MUNZUR ÜNİVERSİTESİ 2008

Sağlık Sigortası Maliyet Tahmini

Bu veri seti, bireylerin sağlık sigortası maliyetlerini tahmin etmek için kullanılmış ve iki farklı makine öğrenimi algoritmasıyla değerlendirilmiştir ; Linear Regression (Doğrusal Regresyon) ve Random Forest (Rastgele Orman). Amaç, sigorta maliyetlerini etkileyen faktörlerin (örneğin, yaş, cinsiyet, vücut kitle indeksi, sigara kullanımı, çocuk sayısı ve bölge) sağlık sigortası ücretleri üzerindeki etkisini analiz ederek bireysel tıbbi maliyetleri tahmin etmektir. Bu analiz, sağlık sigortası maliyetlerini etkileyen faktörlerin detaylı bir şekilde anlaşılmasını ve gelecekteki maliyetlerin daha iyi tahmin edilmesini sağlamaktadır.

Elde edilen modeller:

1. Sigorta şirketleri tarafından maliyet analizi ve fiyatlandırma stratejileri oluşturulmasında,
2. Sağlık politikalarının belirlenmesinde,
3. Kişisel maliyet öngörülerinin geliştirilmesinde kullanılabilir.

Bu dokümantasyon, uygulanan modellerin performansını ve sağlık sigortası maliyetleriyle ilişkili faktörlerin analizine yönelik değerli bilgiler sunmaktadır.

Hazırlayan: Medeni Aba, Ahmet Can İzgi

Veri Setine Genel Bakış

Bu veri seti, ABD'deki bireylerin sağlık sigortası maliyetleriyle ilgili demografik, yaşam tarzı ve sağlık bilgilerini içermektedir. Toplamda 1338 satır ve 7 sütun bulunmaktadır. Veri seti, bireylerin yaşları, cinsiyetleri, sigara kullanımları, çocuk sayıları, yaşadıkları bölge ve sağlık sigortası maliyetleri gibi detayları içermektedir.

Veri Seti Sütunları

1. **Yaş** : Bireyin yaşı (Tam sayı).
2. **Cinsiyet** : Sigorta sahibinin cinsiyeti (Kadın/Erkek).
3. **Vücut Kitle İndeksi** : Bireyin vücut kitle indeksi (kg/m²). 18.5 ile 24.9 arası ideal kabul edilir.
4. **Çocuk Sayısı** : Sağlık sigortası kapsamındaki çocuk sayısı (Tam sayı).
5. **Sigara Kullanımı** : Bireyin sigara kullanıp kullanmadığı (Evet/Hayır).
6. **Bölge** : Sigorta sahibinin ABD'deki yaşadığı bölge (Kuzeybatı, Güneydoğu, Güneybatı, Kuzeydoğu).
7. **Sigorta Ücretleri** : Sağlık sigortası tarafından faturalandırılan bireysel tıbbi maliyet (Ondalıklı sayı).

Sigara Kullanımı Sütunu Hakkında Ayrıntılar:

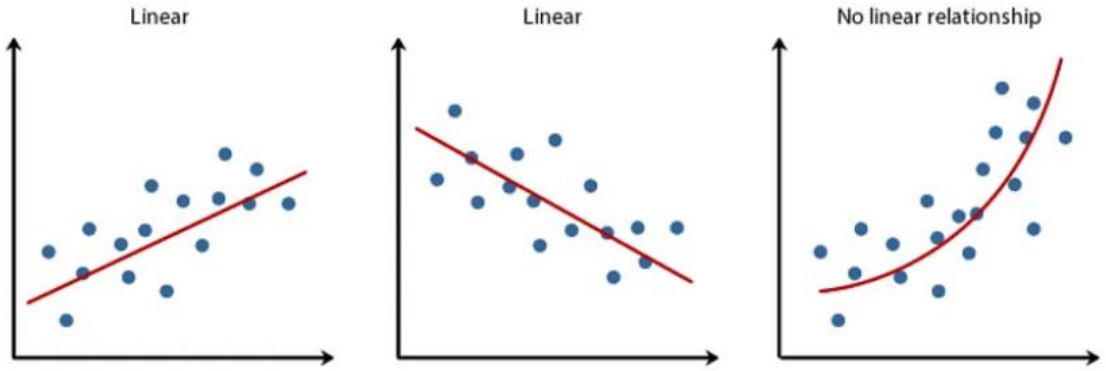
- **Evet** : Bireyin sigara içtiğini gösterir. Genellikle sağlık sigortası maliyetlerinde artışa yol açar.
- **Hayır** : Bireyin sigara içmediğini gösterir.

Bu veri seti, sağlık sigortası maliyetlerini etkileyen çeşitli faktörleri analiz etmek ve bu faktörlerin sağlık giderleri üzerindeki etkilerini incelemek için uygundur.

