**Mushrooms Classification Project**

**Giriş**

Çeşitli türlerin keşfi, özellikle doğanın karmaşık dokusunda yer alan mantarlar gibi, bilim insanlarını ve meraklıları büyüler. Bu veri seti, gilled mantarların karmaşık dünyasına iner, 1981'de Kuzey Amerika Mantarlarına Dair Audubon Derneği Alan Kılavuzu'ndan alınan 23 hayali örneği sunar. Bu örnekler, Agaricus ve Lepiota mantar ailesine ait türleri temsil eder, her biri kendine özgü özelliklere ve gizemlere sahiptir.

Mantarların yenilebilirliğini anlamak karmaşık bir bulmacadır. Zehirli Meşe ve Salkımlı Yapraklı Meyve Ağaçları için "üç yapraklı ise, bırak git" gibi basit bir özdeyiğin aksine, bir mantarın güvenli mi yoksa potansiyel olarak zararlı mı olduğunu belirlemek basit kurallara uymaz. Kılavuz, mantarın yenilebilirliğini ayırt etmek için evrensel bir kılavuzun olmadığını açıkça belirtir, bu da bu araştırmanın karmaşıklığını vurgular.

Proje Açıklaması

Bu proje temelde, sınıflandırma alanında çerçevelenen denetimli bir makine öğrenimi girişimidir. Temel amaç açıktır: belirgin özelliklere dayanarak mantarların yenilebilirliğini tahmin edebilen sağlam bir model geliştirmek. Bu çalışmanın bağlamında, hedef değer mantarları sınıflandırmak etrafında döner, özellikle onları kesinlikle yenilebilir, kesinlikle zehirli veya bilinmeyen yenilebilirlik ve tavsiye edilmeyen olarak tanımlar. Özellikle, sonraki sınıf, mantar toplama ve tüketiminde sıklıkla benimsenen dikkatli yaklaşımla zehirli kategoriyle birleştirilir.

Karşımızdaki karmaşık görev, çeşitli makine öğrenimi tekniklerinin ve algoritmalarının kapsamlı bir keşfini gerektirir. Sınıflandırma algoritmalarının gücünden yararlanarak, mantar veri setinde gizlenmiş karmaşık desenleri ortaya çıkarmak, nihayetinde mantar tüketimi ve güvenliği konusunda bilinçli kararlar için yol açmak amaçlanır.

Proje Amaçları

Mantarların dünyasında bu yolculuğa çıktığımızda, defterin genel hedefi açıkça ortaya çıkar: mantar özelliklerinin labirentinde gezinebilen, yenilebilirliğini kesinlikle ayırt edebilen bir tahmin modeli oluşturmak. Özellikleri titizlikle analiz ederek ve mühendislik yaparak, çeşitli makine öğrenimi algoritmalarını keşfederek ve model parametrelerini ayarlayarak, güvenilir bir mantar yenilebilirliği tahmin etme çerçevesi oluşturmak amaçlanır.

Bu keşif yoluyla, sadece gerçek dünya zorluklarını ele almak için makine öğreniminin potansiyelini kullanmakla kalmayıp aynı zamanda mantar taksonomisi ve güvenliği konusundaki genel anlayışa da katkıda bulunulur. Veri setine daha derinlemesine inerken ve sırlarını çözerken, bireyleri, mantarların büyüleyici ancak esrarengiz dünyasını özgüven ve netlikle gezinmek için gereken bilgi ve araçlarla donatmayı amaçlanır.

**Değişkenler**

cap-shape: bell=b,conical=c,convex=x,flat=f, knobbed=k,sunken=s

cap-surface: fibrous=f,grooves=g,scaly=y,smooth=s

cap-color: brown=n,buff=b,cinnamon=c,gray=g,green=r,pink=p,purple=u,red=e,white=w,yellow=y

bruises: bruises=t,no=f

odor: almond=a,anise=l,creosote=c,fishy=y,foul=f,musty=m,none=n,pungent=p,spicy=s

gill-attachment: attached=a,descending=d,free=f,notched=n

gill-spacing: close=c,crowded=w,distant=d

gill-size: broad=b,narrow=n

gill-color: black=k,brown=n,buff=b,chocolate=h,gray=g, green=r,orange=o,pink=p,purple=u,red=e,white=w,yellow=y

stalk-shape: enlarging=e,tapering=t

stalk-root: bulbous=b,club=c,cup=u,equal=e,rhizomorphs=z,rooted=r,missing=?

stalk-surface-above-ring: fibrous=f,scaly=y,silky=k,smooth=s

stalk-surface-below-ring: fibrous=f,scaly=y,silky=k,smooth=s

stalk-color-above-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y

stalk-color-below-ring: brown=n,buff=b,cinnamon=c,gray=g,orange=o,pink=p,red=e,white=w,yellow=y

veil-type: partial=p,universal=u

veil-color: brown=n,orange=o,white=w,yellow=y

ring-number: none=n,one=o,two=t

ring-type: cobwebby=c,evanescent=e,flaring=f,large=l,none=n,pendant=p,sheathing=s,zone=z

spore-print-color: black=k,brown=n,buff=b,chocolate=h,green=r,orange=o,purple=u,white=w,yellow=y

