

YOZGAT BOZOK ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

VERİ ANALİZİNE GİRİŞ DERSİ

TEKNİK RAPOR

MAKİNE ÖĞRENMESİ

MÜMİN EMİNCAN KURNAZ

KAAN SÖZGEN

FATİH TÜRKMEN

MUSTAFACAN AKGÜN

2022

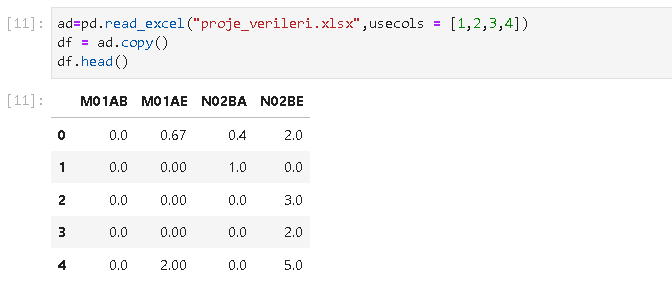
**DOĞRUSAL REGRASYON MODELLERİ**

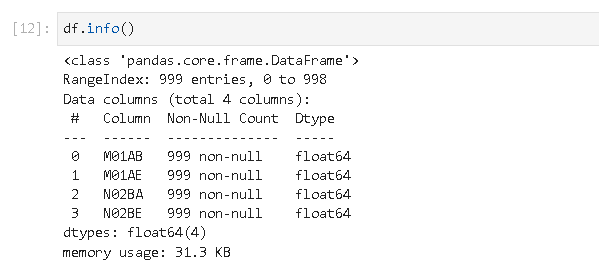
BASİT DOĞRUSAL REGRASYON:

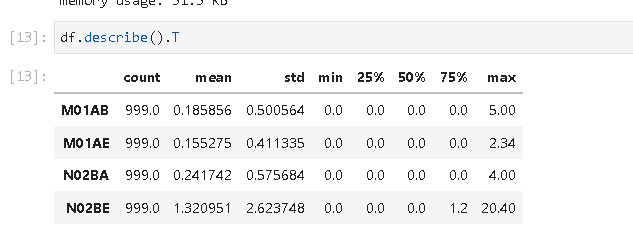
Burada kütüphanelerimizi tanımladık import os kullanarak os.getcwd komutu ile dizin yolunu açtırdık daha sonra os.chdir ile okunmayan excel dosyamızın yolunu elle girdik.

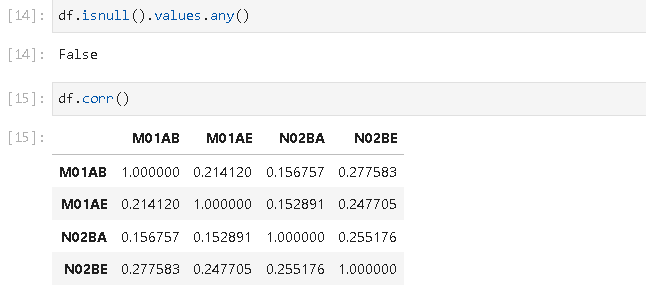


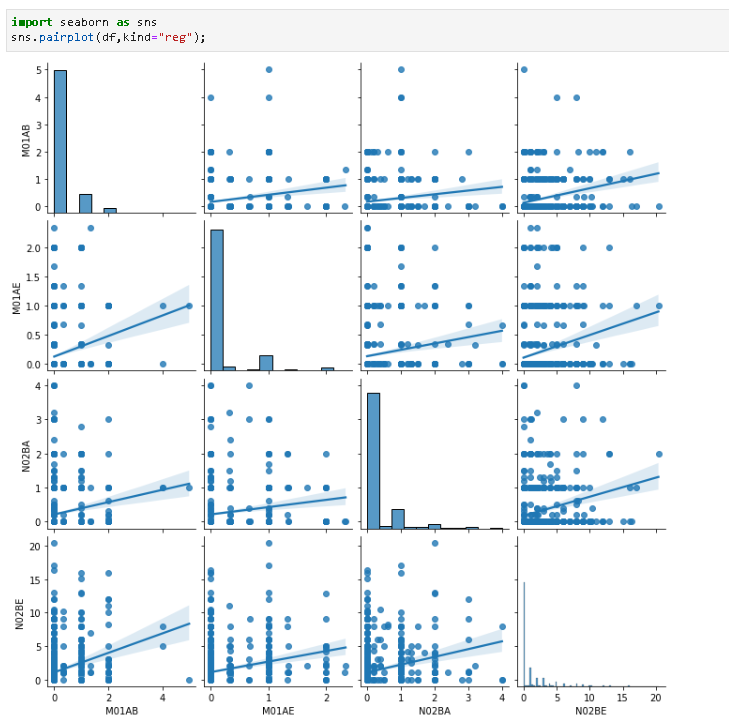
Dosyamızı okuttuk ve 4 satır sütün şeklinde yazdırdık.