Kullanılan Algoritmalar

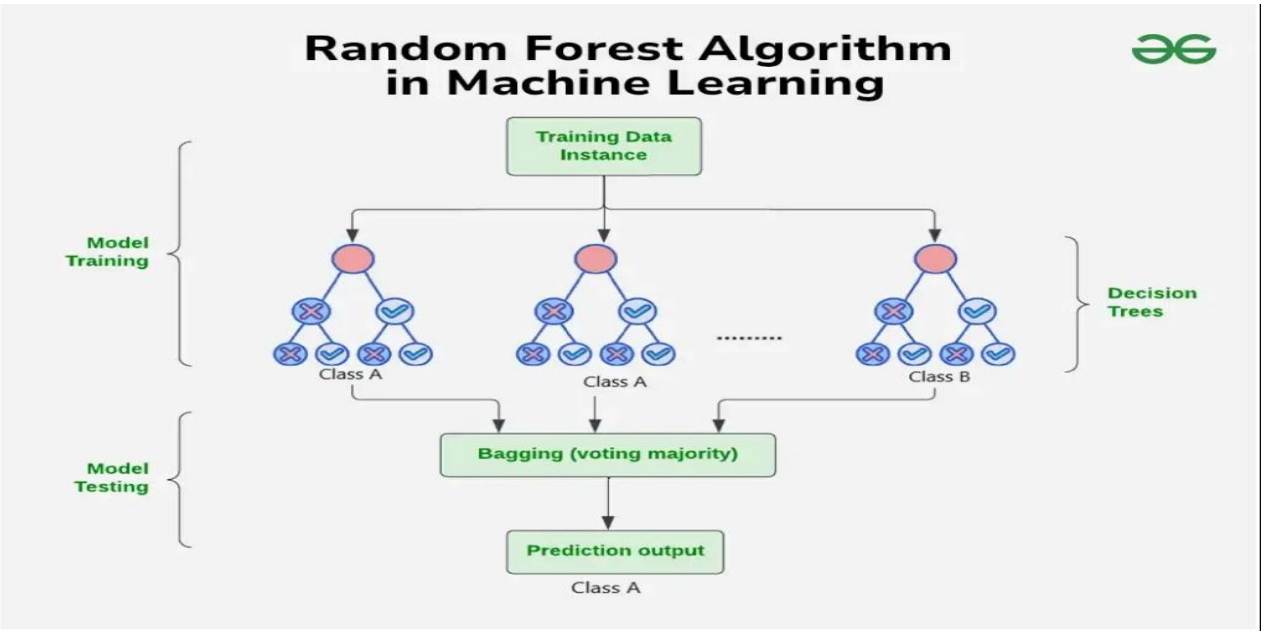
Lineer Regresyon

Lineer Regresyon, bir bağımlı değişken ile bir veya daha fazla bağımsız değişken arasındaki ilişkiyi modellemek için kullanılan basit bir istatistiksel regresyon yöntemidir. Bu yöntem, doğrusal bir ilişkiyi ifade eden bir denklem oluşturarak, bağımsız değişkenlerin değerleri ile bağımlı değişken arasındaki ilişkiyi açıklamaya çalışır.



Random Forest

Random Forest, birçok karar ağacının bir araya gelerek oluşturduğu bir ansambil (ensemble) öğrenme algoritmasıdır. Her bir karar ağacı, rastgele örneklem alınmış veri alt kümesi üzerinde eğitilir ve bu ağaçlar rastgele özelliklerle (değişkenlerle) oluşturulur.



Özellik	Random Forest	Lineer Regresyon
Temel Fikir	Birçok karar ağacının bir araya gelmesiyle oluşturulan bir ansambl öğrenme yöntemi. Her bir karar ağacı, veri kümesinin farklı bir alt kümesi üzerinde eğitilir ve farklı özellikler seçilir.	Bağımlı değişken ile bir veya daha fazla bağımsız değişken arasındaki doğrusal ilişkiyi modellemek.
Model Karmaşıklığı	Genellikle daha karmaşık modeller oluşturabilir. Çok sayıda etkileşim ve doğrusal olmayan ilişkileri yakalayabilir.	Daha basit modellerdir. Sadece doğrusal ilişkileri modelleyebilir.
Veri Tipi	Hem sayısal hem de kategorik verilerle çalışabilir.	Genellikle sayısal verilerle çalışır. Kategorik veriler dönüştürülmelidir.
Aşırı Uyum (Overfitting)	Daha az eğilimlidir. Birden fazla ağaç kullanarak aşırı uyum riskini azaltır.	Daha fazla eğilimlidir. Özellikle veri kümesi küçükse veya çok fazla özellik varsa.
Eğitim Süresi	Genellikle daha uzun sürer. Birden fazla ağaç eğitmek gerekir.	Daha hızlıdır. Tek bir doğrusal denklem çözülür.
Tahmin Süresi	Daha yavaştır. Birden fazla ağaçta tahmin yapmak gerekir.	Daha hızlıdır. Tek bir doğrusal denklem kullanılır.
Özellik Önemliliği	Her bir özelliğin modelde ne kadar önemli olduğunu gösteren bir metrik sağlar.	Özelliklerin önemini belirlemek için ek analizler yapmak gerekebilir.
Aykırı Değerlere Duyarlılık	Daha az duyarlıdır. Birçok ağacın ortalaması alındığı için aykırı değerlerin etkisi azalır.	Daha duyarlıdır. Aykırı değerler modeli önemli ölçüde etkileyebilir.
İnteraksiyonlar	Özellikler arasındaki etkileşimleri yakalayabilir.	Sadece doğrusal etkileşimleri yakalayabilir.
Kullanım Alanları	Sınıflandırma, regresyon, özellik seçimi gibi birçok alanda kullanılır.	Genellikle regresyon problemlerinde kullanılır.

