

İçindekiler

---Machine Learning Days---	3
Data Visualization	3
Veri Setinin Hikayesi	4
Veri Görselleştirme	7
Relational Plots with Matplotlib	7
Scatter plot with Subplots	8
Categorical Plots with Matplotlib	9
Figure Kaydetme	11
Seaborn	11
Data Visualization Quiz	20
Data Preprocessing	23
1. Adım: Büyük resime bakın!	23
NaN kontrolü	24
2. Adım: Manipülasyona Başlayın!	24
Bilgi içermeyen kolonların kaldırılması	24
Eksik değerlerin halledilmesi	26
3. Adım: Eksikleri tamamlayın!	31
1. Standardization	31
2. Kategorik Değerlerin Ayrıstırılması	34
3. Kuantizasyon veya Binning	37
Feature Selection	38
Veri Seti	38
Feature Importance	40
Correlation Matrix	42
Data Preprocessing Quiz	45
Models	49
Regression Analysis	49
Classification Analysis	50
Binary Classification	50
Multi-Class Classification	50
Linear Regression	50
Multiple Linear Regression	53
Polynomial Regression	54
Logistic Regression	56

k-Nearest Neighbor	58
Support Vector Machines.....	60
Classification.....	61
Regression	63
Desicion Trees	67
Classification.....	67
Cart	69
Regression	71
Ensemble Methods & Random Forest	74
Bagging and Pasting.....	75
Models Quiz.....	76
Evaluation, Tuning and Regularization.....	81
Kaynaklar.....	82

---Machine Learning Days---

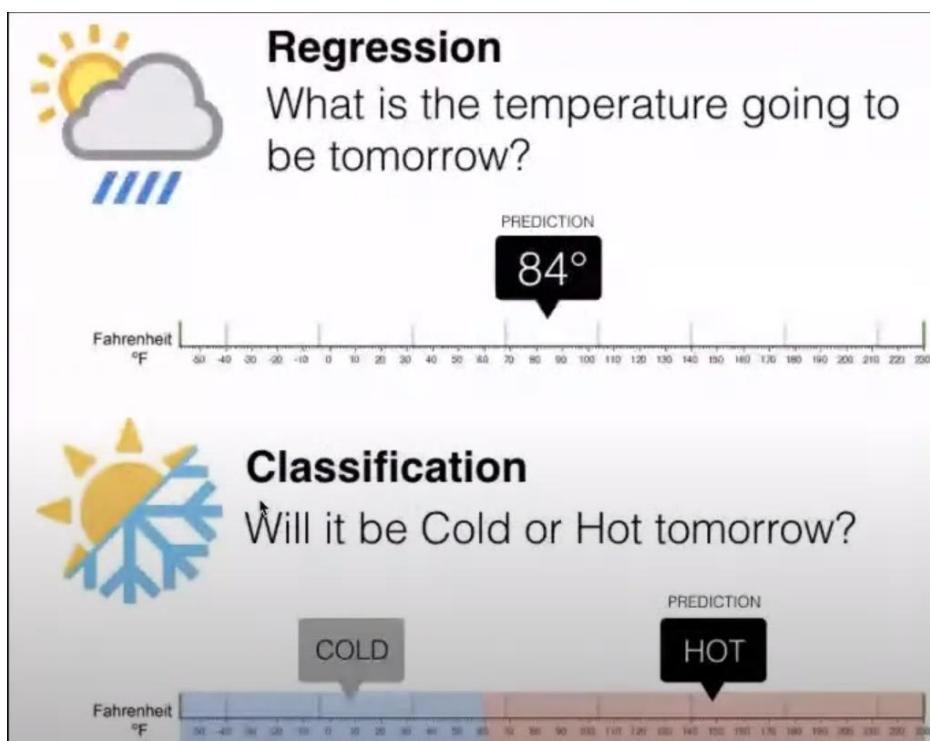
Data Visualization

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]:	<pre>import numpy as np import pandas as pd import matplotlib.pyplot as plt import seaborn as sns #kütüphanelerimizi ekledik.</pre>																																																																	
[7]:	<pre>df = pd.read_csv("kaggle/datasets_228_482_diabetes.csv") #kaggle isimli klasördeki .csv uzantılı veri setimizi aldık. df.head()</pre>																																																																	
[7]:	<table><thead><tr><th></th><th>Pregnancies</th><th>Glucose</th><th>BloodPressure</th><th>SkinThickness</th><th>Insulin</th><th>BMI</th><th>DiabetesPedigreeFunction</th><th>Age</th><th>Outcome</th></tr></thead><tbody><tr><td>0</td><td>6</td><td>148</td><td>72</td><td>35</td><td>0</td><td>33.6</td><td></td><td>0.627</td><td>50</td><td>1</td></tr><tr><td>1</td><td>1</td><td>85</td><td>66</td><td>29</td><td>0</td><td>26.6</td><td></td><td>0.351</td><td>31</td><td>0</td></tr><tr><td>2</td><td>8</td><td>183</td><td>64</td><td>0</td><td>0</td><td>23.3</td><td></td><td>0.672</td><td>32</td><td>1</td></tr><tr><td>3</td><td>1</td><td>89</td><td>66</td><td>23</td><td>94</td><td>28.1</td><td></td><td>0.167</td><td>21</td><td>0</td></tr><tr><td>4</td><td>0</td><td>137</td><td>40</td><td>35</td><td>168</td><td>43.1</td><td></td><td>2.288</td><td>33</td><td>1</td></tr></tbody></table>		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
	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 kümесinin amacı, veri kümесine dahil edilen belirli tanı ölçümülerine 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. içerir.

- **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)
- **DiabetesPedigreeFunction:** 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()
```

	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.

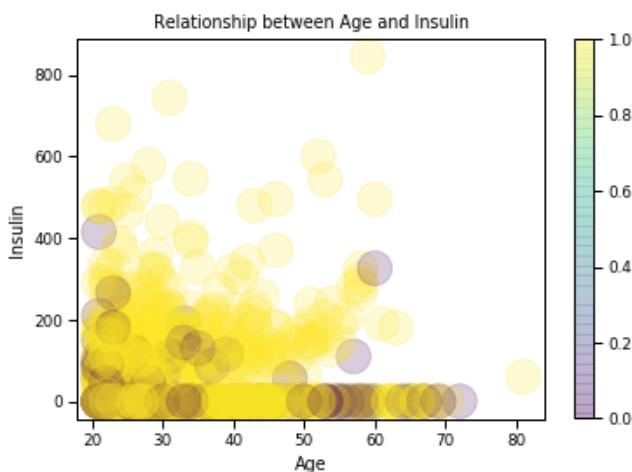
- **Scatter Plot:** İki değişken arasındaki ilişkinin dağılımını veri noktalarıyla gösterir.
- **Lineplot:** İki değişken arasındaki ilişkiyi sürekli gösterir. Veri noktaları birbirine çizgilerle bağlıdır. (Zaman serilerinde kullanılır.)
- **s parametresi:** marker boyutu
- **c parametresi:** marker rengi, hangi değişkeni tuttuğu da yazılabilir.
- **alpha:** marker opaklılığı

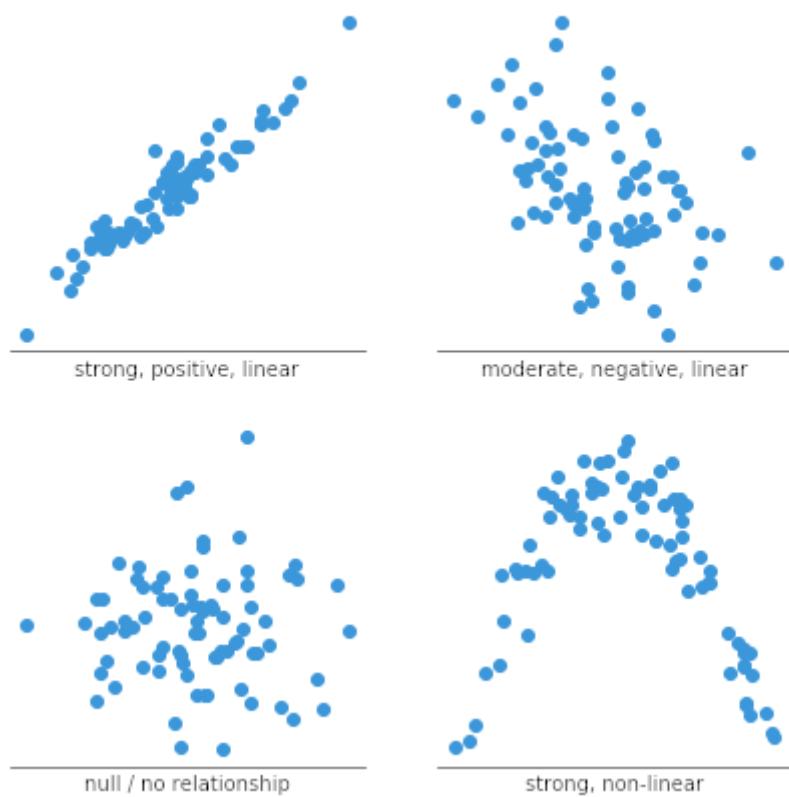
```
[11]: plt.rcParams.update({'font.size': 25})
#grafiklerimizdeki font size'ı bu şekilde güncelleyebiliriz.

[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 yanındaki ölçek
plt.xlabel("Age") #eksen ismi
plt.ylabel("Insulin")
plt.title("Relationship between Age and Insulin") #plot ismi
plt.show()

#insulin değerinde 0'da bir yüksılma var ve bu bir sıkıntı. Olmaması gereklidir.
```



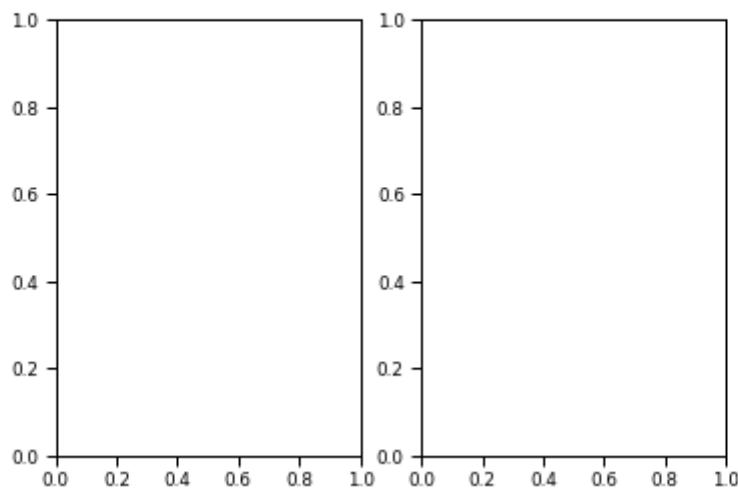


Scatter plot with Subplots

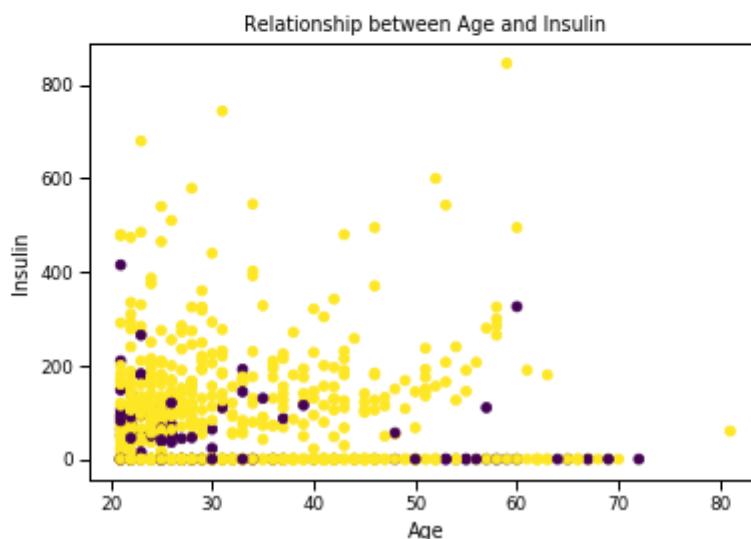
subplot'i bir plottan iki tane küçük plot çıkarıyoruz gibi düşünebiliriz.

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 satır, 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()
```