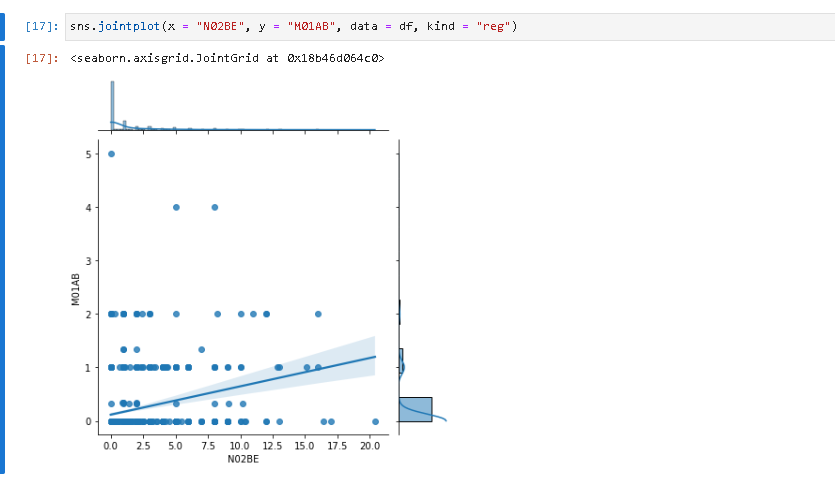




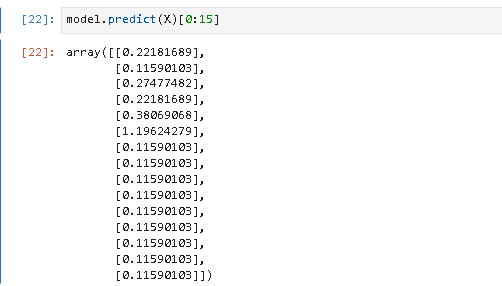


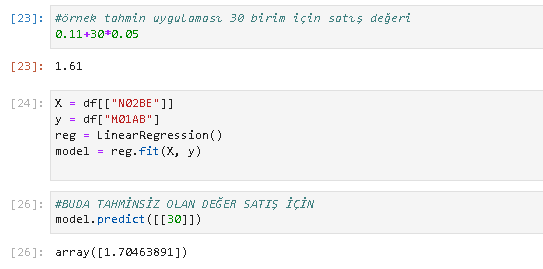


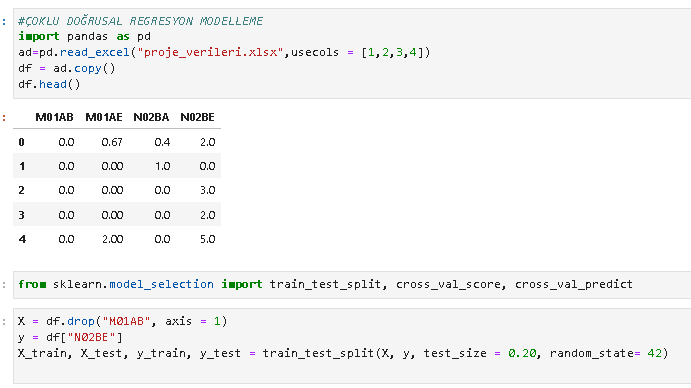




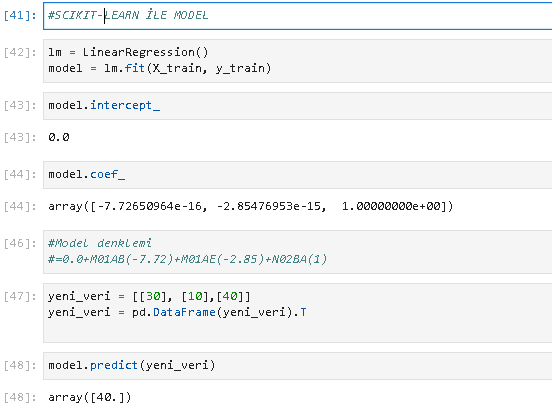






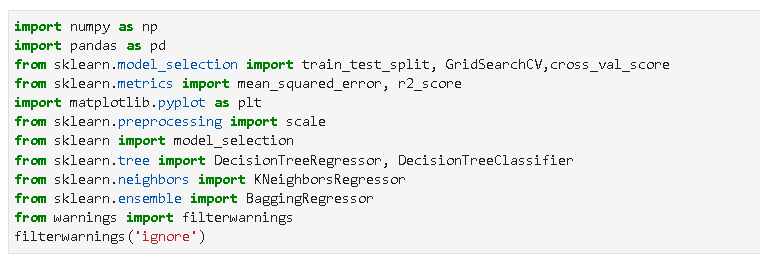


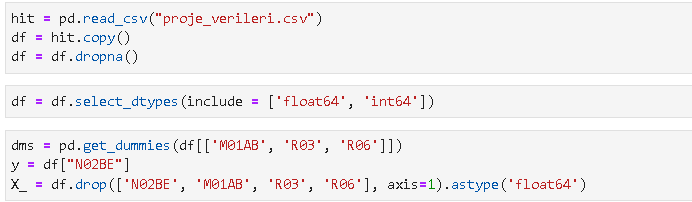


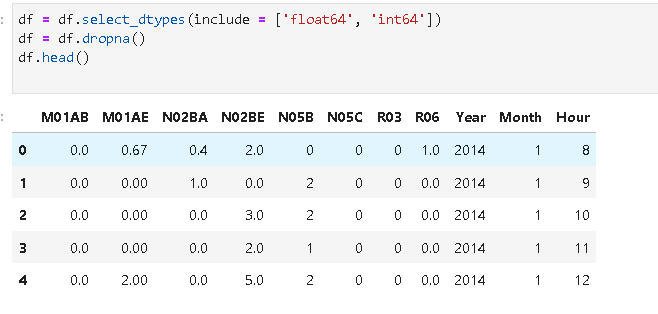


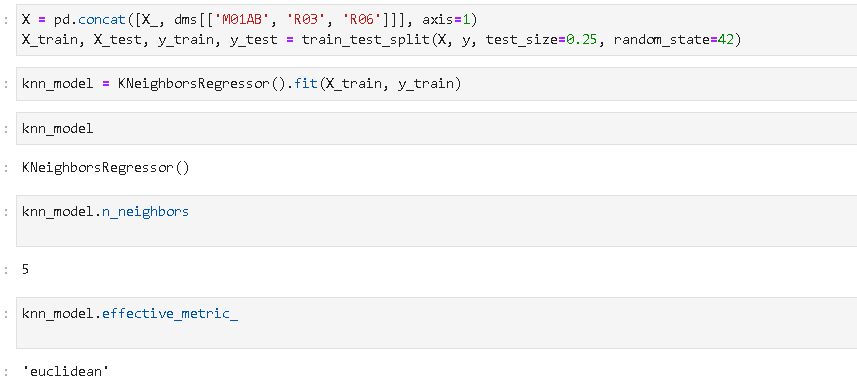
**DOĞRUSAL OLMAYAN REGRASYON MODELLERİ(KNN)**