Kullanılan Yöntemler

Aykırı Değer Tespitinde Kullanılan Yöntemler

Aykırı değer tespiti, bir veri setindeki diğer gözlemlerden önemli ölçüde farklı olan değerleri belirleme işlemidir. IQR (Interquartile Range) yöntemi, aykırı değerleri tespit etmek için yaygın olarak kullanılan istatistiksel bir tekniktir. IQR, bir veri setinin orta %50'lik kısmını temsil eder ve çeyrekler arası aralık olarak bilinir.

IQR Nedir?

IQR, bir veri setinin dağılımını ölçen bir metriktir ve şu şekilde hesaplanır:

$IQR = Q3 - Q1$

- Q1 (Birinci Çeyrek): Verilerin %25'inin altında olduğu değerdir (alt çeyrek).
- Q3 (Üçüncü Çeyrek): Verilerin %75'inin altında olduğu değerdir (üst çeyrek).
- IQR: Verilerin ortadaki %50'lik kısmının genişliğini temsil eder.

veri setindeki aykırı değerleri tespit etmek ve temizlemek için güvenilir bir yöntemdir. Ancak, aykırı değerlerin her zaman "hatalı" olduğunu varsaymadan, veri analizi bağlamına göre değerlendirilmesi gerekir.

Z-Skoru

Z-skoru (Z-score), bir veri noktasının, bir dağılımın ortalamasından kaç standart sapma uzaklıkta olduğunu gösteren bir ölçüdür. Veri analizi ve istatistikte yaygın olarak kullanılan bir yöntemdir ve özellikle **standart normal dağılım** ile ilgili hesaplamalarda kullanılır.

Z-skoru Formülü:

$$Z = \frac{X - \mu}{\sigma}$$

Burada:

- Z: Z-skoru (standartlaştırılmış değer)
- X: İncelenen veri noktası
- μ: Veri setinin ortalaması
- σ: Veri setinin standart sapması

Z-skorunun Kullanım Alanları:

1. **Aykırı Değer Tespiti:**
    - Z-skoru belirli bir eşik değeri (genelde $Z>3$ veya $Z<-3$) aşarsa, o veri noktası **aykırı değer** olarak kabul edilir.
2. **Veri Normalizasyonu:**
    - Z-skoru, farklı ölçekteki veri setlerini standardize etmek için kullanılır. Bu, özellikle **makine öğrenmesi** ve istatistiksel modelleme gibi uygulamalarda faydalıdır.
3. **Olasılık Hesaplamaları:**
    - Z-skoru, standart normal dağılım tablosu kullanılarak belirli bir veri noktasının olasılığını hesaplamak için kullanılır.
4. **Karşılaştırmalar:**
    - Farklı ölçekteki verileri doğrudan karşılaştırmak için Z-skoru kullanılır.

Z-skoru, bir veri noktası ile dağılımın geri kalanı arasındaki farkı standartlaştırılmış bir şekilde ifade eder. Bu, veri analizi ve istatistiksel modellemede güçlü bir araçtır.

Isolation Forest (İzolasyon Ormanı)

Isolation Forest (İzolasyon Ormanı), anormal veya aykırı değerleri tespit etmek için kullanılan bir denetimsiz makine öğrenmesi algoritmasıdır. Temel fikri, aykırı değerlerin veri kümesinde daha kolay izole edilebilmesidir, çünkü aykırı değerler genellikle diğer verilere kıyasla daha seyrek bulunur ve farklı özellikler taşır. Bu yöntem özellikle yüksek boyutlu verilerde etkili ve hesaplama açısından verimli bir algoritmadır.

Nasıl Çalışır?

1. **Rastgele Alt Küme Oluşturma:**
    - Algoritma, veri kümesinden rastgele alt küme seçer ve bu alt kümelerde çalışır.
2. **Rastgele Ayrıştırma (Partitioning):**
    - Veriler, rastgele seçilen bir özellik ve bu özelliğin rastgele bir değeri kullanılarak bölünür.
    - Bölme işlemi, verileri belirli bir sınırdan (threshold) ayırarak bir **ikili ağaç (binary tree)** oluşturur.
3. **İzolasyon Yolu (Path Length):**
    - Ağaç yapısı oluşturulduktan sonra her veri noktası için kökten yaprağa kadar olan yolun uzunluğu hesaplanır.
    - Aykırı değerler, genellikle daha az bölünmeyle (kısa yol) izole edilebilir. Çünkü, aykırı değerler diğer veri noktalarından daha uzakta yer alır.
4. **Skor Hesaplama:**
    - İzolasyon yolları kullanılarak her bir veri noktasına bir **anormallik skoru** atanır.
    - Skor yüksekse (örneğin, score>0.5), veri noktası bir aykırı değer olarak değerlendirilir.

Avantajları:

Verimli**:** Çalışma süresi O(n logn) olduğu için, büyük veri setlerinde hızlı çalışır.

Yüksek Boyutlu Veriler**:** Yüksek boyutlu veri kümeleri üzerinde iyi performans gösterir.

Özelleştirilmiş Aykırılık Skoru**:** Her veri noktası için bir skor ürettiğinden, aykırı değerlerin derecesini belirlemek kolaydır.

Dezavantajları:

Rastgelelik: Rastgele ağaç oluşturma işlemi nedeniyle sonuçlar her çalıştırmada biraz farklı olabilir.

Karmaşık Dağılımlar: Karmaşık ve yoğun veri kümelerinde performansı düşebilir.

Dengeli Aykırılık: Veri setindeki aykırı değerlerin dağılımı çok dengesizse performans azalabilir.