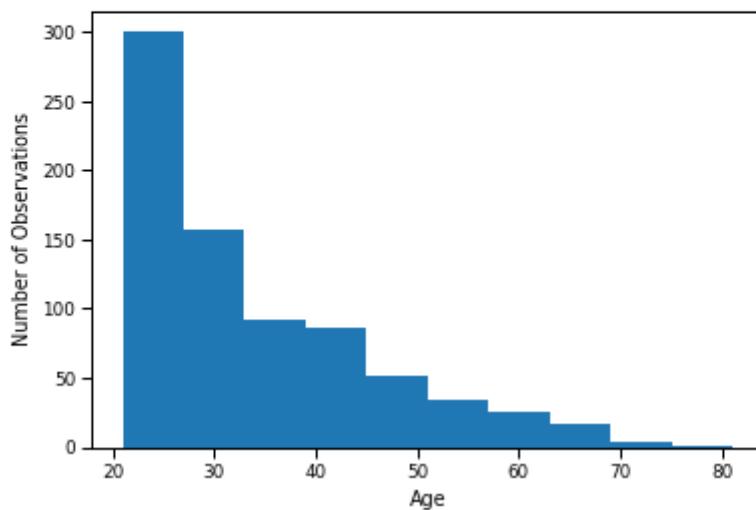


Categorical Plots with Matplotlib

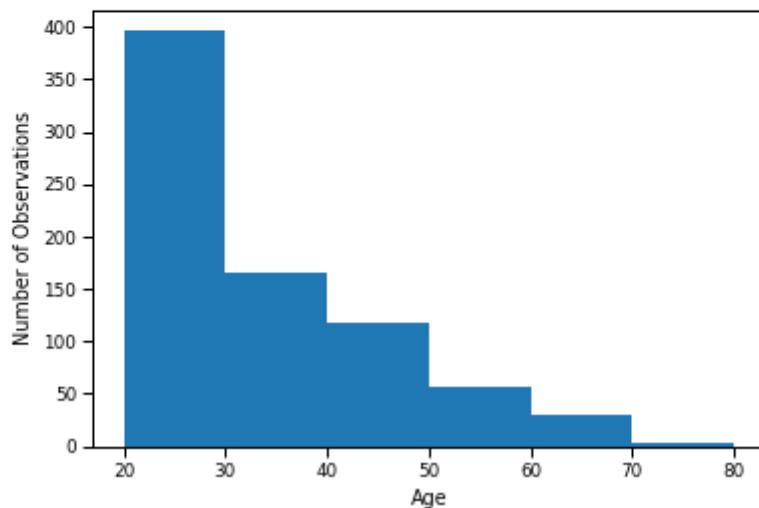
Histogram

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()
```

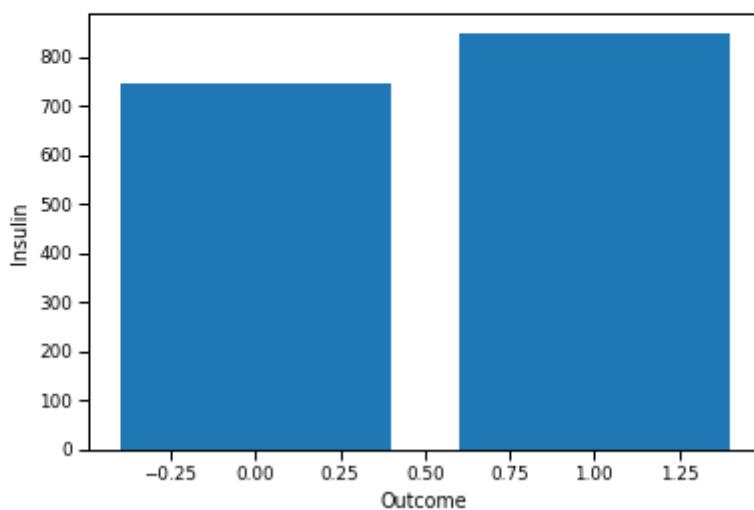
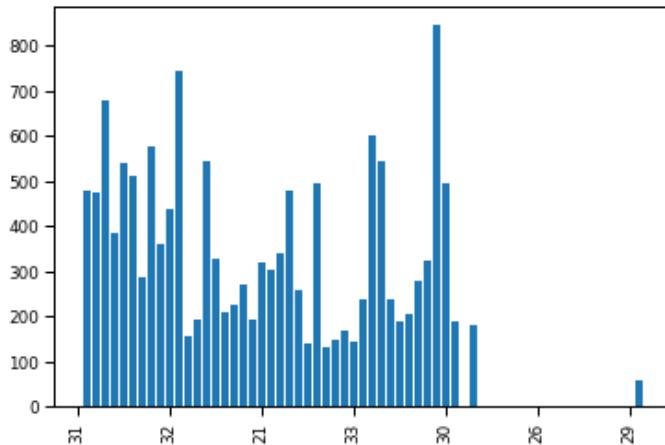


Figure Kaydetme

```
[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

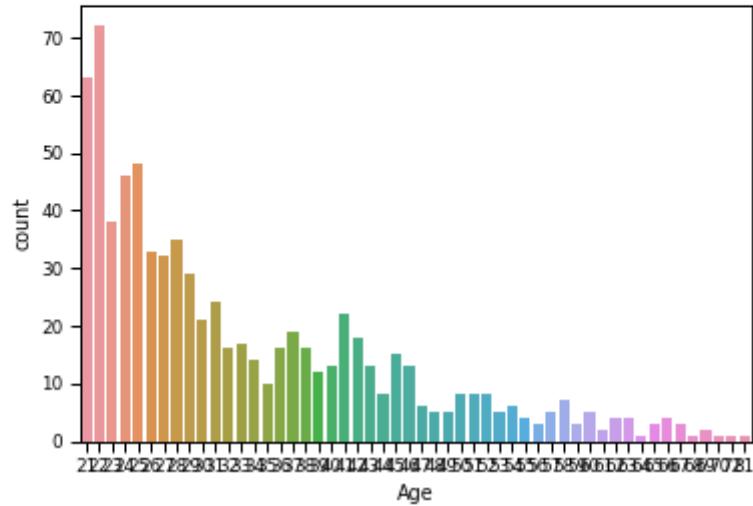
Seaborn

- **FacetGrid** (relplot(), catplot()) subplot'lar oluşturabilir.
- **AxesSubplot(scatterplot, countplot)** bir tane plot oluşturur.

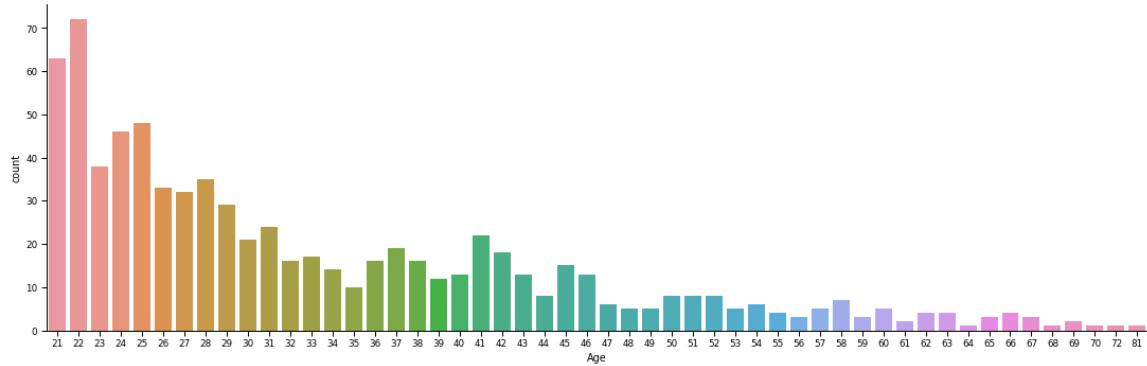
Count Plot & Cat Plot

Count Plot

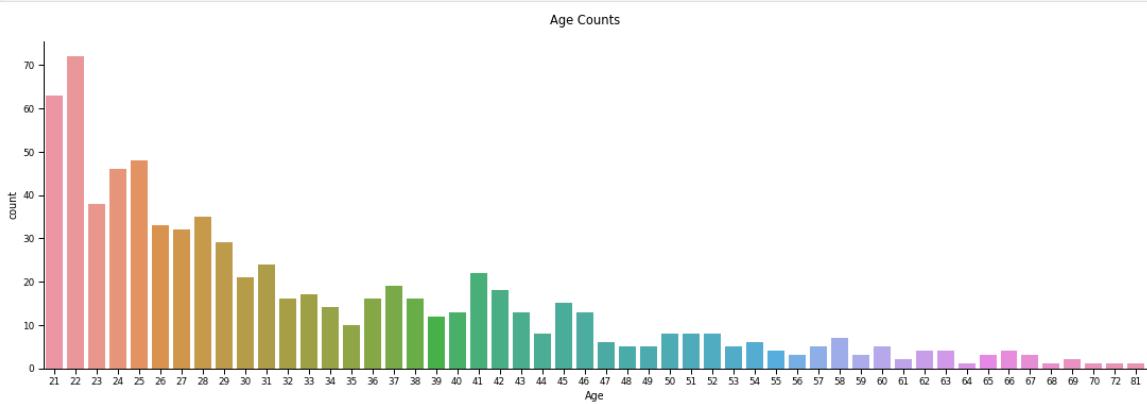
```
[60]: sns.set_palette("RdBu")
sns.countplot(x="Age", data=df)
plt.show()
```



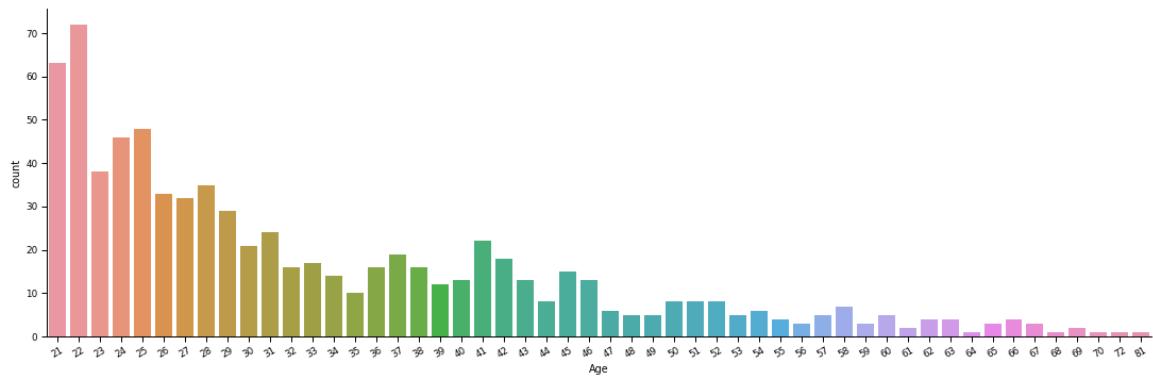
```
[68]: sns.catplot(x="Age", aspect=3, data=df, kind="count") #aspect = x eksenini, y ekseniinin 3 katı kadar olsun.
plt.show()
```



```
[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()
```



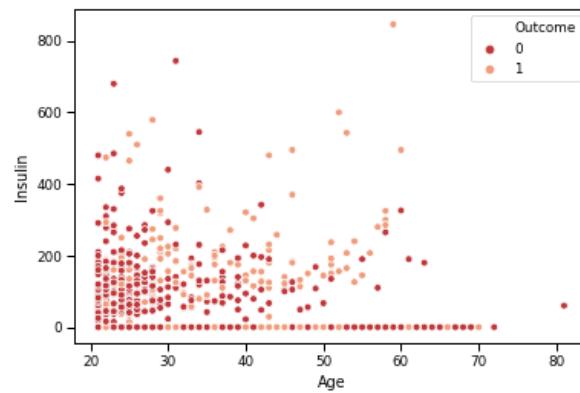
```
[71]: g = sns.catplot(x="Age", aspect=3, data=df, kind="count")
plt.xticks(rotation=30) # x eksenindeki isimleri 30 derece döndürür.
plt.show()
```



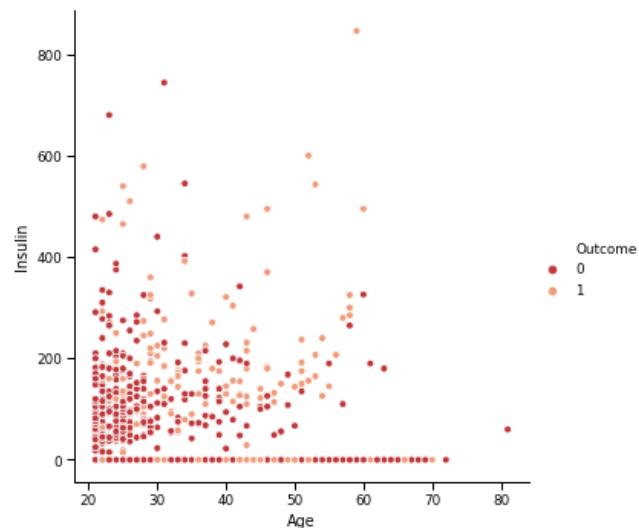
Scatter Plot

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()
```



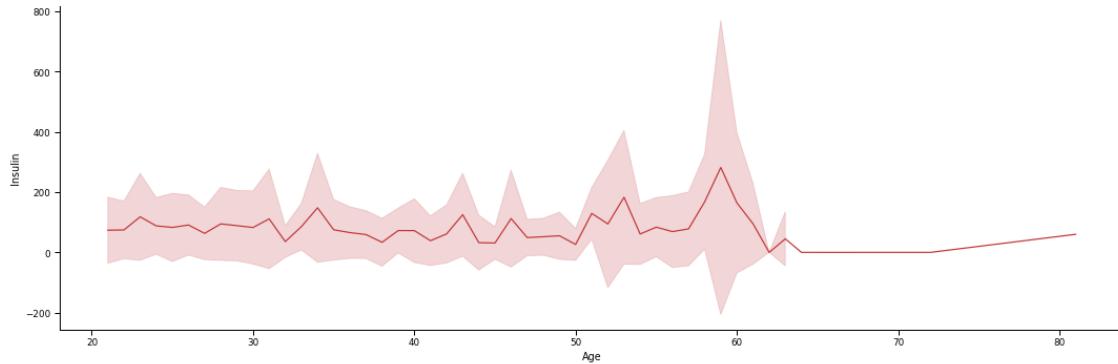
Line Plot

Line Plot

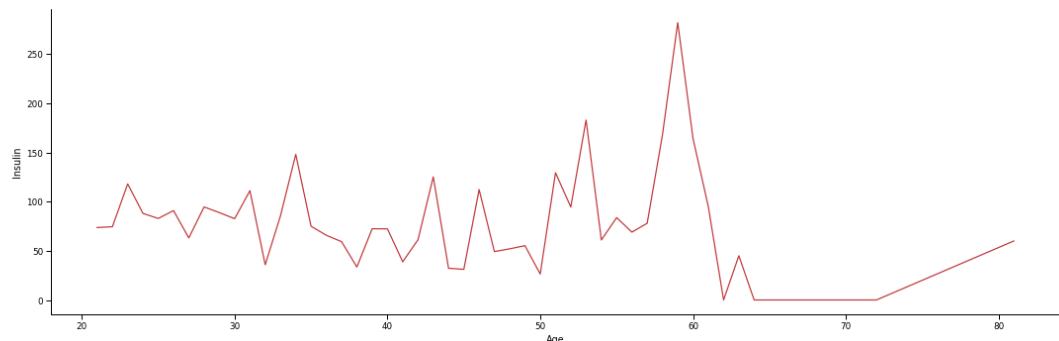
```
[83]: df.head()
```