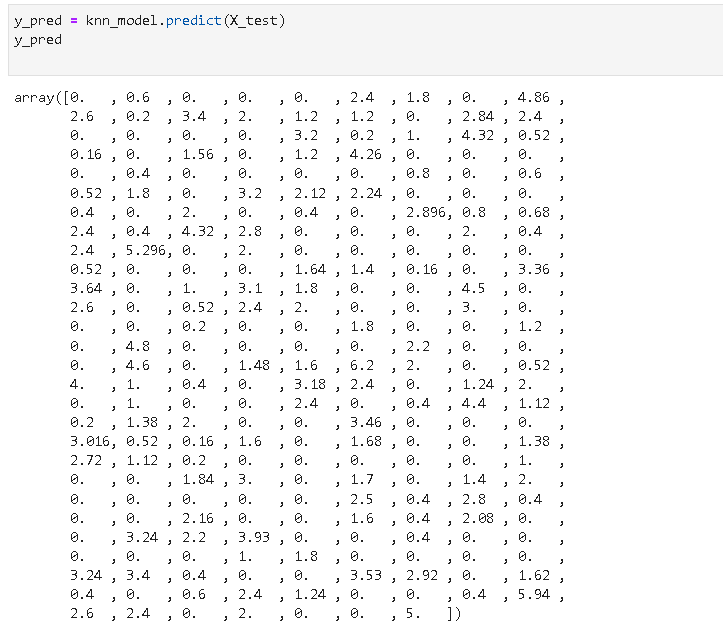
Kütüphane Import İşlemleri

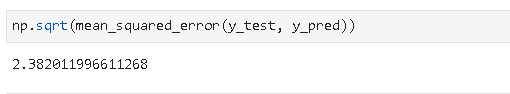


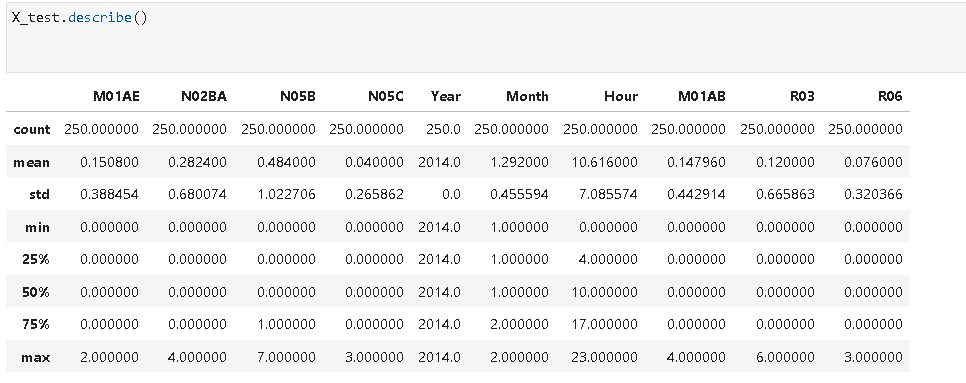


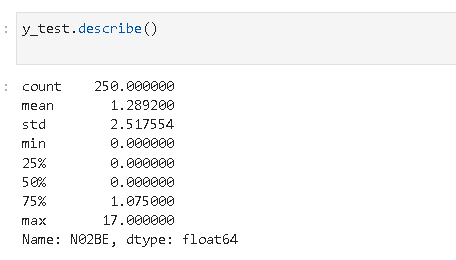




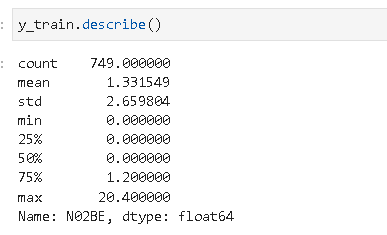


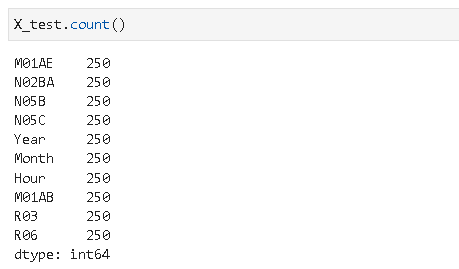




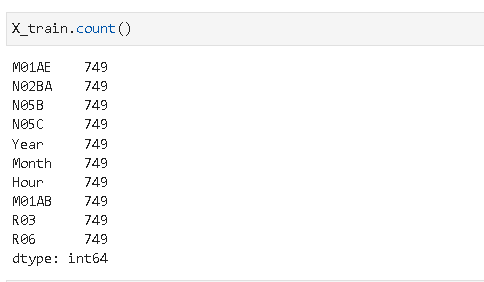


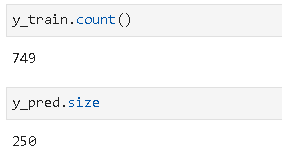


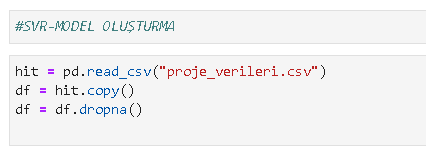


