python™

NumPy

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$

Data Science
Çalışma Dökümanı
Recep Aydoğdu

İçindekiler

Data Science.....21
Data Science Kullanılan Alanlar.....24
Genel Resim.....25
Data Science Proje Döngüsü.....26
Veri Bilimine Giriş Alıştırmalar – 1.....27
Veri Bilimine Giriş Alıştırmalar – 2.....31
Data Literacy (Veri Okuryazarlığı).....34
Veri Okuryazarlığı Nedir?.....34
Population and Sample (Popülasyon ve Örneklem).....35
Observation Unit (Gözlem Birimi).....35
Variables and Variable Types (Değişken ve Değişken Türleri)...35
Scales of Measurement (Ölçek Türleri).....37
Sayısal Değişkenler.....37
Kategorik Değişkenler.....38
Merkezi Eğilim Ölçüleri.....39
Arithmetic Mean (Aritmetik Ortalama).....39
Median (Medyan).....39
Mode (Mod).....41
Quartiles (Kartiller).....41
Merkezi Eğilimin Önemini Anlamak.....42
Measure of Dispersion (Dağılım Ölçüleri).....42
Range (Değişim Aralığı).....42
Standard Deviation (Standart Sapma).....43
Variance (Varyans).....44
Skewness (Çarpıklık).....45
Kurtosis (Basıklık Ölçüsü).....46
Statistical Thinking Models (İstatistiksel Düşünce Modelleri)...47
Mooney Modeli.....47
Statistical Thinking Levels.....48
Verinin Tanımlanması.....48
Verilerin Organize Edilmesi ve İndirgenmesi.....49
Verinin Gösterimi.....50
Verilerin Analiz Edilmesi ve Yorumlanması.....51

Veri Okuryazarlığı Alıştırmalar – 1.....52
Veri Okuryazarlığı Alıştırmalar – 2.....55
----Python Programlama----.....59
Temel Hareketler.....59
Integer, Float ve String.....59
String Metodları.....60
Type Dönüşümleri.....61
print() fonksiyonu.....61
Python Programlama Alıştırmalar – 1.....62
Python Programlama Alıştırmalar – 2.....65
Python Programlama Alıştırmalar – 3.....70
Veri Yapıları (Data Types).....74
Listeler.....74
Liste Elemanlarına Ulaşma.....74
Liste İçi Type Sorgulama.....74
Liste elemanlarını değiştirme.....75
Listeye eleman ekleme.....75
Listeden eleman silme.....75
append ve remove metodlari.....75
insert metodu.....75
pop metodu.....75
count metodu.....76
copy metodu.....76
extend metodu.....76
index metodu.....76
reverse metodu.....76
sort metodu.....76
clear metodu.....77
Tuple (Demet).....77
Tuple Oluşturma.....77
Eleman İşlemleri.....77
Dictionary (Sözlük).....77
Dictionary Nedir?.....77
Dictionary Oluşturma.....78
Eleman Seçme İşlemleri.....78

Eleman Ekleme & Değiştirme.....79
Sets (Kümeler).....79
Set Oluşturma.....79
Set'lere eleman ekleme ve çıkarma işlemleri.....81
Set'lerde Fark İşlemleri.....83
Set'lerde Kesişim ve Birleşim İşlemleri.....83
Set'lerde Sorgu İşlemleri.....84
Veri Yapıları Özet.....84
Python Programlama Alıştırmalar – 4.....85
Python Programlama Alıştırmalar – 5.....88
Python Programlama Alıştırmalar – 6.....93
Fonksiyonlar.....99
Fonksiyon Nedir?.....99
Matematiksel İşlemler.....99
Üs Alma.....99
Fonksiyon Nasıl Yazılır ?.....99
Bilgi Notuyla Çıktı Üretmek.....100
İki Argümanlı Fonksiyon Tanımlamak.....101
Ön Tanımlı Argümanlar.....101
Argümanların Sıralaması.....101
Ne Zaman Fonksiyon Yazılır?.....102
Fonksiyon Çıktılarını Girdi Olarak Kullanmak.....102
Local ve Global Değişkenler.....103
Local Etki Alanından Global Etki Alanını Değiştirme.....104
Karar-Kontrol Yapıları (Koşullar).....105
Koşul Nedir?.....105
True – False Sorgulamaları (Boolean).....105
if – else – elif.....105
Uygulama: if ve input ile kullanıcı etkileşimli program.....107
Döngüler.....109
For Döngüsü.....109
Döngü ve Fonksiyonların Birlikte Kullanımı.....109
Uygulama: if, for ve fonksiyonların birlikte kullanımı.....109
break & continue.....110
while.....112

Python Programlama Alıştırmalar – 7.....113
Python Programlama Alıştırmalar - 8.....119
Python Programlama Alıştırmalar – 9.....124
Generators.....130
Object Oriented Programming.....132
Class'lara Giriş ve Class Tanımlamak.....132
Class Nedir?.....132
Class Özellikleri.....132
Class Özelliklerine Erişmek.....132
Class Özelliklerini Değiştirmek.....132
Class Örneklendirmesi (instantiniation).....132
Örnek Özellikleri.....133
Örnek Metodları.....135
Miras Yapıları (inheritance).....135
Functional Programming.....136
Fonksiyonel Programlamaya Giriş.....136
Yan Etkisiz Fonksiyonlar (Pure Functions).....136
Örnek-1: Bağımsızlık.....137
Örnek-2: Ölümcül Yan Etkiler.....137
İsimsiz Fonksiyonlar (Lambda) (Anonymous Functions).....138
Vektörel Operasyonlar (Vectorel Operations).....138
OOP ile iki listeyi çarpmak.....138
Functionel Programming ile.....139
Map & Filter & Reduce.....140
Map.....140
Filter.....140
Reduce.....140
Modül Oluşturma.....140
Hatalar/İstisnalar (exception).....141
Python Programlama Alıştırmalar – 10.....143
Python Programlama Alıştırmalar – 11.....149
Python Programlama Alıştırmalar – 12.....154
--Python ile Veri Manipülasyonu: NumPy & Pandas--.....160
NumPy (Numerical Python).....160
NumPy Giriş.....160

Neden NumPy?.....161
NumPy Array'i Oluşturmak.....161
zeros, ones, full, random, arange, linspace, random.normal, random.randint.....163
NumPy Array Özellikleri.....164
Matris Oluşturma.....164
Reshaping (Array'i Yeniden Şekillendirme).....165
Ravel (Flatten).....166
Reshape ile Resize farkı nedir?.....166
Concantenation (Array Birleştirme).....167
Stacking Array.....168
Convert and Copy.....169
Splitting (Array Ayırma).....169
İki Boyutlu Array Ayırma.....170
Sorting (Sıralama).....170
Matris sıralama.....171
Index ile Elemana Erişmek.....171
Matrislerde elemana erişme işlemleri.....172
Slicing (Array Alt Küme İşlemleri).....173
Matrislerde Slicing İşlemleri.....173
Alt Küme Üzerinde İşlem Yapmak.....174
Fancy Index ile Elemanlara Erişmek.....175
Matrislerde Fancy Index Kullanımı.....176
Basit Index ile Fancy kullanımı.....176
Slice ile Fancy kullanımı.....176
Koşullu Eleman İşlemleri.....177
Matematiksel İşlemler.....178
Trigonometrik Fonksiyonlar.....179
Logaritmik İşlemler.....180
Numpy ile İki Bilinmeyenli Denklem Çözümü.....181
NumPy Alıştırmalar– 1.....182
NumPy Alıştırmalar – 2.....186
NumPy Alıştırmalar – 3.....191
Pandas.....196
Pandas Giriş.....196

Pandas Serisi Oluşturmak.....197
Index İsimlendirmesi.....198
Sözlük Üzerinden Seri Oluşturmak.....198
İki Seriyi Birleştirerek Seri Oluşturma.....199
Eleman İşlemleri.....199
Eleman Sorgulama.....200
Fancy Eleman.....200
Eleman Değiştirme.....200
Pandas DataFrame Oluşturma.....201
DataFrame İsimlendirme.....202
DataFrame Özellikleri.....202
Filtering Pandas Data Frame.....204
DataFrame Eleman İşlemleri.....205
Eleman Silme.....207
Fancy ile eleman silme.....207
Değişkenler için eleman işlemleri.....208
Değişken Silme.....209
Gözlem ve Değişken Seçimi: loc & iloc.....210
Koşullu Eleman İşlemleri.....212
Birleştirme (Join) İşlemleri.....214
İleri Birleştirme İşlemleri.....218
Birebir Birleştirme.....218
Many to one (Çoktan teke).....219
Many to Many (Çoktan çoka).....220
Aggregation & Grouping (Toplulaştırma ve Gruplama).....221
Grouping.....225
İleri Toplulaştırma İşlemleri(Aggregate, filter, transform, apply) 227
Aggregate.....227
filter.....229
transform.....231
Apply.....233
Pivot Tablolar.....235
pivot_table.....236
Dış Kaynaklı Veri Okuma.....238

csv okuma....239
txt okuma....240
Excel dosyası okuma....241
Sıfırdan txt okuma....242
Pandas Alıştırmalar-1....243
Pandas Alıştırmalar-2....247
Pandas Alıştırmalar – 3....252
List Comprehensions....257
--Python ile Veri Görselleştirme--....258
Seaborn....258
Veri Görselleştirme Kütüphaneleri....259
Veriye İlk Bakış....259
Veri Setinin Hikayesi Nedir?....259
Veri Seti Yapısal Bilgileri....260
Veri Setinin Betimlenmesi....261
Eksik Değerlerin İncelenmesi....262
Kategorik Değişken Özetleri....265
Sadece Kategorik Değişkenler ve Özetleri....265
Kategorik Değişkenlerin Sınıflarına ve Sınıf Sayısına Erişmek....266
Kategorik Değişkenin Sınıflarının Frekanslarına Erişmek....266
Sürekli Değişken Özetleri....267
Dağılım Grafikleri....268
Barplot (Sütun Grafiği)....268
Veri Setinin Hikayesi....268
Veri Setine Hızlı Bakış....270
Bar Plot (Sütun Grafiğin) Oluşturulması....274
Sütun Grafik Çaprazlamalar....275
Histogram ve Yoğunluk Grafiği....277
Histogram ve Yoğunluk Çaprazlamalar....280
Boxplot....281
Veri Seti Hikayesi....281
Boxplot Oluşturma....283
Boxplot Çaprazlamalar....284
Violin Grafiği....287
Violin Grafiği Çaprazlamalar....288

Korelasyon Grafiği.....289
Scatterplot (Saçılım Grafiği).....289
Korelasyon Çaprazlamalar.....290
Doğrusal İlişkinin Gösterilmesi.....292
Scatterplot Matrisi (pairplot).....295
Heat Map (Isı Haritası).....300
Çizgi Grafik (Lineplot).....303
Veri Seti Hikayesi.....303
Lineplot Oluşturulması.....306
Basit Zaman Serisi Grafiği.....308
Seaborn Alıştırmalar – 1.....310
Seaborn Alıştırmalar – 2.....315
Seaborn Alıştırmalar – 3.....320
Python Final Sınavı.....328
---Statistic for Data Science---.....339
Giriş.....339
Örnek Teorisi.....339
Örneklem.....340
Örneklem Dağılımı.....341
Merkezi Limit Teoremi.....342
Örnek Teorisi: Uygulama.....343
Örneklem Çekimi.....343
Örneklem Dağılımı.....344
Betimsel İstatistikler.....345
Kovaryans.....345
Korelasyon.....345
Betimsel İstatistikler: Uygulama.....346
Güven Aralığı.....349
Güven Aralığı Nedir?.....349
Güven Aralığı Nasıl Hesaplanır?.....350
İş Uygulaması: Fiyat Stratejisi Karar Destek Sistemi.....350
Olasılığa Giriş ve Olasılık Dağılımları.....351
Rassal Değişkenler ve Olasılık Dağılımları.....351
Dağılım Nedir?.....351
Olasılık Dağılımı Nedir?.....351

Kesikli ve Sürekli Olasılık Dağılımları.....351
Kesikli Olasılık Dağılımları.....351
Sürekli Olasılık Dağılımları.....351
Bernoulli Dağılımı.....352
Bernoulli Dağılımı Uygulama.....352
Büyük Sayılar Yasası.....353
Binom Dağılımı.....354
İş Uygulaması: Reklam Harcaması Optimizasyonu.....355
Poisson Dağılımı.....356
Örnek.....359
İş Uygulaması: İlan Girişi Hata Olasılıklarının Hesaplanması.....359
Normal Dağılım.....360
İş Uygulaması: Ürün Satış Olasılıklarının Hesaplanması.....361
Hipotez Testleri.....363
Hipotezler ve Türleri.....363
Hata Tipleri.....364
p-value.....364
Hipotez Testi Adımları.....365
Tek Örneklem T Testi.....365
T Testi Nedir?.....366
İş Uygulaması: Ürün Satın Alma Adım Optimizasyonu.....366
İş Uygulaması: Web Sitesinde Geçirilen Sürenin Testi.....368
Varsayımlarımız.....369
Tek Örneklem T Testi Uygulaması.....370
Nonparametrik Tek Örneklem T Testi.....371
Tek Örneklem Oran Testi.....371
İş Uygulaması: Dönüşüm Oranı Testi.....371
Bağımsız İki Örneklem T Testi (AB Testi).....372
Test İstatistiği.....373
Varsayımlar.....373
İş Uygulaması: ML Modelinin Başarı Testi.....374
Bağımsız İki Örneklem T Testi Varsayım Kontrolü.....377
Bağımsız İki Örneklem T Testi Uygulama.....378
Nonparametrik Bağımsız İki Örneklem Testi.....379
Bağımlı İki Örneklem T Testi.....379

İş Uygulaması: Şirket İçi Eğitimin Performans Etkisi Ölçümü.....380
Bağımlı İki Örneklem T Testi Varsayım Kontrolü.....383
Bağımlı İki Örneklem T Testi Uygulama.....383
Nonparametrik Bağımlı İki Örneklem Testi.....384
İki Örneklem Oran Testi.....384
İş Uygulaması: Kullanıcı Arayüz Deneyi (AB Testi).....384
Varyans Analizi.....385
İş Uygulaması: Anasayfa İçerik Stratejisi Belirleme.....386
Varsayım Kontrolü.....387
Hipotez Testinin Uygulanması.....388
Nonparametrik Hipotez Testi.....388
Korelasyon Analizi.....389
Varsayımlar.....391
İş Uygulaması: Bahşiş ile Ödenen Hesap Arasındaki İlişkinin İncelenmesi.....392
Korelasyon Varsayım Kontrolü.....393
Korelasyon Katsayısı Hipotez Testi.....394
Nonparametrik Hipotez Testi.....394
---Data Preprocessing---.....395
Veri Ön İşlemeye Genel Bakış.....395
Aykırı Değerler (Outliers).....396
Kime Göre Neye Göre Aykırı Gözlem?.....397
Aykırı Değerleri Yakalamak.....400
Aykırı Değer Problemini Çözmek.....403
Silme Yaklaşımı.....403
Ortalama Değerler ile Doldurma.....404
Baskılama Yöntemi.....405
Çok Değişkenli Aykırı Gözlem Analizi.....407
Local Outlier Factor.....407
Silme Yöntemi.....409
Baskılama Yöntemi.....410
Eksik Gözlem Analizi (Missing Data Analysis).....412
Eksik Veriyi Direk Silmenin Zararları.....413
Eksik Veri Türleri Nelerdir?.....414
Eksik Verinin Rassallığının Testi.....414

Eksik Veri Problemi Nasıl Giderilir?.....414
Eksik Veri Hızlı Çözüm.....416
Eksik Değerlerin Direk Silinmesi.....417
Basit Değer Atama.....418
Eksik Değerlerin Saptanması (Özet).....420
Eksik Veri Yapısının Görselleştirilmesi.....421
Silme Yöntemi.....424
Değer Atama Yöntemleri.....427
Sayısal Değişkenlerde Atama.....427
Kategorik Değişken Kırılımında Değer Atama.....430
Kategorik Değişkenler için Eksik Değer Atama.....432
Tahmine Dayalı Değer Atama Yöntemleri - KNN & Random Forest & EM 434
KNN.....434
Random Forests.....435
EM.....436
Değişken Standardizasyonu (Veri Standardizasyonu).....438
Standardizasyon.....439
Normalizasyon.....439
Min-Max Dönüşümü.....440
Değişken Dönüşümleri.....440
Binarize Dönüşüm.....441
1-0 Dönüşümü.....442
“1 ve Diğerleri (0)” Dönüşümü.....443
Çok Sınıflı Dönüşüm.....444
One-Hot Dönüşümü ve Dummy Değişken Tuzağı.....445
Dummy Değişken Tuzağı.....445
Sürekli Değişkeni Kategorik Değişkene Çevirme.....447
Değişkeni Index'e, Index'i Değişkene Çevirmek.....447
---Machine Learning---.....449
Machine Learning I – Giriş.....449
Makine Öğrenmesine Giriş.....449
Terminoloji.....450
Öğrenme Türleri.....450
Problem Türü.....450

Değişken ve Ölçek Türleri.....450
Test-Train Ayrımı.....451
Feature Engineering.....451
Variable Selection.....451
Model Seçimi.....451
Overfitting.....452
Deterministik Modeller vs Stokastik Modeller.....452
Doğrusal Modeller vs Doğrusal Olmayan Modeller.....452
----------.....453
---Machine Learning Days---.....453
MLD-Data Visualization.....453
Veri Setinin Hikayesi.....455
Veri Görselleştirme.....458
Relational Plots with Matplotlib.....458
Scatter plot with Subplots.....459
Histogram.....460
Bar Plot.....461
Figure Kaydetme.....462
Seaborn.....462
Count Plot & Cat Plot.....463
Scatter Plot.....464
Line Plot.....465
Scatter Subplots.....465
Heatmap.....468
Categoric Plot.....469
Box Plot.....469
Data Visualization Quiz.....471
MLD-Data Preprocessing.....473
1. Adım: Büyük resime bakın!.....473
NaN kontrolü.....474
2. Adım: Manipülasyona Başlayın!.....474
Bilgi içermeyen kolonların kaldırılması.....474
Eksik değerlerin halledilmesi.....476
1. En kolay teknik.....477
Manuel.....477

Scikit....477
2. Enterpolasyon....478
3. En yakın komşular....480
3. Adım: Eksikleri tamamlayın!....481
1. Standardization....482
1.1 Standard Scaler....483
1.2 MinMax Scaler....484
Not:....485
2. Kategorik Değerlerin Ayrıştırılması....485
2.1 Label Encoding....485
2.2 One Hot Encoding....486
3. Kuantizasyon veya Binning....488
Feature Selection....488
Veri Seti....489
Feature Importance....491
Correlation Matrix....493
Data Preprocessing Quiz....496
MLD-Models....500
Regression Analysis....500
Classification Analysis....501
Binary Classification....501
Multi-Class Classification....501
Linear Regression....501
Multiple Linear Regression....504
Polinomial Regression....505
Logistic Regression....507
k-Nearest Neighbor....509
Support Vector Machines....511
Classification....512
Regression....514
Desicion Trees....518
Classification....518
Cart....520
Regression....522
Ensemble Methods & Random Forest....525

Bagging and Pasting.....526
Models Quiz.....527
----------.....532
---Kaggle Master---.....532
Intro to Machine Learning.....532
How Models Work (Modeller Nasıl Çalışır?).....532
Giriş.....532
Decision Tree’nin Geliştirilmesi.....533
Basic Data Exploration (Basit Veri Keşfi).....535
Verilerinizi Tanımak için Pandas Kullanımı.....535
Interpreting Data Description (Verilerin Yorumlanması).....536
Excercise: Explore Your Data.....537
Step 1: Loading Data (Veri Yükleme).....537
Step 2: Review The Data (Verileri Gözden Geçirme).....537
Verilerinizi Düşünün.....538
Your First Machine Learning Model.....540
Selecting Data for Modeling (Modelleme için Veri Seçmek).....540
Selecting The Prediction Target (Tahmin Hedefini Seçme).....541
Choosing "Features" (Özellik Seçimi).....541
Building Your Model (Model Oluşturma).....543
Exercise: Your First Machine Learning Model.....545
Özet.....545
Exercises.....545
Step 1: Prediction Target Belirleme.....545
Step 2: X Oluştur.....546
Verinin İncelenmesi.....548
Step 3: Modelin belirlenmesi ve fit edilmesi.....549
Step 4: Tahmin Yapma.....549
Model Validation (Model Geçerliliği).....550
Model Validation Nedir?.....550
The Problem with "In-Sample" Scores.....552
Coding It.....552
Wow!.....553
Exercise: Model Validation.....553
Exercises.....555

Step 1: Split Your Data (Verinizi Ayırın).....555
Step 2: Specify and Fit the Model (Modeli belirleme ve fit etme)....555
Step 3: Make Predictions with Validation Data.....556
Step 4: Calculate the Mean Absolute Error in Validation Data.....557
Underfitting and Overfitting.....558
Farklı Modellerle Deneme.....558
Examples.....561
Sonuç.....562
Exercise: Underfitting and Overfitting.....563
Exercises.....563
Step 1: Compare Different Tree Sizes (Farklı ağaç boyutlarını karşılaştırın).....564
Step 2: Fit Model Using All Data.....564
Random Forests.....566
Introduction.....566
Example.....566
Sonuç.....567
Exercises: Random Forest.....569
Exercises.....569
Step 1: Use a Random Forest.....570
Exercises: Machine Learning Competitions.....571
Introduction.....571
Creating a Model For the Competition.....573
Make Predictions.....573
Quiz: Intro to Machine Learning.....574
Intermediate Machine Learning.....580
Introduction.....580
Exercises.....581
Step 1 : Eveluate Several Models (Birkaç modeli değerlendirin).....582
Step 2: Generate Test Prediction (Test tahminleri oluşturun).....583
Missing Values (Eksik Veriler).....584
Üç Yaklaşım.....584
1 Basit Bir Seçenek: Eksik Değerli Sütunları Düşürme.....584
2 Daha İyi Bir Seçenek: Imputation.....584
3 An Extension To Imputation.....585

Example....585
Define Function to Measure Quality of Each Approach (Her yaklaşımın kalitesini ölçme yaklaşımı)....586
Score from Approach 1 (Drop Columns with Missing Values)....586
Score from Approach 2 (Imputation)....587
Score from Approach 3 (An Extension to Imputation)....588
Sonuç....589
Exercises (Missing Values)....589
Step 1: Preliminary investigation (Ön Soruşturma)....591
Step 2: Drop columns with missing values (Eksik değer içeren sütunları düşürün)....592
Step 3: Imputation....592
Step 4: Generate test predictions....594
Categorical Variables....596
Introduction....596
Üç Yaklaşım....596
1) Drop Categorical Variables....596
2) Label Encoding....596
3) One-Hot Encoding....598
Example....598
Define Function to Measure Quality of Each Approach....600
Score from Approach 1 (Drop Categorical Variables)....601
Score from Approach 2 (Label Encoding)....601
Score from Approach 3 (One-Hot Encoding)....602
En iyi yaklaşım hangisi?....603
Sonuç....603
Exercises: Categorical Variables....604
Step 1: Drop columns with categorical data....605
Step 2: Label Encoding....606
Step 3: Investigating Cardinality (Kardinalite Araştırması)....608
Step 4: one-hot encoding....610
Step 5: Generate test predictions and submit your results....611
Pipelines....612
Introduction....612
Example....612
Step 1: Önişleme Adımlarını Tanımlayın....614

Step 2: Modeli tanımlayın.....615
Step 3: Pipeline Oluşturun ve Değerlendirin.....615
Sonuç.....616
Exercise: Pipelines.....617
Step 1: Performansı Arttırın.....619
Step 2: Test Tahminleri Oluşturun.....621
Cross-Validation.....623
Introduction.....623
Cross-Validation Nedir?.....623
Ne Zaman Cross-Validation Kullanmalıyız?.....624
Example.....625
Sonuç.....626
Exercise: Cross-Validation.....626
Step 1: Write a Usefull Function.....629
Step 2: Test Different Parameter Values.....629
Step 3: Find the Best Parameter Value.....631
XGBoost.....632
Introduction.....632
Gradient Boosting.....632
Example.....633
Parameter Tuning (Parametre Ayarı).....635
n_estimators.....635
early_stopping_rounds.....635
learning_rate.....636
n_jobs.....637
Sonuç.....637
Exercise: XGBoost.....638
Step 1: Model Oluşturun.....640
Step 2: Modelinizi İyileştirin.....641
Step 3: Modeli Kırın.....642
Data Leakage (Veri Sızıntısı).....643
Introduction.....643
Target Leakage.....643
Train-Test Contamination.....644
Example.....644

Sonuç.....647
Exercise: Data Leakage.....647
Quiz: Intermediate Machine Learning.....648
Data Visualization.....654
Hello, Seaborn.....654
Notebook Kurulumu.....654
Veri Yükleme.....655
Verileri İnceleyelim.....656
Plot the Data (Verileri Çizin).....656
Line Charts (Çizgi Grafikleri).....657
Dataset Seçimi.....657
Veri Yükleme.....658
Verileri İnceleyin.....658
Verileri Çizin.....659
Plot a subset of the data (Verilerin alt kümelerini çizme).....660
Exercise: Line Charts.....663
Senaryo.....663
Step 1: Veri Yükleme.....665
Step 2: Verileri İnceleyin.....665
Step 3: Müze Kurulunu İkna Edin.....665
Step 4: Mevsimsel Değerlendirme.....666
Bar Charts ve Heatmaps (Çubuk Grafikleri ve Isı Haritaları.....668
Dataset Seçimi.....668
Verileri İnceleyelim.....669
Bar Chart.....669
Heatmap.....670
Exercise: Bar Charts ve Heatmaps.....672
Senaryo.....672
Step 1: Veri Yükleme.....673
Step 2: Verileri İnceleyin.....673
Step 3: En iyi platform hangisi?.....674
Step 4: Olası tüm kombinasyonları inceleyelim!.....675
Scatter Plots (Dağılım Grafikleri).....677
Verileri Yükleyelim ve İnceleyelim.....677
Scatter Plots.....678

Renk Kodlu Dağılım Grafikleri....679
Exercise: Scatter Plots....681
Senaryo....681
Step 1: Veri Yükleme....682
Step 2: Verileri İnceleyin....682
Step 3: Şekerin rolü....683
Step 4: Daha Yakından Bak....684
Step 5: Chocolate!....685
Step 6: Chocolate Sütununu İnceleyelim....686
Step 7: Herkes çikolatayı sever....687
Distributions (Dağılımlar)....688
Veri Seti Seçimi....688
Veri Yükleme ve İnceleme....689
Histograms....689
Density Plots (Yoğunluk Grafikleri)....690
2D Kde Plots....691
Color-coded plots (Renk Kodlu Grafikler)....692
Exercise: Distributions....695
Senaryo....695
Step 1: Veri yükleme....696
Step 2: Veri inceleme....697
Step 3: Farklılıkları araştıralım....698
Step 4: En işe yarar sütun....699
Choosing Plot Types and Custom Styles (Grafik Türlerini ve Özel Stilleri Seçme)....700
Ne öğrendin?....700
Seaborn ile stilleri değiştirme....701
Exercise....703
Seaborn Stillerini Deneyelim....704
Quiz: Data Visualization....709
Kaggle Master Final Sınavı....714
Kaynaklar....729

Data Science

VERİ BİLİMİNE GİRİŞ

Veri Bilimci, veriden faydalı bilgi çıkarma sürecini yöneten kişidir.

MODERN DATA SCIENTIST
Data Scientist, the sexiest job of 21th century requires a mixture of multidisciplinary skills ranging from an intersection of mathematics, statistics, computer science, communication and business. Finding a data scientist is hard. Finding people who understand who a data scientist is, is equally hard. So here is a little cheat sheet on who the modern data scientist really is.
MATH & STATISTICS
Machine learning
Statistical modeling
Experiment design
Bayesian inference
Supervised learning: decision trees, random forests, logistic regression
Unsupervised learning: clustering, dimensionality reduction
Optimization: gradient descent and variants
PROGRAMMING & DATABASE
Computer science fundamentals
Scripting language e.g. Python
Statistical computing package e.g. R
Databases SQL and NoSQL
Relational algebra
Parallel databases and parallel query processing
MapReduce concepts
Hadoop and Hive/Pig
Custom reducers
Experience with xaaS like AWS
DOMAIN KNOWLEDGE & SOFT SKILLS
Passionate about the business
Curious about data
Influence without authority
Hacker mindset
Problem solver
Strategic, proactive, creative, innovative and collaborative
COMMUNICATION & VISUALIZATION
Able to engage with senior management
Story telling skills
Translate data-driven insights into decisions and actions
Visual art design
R packages like ggplot or lattice
Knowledge of any of visualization tools e.g. Flare, D3.js, Tableau

VERİDEN FAYDALI BİLGİ ÇIKARMAK

Data Science Kullanılan Alanlar

- Arkadaş önerileri
- Otomatik fotoğraf etiketlemeleri
- Hedefli içerik pazarlama
- Otomatik mesaj tamamlama
- Hedefli ürün pazarlama
- Tavsiye sistemleri

- Müşteri segmentasyonu
- Kanser/Hastalık teşhisi
- Şirketlerin gelir tahmini ile strateji belirlemesi
- Başvuru değerlendirme sistemleri
- Akıllı portföy yönetimi
- Doğal afet modelleme çalışmaları
- E-Spor Analitiği

- Otonom araçlar
- Nesne tanıma/takip uygulamaları
- Sahte videolar
- Eski resimlerin canlandırılması
- Algoritmaların geliştirdiği resimler/var olmayan kişiler
- Robotlar!

Genel Resim

- **Veri bilimi nedir?**

  Veriden faydalı bilgi çıkarma sürecidir.

- **Veri bilimci nedir?**
  Çeşitli araçlar kullanarak veriden faydalı bilgi çıkarma sürecini yöneten kişidir.

- **Veriden öğrenerek ortaya çıkan sisteme (fonksiyon vb.) ne denir?**
  Makine öğrenmesi modeli, istatistiksel model, yapay zeka sistemi, model.

- **Makine öğrenmesi algoritmaları ne işe yarar?**
  Makine öğrenmesi büyük miktarlarda veri içerisinde yer alan ve insan olarak öğrenmemizin mümkün olmadığı yapıları öğrenme işine yarar.

- **Neden biz öğrenmiyoruz da algoritmalardan bir şeyler öğrenmesini bekliyoruz?**
  İnsan olarak bizlerin, araç fiyatının ne olabileceğini bilmemiz için ya **yıllardır bu işi yapan bir uzman olmamız** ya da **yüzbinlerce ilan arasında inceleme yapıp** aracın fiyatının ne olabileceğini öğrenmeye çalışmamız gerekiyor. Bu işlemi insan olarak yaptığımızda hasar durumu, KM gibi faktörlerin fiyata olan etkisini **öğreniyor** oluyoruz. İşte bu işlem bir insan olarak çok kolay ve sürekli mümkün olmayacağı için bu iş programatik olarak algoritmalara yaptırılır.

Data Science Proje Döngüsü

Veri Bilimine Giriş Alıştırmalar – 1

Soru 1:

Aşağıdakilerden hangisi günümüzün yeni petrolü olarak tanımlanmaktadır?

- Sosyal medya
- Veri
- İstatistik
- İnternet

Soru 2:

Aşağıdakilerden hangisi yapay zekayı besleyen temel kaynaktır?

- Sosyal medya
- İnternet
- Veri
- Algoritmalar

Soru 3:

Andrew Ng tarafından ifade edilen günümüzün yeni elektriği nedir?

- Veri
- Algoritmalar
- Yapay Zeka
- Veri Bilimi

Soru 4:

Veriden faydalı bilgi çıkarma sürecine ... denir? Boşluğa hangi ifade gelmelidir?

- Makine öğrenmesi
- Veri bilimi
- a) Yapay zeka
- İstatistik

Soru 5:

Aşağıdakilerden hangisi veri bilimi süreci bileşenlerinden değildir?

- Veri kaynakları
- Bilgi
- Veri işleme
- Aksiyon

Soru 6:

Bir bilgisayarın veya bilgisayar kontrolündeki bir robotun çeşitli faaliyetleri zeki canlılara benzer şekilde yerine getirme kabiliyetine ... denir.

- Makine öğrenmesi
- Veri bilimi
- Yapay zeka
- Derin öğrenme

Soru 7:

Aşağıdakilerden hangisi bir yapay zeka uygulaması değildir?

- (●) Bir dizi matematiksel işlem gerçekleştiren program
- ( ) Belirli görevler için eğitilmiş robotlar
- ( ) Veri içerisindeki yapıları öğrenip genelleme yeteneği kazanmış bir fonksiyon
- ( ) Medikal görüntüler üzerinden hastalık tahmini yapan bir program

Soru 8:

Yapay zeka çağında hayatta kalmak ... ve ... yeteneklerine bağlıdır.

- ( ) Veri bilimi ve yapay zeka
- (●) Veri analitiği ve analitik düşünce becerileri
- ( ) İstatistik ve programlama
- ( ) Programlama ve makine öğrenmesi

Soru 9:

Veri Bilimi çok disiplinli bir alan olarak ele alındığında aşağıdakilerden hangisi veri bilimini meydana getiren *ana unsurlardan* değildir.

- (●) Programlama
- ( ) Bilgisayar Bilimleri
- ( ) İş-Sektör Bilgisi
- ( ) Matematik-İstatistik

Soru 10:

Belirli bir sektörde meydana gelen bilgi birikimine ne denir?

- ( ) Veri Analitiği
- ( ) Uzmanlık
- (●) İş Bilgisi
- ( ) İş Dalı

Veri Bilimine Giriş Alıştırmalar – 2

Soru 1:

Aşağıdakilerden hangisi veri analitiği türlerinden değildir?

- Tahminsel Analitik
- Betimleyici Analitik
- Sektörel Analitik
- Yönergeli Analitik

Soru 2:

"Neden olmuş" sorusuna yanıt arayan veri analitiği türü aşağıdakilerden hangisidir?

- Teşhis/Tanı Analitiği
- Betimleyici Analitik
- Tahminsel Analitik
- Yönergeli Analitik

Soru 3:

Aşağıdaki veri analitiği türlerinden hangisi diğerlerine göre daha kolay uygulanabilmektedir.

- Betimleyici Analitik
- Teşhis/Tanı Analitiği
- Yönergeli Analitik
- Tahminsel Analitik

Soru 4:

Aşağıdakilerden hangisi "ne olmalı" / "nasıl olmalı" sorusuna yanıt arar?

- ○ Betimleyici Analitik
- ○ Teşhis Analitiği
- ● Yönergeli Analitik
- ○ Tanı Analitiği

Soru 5:

Aşağıdakilerden hangisi "ne olacak" sorusuna yanıt arar?

- ○ Betimleyici Analitik
- ○ Teşhis/Tanı Analitiği
- ○ Yönergeli Analitik
- ● Tahminsel Analitik

Soru 6:

Bir ürüne ait satış sayılarının aylara göre görselleştirilmesi hangi veri analitiği türüne girer?

- ○ Normatif Analitik
- ○ Teşhis/Tanı Analitiği
- ● Betimleyici Analitik
- ○ Yönergeli Analitik

Soru 7:

Yıl sonu elde edilecek gelirin ne olacağının araştırılması hangi veri analitiği türüne girer?

- (●) Tahminsel Analitik
- ( ) Betimleyici Analitik
- ( ) Teşhis/Tanı Analitiği
- ( ) Yönergeli Analitik

Soru 8:

ABD başkanlık seçimlerinde en önemli rolü oynayan iki kavram aşağıdakilerden hangisi olabilir?

- ( ) Veri bilimi ve yapay zeka
- (●) Veri analitiği ve analitik düşünce becerileri
- ( ) Veri ve tahminsel analitik
- ( ) Sosyal medya ve yapay zeka

Soru 9:

Aşağıdakilerden hangisi günümüz dünyasında veri bilimi ve yapay zekayı bu kadar önemli hale getiren sebepler birisi olamaz?

- ( ) Anlamlı hale getirilmeyi bekleyen verinin hızla artması
- ( ) Otonomlaştırılması gereken iş alanları
- ( ) Şirketlerin gelir ya da süreçlerinde iyileştirme ihtiyaçları
- (●) Yeni istihdam alanlarının aranması

Soru 10:

Bir şirket gelirlerinde meydana gelen düşüşlerin nedenlerini veriye bakarak anlamak istiyor bu durumda hangi veri analitiğini kullanması gerekir?

- (●) Teşhis/Tanı Analitiği
- ( ) Betimleyici Analitik
- ( ) Normatif Analitik
- ( ) Tahminsel Analitik

Data Literacy (Veri Okuryazarlığı)

Veri Okuryazarlığı Nedir?

Günlük hayatta veriyle temas ettiğimiz ilk anlardaki basit veri yorumlama kabiliyetleridir.

Veri okuryazarlığı; her türden veri tipini, değişken ve ölçek türlerini tanımlayabilme, betimsel istatistikleri ve istatistiksel grafikleri kullanarak veri değerlendirebilme yeteneğidir.

Population and Sample (Popülasyon ve Örneklem)

Verinin tamamına **popülasyon,** veriyi temsil eden alt kümeye ise **örneklem** denir.

Observation Unit (Gözlem Birimi)

Gözlem birimi, araştırmada gözlemlediğimiz birimlerdir.

Örneğin; anket yapılmak üzere mikrofon uzatılan her bir birey, bir gözlem birimidir.

Variables and Variable Types (Değişken ve Değişken Türleri)

Değişken: Birimden birime farklı değerler alan niceliktir

Örneğin; Veri setindeki kolonlar.

Değişken Türleri:

- Sayısal Değişkenler (nicel, kantitatif)
- Kategorik Değişkenler (nitel, kalitatif)

"Rütbe" kategorik değişkeninin sınıfları:
Onbaşı < Yüzbaşı < Binbaşı <Albay

Burada değişkenin kategorilerinin sınıfları arasında bir fark var, burada devreye ölçek türleri giriyor.

Scales of Measurement (Ölçek Türleri)

Bir değişkenin değerlerini insan olarak okuyup anlayabilmemiz adına bunu ölçmemiz gerekiyor.

Ölçek Türleri

- Sayısal değişkenler için: Aralık ve Oran
- Kategorik değişkenler için: Nominal ve Ordinal

Sayısal Değişkenler

Başlangıç noktası sıfır olmayan sayısal değişkenlerin ölçek türü aralıktır.

Başlangıç noktasını sıfır kabul eden sayısal değişkenlerin ölçek türü orandır.

Kategorik Değişkenler

Metin formatında, karakterlerden oluşan. Programlama dilinde string tipinde değişkenlerdir. Sınıfları arasında fark olmayan değişkenler **nominal** değişkenlerdir.

"Cinsiyet" kategorik değişkendir.

"Kadın", "Erkek" ise bu kategorik değişkenin sınıflarıdır.

"Kadın" ve "Erkek" sınıfları arasında fark olmadığı için bu değişken nominal ölçek türüne sahiptir

Sınıfları arasında fark olması durumu ise **ordinal** değişkenler ile tanımlanabilir.

"Rütbe" kategorik bir değişkendir.

Bu kategorik değişkenin sınıfları:

Onbaşı < Yüzbaşı < Binbaşı < Albay

Değişkenin sınıfları arasında fark olduğu için "Rütbe" değişkeni ordinaldir.

"Eğitim Durumu " kategorik bir değişkendir.

Bu kategorik değişkenin sınıfları:

İlkokul < Lise < Üniversite < Lisansüstü

Değişkenin sınıfları arasında fark olduğu için "Rütbe" değişkeni ordinaldir.

Merkezi Eğilim Ölçüleri

Arithmetic Mean (Aritmetik Ortalama)

Bir seride (değişkende) yer alan tüm değerlerin toplanması ve birim sayısına bölünmesi ile elde edilen istatistiktir.

Median (Medyan)

Bir seriyi küçükten büyüğe veya büyükten küçüğe sıraladığımızda tam orta noktadan seriyi iki eşit parçaya ayıran değere medyan adı verilir.

n tek:

13, 10, 15, 12, 17

10, 12, 13, 15, 17

$$\frac{(n+1)}{2} = \frac{(5+1)}{2} = 3.terim$$

$$Medyan = 13$$

n çift:

13, 10, 15, 12, 17, 13

10, 12, 13, 13, 15, 17

$$Medyan = \frac{\left(\frac{n}{2}\right).terim + \left(\frac{n}{2}+1\right).terim}{2} = \frac{13+13}{2}$$

$$= 13$$

Aritmetik ortalama, seri dağılımının (değişkenin dağılımının) simetrik olduğu bilindiğinde kullanılabilir.

Simetrik olmaması; Veride aykırı değerlerin olması anlamına gelir. Bu durumda aritmetik ortalama yanıltıcı bir sonuç gösterir. Medyan kullanılmalıdır.

13, 10, 15, 12, 17, 12, 19, 18, 11, 12, 190

13 **28,5**

Yukarıdaki örnekte **13** veri setinin medyanı, **28,5** ise ortalamasıdır. 190 değeri aykırı bir değer olduğu için 28,5 çıkan ortalamayı dikkate almamız bizi yanıltacaktır. Veriler genel olarak 10-19 aralığında dağılmasına rağmen ortalamanın 28,5 olması mantıksızdır.

Bu değişkenin temsili değeri olarak medyan'ı kullanmak doğru olacaktır.

Mode (Mod)

Bir seride (değişkende) en çok tekrar eden değere Mod adı verilir.

Quartiles (Kartiller)

Küçükten büyüğe sıralanan bir seriyi 4 parçaya ayıran değerlere kartiller denir.

Dağılım ile ilgili bilgi almak adına kullanılır.

8, 10, 12, 14, 15, 17, 20

Q1 Q2 Q3

$$Q_1 = \frac{1}{4}(n+1).terim$$

n = Terim sayısı

$$Q_3 = \frac{3}{4}(n+1).terim$$

$$Q_2 = Q_3 - Q_1$$

Merkezi Eğilimin Önemini Anlamak

Ortalama ve Medyanın birbirine yakın olması, o serinin düzgün dağıldığını(homojen) gösterir.

Measure of Dispersion (Dağılım Ölçüleri)

Elimizdeki değişkenin değerlerinin ne şekilde dağıldığını gösteren ölçülerdir.

Range (Değişim Aralığı)

Bir serideki max. değerden min. değeri çıkardığımızda elde ettiğimiz değerdir.

8, 10, 15, 12, 17, 20, 14

Değişim Aralığı = Maksimum Değer – Minimum Değer

Değişim Aralığı = 20 – 8 = 12

Standard Deviation (Standart Sapma)

Ortalamadan olan sapmaların genel bir ölçüsüdür.

$$s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

Aslında bakarsanız standart sapma bir ortalamadır. Ortalamadan olan sapmaların bir ortalamasıdır.

	Popülasyon (anakitle)	Örneklem
Ortalama	μ	$\bar{x}$
Standart Sapma	σ	s

Kazanç(x_i)	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
12	(12-24) = -12	144
15	(15-24) = -9	81
20	(20-24) = -4	16
30	(30-24) = 6	36
45	(45-24) = 21	441
22	(22-24) = -2	4
Toplam:	0	722

$$s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$\bar{x} = (12+15+20+30+45+22)/6 = 24$$

$$s = \sqrt{\frac{1}{6}\,722}$$

$$s = 10.97$$

Variance (Varyans)

Standart sapmanın karesidir
(Ortalamadan olan sapmaların karelerinin ortalamasıdır)

$$s = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

$$s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

Birden fazla değişkenin dağılımını birbiriyle kıyaslamak için kullanmak istediğimizde varyans kullanabiliriz.

Kazanç(x_i)	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
12	(12-24) = -12	144
15	(15-24) = -9	81
20	(20-24) = -4	16
30	(30-24) = 6	36
45	(45-24) = 21	441
22	(22-24) = -2	4
Toplam:	0	722

$$s^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

$$\bar{x} = (12+15+20+30+45+22)/6 = 24$$

$$s^2 = \frac{1}{6}\,722$$

$$s^2 = 120{,}34$$

Skewness (Çarpıklık)

Çarpıklık, bir değişkenin dağılımının simetrik olamayışıdır.

Negatif Çarpık Pozitif Çarpık

Pearson Çarpıklık Katsayısı:

$$\frac{3(\bar{x} - medyan)}{standart\ sapma}$$

PÇK < 0 → Negatif çarpık(soldan)

PÇK > 0 → Pozitif çarpık(sağdan)

PÇK = 0 → Simetrik

Kazanç(x_i)	$(x_i - \bar{x})$	$(x_i - \bar{x})^2$
12	(12-24) = -12	144
15	(15-24) = -9	81
20	(20-24) = -4	16
30	(30-24) = 6	36
45	(45-24) = 21	441
22	(22-24) = -2	4
Toplam:	0	722

$\bar{x} = 24 \quad s = 10{,}97$

Medyan:

12, 15, 20, 22, 30, 45

(20+22)/2 = 21

$$\text{PÇK} = \frac{3\,(24 - 21)}{10{,}97}$$

$$\text{PÇK} = 0{,}82$$

Pearson Çarpıklık Katsayısı:

$$\frac{3(\bar{x} - medyan)}{standart\ sapma}$$

Dağılım simetrik değildir. 0'dan büyük olduğu için **pozitif çarpık** (sağa çarpık)'tır. 1'e yakın olduğu için yüksek sağa çarpıktır.

Kurtosis (Basıklık Ölçüsü)

Dağılımın basıklığını / sivriliğini gösterir.

$$Basıklık\ Katsayısı = \frac{m_4}{s^4}$$

$$\frac{\sum_{i=1}^{n}(x_i - \bar{x})^4}{n}$$

Standart sapmanın 4. kuvveti

BK = 3 ise dağılım standart normal dağılıma uygundur.
BK > 3 ise dağılım sivridir.
BK < 3 ise dağılım basıktır.

Kazanç(x_i)	($x_i - \bar{x}$)	$(x_i - \bar{x})^4$
12	(12-24) = -12	20736
15	(15-24) = -9	6561
20	(20-24) = -4	256
30	(30-24) = 6	1296
45	(45-24) = 21	194481
22	(22-24) = -2	16
Toplam:	0	223346

$$Basıklık\ Katsayısı = \frac{m_4}{s^4}$$

$$m_4 = 223346/6 = 37224{,}33$$

$$s^4 = (10{,}97)^4 = 14481.93$$

$$\frac{m_4}{s^4} = \frac{37224{,}33}{14481.93} = 2{,}57$$

2,57 < 3 olduğundan dağılım basıktır.

Statistical Thinking Models (İstatistiksel Düşünce Modelleri)

> Veri okuryazarlığından veri analitiğine giden yolu modelleyen yol göstericilerdir

Analitik düşünce becerilerini, veri analitiği kapsamında belirli bir programatik şema ile ele almaya sağlayan akademik modellerdir.

- Ben- Zvi ve Friedlander (1997)
- Jones ve diğerleri (2000)

- Wild ve Pfannkuch (1999)
- Hoerl ve Snee (2001)

- Mooney (2002)

Mooney Modeli

Kabaca 4 basamaktan oluşur.

Verinin Tanımlanması

▼

Verinin Organize Edilmesi ve İndirgenmesi

▼

Veri Gösterimi

▼

Verinin Analiz Edilmesi ve Yorumlanması

Statistical Thinking Levels

- Kişiye Özgülük (Seviye 1)
- Geçici (Seviye 2)
- Nicel (Seviye 3)
- Analitik (Seviye 4)

Verinin Tanımlanması

Veri okuryazarlığı bölümünde ele almış olduğumuz tüm konuları Mooney modelinde yer alan 4 basamakta değerlendirmiş ve pekiştirmiş olacağız.

Bir şirket internet kullanımı ile ilgili yaptığı araştırmada şu açıklamaları yayınlamıştır: Anketi yanıtlayan 2000 kişinin %43,4'ünü erkekler, %66,4'sini kadınlar oluşturmaktadır. Anketi yanıtlayanların %80'i 15-27 yaş aralığındadır. Kadınların %72'si İngilizce bilmektedir. Erkekler günde ortalama 3 saat, kadınlar ise 4 saat internette zaman geçirmektedir. Erkeklerin %72'si, kadınların ise %75'i üniversite mezunudur.

- **Çalışmada ölçülmeye çalışılan değişkenler nelerdir ?**
  - Cinsiyet
  - Yaş
  - İngilizce Bilme
  - İnternette Geçirilen Zaman
  - Eğitim Durumu
- **Değişkenlerin türleri nelerdir ?**
  - Cinsiyet - Kategorik
  - Yaş - Sayısal
  - İngilizce Bilme - Kategorik
  - İnternette Geçirilen Zaman - Sayısal
  - Eğitim Durumu - Kategorik
- **Belirlediğiniz değişkenlerin hangi ölçekle ölçüldüğünü belirtiniz.**
  - Cinsiyet – Kategorik - Nominal
  - Yaş – Sayısal - Oran
  - İngilizce Bilme – Kategorik - Nominal
  - İnternette Geçirilen Zaman – Sayısal - Oran
  - Eğitim Durumu – Kategorik - Ordinal
- **Metni ilk okuduğunuzda dikkatinizi çeken bir anormallik var mı?**
  Yüzdelik değerlerde anormallik var. Ve İngilizce bilme konusunda anlam karmaşası yaşanabilir. Yeterince açıklayıcı anlatılmamış.
- **Bu veri seti üzerinde birkaç dakika yorum yapabilir misiniz?**

Verilerin Organize Edilmesi ve İndirgenmesi

Tabloya nasıl bir düzenleme yapılmalıdır, kısaca açıklayınız.

Saat	Kazanç (TL)	Saat	Kazanç(TL)
06:50	12	12:48	17
07:10	5	13:44	10
07:15	8	14:10	5
07:30	22	17:22	55
08:42	14	18:05	2
09:21	9	19:48	16
09:26	4	20:22	25
10:02	18	20:49	21
11:56	12	22:40	12

Buradaki verilerin toplulaştırma yapılarak belirli saat aralıklarındaki toplam ücretleri yazabiliriz. Ham haliyle okunması zor olacaktır.

Saatler	Kazanç (TL)
06:00 - 09:00	61
09:00 - 12:00	43
12:00 - 15:00	32
15:00 - 18:00	55
18:00 - 21:00	64
21:00 - 24:00	12

Verinin Gösterimi

Bir medya şirketinde, ortalama ev fiyatları için oluşturulmuş aşağıdaki grafik ve yorumlardan hangisinin seçileceği tartışma konusu olmuştur.

"Ev fiyatlarında büyük artış!"

Grafik 1

"Ev fiyatları geçen yıla göre artış göstermiştir"

Grafik 2

- **Grafikler hangi konuda bilgi vermektedir ?**
  Grafikler, ev fiyatlarındaki artışı ortaya koymak için kullanılmıştır.
- **Hangi grafik ve yorum doğrudur ? Seçin sebebinizi açıklayınız.**
  İki grafik karşılaştırıldığında öncelikle bir problem var. Grafik 1'in Y ekseni 79.000'den başlamış. Grafik 2'nin ise başlangıç noktası 0.
  Grafik gösterme tekniklerinde, eksenlere konulacak olan değişkenin grafiksel anlamda ölçeklendirilmesi, eldeki verinin sunumunda suistimale en açık olan konulardan birisidir. Eksenlerin başlangıç noktaları 0 olmak durumundadır.

Grafik 1'in ölçeği biner biner artıyor. Grafik 2'nin ölçeği ise 10.000 10.000 artıyor.
Ev fiyatları göz önünde bulundurulduğunda Grafik 1'in ölçeklendirmesi pek mantıklı olmayacaktır.

Verilerin Analiz Edilmesi ve Yorumlanması

Survivor yarışması için yeni dönem yarışmacıları seçilecektir. Ünlüler ve gönüllüler olarak yarışacak olan iki grubun fiziki ve ruhsal dayanıklılığının ölçülmesi için bir ölçek geliştirilmiş ve grupların dayanıklılıkları ölçülüp aşağıdaki değerler elde edilmiştir.

	Ortalama	**Standart Sapma**
Gönüllüler	70	3
Ünlüler	74	8

Uzun sürecek bir yarışma, açlık, doğayla mücadele ve her türlü çekişmenin olacağı bu ortamda hangi grupla beraber çalışıp liderlik etmek isterdiniz? Sebebiyle beraber açıklayınız.

Takım çalışmasının çok önemli olduğu bir yarışma olduğunu anlıyoruz.

Bu koşulları göz önünde bulundurduğunuzda genel ortalaması diğer gruba göre yüksek olan bir grupla mı çalışmak isterdiniz?

Yoksa grup içi davranış biçimleri birbirine daha yakın bir grupla mı çalışmak isterdiniz?

Veri Okuryazarlığı Alıştırmalar – 1

Soru 1:

Veri okuryazarlığı günlük hayatta veri ile temas ettiğimiz ilk anlardaki basit ... kabiliyetleridir.

Yukarıdaki cümlede boş bırakılan bölüme aşağıdakilerden hangisi gelmelidir?

- Okuma
- Ölçme
- Veri yorumlama
- Hesaplama

Soru 2:

Aşağıdakilerden hangisi veri okuryazarlığı ile ilişkili değildir?

- Veriyi tanımak
- Değişkenleri tanımak
- Ölçek türlerini tanımak
- Algoritmik bakış açısına sahip olmak

Soru 3:

Araştırmacının ilgilendiği kitlenin tamamına ... denir.

Yukarıdaki cümlede boş bırakılan bölüme aşağıdakilerden hangisi gelmelidir?

- Popülasyon (ana kitle)
- Gözlem birimleri
- Örneklem
- Örnek

Soru 4:

Popülasyon içerisinden popülasyonu temsil etmesi amacıyla çekilen alt kümeye ... denir.

Yukarıdaki cümlede boş bırakılan bölüme aşağıdakilerden hangisi gelmelidir?

- Gözlem
- Popülasyon
- Örneklem
- Değişken

Soru 5:

"Cinsiyet" isimli değişkenin türü aşağıdaki değişken türlerinden hangisidir?

- Sayısal değişken
- Kategorik değişken
- Ordinal değişken
- Nominal

Soru 6:

Bir web sayfasında geçirilen süreyi ifade eden "SURE" isimli değişkenin türü aşağıdakilerden hangisidir?

- Sayısal değişken
- Kategorik değişken
- Ordinal değişken
- Nominal Değişken

Soru 7:

Aşağıdakilerden hangisi herhangi bir değişken türü için *ölçek türü* değildir.

- Aralık
- Oran
- Ordinal
- Nitel

Soru 8:

Nominal ölçek türü ... değişkenler için bir ölçek türüdür.

- Sayısal
- Sürekli
- Kategorik
- Sayısal

Soru 9:

Kişilerin eğitim durumunu ifade eden "Eğitim Durumu" isimli değişkenin ölçek türü aşağıdakilerden hangisidir?

- Nominal
- Ordinal
- Kategorik
- Sayısal

Soru 10:

"Cinsiyet" isimli değişkenin ölçek türü aşağıdakilerden hangisidir?

- Oran
- Aralık
- Nominal
- Ordinal

Veri Okuryazarlığı Alıştırmalar - 2

Soru 1:

Aşağıdakilerden hangisi merkezi eğilim ölçülerinden değildir?

- Aritmetik ortalama
- Medyan
- Standart sapma
- Mod

Soru 2:

Aşağıdakilerden hangisi dağılım ölçülerinden değildir?

- Değişim aralığı
- Standart sapma
- Varyans
- Aritmetik ortalama

Soru 3:

Bir serideki tüm değerlerin küçükten büyüğe sıralanması sonrasında serinin ortasında kalan değere ne denir?

- Mod
- Medyan
- Aritmetik Ortalama
- Değişim aralığı

Soru 4:

Bir seri küçükten büyüğe ya da büyükten küçüğe sıralandığında seriyi 4 eşit parçaya ayıran üç değere ne denir?

- Mod
- Medyan
- Aritmetik ortalama
- Kartil

Soru 5:

Terimlerin ortalamadan olan sapmalarının genel ölçüsü nedir?

- Varyans
- Standart sapma
- Değişim aralığı
- Ortalama

Soru 6:

130, 10, 15, 12, 17, 13, 12, 19, 18, 11, 12, 10, 209

Yukarıda verilen seri için hangi merkezi eğilim ölçüsünün kullanılması daha uygundur?

- Mod
- Medyan
- Aritmetik ortalama
- Varyans

Soru 7:

11, 10, 13, 12, 10, 8, 15

Yukarıda verilen serinin modu kaçtır?

- 11
- 10
- 13
- 14

Soru 8:

Pearson Çarpıklık Katsayısı değerinin 0 olması ne ifade etmektedir?

- Dağılım basıktır
- Dağılım çarpıktır
- Dağılım asimetriktir
- Dağılım simetriktir

Soru 9:

Basıklık Katsayısı değerinin 1,8 olması ne ifade etmektedir?

- Dağılım standart normal dağılıma neredeyse uygundur
- Dağılım simetriktir
- Dağılım sivridir
- Dağılım basıktır

Soru 10:

"Ben bir yatırımcı olarak finansal hareketler açısından değişkenliği diğerlerine göre daha az olan şirketlere yatırım yaparım" ifadesinin istatistik anlamında karşılığı aşağıdakilerden hangisi olabilir?

- Ortalaması çok değişmeyen şirketler
- Diğerlerine göre varyansı ya da standart sapması daha düşük olan şirketler
- Diğerlerine göre ortalaması ve medyanın değişken olan şirketler
- Diğerlerine göre varyans ya da standart sapması yüksek olan şirketler

----Python Programlama----

- Python, Google tarafından destekleniyor.
- Python'ın yorumlayıcı özelliği vardır. Etkileşim özelliğine sahiptir. (Soru-cevap mantığıyla çalışır.)
- High Level bir programlama dili.
- OPP (nesneye dayalı) ve FP(Fonksiyonel programlama).

Temel Hareketler

- Spyder'da seçili alanı F9 tuşu ile çalıştırabiliriz.
- Python programlama dilinde oluşturulan her şey bir nesnedir.
- Yorum satırı oluşturmak için satır başına # koyarız.
- #%% ..... #%% noktalı alana yazdığımız bölüm bizim için bir section olur. shift+enter yaparak çalıştırdığımızda hangi section'daysak o section çalışır. Bir .py dosyasındaki farklı kod bölümlerini ayırmak için kullanılır.

Integer, Float ve String

Integer = 9 gibi ondalıksız sayılar.

Float = 9.2 gibi ondalıklı sayılar.

String = Karakter dizileri. "Çift tırnak" veya 'Tek tırnak' içinde yazılır.

Type = type() içersine yazılan nesnenin tipini verir.

```python
print("Hello AI Era")

#type komutu icerisine yazdigimiz nesnenin tipini verir.
type(9) #integer
type(9.2) #float
type("Recep Aydoğdu") #string

###################

type("123") #bunun da ciktisi str olacaktir.

"a"+"a"

"a" " a"

"a"*3

"a"/3 #type error hatasi

"a "*5
```

"a"+"a" ✉ aa
"a""a" ✉ aa
"a"*3 ✉ aaa
"a"-"b" ✉ TypeError alırız. Bu operatör sadece numeric ifadelerde kullanılır.
"a"/3 à TypeError

String Metodları

len()= içerisine yazılan değişkenin uzunluğunu verir.

```
# STRING METODLARI - len()

gel_yaz="gelecegi_yazanlar"

#del mvk #degiskeni silmek icin del kullaniriz. kullandiktan sonra
        # yorum satiri haline getirilmelidir.

a=99
b=10

type(a/b) # a/b=9.9 olacagindan tipi float olur.

len(gel_yaz) # gel_yaz degiskeninin icersindeki string'in krktr uzunlugunu verir.
```

upper() & lower() =

```
#upper() & lower() fonksiyonlari

gel_yaz.upper() #stringi buyuk harflere cevirir.

gel_yaz.lower() #stringi kucuk harflere cevirir.
```

isupper() & islower() =

```
#isupper() & islower() fonksiyonlari

gel_yaz.isupper() #buyuk harf mi? sorusu sorar. T or F getirir.
gel_yaz.islower() #kucuk harf mi? sorusu sorar.

B = gel_yaz.upper() #B degiskenine buyuk harfli gel_yaz atadik.

B.isupper()

Dnm="AsDfGhGgGgG"

Dnm.isupper()
Dnm.islower()   #ikisi de false getirir.
```

replace() =

```
# replace() bir karakteri baska bir karakter ile degistirmek icin kullanilir.

gel_yaz.replace("a","ı")
```

replace(“eski_karakter”,”yeni_karakter”)

gelecegi_yazanlar ✉gelecegi_yızınlır

strip() = Karakter kırpma işlemleri

```
41   # strip() Karakter kirpma islemleri
42
43   gel_yaz= " gelecegi_yazanlar " #basinda ve sonunda bosluk var
44   gel_yaz.strip() #varsayilan olarak bosluklari siler.
45
46   gel_yaz="*gelecegi_yazanlar*" # basina ve sonuna * ekledik.
47   gel_yaz.strip("*") # *(yildiz) arasindaki ifadeyi kirpar.
48
```

dir() =

```
49   # dir() icersine yazdigimiz veri tipi icin kullanilabilir metodlari verir.
50
51   dir(gel_yaz)
52                #ikisi de ayni sonucu verir.
53   dir(str)
```

capitalize() = İlk harfi büyütür.

gel_yaz.capitalized()

title() = Her kelimenin ilk harfini büyütür.

gel_yaz.title()

Substring = Alt küme işlemleri

```
57   # Substring: string ifadeleri ile alt kume islemleri.
58
59   gel_yaz[0] # 0 index'li ifadeyi getirir.
60
61   gel_yaz[0:3] # 0'dan basla 3'e kadar getir.
62
```

Type Dönüşümleri

```
63   #TYPE DONUSUMLERİ
64
65   toplama_bir=input() #input ile kullanıcıdan veri alırız.
66   toplama_iki=input() #kullanıcıdan aldıgımız veri str tipindedir.
67
68   toplama_bir+toplama_iki # 10+20 --> '1020' çıktısı verir.
69                           # bunu engellemek için type dönüşümü yapmalıyız.
70   int(toplama_bir)+int(toplama_iki) #tip donusumlerini bu sekilde yapariz.
71
72   int(12.4) #float to int --> 12
73   float(12) #int to float --> 12.0
74   str(12)   #int to str   --> '12'
```

print() fonksiyonu

print("gelecegi","yazanlar") ✉ gelecegi yazanlar

print("gelecegi","yazanlar",sep = ("_")) ✉ gelecegi_yazanlar

```
76   #Print fonksiyonu
77
78   print("gelecegi","yazanlar")
79
80   print("gelecegi","yazanlar",sep = "_") #sep argumani araya gelecek degeri secmemize olanak saglar.
81
82   ?print #print fonksiyonu ile kullanabilecegimiz argumanlari verir.
83
```

Python Programlama Alıştırmalar – 1

Soru 1:

Kod bloğu içerisinde yapılan bir işlemin sonucunu ekrana bastırmak için hangi fonksiyon kullanılır?

- len()
- print
- print()
- def

Soru 2:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
print("uzaya", "git", sep = "**")
```

- uzaya ** git
- uzaya git
- uzaya_git
- uzaya**git

Soru 3:

Aşağıdaki ifadelerden hangisi sayı (float ya da integer) değildir?

- 64
- 2.3
- "9"
- 2/10

Soru 4:

type() fonksiyonu ne için kullanılmaktadır?

- ○ Değişken dönüştürmek
- ● Tip sorgulamak
- ○ Yazdırmak
- ○ Fonksiyon tanımlamak

Soru 5:

`type(4)` **kodunun çıktısı aşağıdakilerden hangisidir?**

- ○ 4
- ○ float
- ○ str(4)
- ● int

Soru 6:

`type(3.14)` **kodunun çıktısı aşağıdakilerden hangisidir?**

- ○ 3.14
- ○ int
- ● float
- ○ str

Soru 7:

`"a" + "b"` kodunun çıktısı aşağıdakilerden hangisidir?

- a + b
- ab
- 'ab'
- "a" + "b"

Soru 8:

"9" + "1" kodunun çıktısı aşağıdakilerden hangisidir?

- 10
- 9 + 1
- "9" + "1"
- '91'

Soru 9:

`"10" + 2` kodunun çıktısı aşağıdakilerden hangisidir?

- 12
- İşlem hata üretir
- 102
- "102"

Soru 10:

Verilen örnek kodun çıktısı aşağıdakilerden hangisidir?

```
a = 5
b = 10
c = a*b
c
```

- ○ 5
- ○ 10
- ○ 15
- ◉ 50

Python Programlama Alıştırmalar – 2

Soru 1:

Verilen örnek kodun çıktısı aşağıdakilerden hangisidir?

```
degisken = 4
print(degisken*degisken)
```

- ◉ 16
- ○ 4
- ○ degisken
- ○ İşlem hata üretir

Soru 2:

Verilen örnek kodun çıktısı aşağıdakilerden hangisidir?

```
sakla = 9
yeni_sakla = sakla*10
```

- ○ 90
- ○ 9
- ● kod çalışır çıktı üretmez
- ○ yeni_sakla

Soru 3:

Verilen örnek kodun çıktısı aşağıdakilerden hangisidir?

```
ifade = "selam"
type(ifade)
```

- ○ selam
- ○ int
- ○ ifade
- ● str

Soru 4:

Aşağıdakilerden hangisi bir sayı (float ya da integer) değildir?

- ● "3"
- ○ 98
- ○ 1/99
- ○ 2.2

Soru 5:

Verilen örnek kodun çıktısı aşağıdakilerden hangisidir?

```
ifade = "gelecegi yaziyoruz"
ifade[1]
```

- ○ 'gelecegi yaziyoruz'
- ○ 'g'
- ● 'e'
- ○ ifade

Soru 6:

Verilen örnek kodun çıktısı aşağıdakilerden hangisidir?

```
ifade = "gelecegi yaziyoruz"
ifade[0:2]
```

- ○ 'gelecegi yazıyoruz'
- ● 'ge'
- ○ 'gel'
- ○ g

Soru 7:

Verilen örnek kodun çıktısı aşağıdakilerden hangisidir?

```
a = "bu uzun bir metindir"
a[2:5]
```

- ○ 'u uzun'
- ○ 'uzun'
- ● ' uz'
- ○ 'zun'

Soru 8:

Verilen örnek kodun çıktısı aşağıdakilerden hangisidir?

```
a = "bu uzun bir metindir"
a[8]
```

- ○ 'm'
- ○ 'e'
- ● 'b'
- ○ ' '

Soru 9:

`"9" + 1` **kodunun çıktısı aşağıdaki hatalardan hangisini üretir?**

- ● TypeError
- ○ SyntaxError
- ○ Hata üretmez
- ○ 20

Soru 10:

Aşağıdakilerden hangisi bir karakter dizisinin eleman sayısını verir?

- ○ print()
- ○ lenght()
- ● len()
- ○ replace()

Python Programlama Alıştırmalar – 3

Soru 1:

Aşağıdakilerden hangisi bir karakter dizisinin tüm karakterlerini büyütmek için kullanılır?

- len()
- upper()
- lower()
- print()

Soru 2:

Bir karakter dizisi içerisinde yer alan karakterleri değiştirmek için aşağıdakilerden hangisi kullanılır?

- lower()
- upper()
- replace()
- len()

Soru 3:

Verilen örnek kod parçasının çıktısı aşağıdakilerden hangisidir?

```
ifade = "gelecek_geldi"
ifade.replace("i", "ı")
```

- (●) 'gelecek_geldı'
- ( ) Çıktı gelmez
- ( ) 'gelecek_geldi'
- ( ) "gelecek_geldi"

Soru 4:

Verilen örnek kod parçasının çıktısı aşağıdakilerden hangisidir?

```
ifade = "Merhaba!"
ifade = ifade.lower()
ifade = ifade.replace("!","")
ifade
```

- ( ) MERHABA!
- ( ) 'merhaba! '
- (●) 'merhaba'
- ( ) MERHABA

Soru 5:

Verilen örnek kod parçasının çıktısı aşağıdakilerden hangisidir?

```
"_Python_".strip("_")
```

- ○ Çalışmaz
- ● 'Python'
- ○ _Python_
- ○ "Python"

Soru 6:

Karakter dizilerinde sağ ve soldan "kırpma" işlemi yapmak için aşağıdakilerden hangisi kullanılır?

- ○ replace()
- ● strip()
- ○ len()
- ○ lower()

Soru 7:

Verilen örnek kod parçasının çıktısı aşağıdakilerden hangisidir?

```
ifade = "Merhaba! "
ifade.strip("")
```

- ○ Çalışmaz
- ○ Hata Üretir
- ○ Merhaba!
- ● 'Merhaba! '

Soru 8:

Veri yapılarına ilişkin metodlara erişmek için aşağıdakilerden hangisi kullanılır?

- len()
- dir()
- print()
- ?print

Soru 9:

Verilen örnek kod parçasının çıktısı aşağıdakilerden hangisidir?

```
ifade = "1012340"
ifade = ifade + "1"
ifade.strip("1")
```

- Hata üretir
- '1012341'
- ifade1
- '012340'

Soru 10:

Aşağıdakilerden hangisi kullanıcıdan bilgi almak için kullanılır?

- dir()
- replace()
- input()
- put()

Veri Yapıları (Data Types)

Listeler

1. Değiştirilebilir
2. Kapsayıcıdır (Farklı tipte verileri tutabilir.)
3. Sıralıdır

Köşeli parantez [ ] ya da list() fonksiyonu ile liste oluşturabiliriz.

Liste bir üst type'dır içersinde farklı type'da veriler barındırabilir.

```
notlar = [90,80,70,50] #liste olusturma
type(notlar) #--> list

liste=["a",19.5,3] #farkli tipleri barindiran liste

liste_genis=["a",19.5,3,notlar] #kapsayicidir. icersinde farkli veri tipleri hatta liste bile barindirabilir.
len(liste_genis) #boyutu 4 olur.
```

Liste Elemanlarına Ulaşma

```
#liste elamanlarina ulasma

liste_genis[0] #-->"a"
liste_genis[1] #-->19.5
liste_genis[2] #-->3
liste_genis[3] #-->[90,80,70,50]
```

```
liste_genis[0:2] #0'dan 2 indexli elemana kadar alir
liste_genis[:2] #0'dan 2 indexli elemana kadar alir
liste_genis[2:] # 2 indexli elemandan sona kadar alir

liste_genis

liste_genis[3][1] # liste_genis icersindeki notlar listesinin 1 indexli elemani
# --> 80

print(liste_genis[3][0]) #--> 90
```

Liste İçi Type Sorgulama

```
#liste ici type sorgulama

type(liste_genis[0])
type(liste_genis[1])
type(liste_genis[2])
type(liste_genis[3])

tum_liste=[liste,liste_genis]
```

del liste ✉ liste'yi siler

Liste elemanlarını değiştirme

```
# Liste elemanlarini degistirme

liste2=["ali","veli","berkcan","ayse"]
liste2

liste2[1]="velinin babasi" # 1 index'li elemani degistirdik

liste2

liste2[1]="veli"
liste2[:3]="alinin_babasi","velinin_babasi","berkcanin_babasi" #3 elemani degistirdik
liste2
```

Listeye eleman ekleme

```
#listeye eleman ekleme

liste2 + ["kemal"] # bu sekilde kaydetmez sadece görüntüler.

liste2 = liste2 + ["kemal"]
```

Listeden eleman silme

del liste2[5] ✉ 5 index'li elamanı siler.

append ve remove metodlari

liste2.append("berkcan") ✉sona ekleme yapar

liste2.remove("alinin_babasi") ✉silme yapar

liste2.remove("velinin_babasi")

insert metodu

index'e göre ekleme yapar.

```
#insert

liste2.insert(0,"ayca") #0 index'e ayca eklendi

liste2.insert(2,"recep") #2 index'e recep ekledi

liste2.insert(8,"asd") #fazla index girdik fakat sona ekledi

len(liste2)

liste2.insert(len(liste2),"son_eleman") #listenin sonuna ekledi
```

pop metodu

index'e göre silme yapar.

liste2.pop(0) #0 index degerli elemani siler

liste2.pop(1) #1 indexli elemani siler.

count metodu

```
#count

liste=["ali","veli","ayca","veli","ali","ali"]

liste.count("ali") #"ali" elemaninin listede kac kez yer aldigini gosterir.
```

✉ 3

copy metodu

liste_yedek=liste.copy() ✉ liste'yi liste_yedek'e kopyalar.

extend metodu

İki farklı listeyi birleştirir.

```
#extend

liste.extend(liste2) #liste ile liste2'yi birlestirir.
liste

liste2.extend(["a",10]) #liste ile metodun icine yazilan elemanlari birlestirir.

liste2
```

index metodu

```
#index

liste.index("ali") #yazdigimiz elemanin kacinci index oldugunu verir.
```

reverse metodu

liste = [1,2,3]

liste.reverse() ✉ liste elemanlarını ters sırayla kaydeder.

liste = [3,2,1]

sort metodu

Elemanları küçükten büyüğe sıralar.

```
#sort

liste3=[2,1,5,3,4]

liste3.sort() #liste3'ü kucukten buyuge siralayip kaydeder.
liste3
```

clear metodu

liste'nin içini boşaltır.

```
#clear

liste3.clear() #liste3'ün icini bosaltir

del(liste3) #liste3'ü tamamen siler.
```

Tuple (Demet)

1. Kapsayıcıdır
2. Sıralıdır
3. Değiştirilemez (Listeden farkı budur.)

Tuple Oluşturma

```
#Tuple Olusturma

t=(1,2,3,"eleman",[1,2,3,4])
```

NOT= Tek elemanlı tuple oluştururken sonuna virgül koymalıyız. Aksi takdirde tuple oluşturmak istediğimiz anlaşılamaz.

Örneğin; t = ("eleman",)

Eleman İşlemleri

Tuple'larda eleman işlemleri listeler ile birebir aynıdır. (index'e göre erişim vs.)

t=(1,2,3,4)

t[0] ✉ 1

t[-1] ✉4 (sondan birinci eleman demektir.)

Dictionary (Sözlük)

1. Kapsayıcıdır
2. Sırasızdır ✉ Listelerden farkı budur.
3. Değiştirilebilirdir.

Dictionary Nedir?

Key'ler ve bu key'lerin karşılıklarının bir arada tutulduğu veri yapısıdır.

Listelerde olduğu gibi index'leme yapılmaz.

Dictionary Oluşturma

```
# Sozluk Olusturma

sozluk={"REG" : "regresyon modeli",
        "LOJ" : "lojistik regresyon",
        "CART" : "Classification And Reg"}

sozluk
len(sozluk) # --> 3
```

{“key” : “key’in karşılığı”}

NOT= Sözlüklerde key’ler sadece sabit veri yapılarından oluşabilir. list gibi yapılardan olamaz. String ve sayılar sabit ver yapılarıdır.

Sabit veri yapısı değiştirilemez demektir. Tuple’da buna dahildir.

t = (“tuple”,) ✉ sozluk = { t : “tuple’dan key olur” }

Eleman Seçme İşlemleri

```
# Eleman secme islemleri

sozluk={"REG" : "regresyon modeli",
        "LOJ" : "lojistik regresyon",
        "CART" : "Classification And Reg"}

sozluk["REG"] #REG key'inin karsiligini bu sekilde getiririz.

sozluk={"REG" : {"ASD" : 10,
                 "XXX" : 20,
                 "ZZZ" : 30},

        "LOJ" : {"ASD" : 10,
                 "XXX" : 20,    # Sozluk icinde sozluk olusturduk. Ic ice yapi.
                 "ZZZ" : 30},

        "CART" : {"ASD" : 10,
                  "XXX" : 20,
                  "ZZZ" : 30}
        }

sozluk["REG"]["XXX"] #ic ice bir yapida elemana erisim.
```

```
In [6]: sozluk["REG"]["XXX"] #ic ice bir yapida elemana erisim.
Out[6]: 20
```

Eleman Ekleme & Değiştirme

```
In [14]: sozluk={"REG" : "regresyon modeli",
    ...:         "LOJ" : "lojistik regresyon",
    ...:         "CART" : "Classification And Reg"}

In [15]: sozluk["GBM"] = "Gradient Boosting Mac" #sozluk'e eleman ekleme.

In [16]: sozluk
Out[16]:
{'REG': 'regresyon modeli',
 'LOJ': 'lojistik regresyon',
 'CART': 'Classification And Reg',
 'GBM': 'Gradient Boosting Mac'}
```

```
In [17]: sozluk["REG"]= "REG'in yeni karsiligi" #REG Key'inin karsiligini degistirme.
    ...: sozluk
Out[17]:
{'REG': "REG'in yeni karsiligi",
 'LOJ': 'lojistik regresyon',
 'CART': 'Classification And Reg',
 'GBM': 'Gradient Boosting Mac'}
```

REG key'i olmasaydı yeni key oluşturulacaktı.

```
In [22]: t= ("tuple",) # t adinda tuple olusturduk.

In [23]: sozluk[t]="Tuple'dan key olusturuldu."
    ...: sozluk
Out[23]:
{'REG': "REG'in yeni karsiligi",
 'LOJ': 'lojistik regresyon',
 'CART': 'Classification And Reg',
 'GBM': 'Gradient Boosting Mac',
 ('tuple',): "Tuple'dan key olusturuldu."}
```

Sets (Kümeler)

1. Sırasızdır (Index değerleri yok.)
2. Değerleri eşsizdir. (Tekrar eden değeri olmaz.)
3. Değiştirilebilir.
4. Kapsayıcıdır. Farklı türden veri yapıları barındırabilir.

Set'ler performans odaklı veri tipleridir. Programlama anlamında biraz daha hız istediğimizde kullanılır. Matematiksel anlamda bu veri yapıları kümelere benzer.

Set Oluşturma

s = set() ✉ s isminde bir set oluşturuldu.

```
In [1]: l= ["ali","ata","bakma","ali","uzaya","git"]

In [2]: s=set(l) # l listesindeki elemanlari birer kez alir.

In [3]: s #set'in elemanlari essiz olacaginden her eleman bir kez alinir.
Out[3]: {'ali', 'ata', 'bakma', 'git', 'uzaya'}
```

```
In [4]: ali="ali_ata_bakma_uzaya_git_lutfen"

In [5]: s=set(ali) #ali cumlesindeki her bir karakteri bir kez alir.

In [6]: s
Out[6]: {'_', 'a', 'b', 'e', 'f', 'g', 'i', 'k', 'l', 'm', 'n', 't', 'u', 'y', 'z'}
```

Set’lere eleman ekleme ve çıkarma işlemleri

add() fonksiyonu ile ekleme yaparız.

```
In [8]: s.add("ile") #ile stringini set'e ekledi
   ...: s
Out[8]:
{'_',
 'a',
 'b',
 'e',
 'f',
 'g',
 'i',
 'ile',
 'k',
 'l',
 'm',
 'n',
 't',
 'u',
 'y',
 'z'}
```

```
In [9]: t=("ali","bakma")

In [10]: s.add(t) # t isimli tuple'i set'e ekledi
    ...: s
Out[10]:
{('ali', 'bakma'),
 '_',
 'a',
 'b',
 'e',
 'f',
 'g',
 'i',
 'ile',
 'k',
 'l',
 'm',
 'n',
 't',
 'u',
 'y',
 'z'}
```

```
In [12]: s.add(ali) # ali elemanini set'e ekledi.
    ...: s
Out[12]:
{('ali', 'bakma'),
 ' ',
 '_',
 'a',
 'ali_ata_bakma_uzaya_git_lutfen',
 'b',
 'e',
 'f',
 'g',
 'i',
 'ile',
 'k',
 'l',
 'm',
 'n',
 't',
 'u',
 'y',
 'z'}
```

remove() fonksiyonu ile set’lerden eleman silebiliriz.

```
In [13]: s.remove(ali) # ali elemanini sildi.
    ...: s
Out[13]:
{('ali', 'bakma'),
 ' ',
 '_',
 'a',
 'b',
 'e',
 'f',
 'g',
 'i',
 'ile',
 'k',
 'l',
 'm',
 'n',
 't',
 'u',
 'y',
 'z'}
```

```
s.remove(ali) # ali'yi tekrar silmek istedigimizde KeyError hatası verir.

s.discard(t) # discard ile de silme islemi gerceklestirebiliriz
s

s.discard(t) # tekrar silmek istedigimizde discard hata uretmez.
```

Set'lerde Fark İşlemleri

difference & symmetric_difference

difference = kümelerin farkını verir.

```
#difference ve symmetric_difference

set1= set([1,3,5])
set2= set([1,2,3])
```

```
In [2]: set1.difference(set2) #set1'in set2'den farki
Out[2]: {5}
```

```
In [3]: set2.difference(set1) #set2'in set1'den farki
Out[3]: {2}
```

symmetric_difference = ikisinde de ortak olmayan elemanları verir.

```
In [4]: set1.symmetric_difference(set2) #ikisinde de ortak olmayan elemanlari verir
Out[4]: {2, 5}
```

Set'lerde Kesişim ve Birleşim İşlemleri

intersection & union & intersection_update

intersection = kesişim

```
In [5]: set1.intersection(set2) # set1 ve set2'nin ortak elemanlari
Out[5]: {1, 3}

In [6]: set2.intersection(set1)
Out[6]: {1, 3}
```

union = birleşim

```
In [7]: set1.union(set2) # set1 ve set2'nin birlesimi
Out[7]: {1, 2, 3, 5}
```

intersection = set1'in değerini kesişim değerleri olarak değiştirir.

```
In [8]: set1.intersection_update(set2) #set1'in degerini kesisim degerleri olarak degistirir.

In [9]: set1
Out[9]: {1, 3}
```

Set'lerde Sorgu İşlemleri

isdisjoint & issubset & issuperset

isdisjoint = Ayrık küme mi?

İki kümenin kesişiminin boş olup olmadığını sorgular.

Boş ise True değil ise False döndürür.

```
In [10]: set1.isdisjoint(set2) #set1 ve set2'nin kesisimi bos mu? Ayrik kume mi?
Out[10]: False
```

issubset = subset'i mi? Alt kümesi mi? sorgusunu yapar.

```
In [11]: set1.issubset(set2) #set1 set2'nin subset'i mi?
Out[11]: True
```

issuperset = Kapsar mı?

```
In [13]: set2.issuperset(set1) #set2 set1'in superset'i mi? Kapsar mi?
Out[13]: True
```

Veri Yapıları Özet

Listeler	Tuple	Sözlük	Setler
Değiştirilebilir	Değiştirilemez	Değiştirilebilen	Değişebilir
Sıralı	Sıralı	Sırasız	Sırasız + Eşsizdir
Kapsayıcı	Kapsayıcı	Kapsayıcı	Kapsayıcıdır

Python Programlama Alıştırmalar – 4

Soru 1:

Aşağıdakilerden hangisi listelerin özelliklerinden değildir?

- Kapsayıcıdır
- Değiştirilemez
- Sıralıdır
- Index işlemleri yapılabilir

Soru 2:

Aşağıdakilerden hangisi tupleların özelliklerinden değildir?

- Değiştirilemezdir
- Değiştirilebilirdir
- Kapsayıcıdır
- Sıralıdır

Soru 3:

Aşağıdakilerden hangisi sözlük özelliklerinden değildir?

- Kapsayıcıdır
- Sıralıdır
- Sırasızdır
- Değiştirilebilirdir

Soru 4:

Aşağıdakilerden hangisi setlerin özelliklerinden değildir?

- Sırasızdır
- Değiştirilemezdir
- Değerleri eşsizdir
- Değiştirilebilirdir

Soru 5:

Bir liste tanımlanmak istendiğinde aşağıdakilerden hangisini kullanılır?

- " "
- ()
- {}
- []

Soru 6:

"()" ifadesi ile tanımlanan veri yapısı aşağıdakilerden hangisidir?

- liste
- tuple
- vektör
- sözlük

Soru 7:

"{}" ifadesi ile tanımlanan veri yapısı aşağıdakilerden hangisidir?

- (●) sözlük (dictionary)
- ( ) liste
- ( ) tuple
- ( ) vektör

Soru 8:

liste = ["A","B","C"]

Yukarıdaki listeye "D" ifadesini eklemek için aşağıdakilerden kodlardan hangisini yazmak gerekir?

- ( ) liste + "D"
- ( ) liste["D"]
- (●) liste.append("D")
- ( ) liste.insert("D")

Soru 9:

liste = ["A","B","C"]

Yukarıdaki listeye "D" ifadesini 0. indekse eklemek için aşağıdaki kodlardan hangisini yazmak gerekir?

- ( ) liste[0] = "D"
- ( ) liste.insert("D")
- ( ) liste.append(0, "D")
- (●) liste.insert(0, "D")

Soru 10:

Verilen "sozluk" ismindeki veri yapısının içerisine key ve value değerleri ile birlikte yeni bir eleman nasıl eklenir?

```
sozluk = {"reg" : "regresyon modeli",
"loj" : "lojistik regresyon",
"cart" : "classification and regression trees"}
```

- (●) sozluk["gbm"] = "gradient boosting machines"
- ( ) sozluk + "gbm"
- ( ) sozluk[0] + "gbm"
- ( ) sozluk[0] = "gradient boosting machines"

Python Programlama Alıştırmalar – 5

Soru 1:

Verilen "sozluk" ismindeki nesne içerisinden LOJ ifadesinin MSE değerine nasıl ulaşılır?

```
sozluk = {

"REG" : {"RMSE": 10,
"MSE": 11,
"SSE": 12},

"LOJ" : {"RMSE": 111,
"MSE": 2222,
"SSE": 333},

"CART" : {"RMSE": 99,
"MSE": 00,
"SSE": 66}}
```

- ( ) sozluk["LOJ" = "MSE"]
- (●) sozluk["LOJ"]["MSE"]
- ( ) sozluk["LOJ":"MSE"]
- ( ) sozluk["LOJ","MSE"]

Soru 2:

Verilen örnek kodun çıktısı nedir?

```
sozluk = {"REG" : {"RMSE": 10,
"MSE": 11,
"SSE": 12},

"LOJ" : {"RMSE": 111,
"MSE": 2222,
"SSE": 333},

"CART" : {"RMSE": 99,
"MSE": 00,
"SSE": 66}}


sozluk["CART"]["SSE"]
```

- ○ 11
- ○ 00
- ● 66
- ○ 111

Soru 3:

Verilen örnek kod ile yapılan işlem nedir?

`set([1,3,6,19])`

- ○ liste oluşturulmuştur
- ○ tuple oluşturulmuştur
- ● liste üzerinden set oluşturulmuştur
- ○ tuple üzerinden liste oluşturulmuştur

Soru 4:

Verilen kodun çıktısı nedir?

```
set1 = set([5,7,9])
set2 = set([5,6,7])
set2.difference(set1)
```

- ○ {6,9}
- ○ 6
- ○ 5
- ● {6}

Soru 5:

Verilen kodun çıktısı nedir?

```
set1 = set([5,7,9])
set2 = set([5,6,7])

set1.difference(set2)
```

- ● {9}
- ○ {6,9}
- ○ 9
- ○ 6

Soru 6:

Verilen örnek kodun çıktısı nedir?

```
set1 = set([5,7,9])
set2 = set([5,6,7])

set1.symmetric_difference(set2)
```

- ○ {5}
- ● {6,9} ya da {9,6}
- ○ 5,6
- ○ {5,6}

Soru 7:

Verilen örnek kodun çıktısı nedir?

```
set1 = set([5,7,9])
set2 = set([5,6,7])
set1.union(set2)
```

- ○ {5,6}
- ○ {5,6,9}
- ○ {5,7,9}
- ● {5,6,7,9}

Soru 8:

Verilen örnek kodun çıktısı nedir?

```
liste = [1,1,2,3,4,5,1,2,1]
liste.count(1)
```

- (●) 4
- ( ) '1,1,1,1'
- ( ) 1111
- ( ) Çalışmaz

Soru 9:

Bir listeye index sırasına göre eleman eklemek için hangi metod kullanılır.

- ( ) pop()
- ( ) reverse()
- (●) insert()
- ( ) extend()

Soru 10:

Verilen örnek kodun çıktısı nedir?

```
liste = [10,20,30,40]
liste.pop(1)
liste
```

- ○ 10
- ○ 20
- ◉ [10,30,40]
- ○ '10,20,30'

Python Programlama Alıştırmalar – 6

Soru 1:

Verilen örnek kodun çıktısı nedir?

```
liste = ["a","b","c"]
liste.extend(liste)
liste
```

- ○ Hata üretir
- ○ ['a', 'b', 'c']
- ◉ ['a', 'b', 'c', 'a', 'b', 'c']
- ○ ['c', 'b', 'a']

Soru 2:

Bir listeden index sırasına göre eleman silmek için hangi metod kullanılır.

- (●) pop()
- ( ) reverse()
- ( ) insert()
- ( ) append()

Soru 3:

Verilen örnek kodun çıktısı nedir?

```
liste = ["a","b","c"]
liste.reverse()
liste
```

- ( ) 'abc'
- ( ) ['a', 'b', 'c']
- ( ) ['a', 'b', 'c', 'a', 'b', 'c']
- (●) ['c', 'b', 'a']

Soru 4:

Verilen örnek kod parçasının çıktısı nedir?

```
t = ("a",10,"b")
t[0] = 1
```

- (●) Hata üretir
- ( ) ('a', 10, 'b')
- ( ) ('1', 10, 'b')
- ( ) ('a', 1, 'b')

Soru 5:

Verilen kod parçasının çıktısı nedir?

```
liste = ["a","b","c"]
liste.index("b")
```

- (●) 1
- ( ) ["a","b","c","b"]
- ( ) ["a" ,"c"]
- ( ) ["b"]

Soru 6:

Verilen kod parçasının çıktısı nedir?

```
liste = [50,10,30,40]
liste.sort()
liste
```

- ○ 50
- ◉ [10,30,40,50]
- ○ [50,10,30,40]
- ○ [50,40,30,10]

Soru 7:

Verilen kod parçasının çıktısı nedir?

```
liste = [10,10,20,40]
liste.clear()
liste
```

- [10,20,40]
- " "
- []
- ' '

Soru 8:

İki kümenin kesişiminin boş olup olmadığının sorgulanması için hangi metod kullanılır?

- dir()
- isdisjoint()
- issubset()
- isuperset()

Soru 9:

Bir kümenin tüm elemanlarının başka bir küme içerisinde yer alıp almadığı hangi metod ile kontrol edilir?

- dir()
- isdisjoint()
- issubset()
- isuperset()

Soru 10:

Bir kümenin bir diğer kümeyi tamamen kapsayıp kapsamadığını kontrol etmek için hangi metod kullanılır.

- (●) isuperset()
- ( ) dir()
- ( ) isdisjoint()
- ( ) direction()

Fonksiyonlar

Fonksiyon Nedir?

Belirli amaçları yerine getiren işleçlerdir.

Matematiksel İşlemler

```
In [14]: 4*4
Out[14]: 16

In [15]: 4/4
Out[15]: 1.0

In [16]: 4-2
Out[16]: 2

In [17]: 4+2 # bunlar klasik matematiksel operatorlerdir.
Out[17]: 6
```

Üs Alma

3**2 ✉ 3^2 anlamına gelir.

```
In [18]: 3**2 # 3'un 2'nci kuvveti
Out[18]: 9

In [19]: 3**3 # 3'un 3'ncu kuvveti
Out[19]: 27
```

Fonksiyon Nasıl Yazılır ?

def ile fonksiyon oluşturacağımızı belirtiriz.

```
# ==============================================================================
# #Fonksiyon Nasil Yazilir?

def kare_al(x):
    print(x**2) # def ile fonksiyon olusturacagimizi belirtiriz.

kare_al(5) #fonksiyonu bu sekilde calistiririz.

# ==============================================================================
```

```
In [21]: def kare_al(x):
    ...:     print(x**2) # def ile fonksiyon olusturacagimizi belirtiriz.
    ...:
    ...:

In [22]: kare_al(5) #fonksiyonu bu sekilde calistiririz.
25
```

Bilgi Notuyla Çıktı Üretmek

```
#Bilgi notuyla cikti uretme
def kare_al(x):
    print("Girilen sayinin karesi : " + x**2) #str + int

kare_al(3) #hata aldik cunku str ifadeler sadece str ifadeler ile birlestirilebilir.
```

Bu fonksiyonu çalıştırınca aldığımız hata :

```
In [17]: kare_al(3) #hata aldik cunku str ifadeler sadece str ifadeler ile birlestirilebilir.
Traceback (most recent call last):

  File "<ipython-input-17-31e075573f9a>", line 1, in <module>
    kare_al(3) #hata aldik cunku str ifadeler sadece str ifadeler ile birlestirilebilir.

  File "<ipython-input-16-4cc719a79d0b>", line 2, in kare_al
    print("Girilen sayinin karesi : " + x**2) #str + int

TypeError: can only concatenate str (not "int") to str
```

str ifadeler ile sadece str ifadeler birleştirilebilir!

type dönüşümü yapmalıyız.:

```
In [19]: def kare_al(x):
    ...:     print("Girilen sayinin karesi : " + str(x**2)) #str + str(type donusumu)
    ...:
    ...:

In [20]: kare_al(3) #bu kez hata almadan calisti.
Girilen sayinin karesi : 9
```

Başka bir örnek:

```
In [21]: def kare_al(x):
    ...:     print("Girilen sayi: " + str(x)
    ...:           +"\nKaresi: "+str(x**2)) #\n ile alt satira gectik.
    ...:
    ...:

In [22]: kare_al(4)
Girilen sayi: 4
Karesi: 16
```

İki Argümanlı Fonksiyon Tanımlamak

```
In [1]: def carpma_yap(x,y):
   ...:     print("Birinci sayi: "+ str(x)
   ...:             + "\nIkinci sayi: "+ str(y)
   ...:             + "\nCarpimlari: "+str(x*y))
   ...:
   ...:

In [2]: carpma_yap(3,4)
Birinci sayi: 3
Ikinci sayi: 4
Carpimlari: 12
```

Ön Tanımlı Argümanlar

Print() fonksiyonundan hatırlayacağımız gibi sep() ve end() gibi argümanlardır.

```
In [8]: def carpma_yap(x,y=1): # y=1 demeseydik iki degeri de girmek zorunda kalirdik.
   ...:     print(x*y)
   ...:
   ...:

In [9]: carpma_yap(3) #Hata vermeden calisacak.
3

In [10]: carpma_yap(3,5) #yeni bir deger girdigimizde eski degeri ezeriz.
15
```

y=1 yazarak ön tanımlı bir argüman oluşturmuş olduk.

Argümanların Sıralaması

Argümanların sırasını bilmediğimiz fakat isimlerini bildiğimiz zaman aşağıdaki şekilde çalıştırabiliriz.

```
# Argumanların Siralamasi

def carpma_yap(x,y):
    print(x*y)

carpma_yap(y=2, x=4) # Argumanlarin sirasini bilmiyorsam ama isimlerini biliyorsam
                     # Bu sekilde calistirabiliriz.
```

Ne Zaman Fonksiyon Yazılır?

Fonksiyonlar programlama dilleri içerisinde tekrar eden görevleri yerine getirmek ve var olan işleri daha programatik bir şekilde gerçekleştirmek için kullanılır.

Örneğin bir şehirde binlerce sokak lambası var ve bu sokak lambaları için ısı, nem, şarj değerlerini kullanarak bir hesaplama yapmamız gerekiyor. Her lamba için tek tek hesap mı yapacağız?

Hayır, fonksiyonu bir kez yazıp her lambada o fonksiyonu kullanacağız.

```
#Fonksiyonlar ne zaman yaizlir?
def direk_hesap(isi, nem, sarj):
    print((isi+nem)/sarj)

direk_hesap(25,40,70)
```

```
In [14]: direk_hesap(25,40,70)
0.9285714285714286
```

Fonksiyon Çıktılarını Girdi Olarak Kullanmak

Yazdığımız bir fonksiyonun çıktısını başka bir yerde girdi olarak kullanmak istiyorsak return ifadesini kullanmalıyız.

print() ekrana çıktı verir. Programlama anlamında kullanılabileceği anlamına gelmez.

Aşağıdaki örnekte görebiliriz.

```
#Fonksiyon Ciktilarini Girdi Olarak Kullanmak
#Fonksiyonun ciktisini baska bir yerde girdi olarak kullanmak icin
# return ifadesini kullanmaliyiz.

def direk_hesap(isi, nem, sarj):
    print((isi+nem)/sarj) #print ekrana cikti verir. Programlama anlaminda
                          #kullanilabilegi anlamina gelmez.

cikti = direk_hesap(25,40,70)
cikti #fonksiyonun sonucunu cikti'ya atayamadik
```

```
In [24]: def direk_hesap(isi, nem, sarj):
    ...:     return (isi+nem)/sarj #return ifadesini kullanirsak sonucu kullanabiliriz.
    ...:
    ...:

In [25]: cikti = direk_hesap(25,40,70)

In [26]: cikti
Out[26]: 0.9285714285714286
```

Fonksiyon return ifadesine gelince durur:

```
def direk_hesap(isi, nem, sarj):
    return
    (isi+nem)/sarj # bu sekilde calistirirsak fonksiyon islevini yapmaz.
                   # cunku fonksiyon return'un oldugu satira gelince durur.

direk_hesap(25,40,70)
```

Local ve Global Değişkenler

Ana çalışma alanımızdaki değişkenler Global değişkenlerdir.

Her hangi bir fonksiyonun ya da döngünün etkisindeki değişkenler ise Local değişkenlerdir.