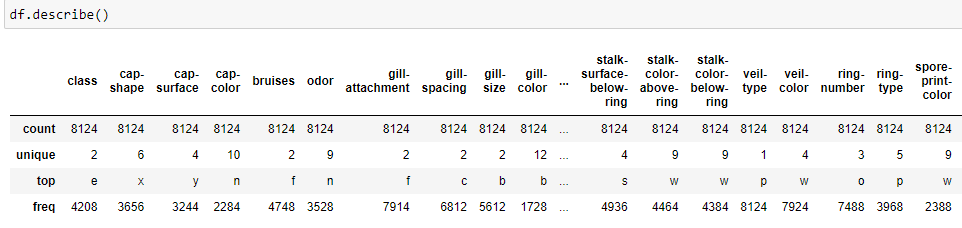
population: abundant=a,clustered=c,numerous=n,scattered=s,several=v,solitary=y

habitat: grasses=g,leaves=l,meadows=m,paths=p,urban=u,waste=w,woods=d

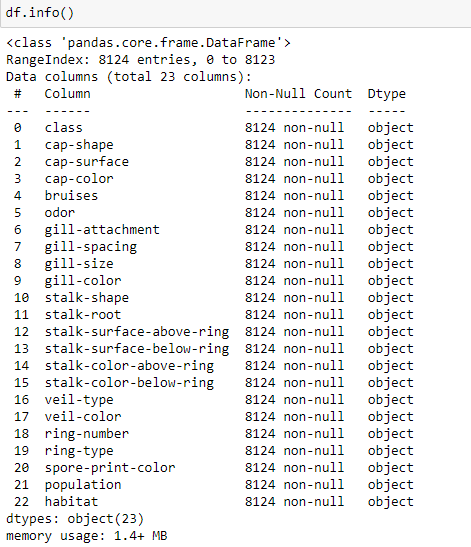
Projemizde bu kategorik değişkenleri kullanıp, mantarlarımızın zehirli olup olmadığının modellemesini yapacağız.



Verimiz 23 değişken, 8124 gözlemden oluşmaktadır.

Verimiz ile alakalı özet istatistikler yukarıdaki görselde verilmiştir.

Gill-attachment, veil-type, veil-color, ring-number gibi değişkenlerdeki değerlerin adetlerinin dengesiz dağıldığı görülmektedir. İlerleyen kısımda görselleştirmeler ile bunu daha iyi bir biçimde göreceğiz.

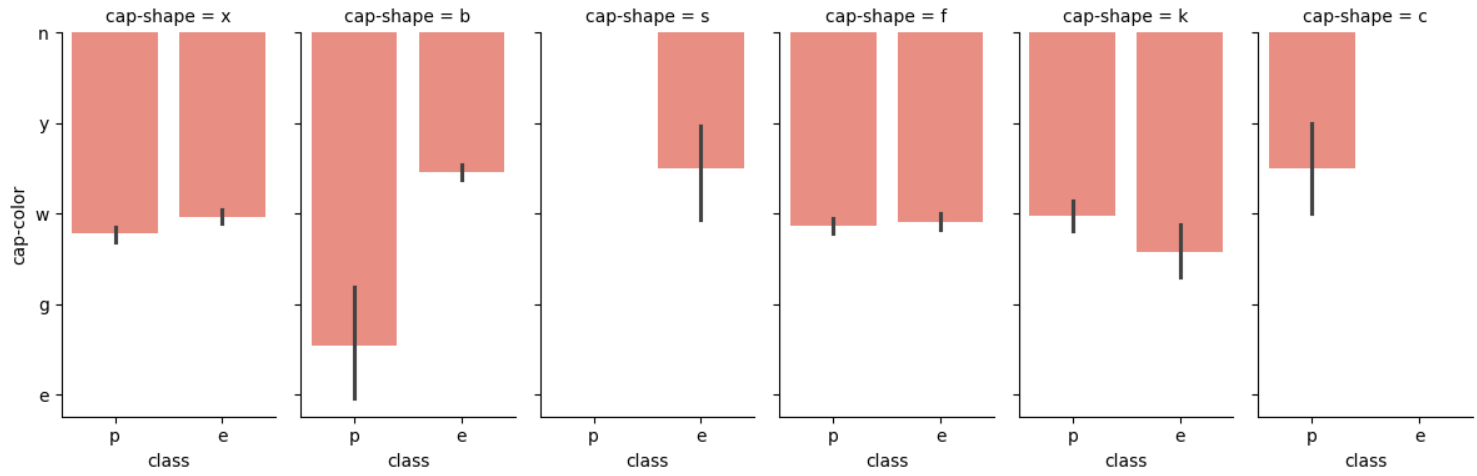


Verimiz tamamen kategorik değişkenlerden oluşup, boş gözlem bulundurmamaktadır.