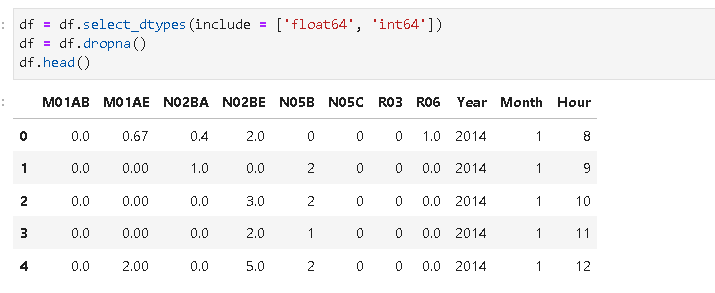


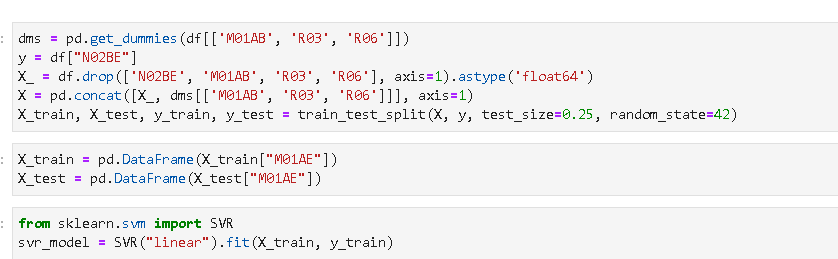


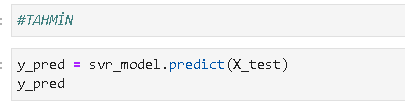


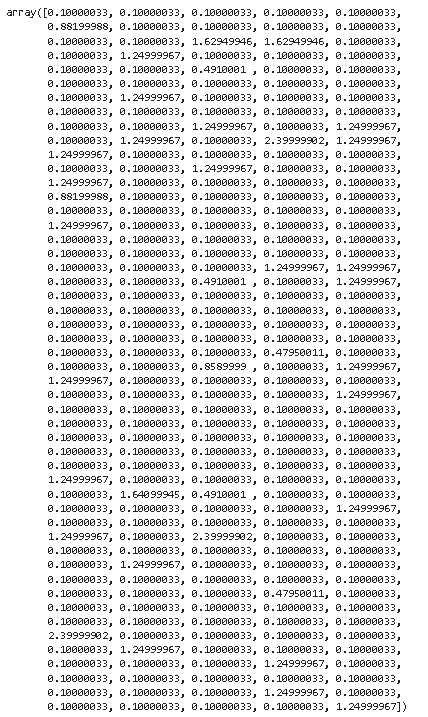


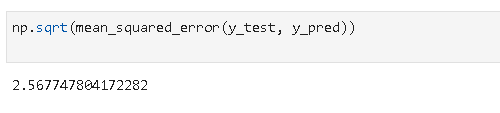




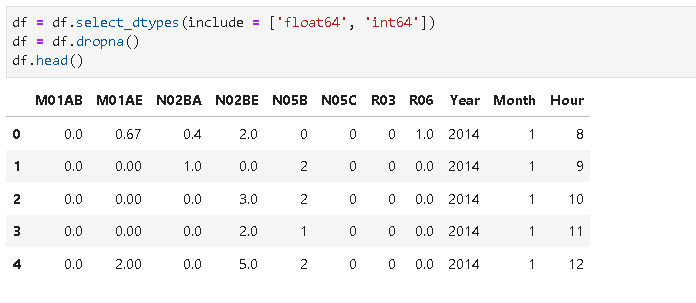




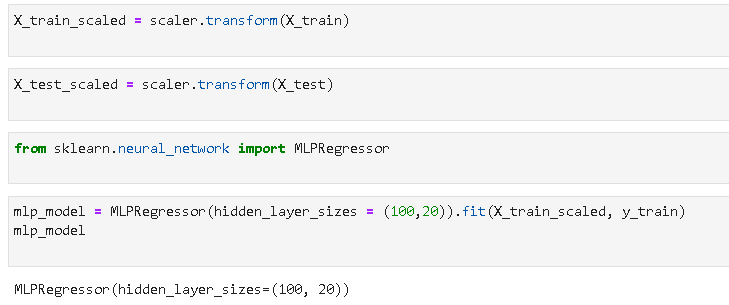




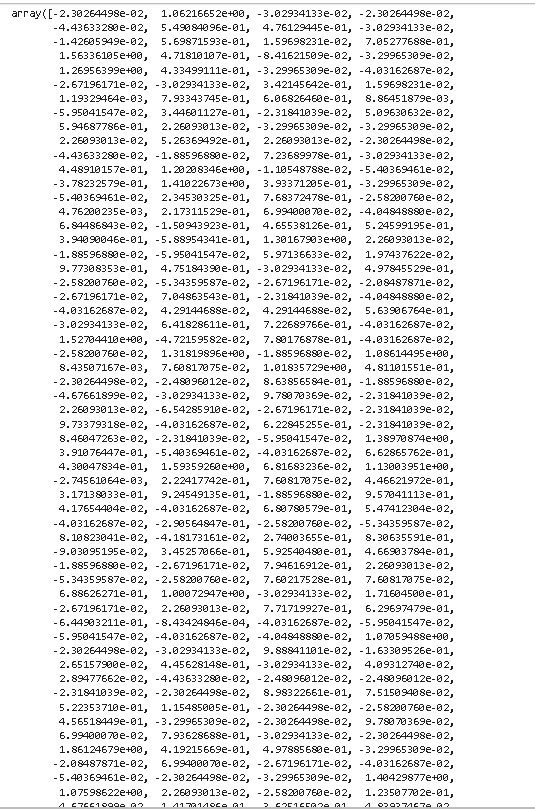


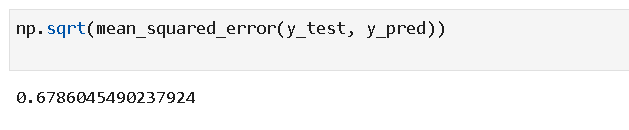




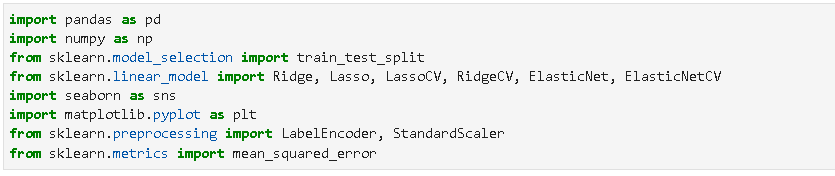






