Introduction

Bir kez potansiyeli olan bir özellik kümesi belirlediyseniz, artık onları geliştirmeye başlama zamanı. Bu derste, tamamen Pandas içinde yapabileceğiniz bir dizi yaygın dönüşümü öğreneceksiniz. Eğer Pandas bilginiz paslandıysa, bu konuda harika bir kursumuz var.

Bu derste, çeşitli özellik türlerine sahip dört veri kümesi kullanacağız: **ABD Trafik Kazaları, 1985 Otomobilleri, Beton Formülasyonları ve Müşteri Yaşam Boyu Değeri**. Aşağıdaki gizli hücre, onları yükler.

import matplotlib.pyplot as plt

import numpy as np

import pandas as pd

import seaborn as sns

plt.style.use("seaborn-whitegrid")

plt.rc("figure", autolayout=True)

plt.rc(

"axes",

labelweight="bold",

labelsize="large",

titleweight="bold",

titlesize=14,

titlepad=10,

)

accidents = pd.read\_csv("../input/fe-course-data/accidents.csv")

autos = pd.read\_csv("../input/fe-course-data/autos.csv")

concrete = pd.read\_csv("../input/fe-course-data/concrete.csv")

customer = pd.read\_csv("../input/fe-course-data/customer.csv")

Yeni Özellikler Keşfetme İpuçları

* **Özellikleri anlayın.** Mümkünse veri kümenizin **veri dokümantasyonuna** başvurun.
* **Alan bilgisi** edinmek için problem alanını araştırın. Örneğin, eğer probleminiz ev fiyatlarını tahmin etmekse, emlak sektörü hakkında biraz araştırma yapın. Wikipedia iyi bir başlangıç noktası olabilir, ancak **kitaplar ve makaleler** genellikle en iyi bilgiyi içerir.
* **Önceki çalışmaları inceleyin.** Geçmiş Kaggle yarışmalarından çıkan **çözüm yazıları** harika bir kaynaktır.
* **Veri görselleştirmesini kullanın.** Görselleştirme, bir özelliğin dağılımındaki patolojileri veya basitleştirilebilecek karmaşık ilişkileri ortaya çıkarabilir. Özellik mühendisliği süreci boyunca veri kümenizi görselleştirmeyi ihmal etmeyin.

# Mathematical Transforms

Sayısal özellikler arasındaki ilişkiler genellikle, alan araştırmalarınızın bir parçası olarak sıklıkla karşınıza çıkacak olan matematiksel formüller aracılığıyla ifade edilir. Pandas'ta, sütunlara tıpkı sıradan sayılarmış gibi aritmetik işlemler uygulayabilirsiniz.

**Otomobil** veri kümesinde, bir arabanın motorunu tanımlayan özellikler bulunmaktadır. Yapılan araştırmalar, potansiyel olarak faydalı yeni özellikler oluşturmak için çeşitli formüller sunar. Örneğin, "**stroke ratio**" (strok oranı), bir motorun ne kadar verimli olduğunu performansına kıyasla ölçen bir ölçüttür.

autos["stroke\_ratio"] = autos.stroke / autos.bore

autos[["stroke", "bore", "stroke\_ratio"]].head()

| stroke | bore | stroke\_ratio |
| --- | --- | --- |
| 0 | 2.68 | 3.47 | 0.772334 |
| 1 | 2.68 | 3.47 | 0.772334 |
| 2 | 3.47 | 2.68 | 1.294776 |
| 3 | 3.40 | 3.19 | 1.065831 |
| 4 | 3.40 | 3.19 | 1.065831 |

Bir kombinasyon ne kadar karmaşıksa, modelin öğrenmesi de o kadar zor olacaktır; tıpkı bir motorun gücünün bir ölçüsü olan "deplasman" formülü gibi:

autos["displacement"] = (

np.pi \* ((0.5 \* autos.bore) \*\* 2) \* autos.stroke \* autos.num\_of\_cylinders

)

Veri görselleştirme, genellikle bir özelliğin kuvvetleri veya logaritmaları aracılığıyla "yeniden şekillendirilmesi" olan dönüşümler önerebilir. Örneğin, **ABD Kazaları** veri kümesindeki **Rüzgar Hızı** (WindSpeed) dağılımı oldukça çarpıktır. Bu durumda, logaritma onu normalleştirmede etkilidir.