```
[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()
```



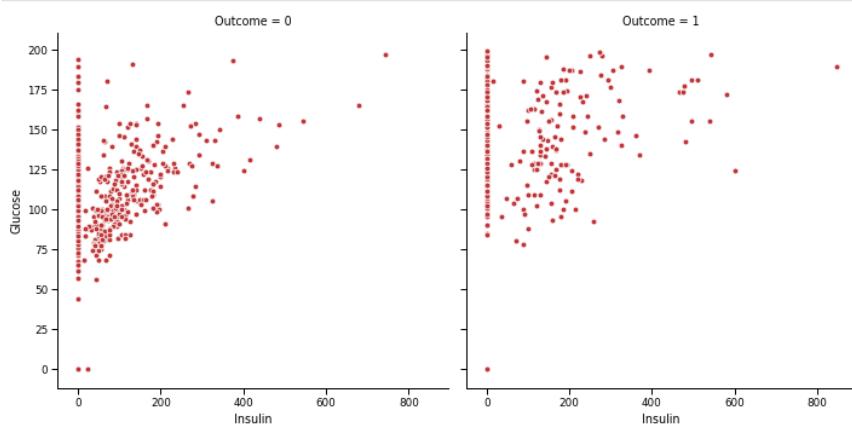
```
[98]: sns.relplot(x="Age", y="Insulin", data=df, kind="line", ci=None, aspect = 3, markers=True, dashes=False)
plt.show()
```



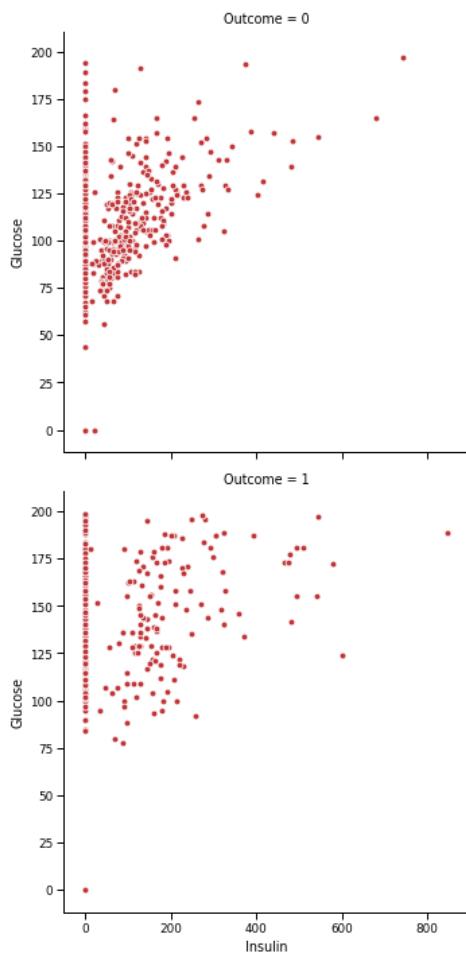
Scatter Subplots

Scatter Subplots

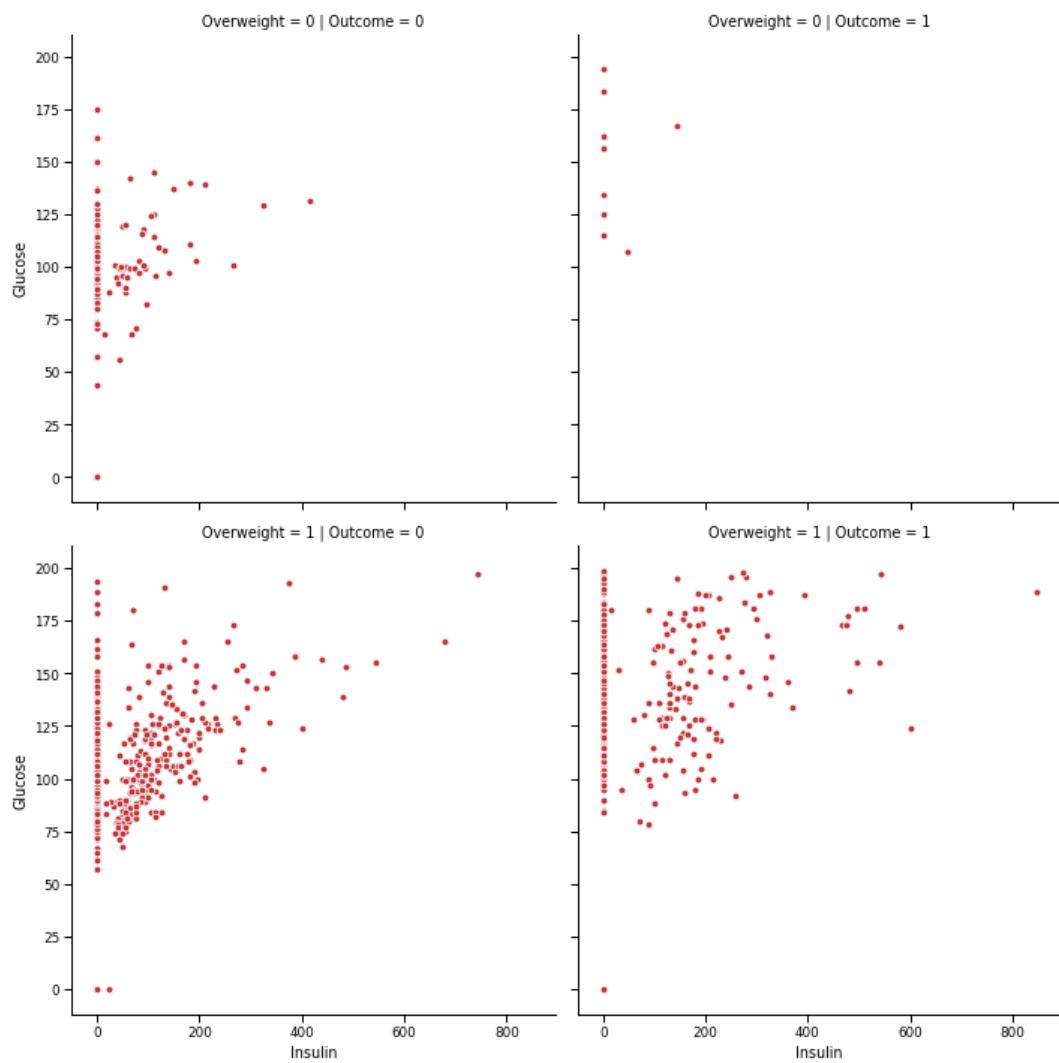
```
[101]: sns.relplot(x="Insulin", y="Glucose", data=df, kind="scatter", col="Outcome") #Glucose'a göre karşılaştırma
plt.show()
```



```
[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()
```

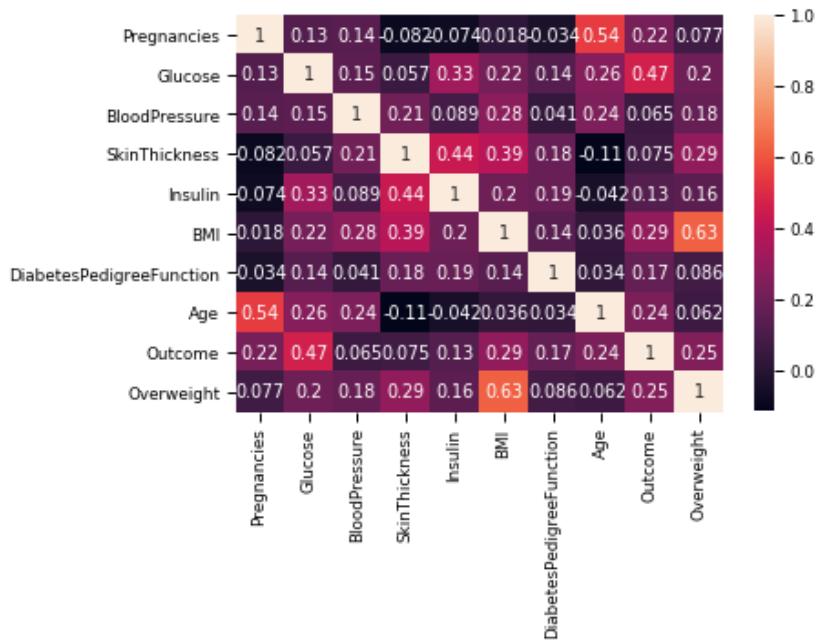


Heatmap

Öznitelikler arasındaki ilişkiye, korelasyona baktırırmızı sağlar.

Korelasyon ne kadar iyiye makine öğrenmesi modelimiz o kadar düzgün çalışır.

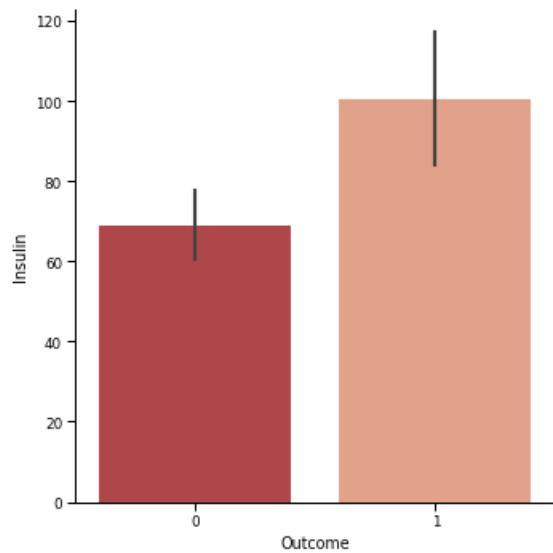
```
: sns.set_palette("RdBu")
correlation=df.corr()
sns.heatmap(correlation, annot=True) #annot: corr değerlerini heatmap üzerine yazar.
plt.show()
#Renk ne kadar açıksa correlation o kadar yüksek demektir.
```



Categoric Plot

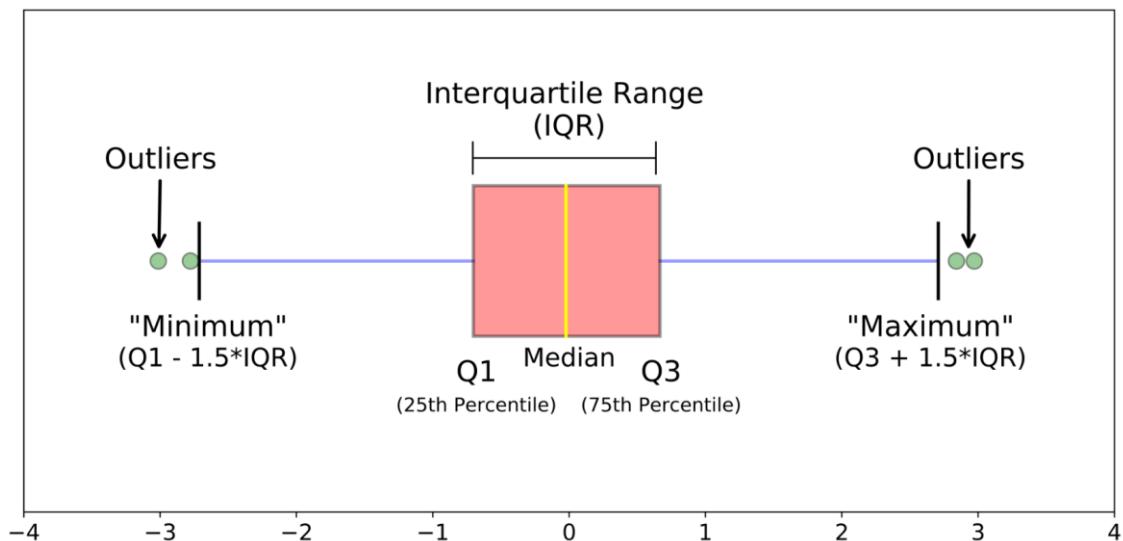
Categorical Plots

```
sns.catplot(x="Outcome",y="Insulin",data=df, kind="bar")
plt.show()
```

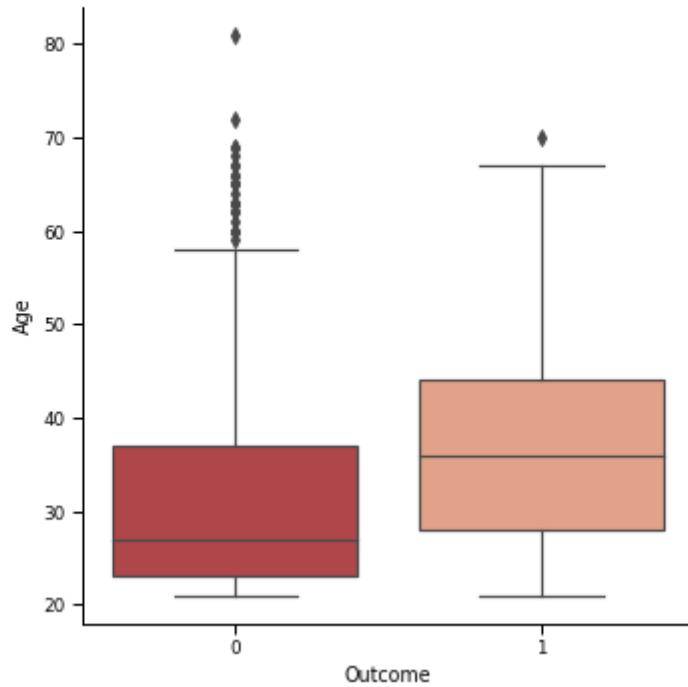


Bar plot bize kategorik veri hakkında bilgi verir.

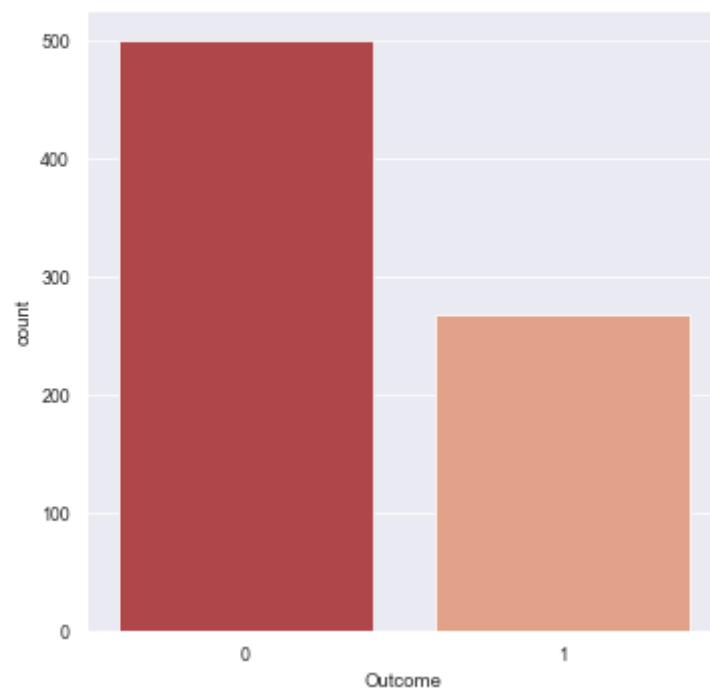
Box Plot



```
[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 istatistik 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

Tamami 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\(\)](#)

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
```

	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 baktığında yaşılardan birinin bilinmiyor olarak kodlandığını görüyorum. Eğer sayılarından 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 kaldırırmak mantıklı olacağını düşünüyorum.
- Ülke 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   Ülke     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
Ülke      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 Ülke satırlarının kaldırılmasına karar vermiştık, 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 tane den 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 direkt 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 olmayacağı.