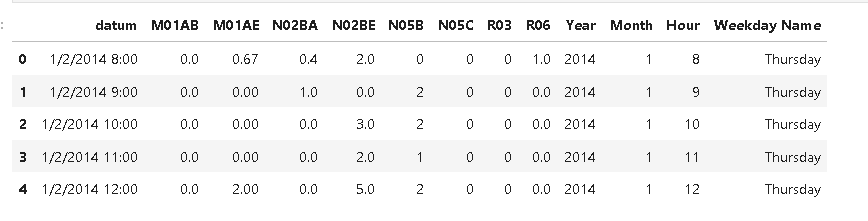


Öncelikle gerekli olan kütüphaneleri dahil ederek başlayalım.

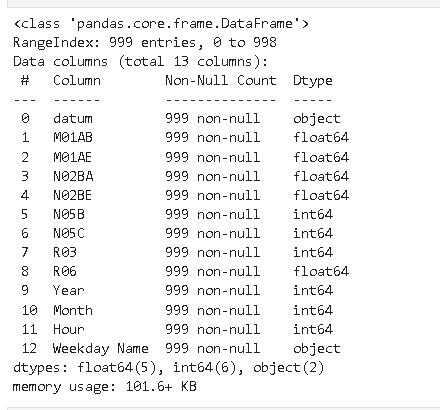


Gerekli kütüphaneleri dahil ettiğimize göre veri setini incelemeye geçebiliriz.



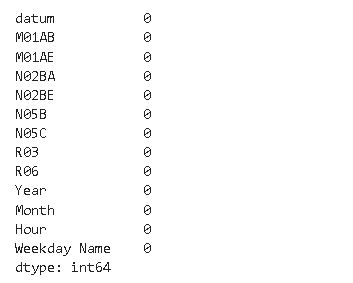


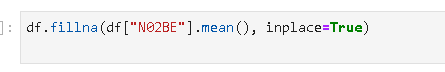
**df.info()** ile veri setimiz hakkında daha geniş bir bilgiye sahip olabiliriz.