*# If the feature has 0.0 values, use np.log1p (log(1+x)) instead of np.log*

accidents["LogWindSpeed"] = accidents.WindSpeed.apply(np.log1p)

*# Plot a comparison*

fig, axs = plt.subplots(1, 2, figsize=(8, 4))

sns.kdeplot(accidents.WindSpeed, shade=True, ax=axs[0])

sns.kdeplot(accidents.LogWindSpeed, shade=True, ax=axs[1]);

/opt/conda/lib/python3.7/site-packages/ipykernel\_launcher.py:6: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.

This will become an error in seaborn v0.14.0; please update your code.

/opt/conda/lib/python3.7/site-packages/ipykernel\_launcher.py:7: FutureWarning:

`shade` is now deprecated in favor of `fill`; setting `fill=True`.

This will become an error in seaborn v0.14.0; please update your code.

import sys

Veri Temizleme'deki normalizasyon dersimize göz atın. Burada ayrıca çok genel bir normalleştirici türü olan Box-Cox dönüşümü hakkında da bilgi edineceksiniz.

# Counts

### Sayısal Özellikler Arasındaki İlişkiler

Sayısal özellikler arasındaki ilişkiler genellikle **matematiksel formüller** aracılığıyla ifade edilir; bu formüllerle alan araştırmalarınız sırasında sıkça karşılaşabilirsiniz. Pandas'ta, sütunlara tıpkı sıradan sayılarmış gibi aritmetik işlemler uygulayabilirsiniz.

**Otomobil** veri kümesinde, bir arabanın motorunu tanımlayan özellikler bulunur. Yapılan araştırmalar, potansiyel olarak faydalı yeni özellikler oluşturmak için çeşitli formüller sunar. Örneğin, "**strok oranı**" (stroke ratio), bir motorun ne kadar verimli olduğunu performansına kıyasla ölçen bir değerdir.

### Veri Görselleştirme

Veri görselleştirme, genellikle bir özelliğin kuvvetleri veya logaritmaları aracılığıyla "yeniden şekillendirilmesi" olan dönüşümler önerebilir. Örneğin, **ABD Kazaları** veri kümesindeki **Rüzgar Hızı** (WindSpeed) dağılımı oldukça çarpıktır. Bu durumda, logaritma onu normalleştirmede etkilidir.

### Özelliklerin Varlığını veya Yokluğunu Sayma

Bir şeyin varlığını veya yokluğunu tanımlayan özellikler, örneğin bir hastalığın risk faktörleri gibi, genellikle kümeler halinde gelir. Bu tür özellikleri, bir **sayım** oluşturarak toplayabilirsiniz.

Bu özellikler **ikili** (1 Varlık, 0 Yokluk) veya **boolean** (True veya False) olacaktır. Python'da, boolean değerler tıpkı tam sayılar gibi toplanabilir.

**Trafik Kazaları** veri kümesinde, bir kaza yakınında bazı yol nesnelerinin olup olmadığını belirten birkaç özellik bulunmaktadır. Aşağıdaki metin, sum metodu kullanarak yakındaki toplam yol özelliklerinin sayısını oluşturacaktır:

roadway\_features = ["Amenity", "Bump", "Crossing", "GiveWay",

"Junction", "NoExit", "Railway", "Roundabout", "Station", "Stop",

"TrafficCalming", "TrafficSignal"]

accidents["RoadwayFeatures"] = accidents[roadway\_features].sum(axis=1)

accidents[roadway\_features + ["RoadwayFeatures"]].head(10)

