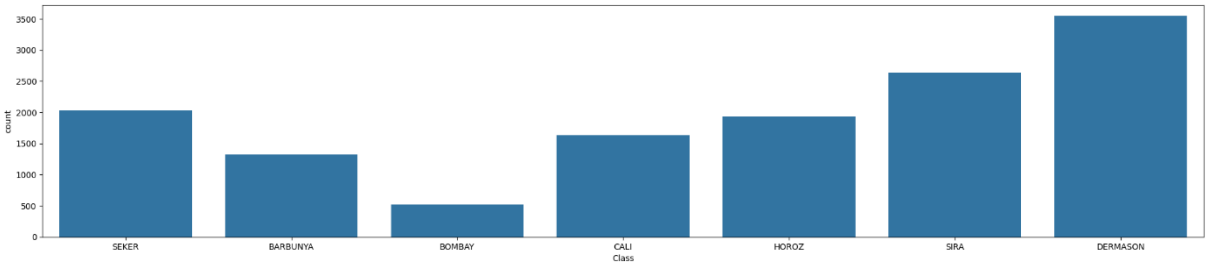
Öznitelik Sayısı (her kayıttaki alanlar): 16

Bu araştırmada, piyasa koşulları göz önünde bulundurularak form, şekil, tür ve yapı gibi özellikler dikkate alınarak yedi farklı bean türü kullanılmıştır. Benzer özelliklere sahip yedi farklı kayıtlı bean türünü ayırt etmek ve homojen tohum sınıflandırması elde etmek amacıyla bir bilgisayarlı görü sistemi geliştirilmiştir. Sınıflandırma modeli için, yedi farklı bean türüne ait 13.611 tane tanelerin görüntüleri yüksek çözünürlüklü bir kamera ile çekilmiştir. Bilgisayarlı görü sistemi ile elde edilen bean görüntüleri segmentasyon ve özellik çıkarma aşamalarına tabi tutulmuş ve toplamda 16 özellik; 12 boyut ve 4 şekil formu tanelerden elde edilmiştir.

Attribute Information:

1. Area (A): Alan Bir fasulye bölgesinin alanı ve sınırları içindeki piksel sayısı.
2. Perimeter (P): Çevre Fasulyenin çevresi, sınırının uzunluğu olarak tanımlanır.
3. Major axis length (L): Büyük eksen uzunluğu (L): Bir fasulyeden çizilebilecek en uzun çizginin uçları arasındaki mesafe.
4. Minor axis length (l): Küçük eksen uzunluğu (l): Ana eksene dik dururken fasulyeden çizilebilecek en uzun çizgi.
5. Aspect ratio (K): En-boy oranı (K): L ve l arasındaki ilişkiyi tanımlar.
6. Eccentricity (Ec): Eksantriklik (Ec): Aynı momentlere sahip elipsin eksantrikliği.

7. Convex area (C): Konveks alan (C): Fasulye tohumunun alanını içerebilecek en küçük konveks çokgen içindeki piksel sayısı.
8. Equivalent diameter (Ed): Eşdeğer çap (Ed): Alanı bir fasulye tohumu alanı ile aynı olan bir çemberin çapı.
9. Extent (Ex): Kapsam (Ex): Sınır kutusundaki piksellerin fasulye alanına oranı.
10. Solidity (S): Yoğunluk (S): Konvekslik olarak da bilinir. Konveks kabukta bulunan piksellerin fasulyelerde bulunanlara oranı.
11. Roundness (R): Roundness (R): Yuvarlaklık (R): Aşağıdaki formülle hesaplanır: $(4\pi A) / (P^2)$.
12. Compactness (CO): Kompaktlık (CO): Bir nesnenin yuvarlaklığını ölçer: Ed / L.
13. ShapeFactor1 (SF1) Şekil Faktörü 1 (SF1)
14. ShapeFactor2 Şekil Faktörü 2 (SF2)
15. ShapeFactor3 Şekil Faktörü 3 (SF3)
16. ShapeFactor4 Şekil Faktörü 4 (SF4)
17. Class Sınıf: (Şeker, Barbunya, Bombay, Çalı, Dermosan, Horoz ve Sıra).



Şekil 1

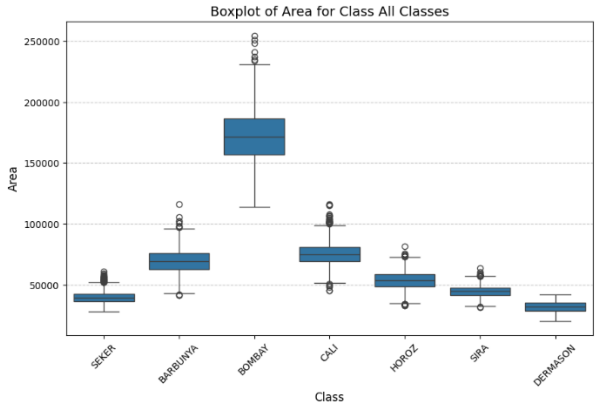
Veri setimiz 7 sınıflı (multi-class) ve dengesiz bir yapıya sahiptir. Özellikle **Bombay** sınıfına ait verilerin oldukça az olduğu görülmektedir. Bu durum, modeli eğitirken sınıflar arasında dengesizlik kaynaklı performans farklılıklarına yol açabilir.

Ayrıca, veri setinde toplam **68 adet duplicate (tekrarlı)** veri tespit edilmiştir. Bu, aynı özelliklere sahip 68 satırın veri setinde yer aldığı anlamına gelir. Bu durum, eğitim sırasında modelin aynı bilgiyi birden fazla kez öğrenmesine neden olabileceği için potansiyel bir sorun teşkil edebilir.

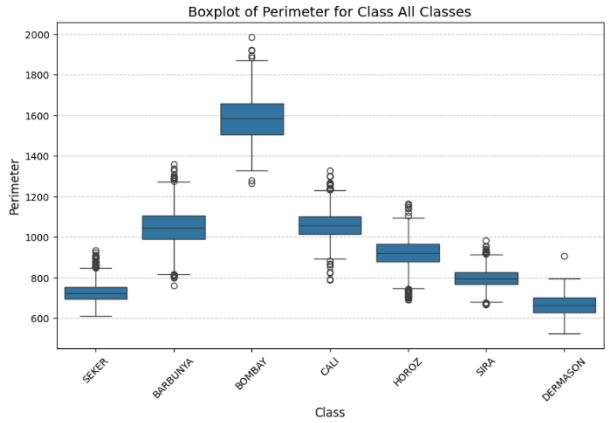
Veri setindeki özellikler (feature'lar) incelendiğinde, aralarında oldukça yüksek bir korelasyon olduğu görülmektedir. Özellikle birbirinden bağımsız olması beklenen bazı özelliklerin dahi yüksek korelasyon göstermesi, **multicollinearity (çoklu doğrusal bağıntı)** problemine işaret etmektedir.

Boxplot analizleri incelendiğinde, **Bombay bean** sınıfının bazı özellikler (features) açısından diğer sınıflardan belirgin şekilde ayrıldığı gözlemlenmiştir. Özellikle **Area, Perimeter, MajorAxisLength** ve **MinorAxisLength** gibi özellikler üzerinde yapılan incelemelerde, Bombay bean'in boxplotlarının diğer sınıflardan oldukça farklı bir konumda olduğu açıkça görülmektedir.

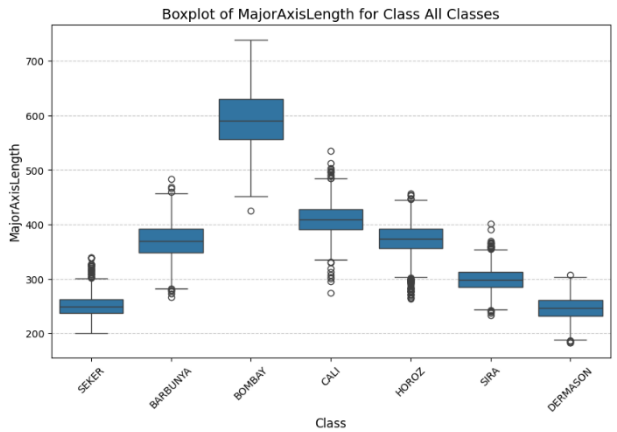
Bu durum, Bombay bean sınıfının veri setinde diğer sınıflardan ayrılabilmesini kolaylaştırabilir. Dolayısıyla, sınıf dengesizliğine rağmen modelin Bombay bean'i doğru bir şekilde sınıflandırmakta zorluk çekmeyeceği tahmin edilmektedir. Ancak, bu sınıfın diğer sınıflardan bu kadar farklı olması, model performansını artırıcı bir faktör olarak değerlendirilebilir.



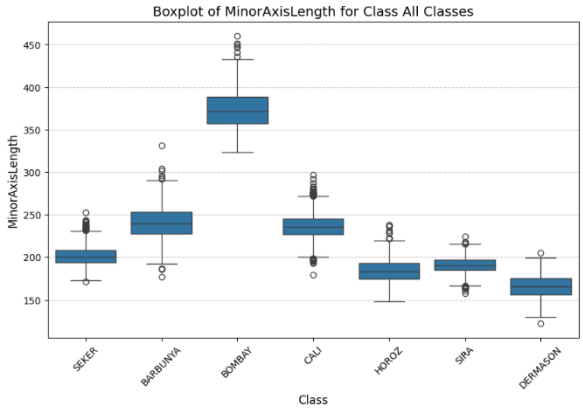
Şekil 2



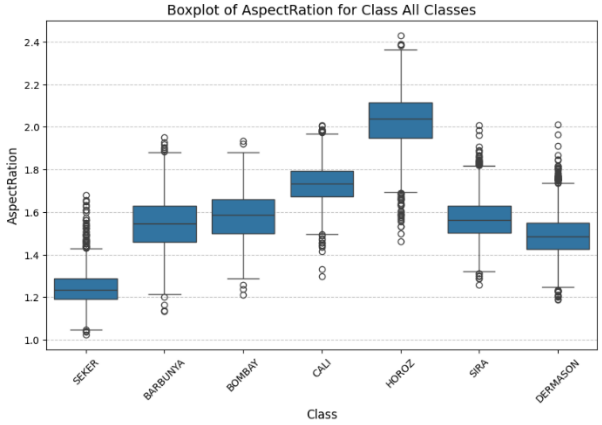
Şekil 3



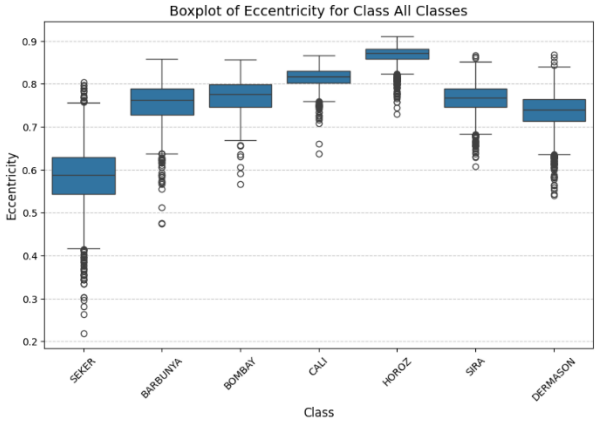
Şekil 4



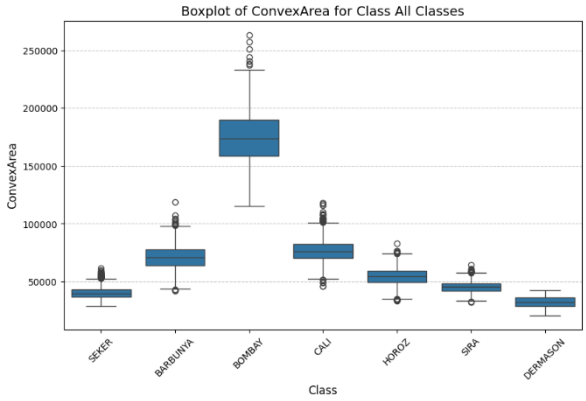
Şekil 5



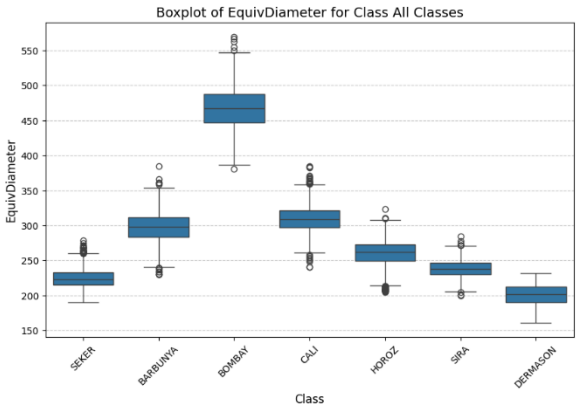
Şekil 6



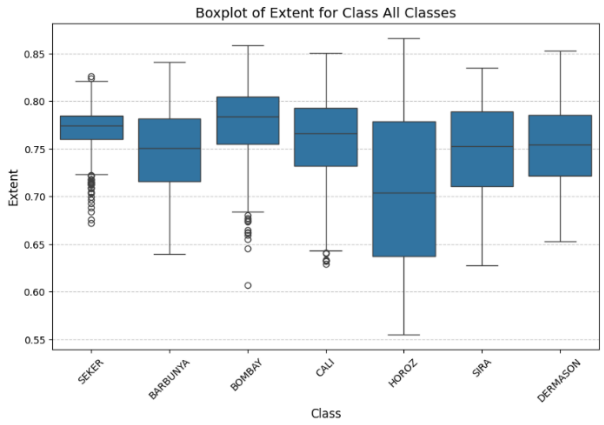
Şekil 7



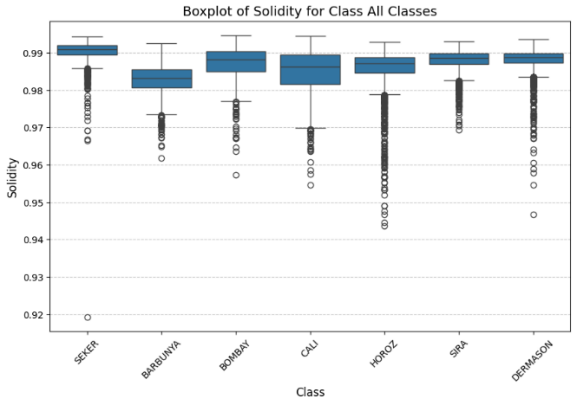
Şekil 8



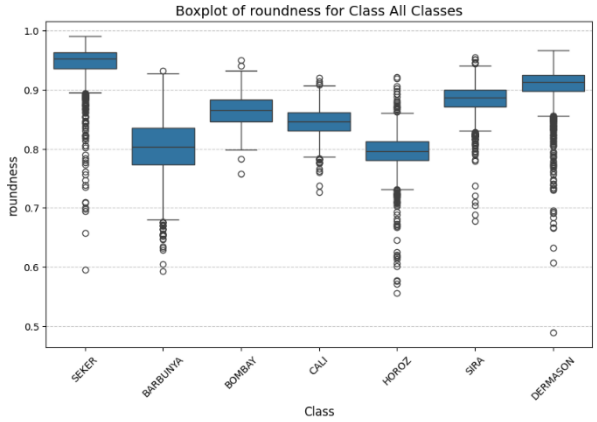
Şekil 9



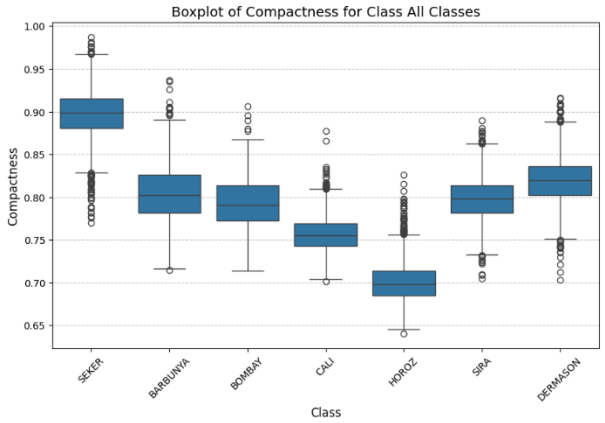
Şekil 10



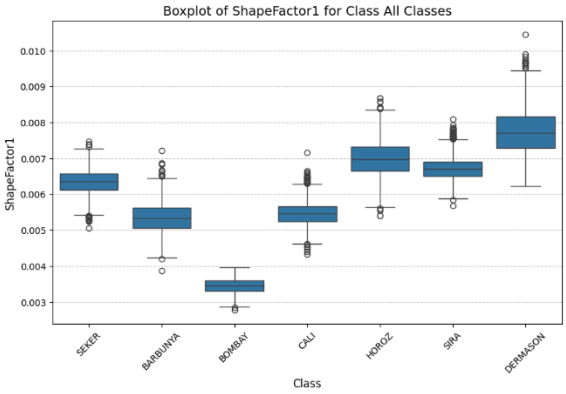
Şekil 11



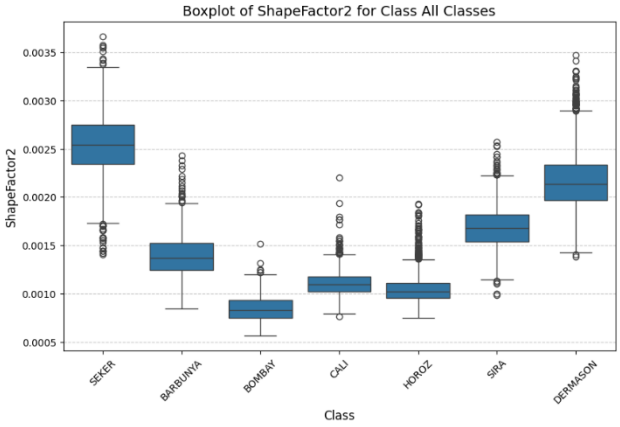
Şekil 12



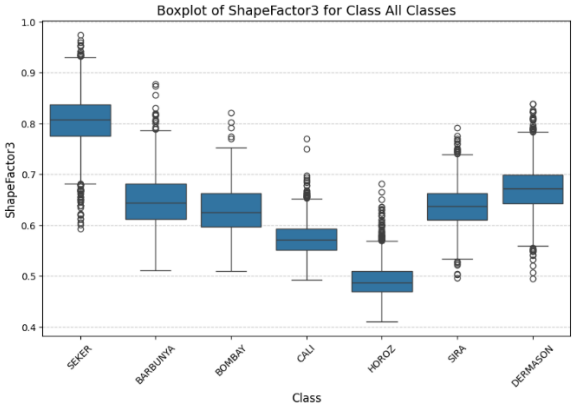
Şekil 13



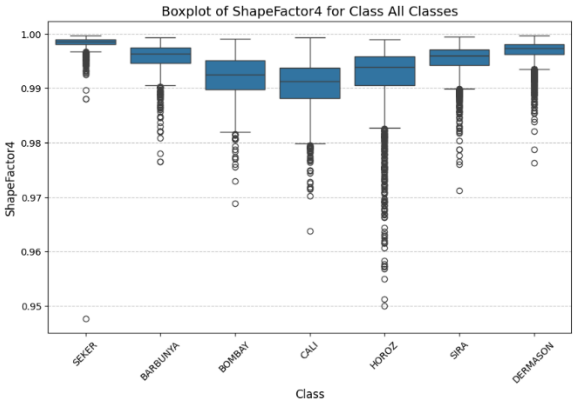
Şekil 14



Şekil 15



Şekil 16



Şekil 17

1.1.2. Regresyon

Cezayir Orman Yangınları Veri Seti: Bu veri seti, Cezayir'in iki bölgesinden (kuzeydoğuda bulunan Bejaia bölgesi ve kuzeybatıda bulunan Sidi Bel-abbes bölgesi) alınan 244 gözlemi içermektedir. Her bir bölge için 122 adet gözlem bulunmaktadır.