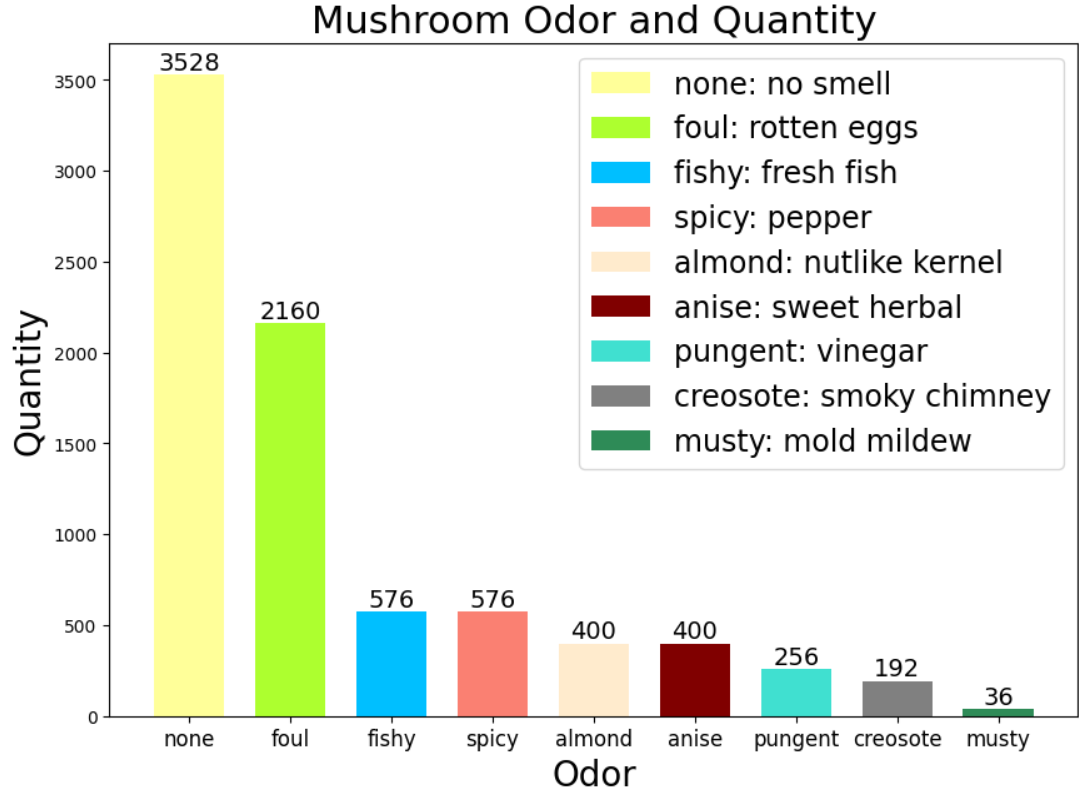


Ancak stalk-root değişkeninde ? şeklinde boş gözlemimiz bulunmaktadır. Bu değerlerin doldurulmasında random forest algoritması temelli imputer olan MissForest ile dolduracağız.

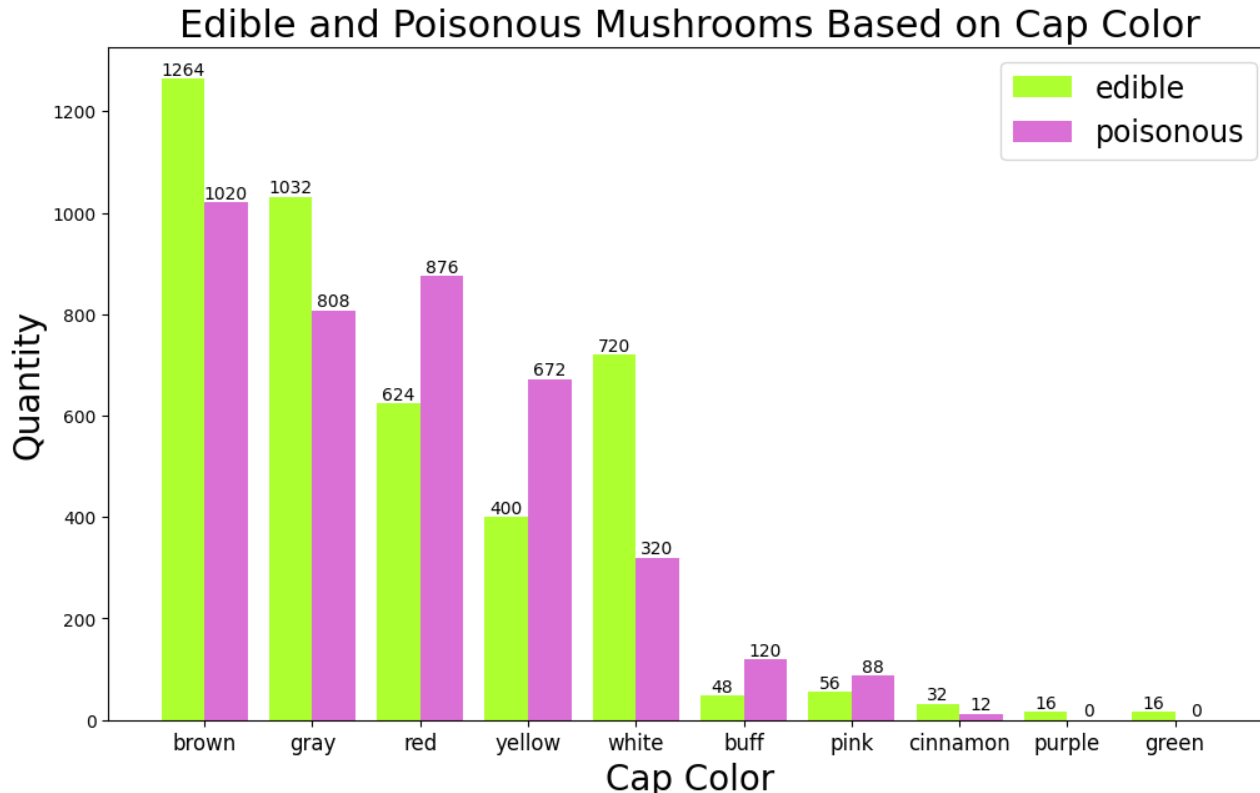
(görsel gelecek)