```
#Local ve Global Degiskenler

x=10
y=10 #Ana calisma alanimizdaki degiskenler Global degiskenlerdir.

def carpma(x,y):
    return x*y #fonksiyon icersindeki degiskenler Local degiskendir.

carpma(2,3)
```

Local Etki Alanından Global Etki Alanını Değiştirme

Yazmış olduğumuz bir döngü içerisinde ya da tanımlamış olduğumuz bir fonksiyon içerisinden global değişkenlerin değerlerinde değişiklik yapmak istediğimiz zaman ne yapmamız gerekiyor ?

Python öncelikle local etki alanındaki değişkenleri tarar, arar ve bulmaya çalışır.

Örneğin bir fonksiyon yazdığımızda değişiklik yapmak istediğimiz değişkeni öncelikle kendi içersinde (local'de)arar, bulamazsa global alana çıkacak. Global alanda o değişkeni bulursa ona etki edecek (Orada da bulamazsa hata üretecek.). Aşağıdaki örnekte bu durumu gözlemleyebiliriz.

```
In [1]: x=[] #bos bir liste olusturuldu

In [2]: def eleman_ekle(y):
   ...:     x.append(y) #x'e y'yi ekle.
   ...:     print(str(y)+" ifadesi eklendi."
   ...:           +"\nListenin yeni hali: "+str(x))
   ...:
   ...:

In [3]: eleman_ekle(4)
4 ifadesi eklendi.
Listenin yeni hali: [4]

In [4]: eleman_ekle(3)
3 ifadesi eklendi.
Listenin yeni hali: [4, 3]
```

NOT=

Argüman sayısı bilinmiyorsa argüman isminden önce * ekleyin

```
def my_function(*kids): #Arguman sayisi bilinmiyorsa arguman isminden once * ekleyin.
  print("The youngest child is " + kids[-1])

my_function("Emil", "Tobias","Linus")
```

```
The youngest child is Linus
```

Karar-Kontrol Yapıları (Koşullar)

Koşul Nedir?

Örneğin günlük hayatta da kullandığımız gibi;

- Yağmur yağarsa şemsiye al
- Kar yağarsa zincir tak

gibi bazı olaylar gerçekleştiğinde bazı olayların gerçekleşmesi gerektiğini programlama diline ifade etmenin yollarıdır.

True – False Sorgulamaları (Boolean)

Doğru mu? sorusu sorar. == ile kullanırız.

```
In [3]: sinir = 5000 #sinir degiskenine deger verdik

In [4]: sinir == 4000 #sinir=4000'mu? sorusu sorar. False
Out[4]: False

In [5]: sinir == 5000
Out[5]: True
```

if – else – elif

if eğer anlamındaki koşuldur.

Eğer yazdığımız sorgu true ise alt satıra geçer ve çalışır.

```
sinir = 50000
gelir = 40000

gelir < sinir #True

if gelir < sinir: #sorgu true ise if alt satira gecer ve calisir.
    print("Gelir sinirdan kucuk.")
```

if = eğer true ise if çalışır.

else= değilse else çalışır.

```
In [14]: sinir = 50000
    ...: gelir = 40000

In [15]: if gelir > sinir: #sorgu true ise if'i calistirir.
    ...:     print("gelir sinirdan buyuk")
    ...: else: # sorgu false ise else'i calistirir.
    ...:     print("gelir sinirdan kucuk")
    ...:
    ...:
gelir sinirdan kucuk
```

```
if gelir==sinir:
    print("gelir sinira esittir.")
else:
    print("gelir sinira esit degildir.")
```

elif= if koşulu sağlanmazsa elif'e bakılır. elif koşulu da sağlanmazsa else çalışır.

Name	Type	Size	
gelir1	int	1	60000
gelir2	int	1	50000
gelir3	int	1	35000
sinir	int	1	50000

```
In [22]: if gelir1 < sinir:
    ...:     print("Geliriniz sinirdan kucuk!!")
    ...: elif gelir1 == sinir:
    ...:     print("Geliriniz sinirda.")
    ...: else:
    ...:     print("Tebrikler. Geliriniz sinirdan yukarida.")
    ...:
    ...:
Tebrikler. Geliriniz sinirdan yukarida.

In [23]: if gelir2 < sinir:
    ...:     print("Geliriniz sinirdan kucuk!!")
    ...: elif gelir2 == sinir: #if kosulu saglanmadiysa elif'e bakilir.
    ...:     print("Geliriniz sinirda.")
    ...: else: #hic bir kosul saglanmiyorsa else calisir.
    ...:     print("Tebrikler. Geliriniz sinirdan yukarida.")
    ...:
    ...:
Geliriniz sinirda.

In [24]: if gelir3 < sinir:
    ...:     print("Geliriniz sinirdan kucuk!!")
    ...: elif gelir3 == sinir: #if kosulu saglanmadiysa elif'e bakilir.
    ...:     print("Geliriniz sinirda.")
    ...: else: #hic bir kosul saglanmiyorsa else calisir.
    ...:     print("Tebrikler. Geliriniz sinirdan yukarida.")
    ...:
    ...:
Geliriniz sinirdan kucuk!!
```

Uygulama: if ve input ile kullanıcı etkileşimli program

Kullanıcıdan mağaza adı ve gelir bilgilerini alalım. Sınır değeri ile gelir değerini karşılaştıralım. Düşük, eşit, yüksek seviyelerine göre 3 farklı sonuç üretelim.

```
#Uygulama: if ve input ile kullanici etkilesimli program

sinir = 50000
magaza_adi=input("Magaza adi nedir?\n ") #kullanicidan magaza_adi aldik
gelir = int(input("Gelirinizi giriniz: ")) #kullanicidan aldigimiz geliri int'e cevirdik.

if gelir > sinir:
    print("Tebrikler "+magaza_adi+ " Geliriniz sinirdan yuksek :)")
elif gelir == sinir:
    print(magaza_adi+" Geliriniz sinirda.")
else:
    print("Uyari! "+magaza_adi +" Cok dusuk gelir: "+str(gelir))
```

Program çıktıları:

```
Magaza adi nedir?
 A Mağazası

Gelirinizi giriniz: 35000
Uyari! A Mağazası Cok dusuk gelir: 35000
```

```
Magaza adi nedir?
 B Mağazası

Gelirinizi giriniz: 50000
B Mağazası Geliriniz sinirda.
```

```
Magaza adi nedir?
 C Mağazası

Gelirinizi giriniz: 65000
Tebrikler C Mağazası Geliriniz sinirdan yuksek :)
```

```
[22]: while True:
    yil = int(input("Yıl giriniz:(Çıkış için 0 giriniz.) "))


    if yil==0:
        break
    elif 1200<=yil<1300:
        print("{} yılı 13. yüzyıldadır.".format(yil))
    elif 1300<=yil<1400:
        print ("{} yılı 14. yüzyıldadır.".format(yil))
    elif 1400<=yil<1500:
        print ("{} yılı 15. yüzyıldadır.".format(yil))
    elif 1500<=yil<1600:
        print ("{} yılı 16. yüzyıldadır.".format(yil))
    elif 1600<=yil<1700:
        print ("{} yılı 17. yüzyıldadır.".format(yil))
    elif 1700<=yil<1800:
        print ("{} yılı 18. yüzyıldadır.".format(yil))
    elif 1800<=yil<1900:
        print ("{} yılı 19. yüzyıldadır.".format(yil))
    elif 1900<=yil<2000:
        print ("{} yılı 20. yüzyıldadır.".format(yil))
    else:
        print("{} yılı 21. yüzyıldadır.".format(yil))
```

```
Yıl giriniz:(Çıkış için 0 giriniz.)  1999
1999 yılı 20. yüzyıldadır.
Yıl giriniz:(Çıkış için 0 giriniz.)  2000
2000 yılı 21. yüzyıldadır.
Yıl giriniz:(Çıkış için 0 giriniz.)  1453
1453 yılı 15. yüzyıldadır.
Yıl giriniz:(Çıkış için 0 giriniz.)  1571
1571 yılı 16. yüzyıldadır.
Yıl giriniz:(Çıkış için 0 giriniz.)  0
```

Döngüler

For Döngüsü

Örneğin bir liste içerisindeki elemanlara işlem yapmak istediğimizde o elemanlara tek tek gitme işlemini gerçekleştiren yapılara döngüler denir.

```
# For Dongusu

ogrenci = ["ali","veli","isik","berk"]

ogrenci[0]
ogrenci[1]

for i in ogrenci: #i gecici degiskendir.
    print(i)
```

à

```
ali
veli
isik
berk
```

Döngü ve Fonksiyonların Birlikte Kullanımı

```
maaslar=[1000,2000,3000,4000,5000]
```

Maaşlara %20 zam yapılacak. Gerekli kodlar nelerdir?

```
#maaslara %20 zam yapilacak gerekli kodlari yaziniz.

def yeni_maas(x):
    print(x*1.20)

yeni_maas(1000) #fonksiyonun calismasina ornek.

for i in maaslar:
    yeni_maas(i)
```

à

```
1200.0
2400.0
3600.0
4800.0
6000.0
```

Uygulama: if, for ve fonksiyonların birlikte kullanımı

Az önceki uygulamadaki maaş listesi kullanılarak; maaşı 3000 tl'den yüksek olanlara %10 zam, maaşı 3000 tl'den az olanlara ise %20 zam yapılacak.

```python
#if, for ve fonksiyonlarin bir arada kullanimi

maaslar=[1000,2000,3000,4000,5000]

def maas_ust(x):
    print(x*1.10) # %10 zam

def maas_alt(x):
    print(x*1.20) # %20 zam

for i in maaslar:
    if i>=3000: #maas 3000'den fazla veya esit ise
        maas_ust(i) # %10 zam uygulanacak
    else:          #değilse
        maas_alt(i) # %20 zam uygulanacak
```

à

```
1200.0
2400.0
3300.0
4400.0
5500.0
```

```python
#Listenin içindeki en küçük elemanı bulma

liste = [2,4,5,3,4,5,1,6,7,4,3,0,-500,456]

min = 100000000000
for each in liste:
    if (each<min):
        min=each
print(min)
```

```
-500
```

break & continue

Döngüler içersinde belirli bir şartı sağlayan ifadeler yakalandığında (if döngüsü ile yakalıyorduk.) döngü bitirilmek istenebilir. Ya da bu şartı sağlayan eleman görmezden gelinmek istenebilir.

Bu gibi durumlarda break ve continue ifadeleri kullanılır.

Örneğin; maaşı 3000 tl'ye kadar olanlarla ilgilendiğimizi düşünelim.

```python
#break & continue

maaslar=[8000,5000,2000,1000,3000,7000,1000]

maaslar.sort() #Karisik yazilmis listeyi kucukten buyuge siraladik.
maaslar
```

```
In [7]: for i in maaslar:
   ...:     if i ==3000: #1000,1000,2000 gecti 3000'e geldi if'e girdi. Durdu.
   ...:         print("kesildi")
   ...:         break
   ...:     print(i)
   ...:
   ...:
1000
1000
2000
kesildi
```

Örneğin; 3000’i atlayıp devam etsin.

```
In [8]: for i in maaslar:
   ...:     if i ==3000: #1000,1000,2000 gecti 3000'e geldi if'e girdi. Atladi.
   ...:         print("atlandi.")
   ...:         continue
   ...:     print(i)
   ...:
   ...:
1000
1000
2000
atlandi.
5000
7000
8000
```

while

Şart sağlandığı sürece devam eden bir döngüdür.

```
In [8]: sayi=1
   ...:
   ...: while sayi<10: #Sayi 10'a gelene kadar bu islemi devam ettir.
   ...:     sayi += 1 #sayiyi 1 arttir ve sayi degerine ata.
   ...:     print(sayi)
   ...:
   ...:
2
3
4
5
6
7
8
9
10
```

Python Programlama Alıştırmalar – 7

Soru 1:

round(3.14) , round(3.14,2), round(3.76), round(3.76,1)

Yukarıda bulunan round fonksiyonlarının çıktıları nelerdir?

- (4, 3.1, 4, 3.7)
- (3, 3.14, 4, 3.8)
- (3, 4, 4.1, 3)

Soru 2:

```
def x(y):
   y = y + [2]
   print(y)
c = [1,2,3]
x(c) = ?
print(len(c)) = ?
```

Yukarıda verilen kodların çıktısı nedir?

- [1, 2, 3, 2]
  3
- [1, 2, 3, 2]
  4
- [1, 2, 3]
  3

Soru 1:

Aşağıdakilerden hangisi fonksiyon tanımlamak için kullanılır?

- definition
- func
- def
- function

Soru 2:

Aşağıdaki verilen kod ne işe yarar?

`?print`

- print fonksiyonu çağırılır
- print fonksiyonu hakkında bilgi alma imkanı sağlar
- Böyle bir kod yoktur çalışmaz
- Boş bir çıktı verir

Soru 3:

Verilen kod parçasında bir fonksiyon tanımlanmıştır. Tanımlanan fonksiyon işlevini yerine getirmek adına nasıl kullanılır?

```
def kup_al(x):
    print(x**3)
```

- kup_al
- kup_al()
- print(kup_al())
- kup_al(2)

Soru 3:

```
def s(x, y = 2):
    c = 2
    for i in range(y):
        c = c + x
    return c
s(2) = ?
s(2,3) = ?
```

- 5 ve 6
- 6 ve 7
- 6 ve 8

Soru 4:

Verilen kodun çıktısı nedir?

```
def yazdir(metin):
    print(metin, "yazanlar")

yazdir("gelecegi")
```

- gelecegi yazanlar
- metin
- yazanlar
- gelecegi

Soru 5:

Verilen kodun çıktısı nedir?

```
def islem(x, y):
    print(x + y)

islem(1,9)
```

- ○ 1
- ● 10
- ○ 0
- ○ 9

Soru 6:

Verilen kodun çıktısı nedir?

```
def islem(x, y):
    print(x - y)

islem(3)
```

- ○ 3
- ○ 13
- ○ Kod çalışır ama çıktı üretmez
- ● İşlem hata üretir

Soru 7:

Verilen kod parçası çalıştırıldığında hata üretecektir. Bu hatanın önüne geçmek adına fonksiyon tanımlama esnasında ne yapmak gerekir.

```
def islem(x, y):
    print(x - y)


işlem(3)
```

- İki argüman değeri de girilmelidir
- y argümanına ön tanımlı değer verilmelidir
- return eklenmelidir
- print kaldırılmalıdır

Soru 8:

Verilen kodun çıktısı nedir?

```
def harf_say(x):
    len(x)

harf_say("Merhaba!")
```

- Kod çalışır ama çıktı üretmez
- Merhaba!
- 8
- 7

Soru 9:

Verilen kod parçası çalışacak fakat çıktı üretmeyecektir. Kodun kullanılabilir bir çıktı üretmesi için ne yapmak gerekir?

```
def harf_say(x):
    len(x)

harf_say("Merhaba!")
```

- Fonksiyon argümansız çalıştırılmalıdır
- Fonksiyon tanımlama bölümüne ek argüman eklenmelidir
- len yerine başka bir fonksiyon kullanılmalıdır
- return ifadesi kullanılmalıdır

Soru 10:

Verilen kodun çıktısı nedir?

```
def islem(x):
    if (x<0):
        return "NO"
    else:
        x*5

islem(2)
```

- Kod çalışır çıktı üretmez
- 10
- YES
- NO

Python Programlama Alıştırmalar - 8

Soru 1:

Verilen kodun çıktısı nedir?

```
def islem(x):
    if (x>10):
        return "YES"
    else:
        return x*5

islem(4)
```

- ○ Çalışmaz
- ○ NO
- ○ YES
- ● 20

Soru 2:

Verilen listenin her bir elemanını iteratif bir şekilde yakalayıp belirli bir işleme tabi tutmak için hangi yapı kullanılır?

- ○ Lambda yapısı
- ● for yapısı
- ○ if
- ○ Index işlemleri

Soru 3:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
a = [2,4,6,8]

for i in a:
    print(i**2)
```

- [2,4,6,8]
- [4,8,12,16]
- [4,16,36,64]
- 4
  16
  36
  64

Soru 4:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
sayilar = [10,20,30]

for i in sayilar:
    if i > 20:
        print(i/2)
```

- Çalışmaz
- 15.0
- 20
- 5

Soru 5:

Verilen kodun çıktısı nedir?

```
urun_fiyatlari = [19,29,39]

for i in urun_fiyatlari:
    if i >= 30:
        print(i/2)
    else:
        print(i*0)
```

- 19
  29
  39

- 9.5
  14.5
  0

- 0
  0
  19.5

- 9
  14
  19

Soru 6:

Verilen kod parçasının çıktısı ne olacaktır?

```
a = [1,2,3]
b = []
for i in a:
    b.append(i**2)

b
```

- [1, 4, 9]
- Çalışmaz
- [1,2,3]
- [2,4,6]

Soru 7:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
def mesaj():
    print("Merhaba!")

mesaj()
```

- ○ Hata üretir
- ○ Çalışır ama çıktı üretmez
- ○ Merhaba
- ◉ Merhaba!

Soru 8:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
for i in ["a",11]:
    print(i)
```

- ○ 11
- ○ a
- ○ Çalışmaz
- ◉ a
11

Soru 9:

Verilen kod parçasının çıktısı ne olacaktır?

```
def harf_say(x):
    return len(x)

harf_say("Merhaba!")
```

- ○ 7
- ● 8
- ○ Kod çalışmaz
- ○ Kod çalışır ama çıktı vermez

Soru 10:

break ifadesi ne için kullanılır?

- ● Kod akşını kesmek için (Örneğin bir şart yakalandığında çalışmayı durdur demek gibi)
- ○ Bir şart yakalandığında ekrana yazdırmak için
- ○ Bir şart yakalandığında ona bir işlem yapmak için
- ○ Yakalanan şartı atlayarak işleme devam etmek için

Python Programlama Alıştırmalar – 9

Soru 1:

continue ifadesi ne için kullanılır?

- ○ Bir şart yakalandığında ona bir işlem yapmak için
- ● Yakalanan şartı atlayarak işleme devam etmek için
- ○ Bir şart yakalandığında ekrana yazdırmak için
- ○ Yakalanan şarta gelindiğinde çalışmayı durdurmak için

Soru 1:

y = 3

z = lambda x:x*y

z(3) = ?

- ○ 7
- ○ 8
- ● 9

Soru 2:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
sayilar = [10,20,30,40]

for i in sayilar:
    if i == 30:
        break
    print(i)
```

- 10
- 10
  20
- 10
  20
  30
- 10
  20
  40

Soru 3:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
A = []

for i in [1,2,3,4]:
    A.append(i)


A[0]
```

- 1
- 3
- 4
- [1]

Soru 4:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
sayilar = [10,20,30,40]

for i in sayilar:
    if i == 30:
        continue
    print(i)
```

- (●) 10
  20
  40
- ( ) 10
- ( ) 10
  20
- ( ) 30
  40

Soru 5:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
if [1,2,3,4][2] == 2:
    print("YES")
else:
    print("NO")
```

- (●) NO
- ( ) YES
- ( ) 2
- ( ) 1

Soru 6:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
if [1,2,3,4][1] == 2:
    print("YES".lower())
else:
    print("NO")
```

- ○ no
- ● yes
- ○ YES
- ○ NO

Soru 7:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
A = "*A*"
if type(A) == str:
    A = A.strip("*")
    print(A)
```

- ● A
- ○ _A_
- ○ **A**
- ○ Hata üretir

Soru 8:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
A = 12

if type(A) == str:
    A = A.strip("*")
    print(A)
else:
    A = "*" + str(A) + "*"
    print(A.strip())
```

- Hata üretir
- A
- *A*
- *12* Çünkü strip() argümanı sadece str ifadelerde çalışır.

Soru 9:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
A = []
B = []


for i in [1,"a",12,"b"]:
    if type(i) == int:
        B.append(i)
    else:
        A.append(i)

A[1]
```

- 1
- 12
- 'a'
- 'b'

Soru 10:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
def islem(x,y):
    A = [x,y]
    return A[0] + A[1]

islem(1,3)
```

- 2
- 1
- 4
- Hata üretir

Generators

Python’u python yapan belli konular var. Generators ve Decorators gibi..

Generator’ler bize hem zaman hem de bellek tasarrufu sağlar.

Generators

```
[3]: #Ağır işlemci gücü kullanan bir fonksiyon yazdığımızı varsayalım.
     from time import sleep #sleep fonksiyonu için

     def compute():
         result = []

         for i in range(10):
             sleep(.5) #0.5 saniye bekleyecek, hesaplama süresini temsil etmek için.
             result.append(i**3)
         return result
     print(compute())
```

```
[0, 1, 8, 27, 64, 125, 216, 343, 512, 729]
```

Program her döngüye girdiğinde 0.5 saniye (10 kez döngüye gireceği için toplam 5 saniye) bekleyerek çalıştı.

İşlem süresi daha uzun sürecek işlemler yaptığımızda (5-10 dakika gibi) sonucu görmek için beklemek pek hoş olmaz.

Sonuçlar oluşmaya başladığı an çıktı olarak görmek istiyoruz.

```
[8]: #Amacımız döngünün her bir iterasyonunda ara değerleri döndürebilmek
     from time import sleep

     def compute2():
         for i in range(10):
             sleep(.5)
             yield i**3 #Generator

     for res in compute2(): #compute2'nin her bir değerini almak için for kullandık.
         print(res)

     #0.5 saniyede bir değer üretti ve 0.5 saniyede bir ekrana o an üretilen değer geldi.
```

```
0
1
8
27
64
125
216
343
512
729
```

Yukarıdaki programda ise bütün result'ı beklemeden ara değerler üzerinde işlem yapabiliriz.

Buradaki **yield,** generator’dür.

```
[12]: def count(n):
          for i in range(n):
              yield i

      res = count(3)
      type(res)
```

```
[12]: generator
```

```
[17]: import time
      def countdown(i):
          while i > 0:
              yield i
              i -= 1
      for i in countdown(9):

          print(list(countdown(i)))
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1]
[8, 7, 6, 5, 4, 3, 2, 1]
[7, 6, 5, 4, 3, 2, 1]
[6, 5, 4, 3, 2, 1]
[5, 4, 3, 2, 1]
[4, 3, 2, 1]
[3, 2, 1]
[2, 1]
[1]
```

Using **generators** results in improved performance, which is the result of the lazy (on demand) generation of values, which translates to lower memory usage. Furthermore, we do not need to wait until all the elements have been generated before we start to use them.

```
[19]: def make_word():
          word=""
          for char in "spam":
              word += char
              yield word
      print(list(make_word()))
```

```
['s', 'sp', 'spa', 'spam']
```

Object Oriented Programming

Class'lara Giriş ve Class Tanımlamak

Class Nedir?

Sınıflar; benzer özellikler, ortak amaçlar taşıyan, içerisinde metod ve değişkenler olan yapılardır.

```
#Siniflar
class VeriBilimci(): #Class tanimlama
    print("Bu bir class'dir.")
```

Class Özellikleri

```
#Siniflarin Ozellikleri
class VeriBilimci(): #Class tanimlama
    bolum = ''
    sql = 'Evet'
    Deneyim_Yili = 0
    bildigi_diller=[]
```

Class Özelliklerine Erişmek

```
#Siniflarin Ozelliklerine Erismek

VeriBilimci.sql
VeriBilimci.Deneyim_Yili
```

Class Özelliklerini Değiştirmek

```
#Siniflarin Ozelliklerini Degistirmek

VeriBilimci.sql = 'Hayir'
VeriBilimci.sql #Ozelligin degeri degisti.
```

Class Örneklendirmesi (instantiniation)

Sınıfın özelliklerini barındıran alt kümeler oluşturma işlemine sınıf örneklendirmesi denir.

```
#Sinif Orneklendirmesi (instantiation)

ali = VeriBilimci() #VeriBilimci sinifinin ozelliklerini tasiyan bir birim olustu.
                    #Yani ornekleme yapmis oldum.

ali.sql
ali.bildigi_diller.append("Python") #ali'nin bildigi_diller'e Python ekledik.
                                    #Ancak bu class'in hepsini etkiledi.
ali.bildigi_diller

veli = VeriBilimci()
veli.bildigi_diller #ali'nin bildigi dillere python eklemistik ancak veli'nin
                    #bildigi dillerde de python oldu.
```

Örnek Özellikleri

Şuan yapmış olduğumuz işlem her bir örneğin kendi içinde değişebilen özelliklerden oluşabildiği bilgisini vermek. Yani her bir ayrı örneklendirme için özellik tutma bilgisini sağlıyor.

Sınıflar için tanımlanan özellikler örnekler içn değişebilir bir formata getirilmedikçe bir örnekte yapılan değişiklik tüm örneklere etki ediyor.

def __init__(self):

self.bildigi_diller = ''

self.bolum = '' fonksiyonunu kullanacağız. Buradaki self temsilci anlamındadır. Her bir örneklemi temsil eder (ali, veli gibi).

Genelde sınıf özelliklerinin isimleri ve örnek niteliklerinin isimleri aynı olmamalıdır. Örneğimizde anlaşılır olması açısından aynı kullandık.

```python
#Ornek Ozellikleri

class VeriBilimci(): #yeni bir sinif tanimladik
    bildigi_diller = ["R","Python"] #Tum class icin ozellik atamasi.
    bolum = ''
    def __init__(self): #Orneklere ayri ayri ozellik atamasi yapmak icin.
        self.bildigi_diller = []
        self.bolum = ''

ali = VeriBilimci()
ali.bildigi_diller #bos

veli = VeriBilimci()
veli.bildigi_diller #bos

ali.bildigi_diller.append("Python") #ali'nin bildigi dillere ekleme yaptik.
ali.bildigi_diller #Bu kez python var.

veli.bildigi_diller.append("R") #veli'nin bildigi dillere ekleme yaptik.
veli.bildigi_diller #sadece veli'ye ekledigimiz R var.

VeriBilimci.bildigi_diller #Classin genelinde R ve Python var.

VeriBilimci.bolum # ''
ali.bolum = 'Istatistik'
veli.bolum = 'bil_sis_muh'
veli.bolum #bil_sis_muh
ali.bolum #istatistik
```

Örnek Metodları

Mesela her bir veri bilimci için bir yeni öğrenilen dili o veri bilimcinin bildiği dillere ekleme işlemi yapsın.

Örnekler üzerinde çalışan fonksiyonlar yazmak istiyoruz.

```
# Ornek Metodlari

class VeriBilimci(): #Bir class tanimladik.
    calisanlar = [] # calisanlar adinda bir nesne
    def __init__(self): #orneklerin ozellikleri
        self.bildigi_diller = [] #orneklerin ozellikleri
        self.bolum = '' #orneklerin ozellikleri
    def dil_ekle(self, yeni_dil): #orneklere etki edecek bir fonksiyon yazdik
        self.bildigi_diller.append(yeni_dil)

ali = VeriBilimci()
ali.bildigi_diller #suan bos
ali.bolum

veli = VeriBilimci()
veli.bildigi_diller
veli.bolum

ali.dil_ekle("R") #dil_ekle fonksiyonunu calistirdik.

VeriBilimci.dil_ekle(ali,"Python") #dil_ekle fonksiyonunu calistirdik.
                    #ali'nin bildigi dillere python eklendi.
                    #dil_ekle fonksiyonu iki sekilde de calistirilabilir.

ali.bildigi_diller #Python ve R var.
```

Miras Yapıları (inheritance)

Başka yerde başka bir class tanımlarken, tanımlayacak olduğumuz bu class daha önceden tanımlamış olduğumuz başka bir class'ın özelliklerini barındırıyorsa ve biz bunları kullanmak istiyorsak eski class'ın özelliklerini miras olarak kullanabiliyoruz.

```python
# Miras Yapilari (inheritance)

class Employees():
    def __init__(self,FirstName, LastName, Address):#Ozellikleri fonksiyonel. Sabit degil.
        self.FirstName = FirstName
        self.LastName = LastName
        self.Address = Address


class DataScience(Employees):  #Employees'den miras aliyor.
    def __init__(self,Programming):#Ozellikleri fonksiyonel. Sabit degil.
        self.Programming = Programming

class Marketing(Employees):   #Employees'den miras aliyor.
    def __init__(self,StoryTelling):#Ozellikleri fonksiyonel. Sabit degil.
        self.StoryTelling = StoryTelling

veribilimci1 = DataScience() #Parametreyi bos birakamayiz. Hata verir.
veribilimci1 = DataScience("Python")
veribilimci1.Programming #Python

pazarlamaci = Marketing("Yes")
pazarlamaci.StoryTelling #Yes
```

Functional Programming

Fonksiyonel Programlamaya Giriş

Python dili ile bir program yazmak istediğimizde bunu OOP(Nesneye Dayalı Programlama) özellikleri ile de yazabiliriz FP(Fonksiyonel Programlama) özellikleri ile de yazabiliriz.

Fonksiyonlar dilin baştacıdır. (Birinci sınıf nesnelerdir.)

Yan etkisiz fonksiyonlar. (stateless(durumsuz), girdi-çıktı ✉Ancak bir girdi verdiğimde çıktı üretir. Ve bu çıktı hep aynı olur. Dışarıdan etkilenemez.)

Yüksek seviye fonksiyonlar.

Yan Etkisiz Fonksiyonlar (Pure Functions)

Fonksiyonun bir şekilde dışarı bağımlı olduğu durumlara yan etkili yani impure(saf olmayan) fonksiyon denir.

Örnek-1: Bağımsızlık

```
In [1]: #Yan Etkisiz Fonksiyonlar (Pure Functions) Ornek-1

In [2]: A = 5

In [3]: def impure_sum(b): #saf olmayan fonksiyon. Sonucu A degiskenine bagimli.
   ...:     return b + A

In [4]: def pure_sum(a,b): #saf
   ...:     return a + b

In [5]: impure_sum(6) #A'yi degistirirsem sonucu degisir.
Out[5]: 11

In [6]: pure_sum(3,4) #Ne yaparsam yapayim sonucu girdilerden baska bir sey ile degismez.
Out[6]: 7

In [7]: A = 9 #eski deger 6 idi.

In [8]: impure_sum(6) #A'yi degistirirsem sonucu degisir. Girdi ayni, sonuc degisti.
Out[8]: 15
```

Örnek-2: Ölümcül Yan Etkiler

```
In [27]: #Ornek-2: Olumcul yan etkiler

In [28]: #OOP

In [29]: class LineCounter:
    ...:     def __init__(self, filename):
    ...:         self.file = open(filename , 'r')
    ...:         self.lines = []
    ...:
    ...:     def read(self):
    ...:         self.lines = [line for line in self.file]
    ...:
    ...:     def count(self):
    ...:         return len(self.lines)

In [30]: lc = LineCounter('deneme.txt')

In [31]: print(lc.lines)
[]

In [32]: print(lc.count())
0

In [33]: lc.read()

In [34]: print(lc.lines)
['Bu bir denemedir.\n', '\n', 'asdasd\n', '\n', 'asdfd\n', 'dhhjfhfg']

In [35]: print(lc.count())
6
```

```
In [36]: #FP

In [37]: def read(filename):
    ...:     with open(filename, 'r') as f:
    ...:         return [line for line in f]

In [38]: def count(lines):
    ...:     return len(lines)

In [39]: example_lines=read('deneme.txt')

In [40]: lines_count = count(example_lines)

In [41]: lines_count
Out[41]: 6

In [42]:
```

İsimsiz Fonksiyonlar (Lambda) (Anonymous Functions)

```
#İsimsiz Fonksiyonlar (Lambda) (Anonymous Functions)

def old_sum(a,b): #Eski tipte bir fonksiyon
    return a+b

new_sum = lambda a,b : a+b #Lambda ile fonksiyon- İsimsiz Fonksiyon
new_sum(4,5)

sirasiz_liste = [('b',3),('a',8),('d',12),('c',1)]
sirasiz_liste

sorted(sirasiz_liste, key=lambda x: x[1]) #Fonksiyon tanimladik.
#Out: [('c', 1), ('b', 3), ('a', 8), ('d', 12)]
```

Sorted bir fonksiyondu. Birinci argümanı bir nesneydi, listeydi. Elemanları da tuple idi. Bu listeye bir fonksiyon uygulamak istiyoruz. x'e bağlı bir fonksiyon, x olarak kendi içine girilen değerin 1. indexli elemanına ulaşsın.

Vektörel Operasyonlar (Vectorel Operations)

OOP ile iki listeyi çarpmak

```
In [13]: #Vektörel Operasyonlar (Vectorel Operations)

In [14]: a = [1,2,3,4] #amacimiz bu listeler icersindeki her bir elemani birbiriyle carpmak
    ...: b = [2,3,4,5] #yani 1*2,2*3,3*4,4*5
    ...:               #Listelerimiz tek boyutlu oldugu icin bunlara 'vektor' denir

In [15]: ab = [] #Carpma islemini saklamak icin global alanda bos liste olusturduk.

In [16]: for i in range(0, len(a)): #a'nin uzunlugu kadar i degeri uretecek(0,1,2,3)
    ...:     ab.append(a[i]*b[i]) #a'nin i'nci elemani ile b'nin i'nci elemanini carp. ab'ye ekle.

In [17]: ab
Out[17]: [2, 6, 12, 20]
```

Functionel Programming ile

Fakat söz konusu matematik, istatistik, veri bilimi, makine öğrenmesi gibi konular olduğunda asla bu tip döngülere vs. girmiyoruz. Vektörel operasyonlara giriyoruz.

```
In [1]: import numpy as np #numpy kutuphanesini calisma ortamima dahil ettim. np kisayolu atadim.

In [2]: a = np.array([1,2,3,4])
   ...: b = np.array([2,3,4,5])

In [3]: a*b
Out[3]: array([ 2,  6, 12, 20])
```

Fonksiyonel Programlama ile daha az çaba ile aynı sonuca ulaşmış olduk.

Map & Filter & Reduce

Fonksiyona argüman olarak fonksiyon yazmamıza izin veren fonksiyonlara First Class fonksiyon denir.

Map

Verilen bir nesne üzerinde tanımlanacak bir fonksiyonu çalıştırma imkanı verir.(lambda yani isimsiz fonksiyonu)

```
In [4]: liste = [1,2,3,4,5]

In [5]: for i in liste:
   ...:     print(i+10) #her elemana 10 ekleyip yazdir
11
12
13
14
15

In [6]: list(map(lambda x:x+10, liste)) #map fonk. ile her elemana 10 ekleyip liste yap.
Out[6]: [11, 12, 13, 14, 15]
```

Filter

filter fonksiyonu iteratif bir nesne alır bu nesne üzerinden başka bir iteratif nesne oluşturulur. Ve iteratif nesne içerisinde aradığı şartın sağlandığı tüm elemanlar listelenir.

Çift sayıları bulan fonksiyonu yazalım.

```
In [9]: liste = [1,2,3,4,5,6,7,8,9,10]

In [10]: list(filter(lambda x:x % 2 == 0, liste)) #2'ye bolumunden kalani 0'a esit olanlari listele.
Out[10]: [2, 4, 6, 8, 10]
```

Reduce

Az önceki filter fonksiyonu bize aradığımız değerleri bulup getirdi. Yani değerler ile ilgili bir işlem yapmadı. Reduce fonksiyonu yine map ve filter'a benzerdir fakat indirgeme işlemi yapar.

```
In [17]: #reduce

In [18]: from functools import reduce

In [19]: liste = [1,2,3,4,5,6,7,8,9,10]

In [20]: reduce(lambda a,b:a+b , liste) #liste elemanlarini toplar.
Out[20]: 55
```

Modül Oluşturma

Bazen modül, bazen kütüphane, bazen de paket dendiğini görebiliriz, bunların üçü de doğrudur. Modüller belirli amaçları yerine getirmek için bir arada bulunan fonksiyonlar topluluğudur.

Maaşlarla ilgili işlemler gerçekleştiren birkaç tane fonksiyonumuz olduğunu düşünelim ve bunu paketleyip bir modül haline getirelim.

Yeni bir .py dosyası açalım ve ismi HesapModulu.py olsun.

```
11- Functional Programming.py*   HesapModulu.py*

def yeni_maas(x):
    print(x*1.20)
```

Modülün içine fonksiyonu yazıp kaydettik. Modülümüz kullanıma hazır.

Başka bir .py dosyasından modüle erişmek:

```
#Modul Olusturma
#HesapModulu.py dosyasina fonksiyonu yazdik.

import HesapModulu #Modulu import ettik.

HesapModulu.yeni_maas(1000) #1200 olarak calisir.
```

Name	Type	Size	
HesapModulu	module	1	HesapModulu module

```
In [7]: import HesapModulu as hm #hm kisaltmasi ile modulu kullanmak icin.

In [8]: hm.yeni_maas(2000)
2400.0
```

Daha da kısa kullanımı için:

```
In [12]: from HesapModulu import yeni_maas # Farkli kullanim sekilleri.

In [13]: yeni_maas(4000)
4800.0
```

```
In [22]: import HesapModulu as hm

In [23]: hm.maaslar #Modulde olusturdugumuz liste tipindeki nesneyi aldik.
Out[23]: [1000, 2000, 3000, 4000, 5000]
```

Hatalar/İstisnalar (exception)

1-Programcı hataları: Bunlar basit hatalardır. Syntax hatası gibi.

2-Program hataları / bug: Bunlar kritik hatalardır çünkü program çalışmaya devam eder ancak çıktılar probremlidir. Çıktıların hatalı olmasının tespiti bile bazen zorlayıcı olabilir.

3-İstisnalar (exceptions): Programda bildiğimiz bazı hatalardır fakat bu hatalar gerçekleştiğinde programı durdurma, çalışmaya devam et demenin yoludur. Bunu try except yapısı ile sağlarız.

```
In [28]: a=10

In [29]: b=0

In [30]: a/b
Traceback (most recent call last):

  File "<ipython-input-30-aae42d317509>", line 1, in <module>
    a/b

ZeroDivisionError: division by zero
```

Gördüğümüz üzere ZeroDivisionError hatası ile karşılaştık. 0'a bölünemez.

```
In [28]: a=10

In [29]: b=0

In [30]: a/b
Traceback (most recent call last):

  File "<ipython-input-30-aae42d317509>", line 1, in <module>
    a/b

ZeroDivisionError: division by zero


In [31]: try: #kodu dene
    ...:     print(a/b)
    ...: except ZeroDivisionError: #calismazsa bu hata ile karsilastiginda ne olacak
    ...:     print("Payda sıfır olamaz.")
Payda sıfır olamaz.
```

```
[69]: try:
          liste = [1,2,3,4]
          print(liste[int(input("index : "))])

      except IndexError:
          print("Geçersiz index talebi...")

      index :  6
      Geçersiz index talebi...
```

Python Programlama Alıştırmalar – 10

Soru 1:

Bir sınıf tanımlamak aşağıdakilerden hangisi kullanılır?

- def
- class
- definition
- function

Soru 2:

Kod parçasında yer alan "fonksiyonlar" ve "OOP" tanımlamaları ne ifade etmektedir?

```
class BolumSorulari():
    fonksiyonlar = []
    OOP = []
```

- Örnek tanımlama
- Sınıf tanımlama
- Sınıf özellikleri tanımlama
- Kod çalışmaz

Soru 3:

Verilen kod parçacığında yapılan işlem ne anlama gelmektedir?

```
class BolumSorulari():
    fonksiyonlar = []
    OOP = []


BolumSorulari.OOP
```

- (●) Bir sınıf özelliğine erişilmiştir
- ( ) Sınıfa erişilmiştir
- ( ) Özelliklere erişilmiştir
- ( ) Fonksiyona erişilmiştir

Soru 4:

Verilen kod parçacığına göre aşağıdakilerden hangisi bir sınıf örneklendirmesidir?

```
class BolumSorulari():
    fonksiyonlar = []
    OOP = []
```

- BolumSorulari.OOP
- BolumSorulari.fonksiyonlar
- BolumSorulari["fonksiyonlar"]
- donguler = BolumSorulari()

Soru 5:

Aşağıdaki fonksiyonel programlama ile ilgili ifadelerden hangisi yanlıştır?

- Fonksiyonlar dilin baş tacıdır
- İsimsiz fonksiyonlar kullanılabilir
- Yan etkili fonksiyonlar vardır
- Vektörel işlemlere imkan sağlanır

Soru 6:

"Ancak bir girdi verildiğinde çıktı üreten fonksiyonlar" ifadesi aşağıdaki fonksiyonel programlama özelliklerinden hangisini işaret etmektedir.

- ○ Vektörel fonksiyonlar
- ○ Döngüsel fonksiyonlar
- ○ İç içe fonksiyonlar
- ◉ Yan etkisiz fonksiyonlar

Soru 7:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
fun = lambda x: x**2
fun(3)
```

- Hata üretir
- 6
- 3
- 9

Soru 8:

Verilen kod parçacığının çıktısı aşağıdakilerden hangisidir?

`list(map(lambda x: x*1, [2,7,4]))`

- 7
- [2, 7, 4]
- 2
- 4

Soru 9:

Verilen kod parçacığının çıktısı aşağıdakilerden hangisidir?

```
a = [1,2,3]
list(map(lambda x: x*2, a))
```

- [1,2,3]
- [1,4,9]
- [2,4,6]
- Çalışmaz

Soru 10:

Var olan sınıfların özelliklerini başka sınıflar için kullanmak için aşağıdakilerden hangisi kullanılır?

- Sınıf özellikleri
- Miras yapıları
- Örnek özellikleri
- Örnek metodları

Python Programlama Alıştırmalar – 11

Soru 1:

Aşağıdakilerden hangisi bir modül import etmek için kullanılamaz.

- (●) from import modul ismi
- ( ) import modul_ismi
- ( ) import modul_ismi as mi
- ( ) import modul_ismi as modül

Soru 2:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
list(map(lambda x: x.upper(), ["Ali","Veli","isik"]))
```

- ( ) [ali, veli, isik]
- ( ) ['ali', 'veli', 'isik']
- ( ) ['Ali', 'Veli', 'Isik']
- (●) ['ALI', 'VELI', 'ISIK']

Soru 3:

Verilen kod parçacığının çıktısı aşağıdakilerden hangisidir?

```
from functools import reduce
a = [1,2,3,4]
reduce(lambda a,b: a*b, a)
```

- (●) 24
- ( ) 10
- ( ) [1,2,3,4]
- ( ) [1,4,9,16]

Soru 4:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
A = [[1,2],[3,4],[5,6]]
list(map(lambda x: x[0]*3, A))
```

- ( ) [1, 3, 5]
- ( ) [2, 4, 6]
- (●) [3, 9, 15]
- ( ) [3, 7, 1]

Soru 5:

Aşağıda verilen for döngüsünde ele alınan matematiksel *işlem* map() fonksiyonu ile nasıl gerçekleştirilir?

```
liste = [1,2,3,4]
A = []

for i in liste:
    A.append(i**2)

print(A)
```

- lambda x: x*2
- list(map(lambda x: x**2))
- **list(map(lambda x: x**2, liste))**
- lambda x: x**2

Soru 6:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
A = [1,2,3,4,5]

if type(A) == ():
    print("islem gecersiz")
else:
    print(list(map(lambda x: x/1, A)))
```

- islem geçersiz
- Hata üretir
- **[1.0, 2.0, 3.0, 4.0, 5.0]**
- [1,1,1,1,1]

Soru 7:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
from functools import reduce
reduce(lambda a,b: a/b, [8,4,2])
```

- (●) 1.0
- ( ) [8,4,2]
- ( ) [2,4,8]
- ( ) 64.0

Soru 8:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
def yap(x,y,z):
    try:
        print(x/y*z)
    except ZeroDivisionError:
        print("gecersiz islem")

yap(1,2,0)
```

- ( ) 0.5
- ( ) 1.0
- ( ) 'gecersiz islem'
- (●) 0.0

Soru 9:

Verilen kod parçası ve çıktı için yazılması gereken kod aşağıdakilerden hangisidir?

```
def islem(x,y,z):
    if y == 0:
        print("hatali islem")
    else:
        return x/y*z
```

Çıktı:

`hatali islem`

- islem(1,2,3)
- **islem(1,0,2)**
- islem(1,2,0)
- islem(1,1,1)

Soru 10:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
import numpy as np
a = np.array([1,1,1])
b = np.array([2])

a+b
```

- [2]
- [2,2,2]
- [3,3,3]
- **array([3, 3, 3])**

Python Programlama Alıştırmalar – 12

Soru 1:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
A = []

for i in ["ali","veli","isik"]:
    A.append(i.replace("i","a"))

print(A)
```

- ○ Hata üretir
- ● ['ala', 'vela', 'asak']
- ○ ['aala', 'avela', 'aasak']
- ○ ['iala', 'ivela', 'iasak']

Soru 2:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

`list(filter(lambda x: x < 2, [1,2,3,4,5]))`

- ○ Çalışır ama çıktı üretmez
- ● [1]
- ○ [1,2,3]
- ○ []

Soru 3:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
liste = ["a",20,10,30,"b"]
list(filter(lambda x: type(x) == int, liste))
```

- [20, 10, 30]
- ["a","b"]
- ["a","20"]
- ["a",20,10,30,"b"]

Soru 4:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
list(filter(lambda x: len(x) > 8, ["pazartesi","sali","carsamba","persembe","cuma"]))
```

- ['pazartesi']
- ['sali']
- ['carsamba']
- ['persembe']

Soru 5:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
list(map(lambda x: x.capitalize(), ["abc","bcd","cde"]))
```

- ['bc', 'cd', 'de']
- ['Ab', 'Bc', 'Cd']
- ['Abc', 'Bcd', 'Cde']
- ['ABC', 'BCD', 'CDE']

Soru 6:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
from functools import reduce
reduce(lambda a,b: a+b, ["a","4","a"])
```

- 4
- 'a4a'
- 4a
- a4a

Soru 7:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
A = ["ali","veli","isik"]
B = [1,2,3]
AB = [A,B]


for i in AB:
    if type(i[0]) == int:
        print(list(map(lambda x: x-3, i)))
```

- [-2,-1,0]
- [1,2,3]
- ["ali","veli","isik"]
- Hata üretir

Soru 8:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
list(map(lambda x: x/10, filter(lambda x: x > 20, [10,20,30,40,50])))
```

- [10.0, 20.0, 30.0, 40.0, 50.0]
- [10.0, 20.0, 30.0, 40.0, 50.0]
- [1.0, 2.0, 3.0, 4.0, 5.0]
- [3.0, 4.0, 5.0]

Soru 9:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
A = ["ali","veli","isik"]
B = [1,2,3]
AB = [A, B]

for i in AB:
    if type(i[0]) == str:
        print(list(map(lambda x: x + " hi", i)))
```

- ['ali hi', 'veli hi', 'isik hi']
- ['ali', 'hi', 'veli', 'hi', 'isik', 'hi']
- ['ali', ' hi', 'veli', ' hi', 'isik', ' hi']
- Çalışır ama çıktı üretmez

Soru 10:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
from functools import reduce
A = ["Veri","Bilimi","Okulu"]
reduce(lambda a,b: a+b, list(map(lambda x: x[0], A)))
```

- (●) VBO
- ( ) VeriBilimiOkulu
- ( ) VeBiOk
- ( ) Veri Bilimi

--Python ile Veri Manipülasyonu: NumPy & Pandas--

NumPy (Numerical Python)

NumPy Giriş

NumPy Python'ın bazı numerik işlemlerde yetersiz kaldığı noktalarda ihtiyaçlarımızı gidermek için ortaya çıkmış bir kütüphanedir/modüldür.

Numpy Giriş

- Numerical Python
- Bilimsel hesaplamalar için kullanılır.
- Arrayler / çok boyutlu arrayler ve matrisler üzerinde yüksek performanslı çalışma imkanı sağlar.
- Temelleri 1995'te (matrix-sig, Guido Van Rossum) atılmış nihai olarak 2005 (Travis Oliphant) yılında hayata geçmiştir.
- Listelere benzerdir, farkı; verimli veri saklama ve vektörel operasyonlardır.

Neden NumPy?

Daha üst seviyeden, daha az çabayla daha büyük işler yapma olanağı sağladığından dolayı kullanıyor olacağız.

Neden NumPy? sorusunun ikinci ve önemli yanı yer tutma maliyetlerini numpy çok azaltmaktadır. Örneğin listede 4 elemanın her biri için type=int bilgisi 4 kez tutulur. Numpy array'inde ise sadece bir kez array'in kendisi için tutulur.

```
[1]: a = [1,2,3,4]
     b = [2,3,4,5]
     a
     b #sadece en sona ne yazdiysak onu yazdirir.

[1]: [2, 3, 4, 5]

[9]: import numpy as np

[11]: a = np.array([1,2,3,4])
      b = np.array([2,3,4,5]) #listeleri numpy'da array olarak tanimliyoruz.

[12]: a*b #uzun uzun islemlere gerek kalmadan listeleri birbiri ile carpar.

[12]: array([ 2,  6, 12, 20])
```

NumPy Array'i Oluşturmak

NumPy Array tıpkı sözlükler gibi listeler gibi bir veri tipidir.

NumPy Array'i Oluşturmak

[1]:
```
import numpy as np
```

[2]:
```
np.array([1,2,3,4,5]) #array olusturma
```

[2]:
```
array([1, 2, 3, 4, 5])
```

[4]:
```
a = np.array([1,2,3,4,5])
```

[5]:
```
type(a)
```

[5]:
```
numpy.ndarray
```

[6]:
```
np.array([3.14,4,2,1,13]) #float ve int karisik array
```

[6]:
```
array([ 3.14,  4.  ,  2.  ,  1.  , 13.  ])
```

Veri saklarken sadece bir veri tipi tutabilmek için bütün sayıları ondalıklı bir değere çevirdi.

[7]:
```
np.array([3.14,4,2,1,13], dtype="int") #Veri tipini kendimiz belirledik.
```

[7]:
```
array([ 3,  4,  2,  1, 13])
```

zeros, ones, full, random, arange, linspace, random.normal, random.randint

Sıfırdan Array Oluşturma

```
[1]: import numpy as np

[3]: np.zeros(10, dtype = int)

[3]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

[6]: np.ones((3,5) , dtype=int) #1'lerden olusan 3'e 5'lik 2 boyutlu array(matris)

[6]: array([[1, 1, 1, 1, 1],
            [1, 1, 1, 1, 1],
            [1, 1, 1, 1, 1]])

[7]: np.full((4,5) , 3) #3'lerden olusan 4x5'lik matris

[7]: array([[3, 3, 3, 3, 3],
            [3, 3, 3, 3, 3],
            [3, 3, 3, 3, 3],
            [3, 3, 3, 3, 3]])

[8]: np.arange(0,31,3) #0'dan 31'e kadar 3'er 3'er artan dogrusal dizi.

[8]: array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27, 30])

[9]: np.linspace(0,1,10) #0 ile 1 arasinda 10 tane sayi olustur.

[9]: array([0.        , 0.11111111, 0.22222222, 0.33333333, 0.44444444,
            0.55555556, 0.66666667, 0.77777778, 0.88888889, 1.        ])

[10]: np.random.normal(10, 4, (3,4)) #ortalamasi=10, standart sapmasi=4 olan 3x4'luk matris

[10]: array([[11.58826043, 11.68947875, 14.08713841, 10.49055712],
             [ 7.37205302, 11.91140801, 11.31641312, 12.05287956],
             [ 3.44476323,  9.28895348,  8.88684624,  6.07701004]])

[11]: np.random.randint(0, 10, (3,3)) #0 ile 10 araliginde rastgele int degerlerden 3x3'luk matris

[11]: array([[8, 4, 3],
             [0, 3, 7],
             [1, 2, 3]])
```

NumPy Array Özellikleri

NumPy Array Özellikleri

- **ndim**: boyut sayısı
- **shape**: boyut bilgisi
- **size**: toplam eleman sayısı
- **dtype**: array veri tipi

```
[1]: import numpy as np
```

```
[4]: np.random.randint(10, size = 10)
[4]: array([3, 3, 9, 6, 7, 8, 5, 1, 6, 1])
```

```
[5]: a = np.random.randint(10, size = 10)
```

```
[6]: a.ndim #boyut sayisi-- Tek boyutlu bir array oldugundan 1 gelecek.
[6]: 1
```

```
[7]: a.shape #boyut bilgisi-- Elimizdeki array tek boyutlu oldugundan sadece tek boyutunun bilgisini verecek.
[7]: (10,)
```

```
[9]: a.size #eleman sayisi
[9]: 10
```

```
[10]: a.dtype #array'in veri tipi
[10]: dtype('int32')
```

Matris Oluşturma

İki boyutlu array oluşturalım

```
[11]: b = np.random.randint(10, size = (3,5)) #3x5'lik 0 ile 10 arasindaki degerlerden olusan matris.
```

```
[12]: b
[12]: array([[4, 8, 4, 6, 0],
             [9, 6, 3, 4, 0],
             [8, 8, 4, 5, 7]])
```

```
[13]: b.ndim
[13]: 2
```

```
[14]: b.shape
[14]: (3, 5)
```

```
[15]: b.size
[15]: 15
```

```
[16]: b.dtype
[16]: dtype('int32')
```

Reshaping (Array'i Yeniden Şekillendirme)

Elimizde var olan bir array'i yeniden şekillendirme işlemi yapacağız. Örneğin elimizde bir array olsun ve bunu yeniden boyutlandıralım.

Fonskiyonlarımızın ürettiği çıktılar tek bir boyutta, tek bir array formunda gerçekleşebiliyor.

Bunları bazen tek boyuttan 2 boyuta ya da 2 boyuttan tek boyuta indirgeme işlemi gerekebiliyor.

Bu ihtiyaçlarla Reshape fonksiyonu ile başa çıkmış oluyoruz.

```
[2]: import numpy as np

[3]: np.arange(1, 10)

[3]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])

[4]: np.arange(1,10).reshape((3,3))

[4]: array([[1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]])
```

Not: Tek boyutlu array: Vektör, 2 boyutlu array: Matris.

```
[5]: a = np.arange(1,10)

[6]: a

[6]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Elimizdeki tek boyutlu array'i 2 boyutlu matris'e çevirmek istiyoruz ama tek boyuttaki bilgisi de olduğu şekilde kalsın.

```
[7]: a.ndim

[7]: 1

[9]: a.reshape((1,9)) #Artık bir matristir fakat tek boyutlu vektörün taşıdığı bilgiyi taşır.

[9]: array([[1, 2, 3, 4, 5, 6, 7, 8, 9]])

[11]: b = a.reshape((1,9)) #b artık matris.

[12]: b.ndim

[12]: 2
```

Ravel (Flatten)

Matris halindeki array'i düz bir vektör haline ravel ile getiririz.

```
[1]: import numpy as np

     array = np.array([[1,2,3],[4,5,6],[7,8,9]])
     array

[1]: array([[1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]])

[3]: a = array.ravel() #matris halindeki array'i düzlestirdik.
     a

[3]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
[5]: a=a.reshape(3,3) #tekrar 3x3 boyutunda matris haline getirdik.
     a

[5]: array([[1, 2, 3],
            [4, 5, 6],
            [7, 8, 9]])
```

Reshape ile Resize farkı nedir?

reshape() fonksiyonu array'e direkt etki etmez tanımlamak gerekir.

resize() fonksiyonu ise yeni bir tanımlama gerektirmez. Array'e kullanıldığı anda etki eder.

Concantenation (Array Birleştirme)

concatenate() fonksiyonu ile array'leri birleştirebiliriz.

```
[1]: import numpy as np

[2]: x = np.array([1,2,3])
     y = np.array([4,5,6])

[3]: np.concatenate([x, y]) #İki adet tek boyutlu array birlestirme

[3]: array([1, 2, 3, 4, 5, 6])

[4]: z = np.array([7,8,9])

[5]: np.concatenate([x,y,z]) #Üç adet tek boyutlu array birleştirme

[5]: array([1, 2, 3, 4, 5, 6, 7, 8, 9])
```

İki boyutlu matrislerde ise:

```
[6]: a = np.array([[1,2,3],
                   [4,5,6]]) #el ile 2 boyutlu matris olusturma

[7]: np.concatenate([a,a]) #standart olarak satir bazinda birlestirme yapar.

[7]: array([[1, 2, 3],
            [4, 5, 6],
            [1, 2, 3],
            [4, 5, 6]])

[8]: np.concatenate([a,a], axis=1) #axis=0 satir, axis=1 sutun bazinda birlestirir.

[8]: array([[1, 2, 3, 1, 2, 3],
            [4, 5, 6, 4, 5, 6]])
```

Stacking Array

2 array'i vertical olarak birleştirmek için vstack() kullanırız.

Horizontal olarak birleştirmek için hstack() kullanırız.

Stacking Array

```
[9]: array1 = np.array([[1,2],[3,4]])
     array1

[9]: array([[1, 2],
            [3, 4]])

[10]: array2 = np.array([[-1,-2],[-3,-4]])
      array2

[10]: array([[-1, -2],
             [-3, -4]])

[11]: #vertical stack

      array3 = np.vstack((array1, array2))
      array3

[11]: array([[ 1,  2],
             [ 3,  4],
             [-1, -2],
             [-3, -4]])

[12]: #horizontal stack

      array4 = np.hstack((array1, array2))
      array4

[12]: array([[ 1,  2, -1, -2],
             [ 3,  4, -3, -4]])
```

Convert and Copy

Convert and Copy

```
[13]: liste = [1,2,3,4] #list

      array = np.array(liste) #list'den array yaratma
      array

[13]: array([1, 2, 3, 4])

[14]: liste2 = list(array) #array'den list yaratma
      liste2

[14]: [1, 2, 3, 4]
```

Splitting (Array Ayırma)

split() fonksiyonu kullanılır.

```
[1]: import numpy as np

[2]: x = np.array([1,2,3,99,99,3,2,1])

[3]: np.split(x, [3,5]) #3. indise kadar ayir, sonra 5. indise kadar ayir, sonra sona kadar.

[3]: [array([1, 2, 3]), array([99, 99]), array([3, 2, 1])]
```

split fonksiyonuna girilen indis sayısı **n** ise çıktı array sayısı **n+1** olur

```
[4]: a,b,c = np.split(x, [3,5])

[5]: a

[5]: array([1, 2, 3])

[6]: b

[6]: array([99, 99])

[7]: c

[7]: array([3, 2, 1])
```

İki Boyutlu Array Ayırma

vsplit() : dikey olarak ayırmak için kullanılır.

hsplit() : yatay olarak ayırmak için kullanılır.

```
[8]: m = np.arange(16).reshape(4,4) #0-16 arasinda 4x4'luk matris.

[9]: m

[9]: array([[ 0,  1,  2,  3],
            [ 4,  5,  6,  7],
            [ 8,  9, 10, 11],
            [12, 13, 14, 15]])

[11]: np.vsplit(m, [2]) # yataydaki 2. indise kadar ve sonrasini ayir.

[11]: [array([[0, 1, 2, 3],
              [4, 5, 6, 7]]),
       array([[ 8,  9, 10, 11],
              [12, 13, 14, 15]])]

[13]: ust, alt = np.vsplit(m, [2])

[14]: ust

[14]: array([[0, 1, 2, 3],
             [4, 5, 6, 7]])

[15]: alt

[15]: array([[ 8,  9, 10, 11],
             [12, 13, 14, 15]])

[16]: np.hsplit(m,[2]) #dikeyde 2. indise kadar ve sonrasini ayir.

[16]: [array([[ 0,  1],
              [ 4,  5],
              [ 8,  9],
              [12, 13]]),
       array([[ 2,  3],
              [ 6,  7],
              [10, 11],
              [14, 15]])]
```

Sorting (Sıralama)

```
[17]: import numpy as np

[18]: v = np.array([2,1,4,3,5])

[19]: np.sort(v) #Kucukten buyuge siralar. Veri setinin orjinal yapisi bozulmadi.

[19]: array([1, 2, 3, 4, 5])

[20]: v.sort() #Veri setinin orjinal yapisini degistirdi.

[21]: v

[21]: array([1, 2, 3, 4, 5])
```

Matris sıralama

```
[23]: m = np.random.normal(20,5, (3,3))#ortalamasi 20, standart sapmasi 3 olan 3x3 matris.

[24]: m

[24]: array([[14.72354718, 25.72515484, 13.24908455],
             [16.62938435, 22.16685623, 22.44070384],
             [22.05424029, 13.64292261, 21.38588038]])

[25]: np.sort(m, axis=1) #Her bir satiri kendi icinde siralar.

[25]: array([[13.24908455, 14.72354718, 25.72515484],
             [16.62938435, 22.16685623, 22.44070384],
             [13.64292261, 21.38588038, 22.05424029]])

[27]: np.sort(m , axis=0) #Sutunlara gore siralama yapar.

[27]: array([[14.72354718, 13.64292261, 13.24908455],
             [16.62938435, 22.16685623, 21.38588038],
             [22.05424029, 25.72515484, 22.44070384]])
```

Index ile Elemana Erişmek

Tek boyutlu array'lerde eleman yakalama işlemleri listeler ile aynıdır.

```
[2]: import numpy as np
     a = np.array([1,2,3,4,5,6,7,8])
     a

[2]: array([1, 2, 3, 4, 5, 6, 7, 8])

[3]: a[0] #0 index'li eleman

[3]: 1

[5]: a[-1] #Sondan birinci eleman

[5]: 8

[6]: a[0] = 100 #eleman degerini degistirmek.

[7]: a

[7]: array([100,   2,   3,   4,   5,   6,   7,   8])
```

Matrislerde elemana erişme işlemleri

```
[14]: m = np.random.randint(10, size = (3,5))
      m
```

```
[14]: array([[3, 1, 4, 6, 3],
             [4, 9, 6, 7, 1],
             [9, 4, 1, 4, 7]])
```

```
[15]: m[0,0] #0'a 0 koordinatindaki eleman (index'e gore)
```

```
[15]: 3
```

```
[16]: m[1,1] #1'e 1 koordinatli eleman
```

```
[16]: 9
```

```
[18]: m[1,4]
```

```
[18]: 1
```

```
[19]: m[1,4] = 99
      m
```

```
[19]: array([[ 3,  1,  4,  6,  3],
             [ 4,  9,  6,  7, 99],
             [ 9,  4,  1,  4,  7]])
```

```
[20]: m[1,4] = 2.2 #float eklemek istiyoruz anca ondalik kismini keserek ekleyecek.
      m            #Daha onceden olusturulan bir array'in tipi sonradan ekleme ile degismez.
```

```
[20]: array([[3, 1, 4, 6, 3],
             [4, 9, 6, 7, 2],
             [9, 4, 1, 4, 7]])
```

Slicing (Array Alt Küme İşlemleri)

Tek boyutlu array'lerde slicing işlemleri

```
[1]: import numpy as np

[4]: a = np.arange(20,30)
     a

[4]: array([20, 21, 22, 23, 24, 25, 26, 27, 28, 29])

[5]: a[0:3]

[5]: array([20, 21, 22])

[6]: a[:3]

[6]: array([20, 21, 22])

[7]: a[3:]

[7]: array([23, 24, 25, 26, 27, 28, 29])

[8]: a[1::2] #1 index'den baslayarak 2'ser 2'ser artar.

[8]: array([21, 23, 25, 27, 29])

[11]: a[0::3] #0'dan baslar 3'er 3'er artar.

[11]: array([20, 23, 26, 29])
```

Matrislerde Slicing İşlemleri

Matrislerde Slicing İşlemleri

```
[12]: m = np.random.randint(10, size=(5,5))

[13]: m

[13]: array([[1, 9, 0, 0, 4],
             [9, 3, 3, 7, 3],
             [5, 2, 6, 8, 7],
             [3, 7, 2, 0, 9],
             [2, 1, 3, 4, 0]])

[15]: m[:,0] #Butun satirlar, 0. sutun

[15]: array([1, 9, 5, 3, 2])

[17]: m[:,1] #Butun satirlar, 1. sutun

[17]: array([9, 3, 2, 7, 1])

[19]: m[0,:] #0. satir, butun sutunlar

[19]: array([1, 9, 0, 0, 4])

[23]: m

[23]: array([[1, 9, 0, 0, 4],
             [9, 3, 3, 7, 3],
             [5, 2, 6, 8, 7],
             [3, 7, 2, 0, 9],
             [2, 1, 3, 4, 0]])

[24]: m[1:3,1:2] #1. ve 2. satirlar, 1. sutun

[24]: array([[3],
             [2]])

[29]: m[::,:2] #butun satirlar, ilk 2 sutun

[29]: array([[1, 9],
             [9, 3],
             [5, 2],
             [3, 7],
             [2, 1]])
```

Alt Küme Üzerinde İşlem Yapmak

Önceki bölümde array'lerin alt kümelerine eriştik fakat burada şöyle bir durum söz konusu;

Örneğin bir array'in alt kümesine eriştikten sonra bunu isimlendirip kaydettiğimizi düşünelim.

Bu kaydetmiş olduğumuz isimlendirme üzerinde bir değişiklik yaptığımızda array'in orjinali de değişiyordu.

Fakat bazen seçilen array'in alt kümesinde o alt kümeye özel işlemler yapılmak istenebilir.

İşte bu yüzden alt kümeleri bağımsızlaştırmak isimli bir işlem yapılması gerekiyor.

```
[30]: #Bir ornek ile yukaridaki durumu daha iyi anlayalim:
      import numpy as np
      a = np.random.randint(10, size=(5,5)) #5x5 matris olusturduk.
      a

[30]: array([[0, 0, 0, 1, 0],
             [8, 4, 5, 2, 9],
             [5, 2, 7, 4, 1],
             [7, 6, 2, 2, 6],
             [3, 6, 2, 1, 0]])

[34]: alt_a = a[0:3,0:2] #alt kume olusturduk
      alt_a

[34]: array([[999,   0],
             [  8, 888],
             [  5,   2]])

[32]: alt_a[0,0]=999 #alt kume elemanlarinda degisiklik yaptik.
      alt_a[1,1]=888
      alt_a

[32]: array([[999,   0],
             [  8, 888],
             [  5,   2]])

[33]: a #orjinal matrisimiz de etkilendi.

[33]: array([[999,   0,   0,   1,   0],
             [  8, 888,   5,   2,   9],
             [  5,   2,   7,   4,   1],
             [  7,   6,   2,   2,   6],
             [  3,   6,   2,   1,   0]])
```

Bu durum bazen çok iş görebilmekte.

Çok büyük boyutta array'ler elimizde olduğunda onların bazı parçalarını seçip spesifik olarak onların üzerinde çalışıp ana parçanın üzerinde değişiklik yapmak açısından çok işe yarar.

copy() metodunu kullanarak bu durumdan vazgeçebiliriz.

```
[38]: alt_b=m[0:3,0:2].copy() #bu islemden sonraki islemler ana array'den bagimsiz olacak.
```

```
[40]: alt_b[0,0]=9999
      alt_b #alt kume etkilendi
```

```
[40]: array([[9999,    9],
             [   9,    3],
             [   5,    2]])
```

```
[41]: m #orjinal array etkilenmedi.
```

```
[41]: array([[1, 9, 0, 0, 4],
             [9, 3, 3, 7, 3],
             [5, 2, 6, 8, 7],
             [3, 7, 2, 0, 9],
             [2, 1, 3, 4, 0]])
```

Fancy Index ile Elemanlara Erişmek

Fancy Index kavramı ilerleyen bölümlerde bizim için en önemli kavramlardan birisi olacak.

Bize hem Pandas data frame'lerinde hem de NumPy array'lerinde ileri düzey eleman seçme imkanları vermektedir.

```
[1]: import numpy as np
     v = np.arange(0,30,3)
     v
```

```
[1]: array([ 0,  3,  6,  9, 12, 15, 18, 21, 24, 27])
```

```
[2]: [v[1], v[2], v[3]] #eski yontemle elemanlara eristik.
```

```
[2]: [3, 6, 9]
```

```
[3]: #Ancak elimizde 100lerce elemanli bir array oldugunda bunu yapmak zor olacak.
```

```
[4]: al_getir = [1,3,5]
```

```
[6]: v[al_getir] #Iste buna Fancy Index denir.
```

```
[6]: array([ 3,  9, 15])
```

Matrislerde Fancy Index Kullanımı

Matrislerde Fancy Index Kullanımı

```
[8]: m = np.arange(9).reshape((3,3))
     m
```

```
[8]: array([[0, 1, 2],
            [3, 4, 5],
            [6, 7, 8]])
```

```
[9]: satir = np.array([0,1])
     sutun = np.array([1,2])
```

```
[10]: m[satir, sutun]
```

```
[10]: array([1, 5])
```

Basit Index ile Fancy kullanımı

```
[11]: #basit index ile fancy index
```

```
[12]: m
```

```
[12]: array([[0, 1, 2],
             [3, 4, 5],
             [6, 7, 8]])
```

```
[13]: m[0, [1,2]] #basit index ile fancy'i ayni anda kullandik.
```

```
[13]: array([1, 2])
```

Slice ile Fancy kullanımı

```
[14]: #slice ile fancy
```

```
[15]: m[0:, [1,2]] #basit index ile fancy'i ayni anda kullandik.
```

```
[15]: array([[1, 2],
             [4, 5],
             [7, 8]])
```

```
[ ]: #Buradaki islemlerin teknik olarak farklı oldugunu anlamamiz gerekir.
```

Koşullu Eleman İşlemleri

Koşullu Eleman İşlemleri

```
[2]: import numpy as np

[3]: v = np.array([1,2,3,4,5])

[4]: v > 5

[4]: array([False, False, False, False, False])

[5]: v < 3

[5]: array([ True,  True, False, False, False])

[6]: v[v < 3] #Fancy

[6]: array([1, 2])

[7]: v[v > 3] #Fancy

[7]: array([4, 5])
```

```
[8]: v[v >= 3] #Fancy
[8]: array([3, 4, 5])

[9]: v[v == 3] #Fancy
[9]: array([3])

[10]: v[v != 3] #Fancy
[10]: array([1, 2, 4, 5])

[11]: v
[11]: array([1, 2, 3, 4, 5])

[12]: v*2
[12]: array([ 2,  4,  6,  8, 10])

[13]: v/5
[13]: array([0.2, 0.4, 0.6, 0.8, 1. ])

[14]: v*5/10
[14]: array([0.5, 1. , 1.5, 2. , 2.5])

[15]: v**2
[15]: array([ 1,  4,  9, 16, 25], dtype=int32)
```

Matematiksel İşlemler

Matematiksel İşlemler

```
[1]: import numpy as np
     v = np.array([1,2,3,4,5])
     v
[1]: array([1, 2, 3, 4, 5])

[2]: v*5
[2]: array([ 5, 10, 15, 20, 25])
```

Biz çarpma işlemi yapsak da arka tarafta bu işlemler bir dönüştürmeye tabi tutulup NumPy içerisindeki spesifik fonksiyonlar çalıştırılıyor.

```
[3]: #bunlara ufunc denir.

[5]: np.subtract(v, 1) # v-1 isleminin arka planinda calisan fonksiyon

[5]: array([0, 1, 2, 3, 4])

[6]: np.add(v, 1) #v+1

[6]: array([2, 3, 4, 5, 6])

[7]: np.multiply(v, 4) #v*4

[7]: array([ 4,  8, 12, 16, 20])

[8]: np.divide(v, 3) #v/3

[8]: array([0.33333333, 0.66666667, 1.        , 1.33333333, 1.66666667])

[9]: np.power(v, 3) #v**3

[9]: array([  1,   8,  27,  64, 125], dtype=int32)

[10]: np.mod(v, 2) #v%2

[10]: array([1, 0, 1, 0, 1], dtype=int32)

[11]: np.absolute(np.array([-3])) #Mutlak deger

[11]: array([3])
```

Trigonometrik Fonksiyonlar

Trigonometrik Fonksiyonlar

```
[12]: np.sin(360)

[12]: 0.9589157234143065

[13]: np.cos(180)

[13]: -0.5984600690578581
```

Logaritmik İşlemler

Logaritmik İşlemler

```
[14]: v = np.array([1,2,3])

[15]: np.log(v)

[15]: array([0.        , 0.69314718, 1.09861229])

[16]: np.log2(v)

[16]: array([0.        , 1.        , 1.5849625])

[17]: np.log10(v)

[17]: array([0.        , 0.30103   , 0.47712125])
```

Numpy ile İki Bilinmeyenli Denklem Çözümü

Numpy ile İki Bilinmeyenli Denklem Çözümü

NumPy'i daha çok matematiğin alt dalı olan Lineer Cebir alanında düşünmeliyiz.

[3]:
```
import numpy as np
```

5 * x0 + x1 = 12

x0 + 3 * x1 = 10

Bu denklemdeki bilinmeyenlerin katsayılarını array'ler cinsinden ifade ederek numpy'in altında yer alan bir fonksiyon aracılığı ile bilinmeyen değerleri çözmüş olacağız.

Bu matematiksel problemi python'ın anlayacağı formata getirmemiz gerekiyor.
Bunun yolu da bilinmeyen ifadelerin katsayılarını bir vektöre koymak, **(a)**
bu denklemler sonucunda oluşan değerleri bir vektöre koymak, **(b)**
ve son olarak, **linalg** paketi içinde geliştirilmiş **solve** isimli fonksiyonu çalıştırmak. **(x)**

[5]:
```
a = np.array([[5,1], [1,3]])
b = np.array([12,10])
```

[6]:
```
a
```

[6]:
```
array([[5, 1],
       [1, 3]])
```

[7]:
```
b
```

[7]:
```
array([12, 10])
```

[8]:
```
x = np.linalg.solve(a,b)
x
```

[8]:
```
array([1.85714286, 2.71428571])
```

x0 ve **x1** değerlerlerini solve fonksiyonu ile bulduk.

NumPy Alıştırmalar- 1

Soru 1:

Aşağıdakilerden hangisi NumPy özelliklerinden değildir?

- ○ Bilimsel hesaplamalar için kullanılır
- ○ Array'ler üzerinde yüksek performanslı çalışma imkanı sağlar
- ○ Temelleri 1995'te atılmış ve nihai olarak 2005 yılında hayata geçmiştir
- ● Daha iyi döngüler yazmaya yardımcı olur

Soru 2:

Verilen kodun çıktısı aşağıdakilerden hangisidir?

```
import numpy as np
np.array([3.14, 4,6,1.2])
```

- ○ Çıktı yoktur
- ○ Kod çalışmaz
- ● array([3.14, 4., 6., 1.2])
- ○ array([3.14, 4, 6, 1.2])

Soru 3:

Aşağıda bir kod parçası ve çıktısı verilmiştir. Buna çıktının bu şekilde (kod bölümünde integer, çıktı bölümünde float tip gözlenmesi) olmasının sebebi nedir?

Kod:

```
import numpy as np
np.array([3.14, 4,6,1.2])
```

Çıktı:

`array([3.14, 4., 6., 1.2])`

- ○ Kütüphane yüklemesi ile ilgilidir
- ● Numpy array'lerinin **sabitlenmiş tip** özelliği ile ilgilidir
- ○ Çıktının bir özelliğidir
- ○ Numpy array'lerinin vektörel olmasındandır

Soru 4:

Aşağıdaki çıktıyı üretmek için hangi kod yazılmalıdır?

```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

○
```
import numpy as np
np.ones((3,2))
```

●
```
import numpy as np
np.ones((2,3))
```

○
```
import numpy as np
np.eye((2,3))
```

○
```
import numpy as np
np.eye((2,1))
```

Soru 5:

Bir NumPy array'i için satır ve sütun bilgisine nasıl erişilir?

- (●) ndim
- ( ) shape — Cevap shape olabilir.
- ( ) dtype
- ( ) dir

Soru 6:

Bir NumPy array'i için toplam eleman sayısı bilgisine nasıl erişilir?

- ( ) shape
- ( ) dtype
- (●) size
- ( ) dir

Soru 7:

Bir NumPy array'i için veri tipi bilgisine nasıl erişilir?

- (●) dtype
- ( ) ndim
- ( ) shape
- ( ) size

Soru 8:

Aşağıda verilen çıktının kodu hangisidir?

```
array([[[7, 9],
        [4, 0],
        [5, 9]],
       [[4, 8],
        [6, 4],
        [4, 5]],
       [[2, 2],
        [8, 2],
        [0, 2]]])
```

```
import numpy as np
np.random.randint(10, size = (1,3,2))
```

```
import numpy as np
np.random.randint(10, size = (2,3,2))
```

```
import numpy as np
np.random.randint(10, size = (3,2,2))
```

```
import numpy as np
np.random.randint(10, size = (3,3,2))
```

Soru 9:

Aşağıda verilen çıktının kodu hangisidir?

`array([1, 2, 3, 4, 5, 6, 7, 8, 9])`

```
import numpy as np
np.arange(0,10)
```

```
import numpy as np
np.arange(2,11)
```

```
import numpy as np
np.arange(1,10)
```

```
import numpy as np
np.arange(1,9)
```

Soru 10:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
import numpy as np
x = np.array([1, 2, 3])
y = np.array([4, 5, 6])
np.concatenate([x,y])
```

- ○
```
array([[1, 2, 3],
       [4, 5, 6]])
```

- ○
```
array([[1, 2, 3, 1, 2, 3],
       [4, 5, 6, 4, 5, 6]])
```

- ● `array([1, 2, 3, 4, 5, 6])`

- ○ array([14, 5, 6,1,2,3])

NumPy Alıştırmalar – 2

Soru 1:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

`5*np.array([1, 2, 3])`

- ● Çalışmaz çünkü gerekli import işlemi yapılmamıştır

- ○ array([11111, 22222, 33333])

- ○ array([5, 10, 15])

- ○ 5

Soru 2:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
import numpy as np
5*np.array([1,2,3])
```

- ○ Çalışmaz
- ● array([5, 10, 15])
- ○ array([11111,22222,33333])
- ○ 5

Soru 3:

Verilen kodun çıktısı aşağıdakilerden hangisidir?

```
import numpy as np
np.arange(0,10, 2)
```

- ○ array([0,10,0,10])
- ○ array([3,8])
- ● array([0,2,4,6,8])
- ○ array([0,2,4,6,8,10])

Soru 4:

Verilen kodun çıktısı aşağıdakilerden hangisidir?

```
import numpy as np
v = np.array([2, 1, 4, 3, 5])
np.sort(v)
```

- ○ array([5,3,4,1,2])
- ● array([1,2,3,4,5])
- ○ array([2,3,4,1,5])
- ○ array([3,2,1,5,4])

Soru 5:

Aşağıda verilen array'de yer alan 9 değerine erişmek için hangi kod yazılmalıdır?

`array([7, 3, 4, 7, 0, 9, 3, 2, 2])`

- ○ v[4]
- ● v[5]
- ○ v[9]
- ○ v[6]

Soru 6:

Verilen kod parçasının çıktısı aşağıdakilerden hangisidir?

```
import numpy as np
v = np.array([7, 3, 4, 7, 0, 9, 3, 2, 9, 2])
v[-2]
```

- 4
- 3
- 2
- 9

Soru 7:

Aşağıda "a" ismindeki bir array'in çıktısı verilmiştir. Buna göre yazılan kodun çıktısı hangisidir?

Array çıktısı:

```
array([[4, 7, 4, 5, 9],
[2, 5, 0, 7, 7],
[1, 9, 0, 8, 2]])
```

Kod:

`a[1,1]`

- 4
- 2
- 7
- 5

Soru 8:

Aşağıda "a" ismindeki bir array'in çıktısı verilmiştir. Buna göre yazılan kodun çıktısı hangisidir?

Çıktı:

```
array([[4, 0, 3, 0, 1],
       [9, 6, 1, 5, 9],
       [1, 9, 0, 8, 2]])
```

Kod:

`a[0:1]`

- array([[4, 0, 3, 0, 1],
  [9, 6, 1, 5, 9]])
- array([4, 0, 3, 0, 1])
- array([[4, 0, 3, 0, 1]])
- array([[1, 9, 0, 8, 2]])

Soru 9:

Aşağıda "a" ismindeki bir array'in çıktısı verilmiştir. Buna göre yazılan kodun çıktısı hangisidir?

Çıktı:

```
array([[4, 7, 4, 5, 9],
[2, 5, 0, 7, 7],
[1, 9, 0, 8, 2]])
```

Kod:

`a[2,3]`

- 5
- 8
- 7
- 2

Soru 10:

Aşağıda "a" ismindeki bir array'in çıktısı verilmiştir. Buna göre yazılan kodun çıktısı hangisidir?

Çıktı:

```
array([[4, 7, 4, 5, 9],
[2, 5, 0, 7, 7],
[1, 9, 0, 8, 2]])
```

Kod:

`a[3,2]`

- 0
- 8
- 7
- Çalışmaz çünkü index karşılığı yok

NumPy Alıştırmalar – 3

Soru 1:

Bir NumPy array'i için boyut sayısı bilgisine nasıl erişilir?

- ndim
- shape
- dtype
- dir

Soru 2:

Aşağıda "a" ismindeki bir array'in çıktısı verilmiştir. Buna göre yazılan kodun çıktısı nedir?

Çıktı:

```
array([[4, 7, 4, 5, 9],
[2, 5, 0, 7, 7],
[1, 9, 0, 8, 2]])
```

Kod:

`a[:,2]`

- (●) array([4, 0, 0])
- ( ) array([[2, 5, 0, 7, 7],
  [1, 9, 0, 8, 2]])
- ( ) array([7, 5, 9])
- ( ) array([[4, 7, 4, 5, 9]
  [2, 5, 0, 7, 7]])

Soru 3:

Aşağıda "a" ismindeki bir array'in çıktısı verilmiştir. Buna göre yazılan kodun çıktısı nedir?

Çıktı:

```
array([[4, 7, 4, 5, 9],
[2, 5, 0, 7, 7],
[1, 9, 0, 8, 2]])
```

Kod:

`a[:2, :3]`

- ( ) array([1, 9, 0, 8, 2])
- (●) array([[4,7,4],
  [2,5,0]])
- ( ) array([5,7,8])
- ( ) array([4, 7, 4, 5, 9])

Soru 4:

Aşağıda "v" ismindeki bir array'in çıktısı verilmiştir. Buna göre yazılan kodun çıktısı nedir?

Çıktı:
`array([ 0, 3, 6, 9, 12, 15, 18, 21, 24, 27])`

Kod:

`[v[1], v[3]]`

- ○ Çalışmaz
- ○ 3,9
- ● [3,9]
- ○ [0,6]

Soru 5:

Aşağıda "v" ismindeki bir array'in çıktısı verilmiştir. Buna göre yazılan kodun çıktısı nedir?

Çıktı:
`array([ 0, 3, 6, 9, 12, 15, 18, 21, 24, 27])`

Kod:

`[v[9], v[0]]`

- ○ Çalışmaz
- ○ 0,9
- ○ [9,0]
- ● [27,0]

Soru 6:

Aşağıdaki seçim işleminin teknik ismi nedir?

```
import numpy as np
v = np.array([ 0, 3, 6, 9, 12, 15, 18, 21, 24, 27])

v[[1,2,3]]
```

- ○ Index seçimi
- ● Fancy index seçimi
- ○ Slice index seçimi
- ○ Vektör seçimi

Soru 7:

Aşağıda "m" ismindeki bir array'in çıktısı verilmiştir. Buna göre yazılan kodun çıktısı nedir?

Çıktı:

```
array([[0, 1, 2],
[3, 4, 5],
[6, 7, 8]])
```

Kod:

`m[0, [1,2]]`

- ○ array([0, 1])
- ● array([1, 2])
- ○ Çalışmaz
- ○ array([0,3])

Soru 8:

Bir numpy array'nin alt kümesi üzerinde işlem yaparken alt küme üzerinde yapılan değişikliklerin array'in ilk halinden bağımsız olması için hangi fonksiyon kullanılır?

- multiply()
- divide()
- copy()
- dir

Soru 9:

Aşağıda verilen fonksiyon ile aynı işlevi gören kod aşağıdakilerden hangisidir?

```
import numpy as np
np.power(v, 3)
```

- 3*v
- v*3
- v***3
- v**3

Soru 10:

Aşağıda verilen fonksiyon ile aynı işlevi gören kod aşağıdakilerden hangisidir?

```
import numpy as np
np.subtract(v, 2)
```

- v/2
- v**2
- v + 2
- v-2

Pandas

Pandas Giriş

- Panel Data
- Veri manipülasyonu ve veri analizi için yazılmış açık kaynak kodlu bir Python kütüphanesidir.
- Ekonometrik ve finansal çalışmalar için doğmuştur.
- Temeli 2008 yılında atılmıştır.
- R DataFrame yapısını Python dünyasına taşımış ve DataFrame'ler üzerinde hızlı ve etkili çalışabilme imkanı sağlamıştır.
- Bir çok farklı veri tipini okuma ve yazma imkanı sağlar.

Pandas Serisi Oluşturmak

Pandas Serisi Oluşturmak

Pandas içerisinde yer alan veri tipleri değerleri, indeksleri ile beraber tutar.

[2]:
```
import pandas as pd
```

[3]:
```
pd.Series([10,88,3,4,5]) #pandas serisi olusturmak.
```

[3]:
```
0    10
1    88
2     3
3     4
4     5
dtype: int64
```

[4]:
```
 seri = pd.Series([10,88,3,4,5])
```

[5]:
```
type(seri)
```

[5]:
```
pandas.core.series.Series
```

[6]:
```
seri.axes #Serinin index bilgisine ulasiriz.
```

[6]:
```
[RangeIndex(start=0, stop=5, step=1)]
```

[8]:
```
seri.dtype
```

[8]:
```
dtype('int64')
```

[9]:
```
seri.size #eleman sayisi
```

[9]:
```
5
```

[10]:
```
seri.ndim #boyutu
```

[10]:
```
1
```

[11]:
```
seri.values #vektor formunda sadece degerlere ulasiriz.
```

[11]:
```
array([10, 88,  3,  4,  5], dtype=int64)
```

[7]:
```
seri.head() #ilk 5 eleman
```

[7]:
```
0    10
1    88
2     3
3     4
4     5
dtype: int64
```

```
seri.head(3) #ilk 3 eleman
```

```
0    10
1    88
2     3
dtype: int64
```

```
seri.tail(3) #son 3 eleman
```

```
2    3
3    4
4    5
dtype: int64
```

Index İsimlendirmesi

Index İsimlendirmesi

```
[10]: pd.Series([23,24,25,26,27], index = [2,4,6,8,10])

[10]: 2     23
      4     24
      6     25
      8     26
      10    27
      dtype: int64

[11]: seri = pd.Series([23,24,25,26,27], index = ["a","b","c","d","e"])

[13]: seri

[13]: a    23
      b    24
      c    25
      d    26
      e    27
      dtype: int64

[14]: seri["a"] #elemana erisme

[14]: 23

[15]: seri["a":"c"] #serilerde slice islemi

[15]: a    23
      b    24
      c    25
      dtype: int64
```

Sözlük Üzerinden Seri Oluşturmak

Sözlük üzerinden seri oluşturmak

```
[16]: sozluk={"reg":10, "log":11, "cart":12}

[18]: seri = pd.Series(sozluk)

[19]: seri

[19]: reg     10
      log     11
      cart    12
      dtype: int64
```

İki Seriyi Birleştirerek Seri Oluşturma

İki Seriyi Birleştirerek Seri Oluşturma

[20]:
```
pd.concat([seri, seri])
```

[20]:
```
reg     10
log     11
cart    12
reg     10
log     11
cart    12
dtype: int64
```

Eleman İşlemleri

Eleman İşlemleri

[23]:
```
import numpy as np
a = np.array([15,233,34,52,64])
seri = pd.Series(a) #NumPy Array'i uzerinden seri olusturalim
seri
```

[23]:
```
0     15
1    233
2     34
3     52
4     64
dtype: int32
```

[24]:
```
seri[0] #0 indexli eleman
```

[24]:
```
15
```

[25]:
```
seri[0:3] #3'e kadar olan elemanlar
```

[25]:
```
0     15
1    233
2     34
dtype: int32
```

[27]:
```
seri = pd.Series([133,244,355,467,234], index = ["reg","log","cart","pcv","rf"])
seri
```

[27]:
```
reg     133
log     244
cart    355
pcv     467
rf      234
dtype: int64
```

[29]:
```
seri.index #sadece indexler
```

[29]:
```
Index(['reg', 'log', 'cart', 'pcv', 'rf'], dtype='object')
```

```
[30]: seri.keys #seri'nin key'lerini gosterir
```

```
[30]: <bound method Series.keys of reg     133
      log     244
      cart    355
      pcv     467
      rf      234
      dtype: int64>
```

```
[31]: list(seri.items()) #key degerine karsilik gelen value'lari bir araya getirerek list olusturur.
```

```
[31]: [('reg', 133), ('log', 244), ('cart', 355), ('pcv', 467), ('rf', 234)]
```

```
[32]: seri.values #seri'nin sadece degerlerini gosterir
```

```
[32]: array([133, 244, 355, 467, 234], dtype=int64)
```

Eleman Sorgulama

Eleman Sorgulama

```
[33]: "reg" in seri
```

```
[33]: True
```

```
[34]: "a" in seri
```

```
[34]: False
```

```
[35]: seri["reg"]
```

```
[35]: 133
```

Fancy Eleman

Fancy Eleman

```
[37]: seri[["rf","reg"]] #fancy ile eleman secme
```

```
[37]: rf     234
      reg    133
      dtype: int64
```

Eleman Değiştirme

Eleman Değiştirme

```
[39]: seri["reg"] = 111
      seri #atama yontemi ile tekrardan eleman atayabiliriz.
```

```
[39]: reg     111
      log     244
      cart    355
      pcv     467
      rf      234
      dtype: int64
```

Pandas DataFrame Oluşturma

Pandas DataFrame Oluşturma

Pandas DataFrame yapısal bir veri tipidir.

[2]:
```
import pandas as pd
l = [5,12,37,62,14] #list olusturduk
l
```

[2]:
```
[5, 12, 37, 62, 14]
```

[3]:
```
pd.DataFrame(l, columns = ["degisken_ismi"]) #DataFrame olusturma
```

[3]:

	degisken_ismi
0	5
1	12
2	37
3	62
4	14

[5]:
```
import numpy as np
m = np.arange(1,10).reshape(3,3)
m #3x3'luk bir matris
```

[5]:
```
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])
```

[6]:
```
pd.DataFrame(m, columns=["var1","var2","var3"]) #2 boyutlu DataFrame
```

[6]:

	var1	var2	var3
0	1	2	3
1	4	5	6
2	7	8	9

Yapay zeka ve Veri Biliminde en çok kullanacağımız veri tipi DataFrame'dir.

DataFrame İsimlendirme

DataFrame İsimlendirme

[7]:
```
df = pd.DataFrame(m, columns=["var1","var2","var3"])
df.head(2)
```

[7]:

	var1	var2	var3
0	1	2	3
1	4	5	6

[8]:
```
df.columns = ("col1","col2","col3") #Sutunlari yeniden isimlendirme
df
```

[8]:

	col1	col2	col3
0	1	2	3
1	4	5	6
2	7	8	9

DataFrame Özellikleri

DataFrame Özellikleri

[9]:
```
type(df)
```

[9]:
```
pandas.core.frame.DataFrame
```

[10]:
```
df.axes #Satir ve sutun bilgisi
```

[10]:
```
[RangeIndex(start=0, stop=3, step=1),
 Index(['col1', 'col2', 'col3'], dtype='object')]
```

[11]:
```
df.shape #boyut bilgisi
```

[11]:
```
(3, 3)
```

[12]:
```
df.ndim #boyut sayisi
```

[12]:
```
2
```

[13]:
```
df.size #eleman sayisi
```

[13]:
```
9
```

```
[14]: df.values #DataFrame tipindeki veri yapisinin icersinden
                #Degerleri array tipinde aliyor.
```

```
[14]: array([[1, 2, 3],
             [4, 5, 6],
             [7, 8, 9]])
```

```
[15]: type(df.values)
```

```
[15]: numpy.ndarray
```

Çok önemli!

```
[17]: df.tail(1) #sondan 1. index
```

[17]:

	col1	col2	col3
2	7	8	9

Diğer veri tiplerinde veri oluşturmak için çeşitli formatlar kullandık. Örneğin; NumPy array'i üzerinden oluşturduk list üzerinden oluşturduk ve buna benzer farklı formatlardan oluşturduk. Bu işlemler DataFrame için de geçerlidir.

```
[18]: a = np.array([1,2,3,4,5])
```

```
[21]: pd.DataFrame(a, columns = ["deg1"]) #numpy array'i ile df olusturduk.
```

[21]:

	deg1
0	1
1	2
2	3
3	4
4	5

Filtering Pandas Data Frame

[16]:
```
import pandas as pd

dictionary = {"Name" : ["recep", "ayca", "serdar"],
              "Age" : [19,20,21],
              "Salery" : [4000,4200,4300]}

dataFrame1 = pd.DataFrame(dictionary)

dataFrame1
```

[16]:

	Name	Age	Salery
0	recep	19	4000
1	ayca	20	4200
2	serdar	21	4300

[18]:
```
df1 = dataFrame1.Salery > 4200
df2 = dataFrame1.Age > 20

df_filtrelenmis = dataFrame1[df1 & df2] #filtreleme

df_filtrelenmis
```

[18]:

	Name	Age	Salery
2	serdar	21	4300

[19]:
```
df1
```

[19]:
```
0    False
1    False
2     True
Name: Salery, dtype: bool
```

[20]:
```
df2
```

[20]:
```
0    False
1    False
2     True
Name: Age, dtype: bool
```

DataFrame Eleman İşlemleri

DataFrame Eleman İşlemleri

[1]:
```
import numpy as np
s1 = np.random.randint(10, size=5)
s2 = np.random.randint(10, size=5)
s3 = np.random.randint(10, size=5)
```

[2]:
```
sozluk={"var1":s1,"var2":s2,"var3":s3} #array'lerden sozluk
sozluk
```

[2]:
```
{'var1': array([0, 6, 2, 5, 7]),
 'var2': array([2, 1, 7, 6, 2]),
 'var3': array([3, 2, 3, 2, 3])}
```

[4]:
```
import pandas as pd
df = pd.DataFrame(sozluk) #sozluk'den df
df
```

[4]:

	var1	var2	var3
0	0	2	3
1	6	1	2
2	2	7	3
3	5	6	2
4	7	2	3

[7]:
```
df[0:2] #0'dan 2'ye kadar
```

[7]:

	var1	var2	var3
0	0	2	3
1	6	1	2

[8]:
```
df.index
```

[8]:
```
RangeIndex(start=0, stop=5, step=1)
```

[10]:
```
df.index = ["a","b","c","d","e"]
```

[11]:
```
df
```

[11]:

	var1	var2	var3
a	0	2	3
b	6	1	2
c	2	7	3
d	5	6	2
e	7	2	3

[12]:
```
df.index
```

[12]:
```
Index(['a', 'b', 'c', 'd', 'e'], dtype='object')
```

Eleman Silme

Eleman Silme

[22]:
```
df.drop("a", axis=0) #0 ekseninden "a" indexli satiri sil.
```

[22]:

	var1	var2	var3
b	6	1	2
c	2	7	3
d	5	6	2
e	7	2	3

[17]:
```
df #sildi ancak kaydetmedi.
```

[17]:

	var1	var2	var3
a	0	2	3
b	6	1	2
c	2	7	3
d	5	6	2
e	7	2	3

[23]:
```
df.drop("a", axis=0, inplace=True) #inplace argumani kalici olsun mu? anlamındadir.
```

[24]:
```
df #kalici olarak silindi.
```

[24]:

	var1	var2	var3
b	6	1	2
c	2	7	3
d	5	6	2
e	7	2	3

Fancy ile eleman silme

[25]:
```
#fancy
```

[26]:
```
l = ["c","e"]
```

[27]:
```
df.drop(l, axis=0) #c ve e silindi
```

[27]:

	var1	var2	var3
b	6	1	2
d	5	6	2

Değişkenler için eleman işlemleri

[30]:
```
#Degiskenler icin
```

[31]:
```
df
```

[31]:

	var1	var2	var3
b	6	1	2
c	2	7	3
d	5	6	2
e	7	2	3

[32]:
```
"var1" in df
```

[32]:
```
True
```

[35]:
```
l = ["var1","var4","var2"]
```

[37]:
```
for i in l:
    print(i in df)
```

```
True
False
True
```

Bir değişken oluşturmak isteyelim fakat bu değişkenimizi DataFrame içinde var olan değişkenlerden yapmak istediğimizi düşünelim.