Veri seti, Haziran 2012 ile Eylül 2012 tarihleri arasındaki dönemi kapsamaktadır. Toplamda 11 bağımsız değişken ve 1 bağımlı değişken (sınıf) içermektedir. Gözlemler, iki sınıfa ayrılmıştır: yangın olan (138 gözlem) ve yangın olmayan (106 gözlem).

Veri Seti Sütunları:

- Tarih: (GG/AA/YYYY) Gün, ay (Haziran'dan Eylül'e kadar) ve yıl (2012). Hava durumu gözlemlerini içerir.
- Sıcaklık (Temp): Öğle saatindeki maksimum sıcaklık (Celsius derece): 22 ile 42 arasında.
- Bağıl Nem (RH): % olarak bağıl nem: 21 ile 90 arasında.
- Rüzgar Hızı (Ws): Rüzgar hızı (km/saat): 6 ile 29 arasında.
- Yağış (Rain): Günlük toplam yağış miktarı (mm): 0 ile 16.8 arasında.
- İnce Yakıt Nem İndeksi (FFMC): FWI (Yangın Hava İndeksi) sistemine ait bir bileşen: 28.6 ile 92.5 arasında.
- Karışık Yakıt Nem İndeksi (DMC): FWI sistemine ait bir bileşen: 1.1 ile 65.9 arasında.
- Kuraklık Kodu (DC): FWI sistemine ait bir bileşen: 7 ile 220.4 arasında.
- Başlangıç Yayılma İndeksi (ISI): FWI sistemine ait bir bileşen: 0 ile 18.5 arasında.
- Birikim İndeksi (BUI): FWI sistemine ait bir bileşen: 1.1 ile 68 arasında.
- Yangın Hava İndeksi (FWI): FWI sistemi genel indeksi: 0 ile 31.1 arasında.
- Sınıflar (Classes): İki sınıf bulunmaktadır: "Yangın" (Fire) ve "Yangın Yok" (Not Fire).

Bu veri seti, yangın tahmini yapmak (Classes) için hava durumu ve yangın hava indeksi (FWI) bileşenlerini analiz etmeyi kullanılabilmektedir. Bizim amacımız labelımız olan FWI hava indeksini tahmin etmek olacaktır.

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
122	Sidi-Bel Abbes Region Dataset	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
167	14	07	2012	37	37	18	0.2	88.9	12.9	14.6 9	12.5	10.4	fire	NaN

Tablo 1

Bu işlemi, veri setini iki farklı bölgeye ("Bejaia Bölgesi" ve "Sidi-Bel Abbes Bölgesi") ayırabilmek için yaptık. Veri setindeki 122. indexten itibaren kayıtların bölgelere göre farklılık gösterdiğini bildiğimizden, her bir bölgeye ait kayıtları belirlemek amacıyla yeni bir "Region" sütunu ekledik. Bu sütun, "Bejaia Region Dataset" için 0, "Sidi-Bel Abbes Region Dataset" için 1 olarak atanmıştır. Bu sayede, analiz ve modelleme sırasında bölgeler arası ayrım yapılması kolaylaştırılmıştır.

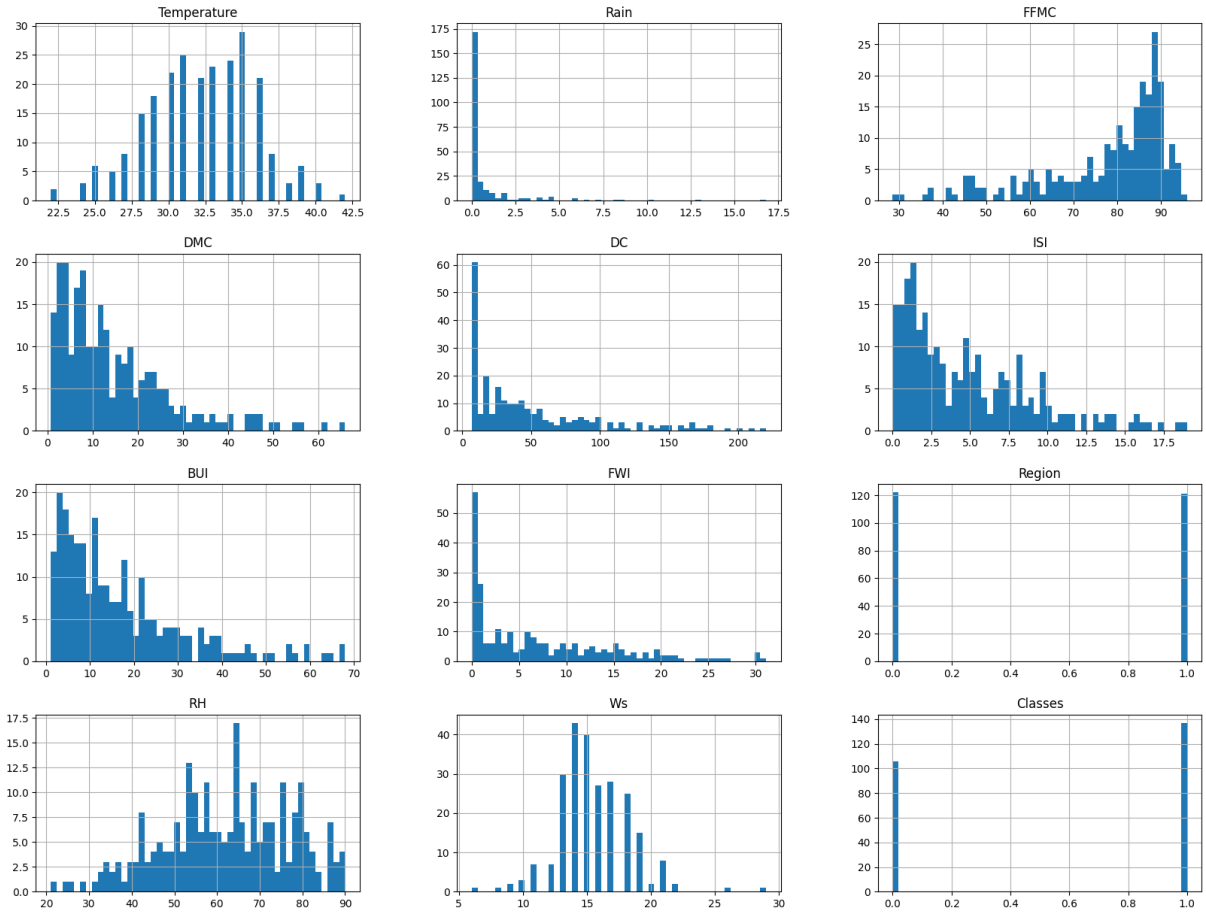
	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Region
0	01	06	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire	0
1	02	06	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire	0
2	03	06	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire	0
3	04	06	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire	0
4	05	06	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire	0

Tablo 2

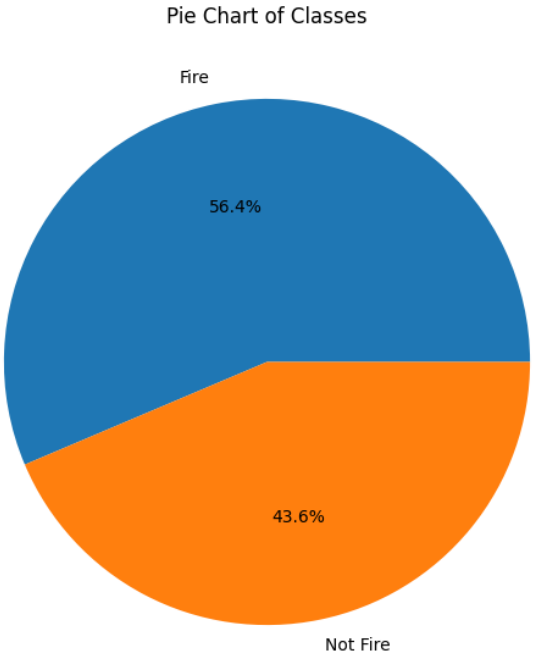
Datada 122. satırda bir object türünde nesneler olduğu için veri türleri obje olarak gözükmektedir. Aşağıda görebileceğiniz üzere dolayısı ile biz 122. satırı da kaldıracağız. Datasetimizde null ve olması gerekenden farklı türde veriler olmadığında hata almadan astype fonksiyonunu kullanarak türlerini değiştirebiliriz. Sütun isimlerinde gereksiz boşluklar da var, bu isimleri düzelteceğiz.

- Örneğin " RH" -> "RH" olmalı.

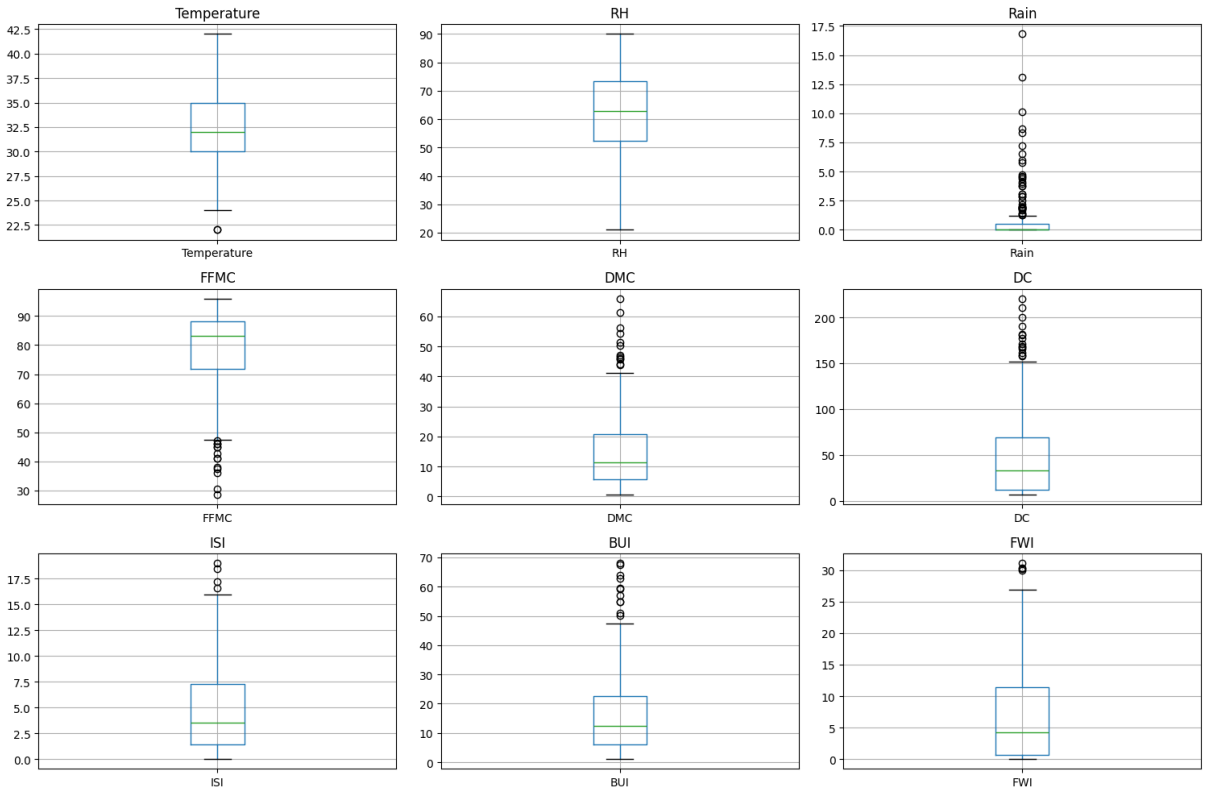
Preprocess edip temizlenen düzenlenen verimizi github'a yükledikten sonra github üzerinden tekrar çektik.