Yukarıdaki çıktıyı incelediğimizde veri setinin boyutunun (999,13) olduğunu görüyoruz. 2 tane kategorik değişkenimiz var bunlar object olarak belirtilmiş eksik verilerin sayısını görmek için;

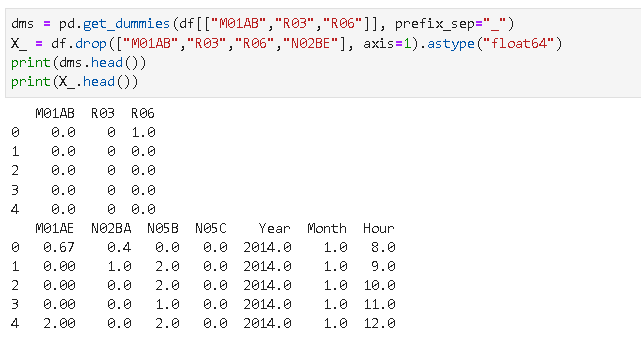






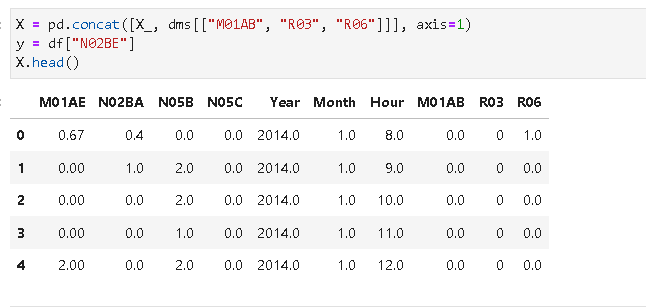
Yukarıdaki **“inplace=True”** parametresi yaptığımız değişikliğin kalıcı bir şekilde uygulanmasını sağlar.

Sıra geldi kategorik değişkenleri numerik değişkenlere dönüştürmeye. Bunun için pandas içindeki **“get\_dummies()”**i kullanacağız.



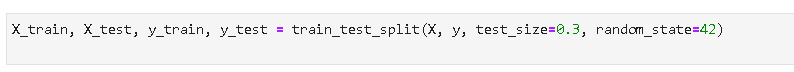
“get\_dummies()” içinde numerik değişkenlere dönüşücek kategorik değişkenleri yazıyoruz. Daha sonra veri setinden bu kategorik değişkenleri ve tahmin etmek istediğimiz değişkeni düşürüyoruz ve bunu **X\_**’ye eşitliyoruz.

dms’i incelediğimizde kategorik değişkenlerin 1 ve 0'lar ile numerik değişkenlere dönüştüğünü görüyoruz. Veri setinde her bir değişkenin bir tanesini kullanmamız yeterli olacaktır.

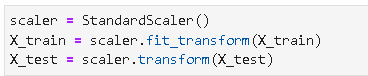


Dönüştürülen değişkenler ve X\_’yi birleştirerek X’i elde ediyoruz. X, y’yi tahmin etmek için kullanacağımız bağımsız değişkenleri içeriyor. y’yi de tahmin etmek istediğimiz değişkene eşitliyoruz.

Bunları yaptıktan sonra artık veriyi train ve test olarak bölebiliriz.

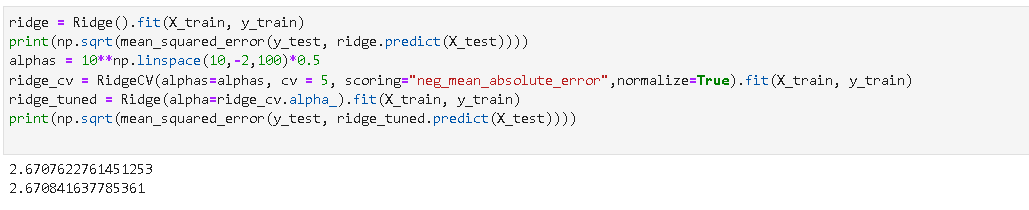


**X\_train**ve **y\_train** modeli geliştirmek için **X\_test** ve **y\_test** ise modeli test etmek için kullanılacak bağımlı ve bağımsız değişkenleri gösteriyor. **Test\_size** verilerin % kaçını test için kullanılacağını (%30) belirtir. Burada herhangi bir değer belirtmezsek defaultta gelen 0.25 değeri kullanılır. **Random\_state** ise programı her çalıştırdığımızda aynı ayrımı görmek için kullanılır.



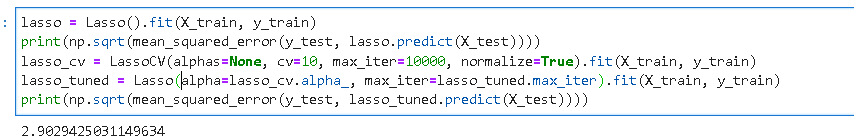
StandartScaler verileri, ortalaması 0, standart sapması 1 olacak şekilde dönüştürür. Kısacası verileri standartlaştırır.