Örnek:

Bir veri kümesi düşünün: [10,12,13,14,15,16,1000]

- Normal değerler yoğun olarak 10 ile 16 arasında.
- 1000, diğer değerlerden oldukça uzakta ve aykırı olarak tespit edilir.

Isolation Forest, 1000 değerini az sayıda bölünme ile izole eder ve buna yüksek bir anormallik skoru atar.

Local Outlier (Yerel Aykırı Değer)

Local Outlier , bir veri noktasının çevresindeki komşu veri noktalarına göre anormal bir davranış sergilemesi durumudur. Bu yaklaşım, global (küresel) dağılımdan ziyade, veri noktasının bulunduğu yerel çevreye odaklanır.

LOF'un Çalışma Adımları:

- Her veri noktası için k-en yakın komşuları belirlenir. Bu, veri noktalarının birbirine olan mesafeleri kullanılarak yapılır.
- Bir veri noktasının, komşularına olan ortalama erişim mesafesi hesaplanır.
- Eğer bir veri noktası komşularına çok uzaksa, bu mesafe yüksek olacaktır.
- LOF, bir veri noktasının yerel yoğunluğunu komşularının yerel yoğunluğuna kıyaslayarak hesaplar.

$$LOF(p) = \frac{\text{Komşuların yoğunluğu ortalaması}}{\text{Veri noktasının yoğunluğu}}$$

LOF(p)>1: Aykırı değer olma ihtimali yüksek.

LOF(p)≈1: Normal bir veri noktası.

LOF(p)<1: Yoğunluğu komşularından daha yüksek olan noktalar.

Random Search

Random Search, makine öğrenmesi modellerinde hiper parametre optimizasyonu için kullanılan bir tekniktir. Hiper parametre optimizasyonu, model performansını artırmak için en uygun hiper parametrelerin aranması sürecidir. Random Search, bu aramayı rastgele hiper parametre kombinasyonlarını deneyerek gerçekleştirir.

Nasıl Çalışır?

1. **Hiper parametre Alanı Belirlenir:** Optimizasyon yapılacak hiper parametrelerin aralıkları veya olası değerleri tanımlanır.

2. **Rastgele Seçim:**
   - Hiper parametre alanından rastgele bir kombinasyon seçilir.

3. **Model Eğitimi:**
   - Seçilen hiper parametre kombinasyonu ile model eğitilir ve performansı değerlendirilir.

4. **Tekrar:**
   - Belirlenen sayıda rastgele kombinasyon denenir.

5. **En İyisini Seçme:**
   - En iyi performansı sağlayan hiper parametre kombinasyonu seçilir.

Random Search ile Grid Search Karşılaştırması

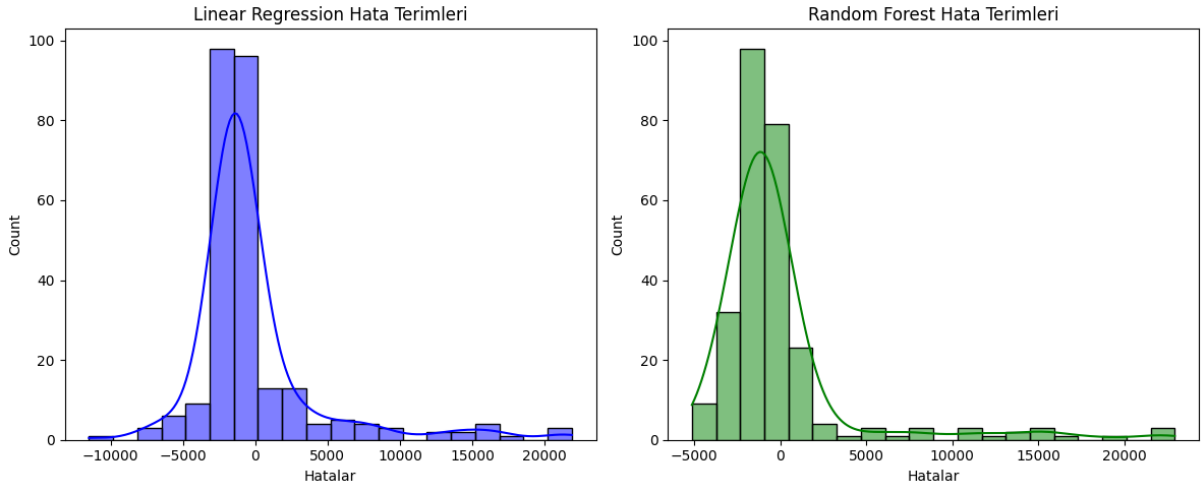
Özellik	Grid Search	Random Search
Arama Yöntemi	Tüm olası hiperparametre kombinasyonlarını sistematik olarak deniyor.	Hiperparametre uzayında rastgele noktalar seçerek değerlendiriyor.
Hesaplama Maliyeti	Yüksek. Özellikle hiperparametre sayısı arttıkça hesaplama süresi önemli ölçüde artar.	Genellikle Grid Search'e göre daha düşük. Ancak, şans faktörü nedeniyle optimum değere ulaşmamak riski vardır.
Optimum Değere Ulaşma Olasılığı	Tüm olası kombinasyonları denediği için teorik olarak en iyi değeri bulma olasılığı daha yüksek.	Tüm olası kombinasyonları denemediği için en iyi değeri bulma olasılığı daha düşüktür. Ancak, iyi bir başlangıç noktası sağlayabilir.
Kullanım Kolaylığı	Uygulaması nispeten kolaydır.	Uygulaması kolaydır.
Verimlilik	Düşük verimlidir. Özellikle yüksek boyutlu hiperparametre uzaylarında zaman alıcı olabilir.	Daha yüksek verimlidir. Özellikle yüksek boyutlu uzaylarda Grid Search'e göre daha hızlı sonuçlar verir.
Ne Zaman Kullanılır?	Tüm olası kombinasyonları denemek istendiğinde veya hesaplama kaynakları sınırlı değilse.	Hızlı bir şekilde iyi bir sonuç elde etmek istendiğinde veya yüksek boyutlu hiperparametre uzaylarında çalışıldığında.