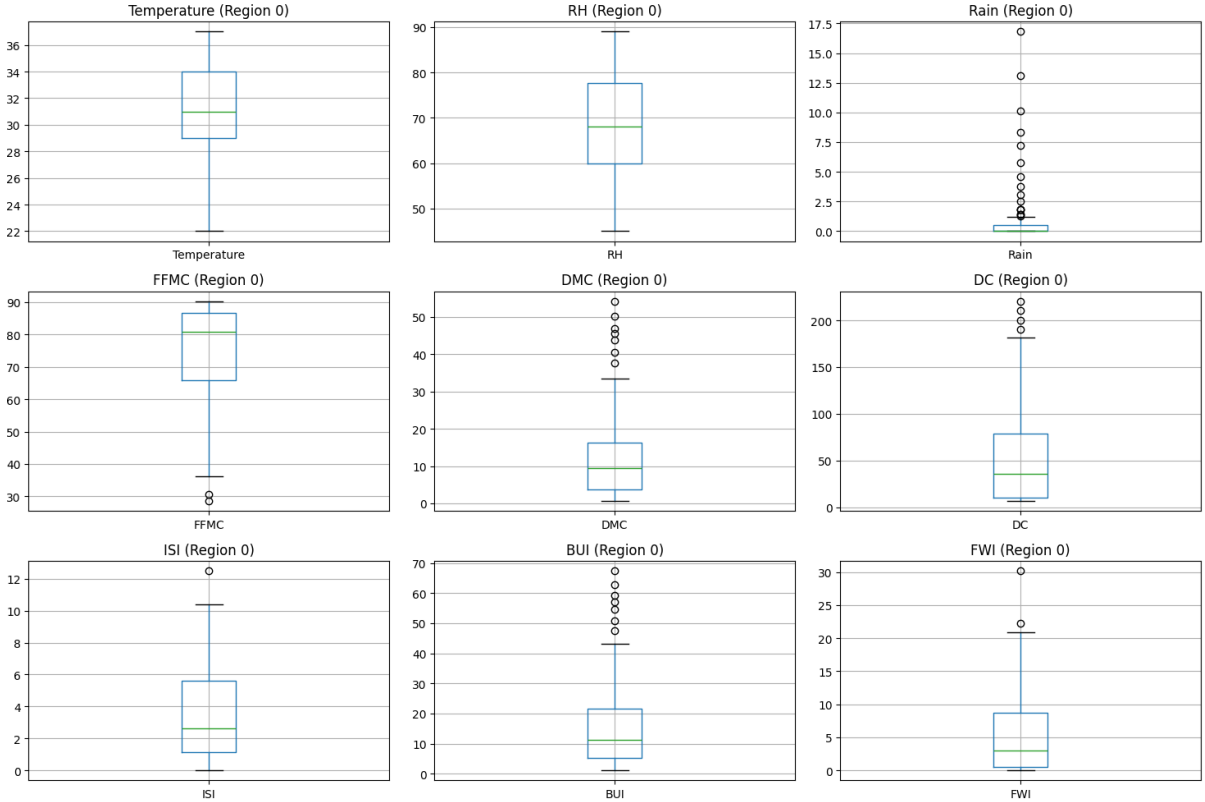
Şekil 18



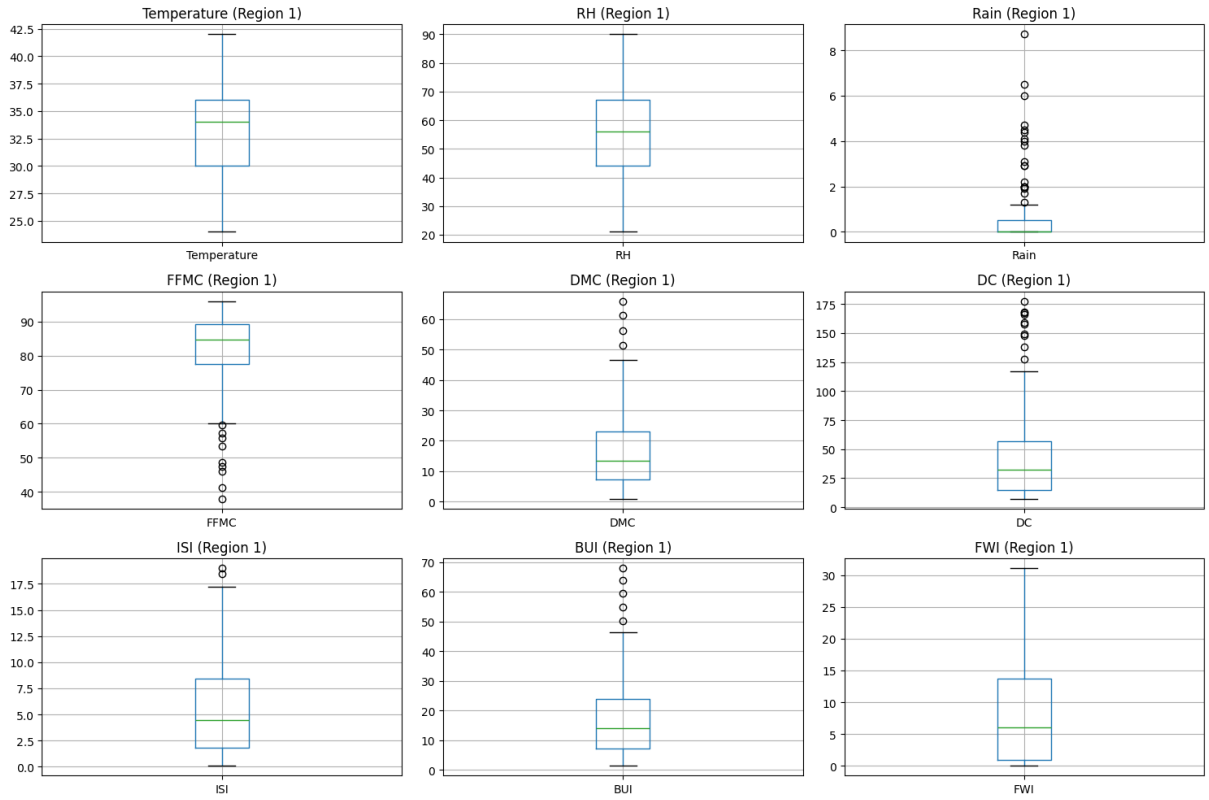
Şekil 19



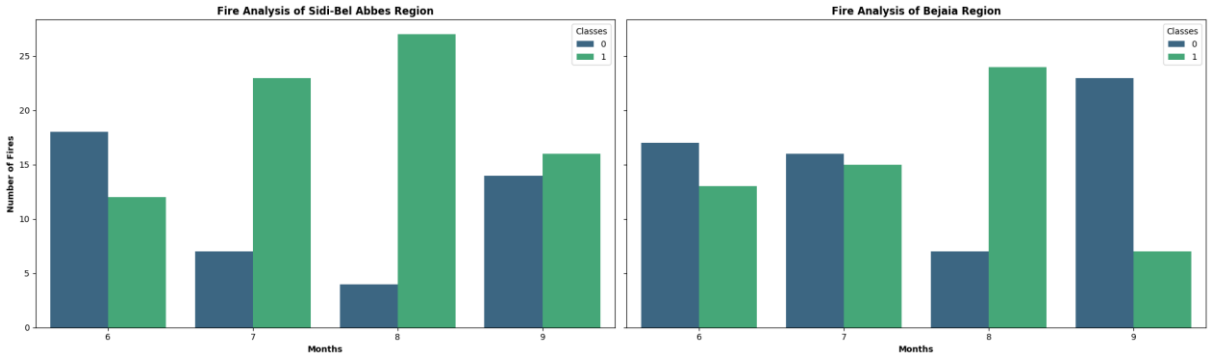
Şekil 20



Şekil 21

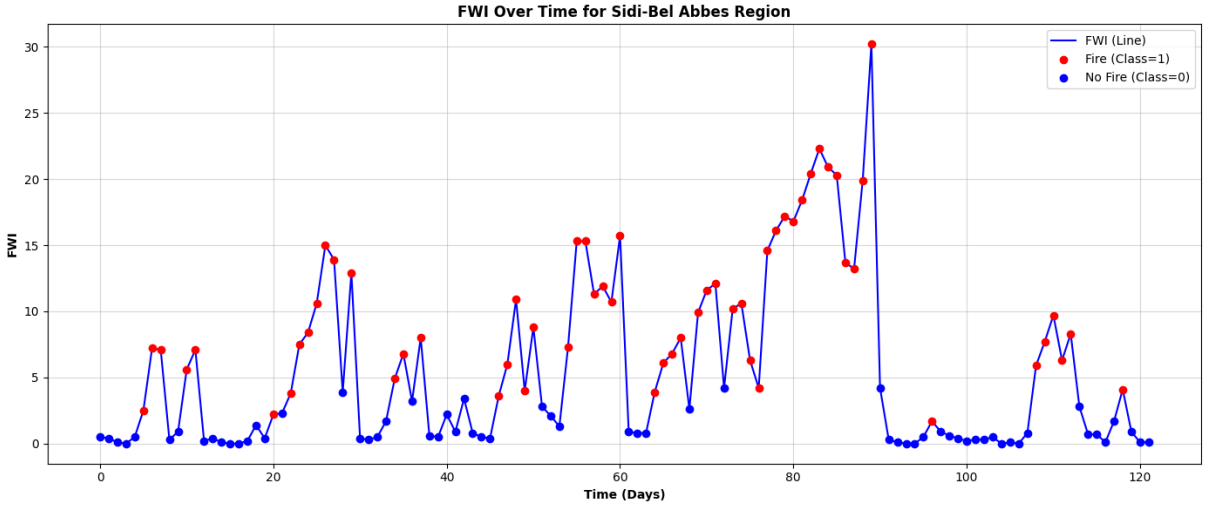


Şekil 22

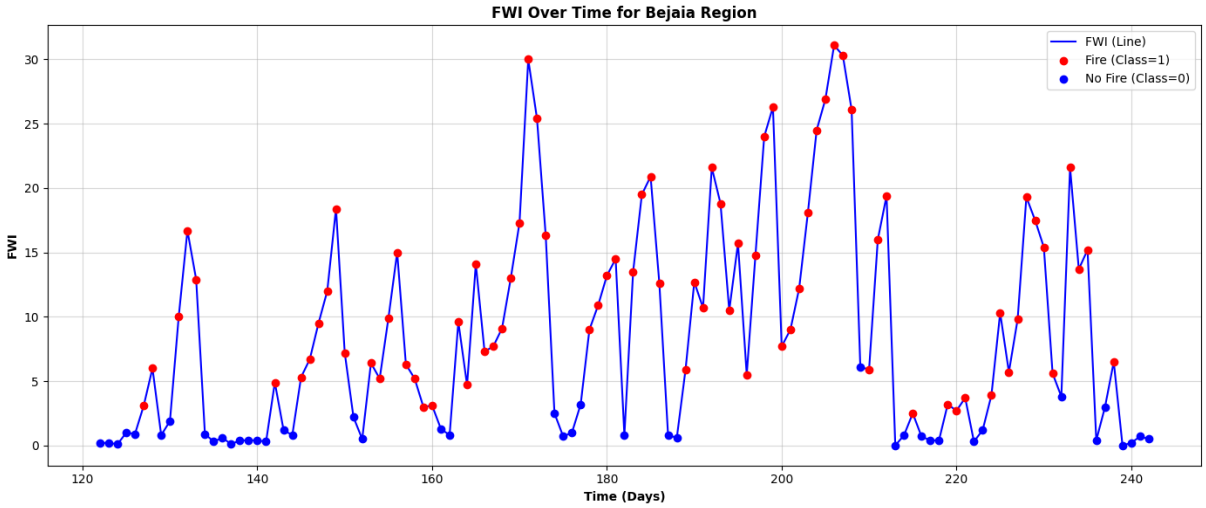


Şekil 23

Bölgesel olarak farklılıklar görebiliyoruz. Özellikle 7. ve 9. ayda farklılıklar oldukça belirgin durumdadır. 6. ayda çok benzer bir grafik izlemiştir.



Şekil 24

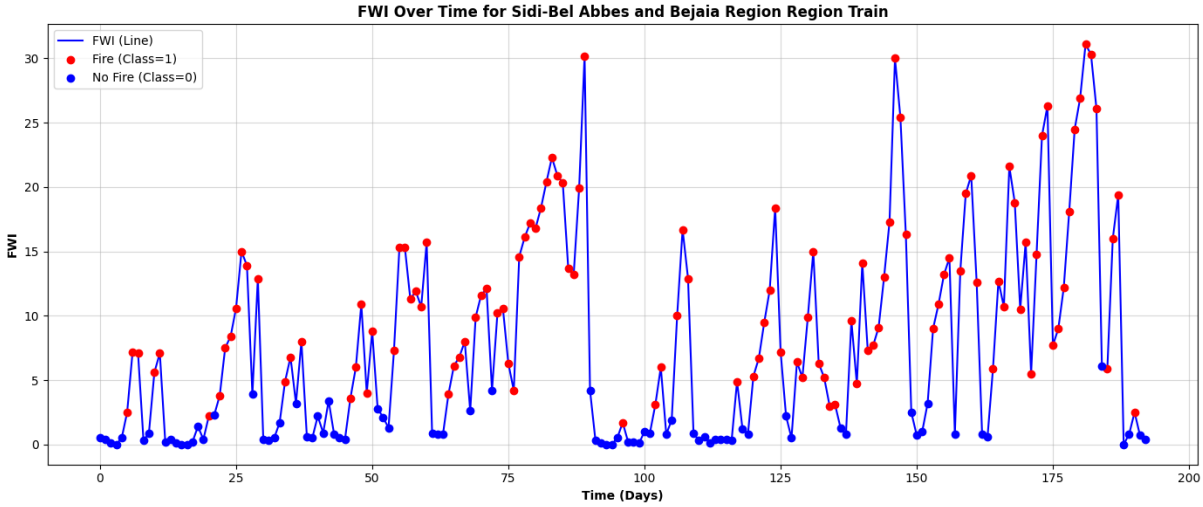


Şekil 25

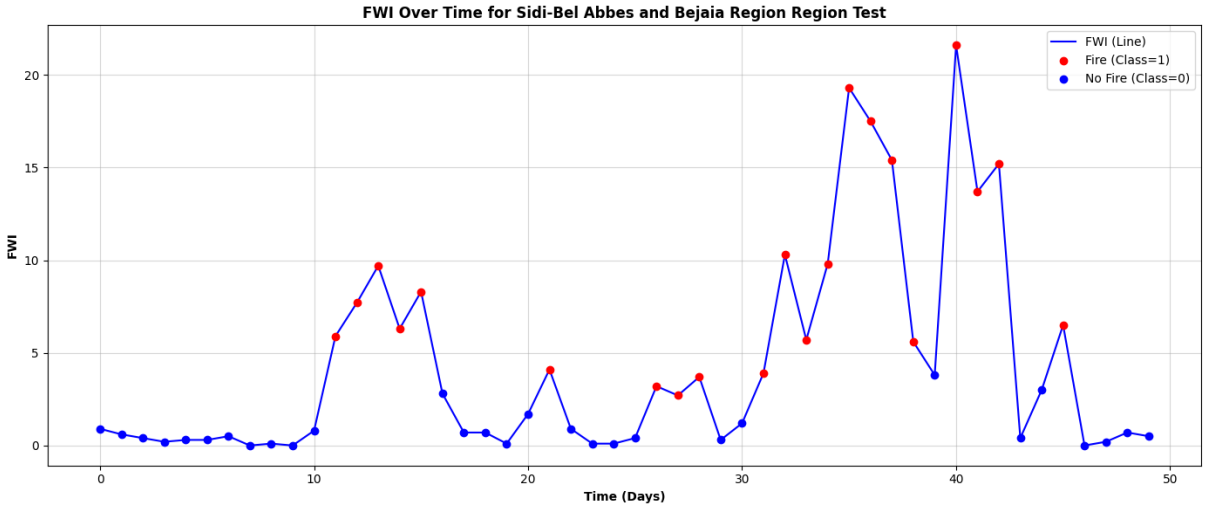
Bu grafik aslında bize ilginç bir fikir verdi. O gün içerisinde yangın olması bir önceki gün yangın yoksa genel olarak o günün FWI değerini oldukça artırmıştır. Hatta bir önceki değerler bir sonraki değerler ile bir örüntü içerisinde gözüküyor şeklinde de düşünebiliriz. Bu amaçla çerçeveleme yöntemi kullanılabilir ilerleyen aşamalarda.

PreProcess: Shuffle = False yapıyoruz böylece ilk %80 lik kısım ile son %20 lik kısmı ayırmış oluyoruz bu şekilde sıra bilgisini de kaybetmemiş oluyoruz. Daha sonra iki bölgeyi de böldükten sonra train setlerini uc uca birleştiriyoruz ve test setlerini de sıraya dikkat ederek uc uca birleştiriyoruz böylece train ve set datalarımı hazırlamış olduk.

Bir sonraki adımlarda scale ve PCA işlemlerini yapacağız data leakage olmasını istemediğiz için. Yani test setim hakkında herhangi bir bilgiye train datam sahip olsun istemediğiz için önce bölme işlemini yaptık.



Şekil 26



Şekil 27

Windowing: Zaman serisi verilerinde, ardışık veri noktalarının birbirleriyle ilişkili olması, modelin bu bağımlılıkları öğrenmesini önemli kılmaktadır. Bu nedenle, veri setine **windowing (kaydırmalı pencere)** yöntemi uyguladım. Bu teknik, veriyi belirli bir pencere boyutunda kesitlere ayırarak, her bir pencereyi ilgili bir hedef değerle eşleştirme imkanı sunar. Böylece model, zaman serisinin dinamik yapısını daha iyi öğrenebilir hale gelmektedir.

Pencere Boyutu (window_size): Her bir pencerenin içerdiği zaman adımı sayısını belirlemek için kullanılmıştır. Bu çalışmada, geçmiş 3 zaman adımını dikkate almak amacıyla window_size=3 olarak belirledim.

Adım Boyutu (step_size): Pencereler arasındaki kaydırma mesafesini ifade eder. Her bir zaman adımı için pencere oluşturmak amacıyla step_size=1 kullanılmıştır.

Bu süreç, zaman serisi verilerinin doğasını koruyarak, hem eğitim hem de test aşamalarında modelin performansını optimize etmek için önemli bir adımdır diye düşünmekteyim.

1.2 Kullanılan Algoritmaların Özellikleri

1.2.1. Classification

Test verisinin temel amacı, modelin **gerçek dünyadaki performansını** doğru bir şekilde değerlendirmektir. Gerçek hayatta veri genellikle dengesiz olduğu için, test verisini olduğu gibi bırakmak, modelin dengesiz veri üzerinde nasıl bir performans gösterdiğini anlamak açısından kritik öneme sahiptir.

Ancak, bu veri setiyle yapılan diğer çalışmaları incelediğimde, test verisine yapılan manipülasyonların model performansı üzerinde yanıltıcı etkiler oluşturduğunu gözlemledim. Veri setinin tamamına **SMOTE**, **PCA** ve **outlier removal** gibi işlemlerin uygulandığı ve bu süreçlerin model başarılarını yapay olarak artırdığı görülmektedir.

Örneğin, *Automated Classification of Dry Bean Varieties Using XGBoost and SVM Models* adlı çalışmada, test verisinin manipüle edilmesiyle yüksek başarı elde edilmiştir. Ancak, bu işlemleri projede denediğimizde, test verisini manipüle etmeden aynı başarıyı tekrarlayamadık. Bu durum, modelin gerçek ayırt edebilme kapasitesinin yanlış değerlendirilmesine neden olabilir.

SMOTE (Synthetic Minority Oversampling Technique), azınlık sınıflarını dengelemek için sentetik veri noktaları oluşturan bir yöntemdir. Ancak SMOTE, bizim problemimiz için uygun bir yöntem değildir. Örneğin, iki veri noktası olan A ve B alınarak (A+B)/2 şeklinde bir sentetik veri noktası oluşturulabilir. Bu işlem, A ve B noktalarını yaklaşık olarak 1.5 ağırlıklandırmaya eşdeğer bir sonuç doğurur. Ancak, bu yöntem her zaman tutarlı bir performans artışı sağlamaz. Özellikle *pairplot* analizleri incelendiğinde, SMOTE'un veri setimize uygun olmadığı açıkça görülmektedir. Bu durumun tartışıldığı bir StackExchange konu başlığı de mevcuttur.

Buna karşılık, **ADASYN (Adaptive Synthetic Sampling)**, azınlık sınıfındaki daha zor öğrenilen ve çoğunluk sınıfına yakın olan veri noktalarına ağırlık vererek bu bölgelerde daha fazla sentetik veri üretir. Bu yönüyle ADASYN, SMOTE'a kıyasla veri setimiz için daha uygun bir yöntem gibi görünmektedir. Ancak, bu yöntemin de yalnızca eğitim verisine uygulanması gerektiği, test verisinin manipüle edilmemesi gerektiği unutulmamalıdır.

Comparison of multiclass classification techniques using dry bean dataset adlı makalede, ADASYN kullanımının etkisi açıkça görülmektedir. Tablolar incelendiğinde, ADASYN kullanılmadan önce **Dermason** ve **Sira** türlerinin sıklıkla karıştırıldığı, ancak **Bombay** türünün ayırt edilmesinde herhangi bir sorun yaşanmadığı gözlemlenmiştir. Bombay fasulyesi, özellikleri bakımından diğer türlerden oldukça farklı olduğu için model tarafından zaten kolaylıkla ayırt edilebilmektedir. Bu nedenle, Bombay fasulyesi için oversampling yapmak veya weighted yöntemler kullanmak mantıklı bir seçenek olabilir.

Table 5
(a). Confusion matrix for XGBoost classifier with and without ADASYN algorithm.

without ADASYN							
Actual	Predict Seker	Barbunya	Bombay	Cali	Horoz	Sira	Dermason
Seker	**454**	6	0	0	0	14	13
Barbunya	0	**316**	0	22	2	2	1
Bombay	0	0	**116**	0	0	1	0
Cali	1	6	0	**399**	8	5	0
Horoz	0	3	0	11	**466**	8	7
Sira	8	2	0	2	4	**558**	62
Dermason	13	0	0	0	0	54	**839**

with ADASYN							
Actual	Predict Seker	Barbunya	Bombay	Cali	Horoz	Sira	Dermason
Seker	**678**	1	0	1	0	19	5
Barbunya	5	**660**	0	29	4	8	0
Bombay	0	0	**710**	0	0	0	0
Cali	5	8	0	**702**	8	3	0
Horoz	0	2	0	6	**670**	14	5
Sira	16	3	0	4	22	**592**	44
Dermason	17	0	0	0	4	63	**625**

Tablo 3

Tabloya baktığımızda ADASYN kullanıldığında, Bombay fasulyesi için veri noktaları 116'dan 710'a çıkartılmış ve bu durum accuracy değerinin artmasına neden olmuştur. Ancak, bu artış gerçek model performansını yansıtmamaktadır. Özellikle **Dermason** ve **Sira** türleri hala karıştırılmaya devam etmekte, diğer türlerde de hatalar gözlemlenmektedir. Yani accuracy artışı, yalnızca veri manipülasyonundan kaynaklanan bir **ilüzyon** oluşturmaktadır.