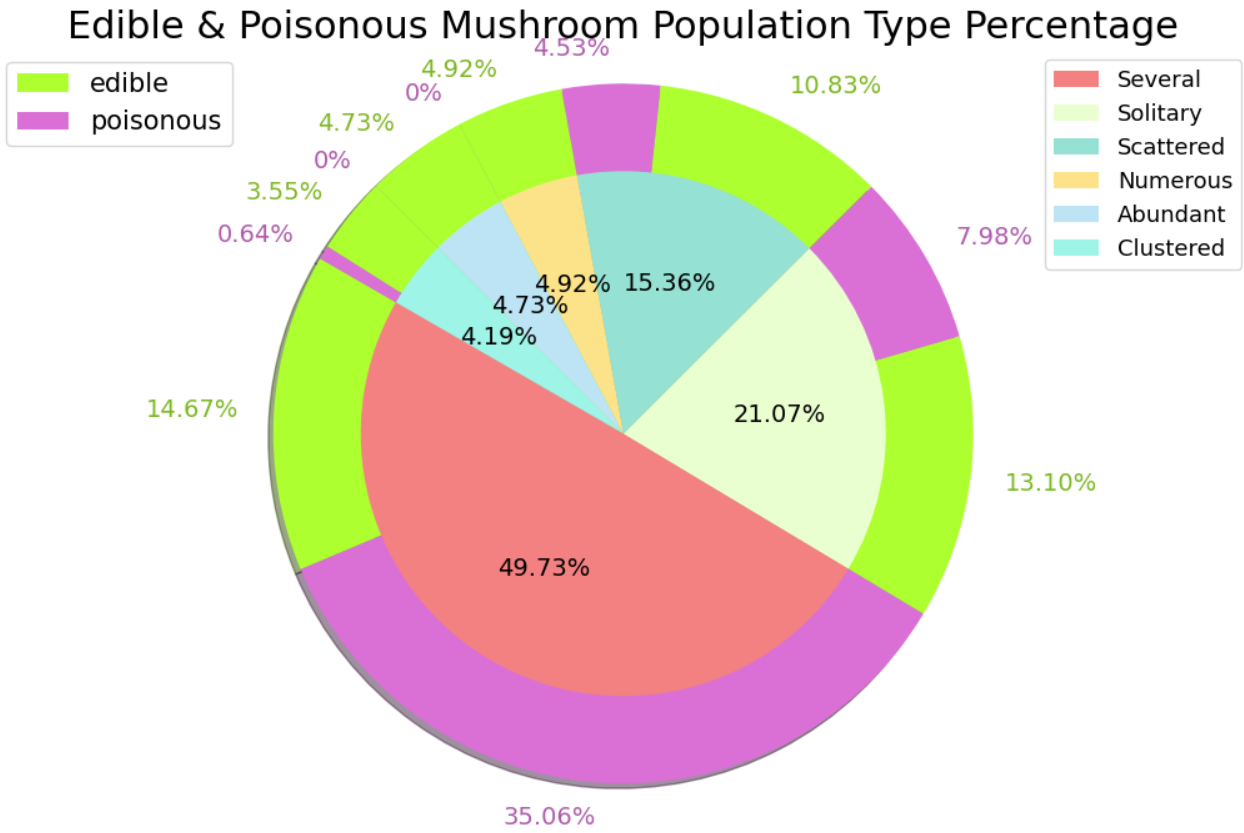
Kahve(n), sarı(y) ve beyaz(w) renklerdeki bombeli(x), toplu(k) ve düz(f) mantarlar genellikle görsel olarak benzerlik gösterir ve ayırt etmeleri zor olabilir. Ancak, çan(b) şeklindeki mantarlarda gri(g) ve kırmızı(e) renklerde olması, zehirli olma ihtimalini artırabilir. Öte yandan, koni(c) şeklindeki ve yine kahve, sarı, beyaz renklerde olan mantarlar genellikle zehirli olabilirler. Çukurlu(s) kahve, sarı ve beyaz renklerdeki mantarlar ise genellikle yenilebilir olarak kabul edilir. Bu nedenle, mantarları tanıma sürecinde renk ve şekil gibi görsel özelliklere dikkat etmek önemlidir.



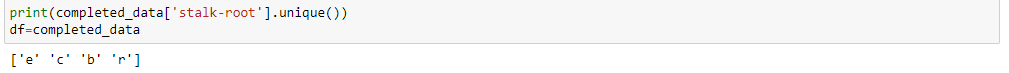
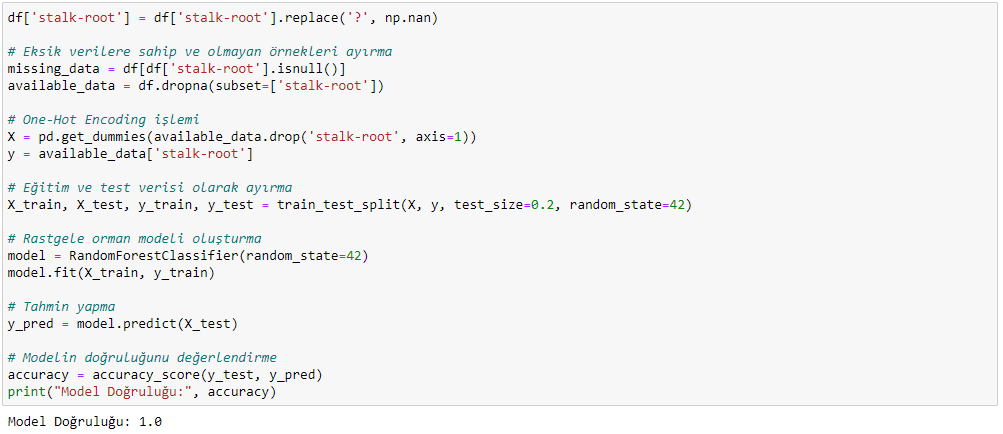
Mantarların çoğunluğu kokusuz(none) ve kötü(foul) kokuya sahiptir.