### Veri Çerçevesi Metotları ile Sayım Yapma

Bir veri çerçevesinin kendi yerleşik metotlarını kullanarak da **boolean** (mantıksal) değerler oluşturabilirsiniz. **Beton** veri kümesinde, bir beton formülasyonundaki bileşenlerin miktarları yer alır. Birçok formülasyon, bir veya daha fazla bileşenden yoksundur (yani, bileşenin değeri 0'dır). Aşağıdaki metin, veri çerçevesinin yerleşik büyük-eşittir (gt) metodu ile bir formülasyondaki bileşenlerin sayısını sayacaktır:

components = [ "Cement", "BlastFurnaceSlag", "FlyAsh", "Water",

"Superplasticizer", "CoarseAggregate", "FineAggregate"]

concrete["Components"] = concrete[components].gt(0).sum(axis=1)

concrete[components + ["Components"]].head(10)

| Cement | BlastFurnaceSlag | FlyAsh | Water | Superplasticizer | CoarseAggregate | FineAggregate | Components |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 540.0 | 0.0 | 0.0 | 162.0 | 2.5 | 1040.0 | 676.0 | 5 |
| 1 | 540.0 | 0.0 | 0.0 | 162.0 | 2.5 | 1055.0 | 676.0 | 5 |
| 2 | 332.5 | 142.5 | 0.0 | 228.0 | 0.0 | 932.0 | 594.0 | 5 |
| 3 | 332.5 | 142.5 | 0.0 | 228.0 | 0.0 | 932.0 | 594.0 | 5 |
| 4 | 198.6 | 132.4 | 0.0 | 192.0 | 0.0 | 978.4 | 825.5 | 5 |
| 5 | 266.0 | 114.0 | 0.0 | 228.0 | 0.0 | 932.0 | 670.0 | 5 |
| 6 | 380.0 | 95.0 | 0.0 | 228.0 | 0.0 | 932.0 | 594.0 | 5 |
| 7 | 380.0 | 95.0 | 0.0 | 228.0 | 0.0 | 932.0 | 594.0 | 5 |
| 8 | 266.0 | 114.0 | 0.0 | 228.0 | 0.0 | 932.0 | 670.0 | 5 |
| 9 | 475.0 | 0.0 | 0.0 | 228.0 | 0.0 | 932.0 | 594.0 | 4 |

# Building-Up and Breaking-Down Features

Genellikle daha basit parçalara bölünebilen karmaşık dizeleriniz olur. Bazı yaygın örnekler:

* ID numbers: '123-45-6789'
* Phone numbers: '(999) 555-0123'
* Street addresses: '8241 Kaggle Ln., Goose City, NV'
* Internet addresses: 'http://www.kaggle.com
* Product codes: '0 36000 29145 2'
* Dates and times: 'Mon Sep 30 07:06:05 2013'

,

Bu gibi özelliklerin genellikle faydalanabileceğiniz bir yapısı vardır. Örneğin, ABD telefon numaralarının, arayanın konumunu söyleyen bir alan kodu ( '(999)' kısmı) bulunur. Her zaman olduğu gibi, biraz araştırma yapmak burada işe yarayabilir. **str** erişimcisi, **split** gibi dize metotlarını doğrudan sütunlara uygulamanızı sağlar. Müşteri Yaşam Boyu Değeri veri seti, bir sigorta şirketinin müşterilerini tanımlayan özellikler içerir. **Policy** özelliğinden, sigorta kapsamının **Type** (Türü) ve **Level** (Seviyesi) kısımlarını ayırabiliriz.