[38]:
```
df
```

[38]:

	var1	var2	var3
b	6	1	2
c	2	7	3
d	5	6	2
e	7	2	3

[39]:
```
df["var4"] = df["var1"] / df["var2"]
```

[40]:
```
df
```

[40]:

	var1	var2	var3	var4
b	6	1	2	6.000000
c	2	7	3	0.285714
d	5	6	2	0.833333
e	7	2	3	3.500000

Değişken Silme

Değişken Silme

[43]:
```
df
```

[43]:

	var1	var2	var3	var4
b	6	1	2	6.000000
c	2	7	3	0.285714
d	5	6	2	0.833333
e	7	2	3	3.500000

[44]:
```
df.drop("var4", axis=1, inplace=True)
df
```

[44]:

	var1	var2	var3
b	6	1	2
c	2	7	3
d	5	6	2
e	7	2	3

[47]:
```
l = ["var1","var2"]
df.drop(l, axis=1)  #Fancy ile silme
```

[47]:

	var3
b	2
c	3
d	2
e	3

Gözlem ve Değişken Seçimi: loc & iloc

Gözlem ve Değişken Seçimi: loc & iloc

[1]:
```
import numpy as np
import pandas as pd
m = np.random.randint(1,30, size=(10,3))
df = pd.DataFrame(m, columns=["var1","var2","var3"])
df
```

[1]:

	var1	var2	var3
0	9	23	14
1	17	12	5
2	4	14	8
3	14	27	6
4	21	28	25
5	3	9	20
6	15	3	18
7	16	27	14
8	9	23	24
9	19	12	14

loc: tanımlandığı şekliyle seçim yapmak için kullanılır

[2]:
```
df.loc[0:3] #veri setinin ilk halindeki indexlemeye sadik kalacak sekilde secim imkani verir.
```

[2]:

	var1	var2	var3
0	9	23	14
1	17	12	5
2	4	14	8
3	14	27	6

iloc: alışık olduğumuz index'leme mantığıyla seçim yapar.

[4]:
```
df.iloc[0:3]
```

[4]:

	var1	var2	var3
0	9	23	14
1	17	12	5
2	4	14	8

```
[5]: df.iloc[0,0]
```

```
[5]: 9
```

```
[10]: df.iloc[:3,:2]
```

[10]:

	var1	var2
0	9	23
1	17	12
2	4	14

```
[11]: df.loc[:3,"var3"]
```

```
[11]: 0    14
      1     5
      2     8
      3     6
      Name: var3, dtype: int32
```

```
[ ]: df.iloc[:3,"var3"] # hata verir.
```

Eğer değişken ya da satırlar ile ilgili mutlak bir değer işaretlemesi yapacaksak bu durumda **loc** kullanmamız gerekiyor.

Yani değişken ismi ile işaretleme yapacaksak **loc** kullanmalıyız.
Index'lere göre işaretleme yapacaksak **iloc** kullanmalıyız.

```
[13]: df.iloc[:3,1:3]
```

[13]:

	var2	var3
0	23	14
1	12	5
2	14	8

```
[14]: df.iloc[:3]["var3"]
```

```
[14]: 0    14
      1     5
      2     8
      Name: var3, dtype: int32
```

Koşullu Eleman İşlemleri

Koşullu Eleman İşlemleri

[1]:
```
import numpy as np
import pandas as pd
m = np.random.randint(1,30, size=(10,3))
df = pd.DataFrame(m, columns=["var1","var2","var3"])
df
```

[1]:

	var1	var2	var3
0	8	14	15
1	5	10	11
2	8	26	10
3	21	8	10
4	24	21	2
5	2	12	9
6	21	28	21
7	25	17	14
8	4	14	11
9	11	19	18

[2]:
```
df.var1
```

[2]:
```
0     8
1     5
2     8
3    21
4    24
5     2
6    21
7    25
8     4
9    11
Name: var1, dtype: int32
```

[4]:
```
df[df.var1 > 15]["var1"]
```

[4]:
```
3    21
4    24
6    21
7    25
Name: var1, dtype: int32
```

[5]:
```
df[(df.var1 > 15) & (df.var3 < 5)]
```

[5]:

	var1	var2	var3
4	24	21	2

[8]:
```
df.loc[(df.var1 > 15),["var1","var2"]]
```

[8]:

	var1	var2
3	21	8
4	24	21
6	21	28
7	25	17

[9]:
```
df[(df.var1 > 15)][["var1","var2"]]
```

[9]:

	var1	var2
3	21	8
4	24	21
6	21	28
7	25	17

Birleştirme (Join) İşlemleri

Birleştirme (Join) İşlemleri

[3]:
```
import pandas as pd
import numpy as np
m = np.random.randint(1,30, size=(5,3))
df1 = pd.DataFrame(m, columns=["var1","var2","var3"])
df1
```

[3]:

	var1	var2	var3
0	12	9	5
1	5	15	12
2	2	28	2
3	20	12	6
4	14	25	19

[5]:
```
df2 = df1 + 99
df2
```

[5]:

	var1	var2	var3
0	111	108	104
1	104	114	111
2	101	127	101
3	119	111	105
4	113	124	118

```
[6]: pd.concat([df1,df2])
```

[6]:

	var1	var2	var3
0	12	9	5
1	5	15	12
2	2	28	2
3	20	12	6
4	14	25	19
0	111	108	104
1	104	114	111
2	101	127	101
3	119	111	105
4	113	124	118

Birleştirme işlemi yaptık fakat indexlerde bir karmaşıklık oldu.

```
[9]: pd.concat([df1,df2],ignore_index = True)
```

[9]:

	var1	var2	var3
0	12	9	5
1	5	15	12
2	2	28	2
3	20	12	6
4	14	25	19
5	111	108	104
6	104	114	111
7	101	127	101
8	119	111	105
9	113	124	118

[12]:
```
df2.columns
```

[12]:
```
Index(['var1', 'var2', 'var3'], dtype='object')
```

[14]:
```
df2.columns = ["var1","var2","deg3"]
df2
```

[14]:

	var1	var2	deg3
0	111	108	104
1	104	114	111
2	101	127	101
3	119	111	105
4	113	124	118

[15]:
```
df1
```

[15]:

	var1	var2	var3
0	12	9	5
1	5	15	12
2	2	28	2
3	20	12	6
4	14	25	19

[16]:
```
pd.concat([df1, df2])
```

[16]:

	var1	var2	var3	deg3
0	12	9	5.0	NaN
1	5	15	12.0	NaN
2	2	28	2.0	NaN
3	20	12	6.0	NaN
4	14	25	19.0	NaN
0	111	108	NaN	104.0
1	104	114	NaN	111.0
2	101	127	NaN	101.0
3	119	111	NaN	105.0
4	113	124	NaN	118.0

Bir veri setinin diğerinde karşılığı olmadığı için böyle bir sorun yaşıyoruz.

Bu sorunu kısmi olarak aşabiliriz.

join = "inner" argümanı ile veri setlerinin kesişimlerini alabiliriz.

[17]:
```
pd.concat([df1,df2], join="inner") #kesisimlerini alir.
```

[17]:

	var1	var2
0	12	9
1	5	15
2	2	28
3	20	12
4	14	25
0	111	108
1	104	114
2	101	127
3	119	111
4	113	124

[57]:
```
pd.concat([df1, df2], axis=1).reindex(df1.index)
```

[57]:

	var1	var2	var3	var1	var2	deg3
0	12	9	5	111	108	104
1	5	15	12	104	114	111
2	2	28	2	101	127	101
3	20	12	6	119	111	105
4	14	25	19	113	124	118

İleri Birleştirme İşlemleri

Birebir Birleştirme

Tüm elemanların iki veri setinde de birebir yer alması durumudur.

```
[1]: import pandas as pd
     df1 = pd.DataFrame({"calisanlar":["Ali","Veli","Ayse","Fatma"],
                         "grup":["Muhasebe","Muhendislik","Muhendislik","IK"]})
     df1
```

[1]:

	calisanlar	grup
0	Ali	Muhasebe
1	Veli	Muhendislik
2	Ayse	Muhendislik
3	Fatma	IK

```
[2]: df2 = pd.DataFrame({"calisanlar":["Ali","Veli","Ayse","Fatma"],
                         "ilk_giris":[2010,2009,2014,2019]})
     df2
```

[2]:

	calisanlar	ilk_giris
0	Ali	2010
1	Veli	2009
2	Ayse	2014
3	Fatma	2019

```
[3]: pd.merge(df1,df2)
```

[3]:

	calisanlar	grup	ilk_giris
0	Ali	Muhasebe	2010
1	Veli	Muhendislik	2009
2	Ayse	Muhendislik	2014
3	Fatma	IK	2019

Merge() Fonksiyonu birleştirme işleminin hangi değişkene göre yapılacağını kendisi anlıyor. Eğer bunu belirtmek istersek **on** argümanı aracılığı ile belirtebiliriz.

Her iki veri setinde de calisanlar olduğu için bu veri setlerini calisanlar'a göre birleştirdi.

```
[8]: pd.merge(df1, df2, on="calisanlar")
```

[8]:

	calisanlar	grup	ilk_giris
0	Ali	Muhasebe	2010
1	Veli	Muhendislik	2009
2	Ayse	Muhendislik	2014
3	Fatma	IK	2019

Many to one (Çoktan teke)

Many to one (Çoktan teke)

```
[9]: df3 = pd.merge(df1,df2)
     df3
```

[9]:

	calisanlar	grup	ilk_giris
0	Ali	Muhasebe	2010
1	Veli	Muhendislik	2009
2	Ayse	Muhendislik	2014
3	Fatma	IK	2019

```
[10]: df4 = pd.DataFrame({"grup":["Muhasebe","Muhendislik","IK"],
                          "mudur":["Caner","Mustafa","Berkcan"]})
      df4
```

[10]:

	grup	mudur
0	Muhasebe	Caner
1	Muhendislik	Mustafa
2	IK	Berkcan

```
[14]: pd.merge(df3,df4) #Many to one birlestirme.
```

[14]:

	calisanlar	grup	ilk_giris	mudur
0	Ali	Muhasebe	2010	Caner
1	Veli	Muhendislik	2009	Mustafa
2	Ayse	Muhendislik	2014	Mustafa
3	Fatma	IK	2019	Berkcan

Many to Many (Çoktan çoka)

Many to Many (Çoktan çoka)

[19]:
```
df5 = pd.DataFrame({'grup' : ['Muhasebe','Muhasebe','Muhendislik','Muhendislik','IK','IK'],
                    'yetenekler' : ['Matematik','Excel','Kodlama','Linux','Excel','Yonetim']})
df5
```

[19]:

	grup	yetenekler
0	Muhasebe	Matematik
1	Muhasebe	Excel
2	Muhendislik	Kodlama
3	Muhendislik	Linux
4	IK	Excel
5	IK	Yonetim

[17]:
```
df1
```

[17]:

	calisanlar	grup
0	Ali	Muhasebe
1	Veli	Muhendislik
2	Ayse	Muhendislik
3	Fatma	IK

[21]:
```
pd.merge(df1,df5) #Many to Many
```

[21]:

	calisanlar	grup	yetenekler
0	Ali	Muhasebe	Matematik
1	Ali	Muhasebe	Excel
2	Veli	Muhendislik	Kodlama
3	Veli	Muhendislik	Linux
4	Ayse	Muhendislik	Kodlama
5	Ayse	Muhendislik	Linux
6	Fatma	IK	Excel
7	Fatma	IK	Yonetim

Aggregation & Grouping (Toplulaştırma ve Gruplama)

Basit toplulaştırma fonksiyonları:

- count()
- first()
- last()
- mean()
- median()
- min()
- max()
- std()
- var()
- sum()

[1]:
```
import seaborn as sns #Bu kütüphanemiz içerisindeki bazı veri setlerini kullanıcaz.
```

[6]:
```
df = sns.load_dataset("planets") #planets isimli dataset'i kullandik.
df
```

[6]:

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300000	7.10	77.40	2006
1	Radial Velocity	1	874.774000	2.21	56.95	2008
2	Radial Velocity	1	763.000000	2.60	19.84	2011
3	Radial Velocity	1	326.030000	19.40	110.62	2007
4	Radial Velocity	1	516.220000	10.50	119.47	2009
...	...	...	...	...	...	...
1030	Transit	1	3.941507	NaN	172.00	2006
1031	Transit	1	2.615864	NaN	148.00	2007
1032	Transit	1	3.191524	NaN	174.00	2007
1033	Transit	1	4.125083	NaN	293.00	2008
1034	Transit	1	4.187757	NaN	260.00	2008

1035 rows × 6 columns

[7]:
```
df.head()
```

[7]:

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009

Artık satırlara **gözlem**, sütunlara ise **değişken** demeye alışmalıyız.

[8]:
```
df.shape
```

[8]:
```
(1035, 6)
```

dataset'in 1035 gözlem, 6 değişkenden oluştuğunu gözlemlemekteyiz.

[9]:
```
df.mean() #Tüm değişkenlerin ortalamaları
```

[9]:
```
number               1.785507
orbital_period    2002.917596
mass                 2.638161
distance           264.069282
year              2009.070531
dtype: float64
```

[10]:
```
df["mass"].mean() # mass değişkeninin ortalaması.
```

[10]:
```
2.6381605847953216
```

[12]:
```
df.count() #Degiskenlerdeki gözlem sayıları.
```

[12]:
```
method            1035
number            1035
orbital_period     992
mass               513
distance           808
year              1035
dtype: int64
```

[13]:
```
df.min() #Minimum değerler
```

[13]:
```
method            Astrometry
number                     1
orbital_period     0.0907063
mass                  0.0036
distance                1.35
year                    1989
dtype: object
```

[14]:
```
df.max() #Maximum değerler
```

[14]:
```
method            Transit Timing Variations
number                                    7
orbital_period                       730000
mass                                     25
distance                               8500
year                                   2014
dtype: object
```

```
[15]: df.sum() # Değişkenlerin değerlerinin toplamı
```

```
[15]: method            Radial VelocityRadial VelocityRadial VelocityR...
      number                                                        1848
      orbital_period                                          1.98689e+06
      mass                                                        1353.38
      distance                                                     213368
      year                                                        2079388
      dtype: object
```

```
[16]: df.std() #Değişkenlerin standart sapması
```

```
[16]: number               1.240976
      orbital_period    26014.728304
      mass                  3.818617
      distance            733.116493
      year                  3.972567
      dtype: float64
```

```
[17]: df.var() #Değişkenlerin varyansı
```

```
[17]: number            1.540022e+00
      orbital_period    6.767661e+08
      mass              1.458183e+01
      distance          5.374598e+05
      year              1.578129e+01
      dtype: float64
```

```
[18]: df.describe() #Verisetindeki tüm değişkenleri betimsel istatistikleri anlamında görebiliyoruz.
```

[18]:

	number	orbital_period	mass	distance	year
count	1035.000000	992.000000	513.000000	808.000000	1035.000000
mean	1.785507	2002.917596	2.638161	264.069282	2009.070531
std	1.240976	26014.728304	3.818617	733.116493	3.972567
min	1.000000	0.090706	0.003600	1.350000	1989.000000
25%	1.000000	5.442540	0.229000	32.560000	2007.000000
50%	1.000000	39.979500	1.260000	55.250000	2010.000000
75%	2.000000	526.005000	3.040000	178.500000	2012.000000
max	7.000000	730000.000000	25.000000	8500.000000	2014.000000

```
[19]: df.describe().T #Transpozu alındığında
```

[19]:

	count	mean	std	min	25%	50%	75%	max
number	1035.0	1.785507	1.240976	1.000000	1.00000	1.0000	2.000	7.0
orbital_period	992.0	2002.917596	26014.728304	0.090706	5.44254	39.9795	526.005	730000.0
mass	513.0	2.638161	3.818617	0.003600	0.22900	1.2600	3.040	25.0
distance	808.0	264.069282	733.116493	1.350000	32.56000	55.2500	178.500	8500.0
year	1035.0	2009.070531	3.972567	1989.000000	2007.00000	2010.0000	2012.000	2014.0

Elimizdeki verisetinin eksik gözlemleri silip describe yapmak istediğimizde **dropna()** fonskiyonunu kullanırız.

[20]:
```
df.dropna().describe().T
```

[20]:

	count	mean	std	min	25%	50%	75%	max
number	498.0	1.734940	1.175720	1.0000	1.00000	1.000	2.0000	6.0
orbital_period	498.0	835.778671	1469.128259	1.3283	38.27225	357.000	999.6000	17337.5
mass	498.0	2.509320	3.636274	0.0036	0.21250	1.245	2.8675	25.0
distance	498.0	52.068213	46.596041	1.3500	24.49750	39.940	59.3325	354.0
year	498.0	2007.377510	4.167284	1989.0000	2005.00000	2009.000	2011.0000	2014.0

Grouping

Grouping

[26]:
```
import pandas as pd
df = pd.DataFrame({'gruplar' : ['A','B','C','A','B','C'],
                   'veri' : [10,11,52,23,43,55]}, columns=['gruplar','veri'])
df
```

[26]:

	gruplar	veri
0	A	10
1	B	11
2	C	52
3	A	23
4	B	43
5	C	55

Genelde gruplama işlemleri ile toplulaştırma(Aggregation) işlemleri bir arada kullanılır.

[27]:
```
df.groupby("gruplar") #gruplar içerisindeki grupları yakaladı.
```

[27]:
```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x0000018B7EA5D648>
```

[28]:
```
df.groupby("gruplar").mean() # Ortalamasini aldi.
```

[28]:

	veri
gruplar	
A	16.5
B	27.0
C	53.5

```
[30]: df.groupby("gruplar").sum()
```

[30]:

gruplar	veri
A	33
B	54
C	107

```
[33]: import seaborn as sns
      df = sns.load_dataset("planets")
      df.head()
```

[33]:

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009

```
[37]: df.groupby("method")["orbital_period"].describe()
      #method'a göre grupla. orbital_period değişkeninin istatistiki degerlerini al.
```

[37]:

method	count	mean	std	min	25%	50%	75%	max
Astrometry	2.0	631.180000	544.217663	246.360000	438.770000	631.180000	823.590000	1016.000000
Eclipse Timing Variations	9.0	4751.644444	2499.130945	1916.250000	2900.000000	4343.500000	5767.000000	10220.000000
Imaging	12.0	118247.737500	213978.177277	4639.150000	8343.900000	27500.000000	94250.000000	730000.000000
Microlensing	7.0	3153.571429	1113.166333	1825.000000	2375.000000	3300.000000	3550.000000	5100.000000
Orbital Brightness Modulation	3.0	0.709307	0.725493	0.240104	0.291496	0.342887	0.943908	1.544929
Pulsar Timing	5.0	7343.021201	16313.265573	0.090706	25.262000	66.541900	98.211400	36525.000000
Pulsation Timing Variations	1.0	1170.000000	NaN	1170.000000	1170.000000	1170.000000	1170.000000	1170.000000
Radial Velocity	553.0	823.354680	1454.926210	0.736540	38.021000	360.200000	982.000000	17337.500000
Transit	397.0	21.102073	46.185893	0.355000	3.160630	5.714932	16.145700	331.600590
Transit Timing Variations	3.0	79.783500	71.599884	22.339500	39.675250	57.011000	108.505500	160.000000

İleri Toplulaştırma İşlemleri(Aggregate, filter, transform, apply)

Aggregate

Yaptığımız gruplama içerisinde istediğimiz istatistiki değerleri bir arada görmek için *aggregate* kullanırız.

Aggregate

[38]:
```
import pandas as pd
df = pd.DataFrame({"gruplar" : ["A","B","C","A","B","C"],
                   "degisken1" : [10,23,33,22,11,99],
                   "degisken2" : [100,253,333,262,111,969]},
                  columns = ["gruplar","degisken1","degisken2"])
df
```

[38]:

	gruplar	degisken1	degisken2
0	A	10	100
1	B	23	253
2	C	33	333
3	A	22	262
4	B	11	111
5	C	99	969

[40]:
```
df.groupby("gruplar").mean()
```

[40]:

gruplar	degisken1	degisken2
A	16	181
B	17	182
C	66	651

```
[45]: import numpy as np
      df.groupby("gruplar").aggregate(["min",np.median, max])

      #Yaptığımız gruplama içerisinde kendi istediğimiz istatistiki değerleri
      #bir arada görmek için aggragate kullanırız.
```

[45]:

	degisken1			degisken2		
	min	median	max	min	median	max
gruplar						
A	10	16	22	100	181	262
B	11	17	23	111	182	253
C	33	66	99	333	651	969

İki değişken için iki ayrı istatistiki hesaplama yapmak istiyoruz.

```
[49]: df.groupby("gruplar").aggregate({"degisken1" : "min", "degisken2" : np.median})
```

[49]:

	degisken1	degisken2
gruplar		
A	10	181
B	11	182
C	33	651

filter

Filter

Pandas'ın sunduğu özelliklerden daha karmaşık bir isteğimiz olduğunda kendi fonksiyonumuzu yazıp ona göre filtreleyebiliriz.

[1]:
```
import pandas as pd
df = pd.DataFrame({"gruplar" : ["A","B","C","A","B","C"],
                   "degisken1" : [10,23,33,22,11,99],
                   "degisken2" : [100,253,333,262,111,969]},
                  columns = ["gruplar","degisken1","degisken2"])
df
```

[1]:

	gruplar	degisken1	degisken2
0	A	10	100
1	B	23	253
2	C	33	333
3	A	22	262
4	B	11	111
5	C	99	969

[6]:
```
def filter_func(x):
    return x["degisken1"].std() > 9
#degisken1'e göre standart sapması 9'dan büyük olan değerler
```

[7]:
```
df.groupby("gruplar").std() #standart sapmalar
```

[7]:

gruplar	degisken1	degisken2
A	8.485281	114.551299
B	8.485281	100.409163
C	46.669048	449.719913

[8]:
```
df.groupby("gruplar").filter(filter_func)
```

[8]:

	gruplar	degisken1	degisken2
2	C	33	333
5	C	99	969

Aynı işlemin **lambda** ile çözümü:

[9]:
```
df.groupby("gruplar").filter(lambda x : x["degisken1"].std() > 9)
```

[9]:

	gruplar	degisken1	degisken2
2	C	33	333
5	C	99	969

transform

transform

Kendi tanımladığımız bir fonksiyonu değişkenler üzerinde uygulayabiliyoruz.

[2]:
```
import pandas as pd
df = pd.DataFrame({"gruplar" : ["A","B","C","A","B","C"],
                   "degisken1" : [10,23,33,22,11,99],
                   "degisken2" : [100,253,333,262,111,969]},
                  columns = ["gruplar","degisken1","degisken2"])
df
```

[2]:

	gruplar	degisken1	degisken2
0	A	10	100
1	B	23	253
2	C	33	333
3	A	22	262
4	B	11	111
5	C	99	969

[3]:
```
df["degisken1"]*9
```

[3]:
```
0     90
1    207
2    297
3    198
4     99
5    891
Name: degisken1, dtype: int64
```

[4]:
```
df_a = df.iloc[:, 1:3]
df_a
```

[4]:

	degisken1	degisken2
0	10	100
1	23	253
2	33	333
3	22	262
4	11	111
5	99	969

Birazdan yapacağımız işlem sayısal bir işlem olduğundan gruplar değişkenine uygulamak istediğimizde hata ile karşılaşmamak için, yukarıdaki gibi gruplar'ı ayrı tutmamız gerekiyor.

[5]:
```
df_a.transform(lambda x : x-x.mean())
#yakalamış olduğu bütün elemanlardan o değişkenin ortalamasını çıkartacak.
```

[5]:

	degisken1	degisken2
0	-23.0	-238.0
1	-10.0	-85.0
2	0.0	-5.0
3	-11.0	-76.0
4	-22.0	-227.0
5	66.0	631.0

Apply

Apply

[4]:
```
import pandas as pd
import numpy as np
df = pd.DataFrame({"degisken1" : [10,23,33,22,11,99],
                   "degisken2" : [100,253,333,262,111,969]},
                  columns = ["degisken1","degisken2"])
df
```

[4]:

	degisken1	degisken2
0	10	100
1	23	253
2	33	333
3	22	262
4	11	111
5	99	969

apply() fonksiyonu tıpkı transform fonksiyonu ve filter fonksiyonu gibi değişkenlerin üzerinde gezinme yeteneği olan ve **aggregation**(toplulaştırma) amacıyla kullanılacak olan bir fonksiyondur.

[9]:
```
df.apply(np.sum)
```

[9]:
```
degisken1     198
degisken2    2028
dtype: int64
```

[8]:
```
df.apply(np.mean)
```

[8]:
```
degisken1     33.0
degisken2    338.0
dtype: float64
```

[12]:
```
df = pd.DataFrame({"gruplar" : ["A","B","C","A","B","C"],
                   "degisken1" : [10,23,33,22,11,99],
                  "degisken2" : [100,253,333,262,111,969]},
                 columns = ["gruplar","degisken1","degisken2"])
df
```

[12]:

	gruplar	degisken1	degisken2
0	A	10	100
1	B	23	253
2	C	33	333
3	A	22	262
4	B	11	111
5	C	99	969

[14]:
```
df.groupby("gruplar").apply(np.sum)
```

[14]:

	gruplar	degisken1	degisken2
gruplar			
A	AA	32	362
B	BB	34	364
C	CC	132	1302

[18]:
```
df
```

[18]:

	gruplar	degisken1	degisken2
0	A	10	100
1	B	23	253
2	C	33	333
3	A	22	262
4	B	11	111
5	C	99	969

[26]:
```
df.groupby("gruplar").apply(lambda x : (x["degisken1"]-x["degisken2"]))
```

[26]:
```
gruplar
A        0    -90
         3   -240
B        1   -230
         4   -100
C        2   -300
         5   -870
dtype: int64
```

Bu işlemi apply ile yapabiliriz ancak transform hata verir.

Pivot Tablolar

Veri setleri üzerinde bazı satır ve sütun işlemleri yaparak, veri setini amaca uygun hale getirmek için kullanılan yapılardır.

groupby()'ın çok boyutlu versiyonu olarak düşünülebilir.

```
[27]: import pandas as pd
      import seaborn as sns
      titanic = sns.load_dataset("titanic")
      titanic
```

[27]:

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
886	0	2	male	27.0	0	0	13.0000	S	Second	man	True	NaN	Southampton	no	True
887	1	1	female	19.0	0	0	30.0000	S	First	woman	False	B	Southampton	yes	True
888	0	3	female	NaN	1	2	23.4500	S	Third	woman	False	NaN	Southampton	no	False
889	1	1	male	26.0	0	0	30.0000	C	First	man	True	C	Cherbourg	yes	True
890	0	3	male	32.0	0	0	7.7500	Q	Third	man	True	NaN	Queenstown	no	True

891 rows × 15 columns

Cinsiyete göre gruplayıp hayatta olma ortalamalarına bakalım.

```
[30]: titanic.groupby("sex")["survived"].mean()
```

```
[30]: sex
      female    0.742038
      male      0.188908
      Name: survived, dtype: float64
```

```
[31]: titanic.groupby("sex")[["survived"]].mean()  #Basit bir pivot işlemi.
      # değişken etrafına köşeli parantez ekleyerek dataset olarak gözlemleyelim.
```

[31]:

sex	survived
female	0.742038
male	0.188908

Cinsiyete ve class'a göre ölüm ortalamaları:

[37]:
```
titanic.groupby(["sex","class"])[["survived"]].aggregate("mean")
```

[37]:

		survived
sex	**class**	
female	**First**	0.968085
	Second	0.921053
	Third	0.500000
male	**First**	0.368852
	Second	0.157407
	Third	0.135447

[39]:
```
titanic.groupby(["sex","class"])["survived"].aggregate("mean").unstack()
#unstack bizi hiyerarsik görünümden kurtarır.,
```

[39]:

class	First	Second	Third
sex			
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447

Bir boyut daha ekleyerek pivot table oluşturduk.

pivot_table

[40]:
```
#Pivot ile table
```

[41]:
```
titanic.pivot_table("survived", index = "sex", columns = "class")
```

[41]:

class	First	Second	Third
sex			
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447

```
[45]: titanic.age
```

```
[45]: 0      22.0
      1      38.0
      2      26.0
      3      35.0
      4      35.0
             ...
      886    27.0
      887    19.0
      888     NaN
      889    26.0
      890    32.0
      Name: age, Length: 891, dtype: float64
```

Bu sürekli değişkeni bir kategorik değişkene çevirip, bu kategorik değişkenin sınıflarını da pivot table'a boyut olarak ekleyelim.

```
[51]: age = pd.cut(titanic["age"], [0,18,90])
      age.head(15)
```

```
[51]: 0     (18.0, 90.0]
      1     (18.0, 90.0]
      2     (18.0, 90.0]
      3     (18.0, 90.0]
      4     (18.0, 90.0]
      5              NaN
      6     (18.0, 90.0]
      7      (0.0, 18.0]
      8     (18.0, 90.0]
      9      (0.0, 18.0]
      10     (0.0, 18.0]
      11    (18.0, 90.0]
      12    (18.0, 90.0]
      13    (18.0, 90.0]
      14     (0.0, 18.0]
      Name: age, dtype: category
      Categories (2, interval[int64]): [(0, 18] < (18, 90]]
```

Sürekli bir değişken olan age değişkenini kategorik değişken haline getirdik.

```
[58]: titanic.pivot_table("survived",["sex",age], "class")
```

[58]:

	class	First	Second	Third
sex	**age**			
female	**(0, 18]**	0.909091	1.000000	0.511628
	(18, 90]	0.972973	0.900000	0.423729
male	**(0, 18]**	0.800000	0.600000	0.215686
	(18, 90]	0.375000	0.071429	0.133663

Dış Kaynaklı Veri Okuma

Dış Kaynaklı Veri Okuma

.txt formunu ve **.csv** formunu aynı fonksiyon ile okuyabiliyoruz.

[1]:
```
import pandas as pd
```

[2]:
```
pd.read_csv("reading_data/ornekcsv.csv")
```

[2]:

	a;b;c
0	78;12;1
1	78;12;2
2	78;324;3
3	7;2;4
4	88;23;5
5	6;2;
6	56;11;6
7	7;12;7
8	56;21;7
9	346;2;8
10	5;1;8
11	456;21;8
12	3;12;88

Veri düzgün biçimde gelmedi. Veri okuma işlemlerinde en sık karşılaşılan problem budur.
Paylaşılan veri seti genellikle csv ya da txt formunda olduğunda değişkenler birbirinden bazı ayraçlar ile ayrılır.

Ayraç olarak öntanımlı değer **","**'dür. Bize gelen veride ise **";"** kullanılmış. **sep** argümanı ile bu sorunu çözebiliriz.

c

c

cc

csv okuma

[4]:
```
#csv okuma
pd.read_csv("reading_data/ornekcsv.csv", sep=";")
```

[4]:

	a	b	c
0	78	12	1.0
1	78	12	2.0
2	78	324	3.0
3	7	2	4.0
4	88	23	5.0
5	6	2	NaN
6	56	11	6.0
7	7	12	7.0
8	56	21	7.0
9	346	2	8.0
10	5	1	8.0
11	456	21	8.0
12	3	12	88.0

txt okuma

```
[6]: #txt okuma
     pd.read_csv("reading_data/duz_metin.txt")
```

[6]:

	1 2
0	2 2
1	3 2
2	4 2
3	5 2
4	6 2
5	7 2
6	8 2
7	9 2
8	10 2

sep argümanını kullanmadık fakat veriler düzgün biçimde geldi.

Düz metinlerde arada boşluk olsa dahi bunu fonksiyon görebiliyor.

Excel dosyası okuma

Excel Dosyası Okuma

[9]:
```
pd.read_excel("reading_data/ornekx.xlsx")
```

[9]:

	a	b	c
0	78	12	1.0
1	78	12	2.0
2	78	324	3.0
3	7	2	4.0
4	88	23	5.0
5	6	2	NaN
6	56	11	6.0
7	7	12	7.0
8	56	21	7.0
9	346	2	8.0
10	5	1	8.0
11	456	21	8.0
12	3	12	88.0

[10]:
```
df = pd.read_excel("reading_data/ornekx.xlsx")
```

[11]:
```
type(df)
```

[11]:
```
pandas.core.frame.DataFrame
```

DataFrame'lere yaptığımız tüm işlemleri artık burada da yapabiliriz.

[12]:
```
df.head()
```

[12]:

	a	b	c
0	78	12	1.0
1	78	12	2.0
2	78	324	3.0
3	7	2	4.0
4	88	23	5.0

```
[14]: df.columns = ("A", "B", "C") #column isimlerini değiştirdik.
      df
```

[14]:

	A	B	C
0	78	12	1.0
1	78	12	2.0
2	78	324	3.0
3	7	2	4.0
4	88	23	5.0
5	6	2	NaN
6	56	11	6.0
7	7	12	7.0
8	56	21	7.0
9	346	2	8.0
10	5	1	8.0
11	456	21	8.0
12	3	12	88.0

Sıfırdan txt okuma

GitHub'da tek bir veri setini almak istediğimizde, veri setinin olduğu sayfada **Raw**
butonuna tıklayıp verisetinin ham haline ulaşabiliriz. Buradaki verileri txt dosyasına
kopyalayıp kullanabiliriz.

```
[15]: #sıfırdan txt okuma
      tips = pd.read_csv("reading_data/data.txt")
```

```
[19]: tips.head()
```

[19]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Pandas Alıştırmalar-1

Question 1:

"df" isimli bir Pandas DataFrame için ilk 2 gözleme erişmek istenilirse aşağıdaki kodlardan hangisi kullanılır?

- ○ df.head()
- ○ df.tail()
- ○ df.describe()
- ● df.head(2)

Question 2:

"df" isimli bir Pandas DataFrame için son 3 gözleme erişmek istenilirse aşağıdaki kodlardan hangisi kullanılır?

- ○ df.head(3)
- ● df.tail(3)
- ○ df.describe()
- ○ df.head()

Question 3:

```
seri = pd.Series([121,200,150,99], argüman_ismi = ["reg","loj","cart","rf"])
```

Yukarıda "argüman_ismi" yazan bölüme aşağıdakilerden hangisi gelmelidir?

- ○ columns
- ○ column
- ○ indexies
- ● index

Question 4:

Bir Pandas DataFrame oluştururken *değişken isimlerini belirtmek için* hangi argüman kullanılır?

- variable
- variables
- column
- columns

Question 5:

Solda verilen "df" isimli Pandas DataFrame için aşağıdaki seçeneklerden hangisi uygulanırsa sağdaki çıktıya ulaşılır?

	col1	col2	col3
a	9	2	7
b	3	3	4
c	2	9	8
d	1	7	0
e	0	5	6

	col1	col2	col3
c	2	9	8
d	1	7	0
e	0	5	6

- df[1:3]
- df[1:3,:]
- df["c":"e"]
- df["col1", "col3"]

Question 7:

Pandas DataFrame üzerinde *indeks isimlendirmelerine bağlı kalarak* (label based) gözlem ve değişken seçimi yapmak için .... kullanılır. Boşluğa aşağıdakilerden hangisi gelmelidir?

- (●) loc
- ( ) iloc
- ( ) slice
- ( ) fancy index

Question 6:

Pandas DataFrame üzerinde hem gözlem hem değişken seçimi için *indeks isimlendirmelerinden (labellardan) bağımsız* seçim yapmak üzere .... kullanılır. Boşluğa aşağıdakilerden hangisi gelmelidir?

- ( ) loc
- (●) iloc
- ( ) slice
- ( ) fancy index

Question 8:

```
seri = pd.Series([121,200,150,99])
seri.values
```

Yukarıda verilen kodun çıktısı aşağıdakilerden hangisidir?

- pd.DataFrame([121, 200, 150, 99])
- array(["reg","loj","cart","rf"])
- pd.Series([121, 200, 150, 99])
- array([121, 200, 150, 99])

Question 9:

```
import numpy as np
m = np.arange(1,7).reshape((3,2))
pd.DataFrame(m, columns = ["var1","var2"])
```

Yukarıda kodu verilen kodun çıktısı hangisidir?

```
     var1 var2

var1  1    2

var1  3    4

var1  5    6
```

```
    var1 var2

0    1    2

1    3    4

2    5    6
```

Question 10:

Aşağıda bir kod ve çıktısı verilmiştir. Buna göre hangisi bir Pandas DataFrame oluşturur?

Kod:

`type(sozluk)`

Çıktı:

`dict`

```
import numpy as np
np.DataFrame(dict)
```

```
import numpy as np
np.DataFrame(sozluk)
```

```
import pandas as pd
pd.DataFrame(dict)
```

```
import pandas as pd
pd.DataFrame(sozluk)
```

Pandas Alıştırmalar-2

Question 1:

Verilen kod parçasına göre aşağıdakilerden hangisi yanlıştır?

```
import numpy as np
import pandas as pd

m = np.random.randint(1,30, size = (10,3))
df = pd.DataFrame(m, columns = ["var1","var2","var3"])
```

- Yukarıdaki kod parçasının 3. satırında , 10x3'lük rastgele integer değerlerden numpy array oluşturma işlemi yapılmıştır.
- df[df.var1 > 15] ile df'nin 1. kolonuna bir filtreleme yapılabilir
- Pandas DataFrame oluşturmayı tamamlamak için numpy array yapısı sözlük yapısına çevrilmelidir
- 'pd', 'pandas' kütüphanesine verilen takma isimdir. 'pan' seklinde de kullanılabilir

Question 2:

Verilen kod parçası ile ilgili aşağıdakilerden hangisi yanlıştır?

```
df[(df.var1 > 15)][[ "var1","var2"]]
```

- [["var1","var2"]] ifadesinde çift köşeli parantez kullanılmasının sebebi, çıktının tablo şeklinde gösterilmesi içindir
- Çıktının türü Pandas DataFrame olacaktır
- df[(df.var1 > 15)]["var1"] kodu ile bir değişken kolonu seçilebilir ve türü pandas.Series olur
- Çıktıda değişken isimleri görünmez

Question 3:

df1 Pandas DataFrame olarak tanımlanmıştır. Verilen koda göre hangisi yanlıştır?

```
df2 = df1 + 99
```

- Çalışmaz
- İlk satırın her elemanına 99 eklenir
- İlk sütunun her elemanına 99 eklenir
- Her elemana 99 eklenir Cevapta yanlışlık olabilir.

Question 4:

df1 ve df2 Pandas DataFrame olarak tanımlanmıştır. Verilen kod parçası ile ilgili aşağıdakilerden hangisi yanlıştır?

```
pd.concat([df1,df2])
```

- Kolon adları aynı ise satır bazında alçalta birleşirler
- "ignore_index = True" argümanı ile oluşan DataFrame indeksleri gösterilmez
- Kolon adları aynı ise sütun bazında alçalta birleşirler
- Kolon adları farklı ise uyarı hatası verir

Question 5:

Verilen df1 ve df2 ile ilgili aşağıdakilerden hangisi yanlıştır?

df1:

	calisanlar	grup
0	Ali	Muhasebe
1	Veli	Muhendislik
2	Ayse	Muhendislik
3	Fatma	IK

df2:

	calisanlar	ise_giris
0	Ayse	2010
1	Ali	2009
2	Veli	2014
3	Fatma	2019

- pd.merge(df1, df2) kodu, ortak olan 'calisanlar' değişkeni üzerinden dataframe'leri sütun bazında birleştirir
- pd.merge(df1, df2) ile Indexlere gore değil, ortak verilere göre birleştirilir
- pd.merge(df1, df2) sonucu toplam 3 sütun oluşur
- pd.merge(df1, df2) sonucu toplam 8 satır oluşur

Question 6:

Pandas kütüphanesinin merge fonksiyonu için yapılan genellemelerden hangisi yanlıştır?

- Dataframe'leri birleştirir
- Dataframlerdeki ortak kolon varsa bu kolon bir defa yazılır
- Dataframe'leri kolon bazında (yanyana) birleştirir
- on='colon' argümanı kullanmaksızın çalışmaz

Question 7:

Aşağıdakilerden hangisi toplulaştırma (aggregation) fonksiyonlarından biri değildir?

- count()
- top()
- last()
- min()

Question 8:

Aşağıdakilerden hangisi yanlıştır?

- var() varyans'ı hesaplar.
- median() en çok tekrar eden veriyi gösterir.
- mean() ortalamayı hesaplar.
- std() standart sapmayı hesaplar.

Question 10:

Dataframe'lere uygulanan "describe()" metodunun çıktısında hangi bilgi yoktur?

- (●) Sum
- ( ) Mean
- ( ) Count
- ( ) Median

Pandas Alıştırmalar – 3

Question 1:

```
df.groupby("gruplar").aggregate([min, np.median, max])
```

yukarıdaki kod ile ne amaçlanmıştır?

- ○ df dataframe'i gruplamak, sırasıyla her satıra min, np.median ve max fonksiyonu uygulamak
- ○ df dataframe'i gruplamak, 1. gruba min, 2. gruba np.median ve 3. gruba max fonksiyonu uygulamak
- ● df dataframe'i gruplamak, sırasıyla her sütun için min, np.median ve max fonksiyonlarını uygulamak ve her sütun için bu 3 sonucu birer sütun olarak göstermek
- ○ df dataframe'i gruplamak, sırasıyla her sütuna min, np.median ve max fonksiyonlarını uygulamak, her sütun için nihai sonuç tek sütun olacak şekilde göstermek

Question 2:

"gruplar" dışında 2 kolona sahip olan "df" isimli Pandas DataFrame önce gruplamak sonra da 1. kolona min fonksiyonu, 2.kolona max fonksiyonu uygulanmak isteniyor. Hangi seçenek doğrudur?

- ● df.groupby("gruplar").aggregate({"degisken1": "min", "degisken2": "max"})
- ○ df.groupby("gruplar").agg(["degisken1": "min", "degisken2": "max"])
- ○ df.groupby("gruplar").agg({"degisken1": "max", "degisken2": "min"})
- ○ df.groupby("gruplar").aggregate(["degisken1": "min", "degisken2": "max"])

Question 3:

Aşağıda çıktısı verilen kod aşağıdakilerden hangisi olabilir?

Çıktı:

class	First	Second	Third
sex			
female	0.968085	0.921053	0.500000
male	0.368852	0.157407	0.135447

- titanic.pivot_table("survived", columns = "sex", index = "class")
- titanic.pivot_table("survived", index = "sex", columns = "class")

Question 4:

Aşağıdakilerden hangileri doğrudur?

I. unstack() fonksiyonu çoklu indeks yapısındaki dataframe'i enine genişletir.

II. unstack() fonksiyonu çoklu indeks yapısındaki dataframe'in bir indeksini kolon başlığı olarak yer değiştirir.

III. stack() fonksiyonu unstack() fonksiyonunun tersidir.

- I,II,III

Question 5:

Aşağıdakilerden hangisi doğrudur?

- ( ) import pandas as np
  pd.read_csv("ornekcsv.csv")
- ( ) import pandas as pd
  np.read_csv("ornekcsv.csv")
- (•) import pandas as pd
  pd.read_csv("ornekcsv.csv")
- ( ) import pandas as np
  np.readcsv("ornekcsv.csv")

Question 6:

Aşağıdakilerden hangisi yanlıştır? (pandas'ın import edildiğini varsayınız)

- ( ) pd.read_csv("reading_data/ornekcsv.csv", sep = ";")
- (•) pd.read_txt("reading_data/duz_metin.txt")
- ( ) pd.read_excel("reading_data/ornekx.xlsx")
- ( ) pd.read_csv("reading_data/duz_metin.txt")

Question 7:

"numbers" değişkeninin bir Pandas Serisi olduğu bilindiğine göre aşağıdaki kodun çıktısı hangisi olabilir?

Kod:

```
numbers.dtype
```

- (●) dtype('int64')
- ( ) str
- ( ) dtype(string)
- ( ) dtype(boolean)

Question 8:

Bir Pandas Serisine "ndim" metodu uygulanırsa ne bilgisini almış oluruz?

- ( ) Eleman sayısı
- ( ) Uzunluğu
- (●) Boyutu
- ( ) Hata verir

Question 9:

Bir Pandas Serisini array olarak almak istersek hangi metodu uygulamak gerekir?

- ( ) column
- ( ) value
- ( ) value()
- (●) values

Question 10:

"seri" değişkeninin bir Pandas Serisi olduğu bilindiğine göre aşağıdaki kod ile ne amaçlanmıştır?

Kod:

```
"KNN" in seri
```

- ○ "KNN" ifadesini seriye dönüştürmek
- ○ "KNN" ifadesini seriye eklemek
- ● "KNN" ifadesinin "seri" içinde olup olmadığını sorgulamak
- ○ "KNN" ifadesinin türünü değiştirmek

List Comprehensions

[17]:
```
import pandas as pd
dataset = {"NAME" : ["recep", "ayca", "serdar"],
           "MAAS" : [6000, 6500, 5900]}
```

[18]:
```
dataFrame1 = pd.DataFrame(dataset)
dataFrame1
```

[18]:

	NAME	MAAS
0	recep	6000
1	ayca	6500
2	serdar	5900

[25]:
```
ortalama_maas = dataFrame1.MAAS.mean()
print(ortalama_maas)
```

```
6133.333333333333
```

[27]:
```
dataFrame1["Maas Durumu"] = ["Dusuk" if ortalama_maas>each else "Yuksek" for each in dataFrame1.MAAS]
dataFrame1
```

[27]:

	NAME	MAAS	Maas Durumu
0	recep	6000	Dusuk
1	ayca	6500	Yuksek
2	serdar	5900	Dusuk

[28]:
```
dataFrame1.columns = [each.lower() for each in dataFrame1.columns]
dataFrame1
```

[28]:

	name	maas	maas durumu
0	recep	6000	Dusuk
1	ayca	6500	Yuksek
2	serdar	5900	Dusuk

[29]:
```
#columns'da bosluklarin yerine _ koyalim.
dataFrame1.columns = [each.split()[0]+"_"+each.split()[1] if len(each.split())>1 else each for each in dataFrame1.columns]
dataFrame1
```

[29]:

	name	maas	maas_durumu
0	recep	6000	Dusuk
1	ayca	6500	Yuksek
2	serdar	5900	Dusuk

--Python ile Veri Görselleştirme--

Seaborn

Python ile Veri Görselleştirme Giriş

- Büyük Resmi Görmek ve Veriyi Temsil Etmek
- Veriye İlk Bakış
- Kategorik Değişken Özetleri
- Sürekli Değişken Özetleri
- Dağılım Grafikleri
- Korelasyon Grafikleri
- Çizgi Grafikler
- Zaman Serisi Grafikleri

Basit bir grafik, veri analistinin zihnine diğer herhangi bir cihazdan daha fazla bilgi getirir.

John Tukey

Keşifçi Veri Analizi Nedir?

Betimsel istatistikler, veri görselleştirme teknikleri ve iş çıktısı hedefiyle veri üzerinde çalışmaktır.

Veri Görselleştirme Kütüphaneleri

- Matplotlib
- Pandas
- Seaborn
- ggplot
- Bokeh
- Plot.ly

Veriye İlk Bakış

[2]:
```
import seaborn as sns
planets = sns.load_dataset("planets")
planets
```

[2]:

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300000	7.10	77.40	2006
1	Radial Velocity	1	874.774000	2.21	56.95	2008
2	Radial Velocity	1	763.000000	2.60	19.84	2011
3	Radial Velocity	1	326.030000	19.40	110.62	2007
4	Radial Velocity	1	516.220000	10.50	119.47	2009
...	...	...	...	...	...	...
1030	Transit	1	3.941507	NaN	172.00	2006
1031	Transit	1	2.615864	NaN	148.00	2007
1032	Transit	1	3.191524	NaN	174.00	2007
1033	Transit	1	4.125083	NaN	293.00	2008
1034	Transit	1	4.187757	NaN	260.00	2008

1035 rows × 6 columns

Veri Setinin Hikayesi Nedir?

Veriye ilk bakış demek teorik olarak verisetinin nasıl oluştuğunun sorgulanmasıdır.

Bu veriseti NASA'nın yayınladığı galaksi keşfi ile ilgili bir veri setidir.

- **method:** gezegenlerin/galaksilerin bulunma şeklini ifade etmektedir.
- **number:** bulunan sistemlerdeki gezegen sayısını ifade etmektedir.
- **orbital_period:** yörünge dönemini ifade etmektedir.
- **mass:** kütleyi ifade etmektedir.
- **distance:** uzaklığı ifade etmektedir.

- **year:** bulunma yılını ifade etmektedir.

[3]:
```
df = planets.copy()
#Orjinal verisetini yedekleyerek yedek üzerinde işlemler yapacağız.
```

[4]:
```
df.head()
```

[4]:

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009

[5]:
```
df.tail()
```

[5]:

	method	number	orbital_period	mass	distance	year
1030	Transit	1	3.941507	NaN	172.0	2006
1031	Transit	1	2.615864	NaN	148.0	2007
1032	Transit	1	3.191524	NaN	174.0	2007
1033	Transit	1	4.125083	NaN	293.0	2008
1034	Transit	1	4.187757	NaN	260.0	2008

Veri Seti Yapısal Bilgileri

[6]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1035 entries, 0 to 1034
Data columns (total 6 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   method          1035 non-null   object
 1   number          1035 non-null   int64
 2   orbital_period  992 non-null    float64
 3   mass            513 non-null    float64
 4   distance        808 non-null    float64
 5   year            1035 non-null   int64
dtypes: float64(3), int64(2), object(1)
memory usage: 48.6+ KB
```

object'i gördüğümüz zaman bunun bir kategorik değişken olduğunu düşüneceğiz. object dışında diğer tüm değişkenler ise kesikli ve sürekli olan sayısal değişkenlerdir.

```
[10]: df.dtypes
```

```
[10]: method             object
      number              int64
      orbital_period    float64
      mass              float64
      distance          float64
      year                int64
      dtype: object
```

object tipindeki değişkeni Categorical tipine dönüştürmeliyiz.

```
[11]: import pandas as pd
      df.method = pd.Categorical(df.method)
```

```
[13]: df.dtypes
```

```
[13]: method            category
      number               int64
      orbital_period     float64
      mass               float64
      distance           float64
      year                 int64
      dtype: object
```

Veri Setinin Betimlenmesi

```
[1]: import seaborn as sns
     planets = sns.load_dataset("planets")
     df = planets.copy()
```

```
[4]: df.shape #degisken ve gözlem sayısı
```

```
[4]: (1035, 6)
```

```
[5]: df.columns
```

```
[5]: Index(['method', 'number', 'orbital_period', 'mass', 'distance', 'year'], dtype='object')
```

```
[13]: df.describe().T
      #describe eksik gözlemleri göz ardı eder ve kategorik değişkenleri dışarıda bırakır.
```

[13]:

	count	mean	std	min	25%	50%	75%	max
number	1035.0	1.785507	1.240976	1.000000	1.00000	1.0000	2.000	7.0
orbital_period	992.0	2002.917596	26014.728304	0.090706	5.44254	39.9795	526.005	730000.0
mass	513.0	2.638161	3.818617	0.003600	0.22900	1.2600	3.040	25.0
distance	808.0	264.069282	733.116493	1.350000	32.56000	55.2500	178.500	8500.0
year	1035.0	2009.070531	3.972567	1989.000000	2007.00000	2010.0000	2012.000	2014.0

[12]:
```
df.describe(include = "all").T #kategorik değişkenleri de dahil eder ancak anlamlı sonuç çıkmaz.
```

[12]:

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
method	1035	10	Radial Velocity	553	NaN	NaN	NaN	NaN	NaN	NaN	NaN
number	1035	NaN	NaN	NaN	1.78551	1.24098	1	1	1	2	7
orbital_period	992	NaN	NaN	NaN	2002.92	26014.7	0.0907063	5.44254	39.9795	526.005	730000
mass	513	NaN	NaN	NaN	2.63816	3.81862	0.0036	0.229	1.26	3.04	25
distance	808	NaN	NaN	NaN	264.069	733.116	1.35	32.56	55.25	178.5	8500
year	1035	NaN	NaN	NaN	2009.07	3.97257	1989	2007	2010	2012	2014

Eksik Değerlerin İncelenmesi

[1]:
```
import seaborn as sns
planets = sns.load_dataset("planets")
df = planets.copy()
df
```

[1]:

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300000	7.10	77.40	2006
1	Radial Velocity	1	874.774000	2.21	56.95	2008
2	Radial Velocity	1	763.000000	2.60	19.84	2011
3	Radial Velocity	1	326.030000	19.40	110.62	2007
4	Radial Velocity	1	516.220000	10.50	119.47	2009
...	...	...	...	...	...	...
1030	Transit	1	3.941507	NaN	172.00	2006
1031	Transit	1	2.615864	NaN	148.00	2007
1032	Transit	1	3.191524	NaN	174.00	2007
1033	Transit	1	4.125083	NaN	293.00	2008
1034	Transit	1	4.187757	NaN	260.00	2008

1035 rows × 6 columns

[5]:
```
#hiç eksik gözlem(değer) var mı?
df.isnull().values.any()
```

[5]:
```
True
```

[6]:
```
#Hangi değişkende kaçar tane eksik değer var?
df.isnull().sum()
```

[6]:
```
method              0
number              0
orbital_period     43
mass              522
distance          227
year                0
dtype: int64
```

[12]:
```
#Eksik değerleri 0 ile doldurmak.
df["orbital_period"].fillna(0, inplace=True)
```

[13]:
```
#orbital_period değişkenindeki eksik değerleri doldurduk.
df.isnull().sum()
```

[13]:
```
method              0
number              0
orbital_period      0
mass              522
distance          227
year                0
dtype: int64
```

Eksik veri doldurma işlemi çok tehlikelidir. Veri setinin yapısını bozabilir.

[15]:
```
#Ortalama ile eksik değer doldurma
df["mass"].fillna(df.mass.mean(), inplace = True)
```

[18]:
```
df.isnull().sum()
```

[18]:
```
method              0
number              0
orbital_period      0
mass                0
distance          227
year                0
dtype: int64
```

[19]:
```
#Veri setindeki tüm eksik değerlerin yerine ortalamalarının atanması
df.fillna(df.mean, inplace = True)
```

[20]:
```
df.isnull().sum()
```

[20]:
```
method            0
number            0
orbital_period    0
mass              0
distance          0
year              0
dtype: int64
```

Eksik değerleri doldurarak veri setinin yapısını bozduk.
Copy metodu ile işlemleri geri alalım.

[21]:
```
df = planets.copy()
```

[22]:
```
df.isnull().sum()
```

[22]:
```
method              0
number              0
orbital_period     43
mass              522
distance          227
year                0
dtype: int64
```

Kategorik Değişken Özetleri

[24]:
```
import seaborn as sns
planets = sns.load_dataset("planets")
df = planets.copy()
df
```

[24]:

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300000	7.10	77.40	2006
1	Radial Velocity	1	874.774000	2.21	56.95	2008
2	Radial Velocity	1	763.000000	2.60	19.84	2011
3	Radial Velocity	1	326.030000	19.40	110.62	2007
4	Radial Velocity	1	516.220000	10.50	119.47	2009
...	...	...	...	...	...	...
1030	Transit	1	3.941507	NaN	172.00	2006
1031	Transit	1	2.615864	NaN	148.00	2007
1032	Transit	1	3.191524	NaN	174.00	2007
1033	Transit	1	4.125083	NaN	293.00	2008
1034	Transit	1	4.187757	NaN	260.00	2008

1035 rows × 6 columns

Sadece Kategorik Değişkenler ve Özetleri

Sadece Kategorik Değişkenler ve Özetleri

[28]:
```
#Kategorik değişkeni seçmek.
kat_df = df.select_dtypes(include = ["object"])
kat_df.head()
```

[28]:

	method
0	Radial Velocity
1	Radial Velocity
2	Radial Velocity
3	Radial Velocity
4	Radial Velocity

Kategorik Değişkenlerin Sınıflarına ve Sınıf Sayısına Erişmek

Kategorik Değişkenlerin Sınıflarına ve Sınıf Sayısına Erişmek

[29]:
```
#Değişkenin içerisindeki sınıf bilgileri
kat_df.method.unique()
```

[29]:
```
array(['Radial Velocity', 'Imaging', 'Eclipse Timing Variations',
       'Transit', 'Astrometry', 'Transit Timing Variations',
       'Orbital Brightness Modulation', 'Microlensing', 'Pulsar Timing',
       'Pulsation Timing Variations'], dtype=object)
```

[31]:
```
#Değişkenimizin kaç adet sınıfı olduğu
kat_df["method"].value_counts().count()
```

[31]:
```
10
```

Kategorik Değişkenin Sınıflarının Frekanslarına Erişmek

Kategorik Değişkenin Sınıflarının Frekanslarına Erişmek

[32]:
```
kat_df["method"].value_counts()
```

[32]:
```
Radial Velocity                  553
Transit                          397
Imaging                           38
Microlensing                      23
Eclipse Timing Variations          9
Pulsar Timing                      5
Transit Timing Variations          4
Orbital Brightness Modulation      3
Astrometry                         2
Pulsation Timing Variations        1
Name: method, dtype: int64
```

[37]:
```
#Sınıfların frekanslarını sütun grafiği şeklinde görelim
df["method"].value_counts().plot.barh(); # ";" bilgi satırını kapatır.
```

Sürekli Değişken Özetleri

[2]:
```
import seaborn as sns
planets = sns.load_dataset("planets")
df = planets.copy()
df.dtypes
```

[2]:
```
method             object
number              int64
orbital_period    float64
mass              float64
distance          float64
year                int64
dtype: object
```

[3]:
```
df_num = df.select_dtypes(include = ["float64", "int64"])
```

[4]:
```
df_num.head()
```

[4]:

	number	orbital_period	mass	distance	year
0	1	269.300	7.10	77.40	2006
1	1	874.774	2.21	56.95	2008
2	1	763.000	2.60	19.84	2011
3	1	326.030	19.40	110.62	2007
4	1	516.220	10.50	119.47	2009

[5]:
```
df_num.describe().T
```

[5]:

	count	mean	std	min	25%	50%	75%	max
number	1035.0	1.785507	1.240976	1.000000	1.00000	1.0000	2.000	7.0
orbital_period	992.0	2002.917596	26014.728304	0.090706	5.44254	39.9795	526.005	730000.0
mass	513.0	2.638161	3.818617	0.003600	0.22900	1.2600	3.040	25.0
distance	808.0	264.069282	733.116493	1.350000	32.56000	55.2500	178.500	8500.0
year	1035.0	2009.070531	3.972567	1989.000000	2007.00000	2010.0000	2012.000	2014.0

[9]:
```
#Sadece belirli bir değişkenin betimsel istatistiği
df_num["distance"].describe()
```

[9]:
```
count     808.000000
mean      264.069282
std       733.116493
min         1.350000
25%        32.560000
50%        55.250000
75%       178.500000
max      8500.000000
Name: distance, dtype: float64
```

```
[12]: print("Ortalama: "+ str(df_num["distance"].mean()))
print("Dolu Gözlem Sayısı: "+ str(df_num["distance"].count()))
print("Maks. Değer: "+ str(df_num["distance"].max()))
print("Min. Değer: "+ str(df_num["distance"].min()))
print("Medyan: "+ str(df_num["distance"].median()))
print("Standart Sapma: "+ str(df_num["distance"].std()))
```

```
Ortalama: 264.06928217821786
Dolu Gözlem Sayısı: 808
Maks. Değer: 8500.0
Min. Değer: 1.35
Medyan: 55.25
Standart Sapma: 733.1164929404421
```

Dağılım Grafikleri

Barplot (Sütun Grafiği)

Sütun grafikler, elimizdeki categoric değişkenleri görselleştirmek için kullanılır.

Veri Setinin Hikayesi

- price: dolar cinsinden fiyat (326-18,823)
- carat: ağırlık (0.2-5.01)
- cut: kalite (Fair, Good, Very Good, Premium, Ideal)
- color: renk (from J(worst) to D(best))
- clarity: temizliği, berraklığı (I1(worst), SI2, VS2, VS1, VVS2, VVS1, IF(best))
- x: length in mm (0-10.74)
- y: width in mm (0-58.9)
- z: depth in mm (0-31.8)
- depth: toplam derinlik yüzdesi = z / mean(x, y) = 2 * z / (x+y) (43-79)
- table: elmasın en geniş noktasına göre genişliği (43-79)

[2]:
```python
import seaborn as sns
diamonds = sns.load_dataset("diamonds")
df = diamonds.copy()
df.head()
```

[2]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

Veri Setine Hızlı Bakış

Veri Setine Hızlı Bakış

[3]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 53940 entries, 0 to 53939
Data columns (total 10 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   carat    53940 non-null  float64
 1   cut      53940 non-null  object
 2   color    53940 non-null  object
 3   clarity  53940 non-null  object
 4   depth    53940 non-null  float64
 5   table    53940 non-null  float64
 6   price    53940 non-null  int64
 7   x        53940 non-null  float64
 8   y        53940 non-null  float64
 9   z        53940 non-null  float64
dtypes: float64(6), int64(1), object(3)
memory usage: 4.1+ MB
```

[5]:
```
df.describe().T
```

[5]:

	count	mean	std	min	25%	50%	75%	max
carat	53940.0	0.797940	0.474011	0.2	0.40	0.70	1.04	5.01
depth	53940.0	61.749405	1.432621	43.0	61.00	61.80	62.50	79.00
table	53940.0	57.457184	2.234491	43.0	56.00	57.00	59.00	95.00
price	53940.0	3932.799722	3989.439738	326.0	950.00	2401.00	5324.25	18823.00
x	53940.0	5.731157	1.121761	0.0	4.71	5.70	6.54	10.74
y	53940.0	5.734526	1.142135	0.0	4.72	5.71	6.54	58.90
z	53940.0	3.538734	0.705699	0.0	2.91	3.53	4.04	31.80

[9]:
```
df.head()
```

[9]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

```
[13]: df["cut"].value_counts() #degiskendeki gozlemlerin frekansı
```

```
[13]: Ideal        21551
      Premium      13791
      Very Good    12082
      Good          4906
      Fair          1610
      Name: cut, dtype: int64
```

```
[14]: df["color"].value_counts()
```

```
[14]: G    11292
      E     9797
      F     9542
      H     8304
      D     6775
      I     5422
      J     2808
      Name: color, dtype: int64
```

Kategorik değişken görselleştirmek üzere ele aldığımız sütun grafiği işlemlerimize devam edeceğiz. Fakat şöyle bir problemimiz var; elimizdeki veri setinin içerisindeki kategorik değişkenlerin nominal değil ordinal olduğunu gözlemliyoruz.
Sınıflar arasında kötüden iyiye gibi bir sıralama var.
Bizim bunu Python programlama diline ifade etmemiz lazım.
Buradaki kategorik değişkenlerin type'ni ordered(sıralı) bir şekilde programa tanıtmalıyız.

```
[15]: #Ordinal tanımlama
      from pandas.api.types import CategoricalDtype
```

```
[16]: df.cut.head()
```

```
[16]: 0      Ideal
      1    Premium
      2       Good
      3    Premium
      4       Good
      Name: cut, dtype: object
```

```
[18]: df.cut = df.cut.astype(CategoricalDtype(ordered = True))
      #cut değişkeninin tipini kategorik değişkene dönüştür.
      #Ve bunu sıralı(ordinal) şekilde yap.
```

```
[19]: df.dtypes
```

```
[19]: carat       float64
      cut        category
      color        object
      clarity      object
      depth       float64
      table       float64
      price         int64
      x           float64
      y           float64
      z           float64
      dtype: object
```

```
[20]: df.cut.head(1)
```

```
[20]: 0    Ideal
      Name: cut, dtype: category
      Categories (5, object): [Fair < Good < Ideal < Premium < Very Good]
```

cut değişkeninin ordinal olduğunu tanıttık fakat sıralamayı yanlış yaptı. Sıralama bilgisini de vermemiz gerekiyor.

```
[21]: cut_kategoriler = ["Fair", "Good", "Very Good", "Premium", "Ideal"]
```

```
[22]: df.cut = df.cut.astype(CategoricalDtype(categories = cut_kategoriler, ordered = True))
```

```
[24]: df.cut.head(1)
      #Doğru sıralamaya ulaştık.
```

```
[24]: 0    Ideal
      Name: cut, dtype: category
      Categories (5, object): [Fair < Good < Very Good < Premium < Ideal]
```

Sütun grafiği oluşturmak üzere bölüme başladık, fakat tıpkı gerçek hayatta olduğu gibi elimizdeki veri (hazır bir kütüphaneden çektiğimiz halde) doğru bir formda değil.

Kullanacak olduğumuz fonksiyonlara göndermek üzere hazır değil.

Dolayısıyla bütün görselleştirme teknikleri işin en kolay kısmı.

Zor olan kısmı ise bu detaylardaki teknik bazı zorlukların farkında olmak ve bunları giderecek yöntemleri bilmek.

```
[25]: df["color"].value_counts()
```

```
[25]: G    11292
      E     9797
      F     9542
      H     8304
      D     6775
      I     5422
      J     2808
      Name: color, dtype: int64
```

```
[28]: color_kategoriler = ["J", "I", "H", "G", "F", "E", "D"]
      df.color = df.color.astype(CategoricalDtype(categories = color_kategoriler, ordered = True))
      df.color.head(1)
      #Doğru sıralamaya ulaştık.
```

```
[28]: 0    E
      Name: color, dtype: category
      Categories (7, object): [J < I < H < G < F < E < D]
```

```
[29]: df.clarity.value_counts()
```

```
[29]: SI1     13065
      VS2     12258
      SI2      9194
      VS1      8171
      VVS2     5066
      VVS1     3655
      IF       1790
      I1        741
      Name: clarity, dtype: int64
```

```
[31]: #(I1(worst), SI2, VS2, VS1, VVS2, VVS1, IF(best))
      clarity_kategoriler = ["I1", "SI2", "VS2", "VS1", "VVS2", "VVS1", "IF"]
      df.clarity = df.clarity.astype(CategoricalDtype(categories = clarity_kategoriler, ordered=True))
      df.clarity.head(1)
      #Doğru sıralamaya ulaştık.
```

```
[31]: 0    SI2
      Name: clarity, dtype: category
      Categories (7, object): [I1 < SI2 < VS2 < VS1 < VVS2 < VVS1 < IF]
```

Veri setinin hikayesi, **veri setine ilk adımın atılması** ve **veri setinin görselleştirmeye hazır hale getirilmesi** işlemlerini gerçekleştirmiş olduk.

Bar Plot (Sütun Grafiğin) Oluşturulması

Bar Plot (Sütun Grafiğin) Oluşturulması

[9]:
```
df["cut"].value_counts()
```

[9]:
```
Ideal        21551
Premium      13791
Very Good    12082
Good          4906
Fair          1610
Name: cut, dtype: int64
```

[10]:
```
df["cut"].value_counts().plot.barh().set_title("Cut Değişkeninin Sınıf Frekansları");
```

[11]:
```
(df["cut"]
 .value_counts()
 .plot.barh()
 .set_title("Cut Değişkeninin Sınıf Frekansları"));
```

[12]:
```
import seaborn as sns
```

[15]:
```
sns.barplot(x = "cut", y = df.cut.index, data=df);
```

Sütun Grafik Çaprazlamalar

Sütun Grafik Çaprazlamalar

Bu bölümlerde ele aldığımız uygulamalar artık grafiklerin teknik özelliklerinin yanında bize daha detaylı, veriye değil de bilgiye erişmek için kullanacak olduğumuz yaklaşımlardır.

[1]:
```
import seaborn as sns
from pandas.api.types import CategoricalDtype
diamonds = sns.load_dataset("diamonds")
df = diamonds.copy()
cut_kategoriler = ["Fair","Good","Very Good", "Premium", "Ideal"]
df.cut = df.cut.astype(CategoricalDtype(categories = cut_kategoriler, ordered = True))
df.head()
```

[1]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

[10]:
```
sns.catplot(x="cut", y="price", data=df);
#catplot grafiği ketagorik değişken çaprazlamak için kullanılır.
```

[21]:
```
sns.barplot(x = "cut", y = "price", hue = "color", data=df);
#Bu grafik cut ve color'a göre gruplama yapar. Price değerlerinin ortalamasını ve std gösterir.
```

Grafikteki verileri doğrulayalım.

[19]:
```
df.groupby(["cut", "color"])["price"].mean().unstack()
```

[19]:

color	D	E	F	G	H	I	J
cut							
Fair	4291.061350	3682.312500	3827.003205	4239.254777	5135.683168	4685.445714	4975.655462
Good	3405.382175	3423.644159	3495.750275	4123.482204	4276.254986	5078.532567	4574.172638
Very Good	3470.467284	3214.652083	3778.820240	3872.753806	4535.390351	5255.879568	5103.513274
Premium	3631.292576	3538.914420	4324.890176	4500.742134	5216.706780	5946.180672	6294.591584
Ideal	2629.094566	2597.550090	3374.939362	3720.706388	3889.334831	4451.970377	4918.186384

Histogram ve Yoğunluk Grafiği

Histogram ve Yoğunluk Grafiği

Histogram ve yoğunluk grafikleri sayısal değişkenlerin dağılımını ifade etmek için kullanılan veri görselleştirme tekniklerdir.

[4]:
```
import seaborn as sns
diamonds = sns.load_dataset("diamonds")
df = diamonds.copy()
df.head()
```

[4]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	0.23	Ideal	E	SI2	61.5	55.0	326	3.95	3.98	2.43
1	0.21	Premium	E	SI1	59.8	61.0	326	3.89	3.84	2.31
2	0.23	Good	E	VS1	56.9	65.0	327	4.05	4.07	2.31
3	0.29	Premium	I	VS2	62.4	58.0	334	4.20	4.23	2.63
4	0.31	Good	J	SI2	63.3	58.0	335	4.34	4.35	2.75

[11]:
```
sns.distplot(df.price, kde=False);
#kde yoğunluk gösterir.
```

İki tepeli bir yapı oluştu. Bu çarpıklık olduğunu gösterir.

[25]:
```
df["price"].describe()
```

[25]:
```
count    53940.000000
mean      3932.799722
std       3989.439738
min        326.000000
25%        950.000000
50%       2401.000000
75%       5324.250000
max      18823.000000
Name: price, dtype: float64
```

[25]:
```
df["price"].describe()
```

[25]:
```
count    53940.000000
mean      3932.799722
std       3989.439738
min        326.000000
25%        950.000000
50%       2401.000000
75%       5324.250000
max      18823.000000
Name: price, dtype: float64
```

[16]:
```
sns.distplot(df.price, bins = 500, kde=False);
#bins: histogramdaki çubuk sayısı
```

[17]:
```
sns.distplot(df.price);
#histogram ve yoğunluk grafiği birlikte
```

[22]:
```
sns.distplot(df.price, hist = False);
#Sadece yoğunluk grafiği
```

[31]:
```
sns.kdeplot(df.price, shade = True);
#yoğunluk grafiğinin altını doldurarak oluşturduk.
```

Histogram ve Yoğunluk Çaprazlamalar

Histogram ve Yoğunluk Çaprazlamalar

[28]:
```
import seaborn as sns
from pandas.api.types import CategoricalDtype
diamonds = sns.load_dataset("diamonds")
df = diamonds.copy()
cut_kategoriler=["Fair","Good","Very Good","Premium","Ideal"]
df.cut = df.cut.astype(CategoricalDtype(categories=cut_kategoriler, ordered=True))
df.cut.head()
```

[28]:
```
0      Ideal
1    Premium
2       Good
3    Premium
4       Good
Name: cut, dtype: category
Categories (5, object): [Fair < Good < Very Good < Premium < Ideal]
```

[29]:
```
color_kategoriler = ["J", "I", "H", "G", "F", "E", "D"]
df.color = df.color.astype(CategoricalDtype(categories = color_kategoriler, ordered = True))
df.color.head(1)
```

[29]:
```
0    E
Name: color, dtype: category
Categories (7, object): [J < I < H < G < F < E < D]
```

[31]:
```
(sns
 .FacetGrid(df,
            hue="cut",
            height=5,
            xlim=(0,10000)) #x ekseninin bas.-bitis degerleri
 .map(sns.kdeplot, "price", shade=True)
 .add_legend() #kategorik bilgiler için
);
```

```
[30]: sns.catplot(x="cut", y="price", hue="color", kind="point", data=df);
```

Boxplot

Veri Seti Hikayesi

total_bill: yemeğin toplam fiyatı (bahşiş ve vergi dahil)
tip: bahşiş
sex: ücreti ödeyen kişinin cinsiyeti (0=male, 1=female)
smoker: grupta sigara içen var mı? (0=No, 1=Yes)
day: gün (3=Thur, 4=Fri, 5=Sat, 6=Sun)
time: ne zaman? (0=Day, 1=Night)
size: grupta kaç kişi var?

```
[4]: import seaborn as sns
     tips = sns.load_dataset("tips")
     df = tips.copy()
     df.head()
```

[4]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

[5]:
```
df.describe().T
```

[5]:

	count	mean	std	min	25%	50%	75%	max
total_bill	244.0	19.785943	8.902412	3.07	13.3475	17.795	24.1275	50.81
tip	244.0	2.998279	1.383638	1.00	2.0000	2.900	3.5625	10.00
size	244.0	2.569672	0.951100	1.00	2.0000	2.000	3.0000	6.00

[6]:
```
df.sex.value_counts()
```

[6]:
```
Male      157
Female     87
Name: sex, dtype: int64
```

[7]:
```
df["smoker"].value_counts()
```

[7]:
```
No     151
Yes     93
Name: smoker, dtype: int64
```

[8]:
```
df["day"].value_counts()
```

[8]:
```
Sat     87
Sun     76
Thur    62
Fri     19
Name: day, dtype: int64
```

[9]:
```
df["time"].value_counts()
```

[9]:
```
Dinner    176
Lunch      68
Name: time, dtype: int64
```

Boxplot Oluşturma

[14]:
```
sns.boxplot(x = df["total_bill"]);
```

Boxplot, bir değerin aykırı değer olarak tanımlanması için bize en fazla yardımı dokunacak araçlardan birisidir.

[16]:
```
sns.boxplot(x = df["total_bill"], orient="v");
#dikey gözlem
#sns.boxplot(y = df["total_bill"]); ile da yapabiliriz.
```

Boxplot Çaprazlamalar

Boxplot Caprazlamalar

[17]:
```
df.describe().T
```

[17]:

	count	mean	std	min	25%	50%	75%	max
total_bill	244.0	19.785943	8.902412	3.07	13.3475	17.795	24.1275	50.81
tip	244.0	2.998279	1.383638	1.00	2.0000	2.900	3.5625	10.00
size	244.0	2.569672	0.951100	1.00	2.0000	2.000	3.0000	6.00

Hangi günler daha fazla kazanıyoruz?

[48]:
```
df.groupby("day")["total_bill"].describe().T
```

[48]:

day	Thur	Fri	Sat	Sun
count	62.000000	19.000000	87.000000	76.000000
mean	17.682742	17.151579	20.441379	21.410000
std	7.886170	8.302660	9.480419	8.832122
min	7.510000	5.750000	3.070000	7.250000
25%	12.442500	12.095000	13.905000	14.987500
50%	16.200000	15.380000	18.240000	19.630000
75%	20.155000	21.750000	24.740000	25.597500
max	43.110000	40.170000	50.810000	48.170000

[22]:
```
sns.boxplot(x="day", y="total_bill", data=df);
```

Sabah mı Akşam mı daha çok kazanıyoruz?

[47]:
```
df.groupby(["time"])["total_bill"].describe().T
```

[47]:

time	Lunch	Dinner
count	68.000000	176.000000
mean	17.168676	20.797159
std	7.713882	9.142029
min	7.510000	3.070000
25%	12.235000	14.437500
50%	15.965000	18.390000
75%	19.532500	25.282500
max	43.110000	50.810000

[37]:
```
sns.boxplot(x="time", y="total_bill", data=df);
```

Kişi sayısına göre ödenen fiyat

[49]:
```
sns.boxplot(x="size", y="total_bill", data=df);
```

Cinsiyet ve günlere göre ödenen fiyat

```
[58]: sns.boxplot(x="day", y="total_bill", hue="sex", data=df);
```

Violin Grafiği

Violin Grafiği

[59]:
```
df.head()
```

[59]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

[61]:
```
sns.catplot(y = "total_bill", kind = "violin", data=df);
```

Violin Grafiği Çaprazlamalar

```
[66]: sns.catplot(x="day", y="total_bill", kind="violin", data=df);
```

```
[67]: sns.catplot(x="day", y="total_bill", hue="sex", kind="violin", data=df);
```

Korelasyon Grafiği

Korelasyon, değişkenler arasındaki ilişkiyi ifade eden istatistiksel bir terimdir.

Scatterplot (Saçılım Grafiği)

İki değişken arasındaki ilişkiyi ifade etmek için kullanılan ve en çok bilinen yaklaşım **Scatterplot** yaklaşımıdır.

Scatterplot bize sayısal değişkenler arasındaki ilişkiyi gösterir.

total_bill: yemeğin toplam fiyatı (bahşiş ve vergi dahil)

tip: bahşiş

sex: ücreti ödeyen kişinin cinsiyeti (0=male, 1=female)

smoker: grupta sigara içen var mı? (0=No, 1=Yes)

day: gün (3=Thur, 4=Fri, 5=Sat, 6=Sun)

time: ne zaman? (0=Day, 1=Night)

size: grupta kaç kişi var?

1]:
```
import seaborn as sns
tips = sns.load_dataset("tips")
df=tips.copy()
df.head()
```

1]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
[3]: sns.scatterplot(x="total_bill", y="tip", data=df);
```

Saçılım sağ tarafa gittikçe artmış, yani toplam ödenen tutar arttıkça bahşiş de artmış diyebiliriz.

Korelasyon Çaprazlamalar

Korelasyon Çaprazlamalar

```
[4]: sns.scatterplot(x="total_bill", y="tip", hue="time", data=df);
```

```
[8]: sns.scatterplot(x="total_bill", y="tip", hue="time", style="time", data=df);
```

```
[6]: sns.scatterplot(x="total_bill", y="tip", hue="day", data=df);
```

```
[10]: sns.scatterplot(x="total_bill", y="tip", hue="day", style="day", data=df);
```

```
[15]: sns.scatterplot(x="total_bill", y="tip", hue="day", style="time", data=df);
```

```
[18]: sns.scatterplot(x="total_bill", y="tip", size="size", hue="size", data=df);
```

Doğrusal İlişkinin Gösterilmesi

Doğrusal İlişkinin Gösterilmesi

```
[1]: import seaborn as sns
     import matplotlib.pyplot as plt
     #bu bölüme özel pyplot fonksiyonunu import ettik.

     tips = sns.load_dataset("tips")
     df = tips.copy()
     df.head()
```

[1]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

[2]:
```
sns.lmplot(x = "total_bill", y = "tip", data=df);
#lmplot = lineer model plot
```

[5]:
```
sns.lmplot(x="total_bill", y="tip", hue="smoker", data=df);
```

```
[6]: sns.lmplot(x="total_bill", y="tip", hue="smoker", col="time", data=df);
```

```
[7]: sns.lmplot(x="total_bill", y="tip", hue="smoker", col="time", row="sex", data=df);
```

Scatterplot Matrisi (pairplot)

Scatterplot Matrisi

[4]:
```
import seaborn as sns
iris = sns.load_dataset("iris")
df = iris.copy()
df.head()
```

[4]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

[5]:
```
df.dtypes
```

[5]:
```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

[6]:
```
df.shape
```

[6]:
```
(150, 5)
```

[9]:
```
df.describe().T
```

[9]:

	count	mean	std	min	25%	50%	75%	max
sepal_length	150.0	5.843333	0.828066	4.3	5.1	5.80	6.4	7.9
sepal_width	150.0	3.057333	0.435866	2.0	2.8	3.00	3.3	4.4
petal_length	150.0	3.758000	1.765298	1.0	1.6	4.35	5.1	6.9
petal_width	150.0	1.199333	0.762238	0.1	0.3	1.30	1.8	2.5

[15]:
```
df.groupby(["species"]).mean().T
```

[15]:

species	setosa	versicolor	virginica
sepal_length	5.006	5.936	6.588
sepal_width	3.428	2.770	2.974
petal_length	1.462	4.260	5.552
petal_width	0.246	1.326	2.026

```
[18]: sns.pairplot(df);
```

Veri setinde yer alan 4 değişkenin birbirleri arasındaki ilişkiler görselleştirilmiş olarak karşımıza geldi.

Eksende yer alan barplot'a benzer grafikler değişkenlerin dağılımlarını göstermektedir.

```
[21]: sns.pairplot(df, hue="species");
```

```
[27]: sns.pairplot(df, hue="species", markers = ["o","s","D"]);
#markers, işaret şekilleri için
```

```
[30]: sns.pairplot(df, kind="reg",hue="species");
```

Heat Map (Isı Haritası)

Heat Map (Isı Haritası)

[1]:
```
import seaborn as sns
flights = sns.load_dataset("flights")
df = flights.copy()
df.head()
```

[1]:

	year	month	passengers
0	1949	January	112
1	1949	February	118
2	1949	March	132
3	1949	April	129
4	1949	May	121

[2]:
```
df.shape
```

[2]:
```
(144, 3)
```

[3]:
```
df.passengers.describe()
```

[3]:
```
count    144.000000
mean     280.298611
std      119.966317
min      104.000000
25%      180.000000
50%      265.500000
75%      360.500000
max      622.000000
Name: passengers, dtype: float64
```

[4]:
```
sns.scatterplot(x="year", y="passengers",hue="passengers", data=df);
```

Heatmap bizden daha yapısal tarzda bir veri ister.

Pivot table şekline getirmeliyiz.

[5]:
```
#df.pivot(index=None, columns=None, values=None)
df = df.pivot("month", "year", "passengers")
```

[6]:
```
df
```

[6]:

year	1949	1950	1951	1952	1953	1954	1955	1956	1957	1958	1959	1960
month												
January	112	115	145	171	196	204	242	284	315	340	360	417
February	118	126	150	180	196	188	233	277	301	318	342	391
March	132	141	178	193	236	235	267	317	356	362	406	419
April	129	135	163	181	235	227	269	313	348	348	396	461
May	121	125	172	183	229	234	270	318	355	363	420	472
June	135	149	178	218	243	264	315	374	422	435	472	535
July	148	170	199	230	264	302	364	413	465	491	548	622
August	148	170	199	242	272	293	347	405	467	505	559	606
September	136	158	184	209	237	259	312	355	404	404	463	508
October	119	133	162	191	211	229	274	306	347	359	407	461
November	104	114	146	172	180	203	237	271	305	310	362	390
December	118	140	166	194	201	229	278	306	336	337	405	432

[8]:
```
sns.heatmap(df);
```

```
[11]: sns.heatmap(df, annot=True, fmt="d");
```

```
[18]: sns.heatmap(df, annot=True, fmt="d", linewidths = .1);
#cbar = False eklersek sağdaki bilgi çubuğu kalkar.
```

Çizgi Grafik (Lineplot)

Veri Seti Hikayesi

"**fmri**" isminde bir veri seti inceleyeceğiz.

Beyine bağlanan bir cihaz aracılığıyla toplanan sinyalleri ifade eden bir veri seti.

subject: Verilerin toplandığı kişiler
timepoint: Zaman noktaları
event: birbirinden farklı olaylar
region: sinyalin toplandığı bölge
signal: gelen sinyal

[1]:
```
import seaborn as sns
fmri = sns.load_dataset("fmri")
df = fmri.copy()
df.head()
```

[1]:

	subject	timepoint	event	region	signal
0	s13	18	stim	parietal	-0.017552
1	s5	14	stim	parietal	-0.080883
2	s12	18	stim	parietal	-0.081033
3	s11	18	stim	parietal	-0.046134
4	s10	18	stim	parietal	-0.037970

[2]:
```
df.shape
```

[2]:
```
(1064, 5)
```

Amacımız buradaki her bir timepoint'e göre signal'in durumunu gözlemlemek olsun.

[3]:
```
df.timepoint.describe()
```

[3]:
```
count    1064.000000
mean        9.000000
std         5.479801
min         0.000000
25%         4.000000
50%         9.000000
75%        14.000000
max        18.000000
Name: timepoint, dtype: float64
```

[4]:
```
df.signal.describe()
```

[4]:
```
count    1064.000000
mean        0.003540
std         0.093930
min        -0.255486
25%        -0.046070
50%        -0.013653
75%         0.024293
max         0.564985
Name: signal, dtype: float64
```

[17]:
```
df.groupby("timepoint")["signal"].count()
#her bir timepoint'deki signal sayilari
```

[17]:
```
timepoint
0     56
1     56
2     56
3     56
4     56
5     56
6     56
7     56
8     56
9     56
10    56
11    56
12    56
13    56
14    56
15    56
16    56
17    56
18    56
Name: signal, dtype: int64
```

[19]:
```
df.groupby("timepoint")["signal"].describe()
```

[19]:

timepoint	count	mean	std	min	25%	50%	75%	max
0	56.0	-0.016662	0.028326	-0.064454	-0.039169	-0.018382	0.003539	0.074399
1	56.0	-0.025002	0.030641	-0.082174	-0.046299	-0.024533	-0.005388	0.063558
2	56.0	-0.012873	0.035440	-0.110565	-0.034944	-0.013183	0.009318	0.077277
3	56.0	0.034446	0.058260	-0.089708	-0.001157	0.028430	0.061840	0.185581
4	56.0	0.098194	0.092838	-0.046347	0.030912	0.070166	0.144911	0.346775
5	56.0	0.137725	0.123353	-0.017946	0.042762	0.096535	0.211638	0.476055
6	56.0	0.127515	0.137332	-0.054405	0.022409	0.068850	0.218919	0.564985
7	56.0	0.075660	0.129704	-0.108222	-0.016252	0.032486	0.144781	0.494787
8	56.0	0.013420	0.104216	-0.181241	-0.049453	-0.012834	0.030396	0.337143
9	56.0	-0.032041	0.072728	-0.152929	-0.075693	-0.038496	0.008717	0.221716
10	56.0	-0.053685	0.053148	-0.176453	-0.078893	-0.052906	-0.015302	0.089231
11	56.0	-0.058194	0.053828	-0.238474	-0.093127	-0.045699	-0.022522	0.030528
12	56.0	-0.052526	0.056991	-0.255486	-0.090391	-0.042294	-0.016239	0.055766
13	56.0	-0.043532	0.053598	-0.224351	-0.069285	-0.031612	-0.012958	0.059510
14	56.0	-0.033660	0.045983	-0.169312	-0.055110	-0.022165	-0.006797	0.050133
15	56.0	-0.025880	0.039092	-0.134828	-0.050536	-0.018207	0.000486	0.047102
16	56.0	-0.022414	0.035035	-0.131641	-0.041122	-0.020777	-0.001380	0.057105
17	56.0	-0.021368	0.034797	-0.121574	-0.042946	-0.017070	-0.000026	0.073757
18	56.0	-0.021867	0.036322	-0.103513	-0.046781	-0.020225	-0.002821	0.090520

Lineplot Oluşturulması

```
[22]: sns.lineplot(x="timepoint", y="signal", data=df);
```

```
[25]: sns.lineplot(x="timepoint", y="signal", hue="event", data=df);
```

```
[47]: sns.lineplot(x="timepoint", y="signal", hue="event", style="event", data=df);
```

[53]:
```
sns.lineplot(x="timepoint",
             y="signal",
             hue="event",
             style="event",
             markers=True, dashes=False, data=df);
#markers= ortalamalari isaretler.
```

[54]:
```
sns.lineplot(x="timepoint",
             y="signal",
             hue="region",
             style="event",
             data=df);
```

Basit Zaman Serisi Grafiği

[2]:
```
!pip install pandas_datareader
import pandas_datareader as pr
```

•••

[19]:
```
import pandas as pd
```

Apple'in borsadaki hisse senedi değerlerini içeren veri setiyle çalışacağız. Zamana bağlı bir veri setidir.

[3]:
```
df = pr.get_data_yahoo("AAPL", start="2016-01-01", end="2019-08-25"
```

[9]:
```
df.head()
```

[9]:

Date	High	Low	Open	Close	Volume	Adj Close
2016-01-04	105.370003	102.000000	102.610001	105.349998	67649400.0	97.948441
2016-01-05	105.849998	102.410004	105.750000	102.709999	55791000.0	95.493919
2016-01-06	102.370003	99.870003	100.559998	100.699997	68457400.0	93.625145
2016-01-07	100.129997	96.430000	98.680000	96.449997	81094400.0	89.673714
2016-01-08	99.110001	96.760002	98.550003	96.959999	70798000.0	90.147873

[10]:
```
df.shape
```

[10]:
```
(917, 6)
```

[15]:
```
kapanis = df["Close"]
kapanis.head()
```

[15]:
```
Date
2016-01-04    105.349998
2016-01-05    102.709999
2016-01-06    100.699997
2016-01-07     96.449997
2016-01-08     96.959999
Name: Close, dtype: float64
```

```
[17]: kapanis.index
      #DatetimeIndex olarak gelmis.
```

```
[17]: DatetimeIndex(['2016-01-04', '2016-01-05', '2016-01-06', '2016-01-07',
                     '2016-01-08', '2016-01-11', '2016-01-12', '2016-01-13',
                     '2016-01-14', '2016-01-15',
                     ...
                     '2019-08-12', '2019-08-13', '2019-08-14', '2019-08-15',
                     '2019-08-16', '2019-08-19', '2019-08-20', '2019-08-21',
                     '2019-08-22', '2019-08-23'],
                    dtype='datetime64[ns]', name='Date', length=917, freq=None)
```

```
[20]: #DatetimeIndex olmadigi durumlarda düzeltmemiz gerekir.
      kapanis.index = pd.DatetimeIndex(kapanis.index)
```

```
[21]: kapanis.head()
```

```
[21]: Date
      2016-01-04    105.349998
      2016-01-05    102.709999
      2016-01-06    100.699997
      2016-01-07     96.449997
      2016-01-08     96.959999
      Name: Close, dtype: float64
```

```
[23]: kapanis.plot();
```

Seaborn Alıştırmalar – 1

Question 1:

"seaborn" kütüphanesini aktif hale getirmek için hangi kod ve genellikle hangi kısaltma kullanılır?

- ○ import sea.born as sns
- ○ import sns as seaborn
- ● import seaborn as sns
- ○ from seaborn import sns

Question 2:

"df"nin bir DataFrame olduğu bilindiğine göre aşağıdaki kod ile hangi bilgilere ulaşırız?

```
df.dtypes
```

- ○ Değişkenlerin (kolon) veri tiplerine
- ○ Gözlemlerin (satır) adlarına
- ○ Gözlemlerin (satır) veri tiplerine
- ● Değişkenlerin (kolon) adlarına ve veri tiplerine

Question 3:

"column" df adlı DataFrame'in bir kolonu ve veri tipi object olduğuna göre, veri tipini category yapan kod aşağıdakilerden hangisidir?

```
import pandas as pd
df.column = pd.Categori(df.column)
```

```
import numpy as np
df.column = pd.Categori(df.method)
```

```
import pandas as pd
df.column = pd.Categorical(df.method)
```

```
import pandas as pd
df.column.dtype = pd.Categorical(df.column)
```

Question 4:

Veri setini betimlerken ilk adımlardan olan df.describe() komutu yerine df.describe().T komutu kullanıldığında ne olur?

- Aynı çıktı tablo şeklinde gösterilir
- Boyca kısa ve enine geniş bir çıktı yerine, okunabilirliği daha iyi olan boyca uzun ve enine dar bir çıktı, yani Transpozu (Devriği) yazdırılır
- Boyca uzun ve enine dar bir çıktı yerine, okunabilirliği daha iyi olan boyca kısa ve enine geniş bir çıktı, yani Transpozu (Devriği) yazdırılır
- Aralarında bir fark yoktur

Question 5:

"df" isimli DataFrame üzerinde eksik gözlem incelenmesi yapılıyor. Aşağıdaki seçeneklerden hangisi -*Tüm veriseti için hiç eksik gözlem(değer) var mı?* sorusuna karşılık gelen koddur?

- df.isnull().sum()
- df.isnull().values.any()
- df.isnull().values.all()
- df.isnull().any()

Question 6:

"df" isimli DataFrame üzerinde eksik gözlem incelenmesi yapılıyor. Aşağıdaki seçeneklerden hangisi -*Hangi değişkende kaçar tane eksik gözlem var?* sorusuna karşılık gelen koddur?

- df.isnull().sum()
- df.isnull().values.any()
- df.isnull().values.all()
- df.isnull().any()

Question 7:

"df" bir DataFrame ve "orbital_period" ise bunun bir değişkeni olmak üzere:

```
df["orbital_period"].fillna(0, inplace = True)
```

kodu ile ilgili hangileri doğrudur?

I. İlgili kolondaki eksik değerlerin sayısını verir

II. İlgili kolondaki eksik değerleri sıfır ile doldurur

III.inplace = True ifadesi yapılan değişikliğin df üzerinde kalıcı olmasını sağlar

- ○ I ve II
- ● II ve III
- ○ I ve III
- ○ Yalnız I

Question 8:

"df" bir DataFrame ve "mass" ise bir değişkeni olmak üzere;

```
df["mass"].fillna(df.mass.mean(), inplace = True)
```

kodu ile ilgili hangisi yanlıştır?

- ○ Eksik gözlem doldurulur
- ○ Eksik gözlemler değişken ortalaması ile doldurulur
- ○ Yapılan etki df üzerinde kalıcıdır
- ● Tüm DataFrame içinde hiç eksik gözlem kalmaz

Question 9:

Aşağıdaki seçeneklerden hangisi DataFrame üzerinde sedece kategorik değişkenleri seçen komuttur?

- (●) kat_df = df.select_dtypes(include = ["object"])
- ( ) kat_df = df.select_dtypes(include = "object")
- ( ) kat_df = df.dtypes(include = ["object"])
- ( ) kat_df = df.dtypes(include = "object")

Question 10:

Aşağıdaki seçeneklerden hangisi DataFrame üzerinde bir kategorik değişkenin sınıflarına ve sınıf sayısına erişmek için kullanılan kodlardır?

(●)
```
kat_df.method.unique();
kat_df["method"].value_counts().count()
```

( )
```
kat_df.method.unique();
kat_df["method"].value_counts()
```

( )
```
kat_df.method.ununique();
kat_df["method"].value_counts().count()
```

( )
```
kat_df.method.ununique();
kat_df["method"].value_counts()
```

Seaborn Alıştırmalar – 2

Question 1:

Aşağıdaki seçeneklerden hangisi DataFrame üzerinde bir kategorik değişkenin sınıflarının frekansını yatay bar grafiği ile görselleştiren koddur?

- (●) df["method"].value_counts().plot.barh();
- ( ) df["method"].value_counts().plot.bar();
- ( ) df["method"].value_counts().barh();
- ( ) df["method"].value_counts().bar();

Question 2:

Aşağıdaki seçeneklerden hangisi DataFrame üzerinde sayısal değişkenleri seçen komuttur?

- (●) df_num = df.select_dtypes(include = ["float64", "int64"])
- ( ) df_num = df.select_dtypes(include = ("float64", "int64"))
- ( ) df_num = df.dtypes(include = ["float64", "int64"])
- ( ) df_num = df.dtypes(include = ("float64", "int64"))

Question 3:

Seaborn kütüphanesine ait olan "diamond" veriseti, çalışma ortamına nasıl yüklenir?

```
import sns as seaborn
diamonds = sns.load_dataset('diamonds')
```

```
import sns as seaborn
diamonds = sns.load('diamonds')
```

```
import seaborn as sns
diamonds = sns.load_dataset('diamonds')
```

import seaborn as sns
diamonds = sns.load('diamonds')

Question 4:

`df["cut"].value_counts()`

Yukarıdaki kodu en iyi açıklayan seçenek hangisidir?

- df adlı DataFrame'e ait olan cut kolonunda bulunan kategorik değişken sınıflarının frekans sayılarını verir
- df adlı DataFrame'e ait olan cut kolonunda bulunan gözlemlerin toplam sayısını verir
- df adlı DataFrame'e ait olan cut kolonunda bulunan gözlemler sayısal ifadeler ise bunların toplamını verir
- df adlı DataFrame'e ait olan cut kolonunda bulunan gözlemlerin tekrarsız olarak toplam sayısını verir

Question 5:

Kod:

```
df.cut.head()
```

Çıktı:

```
0 Ideal
1 Premium
2 Good
3 Premium
4 Good
Name: cut, dtype: object
```

Yukarıda belirtildiği üzere kategorik tipte olmayan bir kolon verilmiştir.
Hangi seçenekteki kod ile bu kolonun tipi ordinal (sıralı) kategorik yapılabilir?

- ○ df.cut = df.cut.type(CategoricalDtype(ordered = False))
- ○ df.cut = df.cut.type(CategoricalDtype(ordered = True))
- ○ df.cut = df.cut.astype(CategoricalDtype(ordered = False))
- ● df.cut = df.cut.astype(CategoricalDtype(ordered = True))

Question 6:

Kod:

```
print(df.cut.head());
df.dtypes.cut;
```

Çıktı:

```
0 Ideal
1 Premium
2 Good
3 Premium
4 Good
Name: cut, dtype: category
Categories (5, object): [Fair < Good < Ideal < Premium < Very Good]
```

Yukarıda belirtildiği üzere ordinal (sıralı) kategorik tipte bir kolon verilmiştir. Ordinal kategorik değişkenin sıralamasını

`cut_kategoriler = ['Fair', 'Good', 'Ideal', 'Premium', 'Very Good']`

parametresini kullanarak değiştiren kod hangisidir?