Kullanım Alanları:

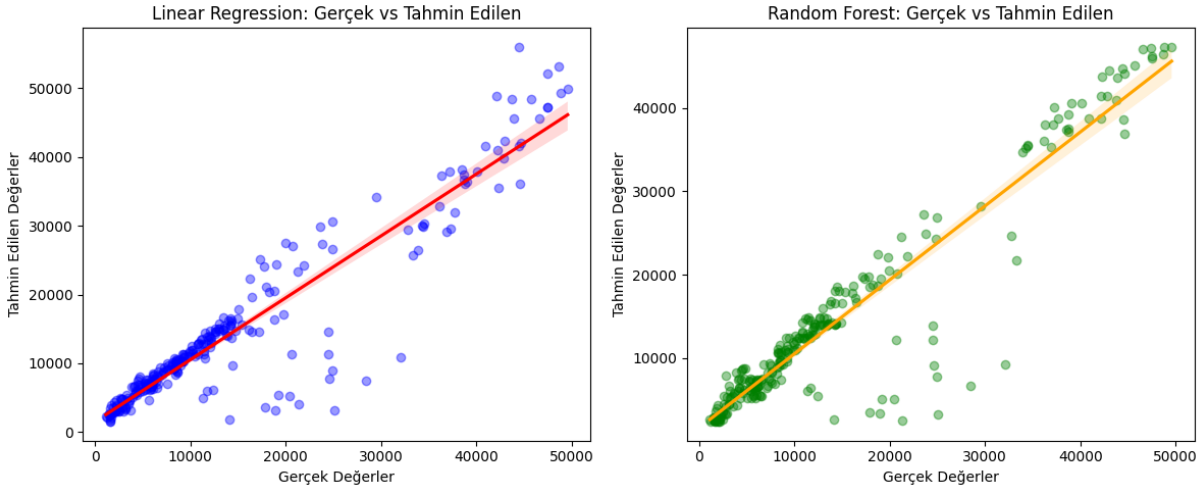
- Makine Öğrenmesi Modelleri: Hiperparametre optimizasyonu (örneğin, Random Forest, XGBoost).
- Derin Öğrenme Modelleri: Sinir ağı hiperparametrelerini optimize etmek.
- Genel Optimizasyon Problemleri: Karmaşık optimizasyon süreçlerinde hızlı çözümler.

Metrik	Anlamı	Değer Aralığı	Ne İfade Eder	Kullanım Alanları
RMSE	Kök Ortalama Kare Hata	0 - ∞	Tahmin hatalarının ortalama büyüklüğü	Regresyon, Sınıflandırma
R-Kare	Belirleme Katsayısı	0 - 1	Modelin verileri açıklama gücü	Regresyon
Açıklanan Varyans	Belirleme Katsayısı	0 - 1	Modelin verileri açıklama gücü	Regresyon

Grafikler



Şekil 1.0



Şekil 1.1

Kütüphaneleri Ekleme

```
[ ] import numpy as np
    import pandas as pd
    import seaborn as sns
    import matplotlib.pyplot as plt
    from sklearn.preprocessing import StandardScaler, LabelEncoder, PolynomialFeatures
    from sklearn.linear_model import LinearRegression
    from sklearn.ensemble import RandomForestRegressor, IsolationForest
    from sklearn.model_selection import train_test_split, RandomizedSearchCV, cross_val_score
    from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score, explained_variance_score
    from sklearn.neighbors import LocalOutlierFactor
```

Veri Setini Okuma İlk 5 Satını Görüntüleme

```
[ ]
    df = pd.read_csv('/content/insurance.csv' , encoding='cp1254')
    df.head()  # Verinin ilk 5 satırını görüntüler
```

	yaş	cinsiyet	vucut_kitle_indeksi	cocuk_sayisi	sigara_kullanimi	bölge	sigorta_ücretleri
0	19	kadın	27.900	0	evet	güneybatı	16884.92400
1	18	erkek	33.770	1	hayır	güneydogu	1725.55230
2	28	erkek	33.000	3	hayır	güneydogu	4449.46200
3	33	erkek	22.705	0	hayır	kuzeybatı	21984.47061
4	32	erkek	28.880	0	hayır	kuzeybatı	3866.85520

Unique değerlerin bulunması

```
[ ] # Kolon bilgilerini saklamak için boş listeler oluşturuluyor
    kolonlar = []
    benzersiz_sayisi = []
    benzersiz_degerler = []
    kolon_tipi = []

    # DataFrame'in her bir kolonu için döngü başlatılıyor
    for kolon in df.columns:
        kolonlar.append(kolon)  # Kolon adını listeye ekle
        benzersiz_sayisi.append(df[kolon].nunique())  # Kolonun benzersiz değer sayısını ekle
        benzersiz_degerler.append(df[kolon].unique())  # Kolonun benzersiz değerlerini ekle
        kolon_tipi.append(df[kolon].dtype)  # Kolonun veri tipini ekle

    # Elde edilen bilgileri bir DataFrame'e dönüştür
    kolon_dict = {'Kolon': kolonlar, 'Benzersiz Değer Sayısı': benzersiz_sayisi, 'Benzersiz Değerler': benzersiz_degerler, 'Veri Tipi': kolon_tipi}
    pd.DataFrame(kolon_dict)
```