customer[["Type", "Level"]] = ( *# Create two new features*

customer["Policy"] *# from the Policy feature*

.str *# through the string accessor*

.split(" ", expand=True) *# by splitting on " "*

*# and expanding the result into separate columns*

)

customer[["Policy", "Type", "Level"]].head(10)

| Policy | Type | Level |
| --- | --- | --- |
| 0 | Corporate L3 | Corporate | L3 |
| 1 | Personal L3 | Personal | L3 |
| 2 | Personal L3 | Personal | L3 |
| 3 | Corporate L2 | Corporate | L2 |
| 4 | Personal L1 | Personal | L1 |
| 5 | Personal L3 | Personal | L3 |
| 6 | Corporate L3 | Corporate | L3 |
| 7 | Corporate L3 | Corporate | L3 |
| 8 | Corporate L3 | Corporate | L3 |
| 9 | Special L2 | Special | L2 |

Eğer kombinasyonda bir etkileşim olduğuna inanmak için sebebiniz varsa, basit özellikleri bir bileşik özelliğe de birleştirebilirsiniz:

autos["make\_and\_style"] = autos["make"] + "\_" + autos["body\_style"]

autos[["make", "body\_style", "make\_and\_style"]].head()

| make | body\_style | make\_and\_style |
| --- | --- | --- |
| 0 | alfa-romero | convertible | alfa-romero\_convertible |
| 1 | alfa-romero | convertible | alfa-romero\_convertible |
| 2 | alfa-romero | hatchback | alfa-romero\_hatchback |
| 3 | audi | sedan | audi\_sedan |
| 4 | audi | sedan | audi\_sedan |

Kaggle Learn'de Başka Yerlerde

Burada bahsetmediğimiz ve özellikle bilgi açısından zengin olan birkaç başka veri türü daha var. Neyse ki, bu konuda da size destek oluyoruz!

Tarih ve saatler için, Veri Temizleme kursumuzdaki **Tarihleri Ayrıştırma** bölümüne bakın.

Enlem ve boylamlar için, **Jeo-uzamsal Analiz** kursumuza bakın.

# Group Transforms

Son olarak, bazı kategorilere göre gruplandırılmış birden çok satırdaki bilgileri bir araya getiren **Grup dönüşümleri** vardır. Bir grup dönüşümüyle şunun gibi özellikler oluşturabilirsiniz: "bir kişinin ikamet ettiği eyaletteki ortalama gelir" veya "türe göre, hafta içi yayımlanan filmlerin oranı". Bir kategori etkileşimi keşfettiyseniz, o kategori üzerinde bir grup dönüşümü incelemek için iyi bir şey olabilir.

Bir toplama işlevi kullanarak, bir grup dönüşümü iki özelliği birleştirir: gruplandırmayı sağlayan bir kategorik özellik ve değerlerini toplamak istediğiniz başka bir özellik. "Eyalete göre ortalama gelir" için, gruplandırma özelliği olarak **State** (Eyalet), toplama işlevi olarak **mean** (ortalama) ve toplanan özellik olarak **Income** (Gelir) seçersiniz. Bunu Pandas'ta hesaplamak için **groupby** ve **transform** metotlarını kullanırız:

customer["AverageIncome"] = (

customer.groupby("State") *# for each state*

["Income"] *# select the income*

.transform("mean") *# and compute its mean*

)

customer[["State", "Income", "AverageIncome"]].head(10)

| State | Income | AverageIncome |
| --- | --- | --- |
| 0 | Washington | 56274 | 38122.733083 |
| 1 | Arizona | 0 | 37405.402231 |
| 2 | Nevada | 48767 | 38369.605442 |
| 3 | California | 0 | 37558.946667 |
| 4 | Washington | 43836 | 38122.733083 |
| 5 | Oregon | 62902 | 37557.283353 |
| 6 | Oregon | 55350 | 37557.283353 |
| 7 | Arizona | 0 | 37405.402231 |
| 8 | Oregon | 14072 | 37557.283353 |
| 9 | Oregon | 28812 | 37557.283353 |