- ( ) `df.cut = df.cut.astype(CategoricalDtype(categories = cut_kategoriler, ordered = False))`
- (•)
```
df.cut = df.cut.astype(CategoricalDtype(categories = cut_kategoriler, ordered = True))
```

Question 7:

Yatay Bar grafiğini belirten kod aşağıdakilerden hangisidir?

- (•) df.plot.barh()
- ( ) df.barh.plot()
- ( ) df.plot.bar()
- ( ) df.bar.plot()

Question 8:

```
df["cut"].value_counts().plot.barh().set_title("Cut Değişkeninin Sınıf Frekansları");
```

Yukarıda verilen kod yerine sadece okunabilirliğini artırmak amacıyla yazılan aşağıdaki kodlardan hangisi aynı grafiği verir?

```
(df["cut"]
 .value_counts()
 .plot.barh()
 .set_title("Cut Değişkeninin Sınıf Frekansları"));
```

Question 9:

```
sns.catplot(x = "cut", y = "price", data = df);
```

Aşağıdakilerden hangileri doğrudur?

I. catplot fonksiyonu seaborn kütüphanesine aittir

II. Çizilen grafik kategorik değişkenler için kullanılır. En az bir parametre kategorik olmalıdır

III. x ve y parametrelerine, data parametresinde belirtilen DataFrame'e ait kolon adları girilmelidir.

IV. x ve y parametre değerleri karşılıklı değiştirilirse grafik, anlam olarak değişmez fakat görünüm olarak değişir

- I,II,IV
- I,III
- Yalnız III
- I,II,III,IV

Question 10:

```
sns.barplot(x = "cut", y = "price", hue = "color", data = df);
```

Yukarıdaki grafik kodundaki hue parametresi nasıl bir etki yapar?

- cut değişkenini color değişkeninin sınıflarına göre alt gruplar halinde gösterir

Seaborn Alıştırmalar – 3

Question 1:

```
?sns.distplot
```

Yukarıda gösterildiği gibi bir fonksiyon veya kod parçacığının başına ? (soru işareti) konularak çalıştırdığında çıktı ne olur?

- Fonksiyonun parametrelerini gösterir
- Fonksiyonun bazı özelliklerini verir
- Fonksiyonla ilgili örnek kod verir
- Hepsi

Question 2:

```
sns.distplot(df.price, bins = 10, kde = False);
```

Verilen grafik ve kodu ile ilgili hangisi doğrudur?

- bins argümanı histogram sütunlarının sayısını belirtir
- kde=False ile dağılım eğrisinin gösterilmemesi sağlanır
- Seaborn kütüphanesine aittir
- Hepsi

Question 3:

```
(sns
 .FacetGrid(df,
     hue = "cut",
     height = 5,
     xlim = (0, 10000))
 .map(sns.kdeplot, "price", shade= True)
 .add_legend()
);
```

Veilen kod ve çıktısı için hangisi yanlıştır?

- FacetGrid fonksiyonu taşıyıcı, taban veya üzerine diğer grafik fonksiyonlarının eklenebileceği bir yapıyı ifade eder
- Dağılım grafiği map metodu ile bağlanmıştır
- shade=True ile çizgi altları dolu olarak gösterilir
- Renkler price değişkenine göre belirlenip rasgele değildir

Question 4:

```
sns.boxplot(x = df["total_bill"]);
```

Verilen kod ve grafik ile ilgili hangileri doğrudur?

I. Aykırı gözlemler hakkında bilgi içerir

II. Mean (ortalama) değeri gözlemlenebilir

III. 1. Çeyrek (%25.) ve 3. Çeyrek(%75.) değerleri gözlemlenebilir

IV. Median (2.Çeyrek veya %50.) değeri gözlemlenebilir

- I ve III
- I,III,IV

Question 5:

Aşağıda verilen kod ve çıktısı olan grafik ile ilgili olarak hangisi yanlıştır?

```
sns.boxplot(x = "day", y = "total_bill", data = df);
```

- ○ **df**'nin day ve **total_bill** adlı iki değişkeni arasında çaprazlama yapılmıştır
- ○ Grafiğin türü kutu grafiktir
- ● Farklı renklerin oluşmasının sebebi **hue** parametresidir
- ○ Verilen günlerin minimum **total_bill** değeri kıyaslanabilir

Question 6:

```
sns.catplot(y = "total_bill", kind = "violin", data = df);
```

Yukarıda bir violin grafiği örneği verilmiştir. Kutu grafiği ile aralarındaki farklar hakkında aşağıdakilerden hangisi yanlıştır? (varsayılan parametrelerle değerlendiriniz)

- ○ İki grafikte de 1. ve 3. Çeyrek değerleri gösterilir
- ○ İki grafikte de Median değeri gösterilir
- ◉ İki grafikte de tamamen aynı bilgiler vardır

Question 7:

Hangi grafik türünün bir amacı iki değişken arası korelasyonu göstermektir?

- Violin
- Boxplot
- Kdeplot
- Scatterplot

Question 8:

```
sns.scatterplot(x = "total_bill", y = "tip",data = df);
```

Yukarıda bir scatter türü grafik ve kodu verilmiştir. Koda hue = "time" parametresi eklenirse grafikte nasıl bir değişme bekleriz? (time, df DataFrame'ine ait iki sınıflı bir kategorik değişkendir)

- Noktaların sayısı artar
- Noktaların sayısı azalır
- Mevcut noktalar aynı yerlerinde iki grup olacak şekilde farklı renkte gösterilir

Question 9:

```
sns.lmplot(x = "total_bill", y = "tip", hue = "smoker", data = df);
```

Yukarıda verilen kod ve grafiği için aşağıdaki yorumlardan hangisi yapılamaz?

- **total_bill** ve **tip** değişkenleri arasındaki korelasyonu gösterir
- **smoker** değişkenin sınıflarına göre iki farklı korelasyon bilgisi göstrerilir
- Aykırı gözlemler çıkarılmıştır
- **smoker** bir kategorik değişkendir

Question 10:

Aşağıda bir grafik ve kodu verilmiştir.

```
sns.pairplot(df);
```

Buna göre hangisi yanlıştır?

- Grafik içinde barplot ve scatterplot vardır
- Değişkenler arasındaki ilişkileri gösterir
- Barplot bir değişkenin dağılımı hakkında bilgi verir
- İncelenen dört değişken arasında kategorik değişken vardır

Python Final Sınavı

Soru 2: **Doğru**

```
import numpy as np
array1 = np.array([[1,2,3,4,5],[6,7,8,9,10]])
print(array1[-1,:]) = ?
```

- ● [6 7 8 9 10] **(Doğru)**
- ○ [1 2 3 4 5]
- ○ [6 7 8 9]

Soru 3: **Doğru**

```
import numpy as np
array = np.array([1, 2, 3, 4, 5, 6, 7, 8, 9])
array.reshape(3,3) = ?
```

- ○ array([[7, 8, 9],
  [4, 5, 6],
  [1, 2, 3]])
- ● array([[1, 2, 3],
  [4, 5, 6], **(Doğru)**
  [7, 8, 9]])

Soru 4: **Doğru**

```
import numpy as np
array1 = np.array([[1,2],[3,4]])
array2 = np.array([[-1,-2],[-3,-4]])
np.hstack((array1,array2)) = ?
```

- (●) array([[1, 2, -1, -2],
  [3, 4, -3, -4]]) **(Doğru)**
- ( ) array([[-1, -2, 1, 2],
  [-3, -4, 3, 4]])

Soru 5: **Doğru**

```
import numpy as np
a = np.array([1,2,3])
print(a.sum()) = ?
print(a.max()) = ?
print(a.min()) = ?
```

- ( ) 5
  1
  3
- (●) 6
  3 **(Doğru)**
  1

Soru 6: **Doğru**

```
import numpy as np
np.linspace(10,15,5) = ?
```

- ○ array([10.0, 11.66666667, 13.33333333, 15.])
- ● array([10. , 11.25, 12.5 , 13.75, 15.]) **(Doğru)**
- ○ array([10., 11., 12., 13., 14., 15.])

Soru 7: **Doğru**

```
import pandas as pd
dictionary = {"NAME":["ali","veli","kenan","hilal","ayse","evren"],
              "AGE":[15,16,17,33,45,66],
              "MAAS": [100,150,240,350,110,220]}
dataFrame1 = pd.DataFrame(dictionary)
dataFrame1[dataFrame1.AGE > 60] = ?
```

- ○ ali 33 350
- ○ hilal 33 350
- ● evren 66 220 **(Doğru)**

Soru 8: **Doğru**

```
import pandas as pd
import numpy as np
dictionary = {"NAME":["ali","veli","kenan","hilal","ayse","evren"],
              "AGE":[15,16,17,33,45,66],
              "MAAS": [100,150,240,350,110,220]}
dataFrame1 = pd.DataFrame(dictionary)
ortalama_maas = dataFrame1.MAAS.mean()
s = np.sum([True if ortalama_maas > each else False for each in dataFrame1.MAAS])
s = ?
```

- ○ 0
- ○ 1
- ○ 2
- ● 3 **(Doğru)**

Soru 9: **Doğru**

```
import pandas as pd
dictionary = {"NAME":["ali","veli","kenan","hilal","ayse","evren"],
              "AGE":[15,16,17,33,45,66],
              "MAAS": [100,150,240,350,110,220]}
dataFrame1 = pd.DataFrame(dictionary)
```

MAAS sütununda bulunan değerlerin standard sapması nedir?

İpucu: dataFrame1.describe() kullanarak std'ye bakmanız lazım.

- ● 94.815611 **(Doğru)**

Soru 10: **Doğru**

```
import pandas as pd
dictionary = {"NAME":["ali","veli","kenan","hilal","ayse","evren"],
              "AGE":[15,16,17,33,45,66],
              "MAAS": [100,150,240,350,110,220]}
dataFrame1 = pd.DataFrame(dictionary)
pd.concat([dataFrame1["NAME"],dataFrame1["MAAS"]],axis=0) = ?
```

```
0     ali
1    veli
2   kenan
3   hilal
4    ayse
5   evren
0     100
1     150
2     240
3     350
4     110
5     220
```

(Doğru)

Soru 11: **Doğru**

```
import pandas as pd
dictionary = {"NAME":["ali","veli","kenan","hilal","ayse","evren"],
              "AGE":[15,16,17,33,45,66],
              "MAAS": [100,150,240,350,110,220]}
dataFrame1 = pd.DataFrame(dictionary)
dataFrame1.iloc[:,2] = ?
```

- 15
  16
  17
  33
  45
  66

- 100
  150
  240
  350
  110
  220

  (Doğru)

Soru 12: **Doğru**

```
import pandas as pd
dictionary = {"NAME":["ali","veli","kenan","hilal","ayse","evren"],
              "AGE":[15,16,17,33,45,66],
              "MAAS": [100,150,240,350,110,220]}
dataFrame1 = pd.DataFrame(dictionary)
[ each*2 for each in dataFrame1.AGE] = ?
```

- ○ [50.0, 75.0, 120.0, 175.0, 55.0, 110.0]
- ● [30, 32, 34, 66, 90, 132] **(Doğru)**
- ○ [45, 48, 51, 99, 135, 198]

Soru 13: **Doğru**

Bu plot türü nedir?

- Scatter
- Line **(Doğru)**

Soru 14: **Doğru**

Bu plot türü nedir?

- Scatter **(Doğru)**

Soru 15: **Doğru**

Bu plot türü nedir?

- Scatter
- Line
- Histogram **(Doğru)**

Soru 16: **Doğru**

Bu plot türü nedir?

- Scatter
- Subplot **(Doğru)**

---Statistic for Data Science---

Giriş

- **Örnek Teorisi :** Genelde inceliyor olduğumuz veri seti bir ana kitlenin alt kümesi olan örneklemdir. Ve genelde örneklemler üzerinden çalışıyor oluruz. Python'da örneklem nasıl çekilir nasıl gibi bazı temel kavramları uygulayarak ele alacağız.

- **Betimsel İstatistikler :** Merkezi eğilim ve merkezi dağılım ölçüleri başlığında buraya kısaca değinmiştik. Fakat burada biraz daha farklılaştığı yön ile kovaryans ve korelasyon kavramlarını da ele almış olacağız.

- **Güven Aralıkları :** Elde ettiğimiz istatistikler için bu istatistiklerin güven aralıklarının nasıl hesaplanacağını öğreneceğiz.

- **Olasılık Dağılımları :** Elimizdeki rastgele değişkenlerin dağılımlarına göre olasılık nasıl hesaplanır, olasılık dağılımları nelerdir konularını öğreneceğiz.

- **Hipotez Testleri :** Veri biliminde ve istatistikte çok önemli bir yere sahip olan hipotez testlerini öğreneceğiz. Burada AB Testi adı verilen sektörde kendisine çok fazla yer bulan AB testlerini öğrenmiş olacağız.

- **Varyans Analizi :** 2'den fazla grup üzerinde ortalamaya ilişkin test yapma işlemlerini öğreneceğiz.

- **Korelasyon Analizi :** Çok değişkenli yöntemlerin girişine gelmiş olacağız.

Örnek Teorisi

Bu bölümde örneklem, örneklem dağılımı ve merkezi limit konularına değindikten sonra python üzerinde uygulamasını gerçekleştireceğiz.

Örneklem

Örneklem Dağılımı

Birden fazla örneklem çektiğimizde ve bunların dağılımıyla ilgilendiğimizde bu durumda örneklem dağılımı konusuyla ilgileniyor oluyoruz.

Merkezi Limit Teoremi

Örnek Teorisi: Uygulama

Varsayalım ki bir ilçedeki kişilerin yaşlarına ilişkin bir çıkarımda bulunmak istiyoruz.
Bu ilçedeki kişilerin yaş ortalamasını merak ediyoruz.
Ama bu ilçede 10.000 kişi yaşıyor ve her birisiyle tek tek görüşmek çok da mümkün değil.
Bu sebeple 10.000 kişinin hepsiyle görüşmek yerine bunun içerisinden 100 kişilik
bir örneklem çekip, bu 100 kişinin yaş ortalamasını inceleyip, bu ilçenin yaş ortalamasının
kaç olabileceğini tahmin etmek istiyoruz.

```
[1]: import numpy as np

[2]: populasyon = np.random.randint(0, 80, 10000)
     #0-80 yas araliginda 10.000 kisi

[3]: populasyon[:10] # ilk 10 kisinin yasi

[3]: array([49,  8, 73, 14, 55, 48, 17, 25, 67,  0])
```

Örneklem Çekimi

Öncelikle **seed** ayarı yapmamız lazım.
seed ayarı ne demek?
Yapılacak olan işlemlerin her tekrar edildiğinde aynı sonuçların getirilmesini
garanti altına alan bir işlem.
random.seed()'i eklemezsek, fonksiyonu her çalıştırdığımda farklı örneklemler çekmiş olacak.

```
[4]: np.random.seed(115) #herhangi bir sayi verebilirsiniz.

     orneklem = np.random.choice(a = populasyon, size=100)
     #populasyon icerisinden 100 tane ornek cekme islemi.

     orneklem

[4]: array([71,  0, 19, 47, 34, 27,  1, 28, 64, 48,  4, 35, 28, 68,  8, 78, 69,
            74,  3, 12, 77, 71, 78, 15, 33, 27, 52, 52, 41, 79, 28, 52, 51, 44,
            57, 53, 15, 53,  2,  9, 73, 78, 23, 27,  9, 25, 58, 12, 74, 61, 75,
             1, 34, 17,  8, 28, 47, 51, 68, 34, 69, 71, 77, 31, 68, 33, 69, 48,
            37, 58, 14, 25, 14, 31,  4, 31,  7,  3,  8, 39, 26, 39, 19, 34, 72,
            42, 50, 48,  7,  2, 41, 76, 11, 40, 65,  2, 26, 71,  2, 33])
```

Ana kitlemiz olan populasyon'da 10.000 gözlem vardı.
Rastgele 100 gözlem çekerek örneklem oluşturduk.

```
[12]: orneklem.mean()
```

```
[12]: 39.74
```

```
[13]: populasyon.mean()
```

```
[13]: 39.6059
```

Örneklemin gücü burada çok açık bir şekilde dikkatimizi çekiyor.

Örneklem Dağılımı

```
[14]: np.random.seed(10)
      orneklem1 = np.random.choice(a = populasyon, size = 100)
      orneklem2 = np.random.choice(a = populasyon, size = 100)
      orneklem3 = np.random.choice(a = populasyon, size = 100)
      orneklem4 = np.random.choice(a = populasyon, size = 100)
      orneklem5 = np.random.choice(a = populasyon, size = 100)
      orneklem6 = np.random.choice(a = populasyon, size = 100)
      orneklem7 = np.random.choice(a = populasyon, size = 100)
      orneklem8 = np.random.choice(a = populasyon, size = 100)
      orneklem9 = np.random.choice(a = populasyon, size = 100)
      orneklem10 = np.random.choice(a = populasyon, size = 100)
```

Birbirinden farklı 10 tane örneklem çekmiş olduk.

Örneklemlerin ortalamalarının, ortalamasını alıyoruz.

```
[16]: (orneklem1.mean() + orneklem2.mean() + orneklem3.mean() + orneklem4.mean() + orneklem5.mean()
      + orneklem6.mean() + orneklem7.mean() + orneklem8.mean() + orneklem9.mean() + orneklem10.mean() )  / 10
```

```
[16]: 38.739
```

Normal şartlarda, daha fazla örneklem çekildiğinde, örneklemlerin ortalamasının ana kitle ortalamasına daha yakın olmasını bekleriz.

Merkezi limit teoremi aracılığı ile ana kitle ortalamasına gitmiş oluyoruz.

Betimsel İstatistikler

- Ortalama
- Medyan
- Mod
- Kartiller
- Değişim Aralığı
- Standart Sapma
- Kovaryans
- Korelasyon

Kovaryans

İki değişken arasındaki ilişkinin değişkenlik ölçüsüdür.

$$\mathrm{cov}(X,Y)=\mathrm{E}[(X-\mathrm{E}[X])(Y-\mathrm{E}[Y])]$$

İki rastgele değişkenin, kendi ortalamalarından olan sapmalarının beklenen değeridir.
Böylece iki değişkenin birlikte ortaya çıkardığı değişim incelenmiş olur.

Korelasyon

İki değişken arasındaki ilişkiyi, ilişkinin anlamlı olup olmadığını, ilişkinin şiddetini ve yönünü ifade eden istatistiksel bir tekniktir.

$$r_{xy} = \frac{\sum x_i y_i - n\bar{x}\,\bar{y}}{\sqrt{(\sum x_i^2 - n\bar{x}^2)}\sqrt{(\sum y_i^2 - n\bar{y}^2)}}$$

Betimsel İstatistikler: Uygulama

Örneklerimizde Tips datasetini kullanacağız.

```
[1]: #tips datasetini kullanacagiz.
     import seaborn as sns
     tips = sns.load_dataset("tips")
     df = tips.copy()
     df.head()
```

[1]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
[4]: df.describe().T
```

[4]:

	count	mean	std	min	25%	50%	75%	max
total_bill	244.0	19.785943	8.902412	3.07	13.3475	17.795	24.1275	50.81
tip	244.0	2.998279	1.383638	1.00	2.0000	2.900	3.5625	10.00
size	244.0	2.569672	0.951100	1.00	2.0000	2.000	3.0000	6.00

```
[5]: #yeni bir kütüphane kullanacağız.
     import researchpy as rp
```

Yukarıda yapmış olduğumuz işlemi bir de benzer şekilde *researchpy* kütüphanesi ile yapalım. *summary_cont* işlevi ile sayısal değişkenleri seçeceğiz.

```
[7]: rp.summary_cont(df[["total_bill", "tip", "size"]])
```

[7]:

	Variable	N	Mean	SD	SE	95% Conf.	Interval
0	total_bill	244.0	19.7859	8.9024	0.5699	18.6633	20.9086
1	tip	244.0	2.9983	1.3836	0.0886	2.8238	3.1728
2	size	244.0	2.5697	0.9511	0.0609	2.4497	2.6896

describe() ile benzer olsa da bizim için belki daha anlamlı olabilecek bazı değerler verdi.
N: Gözlem sayıları
Mean: Ortalama
SD: Standart Sapma
95% Conf. Interval Güven Aralıkları

Bir de bu işlemi categoric değişkenler için inceleyelim.

```
[11]: rp.summary_cat(df[["sex", "smoker", "day"]]).T
```

[11]:

	0	1	2	3	4	5	6	7
Variable	sex		smoker		day			
Outcome	Male	Female	No	Yes	Sat	Sun	Thur	Fri
Count	157	87	151	93	87	76	62	19
Percent	64.34	35.66	61.89	38.11	35.66	31.15	25.41	7.79

Veri okuryazarlığından biraz daha farklı olarak betimsel istatistikleri farklı bir kütüphane ile ele almış olduk.
Bu bölümün asıl farklılaştığı nokta olan **kovaryans** ve **korelasyon**'u da hızlıca bir ele alalım.

Kovaryans: Değişkenlerin ilişkilerine ilişkin bir değişkenlik ölçüsü.

cov() ile kovaryans hesaplaması yapabiliriz.

```
[13]: df[["tip", "total_bill"]].cov()
```

[13]:

	tip	total_bill
tip	1.914455	8.323502
total_bill	8.323502	79.252939

Korelasyon: İki değişken arasındaki ilişki hakkında bilgi verici ölçü.

corr() ile korelasyon hesaplaması yapabiliriz.

```
[14]: df[["tip", "total_bill"]].corr()
```

[14]:

	tip	total_bill
tip	1.000000	0.675734
total_bill	0.675734	1.000000

Güven Aralığı

Güven Aralığı Nedir?

Anakütle parametresinin tahmini değerini kapsayabilecek iki sayıdan oluşan bir aralık bulunmasıdır.

Ölçümün hassasiyetinin bir göstergesidir.

Ayrıca bize, yapmış olduğumuz tahminlerin ne kadar güvenilir olduğuyla ilgili bir değer sunar.

Web sitesinde geçirilen sürenin güven aralığı nedir?

Ortalama: 180 saniye
Standart sapma: 40 saniye

İstatistiksel olarak %95 güvenilirlik ile web sitemizde geçirilen ortalama süre 172-188 saniye aralığındadır. Gibi yorumlar yapmamızı sağlar.

Güven Aralığı Nasıl Hesaplanır?

Adım 1: n, ortalama ve standart sapmayı bul

n = 100, ortalama = 180, standart sapma = 40

Adım 2: Güven aralığına karar ver: 95 mi 99 mu?

Z tablo değerini hesapla (1,96 – 2,57)

Adım 3: Yukarıdaki değerleri kullanarak güven aralığını hesapla:

$$\bar{x} \pm z \frac{s}{\sqrt{n}} = 180 \pm 1{,}96 \times \frac{40}{\sqrt{100}}$$

Sonuç: 180 ± 7,84 yani 172 ile 188 arasıdır.

Web sitemizi ziyaret eden 100 kişiden 95'i 172 ile 188 saniye arasında web sitemizde kalacaktır.

İş Uygulaması: Fiyat Stratejisi Karar Destek Sistemi

- Problem:

  CEO fiyat belirleme konusunda *bilimsel bir dayanak* ve *esneklik* isteniyor

- Detaylar:
  - Satıcı, alıcı ve bir ürün var.
  - Alıcılara ürüne ne kadar ücret öderdiniz diye soruluyor
  - Optimum fiyat bilimsel ve esnek olarak bulunmak isteniyor.

Ürüne gelen fiyat teklifleri için veri toplandığını farzedelim.
Şimdilik bu verileri kendimiz oluşturacağız.

```
[24]: import numpy as np
      # 10 ve 110 TL araliginde 1000 adet teklif
      fiyatlar = np.random.randint(10, 110, 1000)

[25]: fiyatlar.mean()
      #Ortalama ödenmesi göze alınan miktar

[25]: 59.294
```

Bunun etrafına bir güven aralığı koyarak çok daha zengin bir karar mekanizması oluşturmuş olacağız.
Şimdi bunun için yeni bir kütüphane import edeceğiz.

```
[27]: import statsmodels.stats.api as sms

[28]: sms.DescrStatsW(fiyatlar).tconfint_mean()

[28]: (57.55186321697051, 61.036136783029484)
```

Müşterilerin %95'i 57-61 TL aralığında bedel ödemeyi göze almıştır.

Olasılığa Giriş ve Olasılık Dağılımları

Olasılık konusu, çalışıyor olduğumuz veri bilimi alanında belirsizlik ile ilgili yorumlar yapabilmek için en sık başvurduğumuz tekniklerden birisidir.

Olasılık Nedir?

Olayların olabilirliğinin sayısal ifadesidir.

Rassal Değişkenler ve Olasılık Dağılımları

Değerlerini bir deneyin sonucundan alan değişkenlere rassal değişken denir.

Dağılım Nedir?

Evrende gerçekleşen olaylar ya da durumların sayısal karşılıklarının ortaya çıkardığı yapıya dağılım denir.

Olasılık Dağılımı Nedir?

Bir rassal olaya ait değerler ve bu olaya ait değerlerin gerçekleşme olasılıklarının bir arada ifade edilmesine olasılık dağılımı denir.

Kesikli ve Sürekli Olasılık Dağılımları

Kesikli Olasılık Dağılımları

- Bernoulli
- Binom
- Poisson

Sürekli Olasılık Dağılımları

- Normal Dağılım
- Üniform Dağılım
- Üstel Dağılım

Bernoulli Dağılımı

Başarılı – başarısız, olumlu – olumsuz şeklindeki iki sonuçlu olaylar ile ilgilenildiğinde kullanılan kesikli olasılık dağılımıdır.

$$f(x;p) = p^x\,(1-p)^{1-x}, \qquad x \in \{0,1\}$$

$$E(X) = p \qquad Var(X) = pq = p(1-p)$$

Bernoulli Dağılımı Uygulama

Olasılık dağılımları ile ilgili işlemler için *scipy* ismi verilen kütüphaneyi kullanıyoruz.

```
[1]: from scipy.stats import bernoulli

[2]: #p -> 2 sonuçlu bir olayı ifade ediyor. (Yazı Tura gibi)
     p = 0.6 #Tura gelme olasılığı

[3]: rv = bernoulli(p)

[4]: rv.pmf(k = 1) #pmf -> probability mass function(olasılık kütle fonksiyonu)
     # k = 1 -> Tura gelme olasılığını hesaplar

[4]: 0.6

[5]: rv.pmf(k = 0) #yazı gelme olasılığını hesaplar

[5]: 0.4
```

Büyük Sayılar Yasası

Bir rassal değişkenin uzun vadeli kararlılığını tanımlayan olasılık teoremidir.

Düşünelim ki bir para atıyoruz, yazı yada tura gelme olasılığı %50'dir. Ancak 5 kez para attığımızı düşünelim 4 kez yazı 1 kez tura geldi. Tura gelme olasılığı %20 çıktı.

Biz bu deneyi sürekli atış sayısını arttırarak yapsaydık görmüş olacaktık ki, atış sayısı arttıkça oranlar %50'ye daha da yaklaşacaktı.

Şimdi biz bu durumu kendi yazdığımız küçük bir döngü aracılığıyla gözlemlemiş olacağız.

```
[15]: import numpy as np
      rng = np.random.RandomState(123) # Yapılacak işlemleri sabitlemek için random state kullandik.
      for i in np.arange(1,21): #1-20 arasinda gez
          deney_sayisi = 2**i
          yazi_turalar = rng.randint(0, 2, size=deney_sayisi)
          yazi_olasiliklari = np.mean(yazi_turalar)
          print("Atış Sayısı:", deney_sayisi, "---", "Yazı Olasılığı: %.2f"%(yazi_olasiliklari*100))

      Atış Sayısı: 2 --- Yazı Olasılığı: 50.00
      Atış Sayısı: 4 --- Yazı Olasılığı: 0.00
      Atış Sayısı: 8 --- Yazı Olasılığı: 62.50
      Atış Sayısı: 16 --- Yazı Olasılığı: 43.75
      Atış Sayısı: 32 --- Yazı Olasılığı: 46.88
      Atış Sayısı: 64 --- Yazı Olasılığı: 56.25
      Atış Sayısı: 128 --- Yazı Olasılığı: 50.78
      Atış Sayısı: 256 --- Yazı Olasılığı: 52.73
      Atış Sayısı: 512 --- Yazı Olasılığı: 52.93
      Atış Sayısı: 1024 --- Yazı Olasılığı: 50.20
      Atış Sayısı: 2048 --- Yazı Olasılığı: 48.58
      Atış Sayısı: 4096 --- Yazı Olasılığı: 49.49
      Atış Sayısı: 8192 --- Yazı Olasılığı: 49.58
      Atış Sayısı: 16384 --- Yazı Olasılığı: 49.96
      Atış Sayısı: 32768 --- Yazı Olasılığı: 50.00
      Atış Sayısı: 65536 --- Yazı Olasılığı: 49.68
      Atış Sayısı: 131072 --- Yazı Olasılığı: 49.97
      Atış Sayısı: 262144 --- Yazı Olasılığı: 50.13
      Atış Sayısı: 524288 --- Yazı Olasılığı: 50.01
      Atış Sayısı: 1048576 --- Yazı Olasılığı: 50.09
```

Binom Dağılımı

Binom dağılımı, bağımsız **n** deneme sonucu **k** başarılı olma olasılığı ile ilgilenildiğinde kullanılan dağılımdır.

$$f(x;n,p) = \binom{n}{x} p^x (1-p)^{n-x}, \qquad x = 0,1,2,\dots,n$$

$$E(X) = np \qquad Var(X) = np(1-p)$$

Bir madeni para 4 kere atılıyor. 2 kere yazı gelmesi olasılığı nedir?

$$f(x; n, p) = \binom{n}{x} p^x (1-p)^{n-x}, \qquad x = 0,1,2, \dots, n$$

$$f(2; 4, 0.50) = \binom{4}{2} 0.50^2 (1-0.50)^{4-2} = 0.375$$

İş Uygulaması: Reklam Harcaması Optimizasyonu

- Problem:

  Çeşitli mecralara reklam veriliyor, reklamların tıklanma ve geri dönüşüm oranları optimize edilmeye çalışılıyor. Buna yönelik olarak belirli bir mecrada çeşitli senaryolara göre reklama tıklama olasılıkları hesaplanmak isteniliyor.

- Detaylar:
  - Bir mecrada reklam verilecek
  - Dağılım ve reklama tıklama olasılığı biliniyor (0.01)
  - **Soru**: Reklamı 100 kişi gördüğünde 1,5,10 tıklanması olasılığı nedir?

Bu reklamı 100 kişinin gördüğünde 1 kişinin tıklaması olasılığı;

Olasıkların Hesaplanması

$$f(1; 100, 0.01) = \binom{100}{1} 0.01^2 (1-0.01)^{100-1} = 0.37$$

100 kişi gördüğünde 5 ve 10 kişi gördüğünde bu reklamlara tıklanma olasılığı;

$$f(5; 100, 0.01) = 0.00289779$$

$$f(10; 100, 0.01) = 0.00000007$$

```
[16]: from scipy.stats import binom

[18]: p = 0.01 #bildiğimiz olasılık değerimiz
      n = 100 #deneme sayısı
      rv = binom(n, p)
      print(rv.pmf(1)) #pmf=probability mass function. 1 kişinin tıklama olasılığı
      print(rv.pmf(5)) # 5 kişinin tıklama olasılığı
      print(rv.pmf(10)) # 10 kişinin tıklama olasılığı

      0.36972963764971983
      0.0028977871237616114
      7.006035693977161e-08
```

Poisson Dağılımı

Belirli bir zaman aralığında belirli bir alanda nadiren rastlanan olayların olasılıklarını hesaplamak için kullanılır.

$$f(x, \lambda) = \frac{\lambda^x e^{-\lambda}}{x!}, \qquad x = 0,1,2, \dots, n$$

$$E(X) = \lambda \quad Var(X) = \lambda$$

$X \sim Poisson(\lambda = 1)$
$P_X(x)$
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
x

$X \sim Poisson(\lambda = 5)$
$P_X(x)$
0.15
0.10
0.05
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
x

- 10 BİN kelimeden oluşan bir kitapta hatalı kelime sayısı
- 4000 öğrencili okulda not girişinde hata yapılması
- Bir iş gününde çağrı merkezine gelen taktir sayısı
- Kredi kartı işlemlerinde sahtekarlık olması
- Rötara düşen uçuş sefer sayısı

Gözlem sayısının çok yüksek ve beklenen sonucun gelme olasılığının çok düşük olduğu zamanlarda Poison Dağılımı kullanılır.
n(gözlem sayısı) : Büyük
p(gerçekleşme olasılığı) : Küçük olmalıdır.

Bir olayın nadir kabul edilmesi için **n > 50** olmalı ve **n*p < 5** olmalıdır.

- Rassal denemeler iki sonuçlu olmalı
- Aynı koşullar altında gerçekleştirilmelidir
- Rassal denemeler birbirinden bağımsız olmalıdır

Örnek

Bir üniversitede 5000 not girişinde 5 tane notun yanlış girilmesi olasılığı nedir?

Dağılımın Poisson olduğu biliniyor ve Lambda = 0.2

$$f(5; 0.2) = \frac{0.2^5\, e^{-0.2}}{5!} = 0.00000218328201$$

İş Uygulaması: İlan Girişi Hata Olasılıklarının Hesaplanması

- Problem:

  Hatalı ilan girişi olasılıkları hesaplanmak isteniyor.

- Detaylar:
  - Bir yıl süresince ölçümler yapılıyor
  - Dağılım biliniyor (Poisson) ve Lambda 0.1 (ortalama hata sayısı)
  - Hiç hata olmaması, 3 hata olması ve 5 hata olması olasılıkları nedir?

$$f(0; 0.1) = \frac{0.1^0\, e^{-0.1}}{0!} = 0.9048374180$$

$$f(3; 0.1) = \frac{0.1^3\, e^{-0.1}}{3!} = 0.0001508062$$

$$f(5; 0.1) = \frac{0.1^5\, e^{-0.1}}{5!} = 0.0000000754$$

```
[1]: from scipy.stats import poisson
```

```
[2]: lambda_ = 0.1 #ortalama hata sayısı
```

```
[4]: rv = poisson(mu = lambda_) #poisson fonksiyonunda mu parametresine lambda değerini veriyoruz.

     print(rv.pmf(k = 0)) #hiç hata olmaması olasılığı
     print(rv.pmf(k = 3)) #3 hata olması olasılığı
     print(rv.pmf(k = 5)) #5 hata olması olasılığı
```

```
0.9048374180359595
0.00015080623633932676
7.54031181696634e-08
```

Normal Dağılım

Normal dağıldığı bilinen sürekli rassal değişkenler için olasılık hesaplaması için kullanılır.

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

- μ ortalama ya da dağılımın beklenen değeri
- σ standart sapma
- σ^2 varyans

İş Uygulaması: Ürün Satış Olasılıklarının Hesaplanması

İş Uygulaması: Ürün Satış Olasılıklarının Hesaplanması

- Problem:

  Bir yatırım/toplantı öncesinde gelecek ay ile ilgili satışların belirli değerlerde gerçekleşmesi olasılıkları belirlenmek isteniyor.

- Detaylar:
  - Dağılımın normal olduğu biliniyor
  - Aylık ortalama satış sayısı 80K, standart sapması 5K
  - 90K'dan fazla satış yapma olasılığı nedir?

90K'dan fazla olması olasılığı nedir?

$P(X>90)=0.0228$

İlgili değerler fonksiyonda yerine yazıldığında 90K'dan fazla satış yapma olasılığı 0.0228 olarak bulunmuş. Birazdan bu problemi Python ile çözeceğiz.

70K'dan fazla olması olasılığı nedir?

$P(X>70)=0.9772$

73K'dan az olması olasılığı nedir?

$P(X<73) = 0.0808$

İş Uygulaması: Ürün Satış Olasılıklarının Hesaplanması

[1]:
```
from scipy.stats import norm
```

[2]:
```
#90'dan fazla olması olasılığı
1-norm.cdf(90, 80, 5)
#cdf = kümülatif yoğunluk fonksiyonu
#cdf(hedef, ortalama, std)
```

[2]:
```
0.02275013194817921
```

[3]:
```
#70'den fazla olması olasılığı
1-norm.cdf(70,80,5)
```

[3]:
```
0.9772498680518208
```

[4]:
```
#73'den az olması
norm.cdf(73,80,5)
```

[4]:
```
0.08075665923377107
```

[6]:
```
#85 ile 90 arasında olması olasılığı
norm.cdf(90,80,5)-norm.cdf(85,80,5)
```

[6]:
```
0.13590512198327787
```

Hipotez Testleri

Bir inanışı (bir savı, bir tahmini vs) test etmek için kullanılan istatistiksel bir tekniktir.

Hipotezler ve Türleri

Hipotezler H_0 ve H_1 hipotezleri olarak ikiye ayrılır. 0 hipotezi ve alternatif hipotez denir bunlara. Parametrelere belirli değerler vererek kurulan hipotezlere H_0 , null, sıfır hipotezi denir. Diğeri ise alternatif hipotezdir.

$$H_0: \mu = 50 \qquad H_0: \mu <= 50 \qquad H_0: \mu >= 50$$
$$H_1: \mu \neq 50 \qquad H_1: \mu > 50 \qquad H_1: \mu < 50$$

Bizim teorik olarak kendisine dayanacak olduğumuz, sabit kabul ettiğimiz, H_0 hipotezi olacak. Buna dayalı olarak iddiamızı test etmeye çalışacağız.

Hata Tipleri

		Hipotez Testi Sonucu Verilen Karar	
		H_0 reddedilmedi	H_0 reddedildi
Gerçek	H_0 doğru	Doğru Karar $(1-a) \rightarrow$ Güven Düzeyi	I. Tip Hata a
	H_0 yanlış	II. Tip Hata β	Doğru Karar $(1\text{-}\beta) \rightarrow$ Testin Gücü

Önceki bölümde kurmuş olduğumuz hipotezleri, test ettiğimizte, gerçekleştirdiğimizde ortaya çıkan sonuçlar ile ilgili değerlendirmelerdir hatalar.

p-value

Hipotez testlerinin sonuçlarını değerlendirmek üzere programlar tarafından p-value değeri verilir. Bu değer üzerinden kolayca yorum yapabiliriz.

$$p<0.05$$

Eğer hipotez testimizin sonucunda p-value değeri 0.05'den küçük ise ilgili H_0 hipotezini reddettiğimiz sonucuna varabiliriz.

Bazı durumlarda bunu yapamayız. Dağılım testlerinde H0 reddedilmek istenilmez. Çünkü H0 "örnek dağılımı ile teorik dağılım arasında fark yoktur" der.

Hipotez Testi Adımları

Adım 1: Hipotezlerin kurulması ve yönlerinin belirlenmesi

$$H_0: \mu = 50$$
$$H_1: \mu \neq 50$$

Adım 2: Anlamlılık düzeyinin ve tablo değerinin belirlenmesi

Adım 3: Test istatistiğinin belirlenmesi ve test istatistiğinin hesaplanması

Adım 4: Hesaplanan test istatistiği ile alfa'ya karşılık gelen tablo

değerinin karşılaştırılması.

Test İstatistiği (Zh) > Tablo Değeri (Zt) ise H0 Red

Adım 5: Yorum

Tek Örneklem T Testi

Popülasyon ortalaması ile varsayımsal bir değer arasında istatistiksel olarak anlamlı bir farklılık olup olmadığını test etmek için kullanılan parametrik bir testtir.

Örnek ortalamasına ilişkin test yapmak için kullanılır.

$$t = \frac{\bar{x} - \mu}{\frac{s}{\sqrt{n}}} \qquad z = \frac{\bar{x} - \mu}{\frac{\sigma}{\sqrt{n}}}$$

1. Anakütle standart sapması biliniyorsa z istatistiği kullanılır.
2. Anakütle standart sapması bilinmiyorsa ve n > 30 ise z istatistiği kullanılır.
3. Anakütle standart sapması bilinmiyor ve n < 30 ise t istatistiği kullanılır.

n büyüdükçe t, z'ye yaklaşır

T Testi Nedir?

Elimizde tek bir örneklemin ortalamasına ilişkin test yapma ihtiyacı olduğunda kullanılan testtir.

Örneğin; Bir ilçenin yaş ortalaması ile ilgili bir hipotez testi kurmak istediğimizde T testini kullanırız.

İş Uygulaması: Ürün Satın Alma Adım Optimizasyonu

- Problem:

  Sepete ürün ekleme işlemi sonrasında ödeme ekranında 5 adım vardır ve bu adımların birisi sorgulanmaktadır.

- Detaylar:
  - Her adımın 20'şer sn. olması hedefi var. 4. adım sorgulanıyor.
  - Bu durumu test etmek için 100 örnek alınıyor.
  - Örnek standart sapması 5 saniyedir. Örnek ortalaması ise 19 saniyedir.

Adım 1: Hipotezlerin kurulması ve yönlerinin belirlenmesi

$$H_0: \mu = 20$$
$$H_1: \mu \neq 20$$

Adım 2: Anlamlılık düzeyinin ve tablo değerinin belirlenmesi

$$\alpha = 0{,}05 \qquad \frac{\alpha}{2} = 0{,}025$$

Ztablo tablo olasılık değeri: 0,5 – 0,025 = 0,475

Ztablo kritik değer = -/+ 1,96

Adım 3: Test istatistiğinin belirlenmesi ve test istatistiğinin hesaplanması

$$z = \frac{\overline{x} - \mu}{\frac{\sigma(s)}{\sqrt{n}}} \qquad zhesap = \frac{19 - 20}{5/\sqrt{100}} = -2{,}00$$

n = 100, standart sapma = 5, örnek ortalaması 19 sn

Adım 4: Ztablo ve Zhesap karşılaştırması Zh > Zt ya da -Zh < -Zt ise H0 Red

Zhesap = -2,00 < Ztablo = -1,96 olduğu için H0 reddedilir.

Adım 5: Yorum

$$H_0\colon \mu = 20$$
$$H_1\colon \mu \neq 20$$

4. adımda geçirilen sürenin 20 saniye olduğunu iddia eden H0 hipotezi reddedilmiştir. Buna göre kullanıcılar istatistiksel olarak yüzde 95 güvenilirlik ile 4. adımda 20 saniyeden farklı zaman geçirmektedir.

İş Uygulaması: Web Sitesinde Geçirilen Sürenin Testi

- Problem:

  Web sitemizde geçirilen ortalama süre gerçekten 170 saniye mi?

- Detaylar:
  - Yazılımlardan elde edilen web sitesinde geçirilen ort. süreler var.
  - Bu veriler incelendiğinde bir yönetici ya da çalışanımız bu değerlerin böyle olmadığına yönelik düşünceler taşıyor ve bu durumu test etmek istiyorlar.

$$H_0: \mu = 170$$
$$H_1: \mu \neq 170$$

```
[1]: import numpy as np

     olcumler = np.array([17, 160, 234, 149, 145, 107, 197, 75, 201, 225, 211, 119,
                          157, 145, 127, 244, 163, 114, 145,  65, 112, 185, 202, 146,
                          203, 224, 203, 114, 188, 156, 187, 154, 177, 95, 165, 50, 110,
                          216, 138, 151, 166, 135, 155, 84, 251, 173, 131, 207, 121, 120])

[2]: olcumler[0:10]

[2]: array([ 17, 160, 234, 149, 145, 107, 197,  75, 201, 225])

[4]: import scipy.stats as stats

[5]: stats.describe(olcumler)

[5]: DescribeResult(nobs=50, minmax=(17, 251), mean=154.38, variance=2578.0363265306123,

       skewness=-0.32398897278694483, kurtosis=-0.05849823498415985)
```

Varsayımlarımız

Normallik varsayımı

Histogram

```
[10]: #histogram
      import pandas as pd
      pd.DataFrame(olcumler).plot.hist();
```

qqplot

```
[11]: ##qqplot
      import pylab
      stats.probplot(olcumler, dist="norm", plot=pylab)
      pylab.show()
```

Shapiro-Wilks Testi

H0: Örnek dağılımı ile teorik normal dağılım arasında istatistiksel olarak anlamlı bir farklılık yoktur.
H1: Örnek dağılımı ile teorik normal dağılım arasında istatistiksel olarak anlamlı bir farklılık vardır.

```
[12]: from scipy.stats import shapiro
```

```
[13]: shapiro(olcumler)
```

```
[13]: (0.9853105545043945, 0.7848747968673706)
```

Sol tarafta görmüş olduğumuz değer test istatistiğini ifade ediyor, sağ tarafta görmüş olduğumuz değer ise p-value değerini ifade ediyor.

p-value değerimiz 0.05(alpha)'dan büyük olduğundan **H0** hipotezi reddedilemez.

```
[15]: print("T Hesap İstatistiği: "+str(shapiro(olcumler)[0]))
      print("Hesaplanan p-value: "+str(shapiro(olcumler)[1]))

      T Hesap İstatistiği: 0.9853105545043945
      Hesaplanan p-value: 0.7848747968673706
```

Tek Örneklem T Testi Uygulaması

```
[16]: #populasyon ortalamamiz gercekten 170 mi?
      stats.ttest_1samp(olcumler, popmean=170)

[16]: Ttest_1sampResult(statistic=-2.1753117985877966, pvalue=0.034460415195071446)
```

H0: Web sitemizde geçirilen ortalama süre 170 saniyedir.
H1: Web sitemizde geçirilen ortalama süre 170 saniye değildir.

pvalue değerimiz de 0.03 yani 0.05'den küçük olduğundan **H0** hipotezi reddedilir.

Nonparametrik Tek Örneklem T Testi

Parametrik test: Çeşitli varsayımların sağlanabildiği durumda uygulanabilen testlerdir.

Ancak gerçek hayatta bazı durumlarda ilgilendiğimiz hipotez testlerine ilişkin gerekli olan varsayımlar sağlanamayabilir.

```
[1]: from statsmodels.stats.descriptivestats import sign_test

[3]: sign_test(olcumler, 170)

[3]: (-7.0, 0.06490864707227219)
```

Tek Örneklem Oran Testi

Oransal bir ifade test edilmek istenildiğinde kullanılır.

$$H_0: P = P_o \qquad H_0: P <= P_o \qquad H_0: P >= P_o$$
$$H_1: P \neq P_o \qquad H_1: P > P_o \qquad H_1: P < P_o$$

$$z = \frac{\hat{p} - p_0}{\sqrt{\frac{p_0\,(1 - p_0)}{n}}}$$

İş Uygulaması: Dönüşüm Oranı Testi

Örneğin; Web sitenizde bir ürün satıyorsunuz, bu ürünü diyelim ki 100 kişi gördü 1 kişi aldı. Bu durumda dönüşüm oranı 0.01'dir.

- Problem:
  Bir yazılım ile bir mecrada reklam verilmiş ve bu reklama ilişkin yazılım tarafından 0.125 dönüşüm oranı elde edildiği ifade edilmiş. Fakat bu durum kontrol edilmek isteniyor. Çünkü bu yüksek bir oran ve gelirler incelendiğinde örtüşmüyor.

- Detaylar:
  - 500 kişi dış mecrada reklamlara tıklamış, 40 tanesi sitemize gelip alışveriş yapmış.
  - Örnek üzerinden elde edilen dönüşüm oranı: 40/500 = 0,08

$$H_0: P = 0.125$$

$$H_1: P \neq 0.125$$

```
[5]: from statsmodels.stats.proportion import proportions_ztest
```

```
[6]: count = 40 #başarı sayısı
     nobs = 500 #gözlem sayısı
     value = 0.125 #test etmek istediğimiz oran
```

```
[7]: proportions_ztest(count, nobs, value)
```

```
[7]: (-3.7090151628513017, 0.0002080669689845979)
```

Burada p-value değerimiz (0.0002) < 0.05 olduğundan H0 hipotezimizi reddediyoruz.
%95 güven ile 0.125 değerinin yanlış olduğunu söyleyebiliriz.

Bağımsız İki Örneklem T Testi (AB Testi)

İki grup ortalaması arasında karşılaştırma yapılmak istenildiğinde kullanılır.

Elimizde gerçek değerlerini bilmediğimiz iki tane ana kitle parametresi var, bunlar iki tane ana kitlenin ortalamaları, bu ortalamaların birbirinden farkını inceliyoruz.

$H_0: \mu_1 = \mu_2$	$H_0: \mu_1 <= \mu_2$	$H_0: \mu_1 >= \mu_2$
$H_1: \mu_1 \neq \mu_2$	$H_1: \mu_1 > \mu_2$	$H_1: \mu_1 < \mu_2$

Test İstatistiği

Örnek sayıları aynı, varyanslar homojen ise:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{S_p\sqrt{\frac{2}{n}}}, \quad S_p = \sqrt{\frac{s^2{}_{X1} + s^2{}_{X2}}{2}}$$

Örnek sayısı farklı, varyanslar homojen ise:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{S_p\sqrt{\frac{1}{n_1} + \frac{1}{n_2}}}, \qquad S_p = \sqrt{\frac{(n_1 - 1)s^2{}_{X1} + (n_2 - 1)s^2{}_{X2}}{n_1 + n_2 - 2}}$$

Örnek sayıları farklı varyanslar homojen değil ise:

$$t = \frac{\overline{X}_1 - \overline{X}_2}{S_{\overline{\Delta}}}, \qquad S_{\overline{\Delta}} = \sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}$$

Varsayımlar

- **Normallik**
- **Varyans Homojenliği**

İş Uygulaması: ML Modelinin Başarı Testi

İş Uygulaması: ML Modelinin Başarı Testi (AB Testi)

- Problem:
  Bir ML projesine yatırım yapılmış. Ürettiği tahminler neticesinde oluşan gelir ile eski sistemin ürettiği gelirler karşılaştırılıp anlamlı farklılık olup olmadığı test edilmek isteniyor.

- Detaylar:
  - Model geliştirilmiş ve web sitesine entegre edilmiş.
  - Site kullanıcıları belirli bir kurala göre ikiye bölünmüş olsun.
  - A grubu eski B grubu yeni sistem.
  - Gelir anlamında anlamlı bir iş yapılıp yapılmadığı test edilmek isteniyor.

ML modeli anlamlı farklılık oluşturabildi mi?

$$H_0: \mu_1 = \mu_2$$

$$H_1: \mu_1 \neq \mu_2$$

İş Uygulaması: ML Modelinin Başarı Testi

H0: M1 = M2

H1: M1 != M2

Veri Tipi 1

[5]:
```
import pandas as pd
A = pd.DataFrame([30,27,21,27,29,30,20,20,27,32,35,22,24,23,25,27,23,27,23,
        25,21,18,24,26,33,26,27,28,19,25])

B = pd.DataFrame([37,39,31,31,34,38,30,36,29,28,38,28,37,37,30,32,31,31,27,
        32,33,33,33,31,32,33,26,32,33,29])

A_B = pd.concat([A, B], axis=1)
A_B.columns = ["A", "B"]

A_B.head()
```

[5]:

	A	B
0	30	37
1	27	39
2	21	31
3	27	31
4	29	34

Veri Tipi 2

[10]:
```
import numpy as np
A = pd.DataFrame([30,27,21,27,29,30,20,20,27,32,35,22,24,23,25,27,23,27,23,
        25,21,18,24,26,33,26,27,28,19,25])

B = pd.DataFrame([37,39,31,31,34,38,30,36,29,28,38,28,37,37,30,32,31,31,27,
        32,33,33,33,31,32,33,26,32,33,29])

#A ve A'nın grubu
Grup_A = np.arange(len(A))
Grup_A = pd.DataFrame(Grup_A)
Grup_A[:] = "A"
A = pd.concat([A, Grup_A], axis = 1)

#B ve B'nin grubu
Grup_B = np.arange(len(B))
Grup_B = pd.DataFrame(Grup_B)
Grup_B[:] = "B"
B = pd.concat([B, Grup_B], axis = 1)

#Tum veri
AB = pd.concat([A,B])
AB.columns = ["gelir", "Grup"]
print(AB.head())
print(AB.tail())
```

```
   gelir Grup
0     30    A
1     27    A
2     21    A
3     27    A
4     29    A
    gelir Grup
25     33    B
26     26    B
27     32    B
28     33    B
29     29    B
```

```
[14]: import seaborn as sns
      sns.boxplot(x="Grup", y="gelir", data=AB)
```

```
[14]: <matplotlib.axes._subplots.AxesSubplot at 0x161f1790888>
```

Bağımsız İki Örneklem T Testi Varsayım Kontrolü

Varsayım Kontrolü

```
[19]: A_B.head()
```

[19]:

	A	B
0	30	37
1	27	39
2	21	31
3	27	31
4	29	34

```
[21]: AB.head()
```

[21]:

	gelir	Grup
0	30	A
1	27	A
2	21	A
3	27	A
4	29	A

Normallik Varsayımı

```
[22]: from scipy.stats import shapiro

[23]: shapiro(A_B.A)

[23]: (0.9789242148399353, 0.7962799668312073)

[24]: shapiro(A_B.B)

[24]: (0.9561260342597961, 0.24584221839904785)
```

Varyans Homojenliği Varsayımı

H0: Varyanslar Homojendir

H1: Varyanslar Homojen Değildir

```
[26]: from scipy import stats
      stats.levene(A_B.A, A_B.B)

[26]: LeveneResult(statistic=1.1101802757158004, pvalue=0.2964124900636569)
```

Bağımsız İki Örneklem T Testi Uygulama

Hipotez Testi

```
[31]: stats.ttest_ind(A_B["A"], A_B["B"], equal_var = True)

[31]: Ttest_indResult(statistic=-7.028690967745927, pvalue=2.6233215605475075e-09)

[32]: test_istatistigi, pvalue = stats.ttest_ind(A_B["A"], A_B["B"], equal_var=True)
      print('Test İstatistiği = %.4f, p-değeri = %.4f' % (test_istatistigi, pvalue))

      Test İstatistiği = -7.0287, p-değeri = 0.0000
```

Nonparametrik Bağımsız İki Örneklem Testi

```
[33]: stats.mannwhitneyu(A_B["A"], A_B["B"])

[33]: MannwhitneyuResult(statistic=89.5, pvalue=4.778975189306267e-08)

[34]: test_istatistigi, pvalue = stats.mannwhitneyu(A_B["A"], A_B["B"])
      print('Test İstatistiği = %.4f, p-değeri = %.4f' % (test_istatistigi, pvalue))

      Test İstatistiği = 89.5000, p-değeri = 0.0000
```

Bağımlı İki Örneklem T Testi

Bağımlı iki grup arasında karşılaştırma yapılmak istenildiğinde kullanılır.

Aynı kitleye, aynı örneğe iki farklı uygulama yapıldığında ve bunun sonuçları incelendiğinde bağımlılık durumu söz konusudur. Bunlara bağımlı gruplar denir.

$$H_0: \mu_ö = \mu_s \qquad H_0: \mu_ö <= \mu_s \qquad H_0: \mu_ö >= \mu_s$$
$$H_1: \mu_ö \neq \mu_s \qquad H_1: \mu_ö > \mu_s \qquad H_1: \mu_ö < \mu_s$$

$$t = \frac{\overline{x}_D - \mu_0}{\frac{s_D}{\sqrt{n}}}$$

Varsayımlar

- Normallik
- Varyans Homojenliği

İş Uygulaması: Şirket İçi Eğitimin Performans Etkisi Ölçümü

- Problem:
  Belirli uğraşlar sonucunda alınan bir eğitimin katma değer sağlayıp sağlamadığı ölçülmek isteniyor.

- Detaylar:
  - Bir departman bir konuda eğitim talep ediyor
  - Gerekli/gereksiz değerlendirmeleri neticesinde eğitim alınıyor
  - Eğitimden önce ve sonra olacak şekilde gerekli ölçümler yapılıyor
  - Eğitim sonrasında eğitimin sağladığı katma değer test edilmek isteniyor

$$H_0: \mu_{ö} = \mu_s$$

$$H_1: \mu_{ö} \neq \mu_s$$

İş Uygulaması: Şirket İçi Eğitimin Performans Etkisi Ölçümü

[1]:
```
oncesi = pd.DataFrame([123,119,119,116,123,123,121,120,117,118,121,121,123,119,
            121,118,124,121,125,115,115,119,118,121,117,117,120,120,
            121,117,118,117,123,118,124,121,115,118,125,115])

sonrasi = pd.DataFrame([118,127,122,132,129,123,129,132,128,130,128,138,140,130,
             134,134,124,140,134,129,129,138,134,124,122,126,133,127,
             130,130,130,132,117,130,125,129,133,120,127,123])
```

[2]:
```
oncesi[0:5]
```

[2]:

	0
0	123
1	119
2	119
3	116
4	123

[3]:
```
sonrasi[0:5]
```

[3]:

	0
0	118
1	127
2	122
3	132
4	129

[5]:
```
np.arange(len(oncesi))
```

[5]:
```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
       17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33,
       34, 35, 36, 37, 38, 39])
```

```
[4]: #BIRINCI VERI SETI
AYRIK = pd.concat([oncesi, sonrasi], axis = 1)
AYRIK.columns = ["ONCESI","SONRASI"]
print("'AYRIK' Veri Seti: \n\n ", AYRIK.head(), "\n\n")


#IKINCI VERI SETI
#ONCESI FLAG/TAG'INI OLUSTURMA
GRUP_ONCESI = np.arange(len(oncesi))
GRUP_ONCESI = pd.DataFrame(GRUP_ONCESI)
GRUP_ONCESI[:] = "ONCESI"
#FLAG VE ONCESI DEGERLERINI BIR ARAYA GETIRME
A = pd.concat([oncesi, GRUP_ONCESI], axis = 1)
#SONRASI FLAG/TAG'INI OLUSTURMA
GRUP_SONRASI = np.arange(len(sonrasi))
GRUP_SONRASI = pd.DataFrame(GRUP_SONRASI)
GRUP_SONRASI[:] = "SONRASI"

#FLAG VE SONRASI DEGERLERINI BIR ARAYA GETIRME
B = pd.concat([sonrasi, GRUP_SONRASI], axis = 1)

#TUM VERIYI BIR ARAYA GETIRME
BIRLIKTE = pd.concat([A,B])
BIRLIKTE

#ISIMLENDIRME
BIRLIKTE.columns = ["PERFORMANS","ONCESI_SONRASI"]
print("'BIRLIKTE' Veri Seti: \n\n", BIRLIKTE.head(), "\n")
```

```
'AYRIK' Veri Seti:

     ONCESI  SONRASI
0     123      118
1     119      127
2     119      122
3     116      132
4     123      129


'BIRLIKTE' Veri Seti:

    PERFORMANS ONCESI_SONRASI
0         123         ONCESI
1         119         ONCESI
2         119         ONCESI
3         116         ONCESI
4         123         ONCESI
```

```
[7]: import seaborn as sns
     sns.boxplot(x = "ONCESI_SONRASI", y = "PERFORMANS", data = BIRLIKTE);
```

Bağımlı İki Örneklem T Testi Varsayım Kontrolü

Varsayım Kontrolleri

```
[8]: from scipy.stats import shapiro
```

```
[10]: shapiro(AYRIK.ONCESI)
```

```
[10]: (0.9543654918670654, 0.10722342133522034)
```

```
[11]: shapiro(AYRIK.SONRASI)
```

```
[11]: (0.9780087471008301, 0.6159457564353943)
```

```
[13]: import scipy.stats as stats
      stats.levene(AYRIK.ONCESI, AYRIK.SONRASI)
```

```
[13]: LeveneResult(statistic=8.31303288672351, pvalue=0.0050844511807370246)
```

Bağımlı İki Örneklem T Testi Uygulama

```
[14]: stats.ttest_rel(AYRIK.ONCESI, AYRIK.SONRASI)
```

```
[14]: Ttest_relResult(statistic=-9.281533480429937, pvalue=2.0235251764440722e-11)
```

```
[16]: test_istatistigi, pvalue = stats.ttest_rel(AYRIK["ONCESI"], AYRIK["SONRASI"])
      print('Test İstatistiği = %.5f, p-değeri = %.5f' % (test_istatistigi, pvalue))
```

```
Test İstatistiği = -9.28153, p-değeri = 0.00000
```

Nonparametrik Bağımlı İki Örneklem Testi

Nonparametrik Bağımlı İki Örneklem Testi

```
[17]: stats.wilcoxon(AYRIK.ONCESI, AYRIK.SONRASI)

[17]: WilcoxonResult(statistic=15.0, pvalue=2.491492033374464e-07)

[18]: test_istatistigi, pvalue = stats.wilcoxon(AYRIK["ONCESI"], AYRIK["SONRASI"])

      print('Test İstatistiği = %.4f, p-değeri = %.4f' % (test_istatistigi, pvalue))

      Test İstatistiği = 15.0000, p-değeri = 0.0000
```

İki Örneklem Oran Testi

İki oran arasında karşılaştırma yapmak için kullanılır.

$H_0: P_1 = P_2$
$H_1: P_1 \neq P_2$

$H_0: P_1 <= P_2$
$H_1: P_1 > P_2$

$H_0: P_1 >= P_2$
$H_1: P_1 < P_2$

$$Z_h = \frac{(p_1 - p_2)}{\sqrt{p(1-p)(\frac{1}{n_1} + \frac{1}{n_2})}}$$

İş Uygulaması: Kullanıcı Arayüz Deneyi (AB Testi)

Kırmızı Buton mu Yeşil Buton mu?

$$H_0 : P_1 \leq P_2$$
$$H_1 : P_1 > P_2$$

Hemen Al	Hemen Al
▪ 1000 görüntülenme	▪ 1100 görüntülenme
▪ 300 tıklama	▪ 250 tıklama

Hangi butonun seçileceğine bir hipotez testi yaparak karar vermiş olacağız.

İş Uygulaması: Kullanıcı Arayüz Deneyi (AB Testi)

```
[19]: from statsmodels.stats.proportion import proportions_ztest
```

```
[20]: import numpy as np
      basari_sayisi = np.array([300, 250])
      gozlem_sayilari = np.array([1000, 1100])
```

```
[22]: proportions_ztest(count = basari_sayisi, nobs = gozlem_sayilari)
```

```
[22]: (3.7857863233209255, 0.0001532232957772221)
```

p-value değerimiz 0.05'den küçük olduğundan dolayı H0 hipotezimiz reddedilir.

Varyans Analizi

İki ya da daha fazla grup ortalaması arasında istatistiksel olarak anlamlı farklılık olup olmadığı öğrenilmek istenildiğinde kullanılır.

$$H_0: \mu_1 = \mu_2 = \mu_3$$

H_1: Eşit değillerdir (en az birisi farklıdır)

$$F_s = \frac{MS_{(between)}}{MS_{(within)}}$$

Varsayımlar

- Gözlemlerin birbirinden bağımsız olması (grupların)
- Normal dağılım
- Varyans homojenliği

İş Uygulaması: Anasayfa İçerik Stratejisi Belirleme

- Problem:
  Anasayfa'da geçirilen süre artırılmak isteniyor

- Detaylar:
  - Bir web sitesi için başarı kriterleri: ortalama ziyaret süresi, hemen çıkış oranı vb
  - Uzun zaman geçiren kullanıcıların reklamlara daha fazla tıkladığı ve markaya olan bağlılıklarının arttığı biliniyor.
  - Buna yönelik olarak benzer haberler farklı resimler ya da farklı formatlarda hazırlanarak oluşturulan test gruplarına gösteriliyor.
  - A: Doğal Şekilde, B: Yönlendirici, C: İlgi Çekici

İş Uygulaması: Anasayfa İçerik Stratejisi Belirleme

```
[24]: A = pd.DataFrame([28,33,30,29,28,29,27,31,30,32,28,33,25,29,27,31,31,30,31,34,30,32,31,34,28,32,31,28,33,29])

B = pd.DataFrame([31,32,30,30,33,32,34,27,36,30,31,30,38,29,30,34,34,31,35,35,33,30,28,29,26,37,31,28,34,33])

C = pd.DataFrame([40,33,38,41,42,43,38,35,39,39,36,34,35,40,38,36,39,36,33,35,38,35,40,40,39,38,38,43,40,42])

dfs = [A, B, C]

ABC = pd.concat(dfs, axis = 1)
ABC.columns = ["GRUP_A","GRUP_B","GRUP_C"]
ABC.head()
```

[24]:

	GRUP_A	GRUP_B	GRUP_C
0	28	31	40
1	33	32	33
2	30	30	38
3	29	30	41
4	28	33	42

Varsayım Kontrolü

Varsayım Kontrolü

```
[25]: from scipy.stats import shapiro

[26]: shapiro(ABC["GRUP_A"])

[26]: (0.9697431921958923, 0.5321715474128723)

[27]: shapiro(ABC["GRUP_B"])

[27]: (0.9789854884147644, 0.7979801297187805)

[28]: shapiro(ABC["GRUP_C"])

[28]: (0.9579201340675354, 0.273820161819458)

[29]: stats.levene(ABC["GRUP_A"], ABC["GRUP_B"],ABC["GRUP_C"])

[29]: LeveneResult(statistic=1.0267403645055275, pvalue=0.36247110117417064)
```

Varyanslar homojendir.

Hipotez Testinin Uygulanması

```
[32]: from scipy.stats import f_oneway
```

```
[33]: f_oneway(ABC["GRUP_A"], ABC["GRUP_B"],ABC["GRUP_C"])
```

```
[33]: F_onewayResult(statistic=74.69278140730431, pvalue=1.3079050746811477e-19)
```

```
[34]: print('{:.5f}'.format(f_oneway(ABC["GRUP_A"], ABC["GRUP_B"],ABC["GRUP_C"])[1]))
      0.00000
```

```
[36]: ABC.describe().T
```

[36]:

	count	mean	std	min	25%	50%	75%	max
GRUP_A	30.0	30.133333	2.224214	25.0	28.25	30.0	31.75	34.0
GRUP_B	30.0	31.700000	2.937862	26.0	30.00	31.0	34.00	38.0
GRUP_C	30.0	38.100000	2.808239	33.0	36.00	38.0	40.00	43.0

p-value değerimiz 0.05’den küçük olduğundan H0 hipotezimizi reddederiz.

Nonparametrik Hipotez Testi

Nonparametrik Hipotez Testi

```
[39]: from scipy.stats import kruskal
```

```
[40]: kruskal(ABC["GRUP_A"], ABC["GRUP_B"],ABC["GRUP_C"])
```

```
[40]: KruskalResult(statistic=54.19819735523783, pvalue=1.7022015426175926e-12)
```

p-value değerimiz 0.05’den küçük olduğundan H0 hipotezimizi reddederiz. Gruplar arasında istatistiki olarak anlamlı bir farklılık vardır.

Korelasyon Analizi

Değişkenler arasındaki ilişki, bu ilişkinin yönü ve şiddeti ile ilgili bilgiler sağlayan istatistiksel bir yöntemdir.

Korelasyonun Anlamlılığının Testi

$$H_0: \rho = 0$$
$$H_1: \rho \neq 0$$

H0: Değişkenler arasında anlamlı bir ilişki yoktur.

H1: Değişkenler arasında anlamlı bir ilişki vardır.

$$r_{xy} = \frac{\sum x_i y_i - n\bar{x}\,\bar{y}}{\sqrt{(\sum x_i^2 - n\bar{x}^2)}\sqrt{(\sum y_i^2 - n\bar{y}^2)}}$$

$$t = r\sqrt{\frac{n-2}{1-r^2}}$$

Varsayımlar

- İki değişken içinde normallik varsayımı.
- Varsayım sağlanıyorsa Pearson Korelasyon Katsayısı
- Varsayım sağlanmıyorsa Spearman Korelasyon Katsayısı

İş Uygulaması: Bahşiş ile Ödenen Hesap Arasındaki İlişkinin İncelenmesi

Bahşiş ile ödenen hesap arasında korelasyon var mı?

$$H_0: \rho = 0$$
$$H_1: \rho \neq 0$$

İş Uygulaması: Bahşiş ile Ödenen Hesap Arasındaki İlişkinin İncelenmesi

Bahşiş veri seti:

total_bill: yemeğin toplam fiyatı (bahşiş ve vergi dahil)

tip: bahşiş

sex: ücreti ödeyen kişinin cinsiyeti (0=male, 1=female)

smoker: grupta sigara içen var mı? (0=No, 1=Yes)

day: gün (3=Thur, 4=Fri, 5=Sat, 6=Sun)

time: ne zaman? (0=Day, 1=Night)

size: grupta kaç kişi var?

[1]:
```
import seaborn as sns
tips = sns.load_dataset('tips')
df = tips.copy()
df.head()
```

[1]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
[2]: df["total_bill"] = df["total_bill"] - df["tip"]
```

```
[3]: df.head()
```

[3]:

	total_bill	tip	sex	smoker	day	time	size
0	15.98	1.01	Female	No	Sun	Dinner	2
1	8.68	1.66	Male	No	Sun	Dinner	3
2	17.51	3.50	Male	No	Sun	Dinner	3
3	20.37	3.31	Male	No	Sun	Dinner	2
4	20.98	3.61	Female	No	Sun	Dinner	4

```
[5]: df.plot.scatter("tip","total_bill");
```

Korelasyon Varsayım Kontrolü

```
[6]: from scipy.stats import shapiro
```

```
[7]: test_istatistigi, pvalue = shapiro(df["tip"])
print('Test İstatistiği = %.4f, p-değeri = %.4f' % (test_istatistigi, pvalue))

test_istatistigi, pvalue = shapiro(df["total_bill"])
print('Test İstatistiği = %.4f, p-değeri = %.4f' % (test_istatistigi, pvalue))
```

```
Test İstatistiği = 0.8978, p-değeri = 0.0000
Test İstatistiği = 0.9136, p-değeri = 0.0000
```

Korelasyon Katsayısı Hipotez Testi

Korelasyon Katsayısı

[8]:
```
df["tip"].corr(df["total_bill"])
```

[8]:
```
0.5766634471096374
```

[9]:
```
df["tip"].corr(df["total_bill"], method = "spearman")
```

[9]:
```
0.593691939408997
```

Korelasyonunu Anlamlılığının Testi

[10]:
```
from scipy.stats.stats import pearsonr
```

[16]:
```
test_istatistigi, pvalue = pearsonr(df["tip"],df["total_bill"])

print('Korelasyon Katsayısı = %.4f, p-değeri = %.4f' % (test_istatistigi, pvalue))
```

```
Korelasyon Katsayısı = 0.5767, p-değeri = 0.0000
```

Nonparametrik Hipotez Testi

Nonparametrik Hipotez Testi

[13]:
```
from scipy.stats import stats
stats.spearmanr(df["tip"],df["total_bill"])
```

[13]:
```
SpearmanrResult(correlation=0.593691939408997, pvalue=1.2452285137560276e-24)
```

[15]:
```
test_istatistigi, pvalue = stats.spearmanr(df["tip"],df["total_bill"])

print('Korelasyon Katsayısı = %.4f, p-değeri = %.4f' % (test_istatistigi, pvalue))
```

```
Korelasyon Katsayısı = 0.5937, p-değeri = 0.0000
```

[17]:
```
test_istatistigi, pvalue = stats.kendalltau(df["tip"],df["total_bill"])

print('Korelasyon Katsayısı = %.4f, p-değeri = %.4f' % (test_istatistigi, pvalue))
```

```
Korelasyon Katsayısı = 0.4401, p-değeri = 0.0000
```

---Data Preprocessing---

- Veri Ön İşleme Genel Bakış
- Aykırı Gözlem Analizi
- Eksik Veri Analizi
- Standartlaştırma
- Değişken Dönüşümleri

Veri Ön İşleme, Veri Manipülasyonu ile birlikte düşünülebilir fakat Veri Manipülasyonu bizim için hem Veri Ön İşlemede hem de genel programcılık yetenekleri kapsamında bir araçtır.

Veri Ön İşleme ise daha çok Machine Learning modelleri öncesinde veri setimiz üzerisinde gerçekleştirmemiz gereken bazı işlemleri ve dönüşümleri ifade etmektedir.

Veri Ön İşlemeye Genel Bakış

Makine öğrenmesi modelinin amacı genellenebilir yapılar ortaya koymaktır.

Belirli olaylar gözlendiğinde belirli tahmin sonuçları vermektdir.

- **Veri Temizleme (Data Cleaning / Cleasing)**
  - Gürültülü Veri (Noisy Data)
  - Eksik Veri Analizi (Missing Data Analysis)
  - Aykırı Gözlem Analizi (Outlier Analysis)
- **Veri Standardizasyonu (Data Standardization, Feature Scaling)**
  - 0-1 Dönüşümü (Normalization)
  - z-skoruna Dönüştürme (Standardization)
  - Logaritmik Dönüşüm (Log Transformation)
- **Veri İndirgeme (Data Reduction)**
  - Gözlem (Observation) Sayısının Azaltılması
  - Değişken (Variable) Sayısının Azaltılması
- **Değişken Dönüşümleri (Variable Transformation)**
  - Sürekli değişkenlerde dönüşümler
  - Kategorik değişkenlerde dönüşümler

Aykırı Değerler (Outliers)

Veride genel eğilimin oldukça dışına çıkan ya da diğer gözlemlerden oldukça farklı olan gözlemlere aykırı gözlem denir.

Aykırılığı ifade eden nümerik değere **aykırı değer** denir.

Aykırı değeri barındıran gözlem birimine **aykırı gözlem** denir.

Genellenebilirlik kaygısı ile oluşturulan **kural setlerini** ya da **fonksiyonları** yanıltır. **Yanlılığa** sebep olur.

Kime Göre Neye Göre Aykırı Gözlem?

«Veride genel eğilimin oldukça dışına çıkan gözlemler.»

Peki veri setinin genel eğiliminin dışına çıkmayı nasıl tanımlarız?

1. Sektör Bilgisi

Örneğin bir ev fiyat tahmin modelinde 1000 metrekarelik evleri modellemeye almamak.

Eğer kurulan modelin bir genelleme kaygısı varsa; zaten çok seyrek olan senoryalar ve genele uymayan yapılar çalışmanın dışında bırakılmalıdır.

2. Standart Sapma Yaklaşımı

Bir değişkenin ortalamasının üzerine aynı değişkenin standart sapması hesaplanarak eklenir. 1,2 ya da 3 standart sapma değeri ortalama üzerine eklenerek ortaya çıkan bu değer eşik değer olarak düşünülür ve bu değerden yukarıda ya da aşağıda olan değerler aykırı değer olarak tanımlanır.

Eşik Değer = Ortalama + 1 x Standart Sapma

Eşik Değer = Ortalama + 2 x Standart Sapma

Eşik Değer = Ortalama + 3 x Standart Sapma

Ortalama = 100,000
Standart Sapma = 20,000

Eşik Değer = 100,000 + 2 x 20,000
Eşik Değer = 140,000

3. Z-Skoru Yaklaşımı

Standart sapma yöntemine benzer şekilde çalışır. Değişken standart normal dağılıma uyarlanır, yani standartlaştırılır. Sonrasında -örneğin- dağılımın sağından ve solundan -+ 2,5 değerine göre bir eşik değer konulur ve bu değerin üzerinde ya da altında olan değerler aykırı değer olarak işaretlenir.

± 2,5

4. Boxplot (interquartile range - IQR) Yöntemi

En sık kullanılan yöntemlerden birisidir. Değişkenin değerleri küçükten büyüğe sıralanır. Çeyrekliklerine (yüzdekliklerine) yani Q1,Q3 değerlerine karşılık değeler üzerinden bir eşik değer hesaplanır ve bu eşik değere göre aykırı değer tanımı yapılır.

IQR = 1.5 x (Q3 -Q1)

Alt Eşik Değer = Q1 - IQR

Üst Eşik Değer = Q3 + IQR

Aykırı Değerleri Yakalamak

```
[2]: import seaborn as sns
     df = sns.load_dataset('diamonds')
     df = df.select_dtypes(include = ['float64', 'int64']) #sadece sayisal degiskenler icin
     df = df.dropna() #eksik gozlemlerin silinmesi
     df.head()
```

[2]:

	carat	depth	table	price	x	y	z
0	0.23	61.5	55.0	326	3.95	3.98	2.43
1	0.21	59.8	61.0	326	3.89	3.84	2.31
2	0.23	56.9	65.0	327	4.05	4.07	2.31
3	0.29	62.4	58.0	334	4.20	4.23	2.63
4	0.31	63.3	58.0	335	4.34	4.35	2.75

```
[3]: df_table = df["table"]
     df_table.head()
```

```
[3]: 0    55.0
     1    61.0
     2    65.0
     3    58.0
     4    58.0
     Name: table, dtype: float64
```

table isimli değişkenimizi aykırı gözlem analizi yapmak amacıyla box plot ile görselleştirelim.

```
[7]: sns.boxplot(x = df_table);
```

Bir aykırı değere, aykırı değer muamelesi yapmak için bir eşik değer belirlememiz gerekiyor.
Box plot kullanarak eşik değeri bulalım.

```
[8]: Q1 = df_table.quantile(0.25)
     Q3 = df_table.quantile(0.75)
     IQR = Q3-Q1 #interquartile
```

```
[9]: Q1
```

```
[9]: 56.0
```

```
[10]: Q3
```

```
[10]: 59.0
```

```
[11]: IQR
```

```
[11]: 3.0
```

```
[14]: alt_sinir = Q1 - 1.5*IQR
      ust_sinir = Q3 + 1.5*IQR

      print("alt sınır: ", alt_sinir,
           "\nüst sınır: ", ust_sinir)

      alt sınır:  51.5
      üst sınır:  63.5
```

Box plot grafiğimizin bize göstermiş olduğu değerler de bunlardı.

Aykırı değer sorgulaması yapalım.
Alt sınırdan ya da üst sınırdan, daha aşağıda ya da daha yukarıda olan değerlere nasıl erişebileceğimize bakalım.

```
[15]: (df_table < alt_sinir) | (df_table > ust_sinir)
```

```
[15]: 0        False
      1        False
      2         True
      3        False
      4        False
               ...
      53935    False
      53936    False
      53937    False
      53938    False
      53939    False
      Name: table, Length: 53940, dtype: bool
```

```
[16]: aykiri_tf = (df_table < alt_sinir)
      aykiri_tf.head()
```

```
[16]: 0    False
      1    False
      2    False
      3    False
      4    False
      Name: table, dtype: bool
```

```
[20]: df_table[aykiri_tf]
      #true-false vektörlerini gönderdiğimizde true olanları getirir.
```

```
[20]: 1515     51.0
      3238     50.1
      3979     51.0
      4150     51.0
      5979     49.0
      7418     50.0
      8853     51.0
      11368    43.0
      22701    49.0
      25179    50.0
      26387    51.0
      33586    51.0
      35633    44.0
      45798    51.0
      46040    51.0
      47630    51.0
      Name: table, dtype: float64
```

```
[19]: df_table[aykiri_tf].index
```

```
[19]: Int64Index([ 1515,  3238,  3979,  4150,  5979,  7418,  8853, 11368, 22701,
                  25179, 26387, 33586, 35633, 45798, 46040, 47630],
                 dtype='int64')
```

Aykırı değerlerin index'lerine de eriştik.

Aykırı Değer Problemini Çözmek

```
[80]: df_table[aykiri_tf]

[80]: 1515     51.0
      3238     50.1
      3979     51.0
      4150     51.0
      5979     49.0
      7418     50.0
      8853     51.0
      11368    43.0
      22701    49.0
      25179    50.0
      26387    51.0
      33586    51.0
      35633    44.0
      45798    51.0
      46040    51.0
      47630    51.0
      Name: table, dtype: float64
```

Silme Yaklaşımı

```
[81]: import pandas as pd

[82]: type(df_table)

[82]: pandas.core.series.Series

[83]: #pandas df'ne çevirme işlemi
      df_table = pd.DataFrame(df_table)

[84]: df_table.shape

[84]: (53940, 1)
```

~ ifadesi, koşulu sağlamayanları al demektir.

```
[85]: t_df = df_table[~((df_table < (alt_sinir)) | (df_table > (ust_sinir))).any(axis=1)]

[86]: t_df.shape

[86]: (53335, 1)
```

Demekki table sütunumuzda 605 adet aykırı gözlem varmış.

Silme işlemini bu şekilde yapmış olduk.

Ortalama Değerler ile Doldurma

```
[104]: import seaborn as sns
       df = sns.load_dataset('diamonds')
       df = df.select_dtypes(include = ['float64', 'int64'])
       df = df.dropna()
       df.head()
```

[104]:

	carat	depth	table	price	x	y	z
0	0.23	61.5	55.0	326	3.95	3.98	2.43
1	0.21	59.8	61.0	326	3.89	3.84	2.31
2	0.23	56.9	65.0	327	4.05	4.07	2.31
3	0.29	62.4	58.0	334	4.20	4.23	2.63
4	0.31	63.3	58.0	335	4.34	4.35	2.75

```
[105]: df_table = df['table']
```

```
[102]: aykiri_tf.head() #alt_sinir'a göre
```

```
[102]: 0    False
       1    False
       2    False
       3    False
       4    False
       Name: table, dtype: bool
```

```
[106]: df_table[aykiri_tf] #alt sınırdaki aykırı değerler.
```

```
[106]: 1515     51.0
       3238     50.1
       3979     51.0
       4150     51.0
       5979     49.0
       7418     50.0
       8853     51.0
       11368    43.0
       22701    49.0
       25179    50.0
       26387    51.0
       33586    51.0
       35633    44.0
       45798    51.0
       46040    51.0
       47630    51.0
       Name: table, dtype: float64
```

Yakaladığımız aykırı değerleri silmek yerine ortalamaları ile değiştirmek istiyoruz.

```
[91]: df_table.mean()
```

```
[91]: 57.45718390804603
```

```
[93]: df_table[aykiri_tf] = df_table.mean() #uyarıyı görmezden geleceğiz.
      •••

[94]: df_table[aykiri_tf]

[94]: 1515     57.457184
      3238     57.457184
      3979     57.457184
      4150     57.457184
      5979     57.457184
      7418     57.457184
      8853     57.457184
      11368    57.457184
      22701    57.457184
      25179    57.457184
      26387    57.457184
      33586    57.457184
      35633    57.457184
      45798    57.457184
      46040    57.457184
      47630    57.457184
      Name: table, dtype: float64
```

Görüldüğü üzere alt sınırdaki aykırı değerler 57.45 değeri ile değiştirildi.

Baskılama Yöntemi

Aykırı değerler yakalandıktan sonra, aykırılar altta ise alt sınıra eşitlenir, üstte ise üst sınıra eşitlenir.

```
[107]: alt_tf = (df_table<alt_sinir)
       ust_tf = (df_table>ust_sinir)
```

```
[109]: df_table[alt_tf] #altta kalan index'ler ve değerleri
```

```
[109]: 1515     51.0
       3238     50.1
       3979     51.0
       4150     51.0
       5979     49.0
       7418     50.0
       8853     51.0
       11368    43.0
       22701    49.0
       25179    50.0
       26387    51.0
       33586    51.0
       35633    44.0
       45798    51.0
       46040    51.0
       47630    51.0
       Name: table, dtype: float64
```

```
[110]: df_table[ust_tf] #üstte kalan index'ler ve değerleri
```

```
[110]: 2        65.0
       91       69.0
       145      64.0
       219      64.0
       227      67.0
                ...
       53695    65.0
       53697    65.0
       53756    64.0
       53757    64.0
       53785    65.0
       Name: table, Length: 589, dtype: float64
```

```
[111]: df_table[alt_tf] = alt_sinir
       df_table[ust_tf] = ust_sinir #uyarıyı görmezden geleceğiz.

       •••

[114]: print("alt sınır: ",alt_sinir,
             "\nüst sınır: ", ust_sinir)

       alt sınır:  51.5
       üst sınır:  63.5

[113]: df_table[alt_tf]

[113]: 1515     51.5
       3238     51.5
       3979     51.5
       4150     51.5
       5979     51.5
       7418     51.5
       8853     51.5
       11368    51.5
       22701    51.5
       25179    51.5
       26387    51.5
       33586    51.5
       35633    51.5
       45798    51.5
       46040    51.5
       47630    51.5
       Name: table, dtype: float64
```

```
[112]: df_table[ust_tf]

[112]: 2        63.5
       91       63.5
       145      63.5
       219      63.5
       227      63.5
                ...
       53695    63.5
       53697    63.5
       53756    63.5
       53757    63.5
       53785    63.5
       Name: table, Length: 589, dtype: float64
```

Çok Değişkenli Aykırı Gözlem Analizi

Değişkenler tek başına iken aykırı gözlem olmayabilir, ancak birlikte düşündüğümüzde aykırı bir gözlem olabilirler.

Örneğin evlilik sayısı ve yaş değişkenlerini tek tek düşündüğümüzde aykırı gözlem olarak gözükmeyebilirler. Ancak 17 yaşında birisinin 3 kez evlenmesi aykırı bir durumdur.

Local Outlier Factor

Gözlemleri bulundukları konumda yoğunluk tabanlı skorlayarak buna göre aykırı değer olabilecek değerleri tanımlayabilmemize imkan sağlıyor.

Bir noktanın local yoğunluğu bu noktanın komşuları ile karşılaştırılıyor. Eğer bir nokta komşularının yoğunluğundan anlamlı şekilde düşük ise bu

nokta komşularından daha seyrek bir bölgede bulunuyordur yorumu yapılabiliyor. Dolayısıyla burada bir komşuluk yapısı söz konusu. Bir değerin çevresi yoğun değilse demek ki bu değer aykırı değerdir şeklinde değerlendiriliyor.

```
[115]: import seaborn as sns
       diamonds = sns.load_dataset('diamonds')
       diamonds = diamonds.select_dtypes(include=['float64', 'int64'])
       df = diamonds.copy()
       df = df.dropna()
       df.head()
```

[115]:

	carat	depth	table	price	x	y	z
0	0.23	61.5	55.0	326	3.95	3.98	2.43
1	0.21	59.8	61.0	326	3.89	3.84	2.31
2	0.23	56.9	65.0	327	4.05	4.07	2.31
3	0.29	62.4	58.0	334	4.20	4.23	2.63
4	0.31	63.3	58.0	335	4.34	4.35	2.75

```
[116]: import numpy as np
       from sklearn.neighbors import LocalOutlierFactor

[117]: clf = LocalOutlierFactor(n_neighbors = 20, contamination = 0.1)

[118]: clf.fit_predict(df)

[118]: array([-1, -1, -1, ...,  1,  1,  1])

[119]: df_scores = clf.negative_outlier_factor_

[120]: df_scores[0:10]

[120]: array([-1.58352526, -1.59732899, -1.62278873, -1.33002541, -1.30712521,
              -1.28408436, -1.28428162, -1.26458706, -1.28422952, -1.27351342])

[123]: np.sort(df_scores)[0:20]

[123]: array([-8.60430658, -8.20889984, -5.86084355, -4.98415175, -4.81502092,
              -4.81502092, -4.61522833, -4.37081214, -4.29842288, -4.10492387,
              -4.0566648 , -4.01831733, -3.94882806, -3.82378797, -3.80135297,
              -3.75680919, -3.65947378, -3.59249261, -3.55564138, -3.47157375])
```

Şimdi bu score'lar arasından bir eşik değer belirleyeceğiz, bu score'un altında kalan değerler aykırı değer olarak tanımlanacak.

```
[125]: np.sort(df_scores)[13] #13. degere gidelim

[125]: -3.823787967755565
```

Bu değeri eşik değer kabul edeceğiz. (rastgele seçtik.)

```
[126]: esik_deger = np.sort(df_scores)[13]

[127]: aykiri_tf = df_scores > esik_deger
       aykiri_tf

[127]: array([ True,  True,  True, ...,  True,  True,  True])
```

Silme Yöntemi

```
[129]: yeni_df = df[df_scores > esik_deger]
       yeni_df
```

[129]:

	carat	depth	table	price	x	y	z
0	0.23	61.5	55.0	326	3.95	3.98	2.43
1	0.21	59.8	61.0	326	3.89	3.84	2.31
2	0.23	56.9	65.0	327	4.05	4.07	2.31
3	0.29	62.4	58.0	334	4.20	4.23	2.63
4	0.31	63.3	58.0	335	4.34	4.35	2.75
...	...	...	...	...	...	...	...
53935	0.72	60.8	57.0	2757	5.75	5.76	3.50
53936	0.72	63.1	55.0	2757	5.69	5.75	3.61
53937	0.70	62.8	60.0	2757	5.66	5.68	3.56
53938	0.86	61.0	58.0	2757	6.15	6.12	3.74
53939	0.75	62.2	55.0	2757	5.83	5.87	3.64

53926 rows × 7 columns

yeni_df ile aykırı olmayan değerlere eriştik. Aykırı değerleri sildik anlamına geliyor.
Şu anda yeni_df ile gözlemlediğimiz tüm değerler, **aykırı olmayan** değerler.

Aykırı değerlere erişmek istersek;

```
[131]: df[df_scores < esik_deger].head()
```

[131]:

	carat	depth	table	price	x	y	z
6341	1.00	44.0	53.0	4032	6.31	6.24	4.12
10377	1.09	43.0	54.0	4778	6.53	6.55	4.12
24067	2.00	58.9	57.0	12210	8.09	58.90	8.06
35633	0.29	62.8	44.0	474	4.20	4.24	2.65
36503	0.30	51.0	67.0	945	4.67	4.62	2.37

Baskılama Yöntemi

```
[132]: df[df_scores == esik_deger]
```

[132]:

	carat	depth	table	price	x	y	z
31230	0.45	68.6	57.0	756	4.73	4.5	3.19

Aykırı gözlemler yerine yukarıda gördüğümüz eşik değeri atayabiliriz.

```
[133]: baski_deger = df[df_scores == esik_deger]

[136]: aykirilar = df[~aykiri_tf]
```

Elimizdeki mevcut aykırıların yerine, eşik değerdeki değerleri atayacak olduğumuzdan dolayı bazı index problemleri ortaya çıkıyor.
Bu index problemlerini giderebilmek adına bir kaç işlem yapacağız.

aykirilar df'ini indexsiz bir array'e çevireceğiz.
Sonrasında baski_deger'i de arraysizleştirip, atama işlemi gerçekleştireceğiz.
Son basamakta da aykırı değerleri baskı değer ile değiştirmiş olacağız.

```
[138]: res = aykirilar.to_records(index = False)
       #index'lerden kurtulduk sadece degerleri kaldi.
       res

[138]: rec.array([(1.  , 44. , 53. ,  4032, 6.31,  6.24,  4.12),
                  (1.09, 43. , 54. ,  4778, 6.53,  6.55,  4.12),
                  (2.  , 58.9, 57. , 12210, 8.09, 58.9 ,  8.06),
                  (0.45, 68.6, 57. ,   756, 4.73,  4.5 ,  3.19),
                  (0.29, 62.8, 44. ,   474, 4.2 ,  4.24,  2.65),
                  (0.3 , 51. , 67. ,   945, 4.67,  4.62,  2.37),
                  (0.73, 70.8, 55. ,  1049, 5.51,  5.34,  3.84),
                  (1.03, 78.2, 54. ,  1262, 5.72,  5.59,  4.42),
                  (0.7 , 71.6, 55. ,  1696, 5.47,  5.28,  3.85),
                  (0.51, 61.8, 54.7,  1970, 5.12,  5.15, 31.8 ),
                  (0.51, 61.8, 55. ,  2075, 5.15, 31.8 ,  5.12),
                  (0.81, 68.8, 79. ,  2301, 5.26,  5.2 ,  3.58),
                  (0.5 , 79. , 73. ,  2579, 5.21,  5.18,  4.09),
                  (0.5 , 79. , 73. ,  2579, 5.21,  5.18,  4.09)],
                 dtype=[('carat', '<f8'), ('depth', '<f8'), ('table', '<f8'),
```

Oluşturduğumus res'lerin yerine baskı değeri atayalım;
(Tüm aykırıların yerine eşik değerimiz)

```
[139]: res[:] = baski_deger.to_records(index = False)
       res

[139]: rec.array([(0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                  (0.45, 68.6, 57., 756, 4.73, 4.5, 3.19),
                 dtype=[('carat', '<f8'), ('depth', '<f8'), ('table', '<f8'),
```

Bu array'i bizim gerçek DataFrame'imizdeki aykırı değerlerin yerine yerleştirmemiz gerekiyor.

```
[141]: df[~aykiri_tf] = pd.DataFrame(res, index = df[~aykiri_tf].index)

[142]: df[~aykiri_tf]
```

[142]:

	carat	depth	table	price	x	y	z
6341	0.45	68.6	57.0	756	4.73	4.5	3.19
10377	0.45	68.6	57.0	756	4.73	4.5	3.19
24067	0.45	68.6	57.0	756	4.73	4.5	3.19
31230	0.45	68.6	57.0	756	4.73	4.5	3.19
35633	0.45	68.6	57.0	756	4.73	4.5	3.19
36503	0.45	68.6	57.0	756	4.73	4.5	3.19
38840	0.45	68.6	57.0	756	4.73	4.5	3.19
41918	0.45	68.6	57.0	756	4.73	4.5	3.19
45688	0.45	68.6	57.0	756	4.73	4.5	3.19
48410	0.45	68.6	57.0	756	4.73	4.5	3.19
49189	0.45	68.6	57.0	756	4.73	4.5	3.19
50773	0.45	68.6	57.0	756	4.73	4.5	3.19
52860	0.45	68.6	57.0	756	4.73	4.5	3.19
52861	0.45	68.6	57.0	756	4.73	4.5	3.19

Başarılı bir şekilde orjinal veri setimizin içerisindeki aykırı değerler yerine eşik değerleri/baskılayıcıları yerleştirdik.

Eksik Gözlem Analizi (Missing Data Analysis)

İncelenen veri setindeki gözlemlerde eksiklik olması durumunu ifade etmektedir.

Araç Fiyatı	KM	Vites Türü	Hasar Durumu	Marka	Model
10000	300000	Manuel	Evet	A	A1
54000	40000	Manuel	Hayır	A	A2
46999	90000	Manuel	Evet	B	B1
89000	1000000	Otomatik	Evet	C	C1
70000	78000	Otomatik	Evet	B	B2
50000	30000	Manuel	Hayır	C	C2
NA	600000	Manuel	Hayır	C	C3
68900	50000	Otomatik	Hayır	B	B2
12000	200000	Manuel	Hayır	A	A1

Eksik değere sahip gözlemlerin veri setinden direk çıkarılması ve rassallığının incelenmemesi yapılacak istatistiksel çıkarımların, modelleme çalışmalarının güvenilirliğini düşürecektir. (Alpar, 2011)

Eksik gözlemlerin veri setinden direk çıkarılabilmesi için veri setindeki eksikliğin bazı durumlarda kısmen bazı durumlarda tamamen rastlantısal olarak oluşmuş olması gerekmektedir. Eğer eksiklikler değişkenler ile ilişkili olarak ortaya çıkan yapısal problemler ile meydana gelmiş ise bu durumda yapılacak silme işlemleri ciddi yanlılıklara sebep olabilecektir.
(Tabachnick ve Fidell, 1996)

Eksik Veriyi Direk Silmenin Zararları

- 1. Veri setindeki eksikliğin yapısal bir eksilik olup olmadığının bilinmesi gerekir!

Müşteriler	Kredi Kartı Harcaması	Kredi Kartı Sahip Olma Durumu
Müşteri1	NA	0 (Sahip değil)

- 2. NA her zaman eksiklik anlamına gelmez!

Müşteriler	Kredi Kartı Harcaması	Kredi Kartı Sahip Olma Durumu
Müşteri1	NA	1 (Kredi Kartına Sahip)

Belki kişi bu ay kredi kartı harcaması yapmadı?

- 3. Bilgi kaybı!

	Değişken1	Değişken2	.	.	.	Değişken100
Müşteri1	.	.	.	.	.	NA
Müşteri2	.	.	.	.	.	.
.	.	.	.	.	.	.
.	.	.	.	.	.	.
Müşteri-n	.	.	.	.	.	.

100 değişkenli bir veri setimiz olduğunu düşünelim, ve bu veri setinin 99 değişkeninde veriler tam ancak 100. değişkeninde veri eksikliği var. Eksik gözlemden kurtulayım düşüncesiyle eksik veri içeren satırı sildiğimizde, 99 tane bilgi içeren değişkeni kaybetmiş oluyoruz.

Eksik Veri Türleri Nelerdir?

- Tümüyle Raslantısal Kayıp: Diğer değişkenlerden ya da yapısal bir problemden kaynaklanmayan tamamen rastgele oluşan gözlemler.

- Raslantısal Kayıp: Diğer değişkenlere bağlı olarak oluşabilen eksiklik türü.

- Raslantısal Olmayan Kayıp: Göz ardı edilemeyecek olan ve yapısal problemler ile ortaya çıkan eksiklik türü.

Eksik Verinin Rassallığının Testi

- Görsel Teknikler
- Bağımsız İki Örneklem T Testi
- Korelasyon Testi
- Little'nin MCAR Testi

Eksik verileri silme ya da doldurma işlemlerini ancak verinin rassallığını tespit ettikten sonra yapabiliriz.

En sık olarak kullanılan yöntemler görsel teknikler ve MCAR testidir.

Eksik Veri Problemi Nasıl Giderilir?

"The idea of imputation is both seductive and dangerous"

(R.J.A Little & D.B. Rubin)

"Atama tekniği, hem çekici hem tehlikelidir."