**RIDGE REGRESYON**



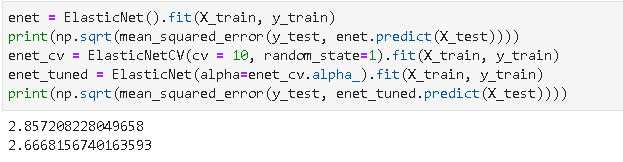
İlk başta **“ridge”** adında bir model yaratıyoruz ve modele hiçbir parametre vermeden X\_train ve y\_train ile eğitiyoruz. Bunun sonucunda aldığımız hata 2.6707622761 oluyor. Daha sonra **RidgeCV** yi kullanarak bir cross validation işlemi uyguluyoruz. Bu işlem veri setimiz için en uygun parametreleri elde etmemizi sağlıyor. İşlem tamamlandıktan sonra elde ettiğimiz en iyi alpha değeri ile yeni bir model (ridge\_tuned) kuruyoruz ve eğitiyoruz. Buradan elde ettiğimiz hata ise 2.67084163 oluyor.

**LASSO REGRESYON**



**“lasso”** adındaki modelimizi oluşturup, eğittikten sonra elde ettiğimiz hata 2.902942503 oluyor. Daha sonra en iyi parametreleri bulmak için**LassoCV** yani cross validation uyguluyoruz. Son olarak da elde ettiğimiz parametrelerle yeni bir model kurup eğittiğimiz zaman hata 2.806451245 oluyor.

**ELASTIC NET**



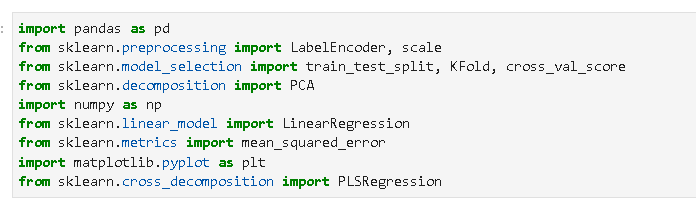
Burada da aynı yukarıdaki işlemleri uyguluyoruz. İlk oluşturduğumuz modelde elde ettiğimiz hata 2.857208228 oluyor. Tune edildikten sonra elde ettiğimiz hata ise 2.666815674 oluyor.

PCR’ın SciKit-Learn’de direkt uygulaması olmadığı için bunu kendimiz yapmalıyız. PCR uygulaması için;

1. PCA yapılacak.
2. Regresyon modeline oturtulacak.

O zaman hazırsanız başlayalım. Kütüphanelerin dahil edilmesi ve verinin incelenip düzenlenmesi bölümleri hem PCR hem de PLS için kullanılacak.

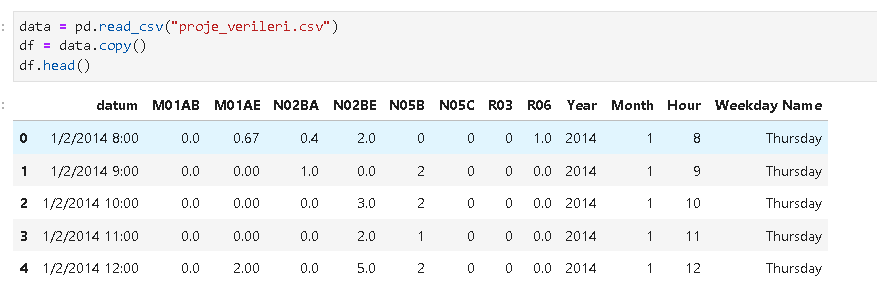
**Kütüphanelerin Dahil Edilmesi**

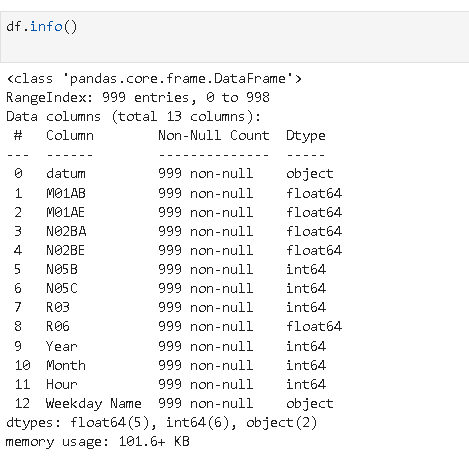


Kullanacağımız kütüphaneler yukarıdaki gibi olacak.

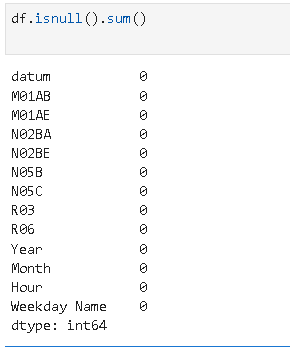
**Verinin İncelenmesi Ve Düzenlenmesi**

Öncelikle veri setini okuyalım ve inceleyelim.

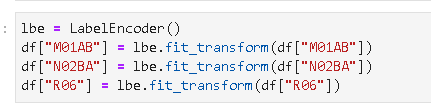




Görüldüğü gibi veri setinde object değişkenleri de bulunuyor. Bunları sürekli değişkene çevirme işlemini az sonra yapacağız. Öncelikle eksik değerleri inceleyelim.



Eksik verimiz bulunmadığına göre ,artık kategorik değişkenleri sürekli değişkene çevirme kısmına geçebiliriz.