Table 6
Performance measures of classification models (%) on the dry bean dataset.

without ADASYN								
Classifiers	AUC	ACC	MS Error	F1-Score	FPR	Kappa	SE	SP
LR	99.35	91.0	9.0	91.0	34.39	89.14	99.65	98.70
KNN	96.76	89.0	11.0	89.0	34.01	86.92	99.68	98.93
DT	94.85	90.0	10.0	90.0	34.64	87.33	99.04	98.46
RF	99.40	92.0	8.0	92.0	34.44	90.18	99.68	98.91
SVM	99. 58	92.0	8.0	92.0	34.40	90.67	99.68	98.50
NB	99.18	89.0	11.0	89.0	34.17	86.94	99.62	98.50
XGB	**99.60**	**93.0**	**7.0**	**93.0**	**34.44**	**90.92**	**1.00**	**98.69**
MLP	99.43	91.0	9.0	91.0	34.50	89.47	99.66	98.47

with ADASYN								
Classifiers	AUC	ACC	MS Error	F1-Score	FPR	Kappa	SE	SP
LR	98.03	83.0	17.0	83.0	34.25	80.60	96.96	97.45
KNN	**98.52**	**95.0**	**5.0**	**95.0**	**33.62**	**92.77**	**99.85**	**99.50**
DT	94.38	90.0	10.0	90.0	33.85	88.83	98.91	98.93
RF	**99.60**	**94.0**	**6.0**	**94.0**	**33.68**	**92.62**	**99.10**	**99.70**
SVM	98. 67	86.0	14.0	86.0	34.08	84.03	99.84	98.10
NB	97.16	79.0	21.0	79.0	34.47	75.87	95.31	97.05
XGB	**99.64**	**95.4**	**6.9**	**94.0**	**33.66**	**93.10**	**99.92**	**99.85**
MLP	98.38	84.0	16.0	84.0	34.19	81.74	98.70	96.95

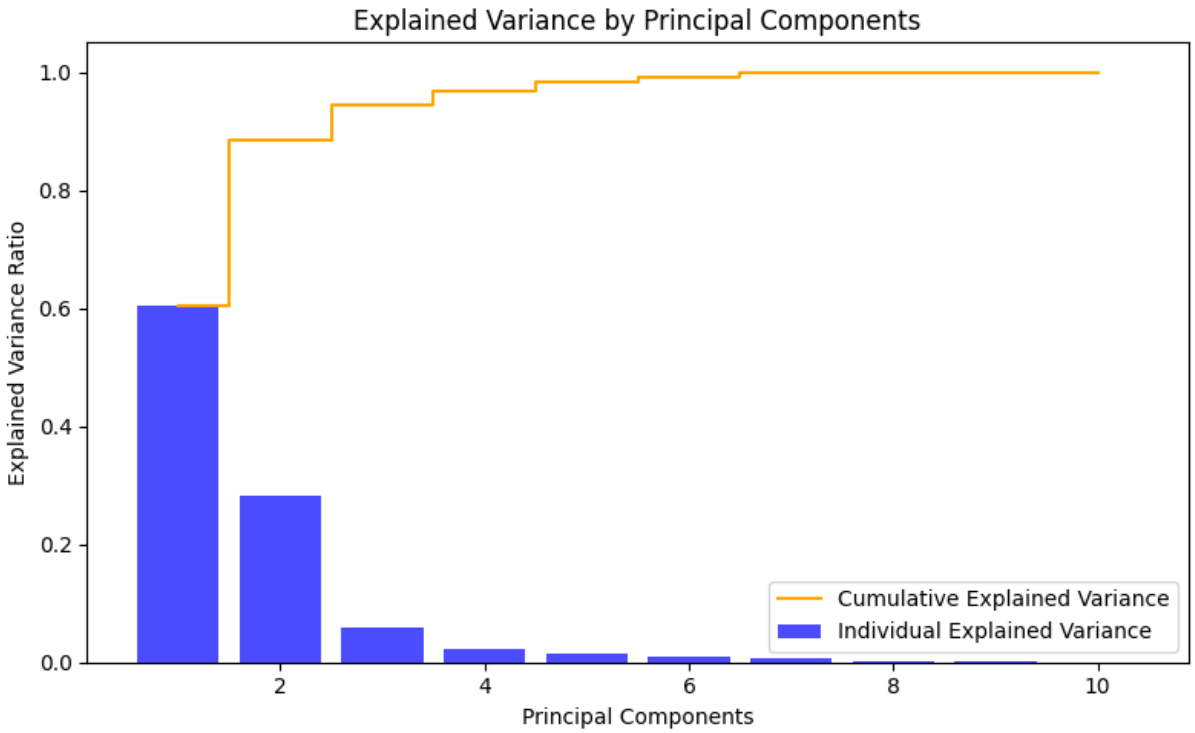
Tablo 4

En büyük problemlerden biri, test verisine de oversampling yapılması ve bu şekilde yanıltıcı bir başarı elde edilmesidir. Test verisinin manipüle edilmesi, modelin gerçek performansını değerlendirmeyi imkânsız hale getirir ve sonuçların doğruluğunu sorgulatır. Bu tür yöntemlerin projede test edilmesi planlanmakta, ancak test verisinin manipüle edilmediği durumlarda gerçek model başarısının daha iyi anlaşılabileceği vurgulanmaktadır.

Evaluation bölümü için X_test ve y_test dataları manipüle edilmeden testlerimizi gerçek dünyaya uygun yapacağız. Hiperparametrelerimizi X_train ve y_train üzerinde validation skorlarımızı ve hiperparametrelerimiz Kfold kullanarak yapacağız.

Model hakkında daha çok bilgi edinmek ve modeldeki varolan multicolinearity problemini azaltmak için PCA kullanacağız. Tanımlayıcı İstatistikler bölümününde verilerin pairplotlarında birbirleri ile feature'larımızın yüksek korelasyonlara sahip olduğunu gördük.

PCA modelini eğitirken (fit), yalnızca train veri setini kullanarak bileşenleri belirleriz. Daha sonra aynı model ile test setine sadece transform işlemi uygularız. Böylece her iki veri seti aynı PCA bileşenleri ile temsil edilmiş olur.



Şekil 28

Burada PC1 ve PC2 yi özniteliklerini kullanmak verimizin %88 olarak açıklanabilirliğini yansıtmaktadır. Yüksek boyuttaki verilerde bunları kullanarak bile yapılacak çalışmalar kullanılabilir. Lakin bizim problemimizde veri setimizin büyüklüğü sorun teşkil etmediği için ve **multicolinearity** den kurtulmak istediğimiz için %99.77 açıklanabilirlik ile ilk 7 Principal Component'i kullanacağız.

Validation Function: Veri setimizde data leakage'i önlemek ve herhangi bir sınıfın tek bir **veri setine toplanmasını engellemek** için **stratified=y** kullanıyoruz. Bu, her sınıfın oranını hem train hem de validation veri setinde koruyarak dengeli bir dağılım sağlıyor. Böylece model, tüm sınıfları adil bir şekilde öğrenme şansı elde ediyor.

Evaluation Function: Bu fonksiyon, bir makine öğrenimi modelinin test verisi üzerindeki performansını değerlendirmek için tasarlanmıştır ve sonuçları metriklerle birlikte görsel olarak sunar. Modelin gerçek dünya, genel performansını test etmek ve genelleme yeteneğini ölçmeyi amaçlar.

1.2.1.1. Model-1: Random Forest Base Model

Ağaç tabanlı modellerde, özelliklerin ölçeklendirilmesi genellikle gerekli değildir. Bunun temel nedeni, bu modellerin bölünme kararlarını **Gini katsayısı** veya **entropi** gibi kriterlere göre vermesidir. Bu kriterler, en iyi bölünme noktalarını belirlerken yalnızca özelliklerin sırasına ve sınıf dağılımına odaklanır. Özelliklerin mutlak değerleri, bölünme kararlarını etkilemez. Bu durum, ölçeklendirme yapılmış ya da yapılmamış olmasının model performansına bir etkisi olmadığı anlamına gelir.

Bu durumu **Model-1** ve **Model-2** üzerinde yaptığımız analizlerle açıklıyoruz. Örneğin, yaş bilgisi içeren bir sütununuz olduğunu varsayalım:

- **Normalizasyon yapılmadan**: yaş > 18
- **Normalizasyon yapıldıktan sonra**: normalize_yaş > 0.4

Bu eşikler örnek olarak verilmiştir, ancak dikkat edilmesi gereken husus, normalizasyon yapılıp yapılmadığından bağımsız olarak, bölünme sonuçlarının aynı olmasıdır. Çünkü ağaç tabanlı modeller, yalnızca özelliğin sırasına odaklanır ve ölçek değişikliklerinden etkilenmez.

Yine de, veri işleme süreçlerinde tutarlılığı sağlamak ve diğer modellerle karşılaştırma yaparken uyumlu bir pipeline oluşturmak adına, ölçeklendirme işlemleri bazı durumlarda kullanılabilir. Ancak, bu işlemler ağaç tabanlı modeller için zorunlu değildir.

1.2.1.2. Model-2: Random Forest + Scale

Burada scale edilmiş data modele verilmiştir. Default hiperparametreler kullanılmıştır. **Sonuç olarak**, ağaç tabanlı modellerde ölçeklendirme işlemi performans üzerinde bir fark yaratmaz. Bu çalışmada da, bu durumu göstermek amacıyla ölçeklendirilmiş ve ölçeklendirilmemiş veriler üzerinde Model-1 ve Model-2 ile karşılaştırma yapılmıştır.

1.2.1.3. Model-3: Random Forest + PCA

Default hiperparametreler kullanılarak Random Forest modeli eğitilmiştir. Data olarak PCA sonucu elde edilmiş ve seçilmiş olan değerler verilmiştir.

1.2.1.4. Model-4: Random Forest + PCA + Balance Weights

```
model_4 = RandomForestClassifier(random_state=42, n_estimators=100, class_weight='balanced')
```

Görüldüğü üzere burada class_weights bölümünde balanced eklenmiştir. Balanced ekleyerek dengesiz verisetimizde güzel değişimler olacak mı kontrol etmek istiyoruz. Belki modelimizi güzel şekilde etkileyebilir. Eğitim datasına PCA ile elde ettiğimiz dönüştürülmüş veriler verilecektir.

1.2.1.5. Model-5: Random Forest + Balance Weights

```
model_5 = RandomForestClassifier(random_state=42, n_estimators=100, class_weight='balanced')
```

Burada da benzer şekilde class_weight = ‘balanced’ kullanıyoruz lakin eğitim dataseti olarak normal datamızı veriyoruz. PCA işlemi yaptığımız dataseti vermiyoruz.

1.2.1.6. Random Forest Modellerin Sonuçları

Modellerde görüldüğü gibi PCA Balanced test üzerinde en yüksek başarıyı gösteriyor bunun olmasının muhtemel sebebi PCA Balanced ile daha genel bir model oluşturulduğu için Test verisini genelleme başarısı da biraz olsun daha yüksek olabilir diye düşünmekteyim.

	Model	Dataset	F1 Score
0	RF + Scale	Train	0.922883
1	RF Base	Train	0.922883
2	RF + PCA	Train	0.922246
3	Balanced	Train	0.921846
4	PCA Balanced	Train	0.920522
5	PCA Balanced	Test	0.916008
6	Balanced	Test	0.915829
7	RF + PCA	Test	0.915296
8	RF + Scale	Test	0.915041
9	RF Base	Test	0.915041

Tablo 5

1.2.1.7. Hiperparametre Optimizasyonu RF PCA + Balanced

```
# Hiperparametreler için grid
param_grid = {
   'n_estimators': [50, 100, 200],        # Ağaç sayısı
   'max_depth': [10, 20, 30],        # Maksimum derinlik
   'min_samples_split': [ 5, 10],        # Dallanma için minimum örnek sayısı
   'min_samples_leaf': [12, 4],          # Yaprak düğümündeki minimum örnek sayısı
   'max_features': ['sqrt', 'log2', None],  # Her bir ağaç için kullanılacak maksimum özellik sayısı
}
```

Hiperparametre optimizasyonu için bir grid belirlenmiş olup bunun üzerinde GridSearchCV fonksiyonu ile optimizasyon yapılmıştır. RandomSearch kullanılmadı çünkü zaten çok uzun süren bir eğitim ilemi yoktu ve datamız çok büyük değildi.

Sonuç olarak: Best Parameters: {'max_depth': 20, 'max_features': 'sqrt', 'min_samples_leaf': 4, 'min_samples_split': 5, 'n_estimators': 100}

n_estimators: Bu, modelde kaç tane ağaç kullanılacağını belirleyen bir hiperparametre. Ağaç sayısı arttıkça model daha fazla öğrenme yapıyor, ama tabii ki eğitim süresi de uzuyor. Örneğin, 50 ağaçla çalıştırdığımızda model daha hızlı sonuç veriyor ama bazen yeterince iyi genelleme yapamayabiliyor.

200 ağaçta ise sonuçlar genelde daha kararlı oluyor. Bu yüzden burada farklı ağaç sayılarıyla denemeler yapıyoruz.

max_depth: Bu parametre, her bir ağacın maksimum derinliğini belirliyor. Daha derin bir ağaç, daha fazla detay öğrenebiliyor ama bu durum overfitting'e (aşırı öğrenme) yol açabilir. Örneğin, derinliği 10 olan bir ağaç, daha genel bir model üretirken; 30 derinliğinde bir ağaç, veri setindeki en küçük ayrıntılara bile uyum sağlayabilir. Burada, farklı derinliklerle deneme yaparak en uygun değeri bulmaya çalışıyoruz.

min_samples_split: Bu, bir düğümün dallanabilmesi için gerekli olan minimum örnek sayısını belirliyor. Örneğin, bir düğümde en az 5 örnek varsa dallanıyor, aksi halde dallanma olmuyor. Eğer bu değeri çok küçük yaparsak modelimiz çok karmaşık hale gelebilir. Büyük yaparsak da model daha genel ve dengeli sonuçlar verebilir.

min_samples_leaf: Bu parametre, yaprak düğümünde bulunması gereken minimum örnek sayısını ifade ediyor. Yaprak düğümünü düşünün, yani artık dallanma yapılmıyor. Buradaki minimum örnek sayısı modelin daha sade mi yoksa daha karmaşık mı olacağını belirliyor. Örneğin, yaprakta 12 örnek olması gerektiğinde model biraz daha basit oluyor ve aşırı öğrenmeyi önlüyor.

max_features: Bu hiperparametre ise her bir ağacın dallanması sırasında kullanabileceği maksimum özellik sayısını belirliyor. Mesela, 'sqrt' dediğimizde toplam özellik sayısının karekökü kadar özelliği kullanıyor. 'log2' dediğimizde ise logaritma (base 2) kadar özellik seçiliyor. Eğer None dersek, tüm özellikler kullanılıyor. Bu parametre aslında çeşitlilik yaratıyor, çünkü her ağaç aynı özelliklere bakmak zorunda kalmıyor.

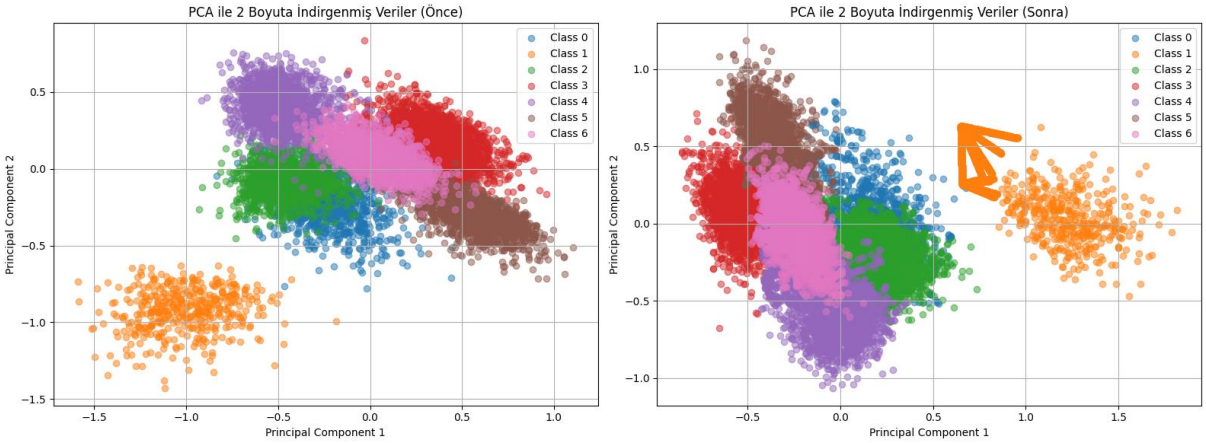
1.2.1.8. Model-6: Random Forest PCA + ADASYN

Çalışmamızın başında bir ADASYN kullanan bir makaleye eleştiride bulunmuştuk. Şimdi bununla ilgili çalışıyoruz. Buradaki bölüm [Comparison of multiclass classification techniques using dry bean dataset] makalesinde yapılan işlemlerin aslında sağlıklı olmadığını göstermek amacıyla yapılmıştır. Öncelikle ADASYN ile oversampling yapıyoruz.

Counter({0: 2881, 2: 2873, 4: 2868, 3: 2837, 1: 2837, 5: 2783, 6: 2749})

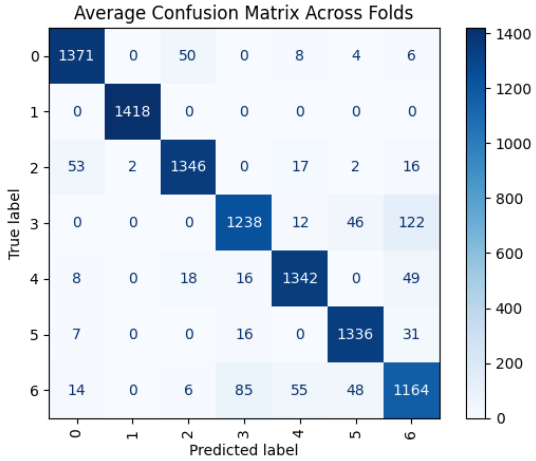
Gördüğümüz gibi ADASYN sonrasında veriler dengeli bir hale geldi. Şimdi Adasyn öncesi ve Adasyn sonrası verileri üzerine PCA dönüşüm işlemleri yapılarak karşılaştırıp Adasyn oversampling işleminin ne yaptığı görselleştirilecektir.