**mean** işlevi, yerleşik bir dataframe metodu olduğundan, onu **transform** metoduna bir dize olarak iletebiliriz. Diğer kullanışlı metotlar arasında **max**, **min**, **median**, **var**, **std** ve **count** bulunur. Her bir eyaletin veri setinde ne sıklıkta geçtiğini nasıl hesaplayabileceğiniz aşağıda gösterilmiştir:

customer["StateFreq"] = (

customer.groupby("State")

["State"]

.transform("count")

/ customer.State.count()

)

customer[["State", "StateFreq"]].head(10)

| State | StateFreq |
| --- | --- |
| 0 | Washington | 0.087366 |
| 1 | Arizona | 0.186446 |
| 2 | Nevada | 0.096562 |
| 3 | California | 0.344865 |
| 4 | Washington | 0.087366 |
| 5 | Oregon | 0.284760 |
| 6 | Oregon | 0.284760 |
| 7 | Arizona | 0.186446 |
| 8 | Oregon | 0.284760 |
| 9 | Oregon | 0.284760 |

Bunun gibi bir dönüşümü, bir kategorik özellik için "frekans kodlaması" oluşturmak amacıyla kullanabilirsiniz.

Eğer eğitim ve doğrulama setlerini ayırarak kullanıyorsanız, bunların bağımsızlığını korumak için, gruplandırılmış bir özelliği yalnızca eğitim setini kullanarak oluşturmak ve ardından doğrulama setine eklemek en iyisidir. Eğitim setinde **drop\_duplicates** ile benzersiz bir değerler kümesi oluşturduktan sonra, doğrulama setinin **merge** metodunu kullanabiliriz:

*# Create splits*

df\_train = customer.sample(frac=0.5)

df\_valid = customer.drop(df\_train.index)

*# Create the average claim amount by coverage type, on the training set*

df\_train["AverageClaim"] = df\_train.groupby("Coverage")["ClaimAmount"].transform("mean")

*# Merge the values into the validation set*

df\_valid = df\_valid.merge(

df\_train[["Coverage", "AverageClaim"]].drop\_duplicates(),

on="Coverage",

how="left",

)

df\_valid[["Coverage", "AverageClaim"]].head(10)

| Coverage | AverageClaim |
| --- | --- |
| 0 | Premium | 671.603973 |
| 1 | Basic | 375.455516 |
| 2 | Basic | 375.455516 |
| 3 | Basic | 375.455516 |
| 4 | Basic | 375.455516 |
| 5 | Basic | 375.455516 |
| 6 | Basic | 375.455516 |
| 7 | Basic | 375.455516 |
| 8 | Extended | 474.483232 |
| 9 | Basic | 375.455516 |

**Özellik Oluşturma İpuçları**

Özellikler oluştururken, modelinizin kendi güçlü ve zayıf yönlerini akılda tutmak iyi bir fikirdir. İşte bazı yönergeler:

* **Lineer modeller**, toplamları ve farkları doğal olarak öğrenir, ancak daha karmaşık bir şey öğrenemez.
* Oranları öğrenmek çoğu model için zor görünür. Oran kombinasyonları genellikle kolay performans artışları sağlar.
* **Lineer modeller** ve **yapay sinir ağları**, normalize edilmiş özelliklerle genellikle daha iyi sonuç verir. Özellikle yapay sinir ağlarının, 0'dan çok uzak olmayan değerlere ölçeklenmiş özelliklere ihtiyacı vardır. Ağaç tabanlı modeller (rastgele ormanlar ve XGBoost gibi) bazen normalleştirmeden fayda sağlayabilir, ancak genellikle bu çok daha azdır.
* **Ağaç modelleri**, neredeyse her türlü özellik kombinasyonunu yakından öğrenmeyi başarabilir, ancak bir kombinasyon özellikle önemli olduğunda, özellikle veri sınırlıysa, açıkça oluşturulmuş olması yine de fayda sağlayabilir.
* Sayım verileri, bu modellerin birden fazla özellikten gelen bilgiyi aynı anda toplamak için doğal bir yola sahip olmadığından, ağaç modelleri için özellikle faydalıdır.