```
[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 değil, bu yüzden ön işleme esnasında kritik konulardan birisini bu kısım oluşturuyor.

Konunun önem derecesi arttıkça yaklaşılarda değişiyor, genel bir yöntem ve herkesin kabul ettiği bir yaklaşım yok fakat size en popüler olanlarını göstermeye çalışacağım.

Bu verileri direkt olarak kaldırabileğiniz durumları yukarıda işledik, şimdi gelin kaldırılmak 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   Yaş   Şehir
 0     Mert Cobanov  24.0  Bursa
 1     Nilay Mertal  22.0  Ankara
 2  Dogancan Mavideniz  24.0  İstanbul
 3     Omer   Cengiz  23.0    NaN
 4     Merve   Noyan    NaN  İzmir
 5     Onur   Sahil  23.0  İstanbul
```

```
[40]: imp_freq = SimpleImputer(missing_values=np.nan, strategy="most_frequent")
```

```
[41]: df_2["Yaş"] = imp_freq.fit_transform(df_2[["Yaş"]])
df_2
```

```
[41]:    İsim   Soyad   Yaş   Şehir
 0     Mert Cobanov  24.0  Bursa
 1     Nilay Mertal  22.0  Ankara
 2  Dogancan Mavideniz  24.0  İstanbul
 3     Omer   Cengiz  23.0    NaN
 4     Merve   Noyan  23.0  İzmir
 5     Onur   Sahil  23.0  İstanbul
```

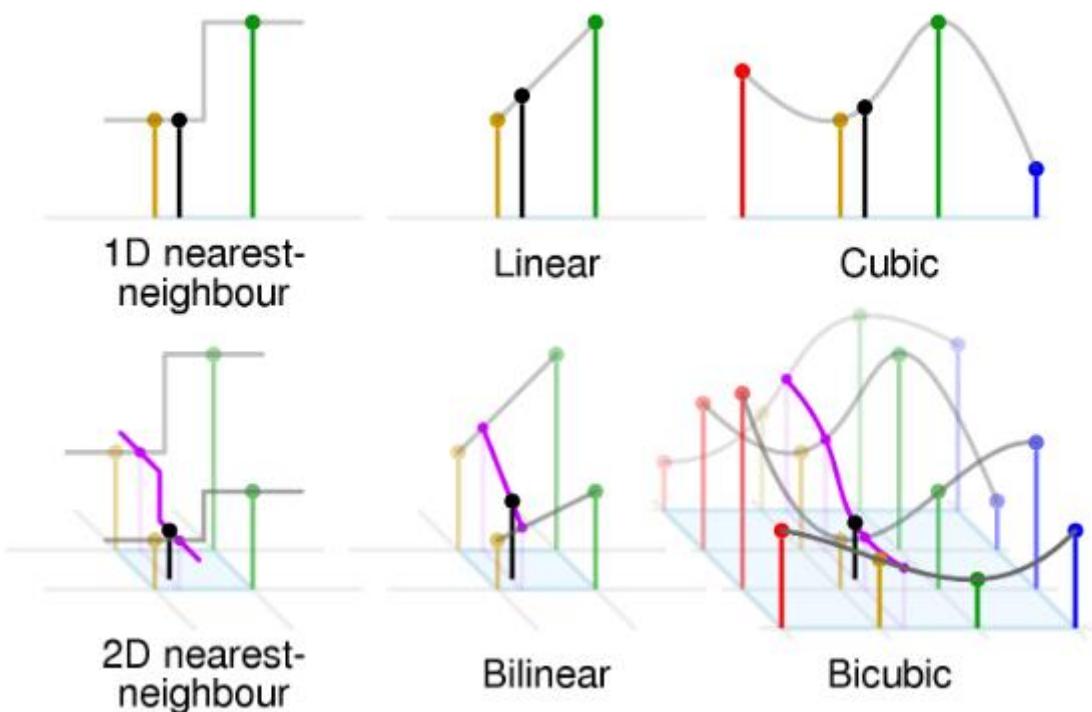
2. Enterpolasyon

Bu teknik biraz trickli olabilir, çünkü sürekli olduğunuz bir veride kullanmanız mantıklı olacaktır. Interpolasyon, elinizdeki veri noktalarının arasında bir değeri bilmemiğiniz, bu iki değer arasındaki bilinmeyen 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 görüporsunuz, lineer bir düzlemede 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.
```

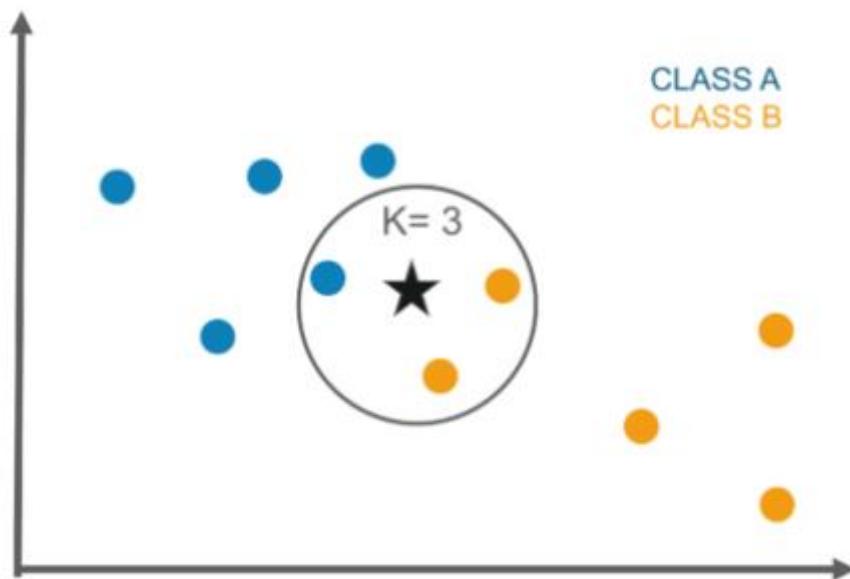
```
[43]: 0    0.0
1    1.0
2    2.0
3    3.0
dtype: float64
```

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 eksigimiz var. Burada bir karar yukarıdaki tekniklerden birini kullanmaktadır. Başka bir yaklaşım olarak burada bilinmeyen şehirlere diğer yazabiliriz.

```
[50]: df_2
```

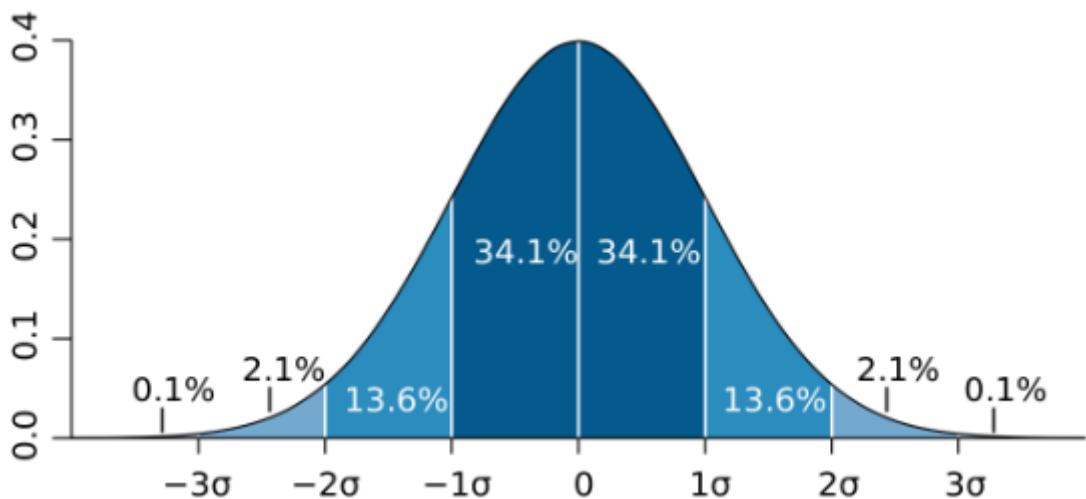
	İsim	Soyad	Yas	Sehir
0	Mert	Cobanov	24.0	Bursa
1	Nilay	Mertal	22.0	Ankara
2	Dogancan	Mavideniz	24.0	Istanbul
3	Omer	Cengiz	23.0	Nan
4	Merve	Noyan	23.0	Izmir
5	Onur	Sahil	23.0	Istanbul

```
[55]: df_2["Sehir"] = df_2["Sehir"].replace(np.nan, "diğer")
df_2
```

	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
```

	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 Ayrıntırılması

2.1 Label Encoding

Bir kolonunuzdaki değerleri sıralı bir biçimde sayısal forma getirmek için kullanılır. Elinizde 4 adet şehir ismi olduğunu varsayıyalım, eğer bu değerler birçok satırda aynı isimlerle tekrarlanıyorsa, bunları

sayılar ile temsil edebilirsiniz. Aşağıdaki örnekte görebileceğiniz gibi Bursa 1 sayısı ile, Ankara 0 ile, İstanbul 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   Yaş     Şehir
 0      Mert  Cobanov  24.0   Bursa
 1      Nilay  Mertal  22.0  Ankara
 2  Dogancan  Mavideniz  24.0  İstanbul
 3      Omer   Cengiz  23.0  diğer
 4      Merve  Noyan  23.0  İzmir
 5      Onur   Sahil  23.0  İstanbul
```

```
[90]: le.fit(df_le["Sehir"])
      #Sehir kolonuna Label encoding islemi yapacagini söyleyorum.
```

```
[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
```

	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 **get_dummies** fonksiyonu ile çok daha hızlı ve rahat bir şekilde kullanabilirsiniz.

```
[99]: pd.get_dummies(df_2["Sehir"])
```

	Ankara	Bursa	Istanbul	Izmir	diger
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 **ayırık** (*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 istediğiniz seçebilirsiniz. Fonksiyon her bir kolona bakarak, n_bins sayısı kadar sınıfa bölecek ve değerlerin hangi sınıfa ait olduğunu bularak bu sayıyla temsil edecktir.

```
[106]: binarizer = preprocessing.Binarizer(threshold = 5.1) #5.1 üzerindeki tüm değerler 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 etkileyebilecek diğer parametrelerden ayrılmaması gereklidir. 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]: 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()
```

	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))

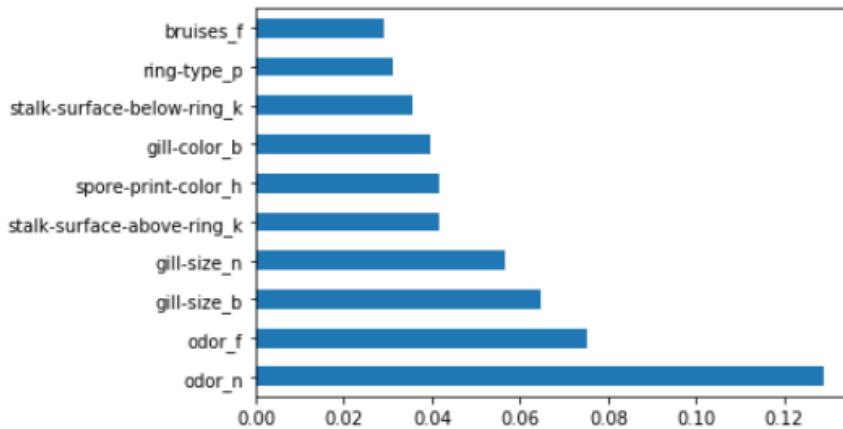
[[1274    0]
 [    0 1164]]
          precision    recall  f1-score   support
           0       1.00     1.00     1.00      1274
           1       1.00     1.00     1.00      1164
    accuracy                           1.00      2438
   macro avg       1.00     1.00     1.00      2438
weighted avg       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 görselleştireyorum.

```
[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 satır sahip bir verisetiyle saatlerce eğittiğiniz bir model olduğunu düşünürseniz kesinlikle gireceğiniz bir tradeoff olacaktır.

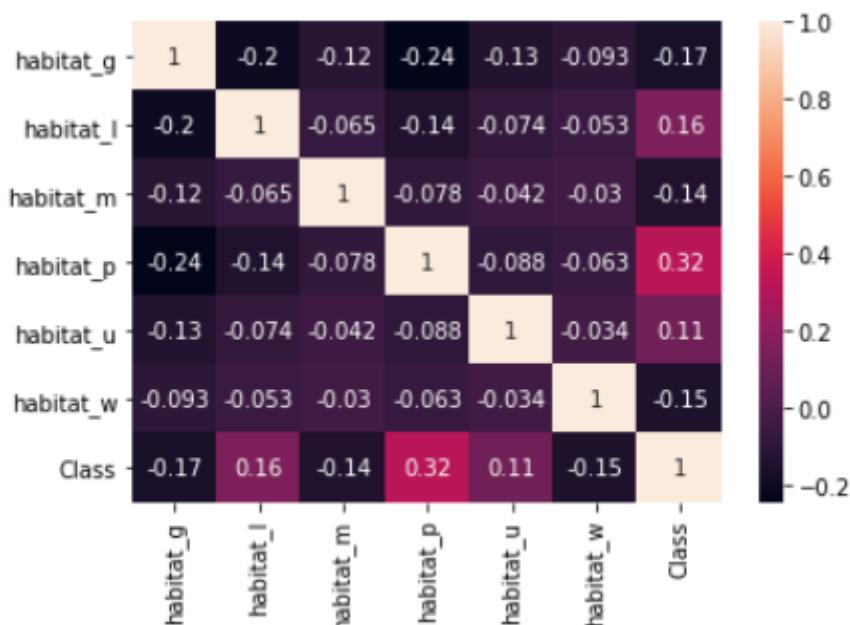
Correlation Matrix