PCA işlemi sonrasında aşağıdaki gibi sonuçlar elde edilmiştir. Oversampling sonrası PCA ile görselleştirme yapıldığında görüldüğü üzere ADASYN aslında sağda görünen turuncu noktalara eklemeler yapmıştır daha çok ve bu şekilde kendi eklediği noktalarla modelin validation bölümünde performansı yükselmiş gibi göstermektedir.

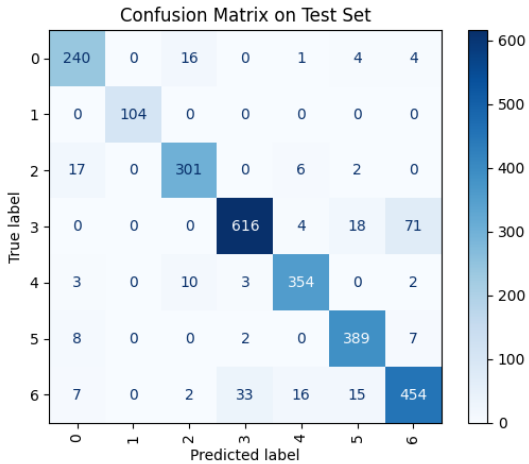


Şekil 29

Aşağıdaki Şekil'de görüldüğü üzere Classification Report sonucunda 3 ve 6'nın hala karıştırıldığı görülmektedir. Ve diğerlerinden daha kötü bir sonuç elde edilmektedir. Makaledeki sonuçları elde edememizin sebebi oversampling işleminin test datasına da uygulanmış olmasıdır. Bireysel çalışmalarda datamızın tamamına oversampling yaptığımızda ve diğer işlemleri de uyguladığımızda biz de oldukça yüksek sonuçlar elde ediyoruz ama bunun sebebi modelin test datası hakkında bilgi edinmesi (data leakage) ve testteki zaten tahmin edilen datanın sayısını çoğaltarak başarının artırılmasıdır.



Şekil 30



Şekil 31

1.2.1.9. Model-1: XGBoost Base Model

Karşılaştırma yapmak amacıyla bir base model oluşturulmuştur.

```
xgb_model_1 = XGBClassifier(
   objective="multi:softprob",  # Çok sınıflı problemler için
   eval_metric="mlogloss",    # Değerlendirme metriği
   random_state=42
)
```

Çok sınıflı problemlerle çalışıldığı için gerekli ayarlamalar yapılmıştır.

1.2.1.10. Model-2: XGBoost PCA

PCA işlemi özellikleri indirgenmiş dataset ile eğitim yapılmıştır.

1.2.1.11. Model-3: XGBoost PCA + Balancing Weights

Bu modelde, sınıf dengesizliklerini ele almak için **weight balancing** yöntemi uygulanmış ve boyut azaltımı amacıyla **PCA (Principal Component Analysis)** kullanılmıştır. Ardından, bu işlemden geçirilen veriler üzerinde **XGBoost** modeli eğitilerek sınıflandırma performansı değerlendirilmiştir.

1.2.12. XGBoost Sonuçlar

	Model	Dataset	F1 Score
0	XGB_Base	Train	0.924796
1	XGB_Bal	Train	0.920522
2	XGB_PCA	Test	0.919295
3	XGB_Bal	Test	0.918962
4	XGB_Base	Test	0.918197
5	XGB_PCA	Train	0.917714

Tablo 6

Test datası üzerinde XGB ve PCA işleminin sonuçları daha iyi gözükmektedir çok da fark olmasa da. Bunun üzerine Hiperparametre optimizasyonu yapılacaktır.

1.2.12. XGBoost Hiperparametre Optimizaasyonu

learning_rate: Bu parametre, modelin her adımda ne kadar öğrenmesi gerektiğini belirliyor. Genelde düşük bir öğrenme oranı (mesela 0.01) modelin daha yavaş ama daha dikkatli öğrenmesini sağlıyor. Yüksek bir oran (0.2 gibi) ise modelin daha hızlı öğrenmesini sağlasa da bazen doğruluğu düşürebiliyor. Bu yüzden farklı değerlerle deneme yaparak en uygun oranı bulmaya çalışıyoruz.

max_depth Bu, her bir ağacın maksimum derinliğini belirleyen parametre. Ağaçlar derinleştikçe model, daha karmaşık ilişkileri öğrenebilir ama aşırı öğrenme (overfitting) riski de artar. Örneğin, max_depth=3 dediğimizde ağaç daha sığ olur ve genelleştirme daha iyi olabilir. Ama max_depth=7 ile daha detaylı bir öğrenme yapılır.

Subsample Bu parametre, her bir ağacın eğitimi sırasında kullanılacak verinin yüzde kaçını belirler. Örneğin, subsample=0.8 dediğimizde model, veri setinin %80'ini kullanır ve bu çeşitlilik sağlar. subsample=1.0 ise tüm veriyi kullanır. Çeşitlilik, overfitting'i azaltmak için faydalı olabilir.

colsample_bytree Bu da, her bir ağacın dallanmasında kullanılacak özelliklerin (features) yüzde kaçını belirler. Mesela, colsample_bytree=0.8 dediğimizde model her seferinde özelliklerin %80'i ile çalışır, böylece farklı ağaçlar arasında çeşitlilik artar. colsample_bytree=1.0 ise tüm özellikleri kullanır.

1.2.2. Regresyon

Ridge regresyon, **çoklu doğrusal bağıntı** (multicollinearity) gibi sorunların yaşandığı durumlarda tercih edilen bir regresyon yöntemidir. Özellikle, bağımsız değişkenler arasında yüksek korelasyon olduğunda, standart **doğrusal regresyon (Linear Regression)** katsayı tahminlerinde kararsız sonuçlara yol açabilir. Ridge regresyon, bu durumu aşmak için **L2 düzenlileştirme** (L2 regularization) kullanır ve bizim datamızda da mevcut bir multicollinearity vardır.

R^2 (Determination Coefficient): Modelin açıklayabildiği varyans oranını ifade eder.

MSE (Mean Squared Error): Ortalama kare hata değeri, modelin tahminlerindeki hata büyüklüğünü ölçer.

MAE (Mean Absolute Error): Ortalama mutlak hata değeri, tahmin hatalarının ortalamasını ifade eder.

MAPE (Mean Absolute Percentage Error): Tahminlerin hedef değerlerden yüzde olarak ne kadar saptığını gösterir.

SMAPE (Symmetric MAPE): Tahminlerin ve gerçek değerlerin yüzdelik farkını simetrik bir şekilde ifade eder.

1.2.2.1. Ridge Regresyon Base Model

Model performansını değerlendirmek ve karşılaştırma yapmak için **Ridge Regresyon** kullanılarak bir **base model** oluşturulmuştur. Base model, daha karmaşık modellerin performansını kıyaslamak için başlangıç noktası olarak seçilmiştir.

1.2.2.2. Ridge Regresyon Hiperparametre Optimizasyonu

Modelin performansı oldukça yüksek çıktı, hiperparametre optimizasyonuyla pek bir beklentim yok. Lakin 5 kfold kullanarak GridSearch uyguluyorum. Parametrelerim aşağıdaki gibidir.

```
param_grid = {
    'alpha': [0.1, 1, 10, 100, 1000],  # L2 regularization strength
}
```

İşlem sonucunda default alpha değeri çıkıyor. Yani en iyi alpha değeri 1 olarak elde ediyorum. Dolayısı ile sonuçlar değişmemektedir.

1.2.2.3. Ridge Regresyon + Çerçeveleme Yöntemi

Çerçeveleme yöntemiyle oluşturulan özellikler kullanılarak Ridge Regresyon modeli eğitilmiş ve test verisi üzerinde değerlendirilmiştir. Modelin performansı oldukça yüksek bir **R^2 (0.97)** skoru ile doğrulanmıştır, bu da modelin hedef değişkenin varyansının %97'sini açıkladığını gösterir.

Düşük bir **MSE (0.014)** ve **MAE (0.076)**, modelin tahmin hatalarının oldukça küçük olduğunu ifade etmektedir. Ancak, **MAPE (514.27)** ve **SMAPE (107.55)** metriklerinin yüksek olması, hedef değerlerin sıfıra yakın olduğu durumlarda yüzdelik hata metriklerinin yanıltıcı olabileceğini göstermektedir. Bu durum, yüzdelik hata metriklerinin yorumlanmasında dikkatli olunması gerektiğini ortaya koymaktadır.

1.2.2.4. Ridge Regresyon + Çerçeveleme Yöntemi Hiperparametre Optimizasyonu

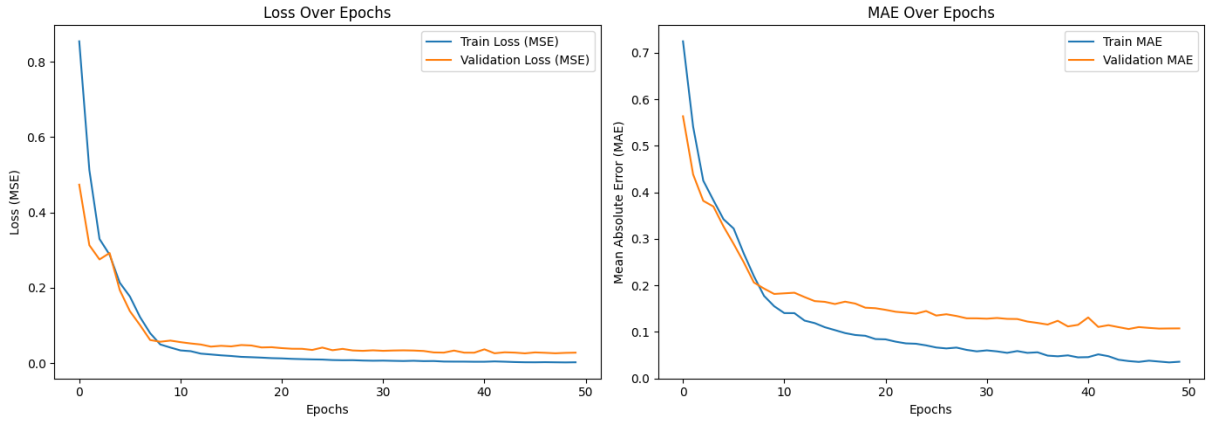
Çerçeveleme yöntemiyle oluşturulan özellikler, Ridge Regresyon modeline uygulanmış ve ardından **GridSearchCV** ile hiperparametre optimizasyonu gerçekleştirilmiştir. Bu süreçte, modelin düzenlileştirme gücünü belirleyen **alpha** parametresi optimize edilmiş ve en iyi değer olarak **alpha=1** bulunmuştur. Bu zaten modelin default değeridir. Sonuçlarda bir değişim olmamıştır.

1.2.2.5. LSTM Model

LSTMRegressionModel sınıfı, zaman serisi verileri için **LSTM tabanlı bir regresyon modeli** oluşturmak, eğitmek ve test etmek amacıyla tasarlanmıştır. TensorFlow Keras kullanılarak inşa edilen model, giriş verilerinin zaman bağımlılıklarını öğrenmek için **LSTM katmanı** ile başlar. Ardından, özellikleri işlemek için bir **Dense (tam bağlı)** katman eklenir ve en sonunda regresyon çıktısını üretmek için bir **tek nöronlu çıkış katmanı** bulunur.

Model, **ReLU aktivasyonu**, 'adam' optimizasyon algoritması ve 'mse' (Mean Squared Error) kayıp fonksiyonuyla derlenir. Eğitim sırasında doğrulama verisi kullanılarak genelleştirme performansı izlenir. train metodu, modeli belirli bir **epoch** ve **batch size** ile eğitir ve eğitim geçmişini döndürür. evaluate metodu ise test veri seti üzerinde modelin **MSE** ve **MAE** metriklerini hesaplayarak performansı değerlendirir.

Bu amaçla önce train datası üzerinde validation yapılarak model performansı değerlendirilmiştir. Daha sonra Evaluation yapılarak doğrulama işlemi test verisi üzerinde gerçekleştirilmiştir.



Şekil 32

2. DENEYSEL BULGULAR

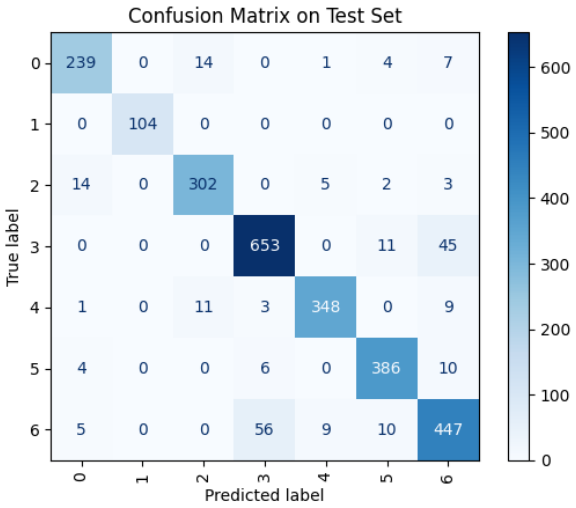
2.1 Sınıflandırma Sonuçları

	Test Verisi üzerinde Sonuçlar				
	accuracy	**precision**	**recall**	**Macro f1**	**weighted f1**
RF: Base Model	**0.9150**	**0.9275**	**0.9262**	**0.9268**	**0.9150**
RF: Scaled	**0.9150**	**0.9275**	**0.9262**	**0.9268**	**0.9150**
RF: PCA	**0.9150**	**0.9283**	**0.9272**	**0.9277**	**0.9152**
RF: PCA + Balance	**0.9158**	**0.9290**	**0.9273**	**0.9281**	**0.9160**
RF: Balanced	**0.9158**	**0.9289**	**0.9274**	**0.9281**	**0.9158**
RF: PCA + Adasyn	**0.9073**	**0.9159**	**0.9241**	**0.91964**	**0.9072**

XGB: Base Model	**0.9180**	**0.9316**	**0.9288**	**0.9301**	**0.9181**
XGB: PCA	**0.9191**	**0.9310**	**0.9301**	**0.9305**	**0.9192**
XGB: PCA + Balance	**0.9187**	**0.9293**	**0.9286**	**0.9288**	**0.9189**