Mantarların şapka renklerinin yenilebilir mi zehirli mi dağılımına baktığımızda. Kahve, gri, beyaz renkteki mantarların yenilebilir olma ihtimali daha yüksektir. Diğer yandan sarı, kırmızı mantarların ise zehirli olma ihtimali yüksektir yorumunda bulunabiliriz.

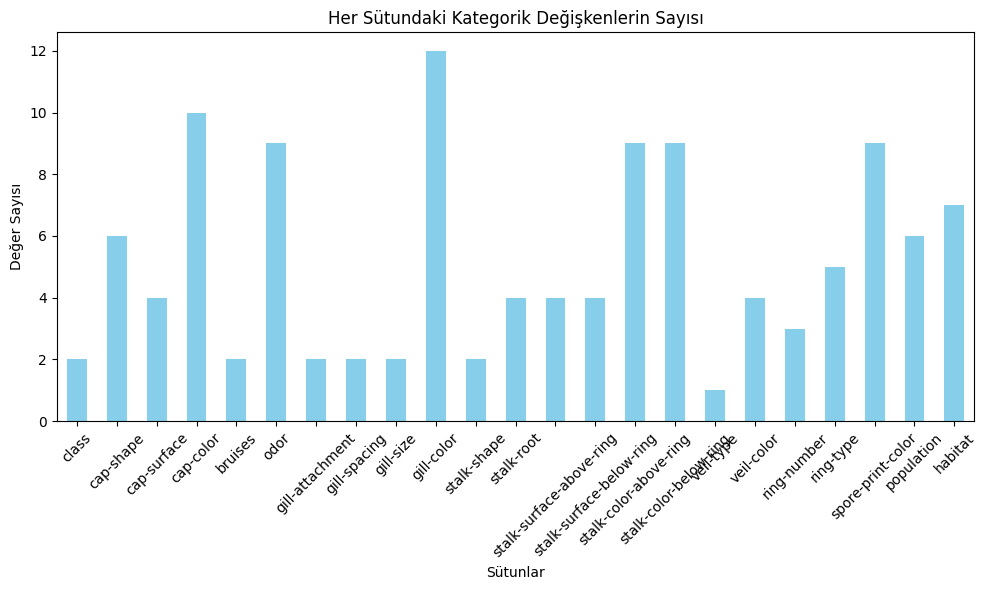


Mantarların neredeyse yarısı 'several' popülasyonuna aittir. Bu popülasyonun %35'i zehirli mantarlardan oluşmaktadır. Bu durum, zehirli mantarların yaygın olduğunu ve toplam popülasyon içinde önemli bir risk oluşturduğunu göstermektedir.