Çünkü, veri setinin kendi içindeki yapı ve değişkenliğe uygun bir şekilde eksik değerler doldurulmaz ise bu durumda ortaya ilk durumdan daha rahatsız edici, daha yanlı, daha gürültülü, veri yapısı bozulmuş durumlar ortaya çıkabilmektedir.

- **Silme Yöntemleri**
  - Gözlem ya da değişken silme yöntemi
  - Liste bazında silme yöntemi (Listwise Method)
  - Çiftler bazında silme yöntemi (Pairwise Method)
- **Değer Atama Yöntemleri**
  - Ortanca, ortalama, medyan
  - En Benzer Birime Atama (hot deck)
  - Dış Kaynaklı Atama
- **Tahmine Dayalı Yöntemler**
  - Makine Öğrenmesi
  - EM
  - Çoklu Atama Yöntemi

Eksik Veri Hızlı Çözüm

[20]:
```
import numpy as np
import pandas as pd
V1 = np.array([1,3,6,np.NaN,7,1,np.NaN,9,15])
V2 = np.array([7,np.NaN,5,8,12,np.NaN,np.NaN,2,3])
V3 = np.array([np.NaN,12,5,6,14,7,np.NaN,2,31])
df = pd.DataFrame(
        {"V1" : V1,
         "V2" : V2,
         "V3" : V3}
)

df
```

[20]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

[5]:
```
df.isnull().sum() #her değişkendeki null değer sayısı
```

[5]:
```
V1    2
V2    3
V3    2
dtype: int64
```

[6]:
```
df.notnull().sum() #her değişkendeki null olmayan değer sayısı
```

[6]:
```
V1    7
V2    6
V3    7
dtype: int64
```

[7]:
```
df.isnull().sum().sum() #dataset'teki toplam null değer sayısı
```

[7]:
```
7
```

[13]:
```
#en az 1 tane null değer içeren gözlemler
df[df.isnull().any(axis=1)]
```

[13]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
3	NaN	8.0	6.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN

```
[14]: #hepsi dolu olan gözlemler
      df[df.notnull().all(axis=1)]
```

[14]:

	V1	V2	V3
2	6.0	5.0	5.0
4	7.0	12.0	14.0
7	9.0	2.0	2.0
8	15.0	3.0	31.0

```
[15]: #hepsi dolu olan gözlemler, farklı yoldan
      df[df["V1"].notnull() & df["V2"].notnull() & df["V3"].notnull()]
```

[15]:

	V1	V2	V3
2	6.0	5.0	5.0
4	7.0	12.0	14.0
7	9.0	2.0	2.0
8	15.0	3.0	31.0

Eksik Değerlerin Direk Silinmesi

```
[16]: df.dropna()
```

[16]:

	V1	V2	V3
2	6.0	5.0	5.0
4	7.0	12.0	14.0
7	9.0	2.0	2.0
8	15.0	3.0	31.0

dropna() , bir gözlem biriminde sadece bir tane bile null değer varsa o gözlem birimini siler(geçici olarak).
Kalıcı olarak etkilemesi için **dropna(inplace = True)** parametresini eklememiz gerekir.

```
[17]: df
```

[17]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

```
[18]: df.dropna(inplace = True)
```

```
[19]: df
```

[19]:

	V1	V2	V3
2	6.0	5.0	5.0
4	7.0	12.0	14.0
7	9.0	2.0	2.0
8	15.0	3.0	31.0

Basit Değer Atama

```
[21]: df["V1"]
```

```
[21]: 0     1.0
      1     3.0
      2     6.0
      3     NaN
      4     7.0
      5     1.0
      6     NaN
      7     9.0
      8    15.0
      Name: V1, dtype: float64
```

```
[24]: df["V1"].mean()
```

```
[24]: 6.0
```

Değişkenin eksik verilerini istediğimiz bir şey ile doldurmak için **fillna()** kullanırız.

Fonksiyonda parantezler içerisine yazılan değer ile eksik değerler doldurulacaktır. (Geçici olarak)

```
[30]: df["V1"].fillna(df["V1"].mean())
```

```
[30]: 0     1.0
      1     3.0
      2     6.0
      3     6.0
      4     7.0
      5     1.0
      6     6.0
      7     9.0
      8    15.0
      Name: V1, dtype: float64
```

Elimizde 10 tane değişken olduğunu düşünelim, hepsi için eksik değerleri ortalama değerleri ile dolduracağız. Bunu nasıl yaparız?

```
[33]: df.apply(lambda x: x.fillna(x.mean()), axis=0)
```

[33]:

	V1	V2	V3
0	1.0	7.000000	11.0
1	3.0	6.166667	12.0
2	6.0	5.000000	5.0
3	6.0	8.000000	6.0
4	7.0	12.000000	14.0
5	1.0	6.166667	7.0
6	6.0	6.166667	11.0
7	9.0	2.000000	2.0
8	15.0	3.000000	31.0

Eksik Değerlerin Saptanması (Özet)

[40]:
```
#değişkenlerdeki tam değer sayısı
df.notnull().sum()
```

[40]:
```
V1    7
V2    6
V3    7
dtype: int64
```

[43]:
```
#değişkenlerdeki eksik değer sayısı
df.isnull().sum()
```

[43]:
```
V1    2
V2    3
V3    2
dtype: int64
```

[41]:
```
#veri setindeki toplam eksik değer sayısı
df.isnull().sum().sum()
```

[41]:
```
7
```

[42]:
```
#en az bir eksik değere sahip gözlemler
df[df.isnull().any(axis=1)]
```

[42]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
3	NaN	8.0	6.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN

[44]:
```
#tüm değerleri tam olan gözlemler
df[df.notnull().all(axis=1)]
```

[44]:

	V1	V2	V3
2	6.0	5.0	5.0
4	7.0	12.0	14.0
7	9.0	2.0	2.0
8	15.0	3.0	31.0

Eksik Veri Yapısının Görselleştirilmesi

Eksik verilerin incelenmeden direk olarak silinmesinin veya doldurulmasının bazı problemlere sebep olabileceğinden bahsetmiştik.

```
[45]: !pip install missingno
```

•••

```
[46]: import missingno as msno
```

```
[47]: msno.bar(df);
```

Sol eksen: Eksikliklerin yüzdesini ya da tam olmanın yüzdesini ifade eder.
Sağ eksen: Elimizdeki veri setindeki gözlem sayısını ifade eder.
Yukarıdaki rakamlar ise her değişkende kaç adet tam dolu gözlem olduğunu ifade eder.

```
[48]: msno.matrix(df);
```

Eksik değerlerin rassallığı ile ilgili gerekli yapısal bozuklukları yakalayabilme imkanı veren bir görsel.

Sol eksen: Gözlem birimleri (1-2-3-4-5-6-7-8-9) gibi.
Sağ eksen: Aynı anda dolu olan/boş olan değerler.

Planets veri seti üzerinde msno.matrix() grafiğini inceleyelim.

[49]:
```
import seaborn as sns
df = sns.load_dataset('planets')
df.head()
```

[49]:

	method	number	orbital_period	mass	distance	year
0	Radial Velocity	1	269.300	7.10	77.40	2006
1	Radial Velocity	1	874.774	2.21	56.95	2008
2	Radial Velocity	1	763.000	2.60	19.84	2011
3	Radial Velocity	1	326.030	19.40	110.62	2007
4	Radial Velocity	1	516.220	10.50	119.47	2009

[50]:
```
df.isnull().sum()
```

[50]:
```
method              0
number              0
orbital_period     43
mass              522
distance          227
year                0
dtype: int64
```

[51]:
```
msno.matrix(df);
```

orbital_period'da her eksiklik olduğunda mass değişkeninde de eksik gözlem oluşmuş.

mass değişkenindeki eksikliklerin bazıları, orbital_period değişkenine bağlı olarak gerçekleşmiş.

```
[52]: msno.heatmap(df);
```

msno.heatmap() bize değişkenler arasındaki **nullity correlation** değerlerini gösterir.
Örneğin mass ile distance arasındaki nullity correlation değeri 0.5 çıkmış, bu demek oluyor ki bu iki değişkenden birinde eksiklik gözlemliyorsanız diğerinde de eksiklik gözlemleme ihtimaliniz yüksektir.

Yapısal olarak da, nümerik bir karşılık olarak da gözlemliyoruz ki bu veri seti rassal bir eksikliğe sahip değildir. Burada direk doldurma yada direk silme gibi işlemleri yapmak bir takım problemleri de beraberinde getirecektir.
Bu işlemler yapılacaksa, veri setindeki değişkenlerin birbirine olan bağımlılığı mutlaka göz önünde bulundurulmalıdır!

Silme Yöntemi

[1]:
```
import numpy as np
import pandas as pd
V1 = np.array([1,3,6,np.NaN,7,1,np.NaN,9,15])
V2 = np.array([7,np.NaN,5,8,12,np.NaN,np.NaN,2,3])
V3 = np.array([np.NaN,12,5,6,14,7,np.NaN,2,31])

df = pd.DataFrame(
        {"V1" : V1,
         "V2" : V2,
         "V3" : V3}
)

df
```

[1]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

En az 1 eksik değişkene sahip tüm gözlemlerin silinmesi:

[2]:
```
df.dropna() #orjinali bozulmaz
```

[2]:

	V1	V2	V3
2	6.0	5.0	5.0
4	7.0	12.0	14.0
7	9.0	2.0	2.0
8	15.0	3.0	31.0

```
[3]: df
```

[3]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

Tüm gözlem değerleri aynı anda NaN olan gözlemleri ele alalım: (Örn; 6. gözlem)

[4]:
```
df.dropna(how = "all")
```

[4]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
7	9.0	2.0	2.0
8	15.0	3.0	31.0

[5]:
```
df.dropna(axis = 1, how = "all")
#tüm değerleri NaN olan değişkeni siler (bizde yok)
```

[5]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

Değer Atama Yöntemleri

Sayısal Değişkenlerde Atama

[6]:
```
import numpy as np
import pandas as pd
V1 = np.array([1,3,6,np.NaN,7,1,np.NaN,9,15])
V2 = np.array([7,np.NaN,5,8,12,np.NaN,np.NaN,2,3])
V3 = np.array([np.NaN,12,5,6,14,7,np.NaN,2,31])

df = pd.DataFrame(
        {"V1" : V1,
         "V2" : V2,
         "V3" : V3}
)

df
```

[6]:

	V1	V2	V3
0	1.0	7.0	NaN
1	3.0	NaN	12.0
2	6.0	5.0	5.0
3	NaN	8.0	6.0
4	7.0	12.0	14.0
5	1.0	NaN	7.0
6	NaN	NaN	NaN
7	9.0	2.0	2.0
8	15.0	3.0	31.0

NaN değerleri değişkenin ortalaması ile doldurma:

```
[18]: df.mean()
```

```
[18]: V1      6.000000
      V2      6.166667
      V3     11.000000
      dtype: float64
```

```
[7]: df.fillna(df.mean())
```

[7]:

	V1	V2	V3
0	1.0	7.000000	11.0
1	3.0	6.166667	12.0
2	6.0	5.000000	5.0
3	6.0	8.000000	6.0
4	7.0	12.000000	14.0
5	1.0	6.166667	7.0
6	6.0	6.166667	11.0
7	9.0	2.000000	2.0
8	15.0	3.000000	31.0

Aynı işlemi **apply** ve **lambda** kullanarak yapalım;

```
[12]: df.apply(lambda x: x.fillna(x.mean()),axis = 0)
```

[12]:

	V1	V2	V3
0	1.0	7.000000	11.0
1	3.0	6.166667	12.0
2	6.0	5.000000	5.0
3	6.0	8.000000	6.0
4	7.0	12.000000	14.0
5	1.0	6.166667	7.0
6	6.0	6.166667	11.0
7	9.0	2.000000	2.0
8	15.0	3.000000	31.0

İstediğimiz değişkenleri ortalamasıyla, istediğimizi medyanıyla doldurabiliriz;

```
[20]: df.fillna(df.mean()["V1":"V2"])
```

[20]:

	V1	V2	V3
0	1.0	7.000000	NaN
1	3.0	6.166667	12.0
2	6.0	5.000000	5.0
3	6.0	8.000000	6.0
4	7.0	12.000000	14.0
5	1.0	6.166667	7.0
6	6.0	6.166667	NaN
7	9.0	2.000000	2.0
8	15.0	3.000000	31.0

```
[22]: df["V3"].fillna(df["V3"].median())
```

```
[22]: 0     7.0
      1    12.0
      2     5.0
      3     6.0
      4    14.0
      5     7.0
      6     7.0
      7     2.0
      8    31.0
      Name: V3, dtype: float64
```

where ve **notna** kullanarak değer atama;

```
[23]: df.where(pd.notna(df), df.mean(), axis = 'columns')
```

[23]:

	V1	V2	V3
0	1.0	7.000000	11.0
1	3.0	6.166667	12.0
2	6.0	5.000000	5.0
3	6.0	8.000000	6.0
4	7.0	12.000000	14.0
5	1.0	6.166667	7.0
6	6.0	6.166667	11.0
7	9.0	2.000000	2.0
8	15.0	3.000000	31.0

Kategorik Değişken Kırılımında Değer Atama

Eğer elimizde, eksikliğini doldurmak üzere bir değişken varsa öncelikle bu değişkeni, kategorik başka değişkenlerce indirgeyebiliyor muyuz? buna bir bakmak lazım.

Yani örneğin, arge departmanına çalışan kişilerin ortalama maaşını atamak da pazarlama departmanının maaş ortalamasını atamak da şirketin tüm maaşlarının ortalamasını atamakdan daha başarılı olacaktır.

Hatta uzmanlık seviyesini de bir kırılım daha olarak eklersek eğer, çok çok daha doğru bir eksik veri doldurma işlemi yapmış oluruz.

```
[6]: import numpy as np
     import pandas as pd
     V1 = np.array([1,3,6,np.NaN,7,1,np.NaN,9,15])
     V2 = np.array([7,np.NaN,5,8,12,np.NaN,np.NaN,2,3])
     V3 = np.array([np.NaN,12,5,6,14,7,np.NaN,2,31])
     V4 = np.array(["IT", "IT", "IK", "IT", "IK", "IK", "IK", "IT", "IT"])

     df = pd.DataFrame(
             {"maas": V1,
              "V2" : V2,
              "V3" : V3,
              "departman" : V4}
             )
```

```
df
```

[6]:

	maas	V2	V3	departman
0	1.0	7.0	NaN	IT
1	3.0	NaN	12.0	IT
2	6.0	5.0	5.0	IK
3	NaN	8.0	6.0	IT
4	7.0	12.0	14.0	IK
5	1.0	NaN	7.0	IK
6	NaN	NaN	NaN	IK
7	9.0	2.0	2.0	IT
8	15.0	3.0	31.0	IT

```
[8]: #Departmanlara göre gruplayarak ortalama maaşları görelim;
df.groupby("departman")["maas"].mean()
```

```
[8]: departman
IK    4.666667
IT    7.000000
Name: maas, dtype: float64
```

```
[9]: df["maas"].fillna(df.groupby("departman")["maas"].transform("mean"))
```

```
[9]: 0     1.000000
1     3.000000
2     6.000000
3     7.000000
4     7.000000
5     1.000000
6     4.666667
7     9.000000
8    15.000000
Name: maas, dtype: float64
```

Maaş değişkeninin eksikliklerini, departmanların maaş ortalamalarını göz önünde bulundurarak doldurduk.

Kategorik Değişkenler için Eksik Değer Atama

Şimdiye kadar hep sayısal değişkenlerdeki eksik değerleri doldurmayı inceledik. Şimdi ise kategorik değişkenlerdeki eksik değerlerle nasıl başa çıkacağımızı öğreneceğiz.

Kategorik Değişkenler için Eksik Değer Atama

```
[97]: V1 = np.array([1,3,6,np.NaN,7,1,np.NaN,9,15])

      V4 = np.array(["IT", np.NaN, "IK", "IT", "IK", "IK", "IK", "IT", "IT"], dtype=object)

      df = pd.DataFrame(
              {"maas": V1,
               "departman" : V4}
              )
      df
```

[97]:

	maas	departman
0	1.0	IT
1	3.0	NaN
2	6.0	IK
3	NaN	IT
4	7.0	IK
5	1.0	IK
6	NaN	IK
7	9.0	IT
8	15.0	IT

Kategorik değişkenlerde eksik verileri doldururken en çok kullanılan yöntemlerden biri **mod** yöntemidir.

Eksik değer, en sık tekrar eden değer ile doldurulur.

```
[102]: df["departman"].mode()
```

```
[102]: 0    IK
       1    IT
       dtype: object
```

```
[98]: df["departman"].fillna(df["departman"].mode()[0])
      #inplace argumani kullanilmadigi icin kalici degil.
```

```
[98]: 0    IT
      1    IK
      2    IK
      3    IT
      4    IK
      5    IK
      6    IK
      7    IT
      8    IT
      Name: departman, dtype: object
```

1 indexli eksik değeri ‘IK’ değeri ile doldurduk.

Bazı durumlarda mod ile doldurmak yerine, eksik değerin öncesindeki ya da sonrasındaki değer ile doldurmak isteyebiliriz.

Sonrasındaki değer ile doldurma:

```
[103]: df["departman"].fillna(method = "bfill")
```

```
[103]: 0    IT
       1    IK
       2    IK
       3    IT
       4    IK
       5    IK
       6    IK
       7    IT
       8    IT
       Name: departman, dtype: object
```

Öncesindeki değer ile doldurma:

```
[104]: df["departman"].fillna(method = "ffill")
```

```
[104]: 0    IT
       1    IT
       2    IK
       3    IT
       4    IK
       5    IK
       6    IK
       7    IT
       8    IT
       Name: departman, dtype: object
```

Tahmine Dayalı Değer Atama Yöntemleri - KNN & Random Forest & EM

Önceki bölümlerde eksik değerleri ya sildik ya da ortalama, medyan, mod gibi basit değerler ile doldurduk.

Bunların ötesinde, makine öğrenmesi algoritmalarını kullanarak eksik değerleri doldurma işlemlerini gerçekleştirebiliriz.

KNN

KNN

[109]:
```
import seaborn as sns
import missingno as msno

df = sns.load_dataset('titanic')
df = df.select_dtypes(include = ['float64', 'int64'])
print(df.head())
df.isnull().sum()
```

```
   survived  pclass   age  sibsp  parch     fare
0         0       3  22.0      1      0   7.2500
1         1       1  38.0      1      0  71.2833
2         1       3  26.0      0      0   7.9250
3         1       1  35.0      1      0  53.1000
4         0       3  35.0      0      0   8.0500
```

[109]:
```
survived      0
pclass        0
age         177
sibsp         0
parch         0
fare          0
dtype: int64
```

[111]:
```
from ycimpute.imputer import knnimput
```

knnimput kullanırken değişken isimlerimiz silineceğinden dolayı onları ayrı bir değişkene atamamız gerekiyor.

[113]:
```
var_names = list(df)
var_names
```

[113]:
```
['survived', 'pclass', 'age', 'sibsp', 'parch', 'fare']
```

knnimput'u numpy array'leri ile kullanabiliyoruz. df'yi np array'ine çevirelim;

[119]:
```
n_df = np.array(df)
```

```
[121]: n_df[:10]
```

```
[121]: array([[ 0.    ,  3.    , 22.    ,  1.    ,  0.    ,  7.25  ],
               [ 1.    ,  1.    , 38.    ,  1.    ,  0.    , 71.2833],
               [ 1.    ,  3.    , 26.    ,  0.    ,  0.    ,  7.925 ],
               [ 1.    ,  1.    , 35.    ,  1.    ,  0.    , 53.1   ],
               [ 0.    ,  3.    , 35.    ,  0.    ,  0.    ,  8.05  ],
               [ 0.    ,  3.    ,     nan,  0.    ,  0.    ,  8.4583],
               [ 0.    ,  1.    , 54.    ,  0.    ,  0.    , 51.8625],
               [ 0.    ,  3.    ,  2.    ,  3.    ,  1.    , 21.075 ],
               [ 1.    ,  3.    , 27.    ,  0.    ,  2.    , 11.1333],
               [ 1.    ,  2.    , 14.    ,  1.    ,  0.    , 30.0708]])
```

```
[122]: dff = knnimput.KNN(k=4).complete(n_df)
```

```
Imputing row 1/891 with 0 missing, elapsed time: 0.521
Imputing row 101/891 with 0 missing, elapsed time: 0.522
Imputing row 201/891 with 0 missing, elapsed time: 0.523
Imputing row 301/891 with 1 missing, elapsed time: 0.524
Imputing row 401/891 with 0 missing, elapsed time: 0.525
Imputing row 501/891 with 0 missing, elapsed time: 0.525
Imputing row 601/891 with 0 missing, elapsed time: 0.526
Imputing row 701/891 with 0 missing, elapsed time: 0.527
Imputing row 801/891 with 0 missing, elapsed time: 0.527
```

```
[123]: dff = pd.DataFrame(dff, columns = var_names)
```

```
[127]: dff.isnull().sum()
```

```
[127]: survived    0
       pclass      0
       age         0
       sibsp       0
       parch       0
       fare        0
       dtype: int64
```

Gördüğümüz gibi, eksik değerleri KNN algoritması ile doldurduk.

Random Forests

Yine aynı dataframe üzerinde işlem yapacağız.

```
[4]: import pandas as pd
     import numpy as np
     import seaborn as sns

     df = sns.load_dataset('titanic')
     df = df.select_dtypes(include = ['float64', 'int64'])
     print(df.head())
     df.isnull().sum()
```

```
   survived  pclass   age  sibsp  parch     fare
0         0       3  22.0      1      0   7.2500
1         1       1  38.0      1      0  71.2833
2         1       3  26.0      0      0   7.9250
3         1       1  35.0      1      0  53.1000
4         0       3  35.0      0      0   8.0500
```

```
[4]: survived      0
     pclass        0
     age         177
     sibsp         0
     parch         0
     fare          0
     dtype: int64
```

```
[5]: var_names = list(df)

[6]: n_df = np.array(df)

[7]: from ycimpute.imputer import iterforest

[11]: dff = iterforest.IterImput().complete(n_df)
      ...

[10]: dff = pd.DataFrame(dff, columns = var_names)
      ...

[9]: dff.isnull().sum()
      ...
```

Random Forest ile eksik verileri doldurduk.

EM

```
[12]: df = sns.load_dataset('titanic')
      df = df.select_dtypes(include = ['float64', 'int64'])
      print(df.head())
      df.isnull().sum()

         survived  pclass   age  sibsp  parch     fare
      0         0       3  22.0      1      0   7.2500
      1         1       1  38.0      1      0  71.2833
      2         1       3  26.0      0      0   7.9250
      3         1       1  35.0      1      0  53.1000
      4         0       3  35.0      0      0   8.0500

[12]: survived      0
      pclass        0
      age         177
      sibsp         0
      parch         0
      fare          0
      dtype: int64

[13]: from ycimpute.imputer import EM

[14]: var_names = list(df)

[15]: n_df = np.array(df)
```

```
[16]: dff = EM().complete(n_df)

[17]: dff = pd.DataFrame(dff, columns = var_names)

[22]: dff.isnull().sum()

[22]: survived    0
      pclass      0
      age         0
      sibsp       0
      parch       0
      fare        0
      dtype: int64
```

Eksik değerleri bu şekilde tahmine dayalı yöntemlerle dolduracak olsak dahi, mutlaka öncesinde bir yapısal problem var mı? yani rastgelelik problemi var mı? diğer bir ifadesiyle bazı değişkenlerdeki eksiklikler, acaba diğer değişkenlere bağlı olarak mı gerçekleşiyor? bunu bir gözlemlememiz gerekir.

Değişken Standardizasyonu (Veri Standardizasyonu)

Değişken dönüşümü ile değişken standardizasyonu arasındaki farkı bilmek çok önemlidir.

Değişkenin standardizasyonundan bahsedildiğinde, değişkenin kendi içerisindeki bilgi yapısı, varyans yapısı bozulmaz. Fakat belli bir standarda oturtulur.

Örneğin veri setimizde 10 değerinin veri seti küçükten büyüğe sıralandığında 80. sırada olduğunu düşünelim. Bu değişken standartlaştırıldığında bu 10 değeri muhtemelen 1, 2 gibi bir değer ya da muhtemelen 0-1 aralığında bir değer olacak. Fakat veri seti yine küçükten büyüğe sıralandığında bu değer yine 80. sırada olacaktır. Dolayısıyla bir değişken standartlaştırıldığında, değişkenin değerleri değişecektir, belirli bir formata sokulacaktır, fakat taşımış olduğu yayılım, dağılım bilgisinin özütü(kendisi değil) mevcutta kalıyor olacaktır, değişmeyecektir.

Değişken dönüştürmek ise, değiştirilen bir değişkenin taşıdığı bilginin taşıdığı şekliyle kalamaması, dönüştürülmesi demektir.

Örneğin cinsiyet değişkenini düşünelim, kadın-erkek şeklinde. Bunu bir dönüştürme işlemine soktuğumuzda, mesela 0-1 dediğimizde komple yapı değişmiş olacak. Ya da elimizde yaş değişkeni olduğunu düşünelim, 0-10 yaş arasındakileri çocuk olarak, 10-20 yaş arasını genç olarak değiştirmek gibi değişkeni dönüştürmek istediğimizde, normalde yaşın içerisinde bulunan yapı tamamiyle değişiyor. Bu da değişken dönüşümüdür.

Genelde dönüştürme işlemleri, veri setindeki değişkenin özütünü bozar. Ya da amaçlara yönelik olarak onları nümerik olarak ifade etmek gerekir, nümerik temsiller yaratır. Cinsiyet örneğinde olduğu gibi kadın-erkek string değerlerini, bazı fonksiyonların beklentilerinden dolayı 1-0 olarak değiştirir, aslında taşımış olduğu bilgi bu açıdan değişmemiş olur ama yapısı değişir.

Sonraki sayfada örnekler ile devam edeceğiz...

Standardizasyon

```
Standardizasyon

[2]: from sklearn import preprocessing

[6]: preprocessing.scale(df)

[6]: array([[-1.57841037, -0.34554737, -0.70920814],
            [-0.64993368, -0.34554737,  0.92742603],
            [ 0.74278135, -1.2094158 , -0.98198051],
            [ 0.27854301,  0.08638684, -0.70920814],
            [ 1.2070197 ,  1.81412369,  1.47297076]])

[4]: df.mean()

[4]: V1    4.4
     V2    7.8
     V3    8.6
     dtype: float64
```

Veri setinin içindeki bütün değerleri standartlaştırmış olduk. Bu işlemi tüm değişkenlere uyguladığından dolayı, verilerin birbirleri ile kıyaslanabilirliği bozulmamış oldu.

Normalizasyon

Değerleri 0 ile 1 arasına dönüştürmek için kullanılır.

```
Normalizasyon

[7]: preprocessing.normalize(df)

[7]: array([[0.10783277, 0.75482941, 0.64699664],
            [0.21107926, 0.49251828, 0.84431705],
            [0.64699664, 0.53916387, 0.53916387],
            [0.4472136 , 0.71554175, 0.53665631],
            [0.35491409, 0.60842415, 0.70982818]])
```

Min-Max Dönüşümü

Değişkenleri bizim belirlemiş olduğumuz min ve max değerlerinin arasına dönüştürür.

Min-Max Dönüşümü

```
[8]:  scaler = preprocessing.MinMaxScaler(feature_range = (10, 20))

[9]:  scaler.fit_transform(df)

[9]: array([[10.        , 12.85714286, 11.11111111],
            [13.33333333, 12.85714286, 17.77777778],
            [18.33333333, 10.        , 10.        ],
            [16.66666667, 14.28571429, 11.11111111],
            [20.        , 20.        , 20.        ]])
```

Değişken Dönüşümleri

Değişken dönüşümleri > standardizasyon

Her bir standardizasyon aslında bir değişken dönüşümüdür.

```
[1]:  import seaborn as sns
      df = sns.load_dataset('tips')
      df.head()
```

[1]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

Binarize Dönüşüm

Belirli bir eşik değerin altında veya üstündekileri 0 ve 1 yap dediğimizde bu dönüşüm değişkenin standartlaştırılması değil, değişkenin dönüştürülmesi işlemidir.

Ve taşımış olduğu bilgiyi etkilemektedir.

Binarize Dönüşüm

[2]:
```
import numpy as np
import pandas as pd

V1 = np.array([1,3,6,5,7])
V2 = np.array([7,7,5,8,12])
V3 = np.array([6,12,5,6,14])

df = pd.DataFrame(
     {"V1" : V1,
      "V2" : V2,
      "V3" : V3})

df = df.astype(float)
df
```

[2]:

	V1	V2	V3
0	1.0	7.0	6.0
1	3.0	7.0	12.0
2	6.0	5.0	5.0
3	5.0	8.0	6.0
4	7.0	12.0	14.0

[3]:
```
from sklearn import preprocessing
```

[6]:
```
binarizer = preprocessing.Binarizer(threshold = 5).fit(df)
#5'den büyük değerleri 1 yapar.
```

[5]:
```
binarizer.transform(df)
```

[5]:
```
array([[0., 1., 1.],
       [0., 1., 1.],
       [1., 0., 0.],
       [0., 1., 1.],
       [1., 1., 1.]])
```

1-0 Dönüşümü

sex değişkenini 1 ve 0 değerlerine dönüştüreceğiz.

Bu dönüşüm **LabelEncoder** isimli bir fonksiyon ile yapılıyor, bu dünyadaki en sık kullanılan dönüşümlerden birisidir.

df["yeni_sex"] = df["sex"].cat.codes ile de 1-0 dönüşümü yapabiliriz.

Sklearn içerisindeki Scaler'ı kullandığımızda bu ilk yakaladığı değişkeni işaratlemeye 0'dan başlar. Eğer cat.codes kullanırsak bu da ilk yakaladığı değişkeni etiketlemeye 1'den başlar.

```
[2]: from sklearn.preprocessing import LabelEncoder

[3]: lbe = LabelEncoder()

[4]: lbe.fit_transform(df["sex"])

[4]: array([0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0,
       0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 1,
       1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0,
       1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1, 1, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0,
       0, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0,
       1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1,
       1, 0])

[5]: df["yeni_sex"] = lbe.fit_transform(df["sex"])
```

```
[6]: df
```

[6]:

	total_bill	tip	sex	smoker	day	time	size	yeni_sex
0	16.99	1.01	Female	No	Sun	Dinner	2	0
1	10.34	1.66	Male	No	Sun	Dinner	3	1
2	21.01	3.50	Male	No	Sun	Dinner	3	1
3	23.68	3.31	Male	No	Sun	Dinner	2	1
4	24.59	3.61	Female	No	Sun	Dinner	4	0
...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3	1
240	27.18	2.00	Female	Yes	Sat	Dinner	2	0
241	22.67	2.00	Male	Yes	Sat	Dinner	2	1
242	17.82	1.75	Male	No	Sat	Dinner	2	1
243	18.78	3.00	Female	No	Thur	Dinner	2	0

244 rows × 8 columns

“1 ve Diğerleri (0)” Dönüşümü

Örneğin veri setimizde ikiden fazla sınıf var, bunların seçmiş olduğumuz bir tanesini 1’e, diğerlerini 0’a dönüştüreceğiz.

"1 ve Diğerleri (0)" Dönüşümü

[7]:
```
df.head()
```

[7]:

	total_bill	tip	sex	smoker	day	time	size	yeni_sex
0	16.99	1.01	Female	No	Sun	Dinner	2	0
1	10.34	1.66	Male	No	Sun	Dinner	3	1
2	21.01	3.50	Male	No	Sun	Dinner	3	1
3	23.68	3.31	Male	No	Sun	Dinner	2	1
4	24.59	3.61	Female	No	Sun	Dinner	4	0

[8]:
```
import numpy as np
df["yeni_day"] = np.where(df["day"].str.contains("Sun"), 1, 0)
df
```

[8]:

	total_bill	tip	sex	smoker	day	time	size	yeni_sex	yeni_day
0	16.99	1.01	Female	No	Sun	Dinner	2	0	1
1	10.34	1.66	Male	No	Sun	Dinner	3	1	1
2	21.01	3.50	Male	No	Sun	Dinner	3	1	1
3	23.68	3.31	Male	No	Sun	Dinner	2	1	1
4	24.59	3.61	Female	No	Sun	Dinner	4	0	1
...	...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3	1	0
240	27.18	2.00	Female	Yes	Sat	Dinner	2	0	0
241	22.67	2.00	Male	Yes	Sat	Dinner	2	1	0
242	17.82	1.75	Male	No	Sat	Dinner	2	1	0
243	18.78	3.00	Female	No	Thur	Dinner	2	0	0

244 rows × 9 columns

Çok Sınıflı Dönüşüm

```
[10]:
1 from sklearn.preprocessing import LabelEncoder
2 lbe = LabelEncoder()
```

```
[12]:
1 lbe.fit_transform(df["day"])
```

```
[12]: array([2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 1, 1, 1, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 3])
```

ÇOK DİKKAT!

Eğer elinizdeki kategorik değişkenin 2'den fazla sınıfı varsa, ve fonksiyonların bizden beklentisi doğrultusunda, bu kategorik değişkeni bu şekilde numerik bir formata çevirmek istiyorsak ve neticesinde böyle bir çevirme işlemi yaparak bu değişkeni algoritmalara gönderirsek; bu algoritmalar artık bunu 0 ve 3 arasında oluşan değerlerden oluşan bir değişken olarak algılayacak. Yani aslında bakarsanız bu dönüşüm, algoritmaların kafasını karıştıracak. Ve kategorik değişkenin sınıflarının, bağımlı değişkene olan etkileri bozulacaktır. Çünkü burada sınıflar 0, 1, 2, 3 gibi numerik değerlere dönüştü. Normalde bir kategorik değişkenin nominal ölçek türüyle ölçülmüş olan durumu, yani sınıflar arası mesafesi eşit olan durumu, bir anda dönüşüm işlemi ile bozulup, sanki aralarında fark varmışçasına sıralama işlemine tabi tutulmuş oldu.

Bu bir çok probleme sebep olabilmektedir. Yanlılıklara, hatalı çıkarımlara, gürültüye sebep olup algoritmaları şaşırtmaktadır.

Peki bu durumda ne yapacağız?

Bu durumda One-Hot Encoding yapmış olacağız.

One-Hot Dönüşümü ve Dummy Değişken Tuzağı

Elimizdeki bir kategorik değişkeni, nümerik değerlere dönüştürdüğümüzde ortaya çıkan bozukluktan bahsetmiştik.

Bu problemi gidermek için One-Hot Dönüşümü yapılır. One-Hot dönüşümü yapıldığında da bir tuzak ortaya çıkıyor. Bu tuzağı nasıl atlatacağımızı değerlendirmiş olacağız.

One-Hot Dönüşümü ve Dummy Değişken Tuzağı

[12]:
```
df.head()
```

[12]:

	total_bill	tip	sex	smoker	day	time	size	yeni_sex	yeni_day
0	16.99	1.01	Female	No	Sun	Dinner	2	0	1
1	10.34	1.66	Male	No	Sun	Dinner	3	1	1
2	21.01	3.50	Male	No	Sun	Dinner	3	1	1
3	23.68	3.31	Male	No	Sun	Dinner	2	1	1
4	24.59	3.61	Female	No	Sun	Dinner	4	0	1

[14]:
```
import pandas as pd
#pandas ile one-hot dönüşümü yapalım;
df_one_hot = pd.get_dummies(df, columns=["sex"], prefix=["sex"])
#prefix = oluşturulacak olan değişkenlerin ön isimlendirmesinin ne olacağını ifade eder.
```

[15]:
```
df_one_hot.head()
```

[15]:

	total_bill	tip	smoker	day	time	size	yeni_sex	yeni_day	sex_Male	sex_Female
0	16.99	1.01	No	Sun	Dinner	2	0	1	0	1
1	10.34	1.66	No	Sun	Dinner	3	1	1	1	0
2	21.01	3.50	No	Sun	Dinner	3	1	1	1	0
3	23.68	3.31	No	Sun	Dinner	2	1	1	1	0
4	24.59	3.61	No	Sun	Dinner	4	0	1	0	1

Elimizdeki iki sınıflı kategorik değişkenin, sınıf sayısı kadar değişken oluştu. Ve oluşmuş olan yeni değişkenler de 1 ve 0 olarak tanımlandı.

İşte bu dönüşüme, **One-Hot Encoding** adı veriliyor.

Dummy Değişken Tuzağı

Veri seti içerisindeki değişkenlere dönüşüm uyguladığımızda, bu dönüşüm sonrasında oluşturulan yeni değişkenler birbirleri üzerinden oluşturulabiliyorsa bu durum **Dummy Değişken Tuzağı** olarak adlandırılır.

Bir başka ifadeyle; Bir değişkeni ifade eden başka bir değişken varsa bu duruma Dummy Değişken Tuzağı denir.

Örneğimizde göreceğimiz üzere, **sex_Male** değişkeninden, **sex_Female** değişkeni oluşturulabilmektedir. Ya da tam tersi oluşturulabilmektedir. Yani aslında bakarsanız bu iki değişkenin ikisi de aynı bilgiyi taşımaktadır.

Peki bu durumu nasıl konrol altında tutabiliriz?

Şöyle ki, kategorik değişkenin sınıf sayısından daha az sayıda Dummy değişkeni olması gerekiyor. Örneğin buradaki değişkenin iki sınıfı vardı, öyleyse bir tane dummy değişkeni olmalı, değişkenin orjinal hali de veri setinde olmamalı. Çünkü iki değişken de artık aynı bilgiyi ifade ediyor.

Daha fazla sınıfa sahip bir değişkenle deneyelim;

```
[17]: pd.get_dummies(df, columns=["day"], prefix=["day"]).head()
      #prefix = oluşturulacak olan değişkenlerin ön isimlendirmesinin ne olacağını ifade eder.
```

[17]:

	total_bill	tip	sex	smoker	time	size	yeni_sex	yeni_day	day_Thur	day_Fri	day_Sat	day_Sun
0	16.99	1.01	Female	No	Dinner	2	0	1	0	0	0	1
1	10.34	1.66	Male	No	Dinner	3	1	1	0	0	0	1
2	21.01	3.50	Male	No	Dinner	3	1	1	0	0	0	1
3	23.68	3.31	Male	No	Dinner	2	1	1	0	0	0	1
4	24.59	3.61	Female	No	Dinner	4	0	1	0	0	0	1

Bir kategorik değişkenin içindeki birbirinden farklı sınıfların etkilerini, gözlem bazında, veri seti dikeyine taşımış olmaktadır. Yani örneğin makine öğrenmesi bölümünde amacımız, değişkenlerin ayırt ediciliğini ortaya çıkarmak olacak. Burada 10 sınıflı kategorik değişkenin belki de 1 sınıfı, bir çok olayın ortaya çıkmasında ağırlığı olan, yani etkisi olan sınıf olacak. İşte bu durumda bu kategorik değişkenin tek bir değişkenden -> 10 değişkene dönüştürdüğümüzde, bu kategorik değişkenin bir sınıfında yoğunluk varsa o sınıfın etkisi, algoritmaya hissettirilmiş oluyor. Aynı zamanda nümeriğe dönüştürme ihtiyacımız da ortadan kalkmış oluyor.

Özetle, One-Hot Encoding'in bize iki faydası var.

Birincisi, genelde nümerik değişkenlere dönüştürerek algoritmaları kullanıyoruz. Buradaki nümerik dönüştürme işlemini gerçekleştirmiş oluyor.

İkincisi, kategorik değişkenlerin içerisindeki sınıfların etkisini veri setinde hissedilir bir hale dönüştürmemize yardımcı oluyor.

Ek olarak One-Hot dönüşümünde kategorik değişkenin sınıf sayısı arttıkça, değişkenlerin birbirleri üzerinden oluşturulabilmesi durumu da güçleşecektir. Örneğin elinizde 10 sınıflı bir kategorik değişken varsa bunun hepsini one-hot ile dönüştürebiliriz. Çünkü burada birbiri üzerinden aynı bilgileri ifade etme şansımız yoktur. Ama örneğin iki sınıflı bir değişkeni one-hot dönüşümüne tabi tuttuğumuzda iki tane aynı şeyi ifade eden değişken olmuş olacak. Ve birbirleri üzerinden oluşturulabiliyor olacak. İşte bundan kaçınmak gerekir.

Sürekli Değişkeni Kategorik Değişkene Çevirme

Sürekli Değişkeni Kategorik Değişkene Çevirme

```
[2]: import pandas as pd
     import numpy as np
     from sklearn import preprocessing
     import seaborn as sns
```

```
[4]: df = sns.load_dataset("tips")
     df.head()
```

[4]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
[6]: #kullanacak olduğumuz veri setini bu fonksiyona göndermeden önce sürekli değişkenleri seçme işlemi yapmamız gerekiyor.
     dff = df.select_dtypes(include = ["float64", "int64"])
```

```
[7]: est = preprocessing.KBinsDiscretizer(n_bins = [3,2,2], encode = "ordinal", strategy = "quantile").fit(dff)
```

```
[8]: est.transform(dff)[0:10]
```

```
[8]: array([[1., 0., 1.],
            [0., 0., 1.],
            [2., 1., 1.],
            [2., 1., 1.],
            [2., 1., 1.],
            [2., 1., 1.],
            [0., 0., 1.],
            [2., 1., 1.],
            [1., 0., 1.],
            [0., 1., 1.]])
```

Çok zorunda kalmadıkça, elimizdeki sürekli değişkeni kategorik değişkene dönüştürüp ölçeğini bozmak genelde pozitif bir durum değildir. Ama tabi ki ihtiyaçlara göre bazen bu duruma ihtiyacımız olabiliyor.

Değişkeni Index'e, Index'i Değişkene Çevirmek

Değişkeni Index'e, Index'i Değişkene Çevirmek

```
[9]: df.head()
```

[9]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

```
[10]: #Index'i değişken olarak atama
      df["YeniDegisken"] = df.index
```

```
[11]: df
```

[11]:

	total_bill	tip	sex	smoker	day	time	size	YeniDegisken
0	16.99	1.01	Female	No	Sun	Dinner	2	0
1	10.34	1.66	Male	No	Sun	Dinner	3	1
2	21.01	3.50	Male	No	Sun	Dinner	3	2
3	23.68	3.31	Male	No	Sun	Dinner	2	3
4	24.59	3.61	Female	No	Sun	Dinner	4	4
...	...	...	...	...	...	...	...	...
239	29.03	5.92	Male	No	Sat	Dinner	3	239
240	27.18	2.00	Female	Yes	Sat	Dinner	2	240
241	22.67	2.00	Male	Yes	Sat	Dinner	2	241
242	17.82	1.75	Male	No	Sat	Dinner	2	242
243	18.78	3.00	Female	No	Thur	Dinner	2	243

244 rows × 8 columns

```
[12]: df["YeniDegisken"] = df["YeniDegisken"] + 10
      df.head()
```

[12]:

	total_bill	tip	sex	smoker	day	time	size	YeniDegisken
0	16.99	1.01	Female	No	Sun	Dinner	2	10
1	10.34	1.66	Male	No	Sun	Dinner	3	11
2	21.01	3.50	Male	No	Sun	Dinner	3	12
3	23.68	3.31	Male	No	Sun	Dinner	2	13
4	24.59	3.61	Female	No	Sun	Dinner	4	14

```
13]: df.index
```

```
13]: RangeIndex(start=0, stop=244, step=1)
```

```
17]: #degiskeni index olarak tanımlama
     df.index = df["YeniDegisken"]
     df.index
```

```
17]: Int64Index([ 10,  11,  12,  13,  14,  15,  16,  17,  18,  19,
                 ...
                 244, 245, 246, 247, 248, 249, 250, 251, 252, 253],
                dtype='int64', name='YeniDegisken', length=244)
```

---Machine Learning---

Machine Learning I – Giriş

Makine Öğrenmesine Giriş

Makine Öğrenmesi Nedir?

Bilgisayarların insanlara benzer şekilde öğrenmesini sağlamak maksadıyla çeşitli algoritma ve tekniklerin geliştirilmesi için çalışılan bilimsel çalışma alanıdır.

Ülke	Yaş	Maaş	Satın Alma Durumu
Türkiye	55	72000	Hayır
İspanya	32	60000	Evet
Almanya	42	50000	Evet
Fransa	45	30000	Hayır
Türkiye	35	32000	Evet
Almanya	32	35000	Hayır
İspanya	50	70000	Hayır
Almanya	29	32000	Evet
Türkiye	30	33000	Evet
Fransa	40	40000	Evet

- 32 yaşında 30 Bin TL geliri olan bir Türkiye vatandaşı ürün alır mı?
- 46 yaşında 32 Bin TL geliri olan bir Fransız ürünü alır mı?
- Türkiye'ye en benzer davranışı gösteren ülke hangisidir?

Burada gözlem sayısı az olduğundan insan olarak kendimiz inceleyerek bu sorulara yanıtlar verebiliriz. Peki ya milyonlarca satır veri olsaydı bu yanıtları verebilir miydik?

İşte makine öğrenmesi ihtiyacının tam olarak doğduğu anlardan bir tanesi burası, ölçeklenebilirlik. Bizim insan olarak yapmış olduğumu bu çıkarımı matematiksel formülasyonlar ile ifade edip, değişkenler arasındaki ilişkileri tanımlayıp, bazı bağımsız değişkenler değerlerini aldığında buna karşılık bağımlı değişken dediğimiz, sürekli ya da kategorik olacak bu değişkenin, hangi değerleri aldığı durumlarını ifade ettiğimizde (denklemler, formülasyonlar, karar kuralları ile) makine öğrenmesi yapmış oluyoruz. Makine öğrenmesi, bu işi bilgisayar yazılımlarına yaptırmaktır.

Terminoloji

Makine Öğrenmesinde sık kullanılan kavramlara kısa bir bakış.

Öğrenme Türleri

Gözetimli Öğrenme; Veri setimiz içerisinde ilgilendiğimiz yanıt değişkeni var ise, yani bağımsız değişkenlerin birlikte etkileşimleri sonucunda ortaya çıkardıkları durumun ne olduğu bilgisi var ise bu durum, gözetimli, denetimli, supervised learning ifadeleri ile tanımlanır.

	y1	x1	x2	x3	x4
1	8.04	10	10	10	8
2	6.95	8	8	8	8
3	7.58	13	13	13	8
4	8.81	9	9	9	8
5	8.33	11	11	11	8
6	9.96	14	14	14	8
7	7.24	6	6	6	8
8	4.26	4	4	4	19
9	10.84	12	12	12	8
10	4.82	7	7	7	8
11	5.68	5	5	5	8

Gözetimsiz Öğrenme; Veri setimizdeki gözlemlere anlamlı bilgiler çıkarmak üzere odaklandığımızda, yanıt değişkeninin yani bağımlı değişkenin olmamasıdır.

Bağımlı değişken için bazen yanıt değişkeni bazen çıktı değişkeni gibi ifadeler de denilebilir. Ağaca dayalı yöntemlerde hedef değişken (target variable) olarak görebiliriz. Yapay sinir ağları gibi yaklaşımlarda bunları çıktı olarak da görebiliriz. Bunların hepsi aynı anlama gelir.

Problem Türü

Regresyon; Bağımlı değişken süreklidir. (Hava Durumu, Yaş, Fiyat)

Sınıflandırma; Bağımlı değişken kategoriktir. (Kadın-Erkek, Kedi-Köpek)

Değişken ve Ölçek Türleri

- Değişken Türleri
  - Sayısal Değişkenler (nicel, kantitatif)
  - Kategorik Değişkenler (nitel, kalitatif)
- Ölçek Türleri
  - Sayısal değişkenler için: Aralık ve Oran
  - Kategorik değişkenler için: Nominal ve Ordinal

Test-Train Ayrımı

Feature Engineering

Kişinin kendisinin veya otomatikleştirdiği sistemlerin, veri setimiz veya veritabanları içerisinden değişkenler oluşturması işlemidir.

Variable Selection

Burada renklerini kaybetmiş olan değişkenler, Y bağımlı değişkenini tahmin etmek söz konusu olduğunda, başarılı olmayan, ayırt ediciği olmayan ve bir şekilde kullanıcı tarafından veya otomatik olarak çalışmanın dışarısında bırakılmış olan değişkenlerdir. Renkli olan değişkenler ise çalışma yapılacak üzere seçilmiş olan değişkenlerdir.

Model Seçimi

Model seçimi için iki durum söz konusu;

- **Birincisi;** Oluşabilecek değişken kombinasyonları ile oluşturulan modeller arasından en iyi modelin seçilmesi.
- **İkincisi;** Kurulan birbirinden farklı modeller arasından model seçimi.

Model neye göre seçilir?

- Regresyon için açıklanabilirlik oranı ve RMSE benzeri bir değer.
- Sınıflandırma için doğru sınıflandırma oranı benzeri bir değer.

Overfitting

Modelin eğitim verilerini aşırı ezberleyip, rassallığını kaybetmesi ve test verilerinde(yeni bir veride) düzgün sonuç verememesidir.

Deterministik Modeller vs Stokastik Modeller

Değişkenler arasında kesin bir ilişki olduğunu varsayan modellere **Deterministik Modeller** denir. X'e vereceğim herhangi bir değere karşılık Y'nin alacağı değeri kesin olarak bilirim.

Bizim ilgilendiğimiz ise, makine öğrenmesi kapsamında *tesadüfi hata* barındıran ve X'e karşılık, Y'nin alacağı değerleri ortaya çıkaran rassallık kavramının da olduğu, hata barındırma durumunun da olduğu olasılıksal modellerdir. Bunlara ise **Stokastik Modeller** denir.

Doğrusal Modeller vs Doğrusal Olmayan Modeller

Genelde makine öğrenmesi söz konusu olduğunda, modellerin doğrusal ve doğrusal olmaması konusunda farkındalığa sahip olmak bize, görecek olduğumuz doğrusal modeller konusundaki modelleri ve bunların yorumlarıyla ilgili farkındalık sağlar.

Doğrusal olmama durumunun (genelde bir çok problemde bununla karşılaşırız.) farkında olmak, kuracak olduğumuz kural tabanlı, ağaca dayalı ya da regression modellerinin arka tarafındaki yapıyla ilgili bir bilgi sahibi olma imkanı tanır.

---Machine Learning Days---

MLD-Data Visualization

Numeric ve **Categoric** veri tiplerimiz var.
Kedi-köpek ya da sıcak-soğuk gibi nitel veriler **categoric** verilerdir.
Eğer categoric verilerle tahminleme yapıyorsak **Classification** problemi çözüyoruz.
İnsan yaşları gibi numeric verilerle tahminleme yapıyorsak **Regression** problemi çözüyoruz.

```
[2]: import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     #kütüphanelerimizi ekledik.
```

```
[7]: df = pd.read_csv("kaggle/datasets_228_482_diabetes.csv")
     #kaggle isimli klasördeki .csv uzantılı veri setimizi aldık.
     df.head()
```

[7]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

Veri Setinin Hikayesi

Veri kümesinin amacı, veri kümesine dahil edilen belirli tanı ölçümlerine dayanarak bir hastanın diyabet olup olmadığını teşhis amaçlı olarak tahmin etmektir.
Veri kümeleri birkaç tıbbi öngörücü değişken ve bir hedef değişkenden oluşur, **Outcome**.
Tahmin değişkenleri hastanın sahip olduğu gebelik sayısını, BMI'sını, insülin seviyesini, yaşını vb. İçerir.

- **Pregnancies**: Hamile sayısı
- **Glucose**: Oral glukoz tolerans testinde 2 saatteki plazma glikoz konsantrasyonu
- **BloodPressure**: Diyastolik kan basıncı (mm Hg)
- **SkinThickness**: Triceps deri kat kalınlığı (mm)
- **Insulin**: 2 saatlik serum insülini (mu U / ml)
- **BMI**: Vücut kitle indeksi (kg olarak ağırlık / (m olarak yükseklik) ^ 2)
- **DiabetesPedigreeFunctio**: Diyabet soyağacı işlevi
- **Age**: Yaş
- **Outcome**: Sonuç (1 yada 0)

Outcome categoric, diğer değişkenler ise numeric veri.

```
[9]: df.describe().T
```

[9]:

	count	mean	std	min	25%	50%	75%	max
Pregnancies	768.0	3.845052	3.369578	0.000	1.00000	3.0000	6.00000	17.00
Glucose	768.0	120.894531	31.972618	0.000	99.00000	117.0000	140.25000	199.00
BloodPressure	768.0	69.105469	19.355807	0.000	62.00000	72.0000	80.00000	122.00
SkinThickness	768.0	20.536458	15.952218	0.000	0.00000	23.0000	32.00000	99.00
Insulin	768.0	79.799479	115.244002	0.000	0.00000	30.5000	127.25000	846.00
BMI	768.0	31.992578	7.884160	0.000	27.30000	32.0000	36.60000	67.10
DiabetesPedigreeFunction	768.0	0.471876	0.331329	0.078	0.24375	0.3725	0.62625	2.42
Age	768.0	33.240885	11.760232	21.000	24.00000	29.0000	41.00000	81.00
Outcome	768.0	0.348958	0.476951	0.000	0.00000	0.0000	1.00000	1.00

```
[10]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 768 entries, 0 to 767
Data columns (total 9 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   Pregnancies               768 non-null    int64
 1   Glucose                   768 non-null    int64
 2   BloodPressure             768 non-null    int64
 3   SkinThickness             768 non-null    int64
 4   Insulin                   768 non-null    int64
 5   BMI                       768 non-null    float64
 6   DiabetesPedigreeFunction  768 non-null    float64
 7   Age                       768 non-null    int64
 8   Outcome                   768 non-null    int64
dtypes: float64(2), int64(7)
memory usage: 54.1 KB
```

```
[12]: df.isna().any()
      #Column'da bir tane bile null deger varsa True olur.

[12]: Pregnancies                 False
      Glucose                     False
      BloodPressure               False
      SkinThickness               False
      Insulin                     False
      BMI                         False
      DiabetesPedigreeFunction    False
      Age                         False
      Outcome                     False
      dtype: bool

[14]: df.notna().any()
      #Column'da bir tane bile dolu deger varsa True olur.

[14]: Pregnancies                 True
      Glucose                     True
      BloodPressure               True
      SkinThickness               True
      Insulin                     True
      BMI                         True
      DiabetesPedigreeFunction    True
      Age                         True
      Outcome                     True
      dtype: bool
```

```
[18]: df.isna().all()
      #tamamı null olan column'lar True olur.
```

```
[18]: Pregnancies                 False
      Glucose                     False
      BloodPressure               False
      SkinThickness               False
      Insulin                     False
      BMI                         False
      DiabetesPedigreeFunction    False
      Age                         False
      Outcome                     False
      dtype: bool
```

```
[20]: df.notna().all()
      #tamamı dolu olan column'lar True gelir.
      #Hepsi True gelirse eksik veri yok demektir.
```

```
[20]: Pregnancies                 True
      Glucose                     True
      BloodPressure               True
      SkinThickness               True
      Insulin                     True
      BMI                         True
      DiabetesPedigreeFunction    True
      Age                         True
      Outcome                     True
      dtype: bool
```

```
[22]: df.isna().sum()
      #degiskenlerdeki eksik veri sayisi.
```

```
[22]: Pregnancies                 0
      Glucose                     0
      BloodPressure               0
      SkinThickness               0
      Insulin                     0
      BMI                         0
      DiabetesPedigreeFunction    0
      Age                         0
      Outcome                     0
      dtype: int64
```

```
[26]: #Sadece 1 tane sınıfımız var. Görselleştirirken 1 tane daha sınıfımız olsa iyi olabilir.
      #Overweight adında yeni bir sınıf ekleyelim.
      #Vücut kitle indeksi 25'den büyük ise 1 değil ise 0 olsun.

      df["Overweight"] = [1 if x > 25 else 0 for x in df.BMI]
      df.head()
```

[26]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Overweight
0	6	148	72	35	0	33.6	0.627	50	1	1
1	1	85	66	29	0	26.6	0.351	31	0	1
2	8	183	64	0	0	23.3	0.672	32	1	0
3	1	89	66	23	94	28.1	0.167	21	0	1
4	0	137	40	35	168	43.1	2.288	33	1	1

Veri Görselleştirme

Relational Plots with Matplotlib

Relational Plots iki tane değişkenin arasındaki ilişkiyi gösteren grafiklerdir.

- **Scatter Plot**: İki değişken arasındaki ilişkinin dağılımını veri noktalarıyla gösterir.
- **Lineplot**: İki değişken arasındaki ilişkiyi sürekli gösterir. Veri noktaları birbirine çizgilerle bağlıdır. (Zaman serilerinde kullanılır.)
- **s parametresi:** marker boyutu
- **c parametresi:** marker rengi, hangi değişkeni tuttuğu da yazılabilir.
- **alpha:** marker opaklığı

```
[11]: plt.rcParams.update({'font.size': 25})
      #grafiklerimizdeki font size'ı bu şekilde güncelleyebiliriz.
```

```
[12]: sns.set_context("paper")
```

```
[28]: plt.scatter(df.Age, df.Insulin, c=df.Overweight, s=389,
                  alpha=0.2, cmap="viridis") #cmap renk paleti
      plt.colorbar(); #hangi rengin hangi değere denk geldiğini gösteren yandaki ölçek
      plt.xlabel("Age") #eksen ismi
      plt.ylabel("Insulin")
      plt.title("Relationship between Age and Insulin") #plot ismi
      plt.show()

      #insulin degerinde 0'da bir yığılma var ve bu bir sıkıntı. Olmaması gerekir.
```

Scatter plot with Subplots

subplot'ı bir plottan iki tane küçük plot çıkarıyoruz gibi düşünebiliriz.

fig, ax = plt.subplots(): figure ve axes object oluşturur. figure'de her şey var, axes data'yı tutuyor.

```
[33]: fig, ax = plt.subplots(1,2) #1 satir, 2 sütundan oluşan plot
      plt.show()
```

```
[34]: fig, ax = plt.subplots()
ax.scatter(df.Age, df.Insulin, c=df.Overweight, cmap="viridis")
ax.set_xlabel("Age")
ax.set_ylabel("Insulin")
ax.set_title("Relationship between Age and Insulin")
plt.show()
```

Categorical Plots with Matplotlib

Histogram

Numerik ya da kategorik verilerde dağılımı yorumlamamıza yardımcı olur.

```
[43]: fig, ax = plt.subplots()
ax.hist(df.Age, label="Age", bins=10) #bins: kaç aralığa bölünecek
ax.set_xlabel("Age") #axis isimleri
ax.set_ylabel("Number of Observations")
plt.show()
```

```
[46]: bins=[20,30,40,50,60,70,80] #bins'i manuel girdik.
fig, ax = plt.subplots()
ax.hist(df.Age, label="Age", bins=bins)
ax.set_xlabel("Age") #axis isimleri
ax.set_ylabel("Number of Observations")
plt.show()
```

Bar Plot

Kategorik verilerin özelliklerine bakmamızı sağlar.

```
[47]: fig, ax = plt.subplots()
ax.bar(df.Outcome, df.Insulin)
ax.set_xlabel("Outcome")
ax.set_ylabel("Insulin")
plt.show()
```

Figure Kaydetme

```
[52]: #Yaşlara göre insulin değerlerine bakalım.
fig, ax = plt.subplots()
ax.bar(df.Age, df.Insulin)
ax.set_xticklabels(df.Age, rotation=90) # x eksenindeki yazıların yazı yönü.
fig.savefig("Age.png", dpi=500) #png formatında kaydeder.
```

- **fig.savefig("Age.png"):** kayıp olmadan kaydeder, yüksek kalitelidir ama çok hafıza tutar
- **fig.savefig("Age.jpg", quality=50):** websitesine konulabilir
- **fig.savefig("Age.png", dpi=200):** dots per inch, dense rendering
- **fig.set_size_inches([5,3]):** aspect ratio

Seaborn

- **FacetGrid** (relplot(), catplot()) subplot'lar oluşturabilir.
- **AxesSubplot**(scatterplot, countplot) bir tane plot oluşturur.

Count Plot & Cat Plot

Count Plot

[60]:
```
sns.set_palette("RdBu")
sns.countplot(x="Age", data=df)
plt.show()
```

[68]:
```
sns.catplot(x="Age", aspect=3, data=df, kind="count") #aspect = x ekseni, y ekseninin 3 katı kadar olsun.
plt.show()
```

[69]:
```
g = sns.catplot(x="Age", aspect=3, data=df, kind="count")
g.fig.suptitle("Age Counts", y=1.04) #ismi yukarı çıkarıyor.
plt.show()
```

```
[71]: g = sns.catplot(x="Age", aspect=3, data=df, kind="count")
plt.xticks(rotation=30) #x eksenindeki isimleri 30 derece döndürür.
plt.show()
```

Scatter Plot

Scatter Plot

```
[72]: sns.scatterplot(x="Age", y="Insulin", data=df, hue="Outcome")
plt.show()
```

```
[73]: sns.relplot(x="Age", y="Insulin", data=df, hue="Outcome",
                  kind="scatter")
plt.show()
```

Line Plot

Line Plot

[83]:
```
df.head()
```

[83]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome	Overweight
0	6	148	72	35	0	33.6	0.627	50	1	1
1	1	85	66	29	0	26.6	0.351	31	0	1
2	8	183	64	0	0	23.3	0.672	32	1	0
3	1	89	66	23	94	28.1	0.167	21	0	1
4	0	137	40	35	168	43.1	2.288	33	1	1

[97]:
```
sns.relplot(x="Age", y="Insulin", data=df, kind="line", ci="sd", aspect = 3, markers=True, dashes=False)
plt.show()
```

[98]:
```
sns.relplot(x="Age", y="Insulin", data=df, kind="line", ci=None, aspect = 3, markers=True, dashes=False)
plt.show()
```

Scatter Subplots

Scatter Subplots

[101]:
```
sns.relplot(x="Insulin", y="Glucose", data=df, kind="scatter", col="Outcome") #Glucose'a göre karşılaştırma
plt.show()
```

```
[102]: sns.relplot(x="Insulin", y="Glucose", data=df, kind="scatter", row="Outcome") #Insulin'e göre karşılaştırma
plt.show()
```

```
[103]: sns.relplot(x="Insulin", y="Glucose", data=df, kind="scatter",col="Outcome", row="Overweight")
plt.show()
```

Heatmap

Öznitelikler arasındaki ilişkiye, korelasyona bakmamızı sağlar.

Korelasyon ne kadar iyiyse makine öğrenmesi modelimiz o kadar düzgün çalışır.

```
sns.set_palette("RdBu")
correlation=df.corr()
sns.heatmap(correlation, annot=True) #annot: corr degerlerini heatmap üzerine yazar.
plt.show()
#Renk ne kadar açıksa correlation o kadar yüksek demektir.
```

Categoric Plot

Categorical Plots

```
sns.catplot(x="Outcome",y="Insulin",data=df, kind="bar")
plt.show()
```

Bar plot bize kategorik veri hakkında bilgi verir.

Box Plot

[122]:
```
sns.catplot(x="Outcome",y="Age", data=df, kind="box")
plt.show()
```

[130]:
```
sns.set_style("darkgrid") #arka plan tasarımı
sns.catplot(x="Outcome", data=df, kind="count")
plt.show()
```

Data Visualization Quiz

Aşağıdakilerden hangisi categorical plot değildir? *

- ○ Bar Plot
- ○ Count Plot
- ○ Box Plot
- ◉ Scatter Plot

"ci" parametresinin işlevi aşağıdakilerden hangisidir? *

- ○ Subplot yapmamızı sağlar
- ○ Veri noktalarında sınıfları renklendirir
- ◉ Line plot'ta güven aralığını (confidence interval) hangi istatistiki ölçüyle göstereceğimizi belirler
- ○ Plot tipini belirler

Aşağıdaki metodlardan hangisi subplot çizmemizi sağlamaz? *

- ○ sns.relplot()
- ○ sns.catplot()
- ◉ sns.heatmap()
- ○ plt.subplots()

Aşağıdaki seaborn parametrelerinden hangisi x eksenini genişletir? *

- ○ hue
- ○ size
- ○ data
- ◉ aspect

Box plot hangi istatistik ölçüsünü göstermez? *

- ◉ Ortalama(mean)
- ○ Medyan
- ○ Kartiller
- ○ Aykırı veriler

Tamamı kayıp verilerden (NA) oluşan kolonları aşağıdaki komutlardan hangisi gösterir? *

- ◉ df.isna().all()
- ○ df.notna().sum()
- ○ df.isna().any()
- ○ df.notna().all()

Aşağıdaki pandas metodlarından hangisi veri tiplerini döndürür? *

- ○ df.head()
- ○ df.tail()
- ○ df.describe()
- ◉ df.info()

MLD-Data Preprocessing

[1]:
```
import pandas as pd
import numpy as np
```

[22]:
```
dataset = {"İsim": ["Mert", "Nilay", "Dogancan", "Omer", "Merve", "Onur"],
           "Soyad": ["Cobanov", "Mertal", "Mavideniz", "Cengiz", "Noyan", "Sahil"],
           "Yas": [24, 22, 24, 23, "bilinmiyor", 23],
           "Sehir": ["Bursa", "Ankara", "Istanbul", np.nan, "Izmir", "Istanbul"],
           "Ulke": ["Turkiye", "Turkiye", "Turkiye", "Turkiye", "Turkiye", "Turkiye"],
           "GANO": [np.nan, np.nan, np.nan, np.nan, 3.90, np.nan]}

df = pd.DataFrame(dataset)
df
```

[22]:

	İsim	Soyad	Yas	Sehir	Ulke	GANO
0	Mert	Cobanov	24	Bursa	Turkiye	NaN
1	Nilay	Mertal	22	Ankara	Turkiye	NaN
2	Dogancan	Mavideniz	24	Istanbul	Turkiye	NaN
3	Omer	Cengiz	23	NaN	Turkiye	NaN
4	Merve	Noyan	bilinmiyor	Izmir	Turkiye	3.9
5	Onur	Sahil	23	Istanbul	Turkiye	NaN

1. Adım: Büyük resime bakın!

Her şeyden önce, bir preprocessing işlemine başlarken, veri tiplerine, satır-sütün sayılarına, eksik verilere ve genel şemaya bakarak başlamalısınız. Burada <DataFrame>.info() fonksiyonu ile bir önbilgi alınabilir.

- İlk dikkatimi çeken unsur Yas kolonunun integer olması yerine object olması. Dataframe'e dönüp baktığımda yaşlardan birinin bilinmiyor olarak kodlandığını görüyorum. Eğer sayılardan oluşan bir kolonda farklı bir datatype varsa, pandas bunun object olarak algılayacaktır.
- Dikkatimi çeken diğer bir unsur Sehir ve GANO kolonundaki eksik değerler, bunların halledilmesi gerekecek.
- Toplam 6 satır olmasına rağmen GANO kolonunda sadece tek bir değer görebiliyorum, burada bu kolonu tamamen kaldırmak mantıklı olacağını düşünüyorum.
- Ulke kolonundaki tüm değerler aynı, bu yüzden kaldırabiliriz.

```
[3]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6 entries, 0 to 5
Data columns (total 6 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   İsim    6 non-null      object
 1   Soyad   6 non-null      object
 2   Yas     6 non-null      object
 3   Sehir   5 non-null      object
 4   Ulke    6 non-null      object
 5   GANO    1 non-null      float64
dtypes: float64(1), object(5)
memory usage: 416.0+ bytes
```

NaN kontrolü

NaN değerleri saydırarak kontrol edelim. Eğer bir kez .sum() fonksiyonunu çağırırsam, kolon bazında toplayacaktır, eğer bir kez daha çağırırsam, eksik değerlerimin toplamını da görebilirim.

```
[7]: df.isna().sum()
```

```
[7]: İsim     0
     Soyad    0
     Yas      0
     Sehir    1
     Ulke     0
     GANO     5
     dtype: int64
```

```
[8]: df.isna().sum().sum() #df'de toplam kaç adet NaN value var?
```

```
[8]: 6
```

2. Adım: Manipülasyona Başlayın!

Bilgi içermeyen kolonların kaldırılması

GANO ve Ulke satırlarının kaldırılmasına karar vermiştik, bunu yapabileceğimiz iki yöntem var:

- Önkabul olarak, eğer kolonlar belirli bir eşik değerinin üzerinde *NaN* değer içerdiğinde kaldırmak istiyorsanız
- Seçtiğiniz kolonları manuel olarak kaldırmak istiyorsanız

```
[10]: # 1. Yöntem
df.dropna(axis=1, how="any", thresh=3) # 3 taneden fazla NaN değeri içeren column'u kaldıracak.
# GANO column'u kaldırıldı.
```

[10]:

	İsim	Soyad	Yas	Sehir	Ulke
0	Mert	Cobanov	24	Bursa	Turkiye
1	Nilay	Mertal	22	Ankara	Turkiye
2	Dogancan	Mavideniz	24	Istanbul	Turkiye
3	Omer	Cengiz	23	NaN	Turkiye
4	Merve	Noyan	bilinmiyor	Izmir	Turkiye
5	Onur	Sahil	23	Istanbul	Turkiye

```
[13]: # 2. Yöntem
df.drop(labels=["GANO"], axis=1)
# Eğer aynı dataframe'inize direk uygulamak istiyorsanız inplace parametresine True değerini verin.
# df.drop(labels=["GANO"], axis=1, inplace=True)
```

[13]:

	İsim	Soyad	Yas	Sehir	Ulke
0	Mert	Cobanov	24	Bursa	Turkiye
1	Nilay	Mertal	22	Ankara	Turkiye
2	Dogancan	Mavideniz	24	Istanbul	Turkiye
3	Omer	Cengiz	23	NaN	Turkiye
4	Merve	Noyan	bilinmiyor	Izmir	Turkiye
5	Onur	Sahil	23	Istanbul	Turkiye

Ayrıca unutmadan Ulke satırındaki her değer aynı olduğu için modelimizin buna ihtiyacı olmayacak.

```
[19]: df
```

[19]:

	İsim	Soyad	Yas	Sehir	Ulke	GANO
0	Mert	Cobanov	24	Bursa	Turkiye	NaN
1	Nilay	Mertal	22	Ankara	Turkiye	NaN
2	Dogancan	Mavideniz	24	Istanbul	Turkiye	NaN
3	Omer	Cengiz	23	NaN	Turkiye	NaN
4	Merve	Noyan	bilinmiyor	Izmir	Turkiye	3.9
5	Onur	Sahil	23	Istanbul	Turkiye	NaN

```
[34]: #df.drop(columns=["GANO", "Ulke"], inplace=True) #bu kez labels yerine columns kullandık.

      df_2 = df.drop(columns=["GANO", "Ulke"])
      #inplace kullanmadan degisiklik yapılmış halini başka bir degiskene tanımlayabiliriz.
```

```
[35]: df_2 #GANO ve Ulke column'ları gitti.
```

[35]:

	İsim	Soyad	Yas	Sehir
0	Mert	Cobanov	24	Bursa
1	Nilay	Mertal	22	Ankara
2	Dogancan	Mavideniz	24	Istanbul
3	Omer	Cengiz	23	NaN
4	Merve	Noyan	bilinmiyor	Izmir
5	Onur	Sahil	23	Istanbul

Eksik değerlerin halledilmesi

Eksik değerlerin halledilmesiyle ilgili basit ve daha kompleks yöntemler var, burada amaç verisetimizde dezenformasyon yaratmadan bu problemlerin halledilmesi olmalı. Özellikle ML algoritmaları eksik verilere uyumlu değiller, bu yüzden ön işleme esnasında kritik konulardan birisini bu kısım oluşturuyor.

Konunun önem derecesi arttıkça yaklaşımlarda değişiyor, genel bir yöntem ve herkesin kabul ettiği bir yaklaşım yok fakat size en popüler olanlarını göstermeye çalışacağım.

Bu verileri direkt olarak kaldırabildiğiniz durumları yukarıda işledik, şimdi gelin kaldırmak istemediğimiz durumlarda neler yapabiliriz bunlara bakalım.

- Mean, Median, Frequent, Constant
- Enterpolasyon
- KNN

1. En kolay teknik

Manuel

```
[36]: df_2["Yas"]
```

```
[36]: 0            24
      1            22
      2            24
      3            23
      4    bilinmiyor
      5            23
      Name: Yas, dtype: object
```

```
[37]: df_2["Yas"].replace("bilinmiyor", np.nan, inplace=True)
      df_2["Yas"] # ilk önce eksik veriyi NaN formatına çeviriyorum
```

```
[37]: 0    24.0
      1    22.0
      2    24.0
      3    23.0
      4     NaN
      5    23.0
      Name: Yas, dtype: float64
```

```
[31]: df_2.fillna(value=df_2["Yas"].mean(), inplace=True) # sonrasında o columnun ortalaması ile dolduruyorum
      df_2["Yas"]
```

```
[31]: 0    24.0
      1    22.0
      2    24.0
      3    23.0
      4    23.2
      5    23.0
      Name: Yas, dtype: float64
```

Scikit

Scikit ile bu işlem oldukça kolaylaştırılmış, tekniğinize göre 4 yöntem seçebiliyorsunuz.

- **mean:** Ortalama değer impute edilir.
- **median:** Medyan impute edilir.
- **most_frequent:** En çok tekrar eden değer eklenir.
- **constant:** sabit bir değer eklenir.

```
[39]: from sklearn.impute import SimpleImputer
```

```
[38]: df_2
```

[38]:

	İsim	Soyad	Yas	Sehir
0	Mert	Cobanov	24.0	Bursa
1	Nilay	Mertal	22.0	Ankara
2	Dogancan	Mavideniz	24.0	Istanbul
3	Omer	Cengiz	23.0	NaN
4	Merve	Noyan	NaN	Izmir
5	Onur	Sahil	23.0	Istanbul

```
[40]: imp_freq = SimpleImputer(missing_values=np.nan, strategy="most_frequent")
```

```
[41]: df_2["Yas"] = imp_freq.fit_transform(df_2[["Yas"]])
      df_2
```

[41]:

	İsim	Soyad	Yas	Sehir
0	Mert	Cobanov	24.0	Bursa
1	Nilay	Mertal	22.0	Ankara
2	Dogancan	Mavideniz	24.0	Istanbul
3	Omer	Cengiz	23.0	NaN
4	Merve	Noyan	23.0	Izmir
5	Onur	Sahil	23.0	Istanbul

2. Enterpolasyon

Bu teknik biraz trickli olabilir, çünkü sürekli olduğunuz bir veride kullanmanız mantıklı olacaktır. Interpolasyon, elinizdeki veri noktalarının arasında bir değeri bilmediğiniz, bu iki değer arasındaki bilinmeyin noktadaki değeri bulmanızı sağlar. Mesela elinizde sıcaklık ile alakalı time-series bir data olduğunu düşünelim burada bir eksik veriniz varsa bu iki nokta arasındaki değeri bulmak için kullanabilirsiniz. Açı/Tork grafiği için verinin frekansını artırmak veya çözünürlük yükseltmek için kullanabilirsiniz.

Interpolasyon için basitçe bir örneğe göz atalım:

- Sıralı giden bir array'de 2 değerinin eksik olduğunu görüyorsunuz, lineer bir düzlemde 1 ve 3 sayısı arasında 2 olması gerekmektedir.

Not: Interpolasyon'u yüksek dereceli polinomlar üzerinde de kullanabilirsiniz.

[42]:
```
s = pd.Series([0, 1, np.nan, 3])

print(s)
```

```
0    0.0
1    1.0
2    NaN
3    3.0
dtype: float64
```

[43]:
```
s.interpolate() #interpolasyon ile eksik veriyi doldurduk.
```

[43]:
```
0    0.0
1    1.0
2    2.0
3    3.0
dtype: float64
```

3. En yakın komşular

Varsayılan olarak, nan_euclidean_distances yakın komşuları bulmak için eksik değerleri destekleyen bir öklid mesafesi metriği kullanılır.

Her eksik özelliği, n_neighbors sayısı kadar olan yakın komşuların değerleri kullanılarak bulunur.

Komşuların özelliklerinin her bir komşuya olan uzaklığının ağırlıklı ortalaması alınır.

```
[45]: from sklearn.impute import KNNImputer
```

```
[46]: X = [[1, 2, np.nan], [3, 4, 3], [np.nan, 6, 5], [8, 8, 7]]
      pd.DataFrame(X)
```

[46]:

	0	1	2
0	1.0	2	NaN
1	3.0	4	3.0
2	NaN	6	5.0
3	8.0	8	7.0

```
[48]: imputer = KNNImputer(n_neighbors=2, weights="uniform")
      X = imputer.fit_transform(X)
```

```
[49]: pd.DataFrame(X)
```

[49]:

	0	1	2
0	1.0	2.0	4.0
1	3.0	4.0	3.0
2	5.5	6.0	5.0
3	8.0	8.0	7.0

3. Adım: Eksikleri tamamlayın!

Gördüğünüz gibi matematiksel ve teorik işleri hallettikten sonra, domain expert'in kendi bilgisiyle ve kararlarıyla tamamlaması gereken konular kalacaktır.

Örnek olarak aşağıda Sehir kolonunda kalan bir eksiğimiz var. Burada bir karar yukarıdaki tekniklerden birini kullanmaktır. Başka bir yaklaşım olarak burada bilinmeyen şehirlere diğer yazabiliriz.

```
[50]: df_2
```

[50]:

	İsim	Soyad	Yas	Sehir
0	Mert	Cobanov	24.0	Bursa
1	Nilay	Mertal	22.0	Ankara
2	Dogancan	Mavideniz	24.0	Istanbul
3	Omer	Cengiz	23.0	NaN
4	Merve	Noyan	23.0	Izmir
5	Onur	Sahil	23.0	Istanbul

```
[55]: df_2["Sehir"] = df_2["Sehir"].replace(np.nan, "diğer")
      df_2
```

[55]:

	İsim	Soyad	Yas	Sehir
0	Mert	Cobanov	24.0	Bursa
1	Nilay	Mertal	22.0	Ankara
2	Dogancan	Mavideniz	24.0	Istanbul
3	Omer	Cengiz	23.0	diğer
4	Merve	Noyan	23.0	Izmir
5	Onur	Sahil	23.0	Istanbul

1. Standardization

Machine learning algoritmalarının büyük bir çoğunluğu iyi bir öğrenme için verinin standartlaştırılması gerekliliği duyar. Eğer veriniz Standart bir dağılım göstermiyorsa, bu modelin öğrenmesinde kötü bir performansa sebep olabilecek etkiler doğurabilir. Bu yüzden modele veriyi vermeden önce bir takım ön işlemler ile bu kötü etki ortadan kaldırılması gerekmektedir.

1.1 Standard Scaler

Standard Scaler, bir column'daki dağılımı ortalaması=0 ve standart sapması=1 olacak şekilde yeniden scale etme işlemine denir.

[57]:
```
from sklearn.preprocessing import StandardScaler
```

[59]:
```
df_ss = df_2.copy()
```

[64]:
```
df_ss["Yas_Scaled"] = StandardScaler().fit_transform(df_ss[["Yas"]])
#Yas_Scaled column'u ekledik ve bu column içine Yas columnundaki verileri
#standard scaler ile scale ederek doldurduk.
```

[74]:
```
df_ss
```

[74]:

	İsim	Soyad	Yas	Sehir	Yas_Scaled
0	Mert	Cobanov	24.0	Bursa	1.212678
1	Nilay	Mertal	22.0	Ankara	-1.697749
2	Dogancan	Mavideniz	24.0	Istanbul	1.212678
3	Omer	Cengiz	23.0	diğer	-0.242536
4	Merve	Noyan	23.0	Izmir	-0.242536
5	Onur	Sahil	23.0	Istanbul	-0.242536

[72]:
```
print(df_ss["Yas"].mean(axis=0))
print(df_ss["Yas"].std(axis=0))
```

```
23.166666666666668
0.752772652709081
```

[76]:
```
print(df_ss["Yas_Scaled"].mean(axis=0))
print(df_ss["Yas_Scaled"].std(axis=0))
```

```
-1.6930901125533637e-15
1.0954451150103321
```

1.2 MinMax Scaler

Eğer çok küçük *standard sapması* olan, küçük sayı değerleriyle çalışıyorsanız **MinMaxScaler** yararlı olacaktır.

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}$$

[77]:
```
from sklearn.preprocessing import MinMaxScaler
```

[78]:
```
df_mm = df_2.copy()
```

[79]:
```
df_mm
```

[79]:

	İsim	Soyad	Yas	Sehir
0	Mert	Cobanov	24.0	Bursa
1	Nilay	Mertal	22.0	Ankara
2	Dogancan	Mavideniz	24.0	Istanbul
3	Omer	Cengiz	23.0	diğer
4	Merve	Noyan	23.0	Izmir
5	Onur	Sahil	23.0	Istanbul

[80]:
```
df_mm["Yas_Scaled"] = MinMaxScaler().fit_transform(df_mm[["Yas"]])
df_mm
```

[80]:

	İsim	Soyad	Yas	Sehir	Yas_Scaled
0	Mert	Cobanov	24.0	Bursa	1.0
1	Nilay	Mertal	22.0	Ankara	0.0
2	Dogancan	Mavideniz	24.0	Istanbul	1.0
3	Omer	Cengiz	23.0	diğer	0.5
4	Merve	Noyan	23.0	Izmir	0.5
5	Onur	Sahil	23.0	Istanbul	0.5

Max değer 24 idi. 24 -> 1.0 oldu.

Min değer 22 idi. 22 -> 0.0 oldu.

Aradaki değerler de 0 ve 1 arasında min max'a göre dağıldı.

Not:

Verilerinizde aykırı değerler varken, scaling işlemleri çok iyi sonuçlar vermez.

Peki neden? Elinizdeki verinin 1 ile 10 arasında dağılımı olduğunu düşünelim, veri setinin içerisinde yanlış olarak yazılmış 1000 değeri sizin scaling işleminizi bozarak, verinizi 1, 10 arasındaki tüm değerleri çok küçük bir alana sıkıştıracaktır.

2. Kategorik Değerlerin Ayrıştırılması

2.1 Label Encoding

Bir kolonunuzdaki değerleri sıralı bir biçimde sayısal forma getirmek için kullanılır. Elinizde 4 adet şehir ismi olduğunu varsayalım, eğer bu değerler birçok satırda aynı isimlerle tekrarlanıyorsa, bunları sayılar ile temsil edebilirsiniz. Aşağıdaki örnekte görebileceğiniz gibi Bursa 1 sayısı ile, Ankara 0 ile, Istanbul 2 ile temsil edilecektir.

inverse_transform fonksiyonu ile geri alınabilir.

[83]:
```
from sklearn.preprocessing import LabelEncoder
```

[84]:
```
le = LabelEncoder()
```

[88]:
```
df_le = df_2.copy()
df_le
```

[88]:

	İsim	Soyad	Yas	Sehir
0	Mert	Cobanov	24.0	Bursa
1	Nilay	Mertal	22.0	Ankara
2	Dogancan	Mavideniz	24.0	Istanbul
3	Omer	Cengiz	23.0	diğer
4	Merve	Noyan	23.0	Izmir
5	Onur	Sahil	23.0	Istanbul

```
[90]: le.fit(df_le["Sehir"])
      #Sehir kolonuna label encoding islemi yapacagimi söylüyorum.
```

```
[90]: LabelEncoder()
```

```
[91]: list(le.classes_) #Sehir kolonundaki unique degerler
```

```
[91]: ['Ankara', 'Bursa', 'Istanbul', 'Izmir', 'diğer']
```

```
[92]: df_le["Sehir"] = le.transform(df_le["Sehir"])
      df_le
```

[92]:

	İsim	Soyad	Yas	Sehir
0	Mert	Cobanov	24.0	1
1	Nilay	Mertal	22.0	0
2	Dogancan	Mavideniz	24.0	2
3	Omer	Cengiz	23.0	4
4	Merve	Noyan	23.0	3
5	Onur	Sahil	23.0	2

```
[94]: # Inverse_transform fonksiyonu ile geri alınabilir
      list(le.inverse_transform([2, 1, 0, 1]))
```

```
[94]: ['Istanbul', 'Bursa', 'Ankara', 'Bursa']
```

2.2 One Hot Encoding

One Hot Encoding yöntemi bir kolon üzerindeki her bir sınıfı, o sınıfın **unique** değerleri uzunluğunda bir **vektöre** dönüştürür. Her değer bu vektör üzerindeki yerini 1 sayısını alarak belli eder, tanımı daha iyi anlamak için örneğe bakalım.

Eğer kolonda [a, b, c] değerleri varsa. a [1, 0, 0] olarak temsil edilir, keza aynı şekilde b [0, 1, 0] şeklinde temsil edilecektir.

One Hot Encoding yöntemini **Sci-kit** yerine pandasın

One Hot Encoding yöntemini **Sci-kit** yerine pandasın **get_dummies** fonksiyonu ile çok daha hızlı ve rahat bir şekilde kullanabilirsiniz.


```
[99]: pd.get_dummies(df_2["Sehir"])
```

[99]:

	Ankara	Bursa	Istanbul	Izmir	diğer
0	0	1	0	0	0
1	1	0	0	0	0
2	0	0	1	0	0
3	0	0	0	0	1
4	0	0	0	1	0
5	0	0	1	0	0

ies fonksiyonu ile çok daha hızlı ve rahat bir şekilde kullanabilirsiniz.


```
[99]: pd.get_dummies(df_2["Sehir"])
```

[99]:

	Ankara	Bursa	Istanbul	Izmir	diğer
0	0	1	0	0	0
1	1	0	0	0	0
2	0	0	1	0	0
3	0	0	0	0	1
4	0	0	0	1	0
5	0	0	1	0	0

3. Kuantizasyon veya Binning

Kuantizasyon aslına bakarsanız, haberleşme, sinyal ve elektronik derslerindeki önemli unsurlardan bir tanesidir. Bildiğiniz gibi veri genellikle iki formda bulunur. Bunlardan ilki **ayrık** *(Discrete)* ve ikincisi **sürekli** (Continuous). Bazen verinizi sınıflara ayırmak istediğinizde bu işlem çok büyük önem arz etmektedir. Sürekli bir değeri sınıflara ayırmak karar ağaçlarında veya hedefinizi sınıflandırmak istediğinizde kullanabileceğiniz bir fonksiyondur.

Burada en basit yöntem yuvarlama olabilir, sayıyı belirli sayıların katlarına basitçe yuvarlayabilirsiniz, fakat daha bilimsel bir yöntem olan K-Bins kullanılabilir.

```
[100]: X = np.array([[ -3., 5., 15 ],
                     [  0., 6., 14 ],
                     [  6., 3., 11 ]])
```

```
[101]: from sklearn import preprocessing
```

```
[102]: preprocessing.KBinsDiscretizer(n_bins=[3,2,2], encode="ordinal").fit_transform(X)
```

```
[102]: array([[0., 1., 1.],
              [1., 1., 1.],
              [2., 0., 0.]])
```

Örneği daha iyi anlamak adına her bir kolona bakabilirsiniz. **n_bins** parametresiyle kaç adet sınıfa bölmek istediğinizi seçebilirsiniz. Fonksiyon her bir kolona bakarak, n_bins sayısı kadar sınıfa bölecek ve değerlerin hangi sınıfa ait olduğunu bularak bu sayıyla temsil edecektir.

```
[106]: binarizer = preprocessing.Binarizer(threshold = 5.1) #5.1 üzerindeki tüm degerler 1 olacak.
       binarizer.transform(X)
```

```
[106]: array([[0., 0., 1.],
              [0., 1., 1.],
              [1., 0., 1.]])
```

Feature Selection

Modelinizin iyi bir performans göstermesi için boyutsallığının azaltılması ve güçlü ilişkilere sahip parametrelerin, performansı kötü etkileyecek diğer parametrelerden ayrılması gerekir. Çünkü bu öznitelikler (features) modele bir bilgi getirmiyor olabilirler.

Pekala boyut düşürmenin veya öznitelik azaltmanın yararları nedir:

- Daha yüksek doğruluk oranı
- Overfitting probleminin önüne geçmek.
- Model eğitim süresinin kısaltılması.
- Daha etkin bir görselleştirme
- Daha açıklanabilir bir model.

Veri Seti

[111]:
```python
import pandas as pd
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split

data = pd.read_csv("mushrooms.csv")
data.head()
```

[111]:

	class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	...
0	p	x	s	n	t	p	f	c	n	k	...
1	e	x	s	y	t	a	f	c	b	k	...
2	e	b	s	w	t	l	f	c	b	n	...
3	p	x	y	w	t	p	f	c	n	n	...
4	e	x	s	g	f	n	f	w	b	k	...

5 rows × 23 columns

[139]:
```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 8124 entries, 0 to 8123
Data columns (total 23 columns):
 #   Column                    Non-Null Count  Dtype
---  ------                    --------------  -----
 0   class                     8124 non-null   object
 1   cap-shape                 8124 non-null   object
 2   cap-surface               8124 non-null   object
 3   cap-color                 8124 non-null   object
 4   bruises                   8124 non-null   object
 5   odor                      8124 non-null   object
 6   gill-attachment           8124 non-null   object
 7   gill-spacing              8124 non-null   object
 8   gill-size                 8124 non-null   object
 9   gill-color                8124 non-null   object
 10  stalk-shape               8124 non-null   object
 11  stalk-root                8124 non-null   object
 12  stalk-surface-above-ring  8124 non-null   object
 13  stalk-surface-below-ring  8124 non-null   object
 14  stalk-color-above-ring    8124 non-null   object
 15  stalk-color-below-ring    8124 non-null   object
 16  veil-type                 8124 non-null   object
 17  veil-color                8124 non-null   object
 18  ring-number               8124 non-null   object
 19  ring-type                 8124 non-null   object
 20  spore-print-color         8124 non-null   object
 21  population                8124 non-null   object
 22  habitat                   8124 non-null   object
dtypes: object(23)
memory usage: 1.4+ MB
```

[112]:
```
X = data.drop(["class"], axis=1)
```

[122]:
```
y = data["class"]
```

[118]:
```
X_encoded = pd.get_dummies(X, prefix_sep="_")
```

[123]:
```
y_encoded = LabelEncoder().fit_transform(y)
```

[126]:
```
X_scaled = StandardScaler().fit_transform(X_encoded)
```

[129]:
```
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y_encoded, test_size = 0.30, random_state=101)
```

Feature Importance

Karar ağaçları çeşitli özniteliklerin önem derecelerini sıralamak için kullanılabilir. Karar ağaçlarındaki dallanma bildiğiniz gibi özniteliklerin sınıflandırıcılığıyla belirlenir. Bu yüzden daha çok kullanılan nodelar daha yüksek öneme sahip olabilirler.

[135]:
```
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.ensemble import RandomForestClassifier
import time
```

[136]:
```
start = time.process_time()

model = RandomForestClassifier(n_estimators=700).fit(X_train, y_train)

print(time.process_time() - start)
```

```
2.28125
```

[137]:
```
preds = model.predict(X_test)

print(confusion_matrix(y_test, preds))
print(classification_report(y_test, preds))
```

```
[[1274    0]
 [   0 1164]]
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      1274
           1       1.00      1.00      1.00      1164

    accuracy                           1.00      2438
   macro avg       1.00      1.00      1.00      2438
weighted avg       1.00      1.00      1.00      2438
```

Tam bir başarı oranına sahibiz fakat burada bakacağımız konu aslında hangi niteliklerin ne kadar önemli olduğu. Bu yüzden feature importance metoduyla eğitilmiş modelin en önemli olduğu 10 parametreyi görselleştiriyorum.

[138]:
```
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure

feature_imp = pd.Series(model.feature_importances_, index= X_encoded.columns)
feature_imp.nlargest(10).plot(kind='barh')
```

[138]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x1dd7066e388>
```

[141]:
```
best_feat = feature_imp.nlargest(4).index.to_list()
best_feat
```

[141]:
```
['odor_n', 'odor_f', 'gill-size_b', 'gill-size_n']
```

[142]:
```
X_reduced = X_encoded[best_feat]
```

[144]:
```
Xr_scaled = StandardScaler().fit_transform(X_reduced)
```

[145]:
```
Xr_train, Xr_test, yr_train, yr_test = train_test_split(Xr_scaled, y, test_size = 0.30,
                                                        random_state = 101)
```

[149]:
```
start = time.process_time()
rmodel = RandomForestClassifier(n_estimators=700).fit(Xr_train, yr_train)
print(time.process_time() - start)
```

```
1.34375
```

[150]:
```
rpred = rmodel.predict(Xr_test)
print(confusion_matrix(yr_test, rpred))
print(classification_report(yr_test, rpred))
```

```
[[1248   26]
 [  53 1111]]
              precision    recall  f1-score   support

           e       0.96      0.98      0.97      1274
           p       0.98      0.95      0.97      1164

    accuracy                           0.97      2438
   macro avg       0.97      0.97      0.97      2438
weighted avg       0.97      0.97      0.97      2438
```

Çok açık bir şekilde görebiliriz ki, eğitim süresi yarı yarıya inerken accuracy'den çok az kaybettik. Aslına bakarsanız bu çok küçük bir veriseti kazancımız 1 saniye kadar fakat bunu milyonlarca satıra sahip bir verisetiyle saatlerce eğittiğiniz bir model olduğunu düşünürseniz kesinlikle gireceğiniz bir tradeoff olacaktır.

Correlation Matrix

[151]:
```
import seaborn as sns

X = data.drop(['class'], axis=1)
y = data['class']
X_encoded = pd.get_dummies(X, prefix_sep="_")
y_encoded = LabelEncoder().fit_transform(y)
X_encoded["Class"] = y_encoded
```

[157]:
```
sns.heatmap(X_encoded.iloc[:, -7:].corr(), annot=True);
```

Belirttiğimiz gibi eksi ve artı değerler güçlü korelasyonu ifade ediyor, burada sayının pozitif ve negatif olması ilişkinin ters veya doğru orantılı olarak değişmesi ile alakalı, her ikisi de bizim için iyi featurelar olabilir bu yüzden dataframe'in mutlak değerini alarak en yüksek değerli olanları getireceğiz.

```
[158]: X_encoded.corr().abs()["Class"].nlargest(10)
```

```
[158]: Class                        1.000000
       odor_n                       0.785557
       odor_f                       0.623842
       stalk-surface-above-ring_k   0.587658
       stalk-surface-below-ring_k   0.573524
       ring-type_p                  0.540469
       gill-size_n                  0.540024
       gill-size_b                  0.540024
       gill-color_b                 0.538808
       bruises_f                    0.501530
       Name: Class, dtype: float64
```

Bu zamana kadar yazdığımız kısmın sonunda index metodunu ekleyerek sadece kolon isimlerini istiyorum ve bunu ana datasetimizden başka bir değişkene aktarıyorum. Birazdan sadece bu kısmı kullanıyor olacağız, bu sayede daha okunaklı ve en yüksek 10 korelasyon değerine sahip kolon ile birlikte çalışıyor olacağız.

```
[159]: X_reduced_col_names = X_encoded.corr().abs()["Class"].nlargest(10).index
       X_encoded[X_reduced_col_names].corr()
```

[159]:

	Class	odor_n	odor_f	stalk-surface-above-ring_k	stalk-surface-below-ring_k	ring-type_p	gill-size_n	gill-size_b	gill-color_b	bruises_f
Class	1.000000	-0.785557	0.623842	0.587658	0.573524	-0.540469	0.540024	-0.540024	0.538808	0.501530
odor_n	-0.785557	1.000000	-0.527269	-0.466499	-0.471920	0.352151	-0.457211	0.457211	-0.455399	-0.285171
odor_f	0.623842	-0.527269	1.000000	0.584189	0.600449	-0.427514	-0.055394	0.055394	0.079360	0.344642
stalk-surface-above-ring_k	0.587658	-0.466499	0.584189	1.000000	0.677074	-0.549484	0.095225	-0.095225	0.237814	0.541494
stalk-surface-below-ring_k	0.573524	-0.471920	0.600449	0.677074	1.000000	-0.536122	0.089569	-0.089569	0.249536	0.530549
ring-type_p	-0.540469	0.352151	-0.427514	-0.549484	-0.536122	1.000000	-0.308466	0.308466	-0.507885	-0.767036
gill-size_n	0.540024	-0.457211	-0.055394	0.095225	0.089569	-0.308466	1.000000	-1.000000	0.776903	0.369596
gill-size_b	-0.540024	0.457211	0.055394	-0.095225	-0.089569	0.308466	-1.000000	1.000000	-0.776903	-0.369596
gill-color_b	0.538808	-0.455399	0.079360	0.237814	0.249536	-0.507885	0.776903	-0.776903	1.000000	0.438292
bruises_f	0.501530	-0.285171	0.344642	0.541494	0.530549	-0.767036	0.369596	-0.369596	0.438292	1.000000

```
[165]: plt.figure(figsize=(7, 7), dpi=100)
       sns.heatmap(X_encoded[X_reduced_col_names].corr().abs(), annot=True)
```

```
[165]: <matplotlib.axes._subplots.AxesSubplot at 0x1dd79b81d08>
```

Data Preprocessing Quiz

✓ 1. Aşağıdakilerden hangisi eksik değerlerin çözümlenmesi için kullanılan bir teknik değildir? * 20/20

- ○ KNN
- ○ Ortalama
- ◉ Standart Sapma ✓
- ○ Enterpolasyon

2.