Burada LabelEncoder kullanarak kategorik değişkenleri 1 veya 0'a dönüştürüyoruz.

Hazırlıkları bitirdik artık model oluşturma kısmına geçebiliriz.

## PCR — Model ve Tahmin

Öncelikle bağımlı ve bağımsız değişkenlerimizi belirlemeliyiz.

## 

Daha sonra verimizi train ve test olarak böleceğiz.

## 

Burada test\_size parametresi ile veri setinin %20'sinin test kısmına ayrılcağını belirttik.

Yukarıda da bahsettiğim gibi PCR’ın pythonda direkt uygulaması olmadığı için sıradaki işlemimiz PCA nesnesi oluşturmak olacak. Ve daha sonra X\_train verimizi scale (ölçeklendirme) işlemini yapacağız.

## 

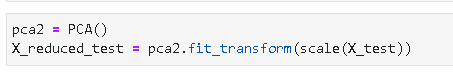
Normalde PCA n\_components yani bileşen sayısı parametresini alır. Eğer parametreyi vermezsek bütün bileşenleri kullanır.

PCA’yı da hallettiğimize göre PCR’ın ikinci kısmı olan regresyon modeline oturtmaya geçebiliriz.

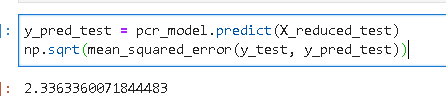
Bunun için LinerRegression objesi oluşturuyoruz. Daha sonra ise PCR modelini oluşturma kısmına geçiyoruz.

## 

Model oluşturma işlemi bitti sıra tahmin etmekte. Bunun için öncelike X\_test verisini de scale etmeliyiz.



Daha sonra tahmin işlemine geçip test hatasına bakabiliriz.



Bunun sonucunda test hatasının2.336336007 olduğunu görürüz.

## PCR — Model Tuning

## Yukarıda yaptığımız işlemlerde n\_components parametresine değer vermedik ve modelimiz veri setindeki tüm bileşenleri kullanarak işlemi gerçekleştirdi. Model tuning kısmında n\_components parametresine verebileceğimiz optimum değişken sayısını cross validation yöntemi ile belirleyip modeli tekrar kuracağız. Daha sonra tahmin etme işlemini gerçekleştirip tune edilmiş modelimizin test hatasına bakacağız.

## 

10 katlı cross validation yöntemi uygulanacak. shuffle parametresi ise gruplara ayrılmadan önce verinin karıştırılıp karıştırılmayacağını belirliyor.

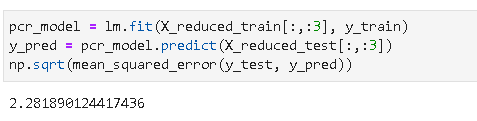
Pythonda direkt PCR’ı kullanabileceğimiz bir yol olmadığından dolayı her parametrenin RMSE değerini bulmayı kendimiz for döngüsü ile yapmalıyız.

## 

RMSE değeri en küçük olan bileşen sayısı optimum bileşen sayımız olacak ve buna görselleştirme yaparak bakabiliriz.

## 

Grafiği incelediğimizde optimum bileşen sayısının 3 olduğunu görüyoruz. Şimdi bu bileşen sayısı üzerinden final modelini oluşturup test hatasına bakabiliriz.



Bunun sonucunda test hatasının 2.2818901244 olduğunu görüyoruz ve bu da model tuning yapmadan önceki test hatasına göre daha iyi bir test hatası.

# PLS

Değişkenlerin az sayıda ve aralarında çoklu doğrusal bağlantı problemi bulunmayan bileşenlere indirgeyip regresyon modeli kurulmasıdır.

PLS’de bileşenler bağımsız değişkenler ile olan kovaryansı maksimum özetleyecek şekilde oluştururlar.

## PLS — Model ve Tahmin

## 

Yukarıda herhangi bir n\_components değeri vermeden modelimizi oluşturduk ve fit ettik.

## 

Oluşturduğumuz model üzerinden tahmin işlemi gerçekleştirdiğimiz zaman elde ettiğimiz test hatası 2.26285876444 oluyor.

## PLS — Model Tuning

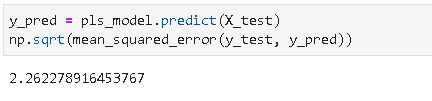
PCR’da olduğu gibi optimum bileşen sayısını bulmak için cross validation yöntemi kullanacağız. Daha sonra her bir bileşen sayısı için RMSE değerlerini bulup görselleştirme yaparak optimum bileşen sayısına bakacağız.

## 

Yukarıdaki grafiği incelediğimizde optimum bileşen sayısının 1,2,4,5,6,7,8,9 olduğunu görüyoruz. Şimdi final modelini oluşturmaya geçebiliriz.



n\_components parametresini 8 değerini vererek final modelimizi oluşturduk. Sıra test hatasına bakmakta.



Bu işlem sonucunda test hatasına 2.262278916 olarak ulaşıyoruz. PCR ve PLS ‘in tune edilmiş modellerinin test hatalarını karşılaştırırsak PLS’in test hatasının daha az olduğunu görüyoruz.