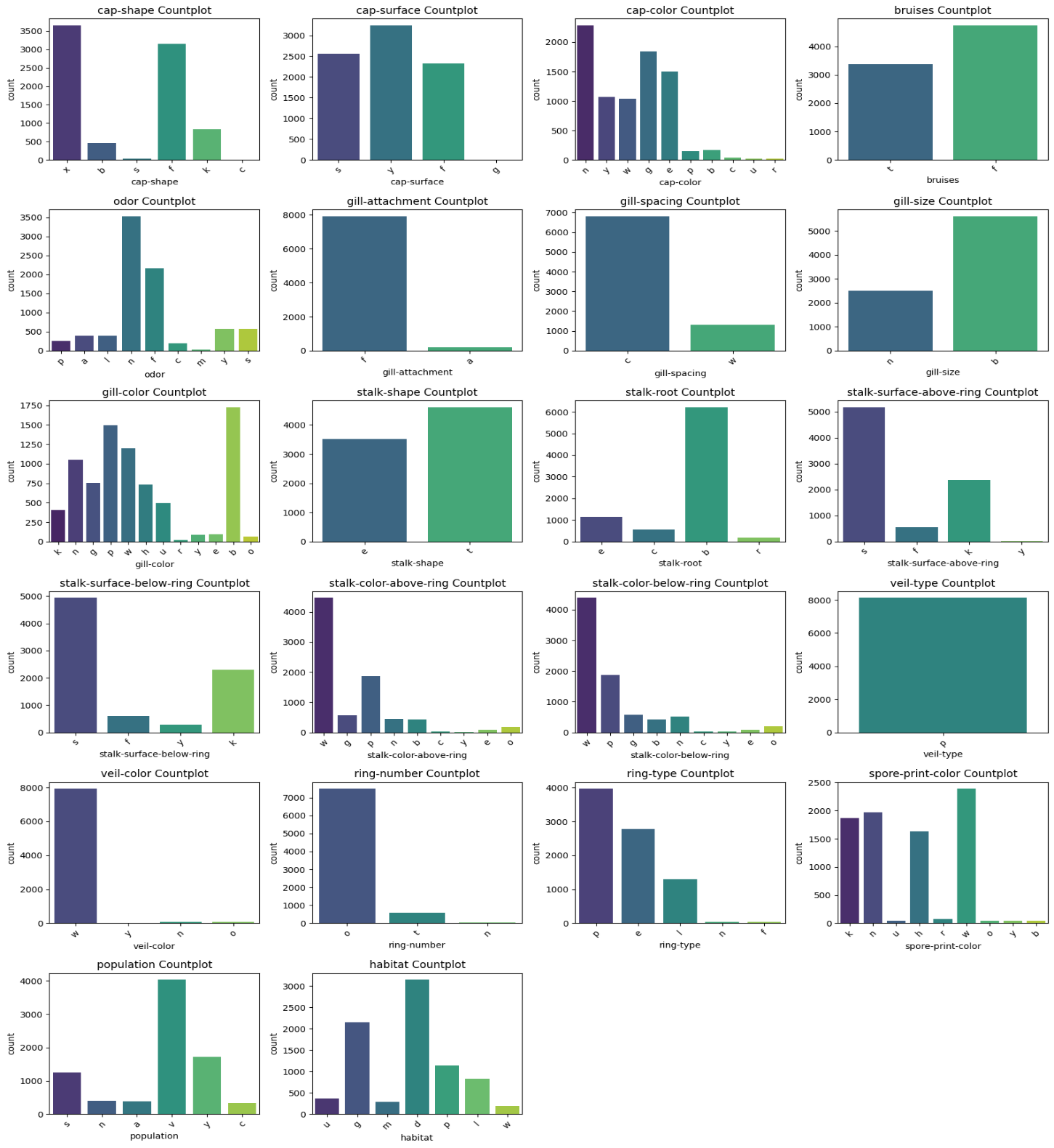


Görüldüğü üzere artık ? işaretli değerlerimiz, randomforrest algoritmasıyla dolduruldu . (Kaggle vs diğer platformlarda bu değerlerin random forest ile doldurulduğu görülmemiştir.)

Şimdi verilerimizi görselleştirip çıkarımlarımızı yapacağız.



Değişkenlerin kaç etiket içerdiğini gösteren grafiğimiz yukarıdaki gibidir.

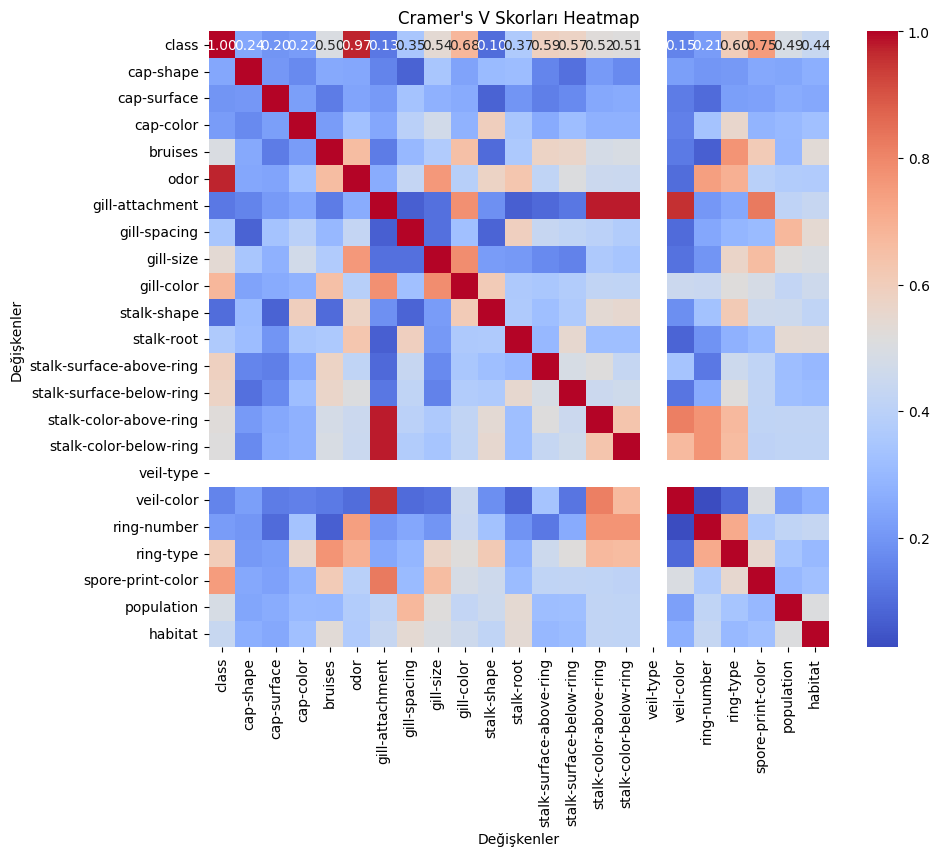


Yukarıda bahsettiğimiz gibi bazı değişkenler homojen dağılmaktadır. Bunların verimizde yaratacağı etkileri modellemelerde ve bazı grafiklerde göreceğiz.

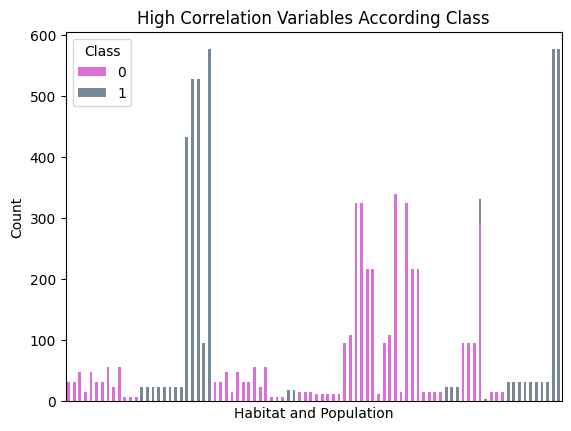


Bu grafiğimizde gördüğümüz gibi bazı değişkenlerin değerleri neredeyse ya zehirli, ya yenilebilir şekilde dağılmakta. Örneğin odor’u f olan mantarların sadece zehirli n olanların neredeyse yenilebilir olduğu gibi yoğunlaşmalar bulunmaktadır.