I. Standard Scaler verilen bir dizinin ortalamasını 1 ve standart sapmasını 0 yapar.
II. Dağılımı küçük bir aralıkta değişen bir seride MinMaxScaler kullanmak yararlı olabilir.
III. MinMax Scaler outlier değerlerden oldukça etkilenir.

Yukarıdaki ifadelerden hangileri doğrudur?

✓ Yukarıdaki metni temel alan ikinci sorunuzun şıkları. * 20/20

- ○ Hepsi
- ○ Yalnız I
- ◉ II ve III ✓
- ○ I ve II

3. Bir DataFrame'de 6 kolon vardır. Ilk 2 kolon hariç geri kalan 4 kolon sırasıyla "5, 3, 6 ,7" adet unique value içermektedir. Bu DataFrame'e one_hot_encoder uygulandığında ve ilk kolonların düşürüldüğü düşünülürse. DataFrame'in son halindeki kolon sayısı kaç olacaktır?

✓ Yukarıdaki metni temel alan üçüncü sorunuzun şıkları. * 20/20

- (●) 23 ✓
- ( ) 6
- ( ) 7
- ( ) 24

4. Aşağıdakilerden hangisi Feature Selection işlemlerinin sebep olacağı yararlardan olabilir?

I. Daha yüksek doğruluk oranı
II. Overfitting probleminin önüne geçmek
III. Model eğitiminin kısaltılması.
IV. Daha etkin bir görselleştirme
V. Daha açıklanabilir bir model

✓ Yukarıdaki metni temel alan dördüncü sorunuzun şıkları. * 20/20

- ○ I ve II
- ○ I, III, IV
- ○ I, III, IV, V
- ◉ Hepsi ✓

5. Korelasyon matrisi ile alakalı verilen bilgilerden hangileri doğrudur?

I. Korelasyon matrisinin diagonali her zaman "1" olmaktadır.
II. 0.3 ve üzeri korelasyon güçlü korelasyondur
III. 0.3 ve altı korelasyon zayıf korelasyondur.
IV. Korelasyon matrisini seaborn ile oluşturup, pandas ile görselleştirilir.
V. Korelasyon matrisinin değerlerini de çizdirmek için "annot" parametresi kullanılır.

✓ Yukarıdaki metni temel alan beşinci sorunuzun şıkları. * 20/20

- ○ I ve II
- ○ II, IV, V
- ◉ I, III, V ✓
- ○ Hepsi

MLD-Models

Supervised Learning

Supervised; Bize doğru cevabı verilmiş bir veri setiyle yaptığımız öğrenme çeşididir.

Regression Analysis

Regression Analysis

Bağımlı Değişken	Bağımsız Değişken
• Bu değişken tahmin etmeye çalıştığımız değişkendir	• Bu değişken tahmin etmek için kullandığımız giriş değişkenidir
• "y" olarak ifade edilir	• "X" olarak ifade edilir

** Bir regresyon probleminde sürekli aralıkta olan sonuçları tahmin etmeye çalışırız.

Classification Analysis

Spesifik sınıfları tahmin etmek için oluşturduğumuz algoritmalardır.

Binary Classification

Y değerlerimiz 2 çeşit veriden oluşuyorsa ve bunları tahmin etmeye çalışıyorsak, bu Binary Classification olur.

Multi-Class Classification

2'den fazla çeşit veriden oluşan Y değerlerimizi tahmin etmeye çalışıyorsak, bu Multi-Class Classification olur.

Linear Regression

Lineer, doğrusal bir çizgi üzerinde değişen değerlerin tahmininde kullanılır.

Linear Regression

Tek-değişkenli lineer regresyon modeli

$$y_i = \beta_0 + \beta_1 x_i + \varepsilon_i = h(x_i) + \varepsilon_i \Rightarrow \varepsilon_i = y_i - h(x_i)$$

y^i => 'i' numaralı gözlem için bağımlı değişken (ev fiyatı)

x^i => 'i' numaralı gözlem için bağımsız değişken (ev özellikleri)

ϵ^i => 'i' numaralı gözlem için hata değeri

β_0 => 'i' numaralı gözlem için sabit katsayı değeri

β_1 => 'i' numaralı gözlem için bağımsız değişkenin katsayı değeri

Linear Regression

Simple Linear Regression

- Eğimi 2 ve kesişim katsayısı -5

[4]:
```
rng = np.random.RandomState()
x = 10 * rng.rand(50) # 0-10 aralığında 50 tane random sayı

#Bu x'i kullanarak y esitsizliği oluşturuyoruz.
y = 2*x-5 + rng.randn(50)

plt.figure(dpi=100)
plt.scatter(x, y)
plt.xlabel("x", fontsize=18)
plt.ylabel("y", rotation=0, fontsize=18)

print("X\n", x, "\n")
print("Y\n", y, "\n")
```

```
X
 [8.84412574 5.00197816 0.63946137 0.49786097 6.87669746 6.66854122
 6.15523793 9.80449642 0.46943739 7.34689932 8.89055318 3.29219302
 8.52284163 9.80551651 7.71514023 0.97821967 8.2174098  4.29785632
 0.48945081 6.12842606 5.32545954 5.75755464 1.33418263 9.46120258
 9.18503326 5.22460153 1.79370667 1.86748058 3.59659877 4.55340721
 5.04836744 3.47111941 5.9217047  2.09524451 5.46713611 5.74801298
 5.01774356 7.53426023 2.30031965 8.11877541 4.24820728 4.07489343
 2.62747963 9.07586615 3.67886355 4.55250081 8.19805214 7.46386135
 6.92521581 9.69219413]

Y
 [12.0714039   7.15239144 -5.82424155 -3.84456053  8.22438289  9.18005935
  8.93959945 15.01957653 -3.37098426  9.34182134 13.78231637  0.59968943
 13.59821148 13.69711111 11.39581603 -2.50774715 12.80333392  2.78334777
 -4.78059439  6.74793861  3.72033702  5.86691548 -1.11361893 13.15325038
 13.94332308  5.05251902 -0.92357869 -2.20888839  0.76937688  3.92648493
  5.07053893  1.08251903  6.34007722  0.22548936  8.0081215   7.1757575
  4.25419236 11.12092573 -1.11012654 11.33377723  4.44955536  2.75891155
  0.16057437 11.6329059   2.428036    4.147602   12.12692786  9.24802285
  9.21478349 15.14929247]
```

Multiple Linear Regression

Multiple Linear Regression

k-değişkenli çoklu lineer regresyon modeli

$$y^i = \beta_0 + \beta_1 x_1^i + \beta_2 x_2^i + \ldots + \beta_k x_k^i + \epsilon^i$$

y^i => 'i' numaralı gözlem için bağımlı değişken (ev fiyatı)

x_j^i => 'i' numaralı gözlem için j numaralı bağımsız değişken

ϵ^i => 'i' numaralı gözlem için hata değeri

β_0 => 'i' numaralı gözlem için sabit katsayı değeri

β_j => 'j' numaralı bağımsız değişken için regresyon katsayısı

[6]:
```
from sklearn.linear_model import LinearRegression
model = LinearRegression(fit_intercept=True) #Bayes değerimizi hesaba katmak

model.fit(x[:, np.newaxis], y)

xfit = np.linspace(0, 10, 1000)
yfit = model.predict(xfit[:, np.newaxis])

plt.figure(dpi=100)
plt.scatter(x, y)
plt.xlabel('x', fontsize=18)
plt.ylabel('y', rotation=0, fontsize=18)
plt.plot(xfit, yfit);
```

Verilerin eğimi ve kesişimi modelin fit parametrelerinde bulunur.

```
[5]: print("Model eğimi:     ", model.coef_[0])
     print("Model kesişimi:", model.intercept_)

     Model eğimi:     1.9464141313879109
     Model kesişimi: -4.823220324774075
```

Polinomial Regression

Yüksek dereceden eşitsizlikler demektir. X değerimiz artarken Y değerimiz her zaman artmaz. Ters orantı ve doğru orantıyı aynı anda görebiliriz.

Polynomial Regression

```
[7]: m = 100
     X = 6 * np.random.rand(m, 1) - 3
     y = 0.5 * X**2 + X + 2 + np.random.randn(m, 1)
     plt.figure(dpi=100)
     plt.xlabel("$x_1$", fontsize=18)
     plt.ylabel('y', rotation=0, fontsize=18)
     plt.scatter(X, y)

[7]: <matplotlib.collections.PathCollection at 0x142d619ef88>
```

[8]:
```
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

polynomial_regression = Pipeline([
        ("poly_features", PolynomialFeatures(degree=2, include_bias=False)),
        ("lin_reg", LinearRegression()),
    ])

polynomial_regression.fit(X, y)
X_new=np.linspace(-3, 3, 100).reshape(100, 1)
y_newbig = polynomial_regression.predict(X_new)
plt.figure(dpi=100)
plt.scatter(X, y)
plt.xlabel("$x_1$", fontsize=18)
plt.ylabel("$y$", rotation=0, fontsize=18)
plt.plot(X_new, y_newbig, "r-", linewidth=2, label="Predictions")
plt.legend(loc="upper left", fontsize=14)
```

[8]:
```
<matplotlib.legend.Legend at 0x142d6203488>
```

Logistic Regression

Logistic Regression daha çok verdiğimiz instance'lardaki belirli sınıfların olasılığını hesaplamak için kullanılır.

Logistic Regression

[10]:
```
from sklearn import datasets
iris = datasets.load_iris()
X = iris["data"][:, 3:]  # petal width
y = (iris["target"] == 2).astype(np.int)  # 1 if Iris-Virginica, else 0
```

[11]:
```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(solver="liblinear", random_state=42)
log_reg.fit(X, y)
```

[11]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='auto', n_jobs=None, penalty='l2',
                   random_state=42, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```

[12]:
```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)

plt.figure(dpi=100)
plt.plot(X_new, y_proba[:, 1], "g-", linewidth=2, label="Iris-Virginica")
plt.plot(X_new, y_proba[:, 0], "b--", linewidth=2, label="Not Iris-Virginica")
plt.xlabel("$x$", fontsize=18)
plt.ylabel("$z$", rotation=0, fontsize=18)
plt.legend(loc="center left", fontsize=12)
```

[12]:
```
<matplotlib.legend.Legend at 0x142d49fe6c8>
```

[13]:
```
X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
decision_boundary = X_new[y_proba[:, 1] >= 0.5][0]

decision_boundary
```

[13]:
```
array([1.61561562])
```

k-Nearest Neighbor

k-Nearest Neighbor

Bir parametre vermeden algoritmamızın classification yapısı ona verdiğimiz veri setiyle değişiyor.

KNN'deki k değeri : Üzerinde durduğumuz bir noktanın çevresindeki en yakın komşularının sayısı.

Steps:

1. **Uzaklığı Hesapla**
2. **Yakın Komşuları Bul**
3. **Etiket/Sınıf için Oy Ver**

Komşulara olan uzaklığı Euclidean Distance ile hesaplarız.

k-Nearest Neighbor - Euclidean Distance

$$A = (x_1, x_2, \ldots, x_m) \qquad B = (y_1, y_2, \ldots, y_m)$$

$$dist\,(A, B) = \sqrt{\frac{\sum_{i=1}^{m} (x_i - y_i)^2}{m}}$$

k-Nearest Neighbor

[231]:
```
from sklearn import datasets

wine = datasets.load_wine()
wine_data = pd.DataFrame(wine.data, columns=wine.feature_names)
wine_data['target'] = wine['target']

wine_data.head(100)
```

[231]:

	alcohol	malic_acid	ash	alcalinity_of_ash	magnesium	total_phenols	flavanoids	nonflavanoid_phenols	proanthocyanins	color_intensity	hue	od280/od315_of_diluted_wines	proline	target
0	14.23	1.71	2.43	15.6	127.0	2.80	3.06	0.28	2.29	5.64	1.04	3.92	1065.0	0
1	13.20	1.78	2.14	11.2	100.0	2.65	2.76	0.26	1.28	4.38	1.05	3.40	1050.0	0
2	13.16	2.36	2.67	18.6	101.0	2.80	3.24	0.30	2.81	5.68	1.03	3.17	1185.0	0
3	14.37	1.95	2.50	16.8	113.0	3.85	3.49	0.24	2.18	7.80	0.86	3.45	1480.0	0
4	13.24	2.59	2.87	21.0	118.0	2.80	2.69	0.39	1.82	4.32	1.04	2.93	735.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
95	12.47	1.52	2.20	19.0	162.0	2.50	2.27	0.32	3.28	2.60	1.16	2.63	937.0	1
96	11.81	2.12	2.74	21.5	134.0	1.60	0.99	0.14	1.56	2.50	0.95	2.26	625.0	1
97	12.29	1.41	1.98	16.0	85.0	2.55	2.50	0.29	1.77	2.90	1.23	2.74	428.0	1
98	12.37	1.07	2.10	18.5	88.0	3.52	3.75	0.24	1.95	4.50	1.04	2.77	660.0	1
99	12.29	3.17	2.21	18.0	88.0	2.85	2.99	0.45	2.81	2.30	1.42	2.83	406.0	1

100 rows × 14 columns

```
[233]: print(wine.target)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2]
```

```
[232]: from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.3)
```

```
[240]: from sklearn.neighbors import KNeighborsClassifier

knn = KNeighborsClassifier(n_neighbors=7)

knn.fit(X_train, y_train)

y_pred = knn.predict(X_test)
```

```
[241]: from sklearn import metrics

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7222222222222222
```

Support Vector Machines

Support Vector Machines genelde diğer algoritmalara göre classification task'lerimizde accuracy oranı daha yüksek çıkan bir algoritmadır.

Hem classification hem regression görevlerinde kullanılabiliyor.

Classification

Yukarıdaki örneklerde lineer bir örnek var.

Support Vector Machines - Kernels

Ancak her zaman lineer olmayabiliyor.

Support Vector Machines - Kernels

SAMPLES

Name of the Kernel	Mathematical Formula
Linear	$k(x,y) = x^T.y$
Polynomial	$k(x,y) = (x^T, y)^P \, or \, k(x,y) = (x^T.y+1)^p$ where p is the polynomial degree
RBF(Gaussian)	$\phi(x) = \exp(-\frac{x^2}{2\sigma^2}), \sigma > 0$

Support Vector Machines

Classification

[298]:
```
from sklearn import datasets
from sklearn.model_selection import train_test_split

iris = datasets.load_iris()
iris_data = pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                         columns= iris['feature_names'] + ['target'])
iris_data.head()
```

[298]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

[299]:
```
X = iris.data
y = iris.target
print(y)
print(iris.target_names)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
['setosa' 'versicolor' 'virginica']
```

[289]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state = 0)
```

[308]:
```
from sklearn.svm import SVC
svm_model_linear = SVC(kernel = 'linear', C = 1).fit(X_train, y_train)
svm_predictions = svm_model_linear.predict(X_test)
```

[309]:
```
svm_predictions
```

[309]:
```
array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 2, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1,
       0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 2])
```

[310]:
```
accuracy = svm_model_linear.score(X_test, y_test)
accuracy
```

[310]:
```
0.9736842105263158
```

Regression

Epsilon değerimizi kullanarak yakınlaştırmaya çalışıyoruz.

Burada amacımız hyper plane içerisine olabildiğince fazla instance sokabilmek.

Regression

[18]:
```
dataset = pd.read_csv('Position_Salaries.csv')
X = dataset.iloc[:,1:2].values.astype(float)
y = dataset.iloc[:,2:3].values.astype(float)
dataset
```

[18]:

	Position	Level	Salary
0	Business Analyst	1	45000
1	Junior Consultant	2	50000
2	Senior Consultant	3	60000
3	Manager	4	80000
4	Country Manager	5	110000
5	Region Manager	6	150000
6	Partner	7	200000
7	Senior Partner	8	300000
8	C-level	9	500000
9	CEO	10	1000000

```
[19]: from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
sc_y = StandardScaler()
X = sc_X.fit_transform(X)
y = sc_y.fit_transform(y)
print(X, "\n")
print(y)
```

```
[[-1.5666989 ]
 [-1.21854359]
 [-0.87038828]
 [-0.52223297]
 [-0.17407766]
 [ 0.17407766]
 [ 0.52223297]
 [ 0.87038828]
 [ 1.21854359]
 [ 1.5666989 ]]

[[-0.72004253]
 [-0.70243757]
 [-0.66722767]
 [-0.59680786]
 [-0.49117815]
 [-0.35033854]
 [-0.17428902]
 [ 0.17781001]
 [ 0.88200808]
 [ 2.64250325]]
```

```
[20]: from sklearn.svm import SVR

regressor = SVR(kernel='rbf')
regressor.fit(X,y)
```

```
F:\Anaconda3\lib\site-packages\sklearn\utils\validation.py:760: DataConversionWar
  y = column_or_1d(y, warn=True)
```

```
[20]: SVR(C=1.0, cache_size=200, coef0=0.0, degree=3, epsilon=0.1, gamma='scale',
    kernel='rbf', max_iter=-1, shrinking=True, tol=0.001, verbose=False)
```

RBF’de Gama parametremiz bize iki nokta arasındaki benzerliğin oranını hesap eder. 0-1 arasındadır.

Daha yüksek Gama değeri verdiğimizde modelimiz datasetimize çok iyi bir şekilde fit olabiliyor. Bu da bazen Overfit olmasına sebep olabiliyor.

[21]:
```
y_pred = regressor.predict([[6.5]])
y_pred
```

[21]:
```
array([0.01158103])
```

[22]:
```
plt.figure(dpi=100)
plt.scatter(X, y, color = 'blue')
plt.plot(X, regressor.predict(X), color = 'green')
plt.title('Support Vector Machines Regression Model')
plt.xlabel('Position level')
plt.ylabel('Salary')
plt.show()
```

Desicion Trees

Classification

Decision Trees - Classification

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0.0
1	4.9	3.0	1.4	0.2	0.0
2	4.7	3.2	1.3	0.2	0.0
3	4.6	3.1	1.5	0.2	0.0
4	5.0	3.6	1.4	0.2	0.0

```
array(['setosa', 'versicolor', 'virginica']
```

Decision Trees - Classification

Gini Impurity

***Impurity: Bir training set verisine etiket verirken o etiketin yanlış olma olma şansı (Hiç hata => Impurity = 0)**

Gini Impurity, verilen bir değerin impurity(yanlış olma oranı) değerini bulur.

$$I_G(n) = 1 - \sum_{i=1}^{J} (p_i)^2$$

Decision Trees - Classification

Gini Impurity

$$I_G(n) = 1 - \sum_{i=1}^{J} (p_i)^2$$

Derinlik-2 Sol Nod:

1 - (0/54)^2 - (49/54)^2 - (5/54)^2 ~= 0.168

Decision Trees - Classification

Entropy (Information Gain: sorulacak en iyi soruyu bulmakta fayda sağlar)

***Entropy: Makine öğrenmesinde entropy bir verinin sadece tek class değeri almasıyla 0 değerini alır.**

$$I_H = -\sum_{j=1}^{c} p_j log_2(p_j)$$

Decision Trees - Classification

Entropy

$$I_H = -\sum_{j=1}^{c} p_j log_2(p_j)$$

Derinlik-2 Sol Nod:

-(49/54)log(49/54) - 5/54log(5/54) ~= 0.44

Cart

Decision Trees - CART

Classification and Regression Tree - CART

CART Cost Function for Classification

$$J(k, t_k) = \frac{m_{\text{left}}}{m} G_{\text{left}} + \frac{m_{\text{right}}}{m} G_{\text{right}}$$

Decision Trees

Decision Tree Classifiers

[105]:
```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

[106]:
```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
pima = pd.read_csv("pima-indians-diabetes.csv")
pima.columns = col_names
pima.head()
```

[106]:

	pregnant	glucose	bp	skin	insulin	bmi	pedigree	age	label
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

[107]:
```
feature_cols = ['pregnant', 'insulin', 'bmi', 'age','glucose','bp','pedigree']
X = pima[feature_cols] # Features
y = pima.label # Target variable
```

[108]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test
```

[109]:
```
clf = DecisionTreeClassifier()

clf = clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)
```

[110]:
```
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.670995670995671
```

[111]:
```
from sklearn.tree import export_graphviz
from sklearn.externals.six import StringIO
from IPython.display import Image
import pydotplus

dot_data = StringIO()
export_graphviz(clf, out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,feature_names = feature_cols,class_names=['0','1'])
graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes_1.png')
Image(graph.create_png())
```

[113]:
```
clf = DecisionTreeClassifier(criterion="entropy", max_depth=3)

clf = clf.fit(X_train,y_train)

y_pred = clf.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.7705627705627706
```

[83]:
```
from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clf,
                out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,
                feature_names = feature_cols,class_names=['0','1']
               )

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('diabetes_2.png')
Image(graph.create_png())
```

Regression

Decision Trees - Regression

CART Cost Function for Regression

$$J(k, t_k) = \frac{m_{\text{left}}}{m}\text{MSE}_{\text{left}} + \frac{m_{\text{right}}}{m}\text{MSE}_{\text{right}}$$

Decision Tree Regressors

[114]:
```
dataset = np.array(
[['Asset Flip', 100, 1000],
['Text Based', 500, 3000],
['Visual Novel', 1500, 5000],
['2D Pixel Art', 3500, 8000],
['2D Vector Art', 5000, 6500],
['Strategy', 6000, 7000],
['First Person Shooter', 8000, 15000],
['Simulator', 9500, 20000],
['Racing', 12000, 21000],
['RPG', 14000, 25000],
['Sandbox', 15500, 27000],
['Open-World', 16500, 30000],
['MMOFPS', 25000, 52000],
['MMORPG', 30000, 80000]
])

dataset
```

[114]:
```
array([['Asset Flip', '100', '1000'],
       ['Text Based', '500', '3000'],
       ['Visual Novel', '1500', '5000'],
       ['2D Pixel Art', '3500', '8000'],
       ['2D Vector Art', '5000', '6500'],
       ['Strategy', '6000', '7000'],
       ['First Person Shooter', '8000', '15000'],
       ['Simulator', '9500', '20000'],
       ['Racing', '12000', '21000'],
       ['RPG', '14000', '25000'],
       ['Sandbox', '15500', '27000'],
       ['Open-World', '16500', '30000'],
       ['MMOFPS', '25000', '52000'],
       ['MMORPG', '30000', '80000']], dtype='<U20')
```

[115]:
```
X = dataset[:,1:2].astype(int)
X
```

[115]:
```
array([[  100],
       [  500],
       [ 1500],
       [ 3500],
       [ 5000],
       [ 6000],
       [ 8000],
       [ 9500],
       [12000],
       [14000],
       [15500],
       [16500],
       [25000],
       [30000]])
```

[116]:
```
y = dataset[:,2].astype(int)
y
```

[116]:
```
array([ 1000,  3000,  5000,  8000,  6500,  7000, 15000, 20000, 21000,
       25000, 27000, 30000, 52000, 80000])
```

[117]:
```
from sklearn.tree import DecisionTreeRegressor

regressor = DecisionTreeRegressor(random_state = 0)

regressor.fit(X, y)
```

[117]:
```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=0, splitter='best')
```

[118]:
```
y_pred = regressor.predict([[3750]])

print("Predicted price: % d\n"% y_pred)
```

```
Predicted price:  8000
```

[126]:
```
dot_data = StringIO()
export_graphviz(regressor,
                out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,
                feature_names = ['Production Cost']
               )

graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
graph.write_png('regression_tree.png')
Image(graph.create_png())
```

Ensemble Methods & Random Forest

Random Forest

Bagging and Pasting

Bagging : Bootstrap Aggregating

Verisetini sampling yöntemiyle classifier'lara ayırıyor. Sürekli verisetini classifier'larda sampling edip sürekli değiştirir.

Pasting : Tek bir classifier için sürekli aynı verisetini sample olarak distribute eder.

Bagging ile daha yüksek accuracy elde etme şansına sahibiz.

Random Forest

Bagging and Pasting

Models Quiz

Time Driving (Hours)	Total Distance (Miles)
0	0
1	55
2	120
3	188
4	252
5	307
6	366

$$y = 61.93x - 1.79$$

✓ 1)Bir linear regression denkliği y = 61.93x - 1.79 olarak verilmiştir. Eğimi kaçtır? * 5/5

- ○ (0,-1.79)
- ○ -1.79
- ◉ 61.93 ✓
- ○ (0,61.93)

✓ 2) β0 ve β1 olarak iki farklı değer için iki farklı logistic model grafikte gösterilmiştir. β0 ve β1 için aşağıdakilerden hangisi doğrudur? (β0: yesil, β1: siyah, Y = β0 + β1*X) * 10/10

- ○ B) β1 değeri her iki model için de aynıdır.
- ◉ C) Yeşil için olan β1 değeri siyahınkinden küçüktür. ✓
- ○ D) Hiçbiri doğru değildir.
- ○ A) Yeşil için olan β1 değeri siyahınkinden büyüktür

✕ 3) Aşağıdakilerden hangisi k-Nearest Neighbor için doğrudur? * 0/10

- (●) Sadece classification için kullanılır. ✕
- ( ) Sadece regression için kullanılır.
- ( ) Hem classification hem de regression için kullanılır.
- ( ) Hepsi yanlıştır.

Doğru cevap

- (●) Hem classification hem de regression için kullanılır.

4) Aşağıdakilerden hangisi A(1,3) ve B(2,3) noktaları arasındaki Euclidean Distance değeridir? * 10/10

- (●) 1 ✓
- ( ) 2
- ( ) 4
- ( ) 8

5) SVM modelinizi RBF kernel ve yüksek gamma değeriyle eğittiğinizde aşağıdakilerden hangisi beklenebilir? * 20/20

- ( ) Model hyperplane'e çok uzak noktalardaki değerleri modelleme işlemine dahil edebilir.
- (●) Model sadece hyperplane'e çok yakın mesafadeki değerleri modelleme işlemine dahil edebilir. ✓
- ( ) Model veri noktalarının hyperplane'e olan uzaklığından etkilenmez.
- ( ) Yukarıdakilerden hiçbiri

6) Aşağıdakilerden hangisi SVM modelin uygulama alanlarındandır? * 15/15

- ( ) Text Kategorileme
- ( ) Görsel Veri Sınıflandırma
- ( ) Yazılı haber verilerinin kümelendirilmesi
- (●) Yukarıdakilerin hepsi ✓

7) Aşağıdaki error metric hesaplama yöntemlerinden hangisinin {0, 1} gibi bir sınıflandırma görevi uygulandığı zaman kullanılması uygun olur? * 0/15

- Worst-case error
- Sum of squares error
- Entropy
- Precision and Recall

Doğru cevap

- Entropy

8)

I. Bagging ağaçlarında bireysel ağaçlar birbirinden bağımsızdır.
II. Bagging ensemble model içerisindeki learner modüllerinin tahmin sonuçlarının aggregate(toplanma) yöntemiyle performanının artırılması için kullanılan bir yöntemdir.

8)Aşağıdakilerden hangisi ya da hangileri bagging ile ilgili doğrudur? * 15/15

- Yalnız I
- Yalnız II
- I ve II
- Hiçbiri

---Kaggle Master---

Bu bölümde Global Ai Hub mentorlüğünde düzenlenen Kaggle Master etkinliğinin notlarına ulaşabilirsiniz.

Intro to Machine Learning

Makine öğrenmesindeki temel fikirleri öğrenin ve ilk modellerinizi oluşturun.

How Models Work (Modeller Nasıl Çalışır?)

Giriş

Makine öğrenimi modellerinin nasıl çalıştığına ve nasıl kullanıldıklarına genel bir bakışla başlayacağız. Daha önce istatistiksel modelleme veya makine öğrenimi yaptıysanız bu temel görünebilir. Endişelenmeyin, yakında güçlü modeller oluşturmaya devam edeceğiz.

Bu mikro kurs, aşağıdaki senaryodan geçerken modeller oluşturmanızı sağlayacaktır:

Kuzeniniz gayrimenkul konusunda spekülasyonlarla milyonlarca dolar kazandı. Veri bilimine gösterdiğiniz ilgi nedeniyle sizinle iş ortağı olmayı teklif etti. Parayı tedarik edecek ve çeşitli evlerin ne kadar değerli olduğunu tahmin eden modeller sunacaksınız.

Kuzeninize geçmişte gayrimenkul değerlerini nasıl tahmin ettiğini soruyorsunuz. Ve bunun sadece sezgi olduğunu söylüyor. Ancak daha fazla sorgulama, geçmişte gördüğü evlerden fiyat örüntülerini belirlediğini ve bu kalıpları düşündüğü yeni evler için tahminler yapmak için kullandığını ortaya koyuyor.

Makine öğrenimi de aynı şekilde çalışır. Decision Tree adlı bir modelle başlayacağız. Daha doğru tahminler veren meraklı modeller var. Ancak Decision Tree'lerin anlaşılması kolaydır ve bunlar veri bilimindeki en iyi modellerin bazıları için temel yapı taşıdır.

Basitlik için, mümkün olan en basit karar ağacıyla başlayacağız.

Evleri sadece iki kategoriye ayırır. Dikkate alınan herhangi bir ev için tahmini fiyat, aynı kategorideki evlerin tarihsel ortalama fiyatıdır.

Verileri, evlerin iki gruba nasıl ayrılacağına karar vermek için ve sonra her grupta öngörülen fiyatı belirlemek için kullanıyoruz. Verilerden pattern yakalamanın bu adımına, modelin fit edilmesi(**fitting**) veya train edilmesi(**training**) denir. Modelin **fit** edilmesi için kullanılan verilere **training data** denir.

Modelin nasıl **fit** edildiğine dair ayrıntılar (örneğin, verilerin nasıl bölüneceği) daha sonra kullanmak üzere kayıt edeceğimiz kadar karmaşıktır. Model **fit** edildikten sonra, yeni evlerin fiyatlarını **predict** edebilmek için yeni verilere uygulayabilirsiniz.

Decision Tree'nin Geliştirilmesi

Aşağıdaki iki karardan hangisinin gayrimenkul eğitim verilerinin fit edilmesinden kaynaklanması daha olasıdır?

Soldaki karar ağacı (Decision Tree 1) muhtemelen daha mantıklıdır, çünkü daha fazla yatak odası olan evlerin daha az yatak odası olan evlerden daha yüksek fiyatlarla satılma eğiliminde olduğu gerçeğini yakalar.

Bu modelin en büyük eksikliği, banyo sayısı, lot büyüklüğü, konum vb. gibi ev fiyatını etkileyen çoğu faktörü yakalamamasıdır.

Daha fazla "splits(bölme)" olan bir ağaç kullanarak daha fazla faktör yakalayabilirsiniz.

Bunlara "deeper(daha derin)" ağaçlar denir.

Her evin toplam lot büyüklüğünü de dikkate alan bir karar ağacı şöyle görünebilir:

Herhangi bir evin fiyatını karar ağacından takip ederek, her zaman o evin özelliklerine karşılık gelen yolu seçerek tahmin edersiniz.

Ev için tahmini fiyat ağacın altındadır.

Altta tahmin yaptığımız noktaya **leaf**(yaprak) denir.

Yapraklardaki splits(bölünmeler) ve values(değerler) veriler tarafından belirlenecektir, bu nedenle çalışacağınız verileri kontrol etmenin zamanı geldi.

Basic Data Exploration (Basit Veri Keşfi)

Verilerinizi Tanımak için Pandas Kullanımı

Herhangi bir makine öğrenimi projesinin ilk adımı, verileri tanımaktır.

Bunun için Pandas kütüphanesini kullanacaksınız.

Pandas, bilim insanlarının verileri keşfetmek ve işlemek için kullandığı temel araç verisidir.

Çoğu kişi kodlarında pandas'ı **pd** olarak kısaltır. Bunu şu komutla yapıyoruz:

```
In [1]:
        import pandas as pd
```

Pandas kütüphanesinin en önemli kısmı DataFrame'dir.

Bir DataFrame, tablo olarak düşünebileceğiniz veri türünü tutar. Bu, Excel'deki bir sayfaya veya SQL veritabanındaki bir tabloya benzer.

Pandas, bu tür verilerle yapmak isteyeceğiniz birçok şey için güçlü yöntemlere sahiptir.

Örnek olarak, Avustralya, Melbourne'daki ev fiyatları hakkındaki verilere bakacağız. (https://www.kaggle.com/dansbecker/melbourne-housing-snapshot)

Uygulamalı alıştırmalarda, aynı işlemleri Iowa'da ev fiyatları olan yeni bir veri kümesine uygulayacaksınız.

Örnek (Melbourne) verileri ../input/melbourne-housing-snapshot/melb_data.csv dosya yolundadır.

Verileri aşağıdaki komutlarla yükler ve keşfederiz:

```
In [2]:
# save filepath to variable for easier access
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
# read the data and store data in DataFrame titled melbourne_data
melbourne_data = pd.read_csv(melbourne_file_path)
# print a summary of the data in Melbourne data
melbourne_data.describe()
```

Out[2]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13518.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242	1.610075
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712	0.962634
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000	10.000000

Out[2]:

om	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longtitude	Propertycount
.000000	13518.000000	13580.000000	7130.000000	8205.000000	13580.000000	13580.000000	13580.000000
242	1.610075	558.416127	151.967650	1964.684217	-37.809203	144.995216	7454.417378
712	0.962634	3990.669241	541.014538	37.273762	0.079260	0.103916	4378.581772
000	0.000000	0.000000	0.000000	1196.000000	-38.182550	144.431810	249.000000
000	1.000000	177.000000	93.000000	1940.000000	-37.856822	144.929600	4380.000000
000	2.000000	440.000000	126.000000	1970.000000	-37.802355	145.000100	6555.000000
000	2.000000	651.000000	174.000000	1999.000000	-37.756400	145.058305	10331.000000
000	10.000000	433014.000000	44515.000000	2018.000000	-37.408530	145.526350	21650.000000

Interpreting Data Description (Verilerin Yorumlanması)

Sonuçlar, orijinal veri kümenizdeki her column(sütun) için 8 sayı gösterir.

İlk sayı, **count**, kaç satırın eksik olmayan değerleri olduğunu gösterir.

Eksik değerler birçok nedenden dolayı ortaya çıkar.

Örneğin, 1 yatak odalı bir ev araştırılırken 2. yatak odasının boyutu toplanmaz.

Eksik veriler konusuna geri döneceğiz.

İkinci değer, **mean** olan ortalamadır.

Bunun altında **std**, değerlerin sayısal olarak ne kadar yayıldığını ölçen standart sapmadır.

Min, **% 25, % 50, % 75 ve max** değerlerini yorumlamak için, her sütunu en düşükten en yüksek değere doğru sıraladığınızı düşünün.

İlk (en küçük) değer min.

Listeyi dörde bölün, dörde bölünen bölümlerden ilkinin son elemanına bakın.

Örneğin, 200 elemanlık listeyi dörde bölünce, ilk bölümün son elemanı 50 olur.

Bu **% 25** değeridir ("25. percentile" olarak telaffuz edilir). 50. ve 75. yüzdelikler benzer şekilde tanımlanır ve **max** en büyük sayıdır.

Excercise: Explore Your Data

Bu alıştırma, bir veri dosyasını okuma ve verilerle ilgili istatistikleri anlama yeteneğinizi test edecektir.

Daha sonraki alıştırmalarda, verileri filtrelemek, bir makine öğrenme modeli oluşturmak ve modelinizi yinelemeli olarak geliştirmek için teknikler uygulayacaksınız.

Kurs örnekleri Melbourne'den gelen verileri kullanır. Bu teknikleri kendi başınıza uygulayabilmeniz için, bunları yeni bir veri kümesine (Iowa'dan konut fiyatları) uygulamanız gerekecektir.

Step 1: Loading Data (Veri Yükleme)

Iowa veri dosyasını home_data adlı bir Pandas DataFrame'de okuyun.

```
import pandas as pd

# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'

# Fill in the line below to read the file into a variable home_data
home_data = pd.read_csv(iowa_file_path)

# Call line below with no argument to check that you've loaded the data correctly
step_1.check()
```

Correct

Step 2: Review The Data (Verileri Gözden Geçirme)

Verilerin özet istatistiklerini görüntülemek için öğrendiğiniz komutu kullanın. Ardından aşağıdaki soruları cevaplamak için değişkenleri doldurun

```python
# Print summary statistics in next line
home_data.describe()
```

Out[3]:

	Id	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	...
count	1460.000000	1460.000000	1201.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1452.000000	1460.000000	...
mean	730.500000	56.897260	70.049958	10516.828082	6.099315	5.575342	1971.267808	1984.865753	103.685262	443.639726	...
std	421.610009	42.300571	24.284752	9981.264932	1.382997	1.112799	30.202904	20.645407	181.066207	456.098091	...
min	1.000000	20.000000	21.000000	1300.000000	1.000000	1.000000	1872.000000	1950.000000	0.000000	0.000000	...
25%	365.750000	20.000000	59.000000	7553.500000	5.000000	5.000000	1954.000000	1967.000000	0.000000	0.000000	...
50%	730.500000	50.000000	69.000000	9478.500000	6.000000	5.000000	1973.000000	1994.000000	0.000000	383.500000	...
75%	1095.250000	70.000000	80.000000	11601.500000	7.000000	6.000000	2000.000000	2004.000000	166.000000	712.250000	...
max	1460.000000	190.000000	313.000000	215245.000000	10.000000	9.000000	2010.000000	2010.000000	1600.000000	5644.000000	...

8 rows × 38 columns

...	WoodDeckSF	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea	MiscVal	MoSold	YrSold	SalePrice
...	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000	1460.000000
...	94.244521	46.660274	21.954110	3.409589	15.060959	2.758904	43.489041	6.321918	2007.815753	180921.195890
...	125.338794	66.256028	61.119149	29.317331	55.757415	40.177307	496.123024	2.703626	1.328095	79442.502883
...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	2006.000000	34900.000000
...	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.000000	2007.000000	129975.000000
...	0.000000	25.000000	0.000000	0.000000	0.000000	0.000000	0.000000	6.000000	2008.000000	163000.000000
...	168.000000	68.000000	0.000000	0.000000	0.000000	0.000000	0.000000	8.000000	2009.000000	214000.000000
...	857.000000	547.000000	552.000000	508.000000	480.000000	738.000000	15500.000000	12.000000	2010.000000	755000.000000

[10]:

```python
# What is the average lot size (rounded to nearest integer)?
avg_lot_size = 10517

# As of today, how old is the newest home (current year - the date in which it was built)
newest_home_age = 10

# Checks your answers
step_2.check()
```

Correct

Verilerinizi Düşünün

Verilerinizdeki en yeni ev o kadar da yeni değil. Bunun için birkaç farklı durum olabilir:

1- Bu verilerin toplandığı yeni evler inşa etmediler.

2- Veriler uzun zaman önce toplanmıştır. Veri toplandıktan sonra inşa edilen evler görünmüyordur.

Nedeni yukarıdaki 1. açıklama ise, bu, bu verilerle oluşturduğunuz modele olan güveninizi etkiler mi? 2. açıklama ise ne olur?

Hangi açıklamanın daha mantıklı olduğunu görmek için verileri nasıl inceleyebilirsiniz?

Your First Machine Learning Model

Selecting Data for Modeling (Modelleme için Veri Seçmek)

Veri kümenizin, kafanızda canlanması veya güzelce ekrana yazdırmak için çok fazla değişkeni vardı. Bu başa çıkılamaz veri miktarını anlayabileceğiniz bir şeye nasıl ayırabilirsiniz?

Sezgimizi kullanarak birkaç değişken seçerek başlayacağız. Daha sonraki kurslar, değişkenleri otomatik olarak önceliklendirmek için istatistiksel teknikleri gösterecektir.

Değişkenleri / sütunları seçmek için veri kümesindeki tüm sütunların bir listesini görmemiz gerekir. Bu, DataFrame'in **columns** özelliği ile yapılır. (Aşağıdaki kodun alt satırı.)

In [1]:
```
import pandas as pd

melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
melbourne_data.columns
```

Out[1]:
```
Index(['Suburb', 'Address', 'Rooms', 'Type', 'Price', 'Method', 'SellerG',
       'Date', 'Distance', 'Postcode', 'Bedroom2', 'Bathroom', 'Car',
       'Landsize', 'BuildingArea', 'YearBuilt', 'CouncilArea', 'Lattitude',
       'Longtitude', 'Regionname', 'Propertycount'],
      dtype='object')
```

Melbourne verilerinin bazı eksik değerleri vardır (bazı değişkenlerin kaydedilmediği bazı evler.)

Daha sonraki bir derste eksik değerleri ele almayı öğreneceğiz.

Iowa verileriniz, kullandığınız sütunlarda eksik değerlere sahip değildi.

Şimdilik en basit seçeneği alacağız ve verilerimizden eksik değere sahip evleri düşüreceğiz.

dropna eksik değerleri düşürüyor (na'yı "mevcut değil" olarak düşünün)

In [2]:
```
# The Melbourne data has some missing values (some houses for which some variables weren't record
ed.)
# We'll learn to handle missing values in a later tutorial.
# Your Iowa data doesn't have missing values in the columns you use.
# So we will take the simplest option for now, and drop houses from our data.
# Don't worry about this much for now, though the code is:

# dropna drops missing values (think of na as "not available")
melbourne_data = melbourne_data.dropna(axis=0)
```

Verilerinizin bir alt kümesini seçmenin birçok yolu vardır. Pandas Micro-Course (https://www.kaggle.com/learn/pandas) bunları daha derinlemesine ele alıyor, ancak şimdilik iki yaklaşıma odaklanacağız.

1 "Prediction Target(Tahmin hedefi)"'ni seçmek için kullandığımız nokta gösterimi(dot notation)
2 "Features(Özellikleri)" seçmek için kullandığımız bir sütun listesiyle seçim yapma

Selecting The Prediction Target (Tahmin Hedefini Seçme)

dot-notation ile bir değişkeni(column) veri setinden çekebilirsiniz. Bu tek sütun, genel olarak yalnızca tek bir column'a sahip DataFrame benzeri bir **Seri**de depolanır.

Tahmin etmek istediğimiz column'u seçmek için dot-notation kullanacağız, buna **prediction target** (tahmin hedefi) denir.

Kural olarak, prediction target (tahmin hedefi) **y** olarak adlandırılır.

Melbourne'deki ev fiyatlarını (price) kaydetmek için gereken kod.

In [3]:
```
y = melbourne_data.Price
```

Choosing "Features" (Özellik Seçimi)

Modelimize girilen sütunlara (ve daha sonra tahminlerde kullanılan sütunlara) "features (özellikler)" denir.

Bizim durumumuzda, bunlar ev fiyatını belirlemek için kullanılan sütunlar olacaktır.

Bazen, target(hedef) hariç tüm sütunları feature(özellik) olarak kullanırsınız. Diğer zamanlarda daha az özellik ile daha iyi olacaksınız.

Şimdilik, sadece birkaç özelliğe sahip bir model oluşturacağız.

Daha sonra, farklı özelliklerle oluşturulan modellerin nasıl tekrarlanacağını ve karşılaştırılacağını göreceksiniz.

Köşeli parantez içine sütun adlarının listesini yazarak birden fazla özellik seçiyoruz. Bu listedeki her öğe bir string (tırnak işaretli) olmalıdır.

Here is an example:

In [4]:
```
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'Lattitude', 'Longtitude']
```

Kural olarak, bu verilere X denir.

In [5]:

```
X = melbourne_data[melbourne_features]
```

En üstteki birkaç satırı gösteren **head** yöntemini ve **describe** yöntemini kullanarak konut fiyatlarını tahmin etmek için kullanacağımız verileri hızlı bir şekilde inceleyelim.

In [6]:

```
X.describe()
```

Out[6]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
count	6196.000000	6196.000000	6196.000000	6196.000000	6196.000000
mean	2.931407	1.576340	471.006940	-37.807904	144.990201
std	0.971079	0.711362	897.449881	0.075850	0.099165
min	1.000000	1.000000	0.000000	-38.164920	144.542370
25%	2.000000	1.000000	152.000000	-37.855438	144.926198
50%	3.000000	1.000000	373.000000	-37.802250	144.995800
75%	4.000000	2.000000	628.000000	-37.758200	145.052700
max	8.000000	8.000000	37000.000000	-37.457090	145.526350

In [7]:

```
X.head()
```

Out[7]:

	Rooms	Bathroom	Landsize	Lattitude	Longtitude
1	2	1.0	156.0	-37.8079	144.9934
2	3	2.0	134.0	-37.8093	144.9944
4	4	1.0	120.0	-37.8072	144.9941
6	3	2.0	245.0	-37.8024	144.9993
7	2	1.0	256.0	-37.8060	144.9954

Verilerinizi bu komutlarla görsel olarak kontrol etmek, bir veri bilim insanının işinin önemli bir parçasıdır. Veri kümesinde sıklıkla daha fazla incelemeyi hak eden sürprizler bulacaksınız.

Building Your Model (Model Oluşturma)

Modellerinizi oluşturmak için **scikit-learn** kütüphanesini kullanacaksınız.

Kodlama yaparken, bu kütüphane örnek kodda göreceğiniz gibi **sklearn** olarak yazılır.

Scikit-learn, tipik olarak DataFrames'da depolanan veri türlerini modellemek için en popüler kütüphanedir.

Bir model oluşturma ve kullanma adımları:

- **define :** Ne tür bir model olacak? Karar ağacı mı? Başka bir model mi? Model tipinin diğer bazı parametreleri de belirtilir.
- **fit :** Sağlanan verilerden pattern(desen) yakalayın. Bu modellemenin kalbidir.
- **predict :** Tahmin
- **evaluate :** Modelin tahminlerinin ne kadar doğru olduğu belirleyin.

İşte **scikit-learn** ile bir **Decision Tree**(Karar Ağaçları) modelini tanımlama ve modeli feature'lara ve target değişkene **fit** etme örneği.

- Modeli tanımlayın. Her çalıştırmada aynı sonuçları sağlamak için random_state için bir sayı belirtin

In [8]:

```
from sklearn.tree import DecisionTreeRegressor

# Define model. Specify a number for random_state to ensure same results each run
melbourne_model = DecisionTreeRegressor(random_state=1)

# Fit model
melbourne_model.fit(X, y)
```

Out[8]:

```
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=1, splitter='best')
```

random_state: Kodu her çalıştırdığımızda aynı çıktıyı alabilmek için girdiğimiz bir ifade. Örneğin, validation ve training olarak datayı ayırırken Python her seferinde datayı farklı yerlerinden böler, bir random state değeri belirlediğimizde de her çalıştırdığımızda aynı şekilde bölmüş olur ve aynı sonucu vermiş olur. Farklı değerler verdiğinde farklı sonuçlar aldığını göreceksin.

En iyi karar ağacını bulma problemi NP-Complete olarak sınıflandırılan problemlerdendir. Bu tip problemlerin çözümlerinde sezgisel algoritmalar kullanılır. Sezgisel algoritmalarda her kullanıldıklarında en iyi çözümü bulabileceklerini garanti etmezler ve her seferinde farklı sonuçlar üretirler. Dolayısıyla her ağaç inşa ettiğinde ağaç yapısı değişiklik gösterecektir. Modeli her çalıştırdığında aynı ağacı elde etmek istersen **random_state** parametresini bir tamsayıya eşitlemen gerekir. Hangi tamsayıya eşitlediğinin bir önemi yok .

Birçok makine öğrenimi modeli, model eğitiminde bazı rasgeleliklere izin verir.

Random_state için bir sayı belirtmek, her çalıştırmada aynı sonuçları almanızı sağlar. Bu iyi bir uygulama olarak kabul edilir.

Herhangi bir sayı kullanabilirsiniz ve model kalitesi tam olarak hangi değeri seçtiğinize bağlı olmayacaktır.

Şimdi tahminler yapmak için kullanabileceğimiz uygun bir modelimiz var.

Uygulamada, halihazırda fiyatlarımız olan evler yerine piyasaya çıkan yeni evler için tahminler yapmak isteyeceksiniz.

Ancak, tahmin işlevinin nasıl çalıştığını görmek için egzersiz verilerinin ilk birkaç satırı için tahminler yapacağız.

In [9]:

```
print("Making predictions for the following 5 houses:")
print(X.head())
print("The predictions are")
print(melbourne_model.predict(X.head()))
```

```
Making predictions for the following 5 houses:
   Rooms  Bathroom  Landsize  Lattitude  Longtitude
1      2       1.0     156.0   -37.8079    144.9934
2      3       2.0     134.0   -37.8093    144.9944
4      4       1.0     120.0   -37.8072    144.9941
6      3       2.0     245.0   -37.8024    144.9993
7      2       1.0     256.0   -37.8060    144.9954
The predictions are
[1035000. 1465000. 1600000. 1876000. 1636000.]
```

Exercise: Your First Machine Learning Model

Özet

Şimdiye kadar, verilerinizi yüklediniz ve aşağıdaki kodla incelediniz. Önceki adımı bıraktığınız yerde kodlama ortamınızı ayarlamak için bu hücreyi çalıştırın.

```
# Code you have previously used to load data
import pandas as pd

# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'

home_data = pd.read_csv(iowa_file_path)

# Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex3 import *

print("Setup Complete")
```

```
Setup Complete
```

Exercises

Step 1: Prediction Target Belirleme

Satış fiyatına karşılık gelen hedef değişkeni seçin. Bunu y adlı yeni bir değişkene kaydedin. İhtiyacınız olan sütunun adını bulmak için sütunların bir listesini yazdırmanız gerekir.

[8]:
```
# print the list of columns in the dataset to find the name of the prediction target
home_data.columns
```

Out[8]:
```
Index(['Id', 'MSSubClass', 'MSZoning', 'LotFrontage', 'LotArea', 'Street',
       'Alley', 'LotShape', 'LandContour', 'Utilities', 'LotConfig',
       'LandSlope', 'Neighborhood', 'Condition1', 'Condition2', 'BldgType',
       'HouseStyle', 'OverallQual', 'OverallCond', 'YearBuilt', 'YearRemodAdd',
       'RoofStyle', 'RoofMatl', 'Exterior1st', 'Exterior2nd', 'MasVnrType',
       'MasVnrArea', 'ExterQual', 'ExterCond', 'Foundation', 'BsmtQual',
       'BsmtCond', 'BsmtExposure', 'BsmtFinType1', 'BsmtFinSF1',
       'BsmtFinType2', 'BsmtFinSF2', 'BsmtUnfSF', 'TotalBsmtSF', 'Heating',
       'HeatingQC', 'CentralAir', 'Electrical', '1stFlrSF', '2ndFlrSF',
       'LowQualFinSF', 'GrLivArea', 'BsmtFullBath', 'BsmtHalfBath', 'FullBath',
       'HalfBath', 'BedroomAbvGr', 'KitchenAbvGr', 'KitchenQual',
       'TotRmsAbvGrd', 'Functional', 'Fireplaces', 'FireplaceQu', 'GarageType',
       'GarageYrBlt', 'GarageFinish', 'GarageCars', 'GarageArea', 'GarageQual',
       'GarageCond', 'PavedDrive', 'WoodDeckSF', 'OpenPorchSF',
       'EnclosedPorch', '3SsnPorch', 'ScreenPorch', 'PoolArea', 'PoolQC',
       'Fence', 'MiscFeature', 'MiscVal', 'MoSold', 'YrSold', 'SaleType',
       'SaleCondition', 'SalePrice'],
      dtype='object')
```

Prediction Target’i y’ye tanımladık.

```
y = home_data.SalePrice

# Check your answer
step_1.check()
```

Correct

Step 2: X Oluştur

Şimdi, predictive feature’ları (tahmin özelliklerini) tutan X adında bir DataFrame oluşturacaksınız.

Orijinal verilerden yalnızca bazı sütunlar istediğiniz için, önce X'de istediğiniz sütunların adlarını içeren bir liste oluşturacaksınız.

Listede yalnızca aşağıdaki sütunları kullanacaksınız :

```
* LotArea
* YearBuilt
* 1stFlrSF
* 2ndFlrSF
* FullBath
* BedroomAbvGr
* TotRmsAbvGrd
```

Bu özellik listesini oluşturduktan sonra, modeli fit etmek için kullanacağınız DataFrame'i oluşturmak için kullanın.

```
# Create the list of features below
feature_names = ["LotArea", "YearBuilt", "1stFlrSF", "2ndFlrSF", "FullBath", "BedroomAbvGr", "TotRmsAbvGrd"]

# Select data corresponding to features in feature_names
X = home_data[feature_names]

# Check your answer
step_2.check()
```

Correct

Verinin İncelenmesi

Bir model oluşturmadan önce, mantıklı göründüğünü doğrulamak için X'e hızlı bir göz atın.

```python
# Review data
# print description or statistics from X
print(X.describe())

# print the top few lines
print("\n",X.head())
```

```
             LotArea    YearBuilt     1stFlrSF     2ndFlrSF     FullBath  \
count    1460.000000  1460.000000  1460.000000  1460.000000  1460.000000
mean    10516.828082  1971.267808  1162.626712   346.992466     1.565068
std      9981.264932    30.202904   386.587738   436.528436     0.550916
min      1300.000000  1872.000000   334.000000     0.000000     0.000000
25%      7553.500000  1954.000000   882.000000     0.000000     1.000000
50%      9478.500000  1973.000000  1087.000000     0.000000     2.000000
75%     11601.500000  2000.000000  1391.250000   728.000000     2.000000
max    215245.000000  2010.000000  4692.000000  2065.000000     3.000000

       BedroomAbvGr  TotRmsAbvGrd
count   1460.000000   1460.000000
mean       2.866438      6.517808
std        0.815778      1.625393
min        0.000000      2.000000
25%        2.000000      5.000000
50%        3.000000      6.000000
75%        3.000000      7.000000
max        8.000000     14.000000
```

```
    LotArea  YearBuilt  1stFlrSF  2ndFlrSF  FullBath  BedroomAbvGr  \
0     8450       2003       856       854         2             3
1     9600       1976      1262         0         2             3
2    11250       2001       920       866         2             3
3     9550       1915       961       756         1             3
4    14260       2000      1145      1053         2             4

   TotRmsAbvGrd
0             8
1             6
2             6
3             7
4             9
```

Step 3: Modelin belirlenmesi ve fit edilmesi

DecisionTreeRegressor oluştur ve iowa_model'e kaydet. Bu komutu çalıştırmak için **sklearn'de** ilgili import işlemini yaptığınızdan emin olun.

[27]:
```
from sklearn.tree import DecisionTreeRegressor
#specify the model.
#For model reproducibility, set a numeric value for random_state when specifying the model
iowa_model = DecisionTreeRegressor(random_state=7)

# Fit the model
iowa_model.fit(X, y)

# Check your answer
step_3.check()
```

Correct

Step 4: Tahmin Yapma

Veri olarak **X**'i kullanarak modelin **predict** komutuyla tahminler yapın. Sonuçları **predictions** adı verilen bir değişkene kaydedin.

[30]:
```
predictions = iowa_model.predict(X)
print(predictions)

# Check your answer
step_4.check()
```

```
[208500. 181500. 223500. ... 266500. 142125. 147500.]
```

Correct

+ Code + Markdown

[33]:
```
home_data.SalePrice.head()
```

Out[33]:
```
0    208500
1    181500
2    223500
3    140000
4    250000
Name: SalePrice, dtype: int64
```

Model Validation (Model Geçerliliği)

Bir model oluşturdunuz. Ama bu model ne kadar iyi?

Bu derste, modelinizin kalitesini ölçmek için model validation(model doğrulamayı) kullanmayı öğreneceksiniz. Model kalitesini ölçmek, modellerinizi tekrar tekrar geliştirmenin anahtarıdır.

Model Validation Nedir?

Oluşturduğunuz hemen hemen her modeli değerlendirmek isteyeceksiniz.

Çoğu uygulamada, model kalitesiyle ilgili ölçü **predictive accuracy**(tahmini doğruluk)'dir.

Başka bir deyişle, modelin tahminleri gerçekte olana yakın olacak mı?

Birçok kişi, tahmin doğruluğunu ölçerken büyük bir hata yapar.

Training data ile tahmin yaparlar ve bu tahminleri training data'daki hedef değerlerle karşılaştırırlar.

Bu yaklaşımla ilgili sorunu ve bir anda nasıl çözüleceğini göreceksiniz, ancak önce bunu nasıl yapacağımızı düşünelim.

Önce model kalitesini anlaşılır bir şekilde özetlemeniz gerekir.

10.000 ev için tahmini ve gerçek ev değerlerini karşılaştırırsanız, muhtemelen iyi ve kötü tahminlerin bir karışımını bulacaksınız.

10.000 tahmini ve gerçek değerin listesine bakmak anlamsız olacaktır. Bunu tek bir metrikte özetlememiz gerekiyor.

Model kalitesini özetlemek için birçok metrik var, ancak **Mean Absolute Error** (Ortalama Mutlak Hata) (MAE olarak da adlandırılır) ile başlayacağız.

Son sözcükten başlayarak bu metriği inceleyelim, error.

Her ev için tahmin hatası:

```
error=actual-predicted
```

hata = gerçek değer – tahmin edilen değer

Yani, bir ev 150.000 dolara mal olduysa ve 100.000 dolara mal olacağını tahmin ederseniz, hata 50.000 dolar olacaktır.

MAE metriğiyle, her bir hatanın mutlak değerini alırız. Bu, her hatayı pozitif bir sayıya dönüştürür.

Daha sonra bu mutlak hataların ortalamasını alırız.

Bu bizim model kalitesi ölçümüzdür. Sade bir dille şöyle denilebilir ;

Ortalama olarak, tahminlerimiz yaklaşık X civarında.

MAE'yi hesaplamak için önce bir modele ihtiyacımız var.

```
In [1]:
# Data Loading Code Hidden Here
import pandas as pd

# Load data
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
# Filter rows with missing price values
filtered_melbourne_data = melbourne_data.dropna(axis=0)
# Choose target and features
y = filtered_melbourne_data.Price
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                      'YearBuilt', 'Lattitude', 'Longtitude']
X = filtered_melbourne_data[melbourne_features]

from sklearn.tree import DecisionTreeRegressor
# Define model
melbourne_model = DecisionTreeRegressor()
# Fit model
melbourne_model.fit(X, y)

Out[1]:
DecisionTreeRegressor(criterion='mse', max_depth=None, max_features=None,
                      max_leaf_nodes=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      presort=False, random_state=None, splitter='best')
```

Bir modelimiz olduğunda, ortalama mutlak hatayı şu şekilde hesaplıyoruz:

```
In [2]:
from sklearn.metrics import mean_absolute_error

predicted_home_prices = melbourne_model.predict(X)
mean_absolute_error(y, predicted_home_prices)

Out[2]:
434.71594577146544
```

The Problem with "In-Sample" Scores

Yeni hesapladığımız ölçüme "in-sample" score'u denilebilir. Hem modeli oluşturmak hem de değerlendirmek için tek bir "sample (örnek)" ev kullandık. Bu yüzden bu kötü bir tercihti.

Büyük emlak piyasasında kapı renginin ev fiyatıyla ilgisi olmadığını düşünün.

Ancak, modeli oluşturmak için kullandığınız veri örneğinde, yeşil kapıya sahip tüm evler çok pahalıydı.

Modelin işi, ev fiyatlarını tahmin eden pattern'ler bulmaktır, bu yüzden bu pattern'i görecek, ve her zaman yeşil kapılı evler için yüksek fiyatları tahmin edecektir.

Bu model training data'dan türetildiği için, model training datalarında doğru görünecektir.

Ancak, model yeni veriler gördüğünde bu pattern(örüntü) tutmazsa, model pratikte kullanıldığında çok inaccurate(yanlış) olur.

Modellerin pratik değeri yeni veriler üzerinde tahminler yapmaktan geldiğinden, modeli oluşturmak için kullanılmayan verilerdeki performansı ölçeriz.

Bunu yapmanın en basit yolu, bazı verileri model oluşturma sürecinden hariç tutmak ve daha sonra bunları, daha önce görmediği veriler üzerinde modelin doğruluğunu test etmek için kullanmaktır.

Bu verilere **validation data** (doğrulama verisi) denir.

Coding It

Scikit-learn kütüphanesi, verileri iki parçaya bölmek için **train_test_split** fonksiyonuna sahiptir.

Bu verilerin bir kısmını modeli fit etmek için *training data* olarak kullanacağız ve diğer verileri **mean_absolute_error** değerini hesaplamak için *validation data* (doğrulama verileri) olarak kullanacağız.

```
In [3]:
from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and target
# The split is based on a random number generator. Supplying a numeric value to
# the random_state argument guarantees we get the same split every time we
# run this script.
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 0)
# Define model
melbourne_model = DecisionTreeRegressor()
# Fit model
melbourne_model.fit(train_X, train_y)

# get predicted prices on validation data
val_predictions = melbourne_model.predict(val_X)
print(mean_absolute_error(val_y, val_predictions))

260991.8108457069
```

Wow!

in-sample veriler için mean absolute error değerimiz yaklaşık 500 dolardı. out-of-sample verilerde ise 250.000 dolardan fazla.

Bu, neredeyse tamamen doğru olan bir model ile en pratik amaçlar için kullanılamayan bir model arasındaki farktır.

Bir referans noktası olarak, validation data'daki (doğrulama verilerindeki) ortalama ev değeri 1,1 milyon dolar.

Yani yeni verilerdeki hata ortalama ev değerinin dörtte biri kadardır.

Bu modeli geliştirmenin daha iyi feature'lar bulmak veya farklı model türleri bulmayı denemek gibi birçok yolu vardır.

Exercise: Model Validation

Bir model oluşturdunuz. Bu alıştırmada modelinizin ne kadar iyi olduğunu test edeceksiniz.

[1]:

```python
# Code you have previously used to load data
import pandas as pd
from sklearn.tree import DecisionTreeRegressor

# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'

home_data = pd.read_csv(iowa_file_path)
y = home_data.SalePrice
feature_columns = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[feature_columns]

# Specify Model
iowa_model = DecisionTreeRegressor()
# Fit Model
iowa_model.fit(X, y)

print("First in-sample predictions:", iowa_model.predict(X.head()))
print("Actual target values for those homes:", y.head().tolist())

# Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex4 import *
print("Setup Complete")
```

```
First in-sample predictions: [208500. 181500. 223500. 140000. 250000.]
Actual target values for those homes: [208500, 181500, 223500, 140000, 250000]
Setup Complete
```

Exercises

Step 1: Split Your Data (Verinizi Ayırın)

Verilerinizi bölmek için **train_test_split** işlevini kullanın.

Hatırlayın, feature'larınız DataFrame X'e yüklenir ve target(hedefiniz) y olarak yüklenir.

```
[3]:
# Import the train_test_split function and uncomment
from sklearn.model_selection import train_test_split

# fill in and uncomment
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# Check your answer
step_1.check()
```

Correct

Step 2: Specify and Fit the Model (Modeli belirleme ve fit etme)

DecisionTreeRegressor modeli oluşturun ve modeli ilgili veriler ile fit edin.

```
[5]:
# You imported DecisionTreeRegressor in your last exercise
# and that code has been copied to the setup code above. So, no need to
# import it again

# Specify the model
iowa_model = DecisionTreeRegressor(random_state=1)

# Fit iowa_model with the training data.
iowa_model.fit(train_X, train_y)

# Check your answer
step_2.check()
```

```
[186500. 184000. 130000.  92000. 164500. 220000. 335000. 144152. 215000.
 262000.]
[186500. 184000. 130000.  92000. 164500. 220000. 335000. 144152. 215000.
 262000.]
```

Step 3: Make Predictions with Validation Data

[6]:
```
# Predict with all validation observations
val_predictions = iowa_model.predict(val_X)

# Check your answer
step_3.check()
```

Correct

Inspect your predictions and actual values from validation data.

+ Code + Markdown

[16]:
```
# print the top few validation predictions
print(val_predictions[:5],"\n")
# print the top few actual prices from validation data
print(val_y.head())
```

```
[186500. 184000. 130000.  92000. 164500.]

258     231500
267     179500
288     122000
649      84500
1233    142000
Name: SalePrice, dtype: int64
```

Bu gördüğünüz çıktıların in-sample tahminlerden neden farklı olduğunu anladınız mı?

Validation predictions'ların neden in-sample (veya train) predictions'larından farklı olduğunu hatırlıyor musunuz?

Step 4: Calculate the Mean Absolute Error in Validation Data

```
from sklearn.metrics import mean_absolute_error
val_mae = mean_absolute_error(val_y, val_predictions)

# uncomment following line to see the validation_mae
print(val_mae)

# Check your answer
step_4.check()
```

```
29652.931506849316
```

Correct

MAE sonucu iyi mi? Uygulamalar arasında geçerli olan değerlerin genel bir kuralı yoktur. Ancak bir sonraki adımda bu sayının nasıl kullanılacağını (ve geliştirileceğini) göreceksiniz.

Underfitting and Overfitting

Bu adımın sonunda, **underfitting**(uygun olmayan) ve **overfitting**(fazla uygunluk) kavramlarını anlayacak ve modellerinizi daha doğru hale getirmek için bu fikirleri uygulayabileceksiniz.

Farklı Modellerle Deneme

Artık model doğruluğunu ölçmenin güvenilir bir yoluna sahip olduğunuza göre, alternatif modelleri deneyebilir ve hangisinin en iyi tahminleri verdiğini görebilirsiniz.

Peki modeller için hangi alternatifleriniz var?

Scikit-learn'un dökümantasyonunda, Decision Tree modelinin birçok seçeneğe sahip olduğunu görebilirsiniz (isteyeceğinizden veya ihtiyacınız olandan daha fazla).

En önemli seçenekler ağacın derinliğini belirler. Bu mikro kursta ilk dersten, bir ağacın derinliğinin bir tahmine gelmeden önce kaç bölünme yaptığının bir ölçüsü olduğunu hatırlayın. Bu nispeten sığ bir ağaçtır:

Uygulamada, bir ağacın en üst seviyesi (tüm evler) ve bir leaf(yaprak) arasında 10 bölünme olması nadir değildir.

Ağaç derinleştikçe, veri kümesi daha az ev içeren yapraklara dilimlenir.

Bir ağacın sadece 1 bölünmesi varsa, verileri 2 gruba ayırır.

Her grup tekrar bölünürse, 4 grup ev alırdık. Bunların her birini tekrar bölmek 8 grup oluşturacaktır.

Her seviyede daha fazla bölme ekleyerek grup sayısını ikiye katlamaya devam edersek, 10. seviyeye ulaştığımızda 2^{10} ev grubumuz olacak. Bu da 1024 yaprak yapar.

Evleri birçok yaprak arasında böldüğümüzde, her yaprakta da daha az ev olur.

Çok az evi olan yapraklar, o evlerin gerçek değerlerine oldukça yakın tahminler yapacak, ancak yeni veriler için çok güvenilir olmayan tahminler yapabilirler (çünkü her tahmin sadece birkaç eve dayanmaktadır).

Bu, bir modelin train(eğitim) verileriyle neredeyse mükemmel şekilde eşleştiği, ancak validation(doğrulama) ve diğer yeni verilerde yetersiz olduğu, **overfitting** takma adı verilen bir fenomendir.

Flip tarafında, eğer ağacımızı çok sığ yaparsak, evleri çok farklı gruplara ayırmaz.

Extreme olarak, bir ağaç evleri sadece 2 veya 4'e ayırırsa, her grubun hala çok çeşitli evleri vardır.

Sonuç tahminleri(predictions), train verilerinde bile çoğu ev için çok uzak olabilir (ve aynı nedenden dolayı validation(doğrulama) da kötü olacaktır).

Bir model verilerdeki önemli ayrımları ve pattern'leri(desenleri) yakalayamadığında, train verilerinde bile yetersiz performans gösterir, buna **underfitting** denir.

Validation data'mızdan(doğrulama verimizden) predict(tahmin) ettiğimiz yeni verilerdeki accuracy'i(doğruluğu) önemsediğimiz için, **underfitting** ve **overfititng** arasındaki tatlı noktayı bulmak istiyoruz.

Görsel olarak, (kırmızı) doğrulama eğrisinin(validation curve) düşük noktasını bulmak istiyoruz.

Mean Average Error
Underfitting
Overfitting
Validation
Training
Tree Depth

Examples

Ağaç derinliğini kontrol etmek için birkaç alternatif vardır ve birçoğu ağaçtaki bazı yolların diğer yollardan daha fazla derinliğe sahip olmasına izin verir.

Ancak max_leaf_nodes argümanı, overfitting ve underfiting'i kontrol etmek için çok mantıklı bir yol sağlar.

Modelin ne kadar fazla leaf(yaprak) yapmasına izin verirsek, yukarıdaki grafikteki underfitting alanından overfitting alanına o kadar fazla hareket ederiz.

Max_leaf_nodes için farklı değerlerden MAE puanlarını karşılaştırmaya yardımcı olması için bir yardımcı program işlevi kullanabiliriz:

In [1]:
```
from sklearn.metrics import mean_absolute_error
from sklearn.tree import DecisionTreeRegressor

def get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y):
    model = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes, random_state=0)
    model.fit(train_X, train_y)
    preds_val = model.predict(val_X)
    mae = mean_absolute_error(val_y, preds_val)
    return(mae)
```

Veriler, daha önce gördüğünüz (ve daha önce yazdığınız) kodu kullanarak train_X, val_X, train_y ve val_y içine yüklenir.

In [2]:
```
# Data Loading Code Runs At This Point
import pandas as pd

# Load data
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
# Filter rows with missing values
filtered_melbourne_data = melbourne_data.dropna(axis=0)
# Choose target and features
y = filtered_melbourne_data.Price
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                      'YearBuilt', 'Lattitude', 'Longtitude']
X = filtered_melbourne_data[melbourne_features]

from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and target
train_X, val_X, train_y, val_y = train_test_split(X, y,random_state = 0)
```

Max_leaf_nodes için farklı değerlerle oluşturulan modellerin doğruluğunu karşılaştırmak için bir for-loop kullanabiliriz.

In [3]:
```
# compare MAE with differing values of max_leaf_nodes
for max_leaf_nodes in [5, 50, 500, 5000]:
    my_mae = get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y)
    print("Max leaf nodes: %d  \t\t Mean Absolute Error:  %d" %(max_leaf_nodes, my_mae))
```

```
Max leaf nodes: 5               Mean Absolute Error:  347380
Max leaf nodes: 50              Mean Absolute Error:  258171
Max leaf nodes: 500             Mean Absolute Error:  243495
Max leaf nodes: 5000            Mean Absolute Error:  254983
```

Listelenen seçeneklerden 500, en uygun yaprak sayısıdır.

Sonuç

Modeller şunlardan herhangi birine sahip olabilir:

- **Overfitting:** gelecekte tekrarlamayacak sahte pattern(desen)leri yakalamak, daha az doğru tahminlere yol açmak veya
- **Underfitting:** alakalı pattern'leri yakalayamama, yine daha az doğru tahminlere yol açma.

Bir aday modelin doğruluğunu(accuracy) ölçmek için model eğitiminde(train) kullanılmayan **doğrulama(validation)** verilerini kullanıyoruz. Bu, birçok aday modeli denememizi ve en iyisini elde etmemizi sağlar.

Exercise: Underfitting and Overfitting

İlk modelinizi oluşturdunuz ve şimdi daha iyi tahminler yapmak için ağacın boyutunu optimize etme zamanı. Önceki adımı bıraktığınız yerde kodlama ortamınızı ayarlamak için bu hücreyi çalıştırın.

```
# Code you have previously used to load data
import pandas as pd
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor


# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'

home_data = pd.read_csv(iowa_file_path)
# Create target object and call it y
y = home_data.SalePrice
# Create X
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[features]

# Split into validation and training data
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# Specify Model
iowa_model = DecisionTreeRegressor(random_state=1)
# Fit Model
iowa_model.fit(train_X, train_y)

# Make validation predictions and calculate mean absolute error
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE: {:,.0f}".format(val_mae))

# Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex5 import *
print("\nSetup complete")
```

```
Validation MAE: 29,653

Setup complete
```

Exercises

Get_mae fonksiyonunu kendiniz yazabilirsiniz. Şimdilik tedarik edeceğiz. Bu, bir önceki derste okuduğunuz işlevle aynıdır. Aşağıdaki hücreyi çalıştırmanız yeterlidir.

[]:
```
def get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y):
    model = DecisionTreeRegressor(max_leaf_nodes=max_leaf_nodes, random_state=0)
    model.fit(train_X, train_y)
    preds_val = model.predict(val_X)
    mae = mean_absolute_error(val_y, preds_val)
    return(mae)
```

Step 1: Compare Different Tree Sizes (Farklı ağaç boyutlarını karşılaştırın)

Bir dizi olası değerden ***max_leaf_nodes*** için aşağıdaki değerleri çalıştıran bir döngü yazın.

Her max_leaf_nodes değerinde get_mae işlevini çağırın. Çıktıyı, verilerinizde en doğru modeli veren max_leaf_nodes değerini seçmenize izin verecek şekilde saklayın.

[10]:
```
candidate_max_leaf_nodes = [5, 25, 50, 100, 250, 500]
# Write loop to find the ideal tree size from candidate_max_leaf_nodes
for max_leaf_nodes in candidate_max_leaf_nodes:
    my_mae = get_mae(max_leaf_nodes, train_X, val_X, train_y, val_y)
    print("Max leaf nodes: {0} \t\t Mean absolute error: {1}".format(max_leaf_nodes, my_mae))

# Store the best value of max_leaf_nodes (it will be either 5, 25, 50, 100, 250 or 500)
best_tree_size = 100

# Check your answer
step_1.check()
```

```
Max leaf nodes: 5                Mean absolute error: 35044.51299744237
Max leaf nodes: 25               Mean absolute error: 29016.41319191076
Max leaf nodes: 50               Mean absolute error: 27405.930473214907
Max leaf nodes: 100              Mean absolute error: 27282.50803885739
Max leaf nodes: 250              Mean absolute error: 27893.822225701646
Max leaf nodes: 500              Mean absolute error: 29454.18598068598
```

Correct

Step 2: Fit Model Using All Data

En iyi ağaç boyutunu biliyorsun. Bu modeli pratikte deploy edecek olsaydınız, tüm verileri kullanarak ve bu ağaç boyutunu koruyarak daha da doğru hale getirirsiniz.

Yani, tüm modelleme kararlarınızı verdiğiniz için doğrulama verilerini saklamanız gerekmez.

[14]:
```
# Fill in argument to make optimal size and uncomment
final_model = DecisionTreeRegressor(max_leaf_nodes=best_tree_size, random_state=1)

# fit the final model and uncomment the next two lines
final_model.fit(X, y)

# Check your answer
step_2.check()
```

Correct

Bu modeli ayarladınız ve sonuçlarınızı geliştirdiniz. Ancak hala modern makine öğrenimi standartlarına göre çok karmaşık olmayan *Decision Tree*

modellerini kullanıyoruz. Bir sonraki adımda, modellerinizi daha da geliştirmek için **Random Forest** kullanmayı öğreneceksiniz.

Random Forests

Introduction

Decision Tree sizi zor bir kararla baş başa bırakır. Çok sayıda yapraklı derin bir ağaç, her tahmin, yaprağındaki sadece birkaç evden gelen tarihsel verilerden geldiğinden fazla olacaktır. Ancak, az yapraklı sığ bir ağaç kötü performans gösterecektir, çünkü ham verilerdeki birçok farklılığı yakalayamaz.

Günümüzün en sofistike modelleme teknikleri bile, underfitting ve overfitting arasındaki bu gerilim ile karşı karşıyadır.

Ancak, birçok model daha iyi performans sağlayabilecek akıllı fikirlere sahiptir. Örnek olarak **Random Forest**'a bakacağız.

Random Forest birçok ağaç kullanır ve her bileşen ağacının tahminlerini ortalayarak bir tahmin yapar.

Genellikle tek bir karar ağacından çok daha iyi tahmin doğruluğu(predictive accuracy) vardır ve varsayılan parametrelerle iyi çalışır.

Modellemeye devam ederseniz, daha iyi performansa sahip daha fazla model öğrenebilirsiniz, ancak bunların çoğu doğru parametreleri almaya duyarlıdır.

Example

Verileri yüklemek için gereken kodu zaten birkaç kez gördünüz. Veri yüklemenin sonunda aşağıdaki değişkenler bulunur:

- train_X
- val_X
- train_y
- val_y

```
In [1]:
import pandas as pd

# Load data
melbourne_file_path = '../input/melbourne-housing-snapshot/melb_data.csv'
melbourne_data = pd.read_csv(melbourne_file_path)
# Filter rows with missing values
melbourne_data = melbourne_data.dropna(axis=0)
# Choose target and features
y = melbourne_data.Price
melbourne_features = ['Rooms', 'Bathroom', 'Landsize', 'BuildingArea',
                      'YearBuilt', 'Lattitude', 'Longtitude']
X = melbourne_data[melbourne_features]

from sklearn.model_selection import train_test_split

# split data into training and validation data, for both features and target
# The split is based on a random number generator. Supplying a numeric value to
# the random_state argument guarantees we get the same split every time we
# run this script.
train_X, val_X, train_y, val_y = train_test_split(X, y,random_state = 0)
```

scikit-learn kütüphanesinde decision tree modeli oluşturduğumuz gibi bu kez **random forest** modeli oluşturacağız. – **DecisionTreeRegressor** yerine **RandomTreeRegressor** kullanacağız.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

forest_model = RandomForestRegressor(random_state=1)
forest_model.fit(train_X, train_y)
melb_preds = forest_model.predict(val_X)
print(mean_absolute_error(val_y, melb_preds))
```

```
/opt/conda/lib/python3.6/site-packages/sklearn/ensemble/fores
t.py:245: FutureWarning: The default value of n_estimators wil
l change from 10 in version 0.20 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)


202888.18157951365
```

Sonuç

Daha da iyileştirilmesi muhtemeldir, ancak bu 250.000 olan en iyi karar ağacı hatası üzerinde büyük bir gelişmedir.

Single decision tree’nin maksimum derinliğini değiştirdiğimiz gibi Random Forest'ın da performansını değiştirmenize izin veren parametreler var.

Ancak Random Forest modellerinin en iyi özelliklerinden biri, bu ayarlama olmadan bile genellikle makul bir şekilde çalışmasıdır.

Yakında, doğru parametrelerle iyi ayarlandığında daha iyi performans sağlayan (ancak doğru model parametrelerini elde etmek için biraz beceri gerektiren) XGBoost modelini öğreneceksiniz.

Exercises: Random Forest

Şimdiye kadar yazdığımız kod:

```
# Code you have previously used to load data
import pandas as pd
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor


# Path of the file to read
iowa_file_path = '../input/home-data-for-ml-course/train.csv'

home_data = pd.read_csv(iowa_file_path)
# Create target object and call it y
y = home_data.SalePrice
# Create X
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[features]

# Split into validation and training data
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# Specify Model
iowa_model = DecisionTreeRegressor(random_state=1)
# Fit Model
iowa_model.fit(train_X, train_y)

# Make validation predictions and calculate mean absolute error
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE when not specifying max_leaf_nodes: {:,.0f}".format(val_mae))

# Using best value for max_leaf_nodes
iowa_model = DecisionTreeRegressor(max_leaf_nodes=100, random_state=1)
iowa_model.fit(train_X, train_y)
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE for best value of max_leaf_nodes: {:,.0f}".format(val_mae))


# Set up code checking
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex6 import *
print("\nSetup complete")
```

```
Validation MAE when not specifying max_leaf_nodes: 29,653
Validation MAE for best value of max_leaf_nodes: 27,283

Setup complete
```

Exercises

Veri bilimi her zaman bu kadar kolay değildir. Ancak Decision Tree'yi Random Forest ile değiştirmek kolay bir kazanç olacaktır.

Step 1: Use a Random Forest

[28]:
```
from sklearn.ensemble import RandomForestRegressor

# Define the model. Set random_state to 1
rf_model = RandomForestRegressor(random_state=1)

# fit your model
rf_model.fit(train_X, train_y)

# Calculate the mean absolute error of your Random Forest model on the validation data
rf_val_predictions = rf_model.predict(val_X)
rf_val_mae = mean_absolute_error(val_y, rf_val_predictions)

print("Validation MAE for Random Forest Model: {:,.0f}".format(rf_val_mae))

# Check your answer
step_1.check()
```

```
Validation MAE for Random Forest Model: 21,857
```

Correct

Şimdiye kadar, projenizin her adımında belirli talimatları izlediniz. Bu, temel fikirleri öğrenmeye ve ilk modelinizi oluşturmaya yardımcı oldu, ancak şimdi işleri kendi başınıza denemek için yeterince bilgi sahibisiniz.

Machine Learning yarışmaları, bağımsız olarak bir machine learning projesinde gezinirken kendi fikirlerinizi denemek ve daha fazla bilgi edinmek için harika bir yoldur.

Exercises: Machine Learning Competitions

Introduction

Makine öğrenimi yarışmaları, veri bilimi becerilerinizi geliştirmenin ve ilerlemenizi ölçmenin harika bir yoludur.

Bu alıştırmada, bir Kaggle yarışması için tahminler oluşturacak ve sunacaksınız.

Bu notebook'daki adımlar:

- Tüm verilerinizle Random Forest modeli oluşturun. (X ve y)
- Target(hedef) içermeyen "test" verilini okuyun. Random Forest modelinizle test verilerindeki ev fiyatlarını tahmin edin.
- Bu tahminleri yarışmaya gönderin ve puanınızı görün.
- İsteğe bağlı olarak, feature'lar ekleyerek veya modelinizi değiştirerek modelinizi geliştirip geliştiremeyeceğinizi görmek için tekrar deneyin. Daha sonra bunun rekabet lider panosunda nasıl etkilediğini görmek için yeniden gönderebilirsiniz.

Şimdiye kadar yazdığımız kod:

```
# Code you have previously used to load data
import pandas as pd
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeRegressor

# Set up code checking
import os
if not os.path.exists("../input/train.csv"):
    os.symlink("../input/home-data-for-ml-course/train.csv", "../input/train.csv")
    os.symlink("../input/home-data-for-ml-course/test.csv", "../input/test.csv")
from learntools.core import binder
binder.bind(globals())
from learntools.machine_learning.ex7 import *

# Path of the file to read. We changed the directory structure to simplify submitting to a competition
iowa_file_path = '../input/train.csv'

home_data = pd.read_csv(iowa_file_path)
# Create target object and call it y
y = home_data.SalePrice
# Create X
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = home_data[features]

# Split into validation and training data
train_X, val_X, train_y, val_y = train_test_split(X, y, random_state=1)

# Specify Model
iowa_model = DecisionTreeRegressor(random_state=1)
# Fit Model
iowa_model.fit(train_X, train_y)
```

```
# Make validation predictions and calculate mean absolute error
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE when not specifying max_leaf_nodes: {:,.0f}".format(val_mae))

# Using best value for max_leaf_nodes
iowa_model = DecisionTreeRegressor(max_leaf_nodes=100, random_state=1)
iowa_model.fit(train_X, train_y)
val_predictions = iowa_model.predict(val_X)
val_mae = mean_absolute_error(val_predictions, val_y)
print("Validation MAE for best value of max_leaf_nodes: {:,.0f}".format(val_mae))

# Define the model. Set random_state to 1
rf_model = RandomForestRegressor(random_state=1)
rf_model.fit(train_X, train_y)
rf_val_predictions = rf_model.predict(val_X)
rf_val_mae = mean_absolute_error(rf_val_predictions, val_y)

print("Validation MAE for Random Forest Model: {:,.0f}".format(rf_val_mae))
```

```
Validation MAE when not specifying max_leaf_nodes: 29,653
Validation MAE for best value of max_leaf_nodes: 27,283
Validation MAE for Random Forest Model: 21,857
```

Creating a Model For the Competition

Random Forest modeli oluşturun ve tüm X ve y ile modeli eğitin.

In [2]:
```
# To improve accuracy, create a new Random Forest model which you will train on all training data
rf_model_on_full_data = RandomForestRegressor(random_state=1)

# fit rf_model_on_full_data on all data from the training data
rf_model_on_full_data.fit(X, y)
```

Out[2]:
```
RandomForestRegressor(random_state=1)
```

Make Predictions

"Test" verileri dosyasını okuyun. Tahmin yapmak için modelinizi uygulayın.

In [3]:
```
# path to file you will use for predictions
test_data_path = '../input/test.csv'

# read test data file using pandas
test_data = pd.read_csv(test_data_path)

# create test_X which comes from test_data but includes only the columns you used for prediction.
# The list of columns is stored in a variable called features
test_X = test_data[features]
# make predictions which we will submit.
test_preds = rf_model_on_full_data.predict(test_X)

# The lines below shows how to save predictions in format used for competition scoring
# Just uncomment them.
output = pd.DataFrame({'Id': test_data.Id,
                       'SalePrice': test_preds})

output.to_csv('submission.csv', index=False)
```

Modelinizi geliştirmenin birçok yolu vardır ve deneme yapmak bu noktada öğrenmenin harika bir yoludur.

Modelinizi geliştirmenin en iyi yolu özellikler eklemektir. Sütun listesine bakın ve konut fiyatlarını nelerin etkileyebileceğini düşünün.

Bazı özellikler, eksik değerler veya sayısal olmayan veri türleri gibi sorunlar nedeniyle hatalara neden olur.

Quiz: Intro to Machine Learning

✓ Q1- After training our decision tree model, we saw that the model is overfitted on the training data and it has bad performance on the test data. Which hyper-parameter could help us to get rid of this problem? Note: You can use sklearn.tree.DecisionTreeClassifier documentation.https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html#sklearn.tree.DecisionTreeClassifier * 10/10

- ○ criterion
- ◉ max_depth ✓
- ○ random_state
- ○ splitter

✓ Q2- Which of the below can be said definitely according to the results table taken from the data.describe() method? I. 75% of the values in the Rooms column are greater than 2. II. There are some houses with a land size of 0. III. There are missing values in the BuildingArea column. IV. There is no house with 9 rooms in the data set * 10/10

In [2]:
```
import pandas as pd

data = pd.read_csv("/home/fatih/Desktop/melb_data.csv")

data.describe()
```

Out[2]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13518.000000	13580.000000	7130.000000	8205.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242	1.610075	558.416127	151.967650	1964.684217
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712	0.962634	3990.669241	541.014538	37.273762
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1196.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000	177.000000	93.000000	1940.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000	440.000000	126.000000	1970.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000	651.000000	174.000000	1999.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000	10.000000	433014.000000	44515.000000	2018.000000

- ○ I, II
- ○ II, III
- ○ II, III, IV
- ◉ I, II, III ✓

✓ Q3- Which one is false about overfitting and underfitting? * 10/10

- ○ Insufficient training (less epoch less batch size), causes underfitting.
- ○ Training on too much epoch and batch size causes overfitting.
- ◉ Splitting dataset as train and test datasets will always be enough to prevent overfitting, no need for validation datasets. ✓
- ○ In overfitting accuracy will be very good at train data but will be very bad at unseen data.

✓ Q4- Which of the following is false regarding pandas and scikit-learn methods? * 10/10

- ○ DataFrame.head(x) shows x samples in the DataFrame from the beginning.
- ○ DataFrame.describe() shows summary of the data.
- ◉ model.predict() determines how accurate the model's predictions are. ✓
- ○ DataFrame.dropna(axis=0) drops missing values.

Q5- According to the shooting clusters scheme above, for each figure which statements are true? Notice that, shooting targets are the centers. * 10/10

1:Low Bias- Low Variance 2:Low Bias-High Variance 3:High Bias-Low Variance 4: High Bias-High Variance

✓ Q6- Which of the below statements are true? * 10/10

I - It is an algorithm that aims to increase the classification value by producing multiple decision trees.

II - It was created by combining Bagging and Random Subspace methods.

III - While creating the tree, it is made performance evaluation with 2/3 of the data set.

- ○ I, III
- ○ II, III
- ◉ I, II ✓
- ○ I, II, III

✗ Q7- What do you think about train_X when line 1 and line 2 are executed separately? The rest of the code is exactly the same. * 0/10

```
Line 1. train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 2,shuffle=False)
Line 2. train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 1,shuffle=False)
```

- ○ They generate different random number so the train_X differs from each other.
- ◉ They generate different same number and the train_X is equal to each other. ✗
- ○ They generate different random number so the train_X is equal to each other.
- ○ They generate different random number ,but the train_X is equal to each other.

✓ Q8- Trees have their length and we call that the depth of the tree. RandomForestRegressor, in scikit-learn library, has a maximum leaf (max_depth) parameter which is None as default which means nodes are expanded until all leaves are pure. What can be said if we change the number of maximum leaf nodes of a random forest? * 10/10

- ○ Length of a tree does not affect any of the results.
- ◉ Model may overfit for large depth values. ✓
- ○ The longer tree is the better tree.
- ○ Short trees more precise than long trees.

✓ Q9- Let assume, we have a data set called home_data with 3 features names; LotArea, YearBuilt, PoolArea. How do you define non-missing values for the feature LotArea? * 10/10

- ○ non_missings = home_data["LotArea"].mean()
- ○ non_missings = home_data.count()
- ◉ non_missings = home_data["LotArea"].count() ✓
- ○ non_missings = home_data.mean()

✓ Q10- What is the aim of the below code pieces? * 10/10

```
from sklearn.metrics import mean_absolute_error

predicted_home_prices = melbourne_model.predict(X)
mean_absolute_error(y, predicted_home_prices)
```

- ○ For splitting the data as test and train
- ○ For interpreting the data description
- ◉ For summarizing model quality ✓
- ○ For data modelling

Intermediate Machine Learning

Introduction

Kaggle Learn'in Orta Düzey Makine Öğrenimi mikro kursuna hoş geldiniz!

Makine öğreniminde biraz geçmişiniz varsa ve modellerinizin kalitesini nasıl hızla artıracağınızı öğrenmek istiyorsanız, doğru yerdesiniz!

Bu mikro kursta, aşağıdakileri nasıl yapacağınızı öğrenerek makine öğrenimi uzmanlığınızı hızlandıracaksınız:

- gerçek dünya veri kümelerinde sıklıkla bulunan veri türlerini ele alır (missing values, categorical variables),
- makine öğrenme kodunuzun kalitesini artırmak için **pipeline'lar** tasarlamak,
- model doğruluğu için gelişmiş teknikler kullanabilecek (**cross validation**),
- Kaggle yarışmalarını kazanmak için yaygın olarak kullanılan son model modeller oluşturmak (**XGBoost**) ve
- yaygın ve önemli veri bilimi hatalarından (**leakege**) kaçının.

Kurs boyunca, her yeni konu için gerçek verilerle uygulamalı bir alıştırma yaparak bilginizi güçlendireceksiniz.

Uygulamalı alıştırmalar Housing Prices Competition for Kaggle Learn Users'dan elde edilen verileri kullanır, burada ev fiyatlarını tahmin etmek için 79 farklı açıklayıcı değişken (type of roof, number of bedrooms, and number of bathrooms gibi) kullanacaksınız.

Bu yarışmaya tahminler göndererek ve liderlik sıralamasında pozisyonunuzun yükselişini izleyerek ilerlemenizi ölçeceksiniz!

Exercises

Bir ısınma olarak, bazı makine öğrenimi temellerini gözden geçirecek ve ilk sonuçlarınızı bir Kaggle yarışmasına sunacaksınız.

Housing Prices Competition for Kaggle Learn Users'dan elde edilen verilerle, evlerin her yönünü (neredeyse) tanımlayan 79 açıklayıcı değişkeni kullanarak Iowa'daki ev fiyatlarını tahmin etmek için çalışacaksınız.

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the data
X_full = pd.read_csv('../input/train.csv', index_col='Id')
X_test_full = pd.read_csv('../input/test.csv', index_col='Id')

# Obtain target and predictors
y = X_full.SalePrice
features = ['LotArea', 'YearBuilt', '1stFlrSF', '2ndFlrSF', 'FullBath', 'BedroomAbvGr', 'TotRmsAbvGrd']
X = X_full[features].copy()
X_test = X_test_full[features].copy()

# Break off validation set from training data
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size=0.2,
                                                      random_state=0)
```

```
[2]: X_train.head()
```

Out[2]:

	LotArea	YearBuilt	1stFlrSF	2ndFlrSF	FullBath	BedroomAbvGr	TotRmsAbvGrd
Id							
619	11694	2007	1828	0	2	3	9
871	6600	1962	894	0	1	2	5
93	13360	1921	964	0	1	2	5
818	13265	2002	1689	0	2	3	7
303	13704	2001	1541	0	2	3	6

Step 1 : Eveluate Several Models (Birkaç modeli değerlendirin)

Bir sonraki kod hücresi, beş farklı Random Forest modelini tanımlar. Bu kod hücresini değişiklik yapmadan çalıştırın.

```
[3]: from sklearn.ensemble import RandomForestRegressor

# Define the models
model_1 = RandomForestRegressor(n_estimators=50, random_state=0)
model_2 = RandomForestRegressor(n_estimators=100, random_state=0)
model_3 = RandomForestRegressor(n_estimators=100, criterion='mae', random_state=0)
model_4 = RandomForestRegressor(n_estimators=200, min_samples_split=20, random_state=0)
model_5 = RandomForestRegressor(n_estimators=100, max_depth=7, random_state=0)

models = [model_1, model_2, model_3, model_4, model_5]
```

Burada kullandığımız parametrelere göz atalım;

n_estimators : Random Forest içerisinde oluşturulacak ağaç sayısı. Default=10

criterion : Bölmenin kalitesini ölçen ölçüt. Desteklenen ölçütler, ortalama kare hatası için “mse” dir; bu özellik özellik seçimi kriteri olarak varyans azaltmaya eşittir ve ortalama mutlak hata için “mae” dir.

min_samples_split : Bir bölünmenin gerçekleşmesi için verilerinizde bulunması gereken minimum örnek sayısını ayarlar. Eğer bir float ise o zaman min_samples_split*n_samples ile hesaplanır.

> **Not:** İyi sonuçlar genellikle **max_depth=None** ayarında **min_samples_split=1** ile birlikte yapılır. Bu değerleri kullanmanın belleği çok fazla işgal eden modellerle sonuçlanabileceğini unutmayın.

max_depth: (integer or none) Default=None. Ağaçlarınızı ne kadar derin yapacağınızı ayarlar. max_depth'inizi ayarlamanız, overfitting ile başa çıkabilmeniz için önerilir.

Beş model içinden en iyi modeli seçmek için, aşağıda **score_model ()** fonksiyonunu tanımlarız. Bu işlev, doğrulama kümesinden ortalama mutlak hatayı (**MAE**) döndürür. En iyi modelin en düşük MAE'yi elde edeceğini hatırlayın.

```
[4]:
from sklearn.metrics import mean_absolute_error

# Function for comparing different models
def score_model(model, X_t=X_train, X_v=X_valid, y_t=y_train, y_v=y_valid):
    model.fit(X_t, y_t)
    preds = model.predict(X_v)
    return mean_absolute_error(y_v, preds)

for i in range(0, len(models)):
    mae = score_model(models[i])
    print("Model %d MAE: %d" % (i+1, mae))

Model 1 MAE: 24015
Model 2 MAE: 23740
Model 3 MAE: 23528
Model 4 MAE: 23996
Model 5 MAE: 23706
```

Aşağıdaki satırı doldurmak için yukarıdaki sonuçları kullanın. Hangi model en iyi modeldir? Cevabınız model_1, model_2, model_3, model_4 veya model_5'ten biri olmalıdır.

```
# Fill in the best model
best_model = model_3

# Check your answer
step_1.check()

Correct
```

Step 2: Generate Test Prediction (Test tahminleri oluşturun)

```
[8]:
# Define a model
my_model = RandomForestRegressor(n_estimators=100, criterion="mae", random_state=0) # Your code here

# Check your answer
step_2.check()

Correct
```

Aşağıdaki kod, modeli train ve validation verilerine fit eder ve ardından bir CSV dosyasına kaydedilen test tahminleri oluşturur.