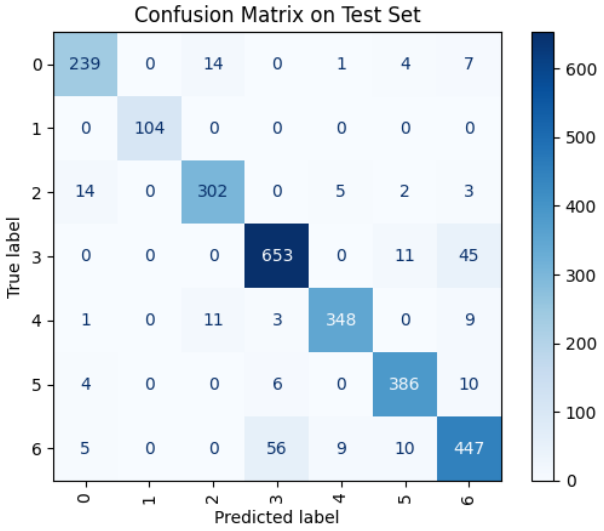
Tablo 7

Hata Matrisleri



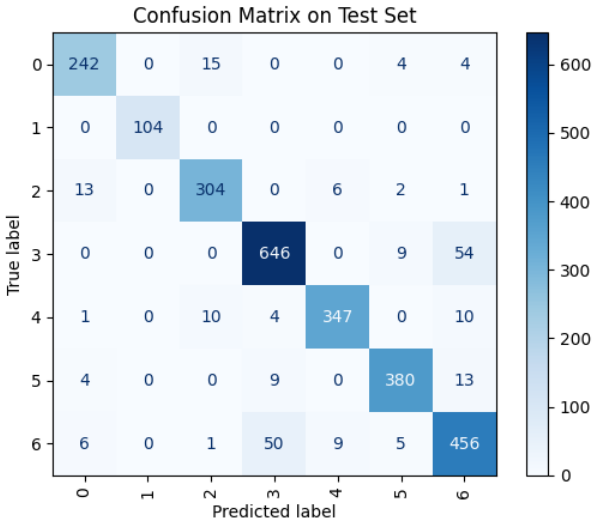
RF: Base Model

Şekil 33



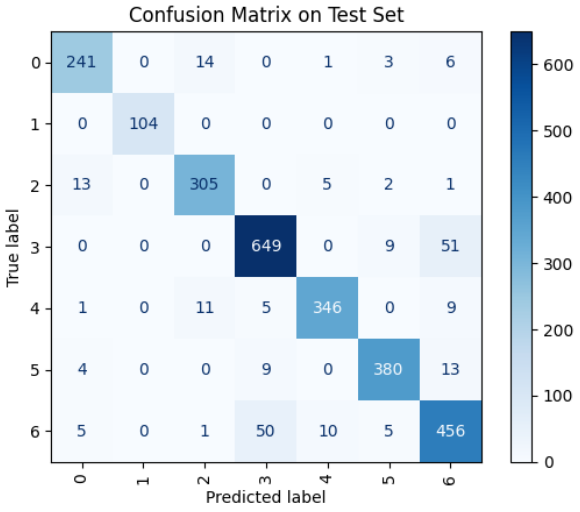
RF: Scaled

Şekil 34



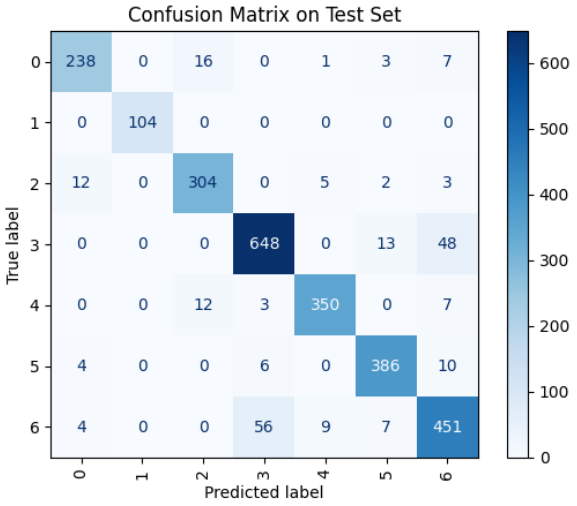
RF: PCA

Şekil 35



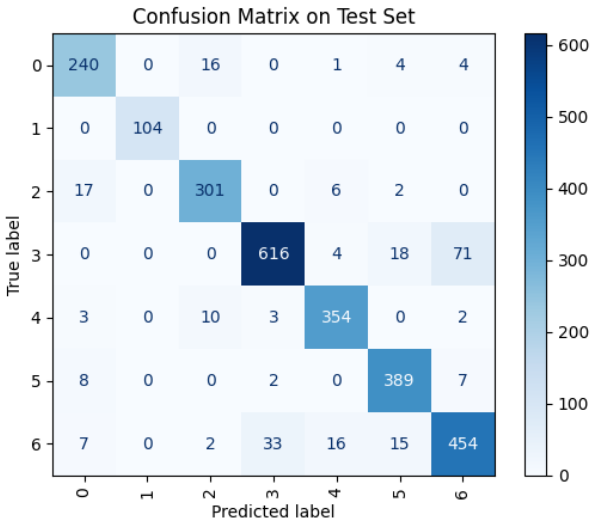
RF: PCA + Balance

Şekil 36



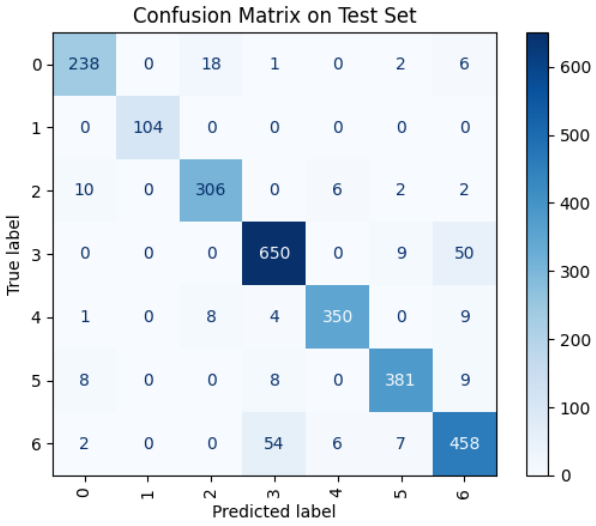
RF: Balanced

Şekil 37



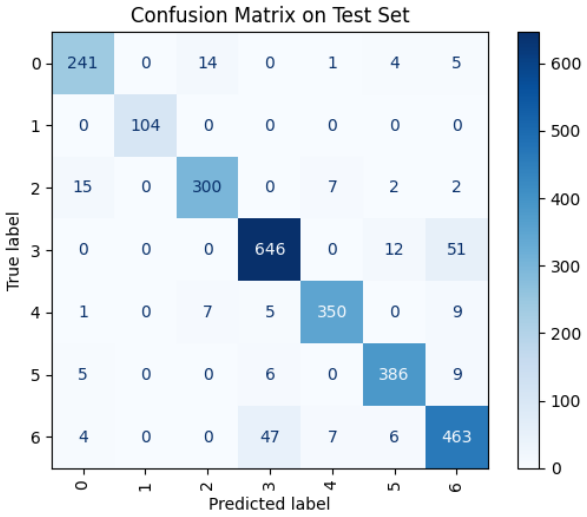
RF: PCA + Adasyn

Şekil 38



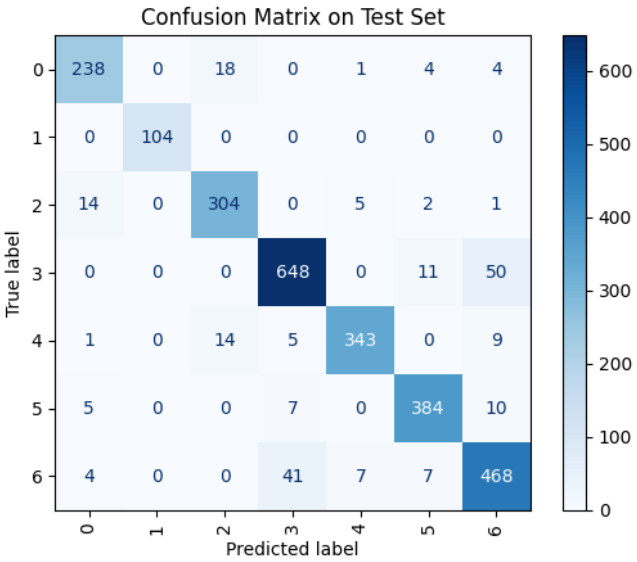
XGB: Base Model

Şekil 39



XGB: PCA

Şekil 40



XGB: PCA + Balance

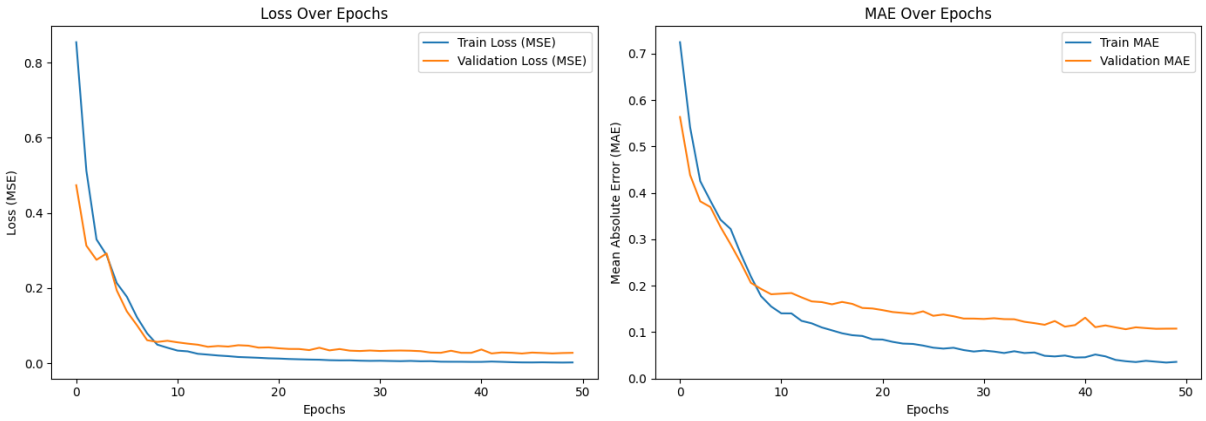
Şekil 41

2.2 Regresyon Sonuçları

	Test Verisi üzerinde Sonuçlar				
	R2	MAE	MSE	MAPE	SMAPE
Ridge Base	0.9815	0.0765	0.0097	49.3835	18.6026
Ridge + Window	0.9737	0.0142	0.0765	514.2763	107.5467
LSTM + Window	0.9693	0.0166	0.1014	63.1526	23.2863

Tablo 8

LSTM grafiği



Şekil 42

3.SONUÇ

3.1. Classification

Random Forest (RF):

- RF modeliyle yapılan deneylerde, ölçeklendirme (scale) işleminin model performansı üzerinde belirgin bir etkisi olmadığı görülmüştür. Bunun sebebi, RF'nin ağaç tabanlı bir yöntem olması ve veri ölçeğinden bağımsız çalışabilmesidir.
- PCA uygulaması, boyut indirgemesi ile model performansında hafif bir artış sağlamıştır (%0.1'lik bir iyileşme).
- Dengesiz veri problemlerini ele almak için kullanılan class_weight='balanced' yaklaşımı, sınıf dengesizliklerinin etkisini azaltarak modelin genelleme kapasitesini artırmıştır.
- ADASYN ile yapılan denemelerde, veri manipülasyonunun test verisine uygulanması durumunda model performansında yanıltıcı artışlar gözlemlenmiştir. Bu durum, test verisinin manipüle edilmemesi gerektiğinin önemini vurgulamaktadır.

XGBoost (XGB):

- XGBoost modelleri, Random Forest'a kıyasla genellikle biraz yüksek performans göstermiştir.
- PCA ve sınıf dengesi ayarlamalarıyla XGB modelinin genelleme başarısında iyileşme sağlanmıştır.

3.2. Regression

Ridge Regresyon:

- Ridge regresyon, veri setindeki çoklu doğrusal bağıntı problemini ele alarak yüksek R^2 skorları elde etmiştir (%98.15).
- Çerçeveleme (windowing) yöntemi maalesef beklenen etkiyi yaratamamıştır. Bunun sebebi belki de Ridge Regresyonun kendi çalışma yapısı ile ilgilidir.

LSTM:

- LSTM tabanlı zaman serisi modelleri, özellikle geçmiş veri örüntülerini öğrenmede başarılı olmuştur.
- Eğitim ve doğrulama aşamalarında düşük kayıp değerleri elde edilmiştir. Ancak, Ridge regresyon ile kıyaslandığında LSTM'nin performansı regresyon görevinde biraz daha düşük kalmıştır.
- Bu düşük kalmanın sebebinin makine öğrenmesi modellerinin düşük veride yapay sinir ağlarından daha başarılı olmasına bağlıyorum.

4.EK

4.1 Sınıflandırma Kodları

Data

```
!pip install ydata_profiling

import pandas as pd
import seaborn as sns
```

```
import ydata_profiling
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split, StratifiedKFold,
cross_val_score
from sklearn.preprocessing import MinMaxScaler, StandardScaler
from imblearn.over_sampling import SMOTE
from imblearn.under_sampling import RandomUnderSampler
from imblearn.combine import SMOTEENN

from sklearn.model_selection import GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, accuracy_score,
ConfusionMatrixDisplay, f1_score, confusion_matrix
from sklearn.decomposition import PCA

from xgboost import XGBClassifier

from collections import Counter

import pandas as pd

# Dataset metin bilgisini okuma
data_url = "https://github.com/AysenurYrr/ML-
Lab/raw/refs/heads/main/DryBeanDataset/Dry_Bean_Dataset.xlsx"

# Dataseti çekme ve okuma
data = pd.read_excel(data_url)
print(data.shape)
data.head()

# Genel bilgi
dataset_info = data.info()

# Benzersiz sınıf isimlerini kontrol etme
class_names = data['Class'].unique()
class_names
```

EDA

```
feature_array = data.columns[:-1]  # Class hariç tüm sütunlar

# Class Analizi

plt.figure(figsize=(25, 5))
sns.countplot(x='Class', data=data)
plt.show()
```

```
# Tanımlayıcı İstatistikler (Descriptive Statistics)

data.describe()

# Box Plot Analizi

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Boxplot çizmek için fonksiyon
def plot_boxplot(data, feature, class_name):
    """
    Verilen bir özellik ve sınıf için boxplot çizer.

    Parameters:
        data: pandas.DataFrame
            Analiz yapılacak veri seti.
        feature: str
            Boxplot oluşturulacak özellik adı.
        class_name: str
            Belirli bir sınıfın adı.
    """
    plt.figure(figsize=(10, 6))
    sns.boxplot(x=data['Class'], y=data[feature], data=data)
    plt.title(f"Boxplot of {feature} for Class {class_name}",
fontsize=14)
    plt.xlabel("Class", fontsize=12)
    plt.ylabel(feature, fontsize=12)
    plt.xticks(rotation=45)
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()

for feature in feature_array:
    plot_boxplot(data, feature, "All Classes")
```

Klasik PreProcess İşlemleri

```
# Delete Duplicate

data = data.drop_duplicates()
data = data.reset_index(drop=True)
data.shape

data

from sklearn.preprocessing import OneHotEncoder,LabelEncoder
```

```
# Label Encoding işlemi
label_encoder = LabelEncoder()
data_encoded_label = data.copy()
data_encoded_label['Class_Encoded'] =
label_encoder.fit_transform(data['Class'])

data_encoded_label.drop("Class", axis=1, inplace = True)
print(data_encoded_label.shape)
data_encoded_label

# OneHotEncoder işlemi
encoder = OneHotEncoder(sparse_output=False)
encoded_classes = encoder.fit_transform(data[['Class']])

# Yeni DataFrame'e ekleme
data_encoded_ohe = pd.DataFrame(encoded_classes,
columns=encoder.get_feature_names_out(['Class']))
data_encoded_ohe = pd.concat([data, data_encoded_ohe], axis=1)
data_encoded_ohe.drop("Class", axis=1, inplace = True)
print(data_encoded_ohe.shape)
data_encoded_ohe

data_encoded_label

X = data_encoded_label.drop("Class_Encoded", axis=1)
y = data_encoded_label["Class_Encoded"]
X.head()

y.head()
```

Veriyi Eğitime Hazırlama

```
# Veriyi test ve eğitim olarak ayırma
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42, stratify=y)

# Eğitim setini ölçekleme
scaler = MinMaxScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

print("\nScale edilmemiş")
print(X_train.shape)
print(y_train.shape)
print("\nScale edilmiş")
print(X_train_scaled.shape)
print(y_train.shape)
```

```
print("\n*****************")
print("Test Datası, Gerçek dünyaya uygun olması açısından balance gibi
şeyler yapılmayacaktır. Sadece Scale işlemi uygulanır")
print(X_test_scaled.shape)
print(y_test.shape)

def visualize_pca_explained_variance(pca_model):
    """
    PCA açıklanabilirlik oranlarını görselleştirir ve oranları
yazdırır.

    Parameters:
        pca_model (PCA): PCA model nesnesi.

    Returns:
        None
    """
    explained_variance = pca_model.explained_variance_ratio_
    cumulative_variance = np.cumsum(explained_variance)

    # Açıklanabilirlik oranlarını yazdırma
    for i, (ind_var, cum_var) in enumerate(zip(explained_variance,
cumulative_variance), start=1):
        print(f"Principal Component {i}: Individual Explained Variance
= {ind_var:.4f}, Cumulative Explained Variance = {cum_var:.4f}")

    # Görselleştirme
    plt.figure(figsize=(8, 5))
    plt.bar(range(1, len(explained_variance) + 1), explained_variance,
alpha=0.7, color="blue", label="Individual Explained Variance")
    plt.step(range(1, len(explained_variance) + 1),
cumulative_variance, where="mid", label="Cumulative Explained
Variance", color="orange")
    plt.xlabel("Principal Components")
    plt.ylabel("Explained Variance Ratio")
    plt.title("Explained Variance by Principal Components")
    plt.legend()
    plt.tight_layout()
    plt.show()

pca_model = PCA(n_components=10)
X_train_pca = pca_model.fit_transform(X_train_scaled)
X_test_pca = pca_model.transform(X_test_scaled)

# PCA açıklanabilirlik oranlarını görselleştirme
visualize_pca_explained_variance(pca_model)

# İlk 5 feature'ı seçme
X_train_reduced = X_train_pca[:, :7]
```

```
X_test_reduced = X_test_pca[:, :7]
print("X_train_reduced Shape",X_train_reduced.shape)
print("X_test_reduced Shape",X_test_reduced.shape)
```

Fonksiyonlar

```
# Validation Function

import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay

def KFold_validation(X, y, model=None, cv=2):
    """
    Label encoded veri ile k-fold çapraz doğrulama uygulayarak bir
modeli değerlendirir ve
    confusion matrix bilgilerini görselleştirir.

    Parameters:
        X (pd.DataFrame or np.ndarray): Özellikler (features).
        y (pd.Series or np.ndarray): Sınıf etiketleri (labels, label
encoded formatında).
        model: Kullanılacak makine öğrenimi modeli (varsayılan
RandomForestClassifier).
        cv (int): Çapraz doğrulama kat sayısı (folds).

    Returns:
        dict: CV doğruluk skorları, ortalama skor, standart sapma ve
confusion matrices.
    """
     # Veriyi NumPy dizisine dönüştürme (Pandas uyumsuzluğu önlemek
için)
    if isinstance(X, pd.DataFrame):
        X = X.values
    if isinstance(y, pd.Series):
        y = y.values

    skf = StratifiedKFold(n_splits=cv, shuffle=True, random_state=42)
    scores = []
    f1_scores = []
    confusion_matrices = []

    for train_index, validation_index in skf.split(X, y):
        X_train, X_validation = X[train_index], X[validation_index]
        y_train, y_validation = y[train_index], y[validation_index]


        # Model eğitimi ve tahmin
        model.fit(X_train, y_train)
```

```
        y_pred = model.predict(X_validation)

        # Skor kaydı
        scores.append(model.score(X_validation, y_validation))

        # F1 score kaydı
        f1 = f1_score(y_validation, y_pred, average="weighted")
        f1_scores.append(f1)

        # Confusion matrix
        cm = confusion_matrix(y_validation, y_pred,
labels=np.unique(y))
        confusion_matrices.append(cm)

    # Ortalama confusion matrix
    mean_cm = np.mean(confusion_matrices, axis=0).astype(int)
    mean_f1 = np.mean(f1_scores)

    # Görselleştirme
    disp = ConfusionMatrixDisplay(confusion_matrix=mean_cm,
display_labels=np.unique(y))
    disp.plot(cmap='Blues', xticks_rotation='vertical')
    plt.title("Average Confusion Matrix Across Folds")
    plt.show()

    # Sonuçları döndürme
    results = {
        #"CV Scores": scores,
        "Mean Accuracy": np.mean(scores),
        #"Mean Standard Deviation": np.std(scores),
        #"F1 Scores": f1_scores,
        "Mean F1 Score": np.mean(f1_scores)
        #"Confusion Matrices": confusion_matrices
    }
    return results, mean_f1

# Evaluation Function

def evaluation_func(X_test, y_test, model):
    """
    Test verisi üzerinde bir modelin performansını değerlendirir ve
görselleştirir.

    Parameters:
        X_test (pd.DataFrame or np.ndarray): Test özellikleri
(features).
        y_test (pd.Series or np.ndarray): Test sınıf etiketleri
(labels).
        model: Eğitimli makine öğrenimi modeli.
```

```
    Returns:
        dict: Classification report, confusion matrix.
    """
    # Veriyi NumPy dizisine dönüştürme (Pandas uyumsuzluğu önlemek
için)
    if isinstance(X_test, pd.DataFrame):
        X_test = X_test.values
    if isinstance(y_test, pd.Series):
        y_test = y_test.values

    # Test verisi üzerinde tahmin yap
    y_pred = model.predict(X_test)

    # Classification report
    class_report = classification_report(y_test, y_pred,
output_dict=True)

    # Confusion matrix
    cm = confusion_matrix(y_test, y_pred, labels=np.unique(y_test))
    f1_score_ = f1_score(y_test, y_pred, average="weighted")

    # Görselleştirme
    disp = ConfusionMatrixDisplay(confusion_matrix=cm,
display_labels=np.unique(y_test))
    disp.plot(cmap='Blues', xticks_rotation='vertical')
    plt.title("Confusion Matrix on Test Set")
    plt.show()

    # Sonuçları döndürme
    results = {
        #"Confusion Matrix": cm,
        "f1 score": f1_score_,
        "Classification Report": class_report
    }
    return results, f1_score_
```