Şimdi kategorik değişkenler için kullanılan korelasyon analizi olan Cramer’s V metodu ile ısı grafiğini inceleyelim.



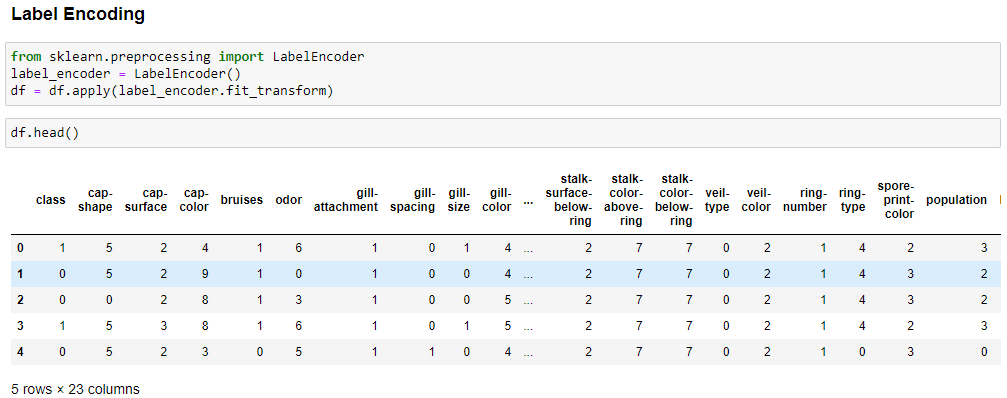
Bağımlı değişkenimiz olan class ile kuvvetli korelasyonları olan değişkenlere baktığımızda, odor 0.97 spore-print-color 0.75 , gill-color 0.68 ile dikkat çekmektedir. Şimdi verimizi bu değişkenlere göre gruplayıp, zehirli zehirsiz olma dağılımlarına bakalım. (Bu analiz ilgili platformlardan alınmamıştır.)



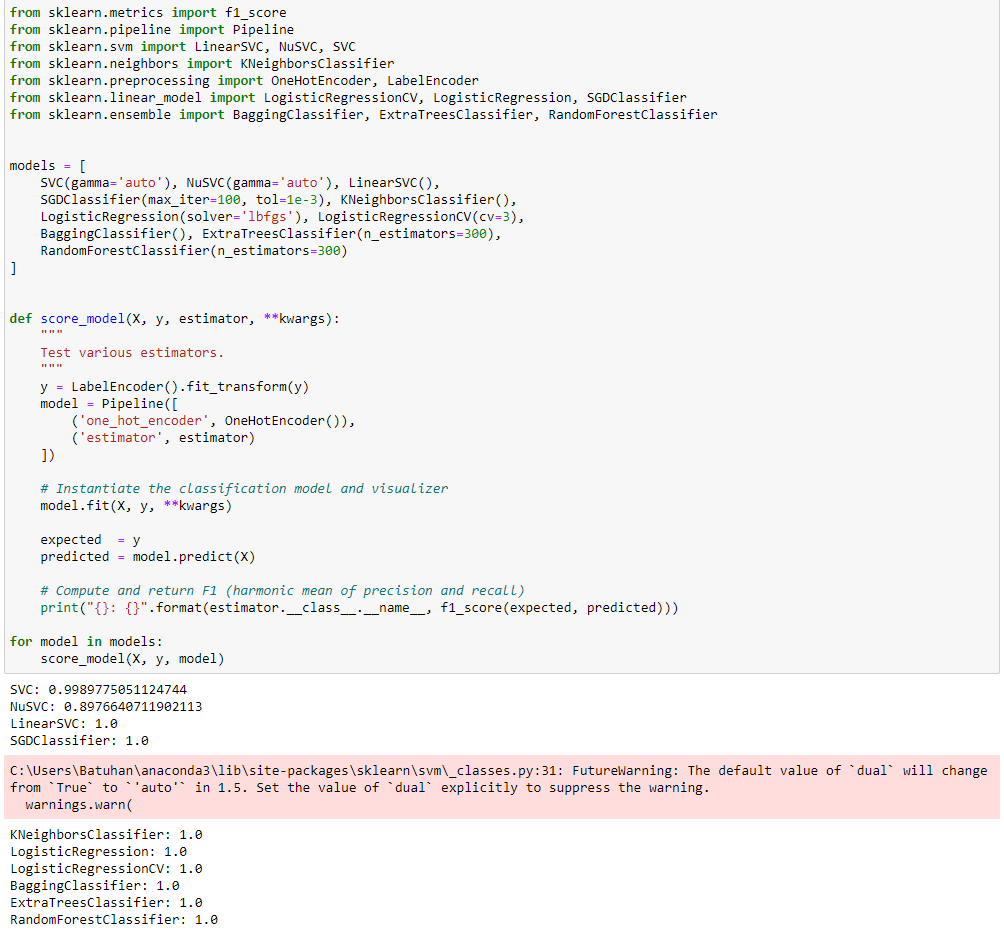
Sadece bu değişkenlere gruplayıp baktığımız zaman neredeyse %100 e yakın mantarların zehirli olup olmadığını görmek mümkün, verimizde bunun gibi bir sürü örüntü bulunup, mantarlarımızın class’ını tahmin ettiğimiz modellemelerin işini oldukça kolaylaştırdığından %100 e yakın doğruluk oranları göreceğiz.