	Kolon	Benzersiz Değer Sayısı	Benzersiz Değerler	Veri Tipi
0	yaş	47	[19, 18, 28, 33, 32, 31, 46, 37, 60, 25, 62, 2...	int64
1	cinsiyet	2	[kadın, erkek]	object
2	vucut_kitle_indeksi	548	[27.9, 33.77, 33.0, 22.705, 28.88, 25.74, 33.4...	float64
3	cocuk_sayisi	6	[0, 1, 3, 2, 5, 4]	int64
4	sigara_kullanimi	2	[evet, hayır]	object
5	bölge	4	[güneybatı, güneydogu, kuzeybatı, kuzeydogu]	object
6	sigorta_ücretleri	1337	[16884.924, 1725.5523, 4449.462, 21984.47061, ...	float64

```
[ ] # Kategorik ve sayısal kolonları ayırma

    # Kategorik veri tipindeki kolonları seç
    kategorik_kolonlar = df.select_dtypes(include=['object', 'category']).columns.tolist()

    # Sayısal veri tipindeki kolonları seç
    sayisal_kolonlar = df.select_dtypes(exclude=['object', 'category']).columns.tolist()
```

Aykırı Değer Tespiti (IQR)

```
[ ]  # IQR yöntemiyle outlier (aykırı değer) tespiti

     # 'charges' kolonunun 1. çeyrek (Q1) ve 3. çeyrek (Q3) değerlerini hesapla
     Q1 = df['sigorta_ücretleri'].quantile(0.25)
     Q3 = df['sigorta_ücretleri'].quantile(0.75)

     # IQR (Çeyrekler Arası Mesafe) hesapla
     IQR = Q3 - Q1

     # Alt ve üst sınırları belirle
     alt_sinir = Q1 - 1.5 * IQR
     ust_sinir = Q3 + 1.5 * IQR

     # Outlier'ları belirle
     aykiri_degerler = (df['sigorta_ücretleri'] < alt_sinir) | (df['sigorta_ücretleri'] > ust_sinir)

     # Aykırı değerlerin sayısını yazdır
     print(f"IQR'ye göre tespit edilen aykırı değer sayısı: {len(df[aykiri_degerler])}")
```

```
IQR'ye göre tespit edilen aykırı değer sayısı: 139
```

Aykırı Değer Tespiti (Z-Score)

```
[ ]  # Z-Score yöntemiyle outlier (aykırı değer) tespiti

     # 'sigorta_ücretleri' kolonunun ortalamasını ve standart sapmasını hesapla
     ortalama = df['sigorta_ücretleri'].mean()
     std_sapma = df['sigorta_ücretleri'].std()

     # Z-Score için eşik değeri belirle
     esik = 3

     # Z-Score ile aykırı değerleri tespit et
     aykiri_degerler_zscore = (abs((df['sigorta_ücretleri'] - ortalama) / std_sapma) > esik)

     # Z-Score yöntemine göre aykırı değerlerin sayısını yazdır
     print(f"Z-Score'a göre tespit edilen aykırı değer sayısı: {len(df[aykiri_degerler_zscore])}")

     # IQR yöntemine göre aykırı değerlerin bulunması için outlier_iqr'yi tekrar kullan
     outlier_iqr = (df['sigorta_ücretleri'] < alt_sinir) | (df['sigorta_ücretleri'] > ust_sinir)

     # IQR ve Z-Score yöntemlerine göre ortak aykırı değerleri (univariate) tespit et
     ortak_aykiri_degerler_univariate = outlier_iqr & aykiri_degerler_zscore

     # Ortak aykırı değerlerin sayısını yazdır
     print(f"Ortak aykırı değerler (univariate): {len(df[ortak_aykiri_degerler_univariate])}")
```

```
Z-Score'a göre tespit edilen aykırı değer sayısı: 7
Ortak aykırı değerler (univariate): 7
```

Aykrı Değer Tespiti (Isolation Forest, Local Outlier)

```
[ ] from sklearn.ensemble import IsolationForest
    from sklearn.neighbors import LocalOutlierFactor

    # Isolation Forest ile outlier (aykırı değer) tespiti
    clf = IsolationForest()
    clf.fit(df[sayisal_kolonlar])  # Sayısal kolonlarla model eğitimi yap
    outliers_if = clf.predict(df[sayisal_kolonlar])  # Aykırı değer tahminleri

    # Local Outlier Factor (LOF) ile outlier tespiti
    lof = LocalOutlierFactor(n_neighbors=10)
    outliers_lof = lof.fit_predict(df[sayisal_kolonlar])  # LOF ile aykırı değer tahminleri

    # Multivariat yöntemle ortak aykırı değerleri tespit et
    ortak_aykiri_degerler_multivariate = (outliers_if == -1) & (outliers_lof == -1)

    # Ortak aykırı değerlerin sayısını yazdır
    print(f"Multivariat yöntemle tespit edilen ortak aykırı değer sayısı: {len(df[ortak_aykiri_degerler_multivariate])}")

    # IQR ve Z-Score yöntemlerine göre aykırı değerlerin ortaklarını univariate tespitiyle bul
    final_aykiri_degerler = df[ortak_aykiri_degerler_univariate & ortak_aykiri_degerler_multivariate]

    # Final aykırı değerlerin sayısını yazdır
    print(f"Final aykırı değer sayısı: {len(final_aykiri_degerler)}")
```

```
Multivariat yöntemle tespit edilen ortak aykırı değer sayısı: 13
Final aykırı değer sayısı: 7
```