```
[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
```

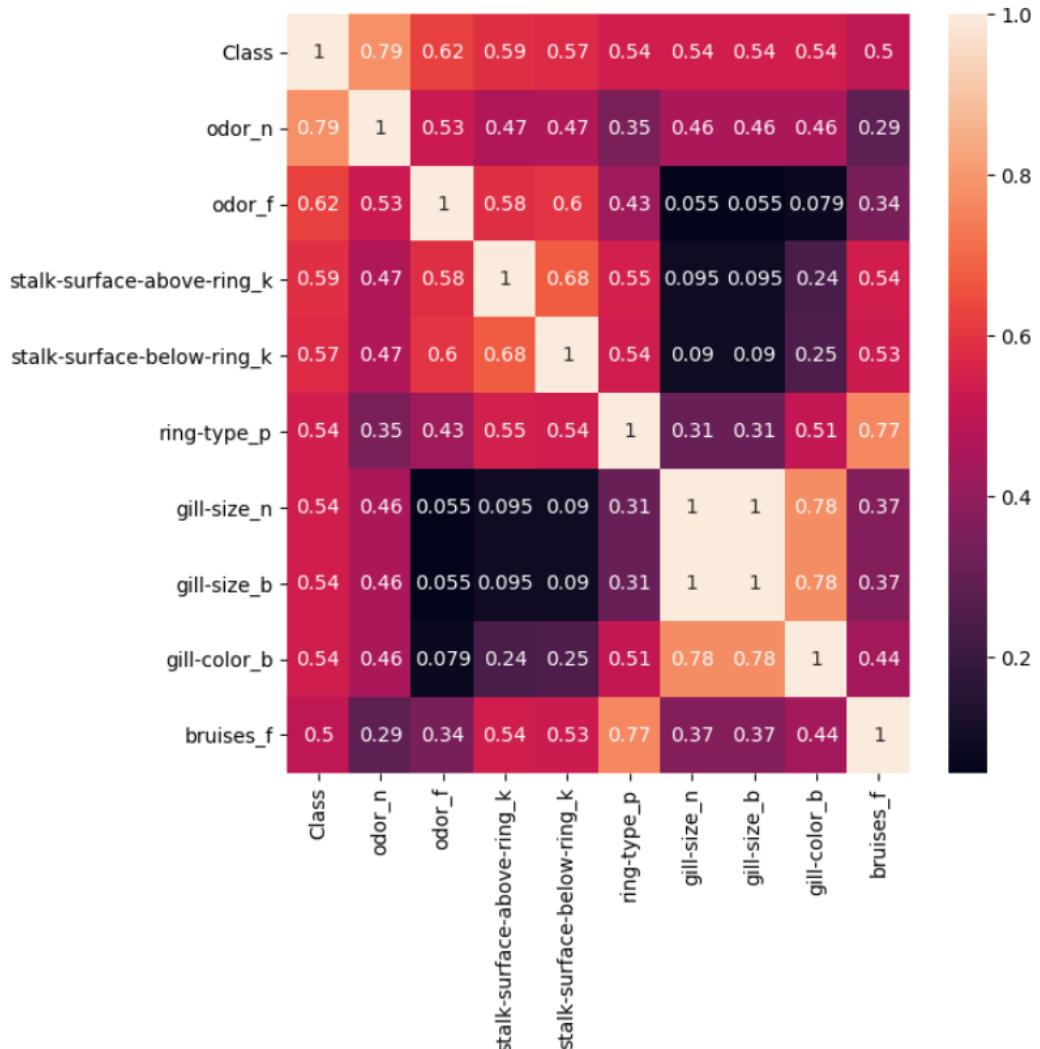
Bu zamana kadar yazdığımız kısmın sonunda index metodunu ekleyerek sadece kolon isimlerini istiyorum ve bunu ana datasetimizden başka bir değişkene aktarıyorum. Birazdan sadece bu kısmını kullanıyor olacağız, bu sayede daha okunaklı ve en yüksek 10 korelasyon değerine sahip kolon ile birlikte çalışıyor 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 20/20 bir teknik değildir? *

- KNN
- Ortalama
- Standart Sapma ✓
- Enterpolasyon

2.

- I. Standard Scaler verilen bir dizinin ortalamasını 1 ve standart sapmasını 0 yapar.
- II. Dağılımı küçük bir aralıktaki 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. İlk 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

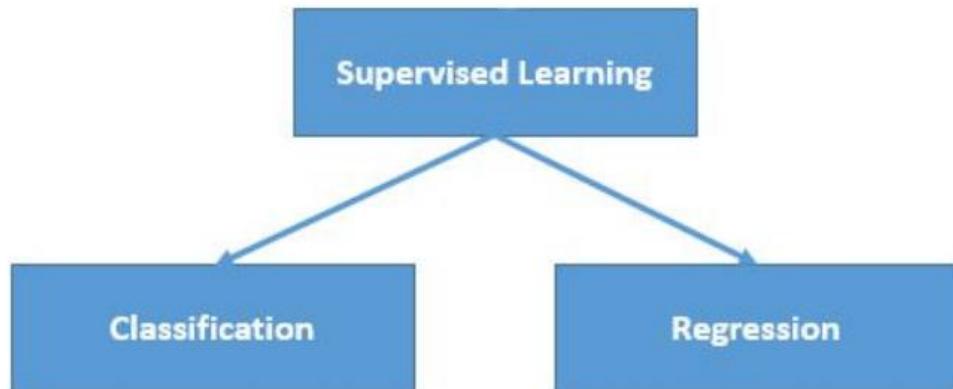
II, IV, V

I, III, V 

Hepsi

Models

Supervised Learning



Supervised; Bize doğru cevabı verilmiş bir veri setiyle yaptığımz öğrenme çeşididir.

Regression Analysis

Regression Analysis

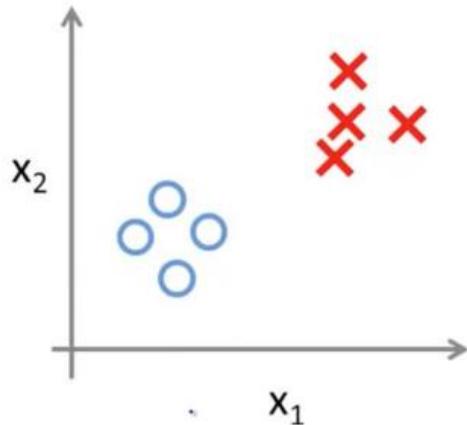
Bağımlı Değişken	Bağımsız Değişken
<ul style="list-style-type: none">Bu değişken tahmin etmeye çalıştığımız değişkendir	<ul style="list-style-type: none">Bu değişken tahmin etmek için kullandığımız giriş değişkenidir
<ul style="list-style-type: none">“y” olarak ifade edilir	<ul style="list-style-type: none">“X” olarak ifade edilir

** Bir regresyon probleminde sürekli aralıkta olan sonuçları tahmin etmeye çalışırız.

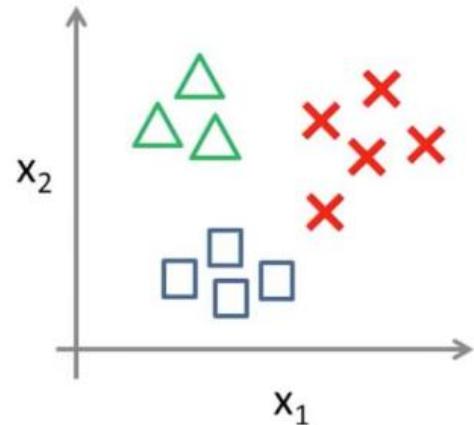
Classification Analysis

Classification Analysis

Binary classification:



Multi-class classification:



Spesifik sınıfları tahmin etmek için oluşturduğumuz algoritmalarıdır.

Binary Classification

Y değerlerimiz 2 çeşit veriden oluşansa 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)

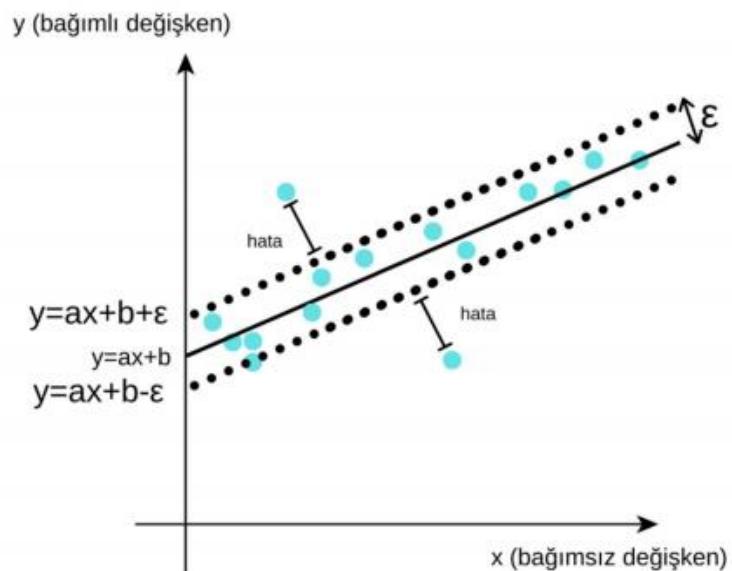
x^i => 'i' numaralı gözlem için bağımsız değişken (ev özellikleri)

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)

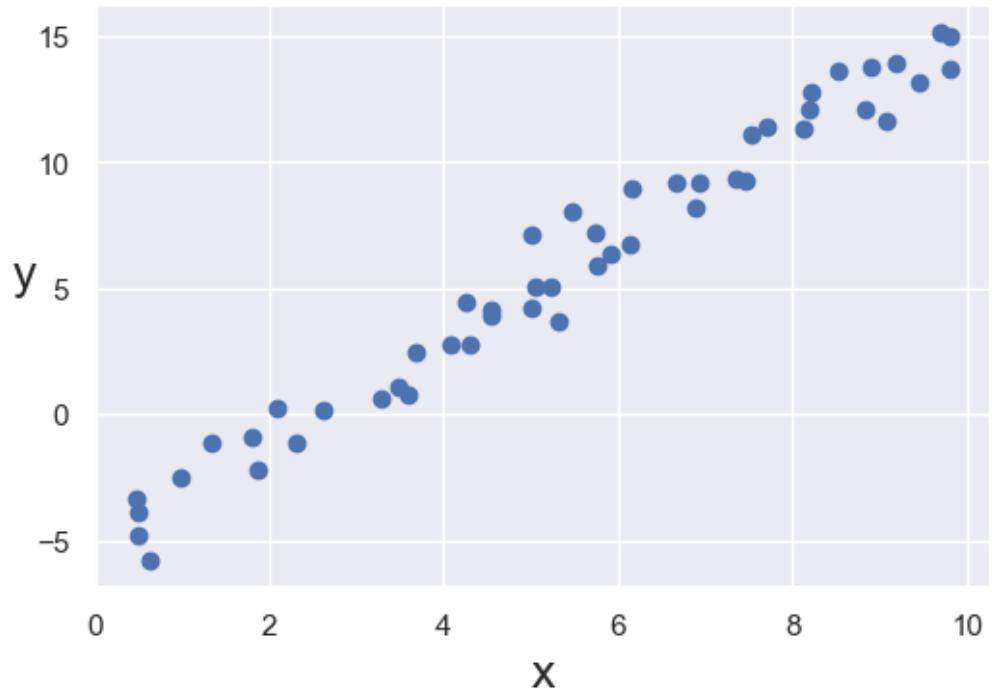
print("X\n", x, "\n")
print("Y\n", y, "\n")
```

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

$$y^i = \beta_0 + \beta_1 x_1^i + \beta_2 x_2^i + \dots + \beta_k x_k^i + \epsilon^i$$

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

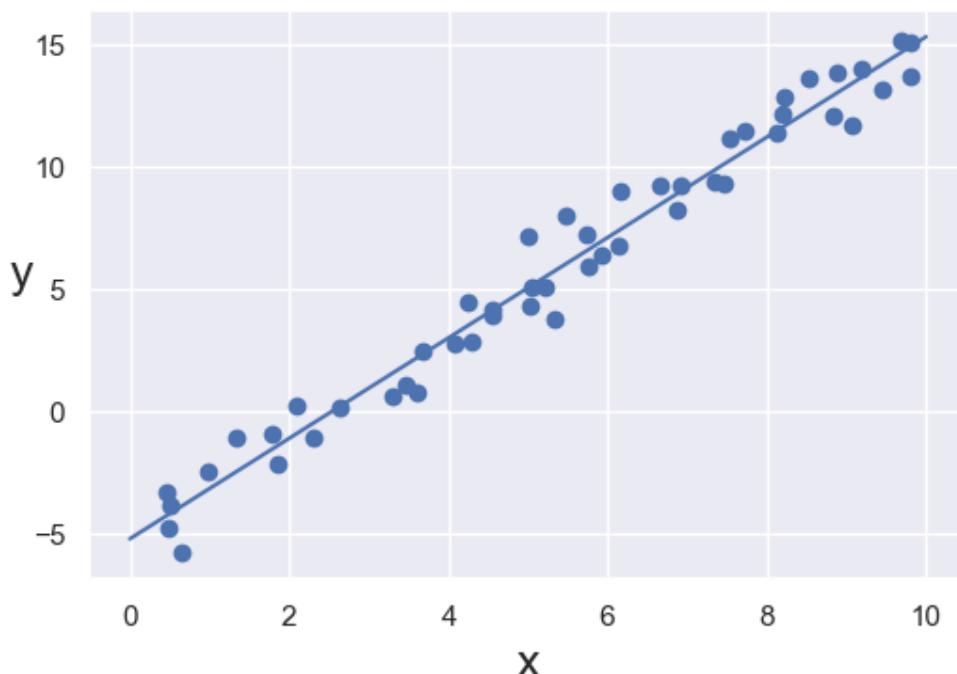
β_j => 'j' numaralı bağımsız değişken için regresyon katsayısı

```
[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 kesimini modelin fit parametrelerinde bulunur.