Model Random Forest

```
# Model-1: Base Model (Scale yok)

# Fonksiyonunu ksiyonu çağır ve sonuçları al
base_model = RandomForestClassifier(random_state=42)
results, rf_base_train_f1 = KFold_validation(X_train, y_train,
base_model, cv=2)
print(results)
```

```
test_results, rf_base_test_f1 = evaluation_func(X_test, y_test,
base_model)
print("Classification Report:")
for label, metrics in test_results["Classification Report"].items():
    print(f"{label}: {metrics}")

# Model-2: Scale Var

print("Scale edilmiş, Balance edilmemiş")
print(X_train_scaled.shape)
print(y_train.shape)

# Fonksiyonunu çağır ve sonuçları al
model_2 = RandomForestClassifier(random_state=42)
results, rf_scale_train_f1 = KFold_validation(X_train_scaled, y_train,
model_2, cv=2)
results

test_results, rf_scale_test_f1 = evaluation_func(X_test_scaled, y_test,
model_2)
print("Classification Report:")
for label, metrics in test_results["Classification Report"].items():
    print(f"{label}: {metrics}")

# Model-3: PCA

# İlk 5 feature'ı seçme
X_train_reduced = X_train_pca[:, :5]
X_test_reduced = X_test_pca[:, :5]
print("X_train_reduced Shape",X_train_reduced.shape)
print("X_test_reduced Shape",X_test_reduced.shape)

model_3 = RandomForestClassifier(random_state=42)
results, rf_pca_train_f1 = KFold_validation(X_train_reduced, y_train,
model_3, cv=2)
results

test_results, rf_pca_test_f1 = evaluation_func(X_test_reduced, y_test,
model_3)
print("Classification Report:")
for label, metrics in test_results["Classification Report"].items():
    print(f"{label}: {metrics}")

# Model-4: Balance Weights + PCA

model_4 = RandomForestClassifier(random_state=42, n_estimators=100,
class_weight='balanced')
results, rf_pca_bal_train_f1 = KFold_validation(X_train_reduced,
y_train, model_4, cv=2)
```

```
results

test_results, rf_pca_bal_test_f1 = evaluation_func(X_test_reduced,
y_test, model_4)
print("Classification Report:")
for label, metrics in test_results["Classification Report"].items():
    print(f"{label}: {metrics}")

# Model-5: RF Balanced

model_5 = RandomForestClassifier(random_state=42, n_estimators=100,
class_weight='balanced')
results, rf_bal_train_f1 = KFold_validation(X_train, y_train, model_5,
cv=2)
results

test_results, rf_bal_test_f1 = evaluation_func(X_test, y_test, model_5)
print("Classification Report:")
for label, metrics in test_results["Classification Report"].items():
    print(f"{label}: {metrics}")

# Verileri bir tabloya yerleştirme
df_scores = {
    'Model': ['RF + PCA', 'RF + PCA', 'RF + Scale', 'RF + Scale', 'RF
Base', 'RF Base','PCA Balanced','PCA Balanced',"Balanced","Balanced"],
    'Dataset': ['Test', 'Train', 'Test', 'Train', 'Test', 'Train',
'Test', 'Train',"Test", "Train"],
    'F1 Score': [
        rf_pca_test_f1, rf_pca_train_f1,
        rf_scale_test_f1, rf_scale_train_f1,
        rf_base_test_f1, rf_base_train_f1,
        rf_pca_bal_test_f1, rf_pca_bal_train_f1,
        rf_bal_test_f1, rf_bal_train_f1
    ]
}

df_scores = pd.DataFrame(df_scores)
df_scores = df_scores.sort_values(by='F1 Score',
ascending=False).reset_index(drop=True)
df_scores
```

Hiperparametre Optimizasyonu

```
# Hiperparametreler için grid
param_grid = {
    'n_estimators': [50, 100, 200],          # Ağaç sayısı
    'max_depth': [10, 20, 30],         # Maksimum derinlik
```

```
    'min_samples_split': [ 5, 10],          # Dallanma için minimum
örnek sayısı
    'min_samples_leaf': [12, 4],            # Yaprak düğümündeki minimum
örnek sayısı
    'max_features': ['sqrt', 'log2', None],  # Her bir ağaç için
kullanılacak maksimum özellik sayısı

# Grid Search
grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42,
class_weight='balanced'),
    param_grid=param_grid,
    cv=2,                       # 5 katlı çapraz doğrulama
    scoring='f1_macro',         # Makro F1 skoru ile değerlendirme
    n_jobs=-1                   # Paralel işlem kullanımı
)

# Modeli eğitme
grid_search.fit(X_train_reduced, y_train)

# En iyi parametreleri görüntüleme
print("Best Parameters:", grid_search.best_params_)

# En iyi modeli seçme
best_model = grid_search.best_estimator_

# Test setinde tahmin yapma
y_test_pred = best_model.predict(X_test_reduced)

# Sonuçları değerlendirme
print("Accuracy:", accuracy_score(y_test, y_test_pred))
print("F1 Score (Weighted):", f1_score(y_test, y_test_pred,
average='weighted'))
print("F1 Score (Macro):", f1_score(y_test, y_test_pred,
average='macro'))

print("\nClassification Report:")
print(classification_report(y_test, y_test_pred))

print("\nConfusion Matrix:")
print(confusion_matrix(y_test, y_test_pred))

# Model-6: RF ADASYN

from imblearn.over_sampling import ADASYN

# ADASYN uygulama
adasyn = ADASYN(random_state=42)
X_adasyn, y_adasyn = adasyn.fit_resample(X_train_scaled, y_train)
```

```
print("ADASYN sonrası veri dağılımı:")
from collections import Counter
print(Counter(y_adasyn))

pca_before_adasyn = PCA(n_components=6)  # 6 bileşene indiriyoruz
(isteğe bağlı artırılabilir)
X_before_adasyn_train_pca =
pca_before_adasyn.fit_transform(X_train_scaled)

# PCA ile açıklanan varyansı görüntüleme
explained_variance = pca_before_adasyn.explained_variance_ratio_
print("Açıklanan varyans oranları:", explained_variance)
print("Toplam açıklanan varyans:", sum(explained_variance))

pca_adasyn = PCA(n_components=6)  # 6 bileşene indiriyoruz (isteğe
bağlı artırılabilir)
X_adasyn_train_pca = pca_adasyn.fit_transform(X_adasyn)
X_adasyn_test_pca = pca_adasyn.transform(X_test_scaled)

# PCA ile açıklanan varyansı görüntüleme
explained_variance = pca_adasyn.explained_variance_ratio_
print("Açıklanan varyans oranları:", explained_variance)
print("Toplam açıklanan varyans:", sum(explained_variance))

# Yan yana iki grafik oluşturmak için subplots ayarı
fig, axs = plt.subplots(1, 2, figsize=(16, 6))

# İlk grafik: PCA ile 2 boyuta indirgenmiş veriler (ADASYN öncesi)
for label in np.unique(y_train):
    axs[0].scatter(
        X_before_adasyn_train_pca[y_train == label, 0],
        X_before_adasyn_train_pca[y_train == label, 1],
        label=f"Class {label}", alpha=0.5
    )
axs[0].set_title("PCA ile 2 Boyuta İndirgenmiş Veriler (Önce)")
axs[0].set_xlabel("Principal Component 1")
axs[0].set_ylabel("Principal Component 2")
axs[0].legend()
axs[0].grid()

# İkinci grafik: PCA ile 2 boyuta indirgenmiş veriler (ADASYN sonrası)
for label in np.unique(y_adasyn):
    axs[1].scatter(
        X_adasyn_train_pca[y_adasyn == label, 0],
        X_adasyn_train_pca[y_adasyn == label, 1],
        label=f"Class {label}", alpha=0.5
    )
axs[1].set_title("PCA ile 2 Boyuta İndirgenmiş Veriler (Sonra)")
```

```
axs[1].set_xlabel("Principal Component 1")
axs[1].set_ylabel("Principal Component 2")
axs[1].legend()
axs[1].grid()

# Grafiklerin gösterimi
plt.tight_layout()
plt.show()

model_6 = RandomForestClassifier(random_state=42)

# Model performansını değerlendirme
results, rf_adasyn_train_f1 = KFold_validation(X_adasyn_train_pca,
y_adasyn, model=model_6, cv=2)
results, rf_adasyn_train_f1

test_results, rf_adasyn_test_f1 = evaluation_func(X_adasyn_test_pca,
y_test, model_6)
print("Classification Report:")
for label, metrics in test_results["Classification Report"].items():
    print(f"{label}: {metrics}")
```

XGBoost

```
# Model-1: Base Model

# XGBoost modeli tanımlama
xgb_model_1 = XGBClassifier(
    objective="multi:softprob",  # Çok sınıflı problemler için
    eval_metric="mlogloss",     # Değerlendirme metriği
    random_state=42
)

# Fonksiyonu çağırma.
results, xgb_train_f1 = KFold_validation(X_train, y_train,
model=xgb_model_1, cv=2)

# Sonuçları yazdırma
print("K-Fold Validation Results:")
print(f"Mean Accuracy: {results['Mean Accuracy']:.4f}")
print(f"Mean F1 Score: {xgb_train_f1:.4f}")

xgb_model_1.fit(X_train,y_train)

test_results, xgb_test_f1 = evaluation_func(X_test, y_test,
xgb_model_1)
print("Classification Report:")
for label, metrics in test_results["Classification Report"].items():
```

```
    print(f"{label}: {metrics}")

# Model-2: PCA

print("X_train_reduced Shape",X_train_reduced.shape)
print("X_test_reduced Shape",X_test_reduced.shape)

# XGBoost modeli tanımlama
xgb_model_2 = XGBClassifier(
    objective="multi:softprob",  # Çok sınıflı problemler için
    eval_metric="mlogloss",     # Değerlendirme metriği
    random_state=42
)

# Fonksiyonu çağırma.
results, xgb_train_pca_f1 = KFold_validation(X_train_reduced, y_train,
model=xgb_model_2, cv=2)

# Sonuçları yazdırma
print("K-Fold Validation Results:")
print(f"Mean Accuracy: {results['Mean Accuracy']:.4f}")
print(f"Mean F1 Score: {xgb_train_pca_f1:.4f}")

xgb_model_2.fit(X_train_reduced,y_train)

test_results, xgb_test_pca_f1 = evaluation_func(X_test_reduced, y_test,
xgb_model_2)
print("Classification Report:")
for label, metrics in test_results["Classification Report"].items():
    print(f"{label}: {metrics}")

# Model-3: Balancing Weights + PCA

xgb_model_3 = RandomForestClassifier(random_state=42, n_estimators=100,
class_weight='balanced')
results, xgb_train_bal_pca_f1 = KFold_validation(X_train_reduced,
y_train, xgb_model_3, cv=2)

# Sonuçları yazdırma
print("K-Fold Validation Results:")
print(f"Mean Accuracy: {results['Mean Accuracy']:.4f}")
print(f"Mean F1 Score: {xgb_train_bal_pca_f1:.4f}")

xgb_model_3.fit(X_train_reduced,y_train)

test_results, xgb_test_bal_pca_f1 = evaluation_func(X_test_reduced,
y_test, xgb_model_3)
print("Classification Report:")
for label, metrics in test_results["Classification Report"].items():
```

```
    print(f"{label}: {metrics}")

# Conculusion

# Verileri bir tabloya yerleştirme
df2_scores = {
    'Model': ['XGB_Base','XGB_Base',
'XGB_PCA','XGB_PCA','XGB_Bal','XGB_Bal'],
    'Dataset': ['Test', 'Train', 'Test', 'Train', 'Test', 'Train'],
    'F1 Score': [
        xgb_test_f1, xgb_train_f1,
        xgb_test_pca_f1, xgb_train_pca_f1,
        xgb_test_bal_pca_f1, xgb_train_bal_pca_f1
    ]
}

df2_scores = pd.DataFrame(df2_scores)
df2_scores = df2_scores.sort_values(by='F1 Score',
ascending=False).reset_index(drop=True)
df2_scores

# HyperParameter Optimization

!pip install scikit-learn==1.5.0

import xgboost as xgb
import pandas as pd
from sklearn.metrics import accuracy_score, f1_score

# XGBoost DMatrices (gerekli format)
dtrain = xgb.DMatrix(X_train_reduced, label=y_train)
dtest = xgb.DMatrix(X_test_reduced, label=y_test)

# Hiperparametre arama alanı
param_grid = {
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7],
    'subsample': [0.8, 1.0],
    'colsample_bytree': [0.8, 1.0],
}

# En iyi sonuçları depolamak için
best_params = None
best_score = float("inf")  # Log loss olduğu için düşük skor daha iyi
best_model = None

# Hiperparametre arama döngüsü
for learning_rate in param_grid['learning_rate']:
    for max_depth in param_grid['max_depth']:
```

```
        for subsample in param_grid['subsample']:
            for colsample_bytree in param_grid['colsample_bytree']:
                params = {
                    'objective': 'multi:softprob',
                    'num_class': len(set(y_train)),
                    'eval_metric': 'mlogloss',
                    'learning_rate': learning_rate,
                    'max_depth': max_depth,
                    'subsample': subsample,
                    'colsample_bytree': colsample_bytree,
                    'seed': 42
                }

                # XGBoost CV
                cv_results = xgb.cv(
                    params=params,
                    dtrain=dtrain,
                    num_boost_round=500,
                    nfold=2,
                    early_stopping_rounds=10,
                    verbose_eval=False
                )

                # En düşük log loss değerini kontrol et
                mean_mlogloss = cv_results['test-mlogloss-mean'].min()
                if mean_mlogloss < best_score:
                    best_score = mean_mlogloss
                    best_params = params

                    # En iyi modeli yeniden eğit
                    best_model = xgb.train(
                        params=params,
                        dtrain=dtrain,
                        num_boost_round=len(cv_results)
                    )

# Test verileri üzerinde tahmin yap
preds = best_model.predict(dtest)
y_pred = preds.argmax(axis=1)

# Accuracy ve F1 Score hesaplama
accuracy = accuracy_score(y_test, y_pred)
f1 = f1_score(y_test, y_pred, average='weighted')

print("Best Parameters:", best_params)
print("Best Log Loss:", best_score)
print("Test Accuracy:", accuracy)
print("Test F1 Score:", f1)
```

Source: Murat KOKLU Faculty of Technology, Selcuk University, TURKEY. ORCID : 0000-0002-2737-2360 mkoklu@selcuk.edu.tr

Ilker Ali OZKAN Faculty of Technology, Selcuk University, TURKEY. ORCID : 0000-0002-5715-1040 ilkerozkan@selcuk.edu.tr

Relevant Papers: KOKLU, M. and OZKAN, I.A., (2020), “Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques.” Computers and Electronics in Agriculture, 174, 105507. DOI: https://doi.org/10.1016/j.compag.2020.105507

Citation Requests / Acknowledgements: KOKLU, M. and OZKAN, I.A., (2020), “Multiclass Classification of Dry Beans Using Computer Vision and Machine Learning Techniques.” Computers and Electronics in Agriculture, 174, 105507.
DOI: https://doi.org/10.1016/j.compag.2020.105507

4.2 Regresyon Kodları

Load Dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, mean_absolute_error,
r2_score
import numpy as np

# Load the dataset while skipping the first row
url = "https://raw.githubusercontent.com/AysenurYrr/ML-
Lab/refs/heads/main/UCI%20Algerian%20Forest%20Fires/Algerian_forest_fir
es_dataset_UPDATE.csv"
dataset = pd.read_csv(url, skiprows=1)

print(dataset.shape)
dataset.head()
```

Veri Ön İşleme

```
## missing values
dataset[dataset.isnull().any(axis=1)]

dataset.loc[:122,"Region"]=0
```

```
dataset.loc[122:,"Region"]=1
df=dataset
df[['Region']]=df[['Region']].astype(int)
df.head()

df.info()

df.iloc[[122]]

## Removing the null values
df=df.dropna().reset_index(drop=True)

##remove the 122nd row
df=df.drop(122).reset_index(drop=True)

##remove the 122nd row
df=df.drop(122).reset_index(drop=True)

df.columns

df[['month','day','year','Temperature','RH','Ws']]=df[['month','day','y
ear','Temperature',' RH',' Ws']].astype(int)
df.drop(columns=[' RH',' Ws'],inplace=True)

df.info()

objects=[features for features in df.columns if
df[features].dtypes=='O']
objects

for i in objects:
    if i!='Classes  ':
        df[i]=df[i].astype(float)

df.rename(columns={'Rain ':'Rain'},inplace=True)

# Rename the "Class  " column to "Class"
df['Classes']=np.where(df['Classes  '].str.contains('not fire'),0,1)
df.drop(columns=['Classes  '], inplace=True)

df.info()

## Let ave the cleaned dataset
df.to_csv('Algerian_forest_fires_cleaned_dataset.csv',index=False)

path = 'https://raw.githubusercontent.com/AysenurYrr/ML-
Lab/refs/heads/main/UCI%20Algerian%20Forest%20Fires/Algerian_forest_fir
es_cleaned_dataset.csv'
df_cleaned = pd.read_csv(path)
```