Aykırı Değerlerin Veri Setinden Çıkarılması

```
[ ] # Aykırı değerleri (outliers) veri setinden çıkarma
    df = df[~ortak_aykiri_degerler_univariate]  # Ortak aykırı değerleri çıkar

    # Aykırı değerler çıkarıldıktan sonra veri setinin yeni boyutunu yazdır
    print(f"Aykırı değerler çıkarıldıktan sonraki veri setinin boyutu: {df.shape}")
```

```
Aykırı değerler çıkarıldıktan sonraki veri setinin boyutu: (1331, 7)
```

Veri Ön İşleme

```
[ ] from sklearn.preprocessing import LabelEncoder

    # Kategorik kolonlar için encoding (etiketleme) işlemi

    encode = LabelEncoder()

    # 'cinsiyet', 'sigara_kullanimi' ve 'bölge' kolonlarını encode et
    df['cinsiyet'] = encode.fit_transform(df['cinsiyet'])  # 'cinsiyet' kolonunu encode et
    df['sigara_kullanimi'] = encode.fit_transform(df['sigara_kullanimi'])  # 'sigara_kullanimi' kolonunu encode et
    df['bölge'] = encode.fit_transform(df['bölge'])  # 'bölge' kolonunu encode et

    df
```

	yaş	cinsiyet	vucut_kitle_indeksi	cocuk_sayisi	sigara_kullanimi	bölge	sigorta_ücretleri
0	19	1	27.900	0	0	0	16884.92400
1	18	0	33.770	1	1	1	1725.55230
2	28	0	33.000	3	1	1	4449.46200
3	33	0	22.705	0	1	2	21984.47061
4	32	0	28.880	0	1	2	3866.85520
...	...	...	...	...	...	...	...
1333	50	0	30.970	3	1	2	10600.54830
1334	18	1	31.920	0	1	3	2205.98080
1335	18	1	36.850	0	1	1	1629.83350
1336	21	1	25.800	0	1	0	2007.94500
1337	61	1	29.070	0	0	2	29141.36030

1331 rows × 7 columns

```
[ ] from sklearn.model_selection import train_test_split

    # Bağımlı ve bağımsız değişkenlerin belirlenmesi
    y = df.pop('sigorta_ücretleri')  # 'charges' kolonunu bağımlı değişken olarak al
    x = df  # Geriye kalan tüm kolonları bağımsız değişkenler olarak al
```

Model Ayarları

```
[ ] # Veri setini eğitim ve test setlerine ayırma
    X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, random_state=42)
```

```
[ ] from sklearn.preprocessing import PolynomialFeatures

    # Linear Regression için Polynomial Features (Polinom Özellikleri) oluşturma
    poly = PolynomialFeatures(degree=2, include_bias=False)  # Polinom derecesini 2 olarak ayarla

    # Eğitim ve test setleri için polinom özelliklerini oluştur
    X_train_poly = poly.fit_transform(X_train)  # Eğitim setine polinom özelliklerini uygula
    X_test_poly = poly.transform(X_test)  # Test setine polinom özelliklerini uygula
```

Lineer Regresyon

```
[ ] from sklearn.linear_model import LinearRegression

    # Lineer Regresyon modelini eğitme
    model = LinearRegression()  # Lineer regresyon modelini oluştur
    model.fit(X_train_poly, y_train)  # Eğitim setiyle modeli eğit
```

```
▾ LinearRegression ❶ ❷
LinearRegression()
```

```
[ ] # Lineer Regresyon ile tahmin yapma
    y_tahmin_poly = model.predict(X_test_poly)  # Test seti için tahmin yap
    train_tahmin_poly = model.predict(X_train_poly)  # Eğitim seti için tahmin yap
```

```
[ ] from sklearn.metrics import mean_squared_error, r2_score, explained_variance_score
    import numpy as np

    # Lineer Regresyon için değerlendirme metrikleri
    rmse_poly = np.sqrt(mean_squared_error(y_test, y_tahmin_poly))  # Kök Ortalama Kare Hata (RMSE)
    r2_poly = r2_score(y_test, y_tahmin_poly)  # R-Kare (R²) skoru
    explained_var_poly = explained_variance_score(y_test, y_tahmin_poly)  # Açıklanan Varyans

    # Sonuçları yazdır
    print("Lineer Regresyon ile Polinom Özellikleri\n")
    print(f"RMSE: {rmse_poly}")
    print(f"R-Kare: {r2_poly}")
    print(f"Açıklanan Varyans: {explained_var_poly}")
```

```
Lineer Regresyon ile Polinom Özellikleri

RMSE: 4503.143401470309
R-Kare: 0.8722756233537974
Açıklanan Varyans: 0.8723573319306637
```