```
[5]: print("Model eğimi:    ", model.coef_[0])
print("Model kesişimi:", model.intercept_)

Model eğimi:    1.9464141313879109
Model kesişimi: -4.823220324774075
```

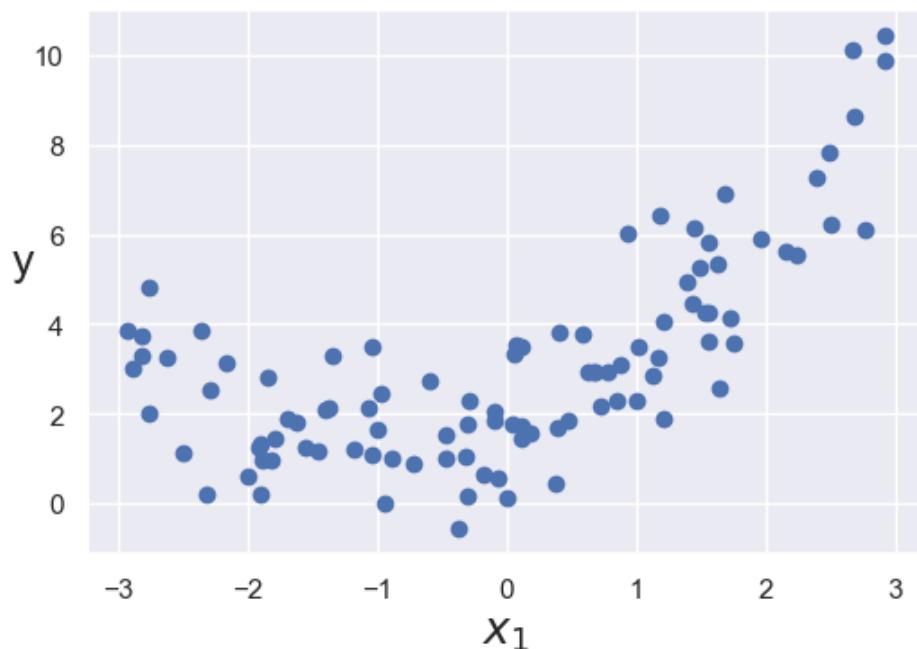
Polynomial Regression

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)
```

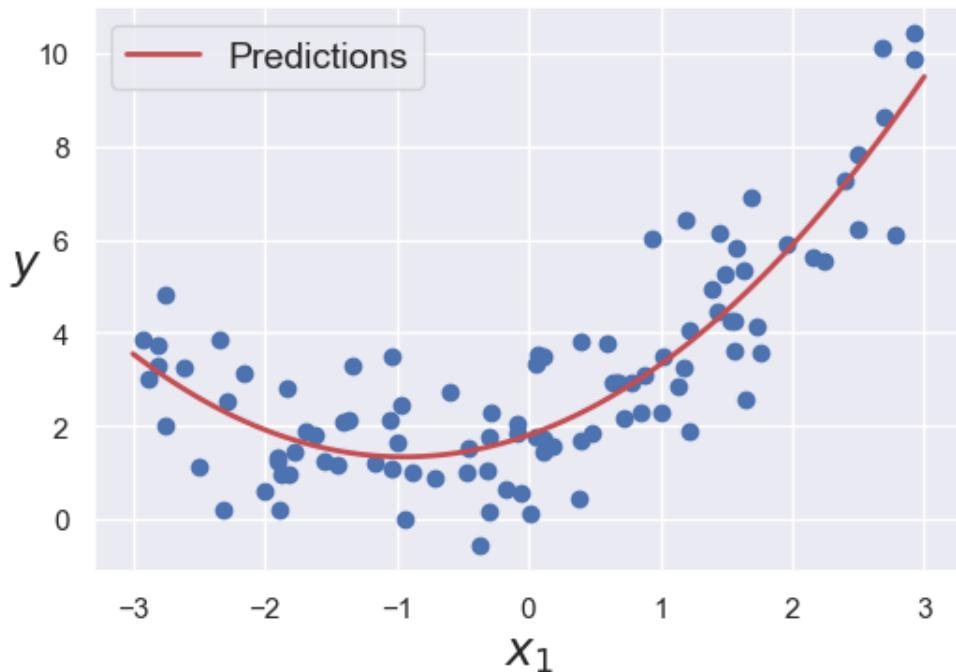
```
[7]: <matplotlib.collections.PathCollection at 0x142d619ef88>
```



```
[8]: from sklearn.pipeline import Pipeline
from sklearn.preprocessing import PolynomialFeatures

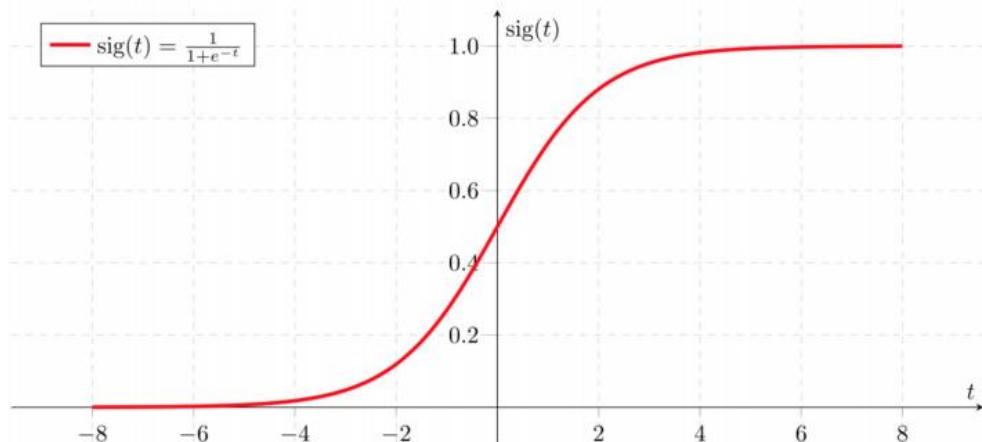
polynomial_regression = Pipeline([
    ("poly_features", PolynomialFeatures(degree=2, include_bias=False)),
    ("lin_reg", LinearRegression()),
])
polynomial_regression.fit(X, y)
X_new=np.linspace(-3, 3, 100).reshape(100, 1)
y_newbig = polynomial_regression.predict(X_new)
plt.figure(dpi=100)
plt.scatter(X, 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



Logistic Regression daha çok verdığımız instance'lardaki belirli sınıfların olasılığını hesaplamak için kullanılır.

$$\text{— } \text{sig}(t) = \frac{1}{1+e^{-t}}$$

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
```

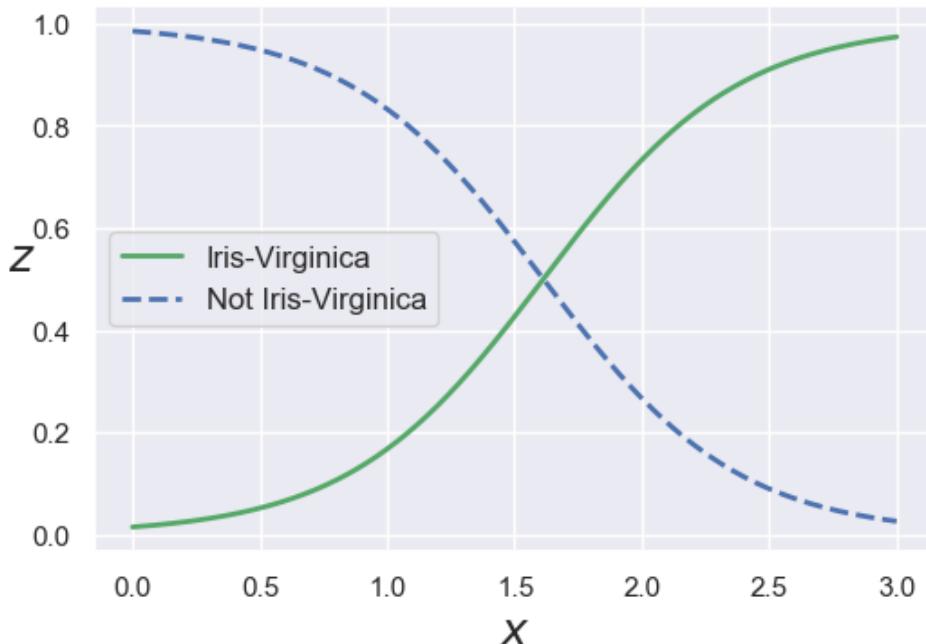
```
[11]: from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(solver="liblinear", random_state=42)
log_reg.fit(X, y)
```

```
[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>
```



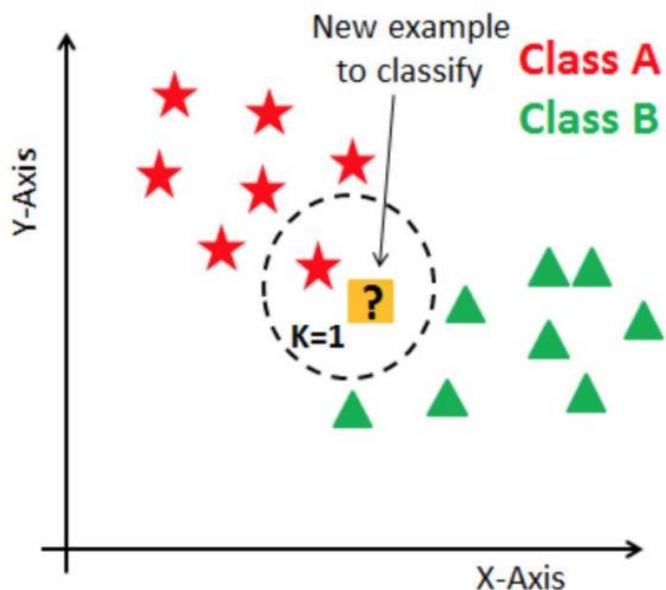
```
[13]: X_new = np.linspace(0, 3, 1000).reshape(-1, 1)
y_proba = log_reg.predict_proba(X_new)
decision_boundary = X_new[y_proba[:, 1] >= 0.5][0]

decision_boundary
```

```
[13]: array([1.61561562])
```

k-Nearest Neighbor

k-Nearest Neighbor

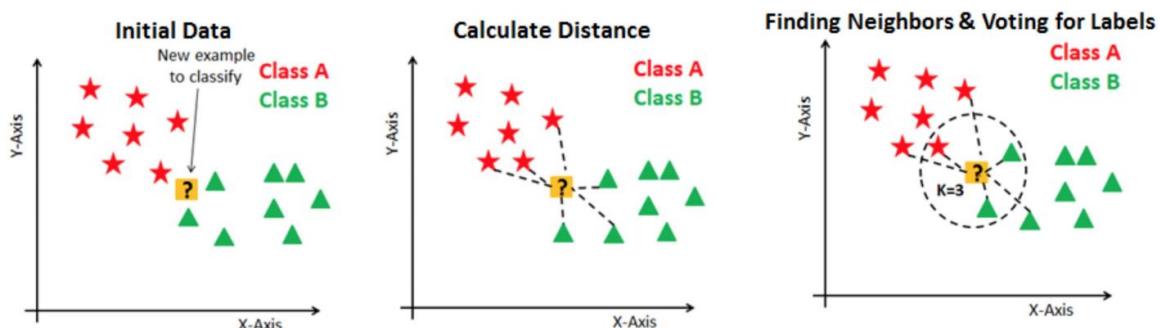


Bir parametre vermeden algoritmamızın classification yapısı ona verdığımız veri setiyle değişiyor.

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

$$A = (x_1, x_2, \dots, x_m) \quad B = (y_1, y_2, \dots, y_m)$$

$$dist(A, B) = \sqrt{\frac{\sum_{i=1}^m (x_i - y_i)^2}{m}}$$

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']

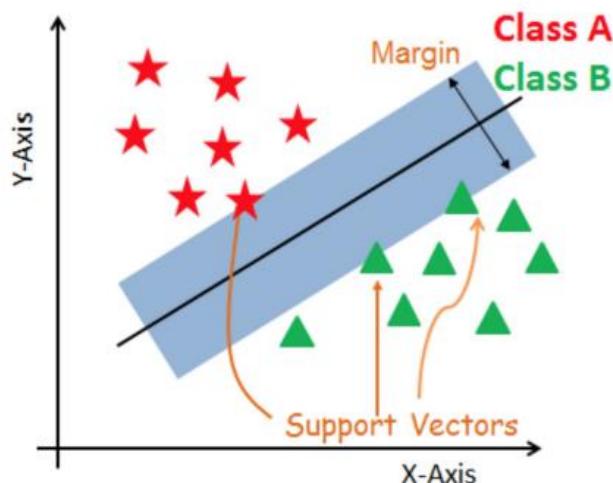
wine_data.head(100)
```

	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

Support Vector Machines

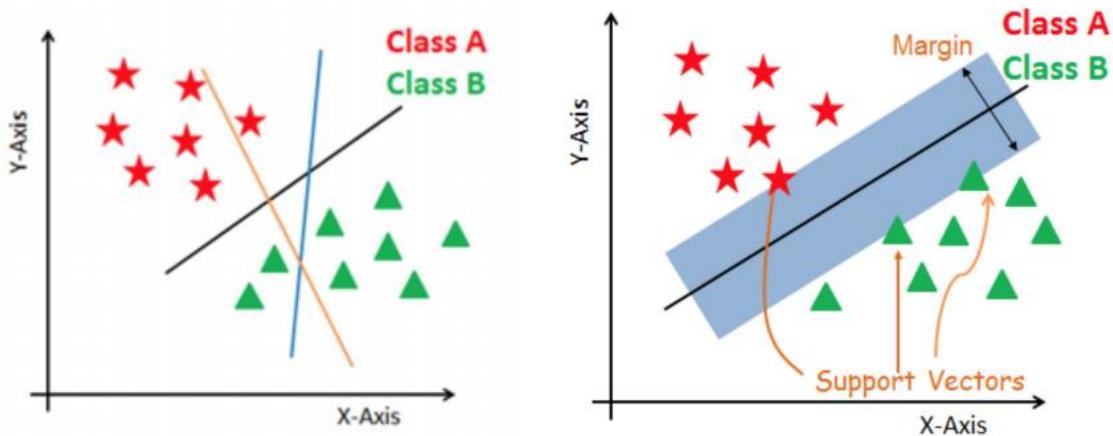
Support Vector Machines



Support Vector Machines genelde diğer algoritmalarla göre classification task'lerimizde accuracy oranı daha yüksek çıkan bir algoritmadır.

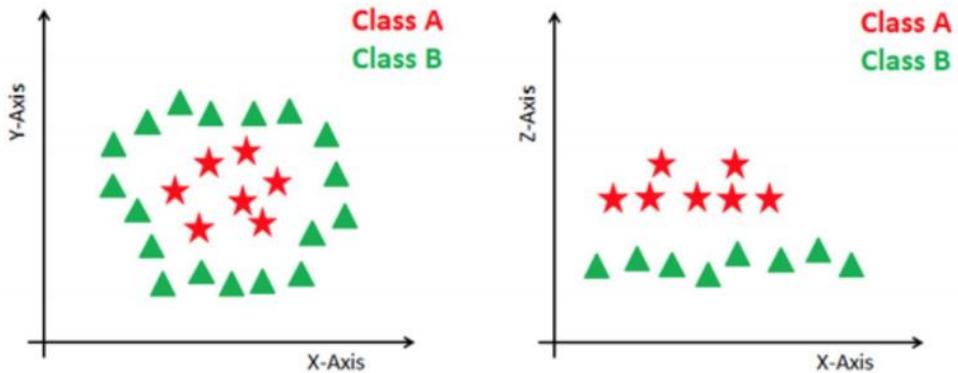
Hem classification hem regression görevlerinde kullanılabilir.

Classification



Yukarıdaki örneklerde lineer bir örnek var.

Support Vector Machines - Kernels



Ancak her zaman lineer olmayıpiliyor.

Support Vector Machines - Kernels

SAMPLES

Name of the Kernel	Mathematical Formula
Linear	$k(x, y) = x^T \cdot y$
Polynomial	$k(x, y) = (x^T, y)^P$ or $k(x, y) = (x^T \cdot y + 1)^P$ 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