```
print(f"Shape: {df_cleaned.shape}")
df_cleaned.head()
```

EDA

```
df_copy=df_cleaned.drop(['day','month','year'],axis=1)
df_copy.head()

## categories in classes
df_copy['Classes'].value_counts()

## Density Plot

## Plot desnity plot for all features
df_copy.hist(bins=50,figsize=(20,15))
plt.show()

## Percentage for Pie Chart

percentage=df_copy['Classes'].value_counts(normalize=True)*100

classlabels=["Fire","Not Fire"]
plt.figure(figsize=(12,7))
plt.pie(percentage,labels=classlabels,autopct='%1.1f%%')
plt.title("Pie Chart of Classes")
plt.show()

## Box Plot

import matplotlib.pyplot as plt

# List of continuous features
continuous_features = ["Temperature", "RH", "Rain", "FFMC", "DMC",
"DC", "ISI", "BUI", "FWI"]

# Create boxplots for continuous features
plt.figure(figsize=(15, 10))
for i, feature in enumerate(continuous_features, 1):
    plt.subplot(3, 3, i)  # Adjust grid layout for clarity
    df_cleaned.boxplot(column=feature)
    plt.title(feature)

plt.tight_layout()
plt.show()

import matplotlib.pyplot as plt
```

```
# List of continuous features
continuous_features = ["Temperature", "RH", "Rain", "FFMC", "DMC",
"DC", "ISI", "BUI", "FWI"]

# Create boxplots for Region 0 and Region 1 separately
for region in [0, 1]:
    region_data = df_cleaned[df_cleaned["Region"] == region]

    plt.figure(figsize=(15, 10))
    for i, feature in enumerate(continuous_features, 1):
        plt.subplot(3, 3, i)  # Adjust grid layout for clarity
        region_data.boxplot(column=feature)
        plt.title(f'{feature} (Region {region})')

    plt.tight_layout()
    plt.show()

import matplotlib.pyplot as plt
import seaborn as sns

# Filter data by Region
dftemp_region_1 = df_cleaned[df_cleaned['Region'] == 1]
dftemp_region_0 = df_cleaned[df_cleaned['Region'] == 0]

# Create subplots
fig, axes = plt.subplots(1, 2, figsize=(20, 6), sharey=True)

# Sidi-Bel Abbes Region (Region 1)
sns.countplot(ax=axes[0], x='month', hue='Classes',
data=dftemp_region_1, palette="viridis")
axes[0].set_title("Fire Analysis of Sidi-Bel Abbes Region",
weight='bold')
axes[0].set_xlabel("Months", weight='bold')
axes[0].set_ylabel("Number of Fires", weight='bold')

# Bejaia Region (Region 0)
sns.countplot(ax=axes[1], x='month', hue='Classes',
data=dftemp_region_0, palette="viridis")
axes[1].set_title("Fire Analysis of Bejaia Region", weight='bold')
axes[1].set_xlabel("Months", weight='bold')
axes[1].set_ylabel("")

# Adjust layout
plt.tight_layout()
plt.show()

df_cleaned.head()
```

```
# Function to plot FWI values over time with fire and no-fire
indicators
def plot_fwi_with_fire_and_no_fire(region_data, region_name):
    plt.figure(figsize=(14, 6))

    # Plot FWI values over time
    plt.plot(region_data.index, region_data['FWI'], label='FWI (Line)',
color='blue', linewidth=1.5)

    # Highlight days with fire (Class == 1) as red points
    fire_days = region_data[region_data['Classes'] == 1]
    plt.scatter(fire_days.index, fire_days['FWI'], color='red',
label='Fire (Class=1)', zorder=5)

    # Highlight days with no fire (Class == 0) as blue points
    no_fire_days = region_data[region_data['Classes'] == 0]
    plt.scatter(no_fire_days.index, no_fire_days['FWI'], color='blue',
label='No Fire (Class=0)', zorder=5)

    # Formatting the plot
    plt.title(f"FWI Over Time for {region_name}", weight='bold')
    plt.xlabel("Time (Days)", weight='bold')
    plt.ylabel("FWI", weight='bold')
    plt.legend()
    plt.grid(alpha=0.5)
    plt.tight_layout()
    plt.show()

# Plot for Region 1 (Sidi-Bel Abbes)
plot_fwi_with_fire_and_no_fire(dftemp_region_0, "Sidi-Bel Abbes
Region")

# Plot for Region 0 (Bejaia)
plot_fwi_with_fire_and_no_fire(dftemp_region_1, "Bejaia Region")
```

Preprocess

```
from sklearn.model_selection import train_test_split

# Split for Region 0
train_region_0, test_region_0 = train_test_split(dftemp_region_0,
test_size=0.2, shuffle=False)

# Split for Region 1
train_region_1, test_region_1 = train_test_split(dftemp_region_1,
test_size=0.2, shuffle=False)

# Combine train and test data from both regions
```

```
train_combined = pd.concat([train_region_0,train_region_1],
ignore_index=True)
test_combined = pd.concat([test_region_0,test_region_1],
ignore_index=True)

plot_fwi_with_fire_and_no_fire(train_combined, "Sidi-Bel Abbes and
Bejaia Region Region Train")
plot_fwi_with_fire_and_no_fire(test_combined, "Sidi-Bel Abbes and
Bejaia Region Region Test")

## Scale

train = train_combined.drop(columns=['day','month','year'])
test = test_combined.drop(columns=['day','month','year'])

# Temporarily store the 'FWI' column and remove it from train and test
fwi_train = train_combined['FWI']
train = train_combined.drop(columns=['FWI'])
train['FWI'] = fwi_train  # Add 'FWI' back at the end

fwi_test = test_combined['FWI']
test = test_combined.drop(columns=['FWI'])
test['FWI'] = fwi_test  # Add 'FWI' back at the end

train.head()

# Initialize scalers for features and target
feature_scaler = StandardScaler()
target_scaler = StandardScaler()

# Scale features
X_train_scaled =
feature_scaler.fit_transform(train.drop(columns=['FWI']))
X_test_scaled = feature_scaler.transform(test.drop(columns=['FWI']))

# Scale target
y_train_scaled = target_scaler.fit_transform(train[['FWI']])
y_test_scaled = target_scaler.transform(test[['FWI']])

print("X_train_scaled Shape:",X_train_scaled.shape)
print("X_test_scaled Shape:",X_test_scaled.shape)
print("y_train_scaled Shape:",y_train_scaled.shape)
print("y_test_scaled Shape:",y_test_scaled.shape)

# Reshape y_train_scaled and y_test_scaled to 1D arrays for
compatibility
y_train_scaled_ = y_train_scaled.reshape(-1)
y_test_scaled_ = y_test_scaled.reshape(-1)
```

```
## Data Windowing

def create_windows_with_targets(data, targets, window_size, step_size):
    """
    Create sliding windows for time-series data along with their
corresponding target values.
    :param data: Numpy array of features (X).
    :param targets: Numpy array of target values (y).
    :param window_size: Number of time steps in each window.
    :param step_size: Step size between windows.
    :return: Sliding windows (X_windows) and corresponding targets
(y_targets).
    """
    X_windows = []
    y_targets = []
    for i in range(0, len(data) - window_size + 1, step_size):
        X_windows.append(data[i:i + window_size])  # Extract the window
        y_targets.append(targets[i + window_size - 1])  # Target
corresponds to the last time step in the window
    return np.array(X_windows), np.array(y_targets)

# Define window and step size
window_size = 3
step_size = 1

# Perform windowing on the scaled train data
X_train_windows, y_train_windows =
create_windows_with_targets(X_train_scaled, y_train_scaled,
window_size, step_size)
X_test_windows, y_test_windows =
create_windows_with_targets(X_test_scaled, y_test_scaled, window_size,
step_size)

# Display shapes of the resulting windows and targets
print("Train Shapes:",X_train_windows.shape, y_train_windows.shape)
print("Test Shapes:",X_test_windows.shape, y_test_windows.shape)

# Reshape y_train and y_test to 1D arrays for compatibility with
scikit-learn
y_train_reshaped = y_train_windows.reshape(-1)
y_test_reshaped = y_test_windows.reshape(-1)

# Flatten X_train and X_test to 2D arrays for compatibility with
logistic regression
X_train_flattened = X_train_windows.reshape(X_train_windows.shape[0], -
1)
X_test_flattened = X_test_windows.reshape(X_test_windows.shape[0], -1)

print(f"x_train_reshaped {X_train_flattened.shape}")
```

```
print(f"y_train_reshaped {y_train_reshaped.shape}")
print(f"x_test_reshaped {X_test_flattened.shape}")
print(f"y_test_reshaped {y_test_reshaped.shape}")

y_train_reshaped.shape
```

Modelling Ridge Regression

```
##Base Model

# Ridge Regressor'ı başlatma
ridge_model_base = Ridge()

# Modeli ölçeklendirilmiş eğitim verisi üzerinde eğitme
ridge_model_base.fit(X_train_scaled, y_train_scaled_)

# Test verisi üzerinde tahmin yapma
y_test_pred = ridge_model_base.predict(X_test_scaled)

# Model performansını değerlendirme
test_mse = mean_squared_error(y_test_scaled_, y_test_pred)
test_mae = mean_absolute_error(y_test_scaled_, y_test_pred)
test_r2 = r2_score(y_test_scaled_, y_test_pred)

print(f"Test MSE: {test_mse}")
print(f"Test MAE: {test_mae}")
print(f"Test R2 Score: {test_r2}")

# Performans metriklerini tuple olarak döndürme
(test_mse, test_mae, test_r2)

# Hyper Parameter Optimization

# Ridge Regression modelini başlat
ridge = Ridge()
# Hiperparametre aralığını tanımla
param_grid = {
    'alpha': [0.1, 1, 10, 100, 1000],  # L2 regularization strength
}

# GridSearchCV ile hiperparametre optimizasyonu
grid_search = GridSearchCV(estimator=ridge, param_grid=param_grid,
cv=5, scoring='r2')
grid_search.fit(X_train_scaled, y_train_scaled_)

# En iyi modeli seç
best_params = grid_search.best_params_
best_params
```

```
ridge_best_model = Ridge(**best_params)
ridge_best_model.fit(X_train_scaled, y_train_scaled_)

# Tahmin yap ve performansı değerlendir
y_test_pred = ridge_best_model.predict(X_test_scaled)

# Performans değerlendirmesi
test_mse = mean_squared_error(y_test_scaled_, y_test_pred)
test_mae = mean_absolute_error(y_test_scaled_, y_test_pred)
test_r2 = r2_score(y_test_scaled_, y_test_pred)

# Sonuçları göster
{
    "Best Parameters": best_params,
    "Test MSE": test_mse,
    "Test Mae": test_mae,
    "Test R2": test_r2
}

# Windowing Technique

# Ridge Regression modelini başlat
ridge_model_window = Ridge()

# Ridge Regression modeli eğit
ridge_model_window.fit(X_train_flattened, y_train_windows)

# Tahmin yap
y_train_pred = ridge_model_window.predict(X_train_flattened)
y_test_pred = ridge_model_window.predict(X_test_flattened)

# Performans değerlendirmesi
train_mse = mean_squared_error(y_train_windows, y_train_pred)
test_mse = mean_squared_error(y_test_windows, y_test_pred)
train_r2 = r2_score(y_train_windows, y_train_pred)
test_r2 = r2_score(y_test_windows, y_test_pred)

# Sonuçları göster
print("Train MSE:", train_mse)
print("Train R2:", train_r2)
print("Test MSE:", test_mse)
print("Test R2:", test_r2)

#HyperParameter Optimization

# Define the parameter grid
param_grid = {
```

```
    'alpha': [0.1, 1, 10, 50, 100, 200, 500, 1000]  # Example range of
alpha values
}

# Define a Ridge model
ridge_model = Ridge()


# Set up GridSearchCV
grid_search = GridSearchCV(
    estimator=ridge_model,
    param_grid=param_grid,
    scoring='r2',
    cv=5,  # Cross-validation folds
    verbose=1,
    n_jobs=-1
)

# Flatten the training data if necessary
# Fit the model
grid_search.fit(X_train_flattened, y_train_windows)

# Get the best parameters and the best estimator
best_params_ridge_window = grid_search.best_params_
best_model_ridge_window = Ridge(**best_params_ridge_window)
best_model_ridge_window.fit(X_train_flattened, y_train_windows)

# Print the best parameters
print("Best parameters:", best_params)

# Evaluate the model on the test set
y_test_pred = best_model_ridge_window.predict(X_test_flattened)
test_mse = mean_squared_error(y_test_windows, y_test_pred)
test_r2 = r2_score(y_test_windows, y_test_pred)

print("Test MSE with best model:", test_mse)
print("Test R2 with best model:", test_r2)
```

Modelling LSTM with Windowed Data

```
# Building Simple LSTM Model

import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

class LSTMRegressionModel:
```

```
    def __init__(self, input_shape, lstm_units=64, dense_units=32,
optimizer='adam', loss='mse', metrics=['mae']):
        """
        Initializes the LSTM Regression Model.

        Parameters:
        - input_shape: Tuple, shape of the input data (time_steps,
features).
        - lstm_units: Integer, number of LSTM units.
        - dense_units: Integer, number of Dense units.
        - optimizer: String, optimizer for the model.
        - loss: String, loss function.
        - metrics: List of metrics to monitor during training.
        """
        self.model = Sequential([
            LSTM(lstm_units, activation='relu',
input_shape=input_shape),
            Dense(dense_units, activation='relu'),
            Dense(1)  # Output layer
        ])

        self.model.compile(optimizer=optimizer, loss=loss,
metrics=metrics)

    def train(self, X_train, y_train, X_valid, y_valid, epochs=50,
batch_size=16):
        """
        Trains the LSTM model.

        Parameters:
        - X_train: Training features.
        - y_train: Training targets.
        - X_valid: Validation features.
        - y_valid: Validation targets.
        - epochs: Integer, number of epochs.
        - batch_size: Integer, batch size for training.

        Returns:
        - history: Training history object.
        """
        history = self.model.fit(
            X_train, y_train,
            epochs=epochs,
            batch_size=batch_size,
            validation_data=(X_valid, y_valid)
        )
        return history

    def evaluate(self, X_test, y_test):
```

```
        """
        Evaluates the LSTM model on test data.

        Parameters:
        - X_test: Test features.
        - y_test: Test targets.

        Returns:
        - loss: Loss value on the test data.
        - metrics: List of metric values on the test data.
        """
        loss, mae = self.model.evaluate(X_test, y_test)
        print("Test Loss (MSE):", loss)
        print("Test MAE:", mae)
        return loss, mae

#Validation

# 80-20 train-validation split
X_train, X_valid, y_train, y_valid = train_test_split(
    X_train_windows, y_train_windows, test_size=0.2, shuffle=True,
random_state=42
)

# Initialize the model
input_shape = (X_train.shape[1], X_train.shape[2])
lstm_model = LSTMRegressionModel(input_shape=input_shape)

# Train the model
history = lstm_model.train(X_train, y_train, X_valid, y_valid,
epochs=50, batch_size=16)

# Evaluate the model
loss, mae = lstm_model.evaluate(X_test_windows, y_test_windows)

# Eğitim geçmişinden değerleri çekme
train_loss = history.history['loss']
val_loss = history.history['val_loss']
train_mae = history.history['mae']
val_mae = history.history['val_mae']

# Loss grafiği
plt.figure(figsize=(14, 5))

plt.subplot(1, 2, 1)  # İlk grafik
plt.plot(train_loss, label='Train Loss (MSE)')
plt.plot(val_loss, label='Validation Loss (MSE)')
plt.xlabel('Epochs')
plt.ylabel('Loss (MSE)')
```

```
plt.title('Loss Over Epochs')
plt.legend()

# MAE grafiği
plt.subplot(1, 2, 2)  # İkinci grafik
plt.plot(train_mae, label='Train MAE')
plt.plot(val_mae, label='Validation MAE')
plt.xlabel('Epochs')
plt.ylabel('Mean Absolute Error (MAE)')
plt.title('MAE Over Epochs')
plt.legend()

plt.tight_layout()
plt.show()

#Evaluation

history_2 = lstm_model.model.fit(X_train_windows, y_train_windows,
epochs=15, batch_size=16)

#Final

# Predictions on validation set
y_valid_pred = lstm_model.model.predict(X_test_windows)

# Compute validation MAPE, MAE, and MSE
mae = np.mean(np.abs(y_test_windows - y_valid_pred))
mse = np.mean((y_test_windows - y_valid_pred) ** 2)
r2 = r2_score(y_test_windows, y_valid_pred)

# Print metrics
print("Validation MAE:", mae)
print("Validation MSE:", mse)
print("Validation R2:", r2)

from sklearn.model_selection import KFold
import numpy as np

# K-Fold Cross-Validation
k = 5  # Number of folds
kf = KFold(n_splits=k, shuffle=True, random_state=42)

fold_no = 1
results = []

for train_index, valid_index in kf.split(X_train_windows):
    print(f"Training on Fold {fold_no}...")

    # Train-validation split
```

```
    X_train_fold, X_valid_fold = X_train_windows[train_index],
X_train_windows[valid_index]
    y_train_fold, y_valid_fold = y_train_windows[train_index],
y_train_windows[valid_index]

    # Model initialization
    input_shape = (X_train_fold.shape[1], X_train_fold.shape[2])
    lstm_model = LSTMRegressionModel(input_shape=input_shape)

    # Model training
    history = lstm_model.train(X_train_fold, y_train_fold,
X_valid_fold, y_valid_fold, epochs=50, batch_size=16)

    # Evaluate on validation set
    val_loss, val_mae = lstm_model.evaluate(X_valid_fold, y_valid_fold)
    print(f"Fold {fold_no} - Validation Loss: {val_loss}, Validation
MAE: {val_mae}")

    results.append((val_loss, val_mae))
    fold_no += 1

# Average results across folds
avg_loss = np.mean([r[0] for r in results])
avg_mae = np.mean([r[1] for r in results])

print(f"\nK-Fold Cross-Validation Results:")
print(f"Average Validation Loss: {avg_loss}")
print(f"Average Validation MAE: {avg_mae}")
```