Random Forest

```
[ ] from sklearn.ensemble import RandomForestRegressor
    import numpy as np

    # Random Forest modelini oluşturma
    rf = RandomForestRegressor(random_state=42)  # Modeli oluştur

    # Random Forest için hiperparametre aralıkları
    param_rf = {
        'n_estimators': np.arange(50, 500, 50),  # Ağaç sayısı (50 ile 500 arasında, 50'lik artışlarla)
        'max_depth': np.arange(3, 15, 1),  # Ağaç derinliği (3 ile 15 arasında, 1'lik artışlarla)
        'min_samples_leaf': np.arange(2, 15, 5),  # Minimum örnek sayısı (dal yaprağında)
        'min_samples_split': [3, 5, 10, 15],  # Minimum örnek sayısı (dallanma için)
        'bootstrap': [True, False],  # Bootstrap örnekleme
        'criterion': ['squared_error', 'absolute_error', 'friedman_mse', 'poisson'],  # Hata ölçütleri
    }
```

```
[ ] from sklearn.model_selection import RandomizedSearchCV

    # Random Forest için RandomizedSearchCV ile hiperparametre optimizasyonu
    rf_search = RandomizedSearchCV(rf, param_rf, cv=5, n_jobs=-1, verbose=0, scoring='neg_mean_squared_error', n_iter=100)

    # Eğitim seti ile RandomizedSearchCV'yi eğit
    rf_search.fit(X_train, y_train)

    # En iyi modeli seç
    rf_model = rf_search.best_estimator_
```

```
[ ] # Random Forest modeli ile test verisi üzerinde tahmin yapma
    y_tahmin_rf = rf_model.predict(X_test)
```

```
[ ] # Gerçek Y, Tahmin Edilen Y ve Hata'nın karşılaştırılması
    comparison_df = pd.DataFrame({
        'Y_Asli': y_test,  # Gerçek değerler
        'Y_Prediksi': y_tahmin_rf,  # Tahmin edilen değerler
        'Error': y_test - y_tahmin_rf  # Gerçek ve tahmin edilen değer arasındaki fark (Hata)
    })

    # Sonuçları yazdırma
    print("\nGerçek Y, Tahmin Edilen Y ve Hata Karşılaştırması\n")
    print(comparison_df)
```

```
Gerçek Y, Tahmin Edilen Y ve Hata Karşılaştırması

          Y_Asli    Y_Prediksi         Error
950   11534.87265  12619.464100  -1084.591450
1131   3693.42800   5682.051595  -1988.623595
1239   3238.43570   5247.546193  -2009.110493
298   38746.35510  37075.284233   1671.070867
657    4058.71245   7455.978535  -3397.266085
...           ...           ...           ...
533   19214.70553   5135.866332  14078.839198
1200   6198.75180   7370.915427  -1172.163627
759   36307.79830  37938.440193  -1630.641893
1298   5261.46945   7321.216279  -2059.746829
1284  47403.88000  47214.847412    189.032588

[267 rows x 3 columns]
```

```python
from sklearn.metrics import mean_squared_error, r2_score, explained_variance_score
import numpy as np

# Random Forest için değerlendirme metrikleri
rmse_rf = np.sqrt(mean_squared_error(y_test, y_tahmin_rf))  # Kök Ortalama Kare Hata (RMSE)
r2_rf = r2_score(y_test, y_tahmin_rf)  # R-Kare (R²) skoru
explained_var_rf = explained_variance_score(y_test, y_tahmin_rf)  # Açıklanan Varyans

# Sonuçları yazdırma
print("\nRandom Forest\n")
print(f"RMSE: {rmse_rf}")
print(f"R-Kare: {r2_rf}")
print(f"Açıklanan Varyans: {explained_var_rf}")
```

```
Random Forest

RMSE: 4278.417642265108
R-Kare: 0.8847054999891255
Açıklanan Varyans: 0.8847129590274405
```

```python
# Linear Regression için polinomal özellikler ile tahmin yapma
y_pred_poly = model.predict(X_test_poly)

# Linear Regression için değerlendirme metrikleri
rmse_poly = np.sqrt(mean_squared_error(y_test, y_pred_poly))  # Kök Ortalama Kare Hata (RMSE)
r2_poly = r2_score(y_test, y_pred_poly)  # R-Kare (R²) skoru
explained_var_poly = explained_variance_score(y_test, y_pred_poly)  # Açıklanan Varyans

# Random Forest için değerlendirme metrikleri
rmse_rf = np.sqrt(mean_squared_error(y_test, y_tahmin_rf))  # Kök Ortalama Kare Hata (RMSE)
r2_rf = r2_score(y_test, y_tahmin_rf)  # R-Kare (R²) skoru
explained_var_rf = explained_variance_score(y_test, y_tahmin_rf)  # Açıklanan Varyans

# İki modelin karşılaştırılması
print("\nLinear Regression ve Random Forest Modeli Karşılaştırması\n")
print(f"{'Metric':<30}{'Linear Regression':<20}{'Random Forest':<20}")
print("-" * 70)
print(f"{'RMSE':<30}{rmse_poly:<20.4f}{rmse_rf:<20.4f}")
print(f"{'R-Kare':<30}{r2_poly:<20.4f}{r2_rf:<20.4f}")
print(f"{'Açıklanan Varyans':<30}{explained_var_poly:<20.4f}{explained_var_rf:<20.4f}")
```

```
Linear Regression ve Random Forest Modeli Karşılaştırması

Metric                         Linear Regression   Random Forest
----------------------------------------------------------------------
RMSE                           4503.1434           4278.4176
R-Kare                         0.8723              0.8847
Açıklanan Varyans              0.8724              0.8847
```