```
[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
```

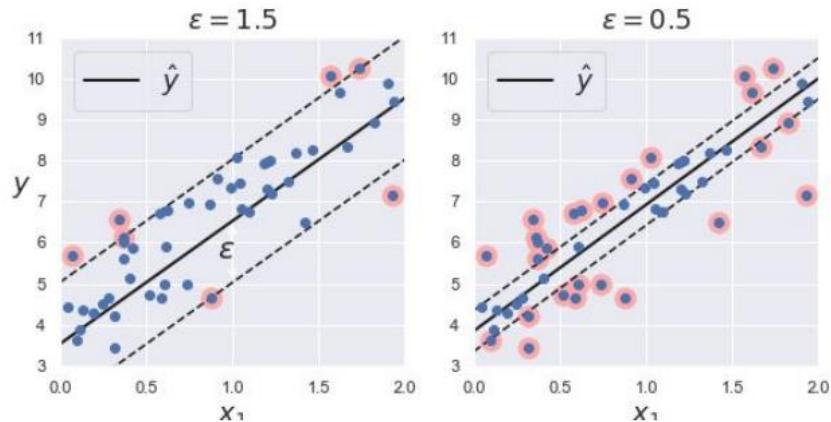
```
[309]: array([2, 1, 0, 2, 0, 2, 0, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 2, 1, 0, 0, 2, 0, 0, 1, 1, 0, 2, 1, 0, 2, 2, 1, 0, 2, 2, 1, 0, 2])
```

```
[310]: accuracy = svm_model_linear.score(X_test, y_test)
accuracy
```

```
[310]: 0.9736842105263158
```

Regression

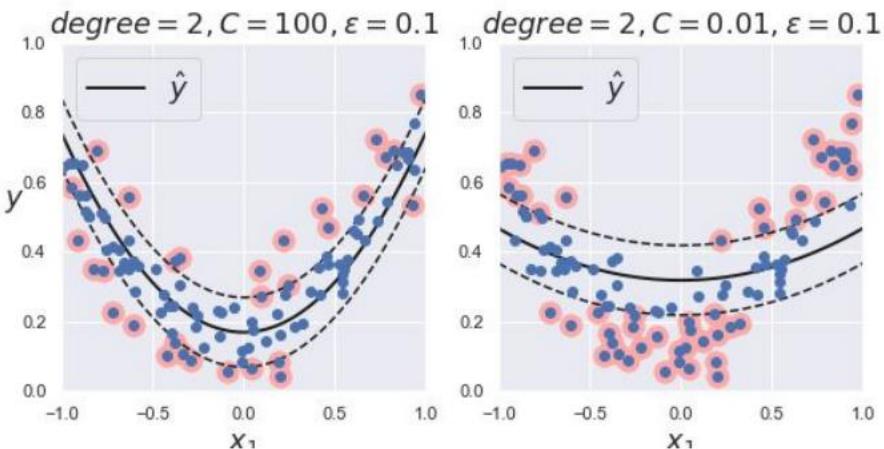
Support Vector Machines - Regression



Epsilon değerimizi kullanarak yakınlaştırmaya çalışıyoruz.

Burada amacımız hyper plane içerisine olabildiğince fazla instance sokabilmek.

Support Vector Machines - Regression



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: DataConversionWarning
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)
```

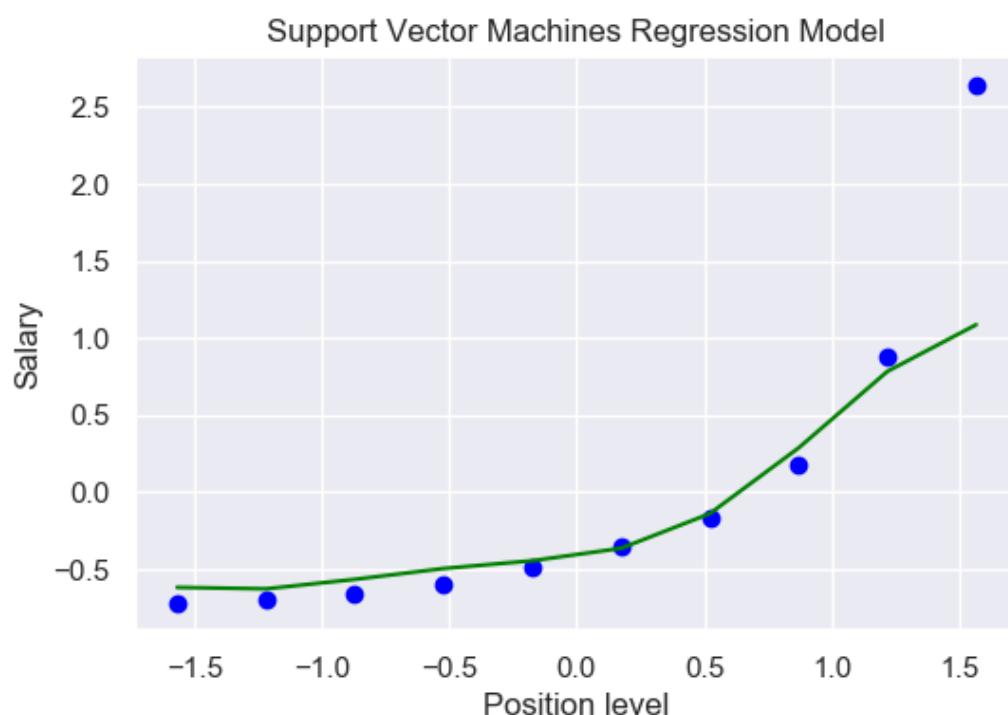
RBF'de Gama parametremiz bize iki nokta arasındaki benzerliğin oranını hesap eder. 0-1 arasındadır.

Daha yüksek Gama değeri verdığımızda 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

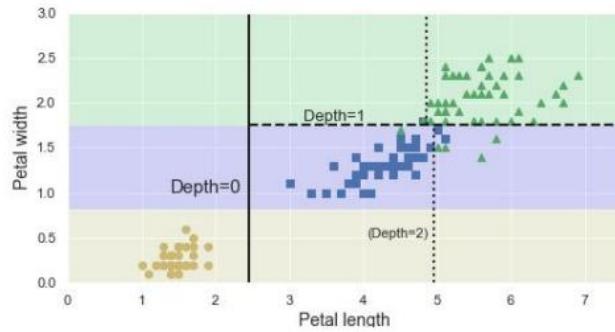
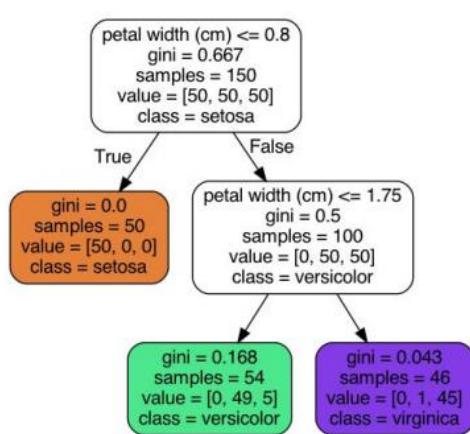
Classification

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.

$$I_G(n) = 1 - \sum_{i=1}^J (p_i)^2$$

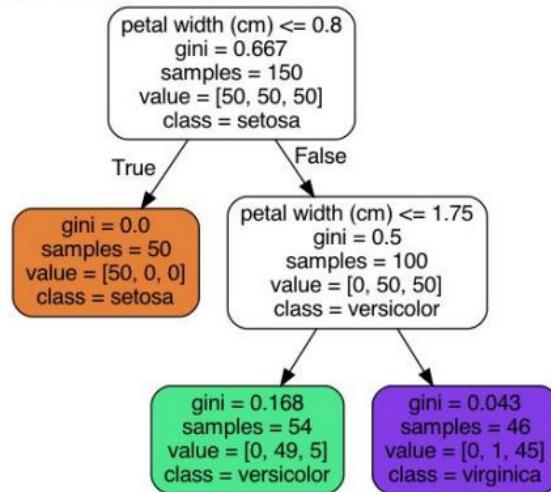
Decision Trees - Classification

Gini Impurity

$$I_G(n) = 1 - \sum_{i=1}^J (p_i)^2$$

Derinlik-2 Sol Nod:

$$1 - (0/54)^2 - (49/54)^2 - (5/54)^2 \approx 0.168$$



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.

$$I_H = - \sum_{j=1}^c p_j \log_2(p_j)$$

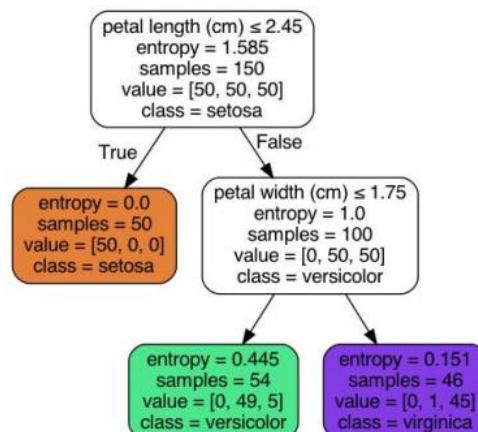
Decision Trees - Classification

Entropy

$$I_H = - \sum_{j=1}^c p_j \log_2(p_j)$$

Derinlik-2 Sol Nod:

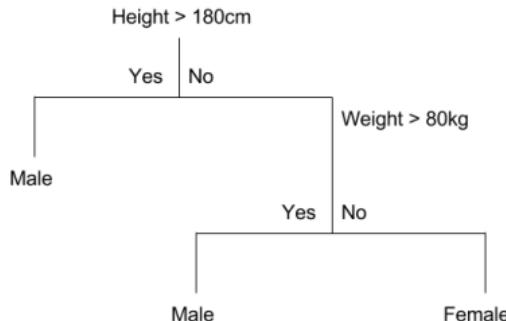
$$-(49/54)\log(49/54) - 5/54\log(5/54) \approx 0.44$$



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

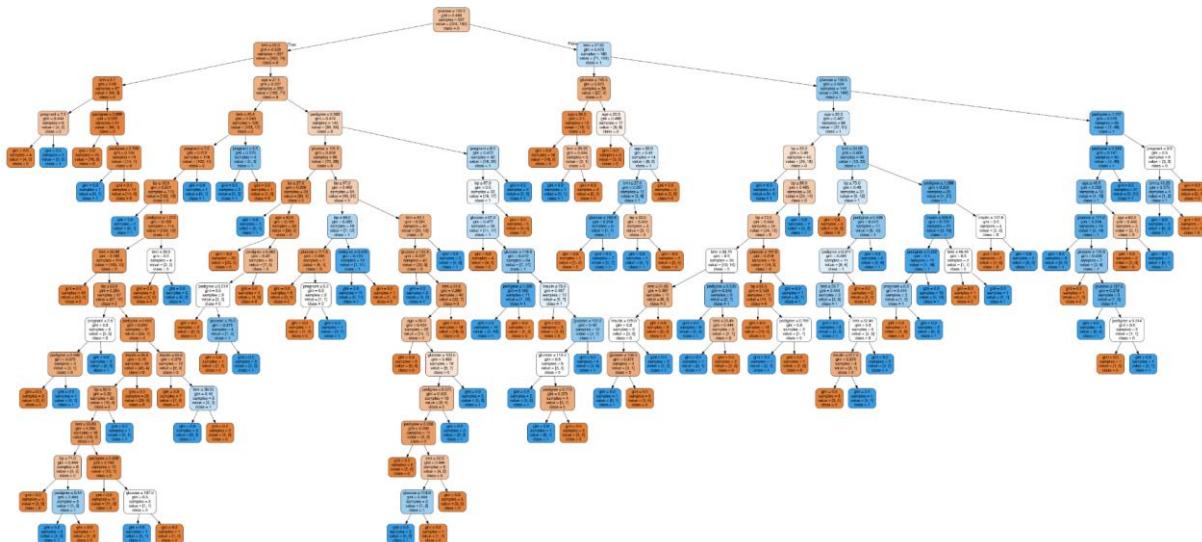
[108]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1) # 70% training and 30% test

[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)