```
[9]:
# Fit the model to the training data
my_model.fit(X, y)

# Generate test predictions
preds_test = my_model.predict(X_test)

# Save predictions in format used for competition scoring
output = pd.DataFrame({'Id': X_test.index,
                       'SalePrice': preds_test})
output.to_csv('submission.csv', index=False)
```

Missing Values (Eksik Veriler)

Bu derste, eksik değerlerle başa çıkmak için üç yaklaşım öğreneceksiniz. Ardından bu yaklaşımların etkilerini gerçek dünyadaki bir veri kümesinde karşılaştıracaksınız.

Verilerin eksik değerlerle sonuçlanmasının birçok yolu vardır. Örneğin,

- 2 yatak odalı bir evde üçüncü bir yatak odası için bir değer bulunmayacaktır.
- Ankete katılan bir kişi gelirini paylaşmamayı tercih edebilir.

Çoğu makine öğrenme kütüphanesi (scikit-learn dahil) eksik değerlere sahip veriler kullanarak bir model oluşturmaya çalışırsanız hata verir.

Üç Yaklaşım

1 Basit Bir Seçenek: Eksik Değerli Sütunları Düşürme

En basit seçenek, eksik değerlere sahip sütunları düşürmektir.

Bed	Bath
1.0	1.0
2.0	1.0
3.0	2.0
NaN	2.0

→

Bath
1.0
1.0
2.0
2.0

Düşürülen sütunlardaki değerlerin çoğu eksik değilse, model bu yaklaşımla çok sayıda bilgiye(potansiyel olarak yararlı!) erişimi kaybeder.

2 Daha İyi Bir Seçenek: Imputation

Empütasyon eksik değerleri bir sayı ile doldurur. Örneğin, her sütun boyunca ortalama değeri doldurabiliriz.

Bed	Bath
1.0	1.0
2.0	1.0
3.0	2.0
NaN	2.0

Bed	Bath
1.0	1.0
2.0	1.0
3.0	2.0
2.0	2.0

Öngörülen değer çoğu durumda tam olarak doğru olmaz, ancak genellikle sütunu tamamen bırakmanızdan daha doğru modellere yol açar.

3 An Extension To Imputation

Imputasyon standart bir yaklaşımdır ve genellikle iyi çalışır. Ancak, doldurulan değerler sistematik olarak gerçek değerlerinin (veri kümesinde toplanmayan) üstünde veya altında olabilir. Veya eksik değerleri olan satırlar başka bir şekilde benzersiz olabilir. Bu durumda, modeliniz başlangıçta hangi değerlerin eksik olduğunu göz önünde bulundurarak daha iyi tahminlerde bulunur.

Bed	Bath
1.0	1.0
2.0	1.0
3.0	2.0
NaN	2.0

Bed	Bath	Bed_was_missing
1.0	1.0	FALSE
2.0	1.0	FALSE
3.0	2.0	FALSE
2.0	2.0	TRUE

Bu yaklaşımda, eksik değerleri önceki gibi impute ediyoruz. Ayrıca, orijinal veri kümesinde eksik girişleri olan her sütun için, etkilenen girişlerin konumunu gösteren yeni bir sütun ekliyoruz.

Bazı durumlarda bu, sonuçları anlamlı şekilde iyileştirir. Diğer durumlarda, hiç yardımcı olmuyor.7

Example

Örnekte, Melbourne Housing dataset ile çalışacağız. Modelimiz, ev fiyatını tahmin etmek için oda sayısı ve arazi büyüklüğü gibi bilgileri kullanacaktır.

Veri yükleme adımına odaklanmayacağız. Bunun yerine, zaten X_train, X_valid, y_train ve y_valid'de train ve validation verilerine sahip olduğunuz bir noktada olduğunuzu hayal edebilirsiniz.

In [1]:

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Load the data
data = pd.read_csv('../input/melbourne-housing-snapshot/melb_data.csv')

# Select target
y = data.Price

# To keep things simple, we'll use only numerical predictors
melb_predictors = data.drop(['Price'], axis=1)
X = melb_predictors.select_dtypes(exclude=['object'])

# Divide data into training and validation subsets
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size=0.
2,
                                                      random_state=0)
```

Define Function to Measure Quality of Each Approach (Her yaklaşımın kalitesini ölçme yaklaşımı)

Eksik değerlerle başa çıkmada farklı yaklaşımları karşılaştırmak için **score_dataset()** işlevini tanımlarız.

Bu işlev Random Forest modelinden gelen ortalama mutlak hatayı (MAE) bildirir.

In [2]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Function for comparing different approaches
def score_dataset(X_train, X_valid, y_train, y_valid):
    model = RandomForestRegressor(n_estimators=10, random_state=0)
    model.fit(X_train, y_train)
    preds = model.predict(X_valid)
    return mean_absolute_error(y_valid, preds)
```

Score from Approach 1 (Drop Columns with Missing Values)

Hem training hem de validation setleri ile çalıştığımızdan, aynı sütunları her iki DataFrame'de de düşürmeye dikkat ediyoruz.

In [3]:

```python
# Get names of columns with missing values
cols_with_missing = [col for col in X_train.columns
                     if X_train[col].isnull().any()]

# Drop columns in training and validation data
reduced_X_train = X_train.drop(cols_with_missing, axis=1)
reduced_X_valid = X_valid.drop(cols_with_missing, axis=1)

print("MAE from Approach 1 (Drop columns with missing values):")
print(score_dataset(reduced_X_train, reduced_X_valid, y_train, y_valid))
```

```
MAE from Approach 1 (Drop columns with missing values):
183550.22137772635
```

Score from Approach 2 (Imputation)

Daha sonra, eksik değerleri her sütun boyunca ortalama değerle değiştirmek için **SimpleImputer** kullanıyoruz.

Basit olmasına rağmen, ortalama değeri doldurmak genellikle oldukça iyi performans gösterir (ancak bu, veri kümesine göre değişir).

İstatistikçiler, çarpık değerleri belirlemek için daha karmaşık yollar denemiş olsa da (örneğin, **regression imputation** gibi), karmaşık stratejiler, sonuçları karmaşık makine öğrenimi modellerine bağladıktan sonra genellikle ek bir fayda sağlamaz.

In [4]:

```python
from sklearn.impute import SimpleImputer

# Imputation
my_imputer = SimpleImputer()
imputed_X_train = pd.DataFrame(my_imputer.fit_transform(X_train))
imputed_X_valid = pd.DataFrame(my_imputer.transform(X_valid))

# Imputation removed column names; put them back
imputed_X_train.columns = X_train.columns
imputed_X_valid.columns = X_valid.columns

print("MAE from Approach 2 (Imputation):")
print(score_dataset(imputed_X_train, imputed_X_valid, y_train, y_valid))
```

```
MAE from Approach 2 (Imputation):
178166.46269899711
```

Yaklaşım 2'nin, Yaklaşım 1'den daha düşük MAE'ye sahip olduğunu görüyoruz, bu nedenle Yaklaşım 2 bu veri kümesinde daha iyi performans gösterdi.

Score from Approach 3 (An Extension to Imputation)

Ardından, hangi değerlerin atfedildiğini takip ederken eksik değerleri de **impute**(empoze) ediyoruz.

In [5]:
```
# Make copy to avoid changing original data (when imputing)
X_train_plus = X_train.copy()
X_valid_plus = X_valid.copy()

# Make new columns indicating what will be imputed
for col in cols_with_missing:
    X_train_plus[col + '_was_missing'] = X_train_plus[col].isnull()
    X_valid_plus[col + '_was_missing'] = X_valid_plus[col].isnull()

# Imputation
my_imputer = SimpleImputer()
imputed_X_train_plus = pd.DataFrame(my_imputer.fit_transform(X_train_plus))
imputed_X_valid_plus = pd.DataFrame(my_imputer.transform(X_valid_plus))

# Imputation removed column names; put them back
imputed_X_train_plus.columns = X_train_plus.columns
imputed_X_valid_plus.columns = X_valid_plus.columns

print("MAE from Approach 3 (An Extension to Imputation):")
print(score_dataset(imputed_X_train_plus, imputed_X_valid_plus, y_train, y_valid))
```

```
MAE from Approach 3 (An Extension to Imputation):
178927.503183954
```

Gördüğümüz gibi, Yaklaşım 3, Yaklaşım 2'den biraz daha kötü performans gösterdi.

Öyleyse, neden impute edilen sütunlar drop edilenlerden daha iyi performans gösterdi?

Training verisinde 10864 satır ve 12 sütun bulunur; burada üç sütun eksik veriler içerir. Her sütun için girişlerin yarısından azı eksik.

Bu nedenle, sütunları bırakmak çok sayıda yararlı bilgiyi kaldırır ve bu nedenle imputasyonun daha iyi performans göstermesi mantıklıdır.

In [6]:
```
# Shape of training data (num_rows, num_columns)
print(X_train.shape)

# Number of missing values in each column of training data
missing_val_count_by_column = (X_train.isnull().sum())
print(missing_val_count_by_column[missing_val_count_by_column > 0])
```

```
(10864, 12)
Car               49
BuildingArea    5156
YearBuilt       4307
dtype: int64
```

Sonuç

Genel olarak, eksik değerlerin (Yaklaşım 2 ve Yaklaşım 3'te) impute edilmesi, eksik değerlere sahip sütunları (Yaklaşım 1'de) basitçe düşürdüğümüz zamana göre daha iyi sonuçlar verdi.

Exercises (Missing Values)

Şimdi, kayıp değerlerin işlenmesi hakkındaki yeni bilginizi test etme sırası sizde. Muhtemelen büyük bir fark yarattığını göreceksiniz.

Bu alıştırmada, Housing Prices Competition for Kaggle Learn Users verileri ile çalışacaksınız.

[2]:

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the data
X_full = pd.read_csv('../input/train.csv', index_col='Id')
X_test_full = pd.read_csv('../input/test.csv', index_col='Id')

# Remove rows with missing target, separate target from predictors
X_full.dropna(axis=0, subset=['SalePrice'], inplace=True)
y = X_full.SalePrice
X_full.drop(['SalePrice'], axis=1, inplace=True)

# To keep things simple, we'll use only numerical predictors
X = X_full.select_dtypes(exclude=['object'])
X_test = X_test_full.select_dtypes(exclude=['object'])

# Break off validation set from training data
X_train, X_valid, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size=0.2,
                                                      random_state=0)
```

```
X_train.head()
```

Out[3]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	BsmtFinSF2	...
Id											
619	20	90.0	11694	9	5	2007	2007	452.0	48	0	...
871	20	60.0	6600	5	5	1962	1962	0.0	0	0	...
93	30	80.0	13360	5	7	1921	2006	0.0	713	0	...
818	20	NaN	13265	8	5	2002	2002	148.0	1218	0	...
303	20	118.0	13704	7	5	2001	2002	150.0	0	0	...

5 rows × 36 columns

İlk birkaç satırda zaten birkaç eksik değer görebilirsiniz. Bir sonraki adımda, veri kümesindeki eksik değerleri daha kapsamlı bir şekilde anlayacaksınız.

Step 1: Preliminary investigation (Ön Soruşturma)

```python
# Shape of training data (num_rows, num_columns)
print(X_train.shape)

# Number of missing values in each column of training data
missing_val_count_by_column = (X_train.isnull().sum())
print(missing_val_count_by_column[missing_val_count_by_column > 0])
```

```
(1168, 36)
LotFrontage    212
MasVnrArea       6
GarageYrBlt     58
dtype: int64
```

Part A

[6]:

```python
# Fill in the line below: How many rows are in the training data?
num_rows = 1168

# Fill in the line below: How many columns in the training data
# have missing values?
num_cols_with_missing = 3

# Fill in the line below: How many missing entries are contained in
# all of the training data?
tot_missing = 276

# Check your answers
step_1.a.check()
```

Part B

Yukarıdaki cevaplarınızı göz önünde bulundurarak, eksik değerlerle başa çıkmanın en iyi yaklaşımı sizce nedir?

Veri kümesinde çok fazla eksik değer var mı, yoksa sadece birkaç tane mi var? Eksik girdileri olan sütunları tamamen görmezden gelirsek çok fazla bilgi kaybeder miyiz?

Verilerde nispeten az eksik giriş olduğundan (eksik değerlerin en büyük yüzdesine sahip sütun girişlerinin% 20'sinden daha az eksiktir), sütunları bırakmanın iyi sonuçlar vermesi beklenmez. Bunun nedeni, çok sayıda değerli veriyi atacağımızdır ve dolayısıyla imputasyon muhtemelen daha iyi performans gösterecektir.

Eksik değerlerle başa çıkmak için farklı yaklaşımları karşılaştırmak için, tutorial ile aynı **score_dataset()** işlevini kullanırsınız. Bu işlev bir Random Forest modelinden gelen ortalama mutlak hatayı (MAE) bildirir.

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Function for comparing different approaches
def score_dataset(X_train, X_valid, y_train, y_valid):
    model = RandomForestRegressor(n_estimators=100, random_state=0)
    model.fit(X_train, y_train)
    preds = model.predict(X_valid)
    return mean_absolute_error(y_valid, preds)
```

Step 2: Drop columns with missing values (Eksik değer içeren sütunları düşürün)

Bu adımda, eksik değerlere sahip sütunları kaldırmak için X_train ve X_valid'deki verileri önceden işlersiniz. Önceden işlenmiş DataFrames değerini sırasıyla low_X_train ve low_X_valid olarak ayarlayın.

[10]:
```python
# Fill in the line below: get names of columns with missing values
cols_with_missing = [col for col in X_train.columns if X_train[col].isnull().any()] # Your code here

# Fill in the lines below: drop columns in training and validation data
reduced_X_train = X_train.drop(cols_with_missing, axis=1)
reduced_X_valid = X_valid.drop(cols_with_missing, axis=1)

# Check your answers
step_2.check()
```

```python
print("MAE (Drop columns with missing values):")
print(score_dataset(reduced_X_train, reduced_X_valid, y_train, y_valid))
```

```
MAE (Drop columns with missing values):
17837.82570776256
```

Step 3: Imputation

Part A

Her sütundaki eksik değerleri, ortalama değerler ile doldurmak için kod parçasını yazın.
Önceden işlenmiş *DataFrames* değerini *imputed_X_train* ve *imputed_X_valid* olarak ayarlayın.

Sütun adlarının *X_train* ve *X_valid* ile aynı olduğundan emin olun.

```
from sklearn.impute import SimpleImputer

# Fill in the lines below: imputation
my_imputer = SimpleImputer() # Your code here
imputed_X_train = pd.DataFrame(my_imputer.fit_transform(X_train))
imputed_X_valid = pd.DataFrame(my_imputer.transform(X_valid))

# Fill in the lines below: imputation removed column names; put them back
imputed_X_train.columns = X_train.columns
imputed_X_valid.columns = X_valid.columns

# Check your answers
step_3.a.check()
```

Bu yaklaşım için MAE elde etmek için değişiklik olmadan sonraki kod hücresini çalıştırın.

[13]:
```
print("MAE (Imputation):")
print(score_dataset(imputed_X_train, imputed_X_valid, y_train, y_valid))
```

```
MAE (Imputation):
18062.894611872147
```

Part B

Her yaklaşımdan MAE'yi karşılaştırın. Sonuçlar hakkında sizi şaşırtan bir şey var mı? Sizce neden bir yaklaşım diğerinden daha iyi performans gösteriyor?

İpucu: Kayıp değerlerin kaldırılması, impütasyondan daha büyük veya daha küçük bir MAE verdi mi? Bu, öğreticideki kodlama örneğiyle uyumlu mu?

Çözüm: Veri kümesinde çok az eksik değer olduğu düşünüldüğünde, imputasyonun sütunları tamamen düşürmekten daha iyi performans göstermesini bekleriz. Ancak bu durumda, sütunları düşürmenin biraz daha iyi performans gösterdiğini görüyoruz! Bu muhtemelen kısmen veri kümesindeki gürültüye atfedilebilirken, başka bir potansiyel açıklama, imputasyon yönteminin bu veri kümesine mükemmel bir uyumunun olmadığıdır. Yani, ortalama değer ile doldurmak yerine, her eksik değeri 0 değerine ayarlamak, en sık karşılaşılan değeri doldurmak veya başka bir yöntem kullanmak daha mantıklıdır. Örneğin, garajın inşa edildiği yılı gösteren *GarageYrBlt* sütununu düşünün. Bazı durumlarda, eksik bir değerin garajı olmayan bir evi göstermesi muhtemeldir. Bu durumda her bir sütun boyunca medyan değerini doldurmak daha anlamlı mıdır? Veya her sütun boyunca minimum değeri doldurarak daha iyi sonuçlar alabilir miyiz? Bu durumda neyin en iyisi olduğu açık değildir, ancak belki de bazı seçenekleri derhal ekarte edebiliriz - örneğin, bu sütundaki eksik değerlerin 0 olarak ayarlanması büyük olasılıkla korkunç sonuçlar verir!

Step 4: Generate test predictions

Bu son adımda, eksik değerlerle başa çıkmak için seçtiğiniz herhangi bir yaklaşımı kullanacaksınız. Training ve validation özelliklerini önceden işledikten sonra, bir Random Forest modelini eğitir ve değerlendirirsiniz. Ardından, yarışmaya sunulabilecek tahminler oluşturmadan önce test verilerini önceden işlersiniz!

Part A

Training ve validation verilerini önceden işlemek için sonraki kod hücresini kullanın. Önceden işlenmiş DataFrames'i *final_X_train* ve *final_X_valid* olarak ayarlayın. Burada seçtiğiniz herhangi bir yaklaşımı kullanabilirsiniz! bu adımın doğru olarak işaretlenmesi için yalnızca şunlardan emin olmanız gerekir:

- önceden işlenmiş DataFrame'ler aynı sayıda sütuna sahiptir,
- önceden işlenmiş DataFrame'lerde eksik değer yoktur,
- final_X_train ve y_train aynı sayıda satıra sahip olmalıdır,
- final_X_valid ve y_valid aynı sayıda satıra sahip olmalıdır.

```
# Preprocessed training and validation features
final_X_train = reduced_X_train
final_X_valid = reduced_X_valid

# Check your answers
step_4.a.check()
```

Eksik değer içeren sütunları drop işlemine tabi tuttuğumuz durumu seçtik.

Random Forest modelini eğitmek ve değerlendirmek için bir sonraki kod hücresini çalıştırın. (Yukarıdaki score_dataset () işlevini kullanmadığımızı unutmayın, çünkü yakında test tahminleri oluşturmak için eğitimli modeli kullanacağız!)

```
# Define and fit model
model = RandomForestRegressor(n_estimators=100, random_state=0)
model.fit(final_X_train, y_train)

# Get validation predictions and MAE
preds_valid = model.predict(final_X_valid)
print("MAE (Your approach):")
print(mean_absolute_error(y_valid, preds_valid))
```

```
MAE (Your approach):
17837.82570776256
```

Part B

Test verilerinizi önceden işlemek için bir sonraki kod hücresini kullanın. Eğitim ve doğrulama verilerini nasıl önceden işleme koyduğunuzu kabul eden bir yöntem kullandığınızdan emin olun ve önceden işlenmiş test feature'larını `final_X_test` olarak ayarlayın.

Ardından, `preds_test 'içinde test tahminleri oluşturmak için önceden işlenmiş test feature'larını ve eğitimli modeli kullanın.

[63]:
```
#X_train'den düşürdüğümüz kolonları X_test'den de düşürmeliyiz.
final_X_test = X_test.drop(cols_with_missing, axis=1)
```

[69]:
```
#X_test icerisinde hala eksik deger iceren kolonlar mevcut.
#bu eksik degerleri bir sonraki satirda ele alacagiz.
final_miss = [col for col in final_X_test.columns if final_X_test[col].isnull().any()]
final_miss
```

Out[69]
```
['BsmtFinSF1',
 'BsmtFinSF2',
 'BsmtUnfSF',
 'TotalBsmtSF',
 'BsmtFullBath',
 'BsmtHalfBath',
 'GarageCars',
 'GarageArea']
```

[75]:
```
#Eksik degerleri drop etmiyoruz. Cunku X_train ile aynı kolonlara sahip olmalilar.
#Eksik degerleri ortalama degerler ile dolduruyoruz.
final_X_test.fillna(final_X_test[final_miss].mean(), inplace=True)
```

```
# Fill in the line below: preprocess test data
final_X_test

# Fill in the line below: get test predictions
preds_test = model.predict(final_X_test)

step_4.b.check()
```

Correct

149...	Recep Aydoğdu	16592.77...	3	2m

Your Best Entry

You advanced 11,921 places on the leaderboard!
Your submission scored 16592.77974, which is an improvement of your previous score of 20998.83780. Great job!

Tweet this!

Categorical Variables

Bu öğreticide, bu tür verileri işlemek için üç yaklaşımla birlikte kategorik bir değişkenin ne olduğunu öğreneceksiniz.

Introduction

Kategorik bir değişken yalnızca sınırlı sayıda değer alır.

- Ne sıklıkta kahvaltı yaptığınızı soran ve dört seçenek sunan bir anket düşünün: "Asla", "Nadiren", "Çoğu gün" veya "Her gün". Bu durumda, veriler kategoriktir, çünkü yanıtlar sabit bir kategori grubuna girer.
- İnsanlar hangi markaya sahip oldukları ile ilgili bir ankete cevap verselerdi, cevaplar "Honda", "Toyota" ve "Ford" gibi kategorilere girerdi. Bu durumda, veriler de kategoriktir.

Bu değişkenleri Python'daki çoğu makine öğrenimi modeline ilk önce ön işlem yapmadan bağlamaya çalışırsanız bir hata alırsınız.

Bu derste, kategorik verilerinizi hazırlamak için kullanabileceğiniz üç yaklaşımı karşılaştıracağız.

Üç Yaklaşım

1) Drop Categorical Variables

Kategorik değişkenlerle başa çıkmanın en kolay yolu, bunları veri kümesinden basitçe kaldırmaktır.
Bu yaklaşım yalnızca sütunlar yararlı bilgiler içermiyorsa iyi sonuç verecektir.

2) Label Encoding

Label Encoding her benzersiz değeri farklı bir tamsayıya atar.

Breakfast
Every day
Never
Rarely
Most days
Never

Breakfast
3
0
1
2
0

Bu yaklaşım, kategorilerin sıralanmasını varsayar: "Asla" (0) <"Nadiren" (1) <"Çoğu gün" (2) <"Her gün" (3).

Bu varsayım bu örnekte anlamlıdır, çünkü kategorilerde tartışılmaz bir sıralama vardır.

Tüm kategorik değişkenlerin değerlerde açık bir sırası yoktur, ancak **ordinal**(sıralı) değişkenler olarak adlandırılanlara atıfta bulunuruz.

Ağaç tabanlı modeller için (decision tree ve random forest gibi) label encoding’in ordinal değişkenleriyle iyi çalışmasını bekleyebilirsiniz.

3) One-Hot Encoding

One-hot encoding, orijinal verilerdeki her olası değerin varlığını (veya yokluğunu) gösteren yeni sütunlar oluşturur.

Bunu anlamak için bir örnek üzerinde çalışacağız.

Color
Red
Red
Yellow
Green
Yellow

Red	Yellow	Green
1	0	0
1	0	0
0	1	0
0	0	1
0	1	0

Orijinal veri kümesinde "Renk", üç kategoriden oluşan kategorik bir değişkendir: "Kırmızı", "Sarı" ve "Yeşil".
Karşılık gelen one-hot encoding, olası her değer için bir sütun ve orijinal veri kümesindeki her satır için bir satır içerir.
Orijinal değer "Kırmızı" olduğunda, "Kırmızı" sütununa 1 koyarız; orijinal değer "Sarı" ise, "Sarı" sütununa 1 koyarız vb.

Label encoding'in aksine, one-hot encoding kategorilerin sıralanmasını kabul etmez.
Dolayısıyla, kategorik verilerde net bir düzen yoksa (örneğin, "Kırmızı" ne "Sarı" dan daha az veya daha az ise) bu yaklaşımın özellikle iyi çalışmasını bekleyebilirsiniz.
İçsel sıralaması olmayan kategorik değişkenleri **nominal değişkenler** olarak adlandırırız.

One-hot encoding, kategorik değişken çok sayıda değer alıyorsa genellikle iyi performans göstermez (yani, genellikle 15'ten fazla farklı değer alan değişkenler için kullanmazsınız).

Example

Önceki derste olduğu gibi Melbourne Housing dataset üzerinde çalışacağız.

Veri yükleme adımına odaklanmayacağız. Bunun yerine, zaten X_train, X_valid, y_train ve y_valid'de eğitim ve doğrulama verilerine sahip olduğunuz bir noktada olduğunuzu hayal edebilirsiniz.

```python
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the data
data = pd.read_csv('../input/melbourne-housing-snapshot/melb_data.csv')

# Separate target from predictors
y = data.Price
X = data.drop(['Price'], axis=1)

# Divide data into training and validation subsets
X_train_full, X_valid_full, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size
=0.2,
                                                                random_state=0)
```

```python
# Drop columns with missing values (simplest approach)
cols_with_missing = [col for col in X_train_full.columns if X_train_full[col].isnull().any()]
X_train_full.drop(cols_with_missing, axis=1, inplace=True)
X_valid_full.drop(cols_with_missing, axis=1, inplace=True)

# "Cardinality" means the number of unique values in a column
# Select categorical columns with relatively low cardinality (convenient but arbitrary)
low_cardinality_cols = [cname for cname in X_train_full.columns if X_train_full[cname].nunique
() < 10 and
                        X_train_full[cname].dtype == "object"]

# Select numerical columns
numerical_cols = [cname for cname in X_train_full.columns if X_train_full[cname].dtype in ['int
64', 'float64']]

# Keep selected columns only
my_cols = low_cardinality_cols + numerical_cols
X_train = X_train_full[my_cols].copy()
X_valid = X_valid_full[my_cols].copy()
```

In [2]:

```python
X_train.head()
```

Out[2]:

	Type	Method	Regionname	Rooms	Distance	Postcode	Bedroom2	Bathroom	Landsize	Lattitude	Longtitu
12167	u	S	Southern Metropolitan	1	5.0	3182.0	1.0	1.0	0.0	-37.85984	144.986
6524	h	SA	Western Metropolitan	2	8.0	3016.0	2.0	2.0	193.0	-37.85800	144.900
8413	h	S	Western Metropolitan	3	12.6	3020.0	3.0	1.0	555.0	-37.79880	144.822
2919	u	SP	Northern Metropolitan	3	13.0	3046.0	3.0	1.0	265.0	-37.70830	144.915
6043	h	S	Western Metropolitan	3	13.3	3020.0	3.0	1.0	673.0	-37.76230	144.827

Longtitude	Propertycount
144.9867	13240.0
144.9005	6380.0
144.8220	3755.0
144.9158	8870.0
144.8272	4217.0

Ardından, training verilerindeki tüm kategorik değişkenlerin bir listesini elde ederiz.

Bunu, her sütunun veri türünü (veya **dtype**) kontrol ederek yaparız. Dtype **object** bir sütunun metne sahip olduğunu gösterir (teorik olarak olabilecek başka şeyler de vardır, ancak bu bizim amaçlarımız için önemsizdir). Bu veri kümesi için, metin içeren sütunlar kategorik değişkenleri gösterir.

In [3]:

```
# Get list of categorical variables
s = (X_train.dtypes == 'object')
object_cols = list(s[s].index)

print("Categorical variables:")
print(object_cols)
```

```
Categorical variables:
['Type', 'Method', 'Regionname']
```

Define Function to Measure Quality of Each Approach

Kategorik değişkenlerle başa çıkmak için üç farklı yaklaşımı karşılaştırmak için score_dataset () fonksiyonunu tanımlarız.

Bu işlev bir Random Forest modelinden gelen ortalama mutlak hatayı (MAE) döndürür. Genel olarak MAE'nin mümkün olduğunca düşük olmasını istiyoruz!

```python
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Function for comparing different approaches
def score_dataset(X_train, X_valid, y_train, y_valid):
    model = RandomForestRegressor(n_estimators=100, random_state=0)
    model.fit(X_train, y_train)
    preds = model.predict(X_valid)
    return mean_absolute_error(y_valid, preds)
```

Score from Approach 1 (Drop Categorical Variables)

Object sütunlarını select_dtypes () yöntemiyle düşürürüz.

```python
drop_X_train = X_train.select_dtypes(exclude=['object'])
drop_X_valid = X_valid.select_dtypes(exclude=['object'])

print("MAE from Approach 1 (Drop categorical variables):")
print(score_dataset(drop_X_train, drop_X_valid, y_train, y_valid))
```

```
MAE from Approach 1 (Drop categorical variables):
175703.48185157913
```

Score from Approach 2 (Label Encoding)

Scikit-learn, etiket kodlamaları almak için kullanılabilecek bir *LabelEncoder* sınıfına sahiptir.

Kategorik değişkenler üzerinde döngü yapar ve Label Encoding'i her sütuna ayrı ayrı uygularız.

```python
from sklearn.preprocessing import LabelEncoder

# Make copy to avoid changing original data
label_X_train = X_train.copy()
label_X_valid = X_valid.copy()

# Apply label encoder to each column with categorical data
label_encoder = LabelEncoder()
for col in object_cols:
    label_X_train[col] = label_encoder.fit_transform(X_train[col])
    label_X_valid[col] = label_encoder.transform(X_valid[col])

print("MAE from Approach 2 (Label Encoding):")
print(score_dataset(label_X_train, label_X_valid, y_train, y_valid))
```

```
MAE from Approach 2 (Label Encoding):
165936.40548390493
```

Yukarıdaki kod hücresinde, her sütun için, her benzersiz değeri rastgele farklı bir tamsayıya atarız.
Bu, özel etiketler sağlamaktan daha basit olan yaygın bir yaklaşımdır; ancak, tüm sıralı değişkenler için daha iyi bilgilendirilmiş etiketler sağlarsak, performansta ek bir artış bekleyebiliriz.

Score from Approach 3 (One-Hot Encoding)

Scikit-learn'un OneHotEncoder sınıfını, one-hot encoding yapmak için kullanıyoruz. Davranışını özelleştirmek için kullanılabilecek bir dizi parametre vardır.

- Validation verileri, training verilerinde gösterilmeyen sınıflar içerdiğinde hataları önlemek için *handle_unknown = 'ignore'* ayarını yaparız ve
- *sparse = False*, kodlanmış sütunların sayısal bir dizi olarak döndürülmesini sağlar (seyrek bir matris yerine).

Encoder'ı kullanmak için yalnızca one-hot encoded olmasını istediğimiz kategorik sütunları sağlıyoruz.
Örneğin, training verilerini encode için **X_train[object_cols]** 'u sağlıyoruz. (aşağıdaki kod hücresindeki *object_cols*, kategorik verileri olan sütun adlarının bir listesidir ve bu nedenle **X_train[object_cols]**, eğitim kümesindeki tüm kategorik verileri içerir.)

```python
from sklearn.preprocessing import OneHotEncoder

# Apply one-hot encoder to each column with categorical data
OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
OH_cols_train = pd.DataFrame(OH_encoder.fit_transform(X_train[object_cols]))
OH_cols_valid = pd.DataFrame(OH_encoder.transform(X_valid[object_cols]))

# One-hot encoding removed index; put it back
OH_cols_train.index = X_train.index
OH_cols_valid.index = X_valid.index

# Remove categorical columns (will replace with one-hot encoding)
num_X_train = X_train.drop(object_cols, axis=1)
num_X_valid = X_valid.drop(object_cols, axis=1)

# Add one-hot encoded columns to numerical features
OH_X_train = pd.concat([num_X_train, OH_cols_train], axis=1)
OH_X_valid = pd.concat([num_X_valid, OH_cols_valid], axis=1)

print("MAE from Approach 3 (One-Hot Encoding):")
print(score_dataset(OH_X_train, OH_X_valid, y_train, y_valid))
```

```
MAE from Approach 3 (One-Hot Encoding):
166089.4893009678
```

En iyi yaklaşım hangisi?

Bu durumda, kategorik sütunları bırakmak (Yaklaşım 1) en kötü performansı gösterdi, çünkü en yüksek MAE puanına sahipti.

Diğer iki yaklaşıma gelince, geri dönen MAE puanları çok yakın olduğundan, birinin diğerine karşı anlamlı bir faydası görünmemektedir.

Genel olarak, **one-hot encoding** (Yaklaşım 3) tipik olarak en iyi performansı gösterir ve kategorik sütunları düşürmek (Yaklaşım 1) genellikle en kötü performansı gösterir, ancak duruma göre değişir.

Sonuç

Dünya kategorik verilerle doludur. Bu ortak veri türünü nasıl kullanacağınızı biliyorsanız çok daha etkili bir veri bilimcisi olacaksınız!

Exercises: Categorical Variables

Kategorik değişkenleri encode ederek şimdiye kadarki en iyi sonucu elde edeceksiniz!

Bu alıştırmada Housing Prices Competition for Kaggle Learn Users ile çalışacağız.

X_train, X_valid, y_train ve y_valid'e training ve validation setlerini yüklemek için bir sonraki kod hücresini değiştirmeden çalıştırın. Test seti X_test'e yüklenir.

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the data
X = pd.read_csv('../input/train.csv', index_col='Id')
X_test = pd.read_csv('../input/test.csv', index_col='Id')

# Remove rows with missing target, separate target from predictors
X.dropna(axis=0, subset=['SalePrice'], inplace=True)
y = X.SalePrice
X.drop(['SalePrice'], axis=1, inplace=True)

# To keep things simple, we'll drop columns with missing values
cols_with_missing = [col for col in X.columns if X[col].isnull().any()]
X.drop(cols_with_missing, axis=1, inplace=True)
X_test.drop(cols_with_missing, axis=1, inplace=True)

# Break off validation set from training data
X_train, X_valid, y_train, y_valid = train_test_split(X, y,
                                                      train_size=0.8, test_size=0.2,
                                                      random_state=0)
```

```
X_train.head()
```

Out[4]:

	MSSubClass	MSZoning	LotArea	Street	LotShape	LandContour	Utilities	LotConfig	LandSlope	Neighborhood	...
Id											
619	20	RL	11694	Pave	Reg	Lvl	AllPub	Inside	Gtl	NridgHt	...
871	20	RL	6600	Pave	Reg	Lvl	AllPub	Inside	Gtl	NAmes	...
93	30	RL	13360	Pave	IR1	HLS	AllPub	Inside	Gtl	Crawfor	...
818	20	RL	13265	Pave	IR1	Lvl	AllPub	CulDSac	Gtl	Mitchel	...
303	20	RL	13704	Pave	IR1	Lvl	AllPub	Corner	Gtl	CollgCr	...

5 rows × 60 columns

...	OpenPorchSF	EnclosedPorch	3SsnPorch	ScreenPorch	PoolArea	MiscVal	MoSold	YrSold	SaleType	SaleCondition
...	108	0	0	260	0	0	7	2007	New	Partial
...	0	0	0	0	0	0	8	2009	WD	Normal
...	0	44	0	0	0	0	8	2009	WD	Normal
...	59	0	0	0	0	0	7	2008	WD	Normal
...	81	0	0	0	0	0	1	2006	WD	Normal

Veri kümesinin hem sayısal hem de kategorik değişkenler içerdiğine dikkat edin. Bir modeli eğitmeden önce kategorik verileri encode işlemine tabi tutmanız gerekir.

Farklı modelleri karşılaştırmak için tutorial'daki ile aynı score_dataset () işlevini kullanırsınız. Bu işlev bir random forest modelinden gelen ortalama mutlak hatayı (MAE) bildirir.

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# function for comparing different approaches
def score_dataset(X_train, X_valid, y_train, y_valid):
    model = RandomForestRegressor(n_estimators=100, random_state=0)
    model.fit(X_train, y_train)
    preds = model.predict(X_valid)
    return mean_absolute_error(y_valid, preds)
```

Step 1: Drop columns with categorical data

En basit yaklaşımla başlayacaksınız. Kategorik veriler içeren sütunları kaldırmak için X_train ve X_valid'deki verileri önceden işlemek için aşağıdaki kod hücresini kullanın. Önceden işlenmiş DataFrames değerini sırasıyla drop_X_train ve drop_X_valid olarak ayarlayın.

```
# Fill in the lines below: drop columns in training and validation data
drop_X_train = X_train.select_dtypes(exclude=["object"])
drop_X_valid = X_valid.select_dtypes(exclude=["object"])

# Check your answers
step_1.check()
```

Bu yaklaşım için MAE hesaplayalım.

```
print("MAE from Approach 1 (Drop categorical variables):")
print(score_dataset(drop_X_train, drop_X_valid, y_train, y_valid))
```

```
MAE from Approach 1 (Drop categorical variables):
17837.82570776256
```

Step 2: Label Encoding

Label Encoding'e geçmeden önce veri kümesini araştıracağız. Özellikle, "Condition2" sütununa bakacağız. Aşağıdaki kod hücresi, hem eğitim hem de doğrulama kümelerindeki benzersiz girişleri yazdırır.

```
print("Unique values in 'Condition2' column in training data:", X_train['Condition2'].unique())
print("\nUnique values in 'Condition2' column in validation data:", X_valid['Condition2'].unique())
```

```
Unique values in 'Condition2' column in training data: ['Norm' 'PosA' 'Feedr' 'PosN' 'Artery' 'RRAe']

Unique values in 'Condition2' column in validation data: ['Norm' 'RRAn' 'RRNn' 'Artery' 'Feedr' 'PosN']
```

Şimdi buna göre kod yazarsanız:

- label encoder'i training data'ya fit ederseniz, ve sonra
- hem training hem validation verilerini transform yaparsanız,

bir hata alırsınız. Durumun neden böyle olduğunu görebiliyor musunuz? (_Bu soruyu cevaplamak için yukarıdaki çıktıyı kullanmanız gerekir._)

Validation verilerinde görünen ancak training verilerinde olmayan değerler var mı?

Çözüm: Training verilerindeki bir sütuna label encoding uygulanması, training verilerinde görünen her bir benzersiz değer için karşılık gelen tamsayı değerli bir etiket oluşturur. Validation verilerinin training verilerinde de görünmeyen değerler içermesi durumunda, kodlayıcı bir hata atar, çünkü bu değerlerde kendilerine atanan bir tamsayı olmaz.

Validation verilerindeki "Condition2" sütununun 'RRAn' ve 'RRNn' değerlerini içerdiğine dikkat edin, ancak bunlar eğitim verilerinde görünmez - bu nedenle, scikit-learn ile bir etiket kodlayıcı kullanmaya çalışırsak, kodu hata verir.

Bu gerçek dünyadaki verilerde karşılaşacağınız yaygın bir sorundur ve bu sorunu düzeltmek için birçok yaklaşım vardır. Örneğin, yeni kategorilerle ilgilenmek için özel bir Label Encoder yazabilirsiniz. Ancak en basit yaklaşım, sorunlu kategorik sütunları düşürmektir.

Sorunlu sütunları *bad_label_cols* Python listesine kaydetmek için aşağıdaki kod hücresini çalıştırın. Benzer şekilde, güvenli bir şekilde etiketlenebilen sütunlar *good_label_cols* içinde saklanır.

```
# All categorical columns
object_cols = [col for col in X_train.columns if X_train[col].dtype == "object"]

# Columns that can be safely label encoded
good_label_cols = [col for col in object_cols if
                   set(X_train[col]) == set(X_valid[col])]

# Problematic columns that will be dropped from the dataset
bad_label_cols = list(set(object_cols)-set(good_label_cols))

print('Categorical columns that will be label encoded:', good_label_cols)
print('\nCategorical columns that will be dropped from the dataset:', bad_label_cols)
```

```
Categorical columns that will be label encoded: ['MSZoning', 'Street', 'LotShape', 'LandContour', 'LotConfi
g', 'BldgType', 'HouseStyle', 'ExterQual', 'CentralAir', 'KitchenQual', 'PavedDrive', 'SaleCondition']

Categorical columns that will be dropped from the dataset: ['Neighborhood', 'Exterior2nd', 'Exterior1st', 'F
unctional', 'SaleType', 'Foundation', 'ExterCond', 'Condition1', 'RoofMatl', 'Utilities', 'Heating', 'RoofSt
yle', 'HeatingQC', 'LandSlope', 'Condition2']
```

X_train ve X_valid içindeki verilere label encode yapmak için sonraki kod hücresini kullanın. Önceden işlenmiş DataFrames değerini sırasıyla label_X_train ve label_X_valid olarak ayarlayın.

- Kategorik sütunları veri kümesinden bad_label_cols içine çekmek için aşağıdaki kodu sağladık.
- Kategorik sütunlar içinden good_label_cols'lara label encode uygulamanız gerekir.

```
from sklearn.preprocessing import LabelEncoder

# Drop categorical columns that will not be encoded
label_X_train = X_train.drop(bad_label_cols, axis=1)
label_X_valid = X_valid.drop(bad_label_cols, axis=1)

# Apply label encoder
label_encoder = LabelEncoder()

for col in good_label_cols:
    label_X_train[col] = label_encoder.fit_transform(X_train[col])
    label_X_valid[col] = label_encoder.transform(X_valid[col])     # Your code here

# Check your answer
step_2.b.check()
```

```
print("MAE from Approach 2 (Label Encoding):")
print(score_dataset(label_X_train, label_X_valid, y_train, y_valid))
```

```
MAE from Approach 2 (Label Encoding):
17575.291883561644
```

Step 3: Investigating Cardinality (Kardinalite Araştırması)

Şimdiye kadar, kategorik değişkenlerle başa çıkmak için iki farklı yaklaşım denediniz. Ve kategorik verileri kodlamanın, sütunları veri kümesinden kaldırmaktan daha iyi sonuçlar verdiğini gördünüz.

Yakında, one-hot encoding deneyeceksiniz. O zamandan önce, ele almamız gereken bir konu daha var. Bir sonraki kod hücresini değişiklik olmadan çalıştırarak başlayın.

```
# Get number of unique entries in each column with categorical data
object_nunique = list(map(lambda col: X_train[col].nunique(), object_cols))
d = dict(zip(object_cols, object_nunique))

# Print number of unique entries by column, in ascending order
sorted(d.items(), key=lambda x: x[1])
```

```
[('Street', 2),
 ('Utilities', 2),
 ('CentralAir', 2),
 ('LandSlope', 3),
 ('PavedDrive', 3),
 ('LotShape', 4),
 ('LandContour', 4),
 ('ExterQual', 4),
 ('KitchenQual', 4),
 ('MSZoning', 5),
 ('LotConfig', 5),
 ('BldgType', 5),
 ('ExterCond', 5),
 ('HeatingQC', 5),
 ('Condition2', 6),
 ('RoofStyle', 6),
 ('Foundation', 6),
 ('Heating', 6),
 ('Functional', 6),
 ('SaleCondition', 6),
 ('RoofMatl', 7),
 ('HouseStyle', 8),
 ('Condition1', 9),
 ('SaleType', 9),
 ('Exterior1st', 15),
 ('Exterior2nd', 16),
 ('Neighborhood', 25)]
```

Yukarıdaki çıktı, kategorik verilere sahip her sütun için sütundaki benzersiz değerlerin sayısını gösterir. Örneğin, training verilerindeki Street sütununun iki benzersiz değeri vardır: sırasıyla bir çakıl yol ve asfalt bir yola karşılık gelen 'Grvl' ve 'Pave'.

Kategorik bir değişkenin benzersiz girişlerinin sayısını, o kategorik değişkenin temel niteliği olarak ifade ederiz. Örneğin, 'Street' değişkeni 2 kardinaliteye sahiptir.

Aşağıdaki soruları cevaplamak için yukarıdaki çıktıyı kullanın.

```
# Fill in the line below: How many categorical variables in the training data
# have cardinality greater than 10?
high_cardinality_numcols = 3

# Fill in the line below: How many columns are needed to one-hot encode the
# 'Neighborhood' variable in the training data?
num_cols_neighborhood = 25

# Check your answers
step_3.a.check()
```

Birçok satıra sahip büyük veri kümeleri için, one-hot encoding, veri kümesinin boyutunu büyük ölçüde genişletebilir. Bu nedenle, yalnızca tipik olarak nispeten düşük kardinaliteye sahip sütunlara one-hot encoding uygulayacağız. Daha sonra, yüksek kardinalite sütunları veri kümesinden kaldırılabilir veya label encoding kullanabiliriz.

Örnek olarak, 10.000 satır içeren ve 100 benzersiz giriş içeren bir kategorik sütun içeren bir veri kümesini düşünün.

- Bu sütun karşılık gelen one-hot encoding ile değiştirilirse, veri kümesine kaç giriş eklenir?
- Bunun yerine sütunu label encoding ile değiştirirsek, kaç giriş eklenir?

Aşağıdaki satırları doldurmak için cevaplarınızı kullanın.

one-hot encoding yoluyla veri kümesine kaç girdi eklendiğini hesaplamak için, kategorik değişkeni kodlamak için kaç girdinin gerekli olduğunu hesaplayarak başlayın (satır sayısını one-hot encoding'deki sütun sayısıyla çarparak). Ardından, veri kümesine kaç girdi eklendiğini öğrenmek için, orijinal sütundaki girdi sayısını çıkarın.

```
# Fill in the line below: How many entries are added to the dataset by
# replacing the column with a one-hot encoding?
OH_entries_added = 1e4*100 - 1e4

# Fill in the line below: How many entries are added to the dataset by
# replacing the column with a label encoding?
label_entries_added = 0

# Check your answers
step_3.b.check()
```

Çözüm Açıklaması: Elinizde 100 unique değeri olan 10000 tane kolonunuz var.
One Hot Encoding her unique kolon değeri için yeni kolon oluşturulması anlamına geliyordu. Buradaki 10e4 aslında 10000 anlamına gelir. (e=exponential yani 10 üzeri 4)
Bu yüzden de one hot encoding'te 10000 kolonumuz zaten vardı. 100 tane

unique entrymiz olduğu için 10000 * 100 -10000 (zaten elimizde 10000 başta vardı o yüzden çıkardık) kolon eklenecektir.

Label Encode ise her unique değer için bir sayı verilmesi demektir burada kolon eklenmez sadece var olan kolonlara sayı değerleri yazılır. O yüzden eklenecek kolon sayısı 0'dır.

Step 4: one-hot encoding

Bu adımda, one-hot encoding deneyeceksiniz. Ancak, veri kümesindeki tüm kategorik değişkenleri kodlamak yerine, kardinalitesi 10'dan az olan sütunlar için yalnızca one-hot encoding oluşturacaksınız.

Low_cardinality_cols değerini one-hot encoding uygulanacak sütunları içeren bir Python listesine ayarlamak için aşağıdaki kod hücresini değiştirmeden çalıştırın. Benzer şekilde, high_cardinality_cols, veri kümesinden bırakılacak kategorik sütunların bir listesini içerir.

```
# Columns that will be one-hot encoded
low_cardinality_cols = [col for col in object_cols if X_train[col].nunique() < 10]

# Columns that will be dropped from the dataset
high_cardinality_cols = list(set(object_cols)-set(low_cardinality_cols))

print('Categorical columns that will be one-hot encoded:', low_cardinality_cols)
print('\nCategorical columns that will be dropped from the dataset:', high_cardinality_cols)
```

```
Categorical columns that will be one-hot encoded: ['MSZoning', 'Street', 'LotShape', 'LandContour', 'Utiliti
es', 'LotConfig', 'LandSlope', 'Condition1', 'Condition2', 'BldgType', 'HouseStyle', 'RoofStyle', 'RoofMat
l', 'ExterQual', 'ExterCond', 'Foundation', 'Heating', 'HeatingQC', 'CentralAir', 'KitchenQual', 'Functiona
l', 'PavedDrive', 'SaleType', 'SaleCondition']

Categorical columns that will be dropped from the dataset: ['Neighborhood', 'Exterior2nd', 'Exterior1st']
```

X_train ve X_valid içindeki verilere one-hot encoding yapmak için sonraki kod hücresini kullanın. Önceden işlenmiş DataFrames değerini sırasıyla OH_X_train ve OH_X_valid olarak ayarlayın.

- Veri kümesindeki kategorik sütunların tam listesi Python listesi object_cols içinde bulunabilir.
- yalnızca Low_cardinality_cols içindeki kategorik sütunlara one-hot encoding uygulanmalı. Diğer tüm kategorik sütunlar veri kümesinden çıkarılmalıdır.

One-hot encoding’i sırasıyla X_train [low_cardinality_cols] ve X_valid [low_cardinality_cols] içindeki eğitim ve doğrulama verilerindeki düşük kardinalite sütunlarına uygulayarak başlayın.

```python
from sklearn.preprocessing import OneHotEncoder

# Apply one-hot encoder to each column with categorical data
OH_encoder = OneHotEncoder(handle_unknown='ignore', sparse=False)
OH_cols_train = pd.DataFrame(OH_encoder.fit_transform(X_train[low_cardinality_cols]))
OH_cols_valid = pd.DataFrame(OH_encoder.transform(X_valid[low_cardinality_cols]))

# One-hot encoding removed index; put it back
OH_cols_train.index = X_train.index
OH_cols_valid.index = X_valid.index

# Remove categorical columns (will replace with one-hot encoding)
num_X_train = X_train.drop(object_cols, axis=1)
num_X_valid = X_valid.drop(object_cols, axis=1)

# Add one-hot encoded columns to numerical features
OH_X_train = pd.concat([num_X_train, OH_cols_train], axis=1)
OH_X_valid = pd.concat([num_X_valid, OH_cols_valid], axis=1)

# Check your answer
step_4.check()
```

```python
print("MAE from Approach 3 (One-Hot Encoding):")
print(score_dataset(OH_X_train, OH_X_valid, y_train, y_valid))
```

```
MAE from Approach 3 (One-Hot Encoding):
17525.345719178084
```

Step 5: Generate test predictions and submit your results

4. Adım'ı tamamladıktan sonra, sonuçlarınızı skor tablosuna göndermek için öğrendiklerinizi kullanmak isterseniz, tahminler oluşturmadan önce test verilerini önceden işlemeniz gerekir.

Pipelines

Bu öğreticide, modelleme kodunuzu temizlemek için **pipeline'ı** nasıl kullanacağınızı öğreneceksiniz.

Introduction

Pipeline'lar, veri önişleme ve modelleme kodunuzu düzenli tutmanın basit bir yoludur. Özellikle, bir ardışık düzen ön işleme ve modelleme adımlarını bir araya getirir, böylece tüm paketi tek bir adımmış gibi kullanabilirsiniz.

Birçok veri bilimcisi modelleri pipeline kullanmadan bir araya getirmektedir, ancak pipeline'nın bazı önemli faydaları vardır. Bunlar arasında:

- **Temiz Kod**: Ön işlemenin her adımındaki verilerin muhasebeleştirilmesi dağınık olabilir. Bir ardışık düzen ile, her adımda egzersiz ve doğrulama verilerinizi manuel olarak takip etmeniz gerekmez.
- **Daha Az Hata**: Bir adımı yanlış uygulama veya bir önişleme adımını unutmak için daha az fırsat vardır.
- **Üretim için Kolaylık**: Bir modeli bir prototipten ölçekte konuşlandırılabilir bir şeye geçirmek şaşırtıcı derecede zor olabilir. Burada birçok ilgili kaygıya girmeyeceğiz, ancak pipeline yardımcı olabilir.
- **Model Validation için Daha Fazla Seçenek**: Bir sonraki öğreticide cross validation'u kapsayan bir örnek göreceksiniz.

Example

Önceki derste olduğu gibi, Melbourne Housing dataset ile çalışacağız.

Veri yükleme adımına odaklanmayacağız. Bunun yerine, X_train, X_valid, y_train ve y_valid'de zaten eğitim ve doğrulama verilerine sahip olduğunuz bir noktada olduğunuzu hayal edebilirsiniz.

```python
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the data
data = pd.read_csv('../input/melbourne-housing-snapshot/melb_data.csv')

# Separate target from predictors
y = data.Price
X = data.drop(['Price'], axis=1)

# Divide data into training and validation subsets
X_train_full, X_valid_full, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size
=0.2,
                                                                random_state=0)

# "Cardinality" means the number of unique values in a column
# Select categorical columns with relatively low cardinality (convenient but arbitrary)
categorical_cols = [cname for cname in X_train_full.columns if X_train_full[cname].nunique() <
10 and
                        X_train_full[cname].dtype == "object"]

# Select numerical columns
numerical_cols = [cname for cname in X_train_full.columns if X_train_full[cname].dtype in ['int
64', 'float64']]

# Keep selected columns only
my_cols = categorical_cols + numerical_cols
X_train = X_train_full[my_cols].copy()
X_valid = X_valid_full[my_cols].copy()
```

Aşağıdaki head () yöntemiyle eğitim verilerine bir göz atın. Verilerin hem kategorik veriler hem de eksik değerleri olan sütunlar içerdiğine dikkat edin. Bir pipeline ile her ikisiyle de başa çıkmak kolay!

In [2]:

```
X_train.head()
```

Out[2]:

	Type	Method	Regionname	Rooms	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea
12167	u	S	Southern Metropolitan	1	5.0	3182.0	1.0	1.0	1.0	0.0	NaN
6524	h	SA	Western Metropolitan	2	8.0	3016.0	2.0	2.0	1.0	193.0	NaN
8413	h	S	Western Metropolitan	3	12.6	3020.0	3.0	1.0	1.0	555.0	NaN
2919	u	SP	Northern Metropolitan	3	13.0	3046.0	3.0	1.0	1.0	265.0	NaN
6043	h	S	Western Metropolitan	3	13.3	3020.0	3.0	1.0	2.0	673.0	673.0

Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt	Lattitude	Longtitude	Propertycount
5.0	3182.0	1.0	1.0	1.0	0.0	NaN	1940.0	-37.85984	144.9867	13240.0
8.0	3016.0	2.0	2.0	1.0	193.0	NaN	NaN	-37.85800	144.9005	6380.0
12.6	3020.0	3.0	1.0	1.0	555.0	NaN	NaN	-37.79880	144.8220	3755.0
13.0	3046.0	3.0	1.0	1.0	265.0	NaN	1995.0	-37.70830	144.9158	8870.0
13.3	3020.0	3.0	1.0	2.0	673.0	673.0	1970.0	-37.76230	144.8272	4217.0

Pipeline'nın tamamını üç adımda inşa ediyoruz.

Step 1: Önişleme Adımlarını Tanımlayın

Bir pipeline'nın ön işleme ve modelleme adımlarını nasıl bir araya getirdiğine benzer şekilde, farklı önişleme adımlarını bir araya getirmek için *ColumnTransformer* sınıfını kullanırız.

Aşağıdaki kod:

- **sayısal** verilerdeki eksik değerleri ifade eder ve
- eksik değerleri ifade eder ve **kategorik** verilere one-hot encoding uygular.

```
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder

# Preprocessing for numerical data
numerical_transformer = SimpleImputer(strategy='constant')

# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])
```

Step 2: Modeli tanımlayın

Ardından, tanıdık RandomForestRegressor sınıfıyla bir Random Forest modeli tanımlarız.

```
In [4]:
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=100, random_state=0)
```

Step 3: Pipeline Oluşturun ve Değerlendirin

Son olarak, ön işleme ve modelleme adımlarını bir araya getiren bir pipeline tanımlamak için *Pipeline* sınıfını kullanırız. Dikkat edilmesi gereken birkaç önemli nokta vardır:

- Pipeline ile, eğitim verilerini önceden işler ve modeli tek bir kod satırına sığdırırız.
  (Aksine, bir pipeline olmadan, ayrı adımlarla imputing, one-hot encoding ve model eğitimi yapmak zorundayız. Hem sayısal hem de kategorik değişkenlerle uğraşmak zorunda kalırsak bu özellikle dağınık hale gelir!)
- Pipeline ile, işlenmemiş özellikleri X_valid'te predict () komutuna sağlarız ve boru hattı, tahminler oluşturmadan önce özellikleri otomatik olarak ön işleme tabi tutar. (Ancak, bir ardışık düzen olmadan, tahminlerde bulunmadan önce doğrulama verilerini önceden işlemeyi hatırlamamız gerekir.)

In [5]:

```
from sklearn.metrics import mean_absolute_error

# Bundle preprocessing and modeling code in a pipeline
my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('model', model)
                             ])

# Preprocessing of training data, fit model
my_pipeline.fit(X_train, y_train)

# Preprocessing of validation data, get predictions
preds = my_pipeline.predict(X_valid)

# Evaluate the model
score = mean_absolute_error(y_valid, preds)
print('MAE:', score)
```

```
MAE: 160679.18917034855
```

Sonuç

Pipeline'lar, makine öğrenmesi kodunu temizlemek ve hatalardan kaçınmak için değerlidir ve özellikle sofistike veri önişlemeli iş akışları için yararlıdır.

Exercise: Pipelines

Bu alıştırmada, makine öğrenme kodunuzun verimliliğini artırmak için **pipeline** kullanacaksınız.

Çalışmamızda Housing Prices Competition for Kaggle Learn Users datasetini kullanacağız.

X_train, X_valid, y_train ve y_valid'e eğitim ve doğrulama kümelerini yüklemek için bir sonraki kod hücresini değiştirmeden çalıştırın. Test seti X_test'e yüklenir.

In [2]:

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the data
X_full = pd.read_csv('../input/train.csv', index_col='Id')
X_test_full = pd.read_csv('../input/test.csv', index_col='Id')

# Remove rows with missing target, separate target from predictors
X_full.dropna(axis=0, subset=['SalePrice'], inplace=True)
y = X_full.SalePrice
X_full.drop(['SalePrice'], axis=1, inplace=True)

# Break off validation set from training data
X_train_full, X_valid_full, y_train, y_valid = train_test_split(X_full, y,
                                                                train_size=0.8, test_size=0.2,
                                                                random_state=0)

# "Cardinality" means the number of unique values in a column
# Select categorical columns with relatively low cardinality (convenient but arbitrary)
categorical_cols = [cname for cname in X_train_full.columns if
                    X_train_full[cname].nunique() < 10 and
                    X_train_full[cname].dtype == "object"]

# Select numerical columns
numerical_cols = [cname for cname in X_train_full.columns if
                X_train_full[cname].dtype in ['int64', 'float64']]

# Keep selected columns only
my_cols = categorical_cols + numerical_cols
X_train = X_train_full[my_cols].copy()
X_valid = X_valid_full[my_cols].copy()
X_test = X_test_full[my_cols].copy()
```

In [3]:

```
X_train.head()
```

Out[3]:

	MSZoning	Street	Alley	LotShape	LandContour	Utilities	LotConfig	LandSlope	Condition1	Condition2	...	Gar
Id												
619	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	Norm	Norm	...	774
871	RL	Pave	NaN	Reg	Lvl	AllPub	Inside	Gtl	PosN	Norm	...	308
93	RL	Pave	Grvl	IR1	HLS	AllPub	Inside	Gtl	Norm	Norm	...	432
818	RL	Pave	NaN	IR1	Lvl	AllPub	CulDSac	Gtl	Norm	Norm	...	857
303	RL	Pave	NaN	IR1	Lvl	AllPub	Corner	Gtl	Norm	Norm	...	843

5 rows × 76 columns

Bir sonraki kod hücresi, verileri önceden işlemek ve bir modeli eğitmek için tutorial'ın kodunu kullanır. Bu kodu değişiklik yapmadan çalıştırın.

In [4]:

```python
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error

# Preprocessing for numerical data
numerical_transformer = SimpleImputer(strategy='constant')

# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

# Define model
model = RandomForestRegressor(n_estimators=100, random_state=0)

# Bundle preprocessing and modeling code in a pipeline
clf = Pipeline(steps=[('preprocessor', preprocessor),
                      ('model', model)
                     ])

# Preprocessing of training data, fit model
clf.fit(X_train, y_train)

# Preprocessing of validation data, get predictions
preds = clf.predict(X_valid)

print('MAE:', mean_absolute_error(y_valid, preds))
```

```
MAE: 17861.780102739725
```

Kod, ortalama mutlak hata (MAE) için 17862 civarında bir değer verir. Bir sonraki adımda, daha iyisini yapmak için kodu değiştireceksiniz.

Step 1: Performansı Arttırın

Part A

Şimdi senin sıran! Aşağıdaki kod hücresinde, kendi önişleme adımlarınızı ve Random Forest modelinizi tanımlayın. Aşağıdaki değişkenler için değerleri girin:

- *numerical_transformer*
- *categorical_transformer*
- *model*

Egzersizin bu kısmını geçmek için, sadece geçerli önişleme adımlarını ve Random Forest modelini tanımlamanız gerekir.

In [5]:
```
# Preprocessing for numerical data
numerical_transformer = SimpleImputer(strategy="median") # Your code here

# Preprocessing for categorical data
categorical_transformer = Pipeline(steps=[
                          ("imputer", SimpleImputer(strategy="constant")),
                          ("onehot", OneHotEncoder(handle_unknown="ignore"))]) # Your code here

# Bundle preprocessing for numerical and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_cols),
        ('cat', categorical_transformer, categorical_cols)
    ])

# Define model
model = RandomForestRegressor(n_estimators=100, random_state=0) # Your code here

# Check your answer
step_1.a.check()
```

İpucu: Bu soruna birçok farklı potansiyel çözüm olsa da, yalnızca *column_transformer*'ı varsayılan değerden değiştirerek tatmin edici sonuçlar elde ettik - özellikle, eksik değerlerin nasıl uygulanacağına karar veren *strategy* parametresini değiştirdik.

Part B

Bu adımı geçmek için, Part A'da, yukarıdaki koddan daha düşük MAE elde eden bir pipeline tanımlamanız gerekir.
Burada zaman ayırıp MAE'yi ne kadar düşük alabileceğinizi görmek için birçok farklı yaklaşımı denemeniz önerilir! (Kodunuz geçmezse, lütfen ön işleme adımlarını ve modelini Part A'da değiştirin.)

In [7]:

```
# Bundle preprocessing and modeling code in a pipeline
my_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                              ('model', model)
                             ])

# Preprocessing of training data, fit model
my_pipeline.fit(X_train, y_train)

# Preprocessing of validation data, get predictions
preds = my_pipeline.predict(X_valid)

# Evaluate the model
score = mean_absolute_error(y_valid, preds)
print('MAE:', score)

# Check your answer
step_1.b.check()
```

```
MAE: 17487.872363013696
```

Correct

İpucu: Daha iyi performans elde etmek için önişleme adımlarının ve modelinin nasıl değiştirileceği hakkında bazı fikirler almak için lütfen Part A'nın ipucuna bakın.

Step 2: Test Tahminleri Oluşturun

Şimdi, test verileriyle tahminler oluşturmak için eğitimli modelinizi kullanacaksınız.

In [9]:

```
# Preprocessing of test data, fit model
preds_test = my_pipeline.predict(X_test) # Your code here

# Check your answer
step_2.check()
```

In [11]:

```
# Save test predictions to file
output = pd.DataFrame({'Id': X_test.index,
                       'SalePrice': preds_test})
output.to_csv('submission.csv', index=False)
```

125...	Recep Aydoğdu		16475.69...	4	18m

Your Best Entry

Your submission scored 16475.69973, which is an improvement of your previous score of 16592.77974. Great job!

Tweet this!

Cross-Validation

Bu tutorial'da, daha iyi model performansı ölçümleri için **cross-validation'un** nasıl kullanılacağını öğreneceksiniz.

Introduction

Makine öğrenmesi yinelemeli(iterative) bir süreçtir.

Hangi öngörücü değişkenlerin kullanılacağı, hangi tür modellerin kullanılacağı, bu modellere hangi argümanların sağlanacağı vb. ile ilgili seçeneklerle karşılaşacaksınız.

Şimdiye kadar, bir validation (veya holdout) seti ile model kalitesini ölçerek bu seçimleri veriye dayalı bir şekilde yaptınız.

Ancak bu yaklaşımın bazı dezavantajları vardır.
Bunu görmek için 5000 sıralı bir veri kümeniz olduğunu hayal edin.
Tipik olarak verilerin yaklaşık % 20'sini veya 1000 satırını validation veri kümesi olarak tutacaktır.
Ancak bu, model puanlarının belirlenmesini rastgele bir şekilde şansa bırakır.
Yani, bir model farklı bir 1000 satırda yanlış olsa bile başka 1000 satırlık bir sette iyi olabilir.

Genel olarak, validation seti ne kadar büyük olursa, model kalitesi ölçümüzde o kadar az rastgelelik ("gürültü") olur ve o kadar güvenilir olur.

Ne yazık ki, yalnızca training verilerimizdeki satırları kaldırarak büyük bir validation kümesi alabiliriz ve daha küçük training veri setleri daha kötü modeller anlamına gelir!

Cross-Validation Nedir?

Cross-Validation'da, model kalitesinin birden fazla ölçüsünü almak için modelleme sürecimizi verilerin farklı alt kümelerinde çalıştırıyoruz.

Örneğin, verileri her biri tam veri kümesinin % 20'si olan 5 parçaya bölerek başlayabiliriz. Bu durumda, verileri 5 "**fold**"'a ayırdığımızı söylüyoruz.

Ardından, her fold için bir deneme gerçekleştiriyoruz:

- Deney 1'de, ilk foldu bir validation (veya holdout) kümesi ve diğer hepsini training verileri olarak kullanıyoruz. Bu bize % 20'lik bir holdout(dağıtım) setine dayanan bir model kalitesi ölçüsü verir.
- Deney 2'de, ikinci fold'daki verileri tutarız (ve ikinci fold dışındaki her şeyi modeli eğitmek için kullanırız). Daha sonra holdout(dağıtım) seti, model kalitesinin ikinci bir tahminini almak için kullanılır.
- Her fold'u holdout(dağıtım) seti olarak bir kez kullanarak bu işlemi tekrarlıyoruz. Bunları bir araya getirerek, verilerin % 100'ü bir noktada holdout olarak kullanılır ve veri kümesindeki tüm satırlara dayanan bir model kalitesi ölçüsü elde ederiz (tüm satırları aynı anda kullanmasak bile) .

Ne Zaman Cross-Validation Kullanmalıyız?

Cross-Validation, model kalitesinin daha doğru bir ölçümünü verir, bu da çok fazla modelleme kararı verirseniz özellikle önemlidir.

Bununla birlikte, birden fazla modeli tahmin ettiğinden (her fold için bir tane) tahmin edilmesi daha uzun sürebilir.

Peki, bu ödünleşmeler göz önüne alındığında, her bir yaklaşımı ne zaman kullanmalısınız?

- Fazladan hesaplama yükünün çok önemli olmadığı küçük veri kümeleri için cross-validation yapmalısınız.
- Daha büyük veri kümeleri için tek bir validation kümesi yeterlidir. Kodunuz daha hızlı çalışacaktır.

Büyük ve küçük veri kümesini oluşturan şey için basit bir eşik yoktur. Ancak modelinizin çalışması birkaç dakika veya daha az sürüyorsa, muhtemelen cross-validation'a geçmeye değer.

Alternatif olarak, cross-validation'ı çalıştırabilir ve her deney için puanların yakın olup olmadığını gözlemleyebilirsiniz.

Her deney aynı sonuçları verirse, tek bir validation seti muhtemelen yeterlidir.

Example

Önceki derslerdeki verilerle çalışacağız. Input verilerini X'e, Output verilerini y'ye yüklüyoruz.

In [1]:
```
import pandas as pd

# Read the data
data = pd.read_csv('../input/melbourne-housing-snapshot/melb_data.csv')

# Select subset of predictors
cols_to_use = ['Rooms', 'Distance', 'Landsize', 'BuildingArea', 'YearBuilt']
X = data[cols_to_use]

# Select target
y = data.Price
```

Ardından, eksik değerleri doldurmak için bir imputer ve tahminler yapmak için bir Random Forest modeli kullanan Pipeline tanımlarız.

Pipeline olmadan cross-validation yapmak mümkün olsa da, oldukça zor! Bir pipeline kullanmak, kodu oldukça basit hale getirecektir.

In [2]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

my_pipeline = Pipeline(steps=[('preprocessor', SimpleImputer()),
                              ('model', RandomForestRegressor(n_estimators=50,
                                                              random_state=0))
                             ])
```

Scikit-learn'dan *cross_val_score()* işleviyle cross-validation skorlarını elde ederiz. Fold sayısını cv parametresi ile ayarladık.

In [3]:

```
from sklearn.model_selection import cross_val_score

# Multiply by -1 since sklearn calculates *negative* MAE
scores = -1 * cross_val_score(my_pipeline, X, y,
                              cv=5,
                              scoring='neg_mean_absolute_error')

print("MAE scores:\n", scores)
```

```
MAE scores:
 [301628.7893587  303164.4782723  287298.331666   236061.84754543
 260383.45111427]
```

scoring parametresi, raporlama için bir model kalitesi ölçüsü seçer: bu durumda negatif ortalama mutlak hata (MAE) seçtik. (https://scikit-learn.org/stable/modules/model_evaluation.html)

Negatif MAE'yi belirtmemiz biraz şaşırtıcı. Scikit-learn, tüm metriklerin tanımlandığı bir kurala sahiptir, bu nedenle yüksek bir sayı daha iyidir. Negatif MAE neredeyse başka bir yerde duyulmamış olsa da, negatifleri burada kullanmak bu kuralla tutarlı olmalarını sağlar.

Alternatif modelleri karşılaştırmak için genellikle tek bir model kalitesi ölçüsü istiyoruz. Bu yüzden deneyler boyunca ortalamayı alıyoruz.

In [4]:

```
print("Average MAE score (across experiments):")
print(scores.mean())
```

```
Average MAE score (across experiments):
277707.3795913405
```

Sonuç

Cross-validation kullanılması, kodumuzu temizlemenin sağladığı ek avantajla birlikte model kalitesinin çok daha iyi bir ölçüsünü verir: artık ayrı eğitim ve doğrulama setlerini takip etmemize gerek olmadığını unutmayın. Bu nedenle, özellikle küçük veri kümeleri için bu iyi bir gelişme!

Exercise: Cross-Validation

Bu alıştırmada, bir makine öğrenme modelini **cross-validation** ile ayarlamak için öğrendiklerinizden yararlanacaksınız.

[Housing Prices Competition for Kaggle Learn Users](#) veri seti ile çalışacağız.

_train, X_valid, y_train ve y_valid'e eğitim ve doğrulama kümelerini yüklemek için bir sonraki kod hücresini değiştirmeden çalıştırın. Test seti X_test'e yüklenir.

Basit olması için kategorik değişkenleri düşürüyoruz.

In [2]:

```
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the data
train_data = pd.read_csv('../input/train.csv', index_col='Id')
test_data = pd.read_csv('../input/test.csv', index_col='Id')

# Remove rows with missing target, separate target from predictors
train_data.dropna(axis=0, subset=['SalePrice'], inplace=True)
y = train_data.SalePrice
train_data.drop(['SalePrice'], axis=1, inplace=True)

# Select numeric columns only
numeric_cols = [cname for cname in train_data.columns if train_data[cname].dtype in ['int64',
'float64']]
X = train_data[numeric_cols].copy()
X_test = test_data[numeric_cols].copy()
```

In [3]:

```
X.head()
```

Out[3]:

	MSSubClass	LotFrontage	LotArea	OverallQual	OverallCond	YearBuilt	YearRemodAdd	MasVnrArea	BsmtFinSF1	Bsı
Id										
1	60	65.0	8450	7	5	2003	2003	196.0	706	0
2	20	80.0	9600	6	8	1976	1976	0.0	978	0
3	60	68.0	11250	7	5	2001	2002	162.0	486	0
4	70	60.0	9550	7	5	1915	1970	0.0	216	0
5	60	84.0	14260	8	5	2000	2000	350.0	655	0

5 rows × 36 columns

Şimdiye kadar, scikit-learn ile pipeline'ların nasıl kurulacağını öğrendiniz.

Örneğin, aşağıdaki pipeline, tahminler yapmak üzere bir Random Forest modeli eğitmek için *RandomForestRegressor ()* kullanmadan önce verilerdeki eksik değerleri değiştirmek için *SimpleImputer()* kullanır.

Random Forest modelindeki ağaç sayısını *n_estimators* parametresi ile ayarladık ve *random_state* ayarı tekrarlanabilirliği sağlıyor.

In [4]:

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer

my_pipeline = Pipeline(steps=[
    ('preprocessor', SimpleImputer()),
    ('model', RandomForestRegressor(n_estimators=50, random_state=0))
])
```

Cross-validation'da pipeline'ların nasıl kullanılacağını da öğrendiniz. Aşağıdaki kod, beş farklı fold arasında ortalaması alınmış ortalama mutlak hatayı (MAE) elde etmek için *cross_val_score()* işlevini kullanır.

Fold sayısını *cv* parametresi ile ayarladığımızı hatırlayın.

In [5]:

```
from sklearn.model_selection import cross_val_score

# Multiply by -1 since sklearn calculates *negative* MAE
scores = -1 * cross_val_score(my_pipeline, X, y,
                              cv=5,
                              scoring='neg_mean_absolute_error')

print("Average MAE score:", scores.mean())
```

```
Average MAE score: 18276.410356164386
```

Step 1: Write a Usefull Function

Bu alıştırmada, bir makine öğrenimi modeli için parametreleri seçmek üzere cross validation kullanacaksınız.

Aşağıdakileri kullanan bir makine öğrenimi pipeline'nın MAE ortalamalarını bildiren (3 fold olacak) bir get_score () işlevi yazarak başlayın:

- kıvrımlar oluşturmak için X ve y'deki veriler,
- Eksik değerleri değiştirmek için *SimpleImputer()* (tüm parametreler varsayılan olarak bırakılmıştır) ve
- Random Forest modelin fit etmek için RandomForestRegressor () (random_state = 0 ile).

Get_score() öğesine sağlanan *n_estimators* parametresi, Random Forest modelindeki ağaç sayısı ayarlanırken kullanılır.

In [6]:
```
def get_score(n_estimators):
    my_pipeline = Pipeline(steps = [("preprocessor", SimpleImputer()),
                                    ("model", RandomForestRegressor(n_estimators, random_state=
0))
                                    ])
    scores = -1 * cross_val_score(my_pipeline, X, y, cv=3, scoring="neg_mean_absolute_error")

    return scores.mean()

# Check your answer
step_1.check()
```

İpucu: *Pipeline* sınıfıyla bir pipeline yaparak başlayın. RandomForestRegressor () içindeki n_estimators değerini get_score işlevine sağlanan bağımsız değişkene ayarladığınızdan emin olun. Ardından, her fold için MAE'yi almak için *cross_val_score()* kullanın ve ortalamayı alın. *Cv* parametresi üzerinden fold sayısını üçe ayarladığınızdan emin olun.

Step 2: Test Different Parameter Values

Şimdi Random Forest'daki ağaç sayısı için sekiz farklı değere karşılık gelen model performansını değerlendirmek için, Adım 1'de tanımladığınız işlevi kullanacaksınız: 50, 100, 150, ..., 300, 350, 400.

Sonuçlarınızı bir Python dictionary olan *results*'da saklayın; burada *results[i], get_score(i)* tarafından döndürülen ortalama MAE'dir.

In [8]:

```
results = {}

for i in range(1,9):
    results[50*i] = get_score(50*i)
# Check your answer
step_2.check()
```

In [9]:

```
results
```

Out[9]:

```
{50: 18353.8393511688,
 100: 18395.2151680032,
 150: 18288.730020956387,
 200: 18248.345889801505,
 250: 18255.26922247291,
 300: 18275.241922621914,
 350: 18270.29183308043,
 400: 18270.197974402367}
```

Step 3: Find the Best Parameter Value

In [11]:

```
import matplotlib.pyplot as plt
%matplotlib inline

plt.plot(list(results.keys()), list(results.values()))
plt.show()
```

Sonuçlar göz önüne alındığında, *n_estimators* için hangi değer Random Forest modeli için en iyisi olarak görünüyor? Cevabınızı *n_estimators_best* değerini ayarlamak için kullanın.

In [12]:

```
n_estimators_best = min(results, key=results.get)

# Check your answer
step_3.check()
```

Bu alıştırmada, bir makine öğrenme modelinde uygun parametreleri seçmek için bir yöntem araştırdınız.

hyperparameter optimization hakkında daha fazla bilgi edinmek isterseniz, bir makine öğrenimi modeli için en iyi parametre kombinasyonunu belirlemek için basit bir yöntem olan **Grid Search** ile başlamanız önerilir. Neyse ki, scikit-learn, Grid Search kodunuzu çok verimli hale getirebilen yerleşik bir işlev olan GridSearchCV() içerir!

Çeşitli veri kümelerinde son teknoloji sonuçlar elde eden güçlü bir teknik olan **gradient_boosting** hakkında bilgi edinmeye devam edin.

XGBoost

Structured veriler için en doğru sonuçları veren modelleme tekniği.

Bu bölümde, **gradient boosting** modellerinin nasıl oluşturulacağını ve optimize edileceğini öğreneceksiniz.

Bu yöntem birçok Kaggle yarışmasında liderdir ve çeşitli veri kümelerinde ustalık derecesinde sonuçlar elde eder.

Introduction

Bu kursun çoğu bölümünde, birçok Decision Tree'nin tahminlerini ortalayarak tek bir Decision Tree'den daha iyi performans elde eden Random Forest yöntemiyle tahminler yaptınız.

Random Forest yöntemini "**ensemble method** (topluluk yöntemi)" olarak adlandırıyoruz.

Tanıma göre, ensemble(topluluk) metodları birkaç modelin tahminlerini birleştirir (örneğin, Random Forest durumunda birkaç ağaç).

Şimdi, gradient boosting adı verilen başka bir topluluk yöntemini öğreneceğiz.

Gradient Boosting

Gradient Boosting, bir ensemble(topluluk)'a tekrarlanan modelleri eklemek için döngülerden geçen bir yöntemdir.

Topluluğun tahminleri oldukça saf olabilen tek bir modelle başlatılmasıyla başlar. (Tahminleri çılgınca yanlış olsa bile, topluluğa daha sonraki eklemeler bu hataları ele alacaktır.)

Sonra döngüye başlıyoruz:

- İlk olarak, veri grubundaki her bir gözlem için tahminler oluşturmak üzere mevcut topluluğu kullanıyoruz. Bir tahmin yapmak için, topluluktaki tüm modellerden tahminleri ekliyoruz.
- Bu tahminler bir loss(kayıp) fonksiyonunu hesaplamak için kullanılır (örneğin mean squared error gibi).
- Daha sonra, loss fonksiyonunu topluluğa eklenecek yeni bir modele uyacak şekilde kullanıyoruz. Özellikle, model parametrelerini belirleriz, böylece bu yeni modeli topluluğa eklemek kaybı azaltır. (Yan not: "gradient boosting" içindeki "gradyan", bu yeni modeldeki parametreleri belirlemek için loss fonksiyonunda *gradient descent* kullanacağımız anlamına gelir.)
- Son olarak, topluluğa yeni modeli ekliyoruz ve ...
- ... Tekrar!!

Example

Eğitim ve doğrulama verilerini X_train, X_valid, y_train ve y_valid'e yükleyerek başlıyoruz.

In [1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the data
data = pd.read_csv('../input/melbourne-housing-snapshot/melb_data.csv')

# Select subset of predictors
cols_to_use = ['Rooms', 'Distance', 'Landsize', 'BuildingArea', 'YearBuilt']
X = data[cols_to_use]

# Select target
y = data.Price

# Separate data into training and validation sets
X_train, X_valid, y_train, y_valid = train_test_split(X, y)
```

Bu örnekte, XGBoost kütüphanesi ile çalışacaksınız. **XGBoost, extreme gradient boosting** (aşırı eğim yükseltme) anlamına gelir. Bu, performans ve hıza odaklanan çeşitli ek özelliklerle bir gradient boosting uygulamasıdır. (Scikit-learn'de gradient boosting'in başka bir versiyonu vardır, ancak XGBoost'un bazı teknik avantajları vardır.)

Bir sonraki kod hücresinde, XGBoost (*xgboost.XGBRegressor*) için scikit-learn API'sini içe aktarıyoruz.

Bu, tıpkı scikit-learn'de yaptığımız gibi bir model oluşturmamıza ve fit etmemize olanak tanır.

Çıktıda göreceğiniz gibi, *XGBRegressor* sınıfının birçok ayarlanabilir parametresi vardır - yakında bunları öğreneceksiniz!

In [2]:

```
from xgboost import XGBRegressor

my_model = XGBRegressor()
my_model.fit(X_train, y_train)
```

```
/opt/conda/lib/python3.6/site-packages/xgboost/core.py:587: FutureWarning: Serie
s.base is deprecated and will be removed in a future version
  if getattr(data, 'base', None) is not None and \


[13:37:05] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear i
s now deprecated in favor of reg:squarederror.
```

Out[2]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

Ayrıca tahminlerde bulunur ve modeli değerlendiririz.

In [3]:

```
from sklearn.metrics import mean_absolute_error

predictions = my_model.predict(X_valid)
print("Mean Absolute Error: " + str(mean_absolute_error(predictions, y_valid)))
```

```
Mean Absolute Error: 280355.04334039026
```

Parameter Tuning (Parametre Ayarı)

XGBoost, doğruluğu ve eğitim hızını önemli ölçüde etkileyebilecek birkaç parametreye sahiptir.

Anlamanız gereken ilk parametreler:

n_estimators

n_estimators, yukarıda açıklanan modelleme döngüsünden kaç kez geçileceğini belirler.
Topluluğa dahil ettiğimiz model sayısına eşittir.

- Çok düşük bir değer *underfitting'e* neden olur, bu da hem eğitim verileri hem de test verileri üzerinde yanlış tahminlere yol açar.
- Çok yüksek bir değer, *overfitting'e* neden olur, bu da eğitim verileri üzerinde doğru tahminlere neden olur, ancak test verileri üzerinde yanlış tahminler yapar (bu bizim için önemli olan şeydir).

Tipik değerler 100-1000 arasındadır, ancak bu aşağıda tartışılan *learning_rate* parametresine çok bağlıdır.

Topluluktaki model sayısını ayarlamak için kod:

In [4]:
```
my_model = XGBRegressor(n_estimators=500)
my_model.fit(X_train, y_train)
```

early_stopping_rounds

early_stopping_rounds, *n_estimators* için ideal değeri otomatik olarak bulmanın bir yolunu sunar.
Early, *n_estimators* için durmak zorunda olmamamıza rağmen, doğrulama skoru iyileşmeyi bıraktığında modelin yinelemeyi durdurmasına neden olur.

n_estimators için yüksek bir değer ayarlamak ve ardından yinelemeyi durdurmak için en uygun zamanı bulmak için *early_stopping_rounds* kullanmak akıllıcadır.

İşi şansa bırakmanın bazen validation puanlarının iyileşmediği tek bir round'a denk geldiğinde döngüyü durdurmaması için, durmadan önce kaç tane doğrusal bozulma round'una izin vereceğinizi bir sayı belirtmeniz gerekir.
early_stopping_rounds = 5 ayarı makul bir seçimdir.
Bu durumda, 5 doğrusal round boyunca kötüleşen doğrulama skorundan sonra duruyoruz.

Early_stopping_rounds kullanırken, validation puanlarını hesaplamak için bazı verileri de ayırmanız gerekir - bu, *eval_set* parametresini ayarlayarak yapılır.

Yukarıda yazdığımız kod örneğini, early stopping rounds'u içerecek şekilde değiştirebiliriz:

In [5]:
```
my_model = XGBRegressor(n_estimators=500)
my_model.fit(X_train, y_train,
             early_stopping_rounds=5,
             eval_set=[(X_valid, y_valid)],
             verbose=False)
```

```
[13:37:07] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear i
s now deprecated in favor of reg:squarederror.
```

Out[5]:
```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.1, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=500,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

Daha sonra tüm verilerinizle bir model fit etmek istiyorsanız, *n_estimators*'ı early stopping ile çalıştırdığınızda en uygun bulduğunuz değere ayarlayın. Bu örneğimizde 500 bulunmuş.

learning_rate

Her bileşen modelinden tahminleri toplayarak tahminler almak yerine, eklemeden önce her modelden gelen tahminleri küçük bir sayı ile (**learning rate** olarak bilinir) çarpabiliriz.

Bu, topluluğa eklediğimiz her ağacın bize daha az yardımcı olduğu anlamına gelir.
Bu nedenle, *n_estimators* için *overfitting* olmadan daha yüksek bir değer ayarlayabiliriz.
Early stopping kullanırsak, uygun sayıda ağaç otomatik olarak belirlenir.

Genel olarak, küçük bir learning rate ve çok sayıda tahminci ağaç daha doğru XGBoost modelleri verecektir, ancak döngü boyunca daha fazla yineleme yaptığı için modelin eğitilmesi daha uzun sürecektir.

Varsayılan olarak, XGBoost *learning_rate = 0.1* değerini ayarlar.

Learning rate'i değiştirmek için yukarıdaki örneği değiştirelim:

In [6]:

```
my_model = XGBRegressor(n_estimators=1000, learning_rate=0.05)
my_model.fit(X_train, y_train,
             early_stopping_rounds=5,
             eval_set=[(X_valid, y_valid)],
             verbose=False)
```

```
[13:37:08] WARNING: /workspace/src/objective/regression_obj.cu:152: reg:linear i
s now deprecated in favor of reg:squarederror.
```

Out[6]:

```
XGBRegressor(base_score=0.5, booster='gbtree', colsample_bylevel=1,
             colsample_bynode=1, colsample_bytree=1, gamma=0,
             importance_type='gain', learning_rate=0.05, max_delta_step=0,
             max_depth=3, min_child_weight=1, missing=None, n_estimators=1000,
             n_jobs=1, nthread=None, objective='reg:linear', random_state=0,
             reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
             silent=None, subsample=1, verbosity=1)
```

n_jobs

Çalışma zamanının dikkate alındığı daha büyük veri kümelerinde, modellerinizi daha hızlı oluşturmak için paralellism (paralellik) kullanabilirsiniz.

n_jobs parametresini makinenizdeki çekirdek sayısına eşit olarak ayarlamak yaygındır.
Daha küçük veri kümelerinde bu pek yardımcı olmaz.

Ortaya çıkan model daha iyi olmayacaktır, bu nedenle uygun zaman için mikro optimizasyon genellikle dikkat dağıtıcı bir şey değildir. Ancak, fit komutu sırasında uzun süre bekleyeceğiniz büyük veri kümelerinde yararlıdır.

Değiştirilmiş örnek:

In [7]:

```
my_model = XGBRegressor(n_estimators=1000, learning_rate=0.05, n_jobs=4)
my_model.fit(X_train, y_train,
             early_stopping_rounds=5,
             eval_set=[(X_valid, y_valid)],
             verbose=False)
```

Sonuç

XGBoost, standart tablo halindeki verilerle (görüntü ve video gibi daha egzotik veri türlerinin aksine Pandas DataFrames'da depoladığınız veri türü) çalışmak için önde gelen bir yazılım kütüphanesidir.

Dikkatli parameter tuning ile son derece hassas modelleri eğitebilirsiniz.

Exercise: XGBoost

Bu alıştırmada, yeni bilgilerinizi **gradient boosting** modeli eğitmek için kullanacaksınız.

Housing Prices Competition for Kaggle Learn Users veri seti üzerinde çalışacağız.

X_train, X_valid, y_train ve y_valid'e eğitim ve doğrulama kümelerini yüklemek için bir sonraki kod hücresini değiştirmeden çalıştırın. Test seti X_test'e yüklenir.

```
In [2]:
import pandas as pd
from sklearn.model_selection import train_test_split

# Read the data
X = pd.read_csv('../input/train.csv', index_col='Id')
X_test_full = pd.read_csv('../input/test.csv', index_col='Id')

# Remove rows with missing target, separate target from predictors
X.dropna(axis=0, subset=['SalePrice'], inplace=True)
y = X.SalePrice
X.drop(['SalePrice'], axis=1, inplace=True)

# Break off validation set from training data
X_train_full, X_valid_full, y_train, y_valid = train_test_split(X, y, train_size=0.8, test_size
=0.2,
                                                                random_state=0)

# "Cardinality" means the number of unique values in a column
# Select categorical columns with relatively low cardinality (convenient but arbitrary)
low_cardinality_cols = [cname for cname in X_train_full.columns if X_train_full[cname].nunique
() < 10 and
                        X_train_full[cname].dtype == "object"]

# Select numeric columns
numeric_cols = [cname for cname in X_train_full.columns if X_train_full[cname].dtype in ['int6
4', 'float64']]

# Keep selected columns only
my_cols = low_cardinality_cols + numeric_cols
X_train = X_train_full[my_cols].copy()
X_valid = X_valid_full[my_cols].copy()
X_test = X_test_full[my_cols].copy()
```

```
# One-hot encode the data (to shorten the code, we use pandas)
X_train = pd.get_dummies(X_train)
X_valid = pd.get_dummies(X_valid)
X_test = pd.get_dummies(X_test)
X_train, X_valid = X_train.align(X_valid, join='left', axis=1)
X_train, X_test = X_train.align(X_test, join='left', axis=1)
```

Step 1: Model Oluşturun

Bu adımda, gradient boosting ile ilk modelinizi oluşturacak ve eğiteceksiniz.

- My_model_1 öğesini bir **XGBoost** modeline ayarlayarak başlayın. XGBRegressor sınıfını kullanın ve random seed'i 0 olarak ayarlayın (*random_state = 0*). Diğer tüm parametreleri varsayılan olarak bırakın.
- X_train ve y_train ile modelinizi fit edin.

```
In [3]:
from xgboost import XGBRegressor

# Define the model
my_model_1 = XGBRegressor(random_state=0) # Your code here

# Fit the model
my_model_1.fit(X_train, y_train) # Your code here

# Check your answer
step_1.a.check()
```

Modelin validation verileri için tahminlerini *predictions_1*'de tutun. Validation verilerinin *X_valid*'de saklandığını hatırlayın.

```
In [5]:
from sklearn.metrics import mean_absolute_error

# Get predictions
predictions_1 = my_model_1.predict(X_valid) # Your code here

# Check your answer
step_1.b.check()
```

Son olarak, validation verilerinin tahminlerine karşılık gelen ortalama mutlak hatayı (MAE) hesaplamak için *mean_absolute_error ()* işlevini kullanın. Validation verilerinin doğru sonuçlarının y_valid içinde saklandığını unutmayın.

```
In [7]:
# Calculate MAE
mae_1 = mean_absolute_error(y_valid, predictions_1) # Your code here

# Uncomment to print MAE
print("Mean Absolute Error:" , mae_1)

# Check your answer
step_1.c.check()

Mean Absolute Error: 17662.736729452055
```

Step 2: Modelinizi İyileştirin

Artık varsayılan bir modeli temel olarak eğittiğinize göre, daha iyi performans elde edip edemeyeceğinizi görmek için parametreleri değiştirmenin zamanı geldi!

- XGBRegressor sınıfını kullanarak my_model_2 öğesini bir XGBoost modeline ayarlayarak başlayın. Daha iyi sonuçlar almak için varsayılan parametreleri (n_estimators ve learning_rate gibi) nasıl değiştireceğinizi öğrenmek için önceki bölümde öğrendiklerinizi kullanın.
- Ardından, modeli X_train ve y_train'deki training verileri ile fit edin.
- Modelin validation verileri için tahminlerini *predictions_2'*de tutun. Validation verilerinin *X_valid'*de saklandığını hatırlayın.
- Son olarak, validation verilerinin tahminlerine karşılık gelen ortalama mutlak hatayı (MAE) hesaplamak için *mean_absolute_error ()* işlevini kullanın. Validation verilerinin doğru sonuçlarının y_valid içinde saklandığını unutmayın.

```
In [9]:
# Define the model
my_model_2 = XGBRegressor(n_estimators=500, learning_rate=0.05) # Your code here

# Fit the model
my_model_2.fit(X_train, y_train) # Your code here

# Get predictions
predictions_2 = my_model_2.predict(X_valid) # Your code here

# Calculate MAE
mae_2 = mean_absolute_error(y_valid, predictions_2) # Your code here

# Uncomment to print MAE
print("Mean Absolute Error:" , mae_2)

# Check your answer
step_2.check()

Mean Absolute Error: 16728.27523009418
```

Step 3: Modeli Kırın

Bu adımda, 1. Adımdaki orijinal modelden daha kötü performans gösteren bir model oluşturacaksınız. Bu, parametreleri nasıl ayarlayacağınıza dair sezginizi geliştirmenize yardımcı olacaktır.

Kazara daha iyi performans elde ettiğinizi bile görebilirsiniz, bu da sonuçta değerli bir öğrenme deneyimi!

In [11]:

```
# Define the model
my_model_3 = XGBRegressor(n_estimators=500, learning_rate=1)

# Fit the model
my_model_3.fit(X_train, y_train) # Your code here

# Get predictions
predictions_3 = my_model_3.predict(X_valid)

# Calculate MAE
mae_3 = mean_absolute_error(y_valid, predictions_3)

# Uncomment to print MAE
print("Mean Absolute Error:" , mae_3)

# Check your answer
step_3.check()
```

```
Mean Absolute Error: 27386.61764233733
```

Data Leakage (Veri Sızıntısı)

Bu bölümde, **Data Leakage**(Veri Sızıntısı)'nın ne olduğunu ve nasıl önleneceğini öğreneceksiniz.
Bunu nasıl önleyeceğinizi bilmiyorsanız, sızıntı sık sık ortaya çıkacak ve modellerinizi ince ve tehlikeli yollarla mahvedecektir.
Bu, veri bilimcilerin uygulamaları için en önemli kavramlardan biridir.

Introduction

Data Leakage (veri sızıntısı), training verileriniz target hakkında bilgi içerdiğinde gerçekleşir, ancak model tahmin için kullanıldığında benzer veriler kullanılamaz.

Bu, training setinde (ve hatta muhtemelen validation verilerinde) yüksek performansa yol açar, ancak model üretimde kötü performans gösterecektir.

Başka bir deyişle, sızıntı, bir modelle karar vermeye başlayana kadar bir modelin doğru görünmesine neden olur ve sonra model çok yanlış bir hale gelir.

İki ana sızıntı türü vardır: **target leakage** ve **train-test contamination**.

Target Leakage

Target Leakage (Hedef Sızıntısı), öngörücüleriniz, tahmin yaptığınız sırada kullanılamayacak veriler içerdiğinde ortaya çıkar.

Target Leakage'ı, yalnızca bir özelliğin iyi tahminlerde bulunmasına yardımcı olup olmadığı değil, verilerin kullanılabilir hale geldiği zamanlama veya kronolojik sıraya göre düşünmek önemlidir.

Bir örnek anlamamıza yardımcı olacaktır. Pneumonia ile kimin hastalanacağını tahmin etmek istediğinizi düşünün. Ham verilerinizin ilk birkaç satırı şöyle görünür:

got_pneumonia	age	weight	male	took_antibiotic_medicine	...
False	65	100	False	False	...
False	72	130	True	False	...
True	58	100	False	True	...

İnsanlar pnömoni olduktan sonra iyileşmek için antibiyotik ilaçlar alırlar. Ham veriler, bu sütunlar arasında güçlü bir ilişki olduğunu gösterir, ancak *got_pneumonia* değeri belirlendikten sonra *took_antibiotic_medicine* sıklıkla değiştirilir. Bu target leakage'dır.

Model, *took_antibiotic_medicine* için *False* değerine sahip olan herkesin pnömonisi olmadığını görecektir. Validation verileri training verileriyle aynı kaynaktan geldiğinden, pattern, validation'da kendini tekrar edecektir ve modelin büyük validation (veya cross-validation'da) puanları olacaktır.

Ancak, model gerçek dünyada kullanıma geçtiğinde çok büyük yanlışlar yapacaktır. Çünkü pnömoni olan hastalar tedaviye başlamadan önce antibiyotik almamış olacaktır.

Bu tür veri sızıntısını önlemek için, hedef değer gerçekleştikten sonra güncellenen (veya oluşturulan) değişkenler hariç tutulmalıdır.

Train-Test Contamination

Training verilerini, validation verilerinden ayırmaya dikkat etmediğinizde farklı bir sızıntı türü oluşur.

Validation'ın modelin daha önce dikkate almadığı veriler üzerinde nasıl bir performans gösterdiğini hatırlayın. Validation verileri preprocessing davranışını etkiliyorsa bu işlemi ince yollarla bozabilirsiniz. Buna bazen **train-test contamination** denir.

Örneğin, *train_test_split ()* öğesini çağırmadan önce önişleme yaptığınızı (eksik değerler için imputer kullanmak gibi) düşünün. Sonuç ne oldu? Modeliniz iyi validation puanları alabilir, bu da size büyük güven verir, ancak karar vermek için uyguladığınızda düşük performans gösterir.

Doğrulamanız basit bir train-test split'e dayanıyorsa, validation verilerini preprocessing adımlarının uygulanması da dahil olmak üzere her tür fitting işleminden hariç tutun.
Scikit-learn pipeline'i kullanıyorsanız bu daha kolaydır. Cross-validation kullanırken, preproccesing'i pipeline içinde yapmanız daha da önemlidir!

Example

Bu örnekte, target leakage'ı tespit etmenin ve kaldırmanın bir yolunu öğreneceksiniz.

Kredi kartı uygulamaları hakkında bir veri kümesi kullanacağız. Sonuç olarak, her kredi kartı uygulaması hakkındaki bilgi bir *X* dataframe'inde saklanır. Bir *y* serisini de hangi uygulamaların kabul edildiğini tahmin etmek için kullanacağız.

```
In [1]:
import pandas as pd

# Read the data
data = pd.read_csv('../input/aer-credit-card-data/AER_credit_card_data.csv',
                   true_values = ['yes'], false_values = ['no'])

# Select target
y = data.card

# Select predictors
X = data.drop(['card'], axis=1)

print("Number of rows in the dataset:", X.shape[0])
X.head()
```

```
Number of rows in the dataset: 1319
```

Out[1]:

	reports	age	income	share	expenditure	owner	selfemp	dependents	months	majorcards
0	0	37.66667	4.5200	0.033270	124.983300	True	False	3	54	1
1	0	33.25000	2.4200	0.005217	9.854167	False	False	3	34	1
2	0	33.66667	4.5000	0.004156	15.000000	True	False	4	58	1
3	0	30.50000	2.5400	0.065214	137.869200	False	False	0	25	1
4	0	32.16667	9.7867	0.067051	546.503300	True	False	2	64	1

Bu küçük bir veri kümesi olduğundan, model kalitesinin doğru ölçümlerini sağlamak için çapraz doğrulamayı kullanacağız.

```
In [2]:
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# Since there is no preprocessing, we don't need a pipeline (used anyway as best prac
tice!)
my_pipeline = make_pipeline(RandomForestClassifier(n_estimators=100))
cv_scores = cross_val_score(my_pipeline, X, y,
                            cv=5,
                            scoring='accuracy')

print("Cross-validation accuracy: %f" % cv_scores.mean())
```

```
Cross-validation accuracy: 0.979525
```

Deneyim kazandıkça, % 98 doğruluk veren modeller bulmanın çok nadir olduğunu göreceksiniz.
Bu olur, ancak verileri target leakage açısından daha yakından incelemeliyiz.

Data sekmesi altında da bulabileceğiniz verilerin bir özeti:

card: Kredi başvurusu kabul edilirse 1, edilmezse 0

reports: Başlıca küçültücü raporların sayısı.(Kredi kabulunu etkiler.)

age: n(yaş) + yılın onikide biri

income: Yıllık gelir (10.000'e bölünür)

share: Aylık kredi kartı harcamalarının yıllık gelire oranı

expenditure: Ortalama aylık kredi kartı harcaması

owner: Ev sahibi ise 1, ev kiralıyorsa 0

selfempl: Serbest meslek sahibi ise 1, değilse 0

dependents: 1 + bakmakla yükümlü kişi sayısı

months: Geçerli adreste yaşanan ay sayısı

majorcards: Sahip olunan kredi kartı sayısı

active: Etkin kredi hesabı sayısı

Birkaç değişken şüpheli görünüyor. Örneğin, expenditure bu kartta veya uygulamadan önce kullanılan kartlarda yapılan harcama anlamına mı geliyor?

Bu noktada, temel veri karşılaştırmaları çok yardımcı olabilir:

```
In [3]:
expenditures_cardholders = X.expenditure[y]
expenditures_noncardholders = X.expenditure[~y]

print('Fraction of those who did not receive a card and had no expenditures: %.2f' \
      %((expenditures_noncardholders == 0).mean()))
print('Fraction of those who received a card and had no expenditures: %.2f' \
      %(( expenditures_cardholders == 0).mean()))

Fraction of those who did not receive a card and had no expenditures: 1.00
Fraction of those who received a card and had no expenditures: 0.02
```

Yukarıda gösterildiği gibi, kart almayan herkesin harcamaları yoktu, kart alanların sadece % 2'sinin harcamaları yoktu. Modelimizin yüksek bir doğruluğa sahip olması şaşırtıcı değil.
Ancak bu, harcamaların muhtemelen başvurdukları karttaki harcamalar anlamına geldiği bir target leakage durumu gibi görünmektedir.

Share kısmen harcama ile belirlendiğinden, hariç tutulmalıdır.

Active ve *majorcard* değişkenleri biraz daha az açıktır, ancak açıklamayla ilgili görünüyorlar.

Çoğu durumda, daha fazla bilgi edinmek için verileri oluşturan kişileri izleyemiyorsanız, üzülmektense güvende olmak daha iyidir.

Target Leakage olmayan bir modeli şu şekilde çalıştırırız:

```
In [4]:
# Drop leaky predictors from dataset
potential_leaks = ['expenditure', 'share', 'active', 'majorcards']
X2 = X.drop(potential_leaks, axis=1)

# Evaluate the model with leaky predictors removed
cv_scores = cross_val_score(my_pipeline, X2, y,
                            cv=5,
                            scoring='accuracy')

print("Cross-val accuracy: %f" % cv_scores.mean())

Cross-val accuracy: 0.830924
```

Burada accuracy biraz daha düşük, bu da hayal kırıklığı yaratabilir.

Bununla birlikte, yeni uygulamalarda kullanıldığı zaman yaklaşık % 80'inin doğru olmasını bekleyebiliriz, oysa leaky(sızdıran) model muhtemelen bundan daha kötü sonuç verecektir (cross-validation'daki yüksek görünen puanına rağmen).

Sonuç

Veri sızıntısı, birçok veri bilimi uygulamasında milyonlarca dolarlık bir hataya sebep olabilir. Eğitim ve doğrulama verilerinin dikkatlice ayrılması, train-test kontaminasyonunu önleyebilir ve pipeline'lar bu ayrılmanın uygulanmasına yardımcı olabilir.
Aynı şekilde, dikkatli olma, sağduyu ve veri keşfi birleşimi de target leakage'ı belirlemeye yardımcı olabilir.

Bu hala soyut görünebilir. Target Leakage ve train-test kontaminasyonunu belirleme becerinizi geliştirmek için aşağıdaki alıştırmadaki örnekleri gözden geçirmeyi deneyin!

Exercise: Data Leakage
https://www.kaggle.com/recepaydogdu/exercise-data-leakage

Quiz: Intermediate Machine Learning

Global AI Hub

Kaggle Master Week-2 Q&A

Q1- Which of the following statements are true about the intended use of cross-validation?

I - To reduce randomness while measuring model performance.

II - To get a better measure of model performance.

III - To increase model's training performance.

IV - To increase MAE (mean absolute error) or MSE (mean squared error).

- I, II, IV
- II, III
- I, II ✓
- All of them

A1- Cross-validation kullanmamızdaki amaç modelimizde kullandığımız metrikleri daha doğru bir şekilde gözlemleyebilmektir. Dolayısı ile modelimizin hatasını düşürmesi veya modeli daha iyi eğitmemiz üzerinde doğrudan bir etkisi yoktur.

Q2- Which of the following statements are true about LabelEncoder and OneHotEncoder?

I-They help us to deal with categorical values.

II-Label Encoding assigns each value to a different integer whether it is unique or not.

III-One Hot Encoding creates new column for every possible value in the original data.

IV-For large number of categorical variable count value (such as 15 different values) it is not good to use One Hot Encoder generally.

- I, II, IV
- I, III, IV ✓
- I, II, III
- All of them

A2- Label Encoder ve One Hot Encoding kategorik verilerin üstesinden gelmek için kullanılırlar. Label Encoding her eşsiz (unique) değer için bir değer üretip atama yapar. One Hot Encoding ise her değer için yeni bir kolon oluşturur. Bu değerler eşsiz (unique) değerlerdir. Kolon sayısı arttıkça, genelde One Hot Encoding iyi bir performans sergilemez. Bu yüzden One Hot Encoding genelde fazla sayıdaki unique kolon içeren kategorik verilerle kullanıldığında iyi sonuç vermez.

Q3- Which of the following statement is inconsistent with pipelines?

- With pipelines, there is less probability to forget a preprocessing step.
- It's hard to productionize a model with pipelines. ✓
- You won't need to manually keep track of your training and validation data at each step with a pipeline.
  With a pipeline, we can use the cross-validation technique easily.

A3- Pipelineları, modelimize input olarak verilecek datanın her zaman aynı işlemlerden geçirilmesi, ön-işlemede meydana gelebilecek hata ve eksiklik risklerinin azaltılması ve cross-validation gibi model değerlendirmesi yaptığımız işlemleri kolayca yapabilmek için kullanıyoruz. Modelleri pipelinelar ile oluşturmak zor değil ve model deployment aşamasında hata yapmanızı büyük ölçüde engelleyeceğinden dolayı oldukça kullanışlılar.

Q4- print(df(head).method())

Assume that you want to print locations of the missing values in the top 10 rows. Which method is suitable for this?

- dropna(how='any')
- isnan
- notnull
- isnull ✓

A4- Bir veri çerçevesinde NULL değerlerini denetlemek ve yönetmek için isnull () ve notnull () yöntemleri kullanılır. isnull () yöntemi, NaN değeri için True ve boş olmayan değer için False döndürür. notnull() bu durumun tam tersidir.

Q5- Which of the following is not a Booster parameter of XGBoost?

- min_child_weight
- objective ✓
- max_leaf_nodes
- colsample_bylevel

A5- "objective" parametresi bir learning task parametresidir. Bunun gibi parametreler, her adımda hesaplanacak

metriğin optimizasyon hedefini tanımlamak için kullanılır.

Q6- What do the highlighted code pieces mean?

```
X_train_plus = X_train.copy()
X_valid_plus = X_valid.copy()
for col in cols_with_missing:
    X_train_plus[col + '_was_missing'] = X_train_plus[col].isnull()
    X_valid_plus[col + '_was_missing'] = X_valid_plus[col].isnull()
my_imputer = SimpleImputer()
imputed_X_train_plus = pd.DataFrame(my_imputer.fit_transform(X_train_plus))
imputed_X_valid_plus = pd.DataFrame(my_imputer.transform(X_valid_plus))
imputed_X_train_plus.columns = X_train_plus.columns
imputed_X_valid_plus.columns = X_valid_plus.columns
```

- To make new columns indicating what will be imputed
- For imputation
- To make copy to avoid changing original data
- To put removed column names back ✓

A6- Yukarıdaki code parçası kayıp verileri işlemede kullanılan bir yöntem olan Imputation adımlarını ifade etmektedir. İşaretli satırlar da, imputing işlemi sırasında kayıp verileri çıkarılmış kolonları, temizlenmiş olarak geri almamızı sağlar.

Q7- Which of the below is/are nominal variable(s)?

I - Gender

II - Genotype

III - Religious preference

IV- IQ

V - Income earned in a week.

- I, II
- I, II, III ✓
- II, III, IV
- All of them

A7- Nominal değişkenler aralarında sıralama yapılamayan kategorik değişkenlerdir. Cinsiyet, genotip ve dini tercihler değerlerinin birbirlerine herhangi bir üstünlüğü bulunmayan değişkenlerdir. IQ ve haftalık kazanç kategorik değişkenler olmadığından nominal değişken olarak değerlendirilemezler.

Q8- Which of the following statements are true about "max_depth" hyperparameter in Random Forest?

```
I-      Lower is better parameter in case of same validation accuracy
II-     Higher is better parameter in case of same validation accuracy
III-    Increase the value of max_depth may overfit the data
IV-     Increase the value of max_depth may underfit the data
```

- I, IV
- II, IV
- I, III ✓
- II, III

A8- Çünkü maksimum derinliği gereğinden fazla artırmamız modelimizin veriyi ezberlemesine ve overfit olmasına yol açar. Farklı derinlikler ile oluşturduğumuz modellerden aynı skoru alırsak modelimiz karmaşıklığını azaltmak için düşük derinlikli olanı tercih etmemiz gerekir.

Q9- You will build a model to predict housing prices. The model will be deployed on an ongoing basis, to predict the price of a new house when a description is added to a website. Here are four features that could be used as predictors. Which of the features is most likely to be a source of leakage?

- Size of the house (in square meters)
- Average sales price of homes in the same neighborhood ✓
- Latitude and longitude of the house
- Whether the house has a basement

A9- Data leakage (veri sızıntısı), eğitim verileri hedef hakkında bilgi içerdiğinde gerçekleşir, ancak model tahmini için kullanıldığında benzer veriler kullanılamaz. Bu, eğitim setinde (ve hatta muhtemelen doğrulama verilerinde) yüksek performansa yol açar, ancak model üretimde kötü performans gösterecektir.

Başka bir deyişle, karar verme mekanizması başlayana kadar o model çok doğru görünür fakat en sonunda modelin çok yanlış kurulduğu ortaya çıkar.

1- Bir evin büyüklüğünün satıldıktan sonra değiştirilmesi olası değildir (teknik olarak mümkün olsa da). Ancak tipik olarak bu bir tahmin yapmamız gerektiğinde kullanılabilir ve veriler ev satıldıktan sonra değiştirilmez. Bu yüzden oldukça güvenlidir.
2- Bunun ne zaman güncellendiğini bilmiyoruz. Bir ev satıldıktan sonra ham verilerde alan güncellenirse ve ortalamanın hesaplanması için evin satışı kullanılırsa, bu veri sızıntısı anlamına gelir. Bir uçta, mahallede sadece bir ev satılıyorsa ve tahmin etmeye çalıştığımız ev ise, o zaman ortalama tahmin etmeye çalıştığımız değere tam olarak eşit olacaktır. Genel olarak, az satış yapılan mahalleler için model, eğitim verileri üzerinde çok iyi performans gösterecektir. Ancak modeli uyguladığınızda, tahmin ettiğiniz ev henüz satılmayacaktır, bu nedenle bu özellik eğitim verilerinde olduğu gibi çalışmaz.
3- Bunlar değişmez ve bir tahmin yapmak istediğimiz zaman hazır olur. Yani burada veri sızıntı riski yoktur.
4- Bu da değişmez ve bir tahmin yapmak istediğimiz anda kullanılabilir. Yani burada veri sızıntı riski yoktur.

Q10- How is the Gradient Boosting cycle proceed? Please choose the correct order from the mixed statements below.

I- We add the new model to ensemble.
II- We use the current ensemble to generate predictions for each observation in the dataset.
III- We use the loss function to fit a new model that will be added to the ensemble.

- I-II-III
- I-III-II
- II-I-III
- II-III-I ✓

A10- Gradient boosting döngüsünde ilk olarak, veri grubundaki her bir gözlem için tahminler oluşturmak üzere mevcut topluluğu (ensemble) kullanıyoruz. Bir tahmin yapmak için, topluluktaki tüm modellerden tahminleri ekliyoruz. Bu tahminler bir kayıp fonksiyonunu (loss function) hesaplamak için kullanılır.

Daha sonra loss function ı, topluluğa (ensemble) eklenecek yeni bir modele uyacak şekilde kullanıyoruz. Özellikle, model parametrelerini belirlemede kullanıyoruz ki böylece bu yeni modeli topluluğa eklemekle olası zaman kayıplarını azaltıyoruz. Son olarak da topluluğa yeni modeli ekliyoruz.

Data Visualization

Hello, Seaborn

Bu uygulamalı mikro kursta, güçlü ama kullanımı kolay bir veri görselleştirme aracı olan seaborn ile veri görselleştirmelerinizi bir sonraki seviyeye nasıl taşıyacağınızı öğreneceksiniz.
Seaborn'u kullanmak için, popüler bir programlama dili olan Python'da kod yazmayı da öğreneceksiniz. Bahsedilen,

- mikro-kurs, önceden programlama deneyimi olmayanlara yöneliktir ve
- her grafik kısa ve basit kod kullanır, bu da seaborn'u diğer birçok veri görselleştirme aracından (örneğin Excel gibi) çok daha hızlı ve kolay hale getirir.

Yani, daha önce hiç bir kod satırı yazmamışsanız ve bugün daha hızlı, daha çekici grafikler yapmaya başlamak için asgari olanı öğrenmek istiyorsanız, doğru yerdesiniz! Yapacağınız grafiklerden bazılarına göz atmak için aşağıdaki figürlere göz atın.

Notebook Kurulumu

Kodlama ortamınızı ayarlamak için her notebook'un üstünde çalıştırmanız gereken birkaç kod satırı vardır. Bu kod satırlarını anlamanız şuan önemli değil ve bu yüzden henüz ayrıntılara girmeyeceğiz.

In [1]:

```
import pandas as pd
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup Complete")
```

```
Setup Complete
```

Veri Yükleme

Bu notebook'da altı ülke için tarihi FIFA sıralaması veri setiyle çalışacağız: Arjantin (ARG), Brezilya (BRA), İspanya (ESP), Fransa (FRA), Almanya (GER) ve İtalya (ITA). Bu veriseti CSV dosyası olarak saklanır (comma-separated values file kısatlması). CSV dosyasını Excel'de açmak, her ülke için bir sütunla birlikte her tarih için bir satır gösterir.

	A	B	C	D	E	F	G
1	Date	ARG	BRA	ESP	FRA	GER	ITA
2	8/8/93	5	8	13	12	1	2
3	9/23/93	12	1	14	7	5	2
4	10/22/93	9	1	7	14	4	3
5	11/19/93	9	4	7	15	3	1
6	12/23/93	8	3	5	15	1	2
7	2/15/94	9	2	6	14	1	7
8	3/15/94	8	2	6	15	1	11
9	4/19/94	10	1	7	15	2	13

Verileri notebook'a yüklemek için aşağıdaki kod hücresine aşağıdaki gibi uygulanan iki ayrı adım kullanacağız:

- veri kümesine erişilebileceği konumu (veya filepath) belirterek başlayın ve ardından
- veri kümesinin içeriğini not defterine yüklemek için dosya yolunu kullanın.

In [2]:
```
# Path of the file to read
fifa_filepath = "../input/fifa.csv"

# Read the file into a variable fifa_data
fifa_data = pd.read_csv(fifa_filepath, index_col="Date", parse_dat
es=True)
```

Verileri İnceleyelim

Şimdi, düzgün yüklendiğinden emin olmak için fifa_data'daki veri kümesine hızlı bir şekilde bakacağız.

In [3]:

```
# Print the first 5 rows of the data
fifa_data.head()
```

Out[3]:

	ARG	BRA	ESP	FRA	GER	ITA
Date						
1993-08-08	5.0	8.0	13.0	12.0	1.0	2.0
1993-09-23	12.0	1.0	14.0	7.0	5.0	2.0
1993-10-22	9.0	1.0	7.0	14.0	4.0	3.0
1993-11-19	9.0	4.0	7.0	15.0	3.0	1.0
1993-12-23	8.0	3.0	5.0	15.0	1.0	2.0

İlk beş satırın yukarıdaki Excel görüntüsüyle aynı olduğunu kontrol edin.

Plot the Data (Verileri Çizin)

Bu kursta, birçok farklı plot türü ile ilgili bilgi edineceksiniz. Öğrendiklerinize bir göz atmak için aşağıdaki line chart(çizgi grafiği) oluşturan kodu inceleyin.

In [4]:

```
# Set the width and height of the figure
plt.figure(figsize=(16,6))

# Line chart showing how FIFA rankings evolved over time
sns.lineplot(data=fifa_data)
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f22bfac3fd0>
```

Bu kod henüz anlam ifade etmiyor olabilir, ilerleyen eğitimlerde kod hakkında daha fazla bilgi edineceksiniz.

Line Charts (Çizgi Grafikleri)

Bu bölümde, profesyonel görünümlü çizgi grafikler oluşturmak için yeterli düzeyde Python öğreneceksiniz.
Ardından, aşağıdaki alıştırmada, yeni becerilerinizi gerçek dünyadaki bir veri kümesiyle çalışacaksınız.

In [1]:
```
import pandas as pd
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup Complete")
```

Dataset Seçimi

Bu bölümün veri kümesi, Spotify'daki küresel günlük akışları izler. 2017 ve 2018'den beş popüler şarkıya odaklanıyoruz:

1. "Shape of You", by Ed Sheeran
2. "Despacito", by Luis Fonzi
3. "Something Just Like This", by The Chainsmokers and Coldplay
4. "HUMBLE.", by Kendrick Lamar
5. "Unforgettable", by French Montana

Date	Shape of You	Despacito	Something Just Like This	HUMBLE.	Unforgettable
1/6/17	12287078				
1/7/17	13190270				
1/8/17	13099919				
1/9/17	14506351				
1/10/17	14275628				
1/11/17	14372699				
1/12/17	14148108				
1/13/17	14536236	275178			
1/14/17	14173311	1144886			
1/15/17	12889849	1288198			
1/16/17	14128468	1827581			

Görüntülenen ilk tarihin, "Shape of You" nun çıkış tarihine karşılık gelen 6 Ocak 2017 olduğuna dikkat edin.
Ve tabloyu kullanarak, "Shape of You" nun yayınlandığı gün küresel olarak 12.287.078 kez dinlendiğini görebilirsiniz.
Diğer şarkıların ilk sıralarda eksik değerleri olduğuna dikkat edin, çünkü daha yayınlanmadılar!

Veri Yükleme

In [2]:

```
# Path of the file to read
spotify_filepath = "../input/spotify.csv"

# Read the file into a variable spotify_data
spotify_data = pd.read_csv(spotify_filepath, index_col="Date", par
se_dates=True)
```

Verileri İnceleyin

In [3]:

```
# Print the first 5 rows of the data
spotify_data.head()
```

Out[3]:

	Shape of You	Despacito	Something Just Like This	HUMBLE.	Unforgettable
Date					
2017-01-06	12287078	NaN	NaN	NaN	NaN
2017-01-07	13190270	NaN	NaN	NaN	NaN
2017-01-08	13099919	NaN	NaN	NaN	NaN
2017-01-09	14506351	NaN	NaN	NaN	NaN
2017-01-10	14275628	NaN	NaN	NaN	NaN

In [4]:

```
# Print the last five rows of the data
spotify_data.tail()
```

Out[4]:

	Shape of You	Despacito	Something Just Like This	HUMBLE.	Unforgettable
Date					
2018-01-05	4492978	3450315.0	2408365.0	2685857.0	2869783.0
2018-01-06	4416476	3394284.0	2188035.0	2559044.0	2743748.0
2018-01-07	4009104	3020789.0	1908129.0	2350985.0	2441045.0
2018-01-08	4135505	2755266.0	2023251.0	2523265.0	2622693.0
2018-01-09	4168506	2791601.0	2058016.0	2727678.0	2627334.0

Neyse ki, her şarkı için her gün milyonlarca günlük küresel akış doğru görünüyor ve verileri çizmeye devam edebiliriz!

Verileri Çizin

Veri seti notebook'a yüklendiğine göre yanlızca tek bir satır koda ihtiyacımız var.

In [5]:
```
# Line chart showing daily global streams of each song
sns.lineplot(data=spotify_data)
```

Out[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0cf1118e48>
```

Yukarıda görebileceğiniz gibi, kod satırı nispeten kısadır ve iki ana bileşene sahiptir:

- *sns.lineplot* notebook'a çizgi grafik oluşturmak istediğimizi söyler.
- *data = spotify_data* grafiği oluşturmak için kullanılacak verileri seçer.

Bazen, şeklin boyutu ve grafiğin başlığı gibi değiştirmek istediğimiz ek ayrıntılar vardır. Bu seçeneklerin her biri tek bir kod satırı ile kolayca ayarlanabilir.

In [6]:

```
# Set the width and height of the figure
plt.figure(figsize=(14,6))

# Add title
plt.title("Daily Global Streams of Popular Songs in 2017-2018")

# Line chart showing daily global streams of each song
sns.lineplot(data=spotify_data)
```

Out[6]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0cf0fd9da0>
```

İlk kod satırı, *figure*'ün boyutunu 14 inç (genişlik) x 6 inç (yükseklik) olarak ayarlar. Herhangi bir *figure*'ün boyutunu ayarlamak için yalnızca göründüğü gibi aynı kod satırını kopyalamanız gerekir. Ardından, özel bir boyut kullanmak isterseniz, sağlanan 14 ve 6 değerlerini istediğiniz genişlik ve yüksekliğe değiştirin.

İkinci kod satırı şeklin başlığını belirler. Başlık her zaman tırnak içine alınmalıdır ("...")!

Plot a subset of the data (Verilerin alt kümelerini çizme)

Şimdiye kadar, veri kümesindeki *her sütun* için nasıl bir çizgi çizeceğinizi öğrendiniz. Bu bölümde, sütunların bir *alt kümesini* nasıl çizeceğinizi öğreneceksiniz.

Tüm sütunların adlarını yazarak başlayacağız. Bu, bir kod satırı ile yapılır ve sadece veri kümesinin adını değiştirerek (bu durumda *spotify_data*) herhangi bir veri kümesi için uyarlanabilir.

In [7]:

```
list(spotify_data.columns)
```

Out[7]:

```
['Shape of You',
 'Despacito',
 'Something Just Like This',
 'HUMBLE.',
 'Unforgettable']
```

Bir sonraki kod hücresinde, veri kümesindeki ilk iki sütuna karşılık gelen satırları çizeriz.

In [8]:

```
# Set the width and height of the figure
plt.figure(figsize=(14,6))

# Add title
plt.title("Daily Global Streams of Popular Songs in 2017-2018")

# Line chart showing daily global streams of 'Shape of You'
sns.lineplot(data=spotify_data['Shape of You'], label="Shape of Yo
u")

# Line chart showing daily global streams of 'Despacito'
sns.lineplot(data=spotify_data['Despacito'], label="Despacito")

# Add label for horizontal axis
plt.xlabel("Date")
```

Out[8]:

```
Text(0.5, 0, 'Date')
```

Kodun ilk iki satırı şeklin başlığını ve boyutunu belirler.

Sonraki iki satırın her biri çizgi grafiğine bir çizgi ekler. Örneğin, "Shape of You" satırını ekleyen ilkini düşünün:

```
# Line chart showing daily global streams of 'Shape of You'
sns.lineplot(data=spotify_data['Shape of You'], label="Shape of
You")
```

Satırın göstergede görünmesi ve karşılık gelen etiketinin ayarlanması için *label* = "*Shape of You*" ekliyoruz.

Exercise: Line Charts

Bu alıştırmada, yeni bilginizi gerçek dünya senaryosuna çözüm önermek için kullanacaksınız. Başarılı olmak için verileri Python'a aktarmanız, verileri kullanarak soruları yanıtlamanız ve verilerdeki kalıpları anlamak için çizgi grafikler oluşturmanız gerekir.

Senaryo

Kısa bir süre önce Los Angeles şehrinde müzeleri yönetmek için işe alındınız. İlk projeniz aşağıdaki resimlerde gösterilen dört müzeye odaklanıyor.

Avila Adobe

Firehouse Museum

Chinese American Museum

America Tropical Inventive Center

Her müzeye aylık ziyaretçileri izleyen Los Angeles Data Portal'ından verileri kullanacaksınız.

	A	B	C	D	E
1	Date	Avila Adobe	Firehouse Museum	Chinese American Museum	America Tropical Interpretive Center
2	1/1/14	24778	4486	1581	6602
3	2/1/14	18976	4172	1785	5029
4	3/1/14	25231	7082	3229	8129
5	4/1/14	26989	6756	2129	2824
6	5/1/14	36883	10858	3676	10694
7	6/1/14	29487	5751	2121	11036
8	7/1/14	32378	5406	2239	13490
9	8/1/14	37680	8619	1769	9139
10	9/1/14	28473	61192	1073	5661
11	10/1/14	27995	6488	1979	7356
12	11/1/14	25691	4189	2404	9773
13	12/1/14	18754	4339	1319	7184
14	1/1/15	20438	3858	1823	6250
15	2/1/15	15578	3742	1558	5907
16	3/1/15	21297	5390	2336	9884

In [1]:

```
import pandas as pd
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup Complete")
```

```
Setup Complete
```

Step 1: Veri Yükleme

In [3]:

```
# Path of the file to read
museum_filepath = "../input/museum_visitors.csv"

# Fill in the line below to read the file into a variable museum_data
museum_data = pd.read_csv(museum_filepath, index_col="Date", parse_dates=True)

# Run the line below with no changes to check that you've loaded the data correctly
step_1.check()
```

Step 2: Verileri İnceleyin

In [5]:

```
# Print the last five rows of the data
museum_data.tail() # Your code here
```

Out[5]:

	Avila Adobe	Firehouse Museum	Chinese American Museum	America Tropical Interpretive Center
Date				
2018-07-01	23136	4191	2620	4718
2018-08-01	20815	4866	2409	3891
2018-09-01	21020	4956	2146	3180
2018-10-01	19280	4622	2364	3775
2018-11-01	17163	4082	2385	4562

Son sıra (2018-11-01 için) Kasım 2018'de her müzeye ziyaretçi sayısını, bir sonraki son sıra (2018-10-01 için) Ekim 2018'de her müzeye ziyaretçi sayısını gösterir vb.

Step 3: Müze Kurulunu İkna Edin

Firehouse Museum, 2014'te inanılmaz sayıda ziyaretçi getiren bir etkinlik düzenlediklerini ve benzer bir etkinliği tekrar gerçekleştirmek için ekstra bütçe almaları gerektiğini iddia ediyor.
Diğer müzeler bu tür etkinliklerin o kadar da önemli olmadığını ve bütçelerin ortalama bir gündeki son ziyaretçilere göre bölünmesi gerektiğini düşünüyor.

Müze kuruluna etkinliğin her müzedeki düzenli trafiğe kıyasla nasıl olduğunu göstermek için, her müzeye ziyaretçi sayısının zaman içinde nasıl geliştiğini gösteren bir çizgi grafik oluşturun.

In [8]:

```
# Line chart showing the number of visitors to each museum over time
plt.figure(figsize=(18, 5))
plt.title("Monthly Visitors to Los Angeles City Museums")
sns.lineplot(data=museum_data) # Your code here

# Check your answer
step_3.check()
```

Step 4: Mevsimsel Değerlendirme

Avila Adobe’daki çalışanlarla toplantıdayken, çalışanların bazı sezonlarda sıkıntı yaşadıkları duyuluyor. Düşük ziyaretçili sezonlarda çalışanlar verimli ve mutlular, yüksek ziyaretçili sezonlarda ise çalışanlar verimsiz ve stresliler. Bu sezonların ne zaman etkili olduğunu tahmin edebiliyorsanız, çalışmalara yardımcı olacak ek çalışanların ne zaman işe alınacağının planını yapabilirsiniz.

Part A

Avila Adobe'ye gelen ziyaretçi sayısının zamanla nasıl geliştiğini gösteren bir çizgi grafik oluşturun.

In [10]:

```
# Line plot showing the number of visitors to Avila Adobe over time
plt.figure(figsize=(20,6))
plt.title("Monthly Visitors of Avila Adobe")
sns.lineplot(data=museum_data["Avila Adobe"], label="Avila Adobe") # Your code here

# Check your answer
step_4.a.check()
```

```
Thank you for creating a line chart!  To see how your code compares to the official s
olution, please use the code cell below.
```

Part B

Avila Adobe daha fazla ziyaretçi alıyor:

- Eylül-Şubat aylarında (LA'da, sonbahar ve kış aylarında) veya
- Mart-Ağustos aylarında (LA, ilkbahar ve yaz aylarında)?

Bu bilgileri kullanarak, müze personeli mevsimlik ek çalışanları ne zaman kullanmalıdır?

İpucu: Her yılın başlarına bakın (Ocak ayı civarında). Çizgi grafik düşük değerlere düşüyor mu veya nispeten yüksek değerlere ulaşıyor mu?

Çizgi grafik genellikle her yılın başlarında (Aralık ve Ocak aylarında) nispeten düşük değerlere düşer ve yılın ortasında (özellikle Mayıs ve Haziran aylarında) en yüksek değerlerine ulaşır. Böylece, Avila Adobe genellikle Mart-Ağustos aylarında (veya ilkbahar ve yaz aylarında) daha fazla ziyaretçi alır. Bunu göz önünde bulundurarak, Avila Adobe Mart-Ağustos aylarında (ilkbahar ve yaz) ekstra çalışmaya yardımcı olmak için daha fazla mevsimlik çalışan işe almaktan kesinlikle yararlanabilir!

Bar Charts ve Heatmaps (Çubuk Grafikleri ve Isı Haritaları

Artık kendi çizgi grafiklerinizi oluşturabileceğinize göre, daha fazla grafik türü hakkında bilgi edinme zamanı!

In [1]:
```
import pandas as pd
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup Complete")
```

Dataset Seçimi

Bu eğitimde, ABD Ulaştırma Bakanlığı'ndan uçuş gecikmelerini takip eden bir veri kümesi ile çalışacağız.

Bu CSV dosyasını Excel'de açtığınızda, her ay için bir satır (burada 1 = Ocak, 2 = Şubat vb.) Ve her havayolu kodu için bir sütun gösterilir. Her bir giriş, farklı bir havayolu ve ay için ortalama varış gecikmesini (dakika olarak) gösterir (tümü 2015 yılında). Negatif girişler (ortalama olarak) erken varma eğilimindeki uçuşları gösterir. Örneğin, Ocak ayında ortalama bir American Airlines uçuşu (havayolu kodu: AA) yaklaşık 7 dakika geç geldi ve Nisan ayında ortalama Alaska Airlines uçuşu (havayolu kodu: AS) yaklaşık 3 dakika erken geldi.

In [2]:
```
# Path of the file to read
flight_filepath = "../input/flight_delays.csv"

# Read the file into a variable flight_data
flight_data = pd.read_csv(flight_filepath, index_col="Month")
```

Verileri İnceleyelim

Veri kümesi küçük olduğundan, tüm içeriğini kolayca yazdırabiliriz. Bu, yalnızca veri kümesinin adı ile tek bir kod satırı yazarak yapılır.

In [3]:

```
# Print the data
flight_data
```

Out[3]:

	AA	AS	B6	DL	EV	F9	HA	MQ	NK
Month									
1	6.955843	-0.320888	7.347281	-2.043847	8.537497	18.357238	3.512640	18.164974	11.3
2	7.530204	-0.782923	18.657673	5.614745	10.417236	27.424179	6.029967	21.301627	16.4
3	6.693587	-0.544731	10.741317	2.077965	6.730101	20.074855	3.468383	11.018418	10.0
4	4.931778	-3.009003	2.780105	0.083343	4.821253	12.640440	0.011022	5.131228	8.76
5	5.173878	-1.716398	-0.709019	0.149333	7.724290	13.007554	0.826426	5.466790	22.3
6	8.191017	-0.220621	5.047155	4.419594	13.952793	19.712951	0.882786	9.639323	35.5
7	3.870440	0.377408	5.841454	1.204862	6.926421	14.464543	2.001586	3.980289	14.3
8	3.193907	2.503899	9.280950	0.653114	5.154422	9.175737	7.448029	1.896565	20.5
9	-1.432732	-1.813800	3.539154	-3.703377	0.851062	0.978460	3.696915	-2.167268	8.00
10	-0.580930	-2.993617	3.676787	-5.011516	2.303760	0.082127	0.467074	-3.735054	6.81
11	0.772630	-1.916516	1.418299	-3.175414	4.415930	11.164527	-2.719894	0.220061	7.54
12	4.149684	-1.846681	13.839290	2.504595	6.685176	9.346221	-1.706475	0.662486	12.7

Bar Chart

Diyelim ki aylara göre Spirit Airlines (havayolu kodu: NK) uçuşları için ortalama varış gecikmesini gösteren bir çubuk grafik oluşturmak istiyoruz.

In [4]:

```
# Set the width and height of the figure
plt.figure(figsize=(10,6))

# Add title
plt.title("Average Arrival Delay for Spirit Airlines Flights, by Month")

# Bar chart showing average arrival delay for Spirit Airlines flights by month
sns.barplot(x=flight_data.index, y=flight_data['NK'])

# Add label for vertical axis
plt.ylabel("Arrival delay (in minutes)")
```

Out[4]:

```
Text(0, 0.5, 'Arrival delay (in minutes)')
```

Metnin (başlık ve dikey eksen etiketi) ve şeklin boyutunun özelleştirilmesine yönelik komutlar, önceki öğreticiden aşinadır. Çubuk grafiği oluşturan kod yenidir:

```
# Bar chart showing average arrival delay for Spirit Airlines flights by month
sns.barplot(x=flight_data.index, y=flight_data['NK'])
```

Heatmap

Aşağıdaki kod hücresinde, flight_data'daki patternleri hızlı bir şekilde görselleştirmek için bir ısı haritası oluşturuyoruz. Her hücre karşılık gelen değerine göre renk kodludur.

In [5]:

```
# Set the width and height of the figure
plt.figure(figsize=(14,7))

# Add title
plt.title("Average Arrival Delay for Each Airline, by Month")

# Heatmap showing average arrival delay for each airline by month
sns.heatmap(data=flight_data, annot=True)

# Add label for horizontal axis
plt.xlabel("Airline")
```

Out[5]:

```
Text(0.5, 42.0, 'Airline')
```

Isı haritasını oluşturmak için ilgili kod aşağıdaki gibidir:

```
# Heatmap showing average arrival delay for each airline by month
sns.heatmap(data=flight_data, annot=True)
```

Bu kodun üç ana bileşeni vardır:

- *sns.heatmap* **-** Bu notebook'a bir ısı haritası oluşturmak istediğimizi söyler.
- *data=flight_data* – Bu ise ısı haritası oluşturmak için *flight_data* içindeki tüm verileri kullanacağımızı söyler.
- *annot=True* – Bu ise her hücre için değerlerin içerisinde yazılmasını sağlar. Bunu kaldırırsak hücrelerin içindeki sayılar silinecektir.

Tabloda hangi patternleri tespit edebilirsiniz? Örneğin, yakından bakarsanız, yıl sonuna doğru (özellikle 9-11. Aylar) tüm havayolları için

nispeten karanlık görünür. Bu, havayolu şirketlerinin bu aylarda program tutma konusunda daha iyi (ortalama olarak) olduğunu göstermektedir!

Exercise: Bar Charts ve Heatmaps

Bu alıştırmada, yeni bilginizi gerçek dünya senaryosuna çözüm önermek için kullanacaksınız. Başarılı olmak için verileri Python'a aktarmanız, verileri kullanarak soruları yanıtlamanız ve verilerdeki patternleri anlamak için çubuk grafikler ve ısı haritaları oluşturmanız gerekir.

Senaryo

Kısa süre önce kendi video oyununuzu yaratmaya karar verdiniz! IGN Game Reviews'in hevesli bir okuyucusu olarak, en son oyun sürümlerinin yanı sıra uzmanlardan aldıkları sıralama ile 0 (Disaster) ile 10 (Masterpiece) arasında değişen sıralamayı duyarsınız.

Çıkış tarihi yaklaşan oyununuzun tasarımını yönetmek için IGN incelemelerini kullanmak istiyorsunuz. Neyse ki, birisi analizinize rehberlik etmek için kullanabileceğiniz gerçekten kullanışlı bir CSV dosyasındaki sıralamaları özetledi.

In [1]:

```
import pandas as pd
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup Complete")
```

```
Setup Complete
```

Step 1: Veri Yükleme

In [3]:

```
# Path of the file to read
ign_filepath = "../input/ign_scores.csv"

# Fill in the line below to read the file into a variable ign_data
ign_data = pd.read_csv(ign_filepath, index_col = "Platform")

# Run the line below with no changes to check that you've loaded the data correctly
step_1.check()
```

Step 2: Verileri İnceleyin

In [5]:

```
# Print the data
ign_data # Your code here
```

Out[5]:

	Action	Action, Adventure	Adventure	Fighting	Platformer	Puzzle	RPG	Racing	Shooter	Simul
Platform										
Dreamcast	6.882857	7.511111	6.281818	8.200000	8.340000	8.088889	7.700000	7.042500	7.616667	7.628
Game Boy Advance	6.373077	7.507692	6.057143	6.226316	6.970588	6.532143	7.542857	6.657143	6.444444	6.928
Game Boy Color	6.272727	8.166667	5.307692	4.500000	6.352941	6.583333	7.285714	5.897436	4.500000	5.900
GameCube	6.532584	7.608333	6.753846	7.422222	6.665714	6.133333	7.890909	6.852632	6.981818	8.028
Nintendo 3DS	6.670833	7.481818	7.414286	6.614286	7.503448	8.000000	7.719231	6.900000	7.033333	7.700
Nintendo 64	6.649057	8.250000	7.000000	5.681250	6.889655	7.461538	6.050000	6.939623	8.042857	5.675
Nintendo DS	5.903608	7.240000	6.259804	6.320000	6.840000	6.604615	7.222619	6.038636	6.965217	5.874
Nintendo DSi	6.827027	8.500000	6.090909	7.500000	7.250000	6.810526	7.166667	6.563636	6.500000	5.195
PC	6.805791	7.334746	7.136798	7.166667	7.410938	6.924706	7.759930	7.032418	7.084878	7.104
PlayStation	6.016406	7.933333	6.313725	6.553731	6.579070	6.757895	7.910000	6.773387	6.424000	6.918
PlayStation 2	6.467361	7.250000	6.315152	7.306349	7.068421	6.354545	7.473077	6.585065	6.641667	7.152
PlayStation 3	6.853819	7.306154	6.820988	7.710938	7.735714	7.350000	7.436111	6.978571	7.219553	7.142
PlayStation 4	7.550000	7.835294	7.388571	7.280000	8.390909	7.400000	7.944000	7.590000	7.804444	9.250
PlayStation Portable	6.467797	7.000000	6.938095	6.822222	7.194737	6.726667	6.817778	6.401961	7.071053	6.761
PlayStation Vita	7.173077	6.133333	8.057143	7.527273	8.568750	8.250000	7.337500	6.300000	7.660000	5.725
Wii	6.262718	7.294643	6.234043	6.733333	7.054255	6.426984	7.410345	5.011667	6.479798	6.327
Wireless	7.041699	7.312500	6.972414	6.740000	7.509091	7.360550	8.260000	6.898305	6.906780	7.802
Xbox	6.819512	7.479032	6.821429	7.029630	7.303448	5.125000	8.277778	7.021591	7.485417	7.155
Xbox 360	6.719048	7.137838	6.857353	7.552239	7.559574	7.141026	7.650000	6.996154	7.338153	7.325
Xbox One	7.702857	7.566667	7.254545	7.171429	6.733333	8.100000	8.291667	8.163636	8.020000	7.733
iPhone	6.865445	7.764286	7.745833	6.087500	7.471930	7.810784	7.185185	7.315789	6.995588	7.328

Yeni yazdırdığınız veri kümesi, platforma ve türe göre ortalama puanı gösterir. Aşağıdaki soruları cevaplamak için verileri kullanın.

In [6]:

```
# Fill in the line below: What is the highest average score received by PC games,
# for any platform?
high_score = 7.759930

# Fill in the line below: On the Playstation Vita platform, which genre has the
# lowest average score? Please provide the name of the column, and put your answer
# in single quotes (e.g., 'Action', 'Adventure', 'Fighting', etc.)
worst_genre = "Simulation"

# Check your answers
step_2.check()
```

Step 3: En iyi platform hangisi?

Hatırlayacağınız gibi, en sevdiğiniz video oyunu 2008'de Wii platformu için piyasaya sürülen bir yarış oyunu Mario Kart Wii oldu. Ve IGN'de sizinle aynı fikirde olup harika bir oyun olduğunu kabul ediyor - bu oyundaki puanları 8.9! Bu oyunun başarısından esinlenerek, Wii platformu için kendi yarış oyununuzu yaratmayı düşünüyorsunuz.

Part A

Her platform için yarış oyunları için ortalama puanı gösteren bir çubuk grafik oluşturun. Grafiğinizde her platform için bir çubuk bulunmalıdır.

In [8]:

```
# Bar chart showing average score for racing games by platform
plt.figure(figsize = (16, 6)) # Your code here

sns.barplot(x = ign_data.index, y=ign_data["Racing"])

# Check your answer
step_3.a.check()
```

Part B

Çubuk grafiğe dayanarak, Wii platformunun yüksek puan almasını bekliyor musunuz? Değilse, hangi oyun platformu en iyi alternatif gibi görünüyor?

Sonuç: Verilere dayanarak, Wii platformunun yüksek puan almasını beklememeliyiz. Aslında, ortalama olarak, Wii için yarış oyunları diğer platformlardan daha düşük puan. Xbox One en yüksek ortalama dereceye sahip olduğu için en iyi alternatif gibi görünüyor.

Step 4: Olası tüm kombinasyonları inceleyelim!

Sonunda, Wii için bir yarış oyunu oluşturmaya karar veriyorsun, ama yine de kendi video oyununu yaratmaya kararlısın! Oyun ilgi alanlarınız oldukça geniş olduğundan (... genellikle çoğu video oyununu seviyorsunuz), yeni tür ve platform seçiminizi bilgilendirmek için IGN verilerini kullanmaya karar verdiniz.

Part A

Verileri türe ve platforma göre ortalama bir puan haritası oluşturmak için kullanın.

In [12]:

```
# Heatmap showing average game score by platform and genre
plt.figure(figsize = (12, 8))
sns.heatmap(data = ign_data, annot=True) # Your code here

# Check your answer
step_4.a.check()
```

Part B

Hangi tür ve platform kombinasyonu en yüksek ortalama derecelendirmeyi alır? Hangi kombinasyon en düşük ortalama sıralamayı alır?

Çözüm: Playstation 4 için **simulation** oyunları en yüksek ortalama puanı alır (9.2). Game Boy Color için **shooting** ve **fighting** oyunları en düşük ortalama sıralamayı (4.5) alır.

Scatter Plots (Dağılım Grafikleri)

Bu öğreticide, gelişmiş **Scatter Plots** (dağılım grafikleri) oluşturmayı öğreneceksiniz.

In [1]:

```
import pandas as pd
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup Complete")
```

Verileri Yükleyelim ve İnceleyelim

Bazı müşterilerin neden diğerlerinden daha fazla ödeme yaptığını anlayabilmemiz için sağlık sigortası ücretlerinin (sentetik) bir veri kümesiyle çalışacağız.

age	sex	bmi	children	smoker	region	charges
19	female	27.9	0	yes	southwest	16884.924
18	male	33.77	1	no	southeast	1725.5523
28	male	33	3	no	southeast	4449.462
33	male	22.705	0	no	northwest	21984.4706
32	male	28.88	0	no	northwest	3866.8552
31	female	25.74	0	no	southeast	3756.6216
46	female	33.44	1	no	southeast	8240.5896
37	female	27.74	3	no	northwest	7281.5056

İsterseniz, veri seti ile ilgili daha fazla bilgi edinebilirsiniz; https://www.kaggle.com/mirichoi0218/insurance/home

In [2]:

```
# Path of the file to read
insurance_filepath = "../input/insurance.csv"

# Read the file into a variable insurance_data
insurance_data = pd.read_csv(insurance_filepath)
```

In [3]:

```
insurance_data.head()
```

Out[3]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520

Scatter Plots

Basit bir dağılım grafiği oluşturmak için *sns.scatterplot* komutunu kullanırız ve aşağıdakiler için değerleri belirleriz:

- yatay x ekseni *(x = insurance_data ['bmi'])* ve
- dikey y ekseni *(y = insurance_data ['charges']).*

In [4]:

```
sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'])
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f113b43efd0>
```

Yukarıdaki dağılım grafiği, vücut kitle indeksi (BMI) ve sigorta ücretlerinin pozitif olarak ilişkili olduğunu (pozitive correlated) ve daha yüksek BMI'li müşterilerin genellikle sigorta maliyetlerinde daha fazla ödeme yapma eğiliminde olduğunu göstermektedir.

(Yüksek BMI tipik olarak daha yüksek kronik hastalık riski ile ilişkili olduğundan, bu pattern mantıklıdır.)

Bu ilişkinin gücünü iki kez kontrol etmek için bir regresyon çizgisi veya verilere en uygun çizgiyi eklemek isteyebilirsiniz.

Bunu komutu *sns.regplot* olarak değiştirerek yaparız.

In [5]:
```
sns.regplot(x=insurance_data['bmi'], y=insurance_data['charges'])
```

Out[5]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f113b30fa20>
```

Renk Kodlu Dağılım Grafikleri

Üç değişken (iki değil, ...) arasındaki ilişkileri görüntülemek için dağılım grafiklerini kullanabiliriz! Bunu yapmanın bir yolu noktaları renklerle kodlamaktır.

Örneğin, sigaranın BMI ve sigorta maliyetleri arasındaki ilişkiyi nasıl etkilediğini anlamak için, noktaları '*smoker*' ile renk kodlaması yapabilir ve diğer iki sütunu ('*bmi*', *'charge'*) eksenler üzerinde çizebiliriz.

In [6]:
```
sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'], hue=insurance_data['smoke
r'])
```

Out[6]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f113aa99748>
```

Bu dağılım grafiği, sigara içmeyenlerin artan BMI ile biraz daha fazla ödeme yapma eğilimine sahipken, sigara içenlerin **ÇOK** daha fazla ödeme yaptığını göstermektedir.

Bu gerçeği daha da vurgulamak için, sigara içenler ve içmeyenlere karşılık gelen iki regresyon satırı eklemek için *sns.lmplot* komutunu kullanabiliriz.

Sigara içenler için regresyon çizgisinin, sigara içmeyenler için olan çizgiye göre çok daha dik bir eğime sahip olduğunu göreceksiniz!

In [7]:
```
sns.lmplot(x="bmi", y="charges", hue="smoker", data=insurance_data)
```

Out[7]:
```
<seaborn.axisgrid.FacetGrid at 0x7f113aa13518>
```

Son olarak, öğreneceğiniz ve dağılım grafiklerinde görmeye alışık olduğunuzdan biraz farklı olabilecek bir konu daha var.

Genellikle, iki sürekli değişken *("bmi" ve "charge")* arasındaki ilişkiyi vurgulamak için dağılım grafikleri kullanırız.

Bununla birlikte, dağılım grafiğinin tasarımını ana eksenlerden birinde kategorik bir değişken *("smoker" gibi)* içerecek şekilde uyarlayabiliriz.

Bu çizim türüne **categorical scatter plot** olarak değineceğiz ve bunu *sns.swarmplot* komutuyla oluşturuyoruz.

In [8]:

```
sns.swarmplot(x=insurance_data['smoker'],
              y=insurance_data['charges'])
```

Out[8]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f1139192160>
```

Diğer şeylerin yanı sıra, bu grafik bize şunları gösteriyor:

- ortalama olarak, sigara içmeyenler, sigara içenlerden daha az ücretlendirilir ve
- en fazla ödeme yapan müşteriler sigara içiyor; en az ödeme yapan müşteriler sigara içmeyen kişilerdir.

Exercise: Scatter Plots

Bu alıştırmada, yeni bilginizi gerçek dünya senaryosuna çözüm önermek için kullanacaksınız. Başarılı olmak için verileri Python'a aktarmanız, verileri kullanarak soruları yanıtlamanız ve verilerdeki patternleri anlamak için dağılım grafikleri oluşturmanız gerekir.

Senaryo

Büyük bir şeker üreticisi için çalışıyorsunuz ve hedefiniz, şirketinizin bir sonraki ürününün tasarımını yönlendirmek için kullanabileceği bir rapor yazmak. Araştırmanıza başladıktan kısa bir süre sonra, eğlenceli bir anketten en sevdiğiniz şekerleri kitleye çıkarmak için sonuçları içeren bu çok ilginç veri kümesiyle karşılaşıyorsunuz.

In [1]:

```
import pandas as pd
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup Complete")
```

```
Setup Complete
```

Step 1: Veri Yükleme

In [3]:

```
# Path of the file to read
candy_filepath = "../input/candy.csv"

# Fill in the line below to read the file into a variable candy_data
candy_data = pd.read_csv(candy_filepath, index_col="id")

# Run the line below with no changes to check that you've loaded the data correctly
step_1.check()
```

Step 2: Verileri İnceleyin

In [5]:

```
# Print the first five rows of the data
candy_data.head() # Your code here
```

Out[5]:

	competitorname	chocolate	fruity	caramel	peanutyalmondy	nougat	crispedricewafer	hard	bar	pluribus	sugarpe
id											
0	100 Grand	Yes	No	Yes	No	No	Yes	No	Yes	No	0.732
1	3 Musketeers	Yes	No	No	No	Yes	No	No	Yes	No	0.604
2	Air Heads	No	Yes	No	No	No	No	No	No	No	0.906
3	Almond Joy	Yes	No	No	Yes	No	No	No	Yes	No	0.465
4	Baby Ruth	Yes	No	Yes	Yes	Yes	No	No	Yes	No	0.604

Out[5]:

fruity	caramel	peanutyalmondy	nougat	crispedricewafer	hard	bar	pluribus	sugarpercent	pricepercent	winpercent
No	Yes	No	No	Yes	No	Yes	No	0.732	0.860	66.971725
No	No	No	Yes	No	No	Yes	No	0.604	0.511	67.602936
Yes	No	No	No	No	No	No	No	0.906	0.511	52.341465
No	No	Yes	No	No	No	Yes	No	0.465	0.767	50.347546
No	Yes	Yes	Yes	No	No	Yes	No	0.604	0.767	56.914547

Veri kümesi, her biri farklı bir şekerleme çubuğuna karşılık gelen 83 satır içerir. 13 sütun var:

- **competitorname:** Şeker çubuğunun adını içerir.
- sonraki 9 sütun ('**chocolate**''den '**pluribus**'a) şekeri tanımlar. Örneğin, çikolata şekerlemesi olan satırların "**chocolate**" sütununda "**Yes**" vardır (ve çikolata içermeyen şekerlerin aynı sütunda "**No**" değeri vardır).

- **sugarpercent:** daha yüksek değerlerin daha yüksek şeker içeriğini ifade ettiği şeker miktarının bir göstergesidir.
- **pricepercent:** veri kümesindeki diğer şekerlere göre birim fiyatı gösterir.
- **winpercent:** anket sonuçlarından hesaplanır; daha yüksek değerler, şekerin anket katılımcıları arasında daha popüler olduğunu göstermektedir.

Aşağıdaki soruları cevaplamak için verilerin ilk beş satırını kullanın.

In [6]:
```
# Fill in the line below: Which candy was more popular with survey respondents:
# '3 Musketeers' or 'Almond Joy'?  (Please enclose your answer in single quotes.)
more_popular = '3 Musketeers'

# Fill in the line below: Which candy has higher sugar content: 'Air Heads'
# or 'Baby Ruth'? (Please enclose your answer in single quotes.)
more_sugar = 'Air Heads'

# Check your answers
step_2.check()
```

Step 3: Şekerin rolü

İnsanlar şeker içeriği daha yüksek olan şekerleri tercih ediyor mu?

Part A

'*Sugarpercent*' (yatay x ekseninde) ve '*winpercent*' (dikey y ekseninde) arasındaki ilişkiyi gösteren bir dağılım grafiği oluşturun. Henüz bir regresyon çizgisi eklemeyin - bunu bir sonraki adımda yapacaksınız!

In [8]:

```
# Scatter plot showing the relationship between 'sugarpercent' and 'winpercent'
sns.scatterplot(x="sugarpercent", y="winpercent", data=candy_data) # Your code here

# Check your answer
step_3.a.check()
```

Part B

Dağılım grafiği iki değişken arasında güçlü bir korelasyon gösteriyor mu? Öyleyse, daha fazla şekerli şekerler anket katılımcıları tarafından nispeten daha mı az yoksa daha mı fazla popüler?

İpucu: Daha yüksek şeker içeriği olan şekerleri (grafiğin sağ tarafında) daha düşük şeker içeriği olan şekerlerle (grafiğin sol tarafında) karşılaştırın. Bir grup açıkça diğerinden daha popüler mi?

Çözüm: Dağılım grafiği iki değişken arasında güçlü bir korelasyon göstermiyor. İki değişken arasında net bir ilişki olmadığından, bu bize şeker içeriğinin şeker popülaritesinde güçlü bir rol oynamadığını söyler.

Step 4: Daha Yakından Bak

Part A

3. Adımda oluşturduğunuz aynı dağılım grafiğini oluşturun, ancak şimdi bir regresyon çizgisi ile!

In [12]:

```
# Scatter plot w/ regression line showing the relationship between 'sugarpercent' and 'winpercen
t'
sns.regplot(x="sugarpercent", y="winpercent", data=candy_data) # Your code here

# Check your answer
step_4.a.check()
```

Part B

Yukarıdaki tabloya göre, '*winpercent*' ve '*sugarpercent*' arasında hafif bir korelasyon var mı? Bu insanların tercih ettikleri şeker hakkında ne anlatıyor?

Çözüm: Regresyon çizgisi biraz pozitif bir eğime sahip olduğundan, bu bize 'winpercent' ve 'sugarpercent' arasında biraz pozitif bir korelasyon olduğunu söyler. Bu nedenle, insanlar nispeten daha fazla şeker içeren şekerler için hafifçe daha fazla bir tercihe sahiptir.

Step 5: Chocolate!

Aşağıdaki kod hücresinde, '*pricepercent*' (yatay x ekseninde) ve '*winpercent*' (dikey y ekseninde) arasındaki ilişkiyi göstermek için bir dağılım grafiği oluşturun. Noktaları renkle kodlamak için "*chocolate*" sütununu kullanın.

In [16]:

```
# Scatter plot showing the relationship between 'pricepercent', 'winpercent', and 'chocolate'
sns.scatterplot(x="pricepercent", y="winpercent", hue="chocolate", data=candy_data) # Your code
here

# Check your answer
step_5.check()
```

Dağılım grafiğinde ilginç patternler görebiliyor musunuz? Bir sonraki adımda regresyon çizgileri ekleyerek bu çizimi daha ayrıntılı bir şekilde araştıracağız!

Step 6: Chocolate Sütununu İnceleyelim

Part A

Adım 5'te yarattığınız aynı dağılım grafiğini oluşturun, ancak şimdi (1) çikolata şekerleme ve (2) çikolata içermeyen şekerlere karşılık gelen iki regresyon çizgisi ile.

In [18]:

```
# Color-coded scatter plot w/ regression lines
sns.lmplot(x="pricepercent", y="winpercent", hue="chocolate", data=candy_data) # Your code here

# Check your answer
step_6.a.check()
```

Part B

Regresyon çizgilerini kullanarak çikolatanın ve fiyatın şeker popülaritesi üzerindeki etkileri hakkında ne gibi sonuçlar çıkarabilirsiniz?

Çözüm: Çikolata şekerlemeleri için regresyon çizgisi ile başlayacağız. Bu çizgi biraz pozitif bir eğime sahip olduğundan, daha pahalı çikolata şekerlerinin daha popüler olma eğiliminde olduğunu söyleyebiliriz (nispeten daha ucuz çikolata şekerlerinden).
Benzer şekilde, çikolata içermeyen şekerler için regresyon çizgisinin negatif bir eğimi olduğundan, şekerler çikolata içermiyorsa, daha ucuz olduklarında daha popüler olma eğiliminde olduklarını söyleyebiliriz. Bununla birlikte, önemli bir not, veri kümesinin oldukça küçük olmasıdır - bu yüzden bu patternlere çok fazla güvenmemeliyiz! Sonuçlara daha fazla güvenmek için veri kümesine daha fazla şeker eklemeliyiz.

Step 7: Herkes çikolatayı sever.

Part A

"*Chocolate*" ve '*winpercent*' arasındaki ilişkiyi vurgulamak için kategorik bir dağılım grafiği oluşturun. (Yatay) x eksenine "*chocolate*" ve (dikey) y eksenine '*winpercent*' koyun.

In [22]:

```
# Scatter plot showing the relationship between 'chocolate' and 'winpercent'
sns.swarmplot(x="chocolate", y="winpercent", data=candy_data) # Your code here

# Check your answer
step_7.a.check()
```

Part B

Raporunuzun bir bölümünü çikolata şekerlerinin çikolata içermeyen şekerlerden daha popüler olma eğilimine adamaya karar veriyorsunuz. Bu hikayeyi anlatmak için hangi grafik daha uygundur: 6. Adımdaki çizim veya 7. Adım'daki çizim?

Çözüm: Bu durumda, Adım 7'deki kategorik dağılım grafiği daha uygun grafiktir. Her iki grafik de istenen hikayeyi anlatırken, 6. Adımdaki grafik ana noktadan uzaklaşabilecek çok daha fazla bilgi aktarmaktadır.

Distributions (Dağılımlar)

Bu eğitimde **histogramlar** ve **density plots** (yoğunluk grafikleri) hakkında her şeyi öğreneceksiniz.

In [1]:

```
import pandas as pd
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup Complete")
```

Veri Seti Seçimi

150 farklı çiçek veya üç farklı iris türünden (Iris setosa, Iris versicolor ve Iris virginica) her birinden 50 örnek olan bir veri kümesiyle çalışacağız.

Iris Setosa | Iris Versicolor | Iris Virginica

Veri Yükleme ve İnceleme

Veri kümesindeki her satır farklı bir çiçeğe karşılık gelir. Dört ölçüm vardır: petal(yaprak) uzunluğu ve genişliği ile birlikte sepal(çanak) uzunluğu ve genişliği. Ayrıca ilgili türleri de takip ediyoruz.

In [2]:

```
# Path of the file to read
iris_filepath = "../input/iris.csv"

# Read the file into a variable iris_data
iris_data = pd.read_csv(iris_filepath, index_col="Id")

# Print the first 5 rows of the data
iris_data.head()
```

Out[2]:

	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)	Species
Id					
1	5.1	3.5	1.4	0.2	Iris-setosa
2	4.9	3.0	1.4	0.2	Iris-setosa
3	4.7	3.2	1.3	0.2	Iris-setosa
4	4.6	3.1	1.5	0.2	Iris-setosa
5	5.0	3.6	1.4	0.2	Iris-setosa

Histograms

İris çiçeklerinde petal uzunluğunun nasıl değiştiğini görmek için bir histogram oluşturmak istediğimizi varsayalım. Bunu sns.distplot komutuyla yapabiliriz.

In [3]:

```
# Histogram
sns.distplot(a=iris_data['Petal Length (cm)'], kde=False)
```

Out[3]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f136e29e8d0>
```

Komutun davranışını iki ek bilgi parçasıyla özelleştiriyoruz:

- **a** = çizmek istediğimiz sütunu seçer (bu durumda '*Petal Length (cm)*' seçtik).
- **kde = False**, histogram oluştururken her zaman sağlayacağımız bir şeydir, çünkü serbest bırakmak biraz farklı bir grafik oluşturacaktır.

Density Plots (Yoğunluk Grafikleri)

Sonraki grafik türü, **kernel density estimate (KDE)** grafiğidir. KDE grafiklerine aşina değilseniz, düzgünleştirilmiş bir histogram olarak düşünebilirsiniz.

KDE grafiği oluşturmak için sns.kdeplot komutunu kullanırız. *shade=True* ayarı eğrinin altındaki alanı renklendirir (ve *data=* yukarıdaki histogramı yaptığımız gibi aynı işleve sahiptir).

In [4]:

```
# KDE plot
sns.kdeplot(data=iris_data['Petal Length (cm)'], shade=True)
```

Out[4]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f136e1e3da0>
```

2D Kde Plots

Bir KDE grafiği oluştururken tek bir sütunla sınırlı değiliz. **sns.jointplot** komutuyla iki boyutlu (2D) bir KDE grafiği oluşturabiliriz.

Aşağıdaki grafikte, renk kodlaması, şeklin daha koyu kısımlarının daha olası olduğu farklı sepal genişlik ve petal uzunluğu kombinasyonlarını görme olasılığımızı gösterir.

In [5]:

```
# 2D KDE plot
sns.jointplot(x=iris_data['Petal Length (cm)'], y=iris_data['Sepal Width (cm)'], kind="kde")
```

Out[5]:

```
<seaborn.axisgrid.JointGrid at 0x7f136e138ba8>
```

Ortadaki 2D KDE grafiğine ek olarak,

- şeklin üstündeki eğri, x eksenindeki veriler için bir KDE grafiğidir (bu durumda, *iris_data['Petal Length (cm)']*) ve
- şeklin sağındaki eğri, y eksenindeki veriler için bir KDE grafiğidir (bu durumda, *iris_data ['Sepal Width (cm)']*).

Color-coded plots (Renk Kodlu Grafikler)

Eğiticinin bir sonraki bölümü için, türler arasındaki farklılıkları anlamak için grafikler oluşturacağız. Bunu başarmak için, veri kümesini her tür için bir tane olmak üzere üç ayrı dosyaya bölerek başlıyoruz.

In [6]:

```
# Paths of the files to read
iris_set_filepath = "../input/iris_setosa.csv"
iris_ver_filepath = "../input/iris_versicolor.csv"
iris_vir_filepath = "../input/iris_virginica.csv"

# Read the files into variables
iris_set_data = pd.read_csv(iris_set_filepath, index_col="Id")
iris_ver_data = pd.read_csv(iris_ver_filepath, index_col="Id")
iris_vir_data = pd.read_csv(iris_vir_filepath, index_col="Id")

# Print the first 5 rows of the Iris versicolor data
iris_ver_data.head()
```

Out[6]:

	Sepal Length (cm)	Sepal Width (cm)	Petal Length (cm)	Petal Width (cm)	Species
Id					
51	7.0	3.2	4.7	1.4	Iris-versicolor
52	6.4	3.2	4.5	1.5	Iris-versicolor
53	6.9	3.1	4.9	1.5	Iris-versicolor
54	5.5	2.3	4.0	1.3	Iris-versicolor
55	6.5	2.8	4.6	1.5	Iris-versicolor

Aşağıdaki kod hücresinde, **sns.distplot** komutunu (yukarıdaki gibi) üç kez kullanarak her tür için farklı bir histogram oluşturuyoruz. Her histogramın göstergede nasıl görüneceğini ayarlamak için **label=** öğesini kullanırız.

In [7]:

```
# Histograms for each species
sns.distplot(a=iris_set_data['Petal Length (cm)'], label="Iris-setosa", kde=False)
sns.distplot(a=iris_ver_data['Petal Length (cm)'], label="Iris-versicolor", kde=False)
sns.distplot(a=iris_vir_data['Petal Length (cm)'], label="Iris-virginica", kde=False)

# Add title
plt.title("Histogram of Petal Lengths, by Species")

# Force legend to appear
plt.legend()
```

Out[7]:

```
<matplotlib.legend.Legend at 0x7f136dfc8400>
```

Bu durumda, gösterge grafikte otomatik olarak görünmez. Göstermeye zorlamak için (herhangi bir çizim türü için), her zaman **plt.legend ()** öğesini kullanabiliriz.

Ayrıca her tür için **sns.kdeplot** (yukarıdaki gibi) kullanarak bir KDE grafiği oluşturabiliriz. Yine, **label=** göstergedeki değerleri ayarlamak için kullanılır.

In [8]:

```
# KDE plots for each species
sns.kdeplot(data=iris_set_data['Petal Length (cm)'], label="Iris-setosa", shade=True)
sns.kdeplot(data=iris_ver_data['Petal Length (cm)'], label="Iris-versicolor", shade=Tru
e)
sns.kdeplot(data=iris_vir_data['Petal Length (cm)'], label="Iris-virginica", shade=True)

# Add title
plt.title("Distribution of Petal Lengths, by Species")
```

Out[8]:

```
Text(0.5, 1.0, 'Distribution of Petal Lengths, by Species')
```

Grafiklerde görülebilen ilginç bir pattern, bitkilerin Iris versicolor ve Iris virginica'nın taç uzunluğu için benzer değerlere sahip olduğu iki gruptan birine ait olduğu, Iris setosa'nın kendi başına bir kategoriye ait olduğudur.

Aslında, bu veri kümesine göre, herhangi bir iris bitkisini sadece taç uzunluğuna bakarak Iris setosa (Iris versicolor veya Iris virginica'nın aksine) olarak sınıflandırabiliriz: bir iris çiçeğinin taç yaprağı uzunluğu 2 cm'den azsa, büyük olasılıkla *Iris setosa* olur!

Exercise: Distributions

Bu alıştırmada, yeni bilginizi gerçek dünya senaryosuna çözüm önermek için kullanacaksınız. Başarılı olmak için verileri Python'a aktarmanız, verileri kullanarak soruları yanıtlamanız ve verilerdeki patternleri anlamak için **histogramlar** ve **yoğunluk grafikleri** oluşturmanız gerekir.

Senaryo

Aşağıdaki resme benzer şekilde, meme kanseri tümörlerinin mikroskobik görüntülerinden toplanan bilgileri içeren gerçek dünyadaki bir veri kümesiyle çalışacaksınız.

Her tümör ya **benign** = iyi huylu (kanserli olmayan) ya da **malignant** = kötü huylu (kanserli) olarak etiketlenmiştir.

Bu tür verilerin tümörleri tıbbi ortamlarda sınıflandırmak için akıllı algoritmalar oluşturmak için nasıl kullanıldığı hakkında daha fazla bilgi edinmek için [bu bağlantıdaki](#) kısa videoyu izleyin!

In [1]:

```
import pandas as pd
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup Complete")
```

```
Setup Complete
```

Step 1: Veri yükleme

In [3]:

```
# Paths of the files to read
cancer_b_filepath = "../input/cancer_b.csv"
cancer_m_filepath = "../input/cancer_m.csv"

# Fill in the line below to read the (benign) file into a variable cancer_b_data
cancer_b_data = pd.read_csv(cancer_b_filepath, index_col = "Id")

# Fill in the line below to read the (malignant) file into a variable cancer_m_data
cancer_m_data = pd.read_csv(cancer_m_filepath, index_col="Id")

# Run the line below with no changes to check that you've loaded the data correctly
step_1.check()
```

Step 2: Veri inceleme

In [5]:

```
# Print the first five rows of the (benign) data
cancer_b_data.head() # Your code here
```

Out[5]:

	Diagnosis	Radius (mean)	Texture (mean)	Perimeter (mean)	Area (mean)	Smoothness (mean)	Compactness (mean)	Concavity (mean)	Concave points (mean)	S (
Id										
8510426	B	13.540	14.36	87.46	566.3	0.09779	0.08129	0.06664	0.047810	(
8510653	B	13.080	15.71	85.63	520.0	0.10750	0.12700	0.04568	0.031100	(
8510824	B	9.504	12.44	60.34	273.9	0.10240	0.06492	0.02956	0.020760	(
854941	B	13.030	18.42	82.61	523.8	0.08983	0.03766	0.02562	0.029230	(
85713702	B	8.196	16.84	51.71	201.9	0.08600	0.05943	0.01588	0.005917	(

5 rows × 31 columns

In [6]:

```
# Print the first five rows of the (malignant) data
cancer_m_data.head() # Your code here
```

Out[6]:

	Diagnosis	Radius (mean)	Texture (mean)	Perimeter (mean)	Area (mean)	Smoothness (mean)	Compactness (mean)	Concavity (mean)	Concave points (mean)	Sy (m
Id										
842302	M	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.
842517	M	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.
84300903	M	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.
84348301	M	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.
84358402	M	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.

5 rows × 31 columns

Veri kümelerinde, her satır farklı bir görüntüye karşılık gelir. Her veri kümesinde aşağıdakilere karşılık gelen 31 farklı sütun bulunur:

- Tümörleri iyi huylu (veri kümesinde **B** olarak görünen) veya kötü huylu (**M**) olarak sınıflandıran 1 sütun (*Diagnosis*) ve
- Görüntülerden toplanan farklı ölçümleri içeren 30 sütun.

In [7]:

```
# Fill in the line below: In the first five rows of the data for benign tumors, what is th
e
# largest value for 'Perimeter (mean)'?
max_perim = 87.46

# Fill in the line below: What is the value for 'Radius (mean)' for the tumor with Id 8425
17?
mean_radius = 20.57

# Check your answers
step_2.check()
```

Step 3: Farklılıkları araştıralım

Part A

Hem benign hem de malignant tümörler için "*Area (mean)*" değerlerindeki dağılımı gösteren iki histogram oluşturmak için aşağıdaki kod hücresini kullanın.

```
# Histograms for benign and maligant tumors
sns.distplot(a = cancer_b_data["Area (mean)"], label="Benign - Area") # Your code here
(benign tumors)
sns.distplot(a = cancer_m_data["Area (mean)"], label="Malignant - Area") # Your code her
e (malignant tumors)
plt.legend()
# Check your answer
step_3.a.check()
```

Part B

Bir araştırmacı, iyi huylu ve kötü huylu tümörler arasındaki farkı anlamak için '*Area (mean)*' sütununun nasıl kullanılabileceğini belirleme konusunda size yardım eder. Yukarıdaki histogramlara dayanarak,

- Kötü huylu tümörler ortalama olarak '*Area (mean)*' (iyi huylu tümörlere göre) için daha yüksek veya daha düşük değerlere sahip mi?
- Hangi tümör tipi daha geniş bir potansiyel değer aralığına sahip gibi görünüyor?

Çözüm: Kötü huylu tümörler ortalama olarak '*Area (mean)*' için daha yüksek değerlere sahiptir. Malignant tümörler daha geniş bir potansiyel değer aralığına sahiptir.

Step 4: En işe yarar sütun

Part A

Hem benign hem de malign tümörler için 'Radius (worst)' değerlerinin dağılımını gösteren iki KDE grafiği oluşturmak için aşağıdaki kod hücresini kullanın.

```
# KDE plots for benign and malignant tumors
sns.kdeplot(data=cancer_b_data["Radius (worst)"], label="Benign - Radius (worst)", shade
=True) # Your code here (benign tumors)
sns.kdeplot(data=cancer_m_data["Radius (worst)"], label="Malignant - Radius (worst)", sh
ade=True) # Your code here (malignant tumors)
plt.legend()
# Check your answer
step_4.a.check()
```

Part B

Bir hastane yakın zamanda tümörleri yüksek doğrulukla teşhis edebilen bir algoritma kullanmaya başladı.
25 'Radius (worst)' değeri olan bir tümör verildiğinde, algoritmanın tümörü iyi huylu veya kötü huylu olarak sınıflandırmasının daha olası olduğunu düşünüyor musunuz?

Çözüm: Algoritmanın, tümörü malignant olarak sınıflandırması daha olasıdır. Bunun nedeni, malignant tümörlerin eğrisinin 25'lik bir değer etrafında benign tümörlerin eğrisinden çok daha yüksek olmasıdır - ve yüksek doğruluk elde eden bir algoritmanın verilerde bu kalıba dayalı kararlar vermesi muhtemeldir.

Choosing Plot Types and Custom Styles (Grafik Türlerini ve Özel Stilleri Seçme)

Bu mikro derste, birçok farklı grafik türünün nasıl oluşturulacağını öğrendiniz. Şimdi, grafiklerinizin stilini değiştirmek için kullanabileceğiniz bazı hızlı komutları öğrenmeden önce bilgilerinizi düzenleyeceksiniz.

Ne öğrendin?

Verilerinizin arkasındaki hikayeyi en iyi nasıl anlatacağınıza karar vermek her zaman kolay olmadığından, grafik türlerini bu konuda yardımcı olmak için üç geniş kategoriye ayırdık.

- **Trends** (Eğilimler) - Eğilim, bir değişim patterni olarak tanımlanır.
    - *sns.lineplot* - Çizgi grafikler, belirli bir süre boyunca eğilimleri göstermek için en iyisidir ve birden fazla gruptaki eğilimleri göstermek için birden çok çizgi kullanılabilir.
- **Relationship** (İlişki) - Verilerinizdeki değişkenler arasındaki ilişkileri anlamak için kullanabileceğiniz birçok farklı grafik türü vardır.
    - *sns.barplot* - Çubuk grafikler, farklı gruplara karşılık gelen miktarları karşılaştırmak için kullanışlıdır.
    - *sns.heatmap* - Sayı tablolarında renk kodlu kalıpları bulmak için ısı haritaları kullanılabilir.
    - *sns.scatterplot* - Dağılım grafikleri iki sürekli değişken arasındaki ilişkiyi gösterir; renk kodluysa, üçüncü bir kategorik değişkenle olan ilişkiyi de gösterebiliriz.
    - *sns.regplot* - Dağılım grafiğine bir regresyon çizgisi eklenmesi, iki değişken arasındaki herhangi bir doğrusal ilişkiyi görmeyi kolaylaştırır.
    - *sns.lmplot* - Dağılım grafiği birden fazla renk kodlu grup içeriyorsa, bu komut birden çok regresyon çizgisi çizmek için kullanışlıdır.

    - *sns.swarmplot* - Kategorik dağılım grafikleri, sürekli değişken ile kategorik değişken arasındaki ilişkiyi gösterir.
- **Distribution (**Dağılım) - Bir değişkende görmeyi bekleyebileceğimiz olası değerleri ve ne kadar olası olduklarını göstermek için dağılımları görselleştiririz.
    - *sns.distplot* - Histogramlar tek bir sayısal değişkenin dağılımını gösterir.
    - *sns.kdeplot* - KDE grafikleri (veya 2D KDE grafikleri) tek bir sayısal değişkenin (veya iki sayısal değişkenin) tahmini, düzgün bir dağılımını gösterir.
    - *sns.jointplot* - Bu komut, her bir değişken için karşılık gelen KDE grafikleriyle bir 2D KDE grafiğini aynı anda görüntülemek için kullanışlıdır.

Seaborn ile stilleri değiştirme

Tüm komutlar, çizimlerin her birine güzel, varsayılan bir stil sağlamıştır. Ancak, grafiklerinizin görünümünü özelleştirmek yararlı olabilir ve neyse ki, bu sadece bir satır kod daha ekleyerek gerçekleştirilebilir!

Önceki bir bölümde çizgi grafik oluşturmak için kullandığımız kodla çalışacağız. Aşağıdaki kod veri kümesini yükler ve grafiği oluşturur.

In [2]:

```
# Path of the file to read
spotify_filepath = "../input/spotify.csv"

# Read the file into a variable spotify_data
spotify_data = pd.read_csv(spotify_filepath, index_col="Date", parse_dates=True)

# Line chart
plt.figure(figsize=(12,6))
sns.lineplot(data=spotify_data)
```

Out[2]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f303783eef0>
```

Şeklin stilini sadece tek bir kod satırı ile farklı bir temayla hızla değiştirebiliriz.

In [3]:

```
# Change the style of the figure to the "dark" theme
sns.set_style("dark")

# Line chart
plt.figure(figsize=(12,6))
sns.lineplot(data=spotify_data)
```

Out[3]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f30451e2c88>
```

Seaborn'un beş farklı teması vardır: (1)"*darkgrid*", (2)"*whitegrid*", (3)"*dark*", (4)"*white*" ve (5)"*ticks*" ve yalnızca benzer bir komut kullanmanız gerekir.

Exercise

Bu alıştırmada, en sevdiğiniz renk kombinasyonlarını ve yazı tiplerini görmek için farklı grafik stillerini keşfedeceksiniz!

In [1]:

```
import pandas as pd
pd.plotting.register_matplotlib_converters()
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
print("Setup Complete")
```

```
Setup Complete
```

Önceki bölümdeki bir grafikle çalışacaksınız. Verileri yüklemek için sonraki hücreyi çalıştırın.

In [3]:
```
# Path of the file to read
spotify_filepath = "../input/spotify.csv"

# Read the file into a variable spotify_data
spotify_data = pd.read_csv(spotify_filepath, index_col="Date", parse_dates=True)
```

Seaborn Stillerini Deneyelim

Temaları denemek için aşağıdaki komutları çalıştırın.

In [4]:
```
# Change the style of the figure
sns.set_style("dark")

# Line chart
plt.figure(figsize=(12,6))
sns.lineplot(data=spotify_data)

# Mark the exercise complete after the code cell is run
step_1.check()
```

In [5]:

```
# Change the style of the figure
sns.set_style("darkgrid")

# Line chart
plt.figure(figsize=(12,6))
sns.lineplot(data=spotify_data)

# Mark the exercise complete after the code cell is run
step_1.check()
```

In [6]:

```
# Change the style of the figure
sns.set_style("whitegrid")

# Line chart
plt.figure(figsize=(12,6))
sns.lineplot(data=spotify_data)

# Mark the exercise complete after the code cell is run
step_1.check()
```

In [7]:

```
# Change the style of the figure
sns.set_style("white")

# Line chart
plt.figure(figsize=(12,6))
sns.lineplot(data=spotify_data)

# Mark the exercise complete after the code cell is run
step_1.check()
```

In [8]:

```
# Change the style of the figure
sns.set_style("ticks")

# Line chart
plt.figure(figsize=(12,6))
sns.lineplot(data=spotify_data)

# Mark the exercise complete after the code cell is run
step_1.check()
```

Quiz: Data Visualization

Global AI Hub

Kaggle Master Week-3 Q&A

Q1- Which of the following methods does not allow us to draw a subplot?

- sns.relplot()
- sns.catplot()
- sns.heatmap() ✓
- plt.subplots()

A1- sns.heatmap methodu ısı haritalarını çizmek için kullanılır. Matris tarzında verileri görselleştirmek için kullanılır.

Q2- Which statement is not true with the following code?
museum_data = pd.read_csv(museum_filepath,index_col="Date",parse_dates=True)

- When parse_dates = True, the type of Date column in museum_data becomes datetime
- The index value of the museum_data is the Date column in the csv file.
- Above code means that creating a new Date column that is not in csv and this column is defined as the index of the museum_data. ✓
- If we did not use parse_dates, the type of the Date column would not change to datetime. (Assuming that the original type of the column is not datetime)

A2- index_col parametresi dataframe'imimizin indexi olacak kolonu belirler. Burada csv'de olmayan yeni bir kolon oluşturulmaz, csv'de yer alan kolon kullanılır. parse_dates parametresi True olduğunda ise bu index kolonunun tipi datetime olur. parse_dates kullanmasaydık veya False verseydik kolonun tipi ilk halinde kalacak ve datetime olarak değişmeyecekti.

Q3- Imagine that you have a company and you would like to create a plot which shows the sales based on date. Which one of the following plot function is less likely suitable for this job?

- sns.scatterplot
- sns.barplot
- sns.lineplot
- sns.heatmap ✓

A3- çünkü heatmap'ler iki değişken arasındaki ilişkiyi gösterir. Bizim elimizde satışların sayıları ve tarihler olacağı için bir heatmap oluşturmaya gerek yok ve buna binaen diğer şıklardaki çizimler işimize daha çok yarar ve heatmap'e oranla bazı şeyleri (trend, vs.) görmemizi kolaylaştırır.

Q4- When do we use bar charts?

- To see how the data distributed.
- To see if data is a normal distribution.
- To compare categories by a feature like quantities. ✓
- To see trend of a time series data.

A4- Elimizde kategorik veri varsa ve bu verinin nümerik bir özelliğe göre karşılaştırılması isteniyorsa bar plot kullanırız.

Q5- According to the residual plot below, which statement is false?

- It suggests that the linear model would be appropriate.
- Data points distributed on a curvature. ✓
- Data points spread around randomly.
  The plot is created using seaborn.

A5- Grafikte veri ortalama değer etrafında rastgele dağılmıştır. Bu tipteki residual plot'lar, o veri özelinde lineer modeleme yapılabileceğini gösterir. sns.residplot ile de çizim yapılır.

Q6- What are the basic Data Visualization steps? Please choose the correct order from the mixed statements below.

```
I-      sns.lineplot(data=fifa_data) --> Plot the data
II-     fifa_data = pd.read_csv(fifa_filepath, index_col="Date", parse_dates=True) --> Load the data
III-    fifa_data.head() --> Examine the data
```

- I-II-III
- II-I-III
- III-II-I
- II-III-I ✓

A6- Veri görselleştirme stepleri olarak, öncelikle veriyi okuyup çekmemiz gerekiyor. Daha sonra veriyi inceleme kısmına geçiyoruz; nümerik, kategorik, eksik veri gibi kısımlara bakıyoruz. Bu eldeki bulguları daha iyi inceleyip analiz edebilmek için de görselleştirme adımına geçiyoruz.

Q7- Which one is the correct option for the below visualization?

- sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'])
- sns.lmplot(x="bmi", y="charges", hue="smoker", data=insurance_data)
- sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'], hue=insurance_data['smoker']) ✓
- sns.regplot(x=insurance_data['bmi'], y=insurance_data['charges'])

A7- Verilen grafik, scatterplot gösterimi olup, "bmi" ve "charge" özellikleri arasındaki ilişkinin sigara kullananlara (smoker) göre dağılımını göstermektedir. Referans alınan 3. Özelliği farklı renklerde göstermek için de "hue" özelliğini kullanıyoruz.

Q8- Which of the below statement is false?

- sns.lmplot - Useful for drawing multiple regression lines, if the scatter plot contains multiple, color-coded groups.
- sns.swarmplot - Useful for comparing quantities corresponding to different groups. ✓
- sns.heatmap - Used to find color-coded patterns in tables of numbers
- sns.distplot - Show the distribution of a single numerical variable

A8- Swarmplot için tanımlanan ifade, farklı gruplara göre sayısal miktarların karşılaştırılması, "barplot" tanımıdır. Swarmplot, kategorik scatterplot olarak tanımlanır yani kategorik ve sürekli değişkenler arasındaki ilişkiyi gösterir.

Q9- There are many different chart types that you can use to understand relationships between variables in your data. Which one is not used for showing a relationship?

- sns.regplot
- sns.lmplot
- sns.scatterplot
- sns.jointplot ✓

A9- Regplot, lmplot ve scatterplot değişkenler arası ilişkileri gösterirken; jointplot, bir değişkende görmeyi beklediğimiz olası değerleri ve ne kadar olası olduklarını göstermek için kullandığımız dağılım yöntemlerinden biridir ve her bir bireysel değişken için 2D gösterimi ifade eder.

Kaggle Master Final Sınavı

✓ Q1- Assume you have a dataset named museum_data and its index is Date column. When you run "museum_data.head()" statement you get the following://Image in Cell D4Which code below achieves the following requirement?"In October 2018, how many more visitors did Avila Adobe receive than the Firehouse Museum?"Note: All dates are in the format of: YYYY-MM-01 (Day is always 01 for every year and month) * 4/4

Date	Avila Adobe	Firehouse Museum	Chinese American Museum	America Tropical Interpretive Center
2014-01-01	24778	4485	1581	6502
2014-02-01	18976	4172	1785	5029
2014-03-01	25231	7082	3229	8129
2014-04-01	26989	6756	2129	2824
2014-05-01	36883	10858	3676	10894

- (●) museum_data[museum_data.index.isin(['2018-10-01'])]['Avila Adobe'].sum() - museum_data[museum_data.index.isin(['2018-10-01'])]['Firehouse Museum'].sum() ✓
- ( ) museum_data[museum_data.index.isin(['2018-10-01'])]['Firehouse Museum'].sum() - museum_data[museum_data.index.isin(['2018-10-01'])]['Avila Adobe'].sum()
- ( ) museum_data[museum_data.index.isin(['2019-10-01'])]['Firehouse Museum'].sum() - museum_data[museum_data.index.isin(['2018-10-01'])]['Chinese American Museum'].sum()
- ( ) museum_data[museum_data.index.isin(['2019-10-01'])]['Firehouse Museum'].sum() - museum_data[museum_data.index.isin(['2020-10-01'])]['Avila Adobe'].sum()

✓ Q2- What is the aim of the below code pieces? * 4/4

```
from sklearn.metrics import mean_absolute_error

predicted_home_prices = melbourne_model.predict(X)
mean_absolute_error(y, predicted_home_prices)
```

- ○ For splitting the data as test and train
- ○ For data modelling
- ○ For interpreting the data description
- ◉ For summarizing model quality ✓

✓ Q3- Which one is false about overfitting and underfitting? * 4/4

- ○ Training on too much epoch and batch size causes overfitting.
- ◉ Splitting dataset as train and test datasets will always be enough to prevent overfitting, no need for validation datasets. ✓
- ○ Insufficient training (less epoch less batch size), causes underfitting.
- ○ In overfitting accuracy will be very good at train data but will be very bad at unseen data.

✓ Q4- What do the highlighted code pieces mean? * 4/4

```
X_train_plus = X_train.copy()
X_valid_plus = X_valid.copy()
for col in cols_with_missing:
    X_train_plus[col + '_was_missing'] = X_train_plus[col].isnull()
    X_valid_plus[col + '_was_missing'] = X_valid_plus[col].isnull()
my_imputer = SimpleImputer()
imputed_X_train_plus = pd.DataFrame(my_imputer.fit_transform(X_train_plus))
imputed_X_valid_plus = pd.DataFrame(my_imputer.transform(X_valid_plus))
imputed_X_train_plus.columns = X_train_plus.columns
imputed_X_valid_plus.columns = X_valid_plus.columns
```

- ○ To make copy to avoid changing original data
- ○ For imputation
- ◉ To put removed column names back ✓
- ○ To make new columns indicating what will be imputed

✓ Q5- Which of the following is not an approach that we can use to prepare our categorical data? * 4/4

- ○ One-Hot Encoding
- ◉ Normalization ✓
- ○ Label Encoding
- ○ Drop categorical values

✓ Q6- According to the residual plot below, which statement is false? * 4/4

- ○ Data points spread around randomly.
- ○ The plot is created using seaborn.
- ○ It suggest that the linear model would be appropriate.
- ◉ Data points distributed on a curvature. ✓

✓ Q7- Which of the below statement is false? * 4/4

- ○ sns.heatmap - Used to find color-coded patterns in tables of numbers
- ○ sns.distplot - Show the distribution of a single numerical variable
- ◉ sns.swarmplot - Useful for comparing quantities corresponding to different groups ✓
- ○ sns.lmplot - Useful for drawing multiple regression lines, if the scatter plot contains multiple, color-coded groups.

✓ Q8- Which of the following statements are true about "max_depth" hyperparameter in Random Forest? * 4/4

```
I-      Lower is better parameter in case of same validation accuracy
II-     Higher is better parameter in case of same validation accuracy
III-    Increase the value of max_depth may overfit the data
IV-     Increase the value of max_depth may underfit the data
```

- ○ II, III
- ◉ I, III ✓
- ○ I, IV
- ○ II, IV

✓ Q9- Let assume, we have a data set called home_data with 3 features names; LotArea, YearBuilt, PoolArea. How do you define non-missing values for the feature LotArea? * 4/4

- ◉ non_missings = home_data["LotArea"].count() ✓
- ○ non_missings = home_data.count()
- ○ non_missings = home_data["LotArea"].mean()
- ○ non_missings = home_data.mean()

✓ Q10- Which one is a true statement about the below visual? * 4/4

- ○ AS Airline has the most delayed flights.
- ○ This shows a bar chart.
- ○ The light boxes are the desired relations if we want to define the least delayed flights.
- ◉ The months 9-11 are the best schedules in that year. ✓

✓ Q11- Which plot type does Scatterplot fall into? * 4/4

- ○ Categorical
- ○ Distribution
- ○ Regression
- ◉ Relational ✓

✓ Q12-You will build a model to predict housing prices. The model will be deployed on an ongoing basis, to predict the price of a new house when a description is added to a website. Here are four features that could be used as predictors. Which of the features is most likely to be a source of leakage? * 4/4

- ○ Whether the house has a basement
- ○ Latitude and longitude of the house
- ◉ Average sales price of homes in the same neighborhood ✓
- ○ Size of the house (in square meters)

✓ Q13- How is the Gradient Boosting cycle proceed? Please choose the correct order from the mixed statements below. * 4/4

I- We add the new model to ensemble.
II- We use the current ensemble to generate predictions for each observation in the dataset.
III- We use the loss function to fit a new model that will be added to the ensemble.

- ○ II-I-III
- ◉ II-III-I ✓
- ○ I-II-III
- ○ I-III-II

Q14- What do you think about train_X when line 1 and line 2 are executed separately? The rest of the code is exactly the same. * 4/4

```
Line 1. train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 2,shuffle=False)
Line 2. train_X, val_X, train_y, val_y = train_test_split(X, y, random_state = 1,shuffle=False)
```

- (●) They generate different random number ,but the train_X is equal to each other. ✓
- ( ) They generate different random number so the train_X is equal to each other.
- ( ) They generate different random number so the train_X differs from each other.
- ( ) They generate different same number and the train_X is equal to each other.

✓ Q15- According to the shooting clusters scheme above, for each figure which statements are true? Notice that, shooting targets are the centers. * 4/4

- ○ 1:High Bias- Low Variance 2: High Bias-High Variance 3:Low Bias-Low Variance 4:Low Bias-High Variance
- ○ 1:High Bias- High Variance 2:High Bias-Low Variance 3:Low Bias-High Variance 4:Low Bias-Low Variance
- ○ 1:Low Bias- High Variance 2:Low Bias-Low Variance 3:High Bias-High Variance 4: High Bias-Low Variance
- ◉ 1:Low Bias- Low Variance 2:Low Bias-High Variance 3:High Bias-Low Variance 4: High Bias-High Variance ✓

✓ Q16- Which of the statements below is incorrect for ensemble learning and its techniques? * 4/4

- ○ Its techniques use a combination of learning algorithms to optimize better predictive performance.
- ○ It makes the model more robust.
- ○ Typically, It reduces overfitting in the data.
- ◉ Typically, it reduces underfitting in the data. ✓

✓ Q17- Which one is the correct option for the below visualization? * 4/4

- ○ sns.lmplot(x="bmi", y="charges", hue="smoker", data=insurance_data)
- ○ sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'])
- ◉ sns.scatterplot(x=insurance_data['bmi'], y=insurance_data['charges'], hue=insurance_data['smoker']) ✓
- ○ sns.regplot(x=insurance_data['bmi'], y=insurance_data['charges'])

✓ Q18- Which of the below is/are nominal variable(s)? * 4/4

I - Gender

II - Genotype

III - Religious preference

IV- IQ

V - Income earned in a week.

- ○ I, II
- ◉ I, II, III ✓
- ○ II, III, IV
- ○ I, III, IV

✓ Q19- What is the function of parameter ci? * 4/4

- ○ Defining the plot type
- ○ To make subplot
- ○ To highlight the classes of data points
- ◉ Defining to show confidence interval ✓

✓ Q20- Imagine that you have a company and you would like to create a plot which shows the sales based on date. Which one of the following plot function is less likely suitable for this job? * 4/4

- ○ sns.lineplot
- ◉ sns.heatmap ✓
- ○ sns.barplot
- ○ sns.scatterplot

✓ Q21- Which of the following statement is inconsistent with pipelines? * 4/4

- ○ You won't need to manually keep track of your training and validation data at each step with a pipeline.
- ○ With a pipeline, we can use the cross-validation technique easily.
- ○ With pipelines, there is less probability to forget a preprocessing step.
- ◉ It's hard to productionize a model with pipelines. ✓

Q22- Which of the following statements are true about the intended use of cross-validation? * 4/4

I - To reduce randomness while measuring model performance.

II - To get a better measure of model performance.

III - To increase model's training performance.

IV - To increase MAE (mean absolute error) or MSE (mean squared error).

- (●) I, II ✓
- ( ) II, III
- ( ) II, IV
- ( ) I, IV

✓ Q23- Which of the below can be said definitely according to the results table taken from the data.describe() method? I. 75% of the values in the Rooms column are greater than 2. II. There are some houses with a land size of 0. III. There are missing values in the BuildingArea column. IV. There is no house with 9 rooms in the data set * 4/4

In [2]:
```
import pandas as pd

data = pd.read_csv("/home/fatih/Desktop/melb_data.csv")

data.describe()
```

Out[2]:

	Rooms	Price	Distance	Postcode	Bedroom2	Bathroom	Car	Landsize	BuildingArea	YearBuilt
count	13580.000000	1.358000e+04	13580.000000	13580.000000	13580.000000	13580.000000	13518.000000	13580.000000	7130.000000	8205.000000
mean	2.937997	1.075684e+06	10.137776	3105.301915	2.914728	1.534242	1.610075	558.416127	151.967650	1964.684217
std	0.955748	6.393107e+05	5.868725	90.676964	0.965921	0.691712	0.962634	3990.669241	541.014538	37.273762
min	1.000000	8.500000e+04	0.000000	3000.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1196.000000
25%	2.000000	6.500000e+05	6.100000	3044.000000	2.000000	1.000000	1.000000	177.000000	93.000000	1940.000000
50%	3.000000	9.030000e+05	9.200000	3084.000000	3.000000	1.000000	2.000000	440.000000	126.000000	1970.000000
75%	3.000000	1.330000e+06	13.000000	3148.000000	3.000000	2.000000	2.000000	651.000000	174.000000	1999.000000
max	10.000000	9.000000e+06	48.100000	3977.000000	20.000000	8.000000	10.000000	433014.000000	44515.000000	2018.000000

- ○ I, II
- ○ II, III
- ◉ I, II, III ✓
- ○ II, III, IV

✓ Q24- Which of the following statements are true about LabelEncoder and OneHotEncoder? * 4/4

I-They help us to deal with categorical values.

II-Label Encoding assigns each value to a different integer whether it is unique or not.

III-One Hot Encoding creates new column for every possible value in the original data.

IV-For large number of categorical variable count value (such as 15 different values) it is not good to use One Hot Encoder generally.

- ○ I, II, III
- ○ II, III, IV
- ◉ I, III, IV ✓
- ○ All of them

✓ Q25- Which statement is not true with the following code? museum_data = pd.read_csv(museum_filepath,index_col="Date",parse_dates=True) * 4/4

- ○ If we did not use parse_dates, the type of the Date column would not change to datetime. (Assuming that the original type of the column is not datetime)
- ◉ Above code means that creating a new Date column that is not in csv and this column is defined as the index of the museum_data. ✓
- ○ When parse_dates = True, the type of Date column in museum_data becomes datetime
- ○ The index value of the museum_data is the Date column in the csv file.

Kaynaklar

- M. Vahit Keskin'in Python: Yapay Zeka ve Veri Bilimi için Python Programlama kursu
  https://www.udemy.com/course/veri-bilimine-giris/
- Machine Learning Days – Merve Noyan – Data Visualization
  https://youtu.be/JL35pUrth4g
- Datai Team - Python: Yapay Zeka için Python Programlama (1)
  https://www.udemy.com/course/python-sfrdan-uzmanlga-programlama-1
- Machine Learning Days – Mert Cobanov – Data Preprocessing
  https://www.youtube.com/watch?v=a1vEa7jG4kE
- Machine Learning Days – Onur Sahil – Models
  https://www.youtube.com/watch?v=0rTmVg_bDMc&
- Kaggle – Intro to Machine Learning Course
  https://www.kaggle.com/learn/intro-to-machine-learning
- Machine Learning Days – Merve Noyan – Evaluation, Tuning and Regularization
  https://www.youtube.com/watch?v=rjoyM64pkiE
- M. Vahit Keskin'in Python A-Z™: Veri Bilimi ve Machine Learning kursu
  https://www.udemy.com/course/python-egitimi/
- Kaggle – Intermediate Machine Learning Course
  https://www.kaggle.com/learn/intermediate-machine-learning
- Kaggle – Data Visualization Course
  https://www.kaggle.com/learn/data-visualization