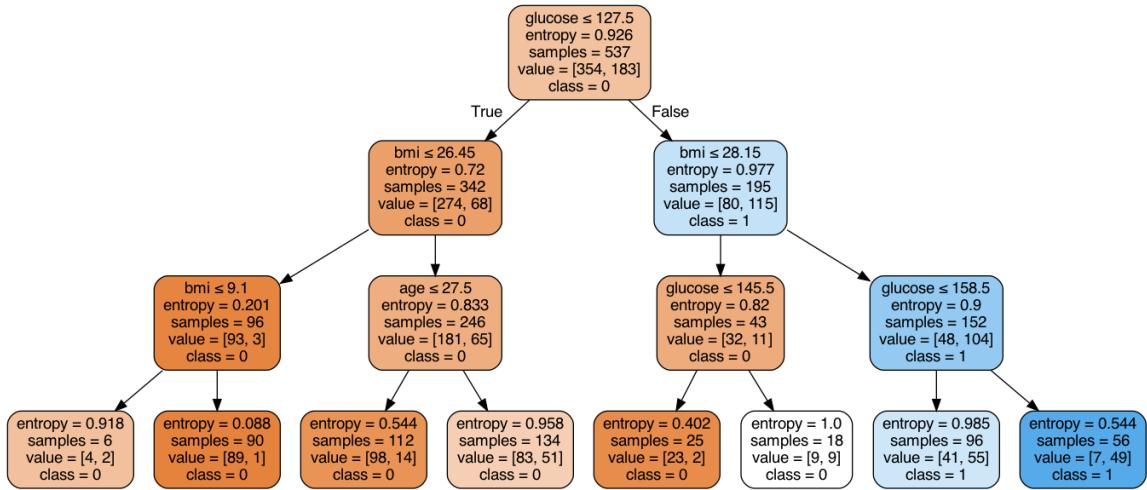
y_pred = clf.predict(X_test)

print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

Accuracy: 0.7705627705627706

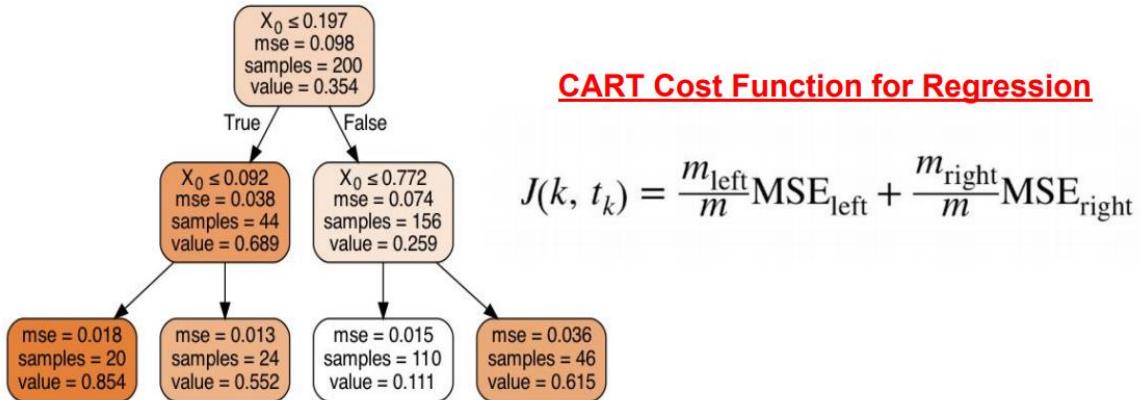
```
[83]: from sklearn.externals.six import StringIO
from IPython.display import Image
from sklearn.tree import export_graphviz
import pydotplus
dot_data = StringIO()
export_graphviz(clf,
                out_file=dot_data,
                filled=True, rounded=True,
                special_characters=True,
                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



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
```

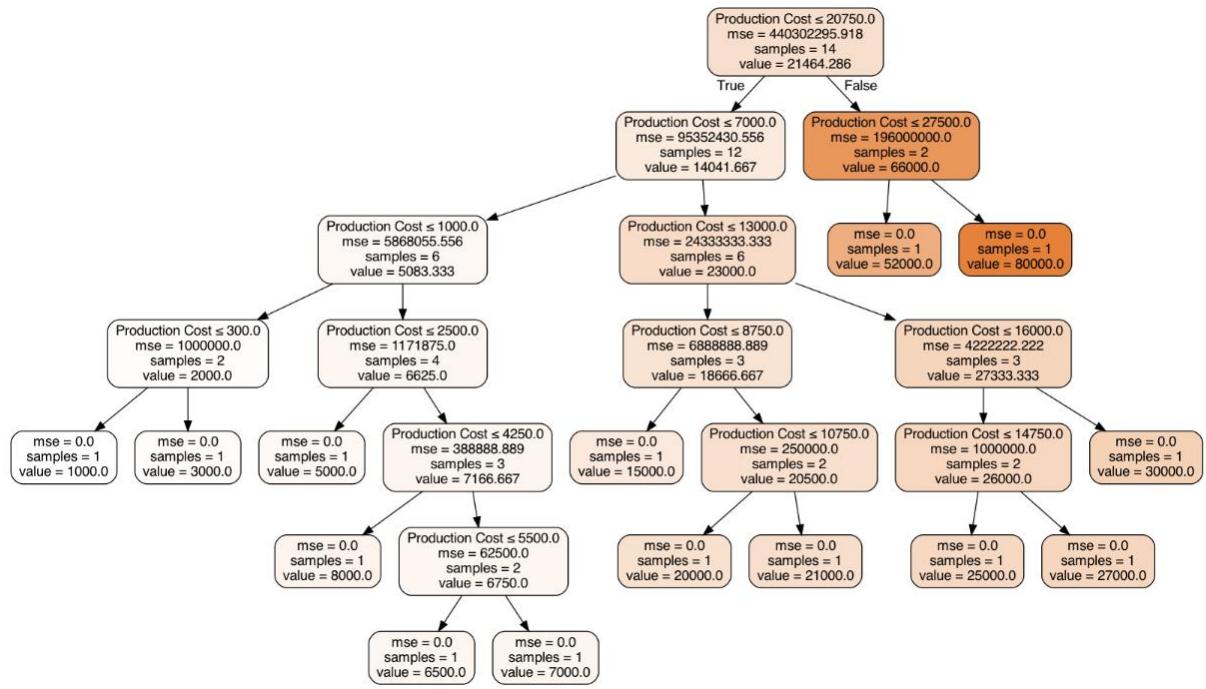
```
[116]: array([ 1000,  3000,  5000,  8000,  6500,  7000, 15000, 20000, 21000,
       25000, 27000, 30000, 52000, 80000])
```

```
[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')
```

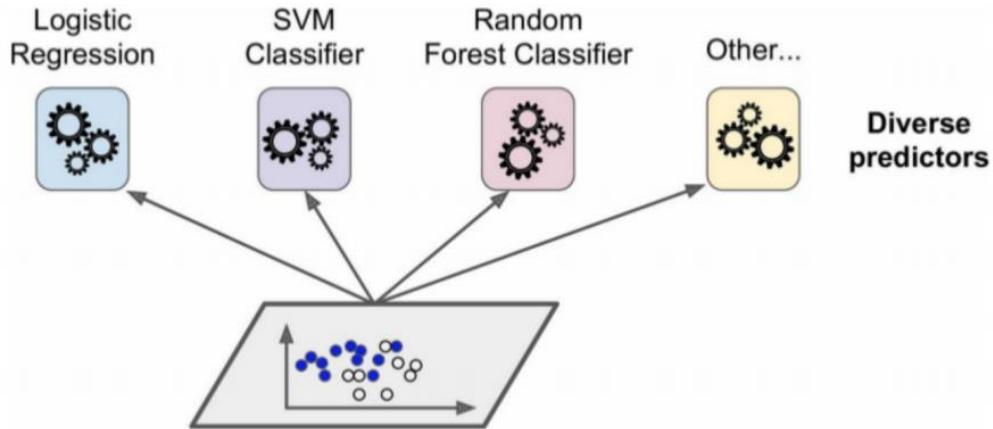
```
[118]: y_pred = regressor.predict([[3750]])
print("Predicted price: % d\n"% y_pred)
Predicted price:  8000
```

```
[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())
```

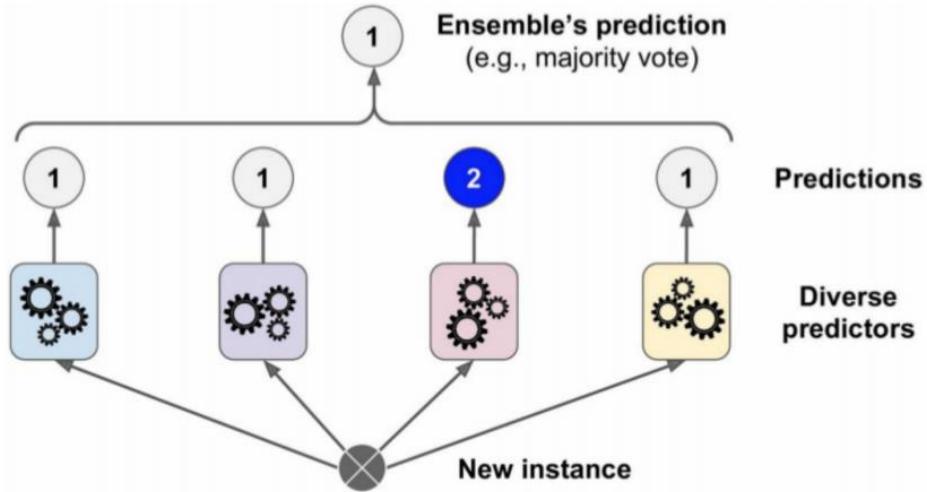


Ensemble Methods & Random Forest

Random Forest



Random Forest



Bagging and Pasting

Bagging : Bootstrap Aggregating

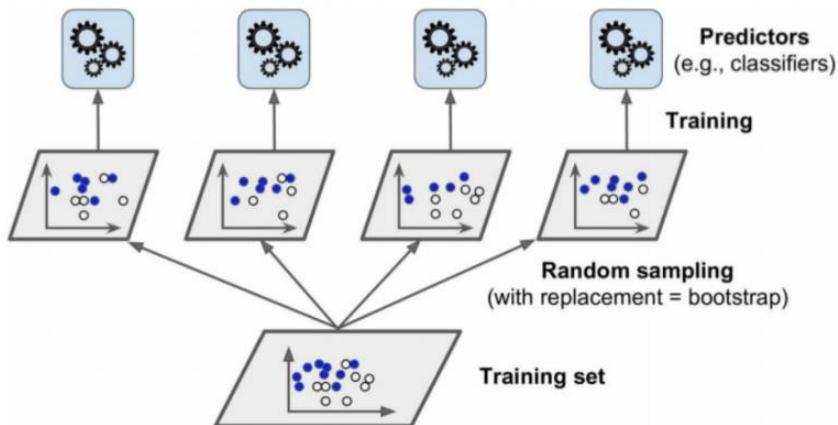
Verisetini sampling yöntemiyle classifier'lara ayırıyor. Sürekli verisetini classifier'larda sampling edip sürekli değiştirir.

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

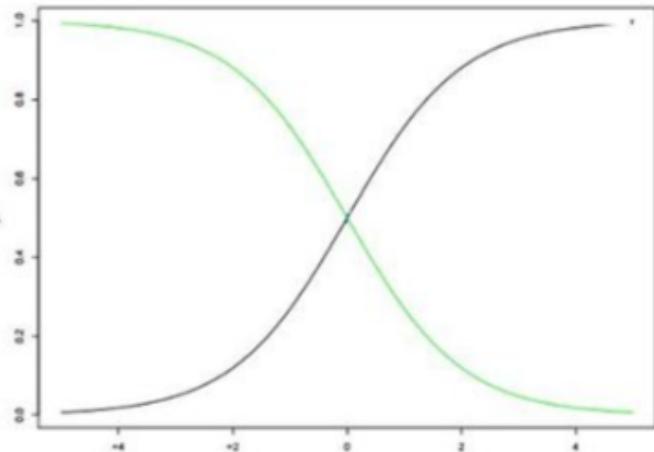
(0,-1.79)

-1.79

61.93



(0,61.93)



- ✓ 2) β_0 ve β_1 olarak iki farklı değer için iki farklı logistic model grafikte gösterilmiştir. β_0 ve β_1 için aşağıdakilerden hangisi doğrudur? (β_0 : yeşil, β_1 : siyah, $Y = \beta_0 + \beta_1 \cdot X$) *
- 10/10

- B) β_1 değeri her iki model için de aynıdır.
- C) Yeşil için olan β_1 değeri siyahından küçüktür. ✓
- D) Hiçbiri doğru değildir.
- A) Yeşil için olan β_1 değeri siyahından büyüktür

X 3) Aşağıdakilerden hangisi k-Nearest Neighbor için doğrudur? *

0/10

- Sadece classification için kullanılır. X
- 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? *

Model hyperplane'e çok uzak noktalardaki değerleri modelleme işlemine dahil edebilir.

Model sadece hyperplane'e çok yakın mesafedeki değerleri modelleme işlemine dahil edebilir. ✓

Model veri noktalarının hyperplane'e olan uzaklıından etkilenmez.

Yukarıdakilerden hiçbiri

✓ 6) Aşağıdakilerden hangisi SVM modelin uygulama alanlarındanandır? * 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

Evaluation, Tuning and Regularization

r² score : Model oluşturup tahmin etmek yerine direkt olarak verilerin ortalamasını yazsaydık, bu sonuçlar bizim kurduğumuz modelden daha iyi mi olur yoksa daha kötü mü olur?

Eğer daha kötüyse 0'ın altında olacak, iyiye 1'e yakın olacak. negatif ise gerçekten kötü bir model oluşturduk demektir.

```
389]: from sklearn.linear_model import LinearRegression  
linreg=LinearRegression(fit_intercept=True)  
linreg.fit(X_train, y_train)  
predict_linear=linreg.predict(X_test)  
  
390]: from sklearn.metrics import r2_score  
print(r2_score(predict_linear,y_test))  
  
-2.1365961225794967
```

Kaynaklar

- Machine Learning Days – Merve Noyan – Data Visualization
<https://youtu.be/JL35pUrth4g>
- 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&
- Machine Learning Days – Merve Noyan – Evaluation, Tuning and Regularization
<https://www.youtube.com/watch?v=rjoyM64pkiE>