Modelleme

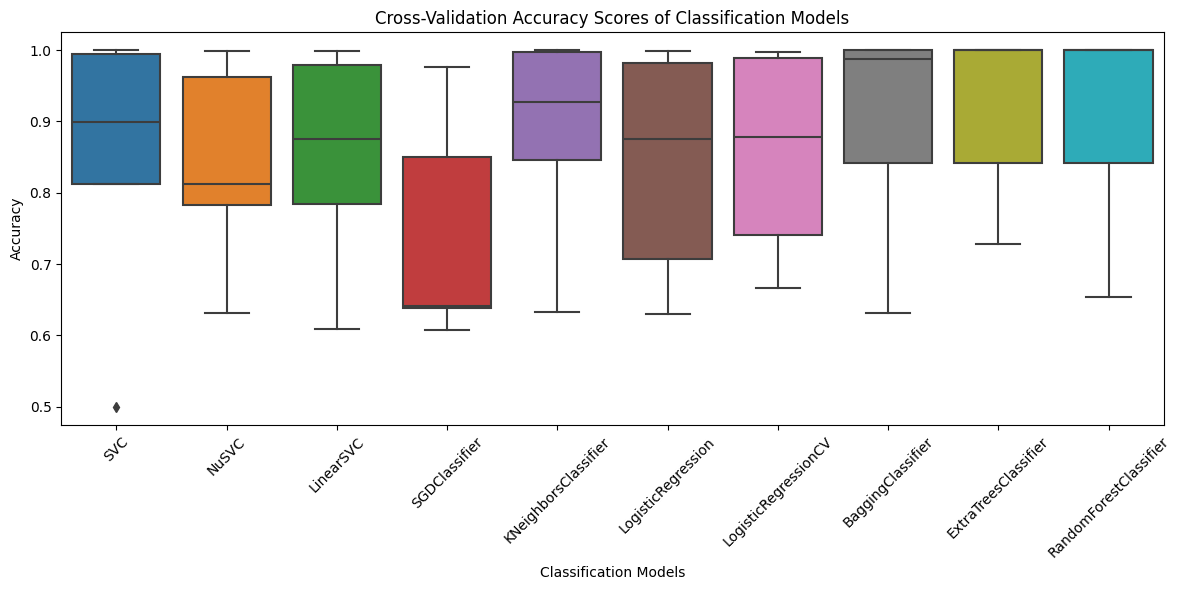
Önce modellemeye hazır hale getirmek için etiketleyelim.

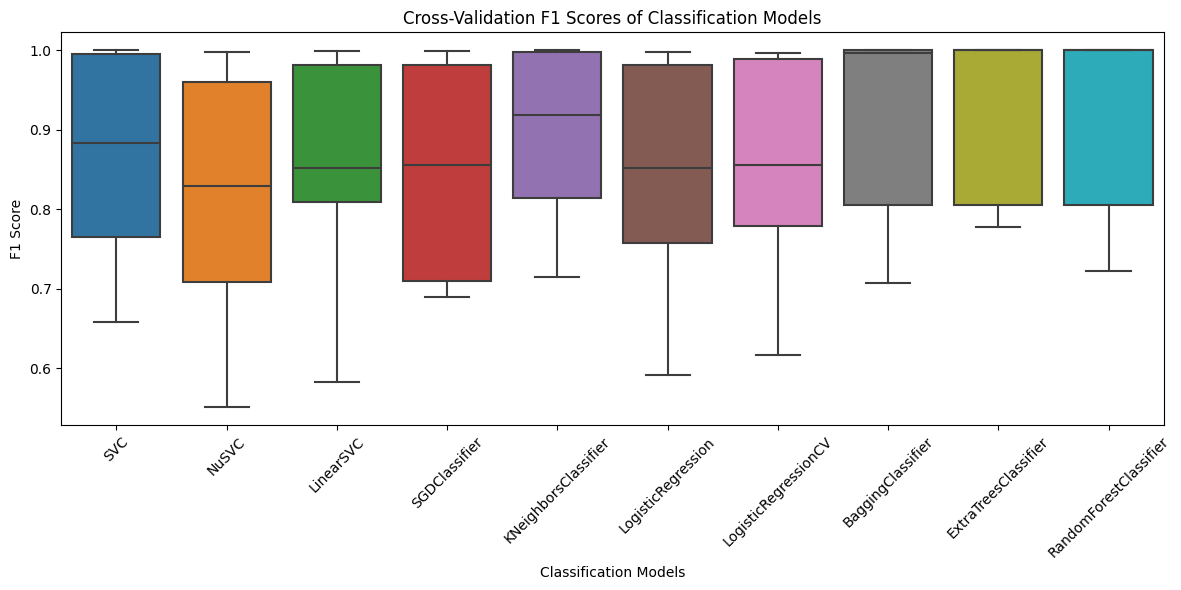


Şimdi sınıflandırma modellememizi seçmek adına aday classifier ve ensemble algoritmalarımızı deneyip doğruluklarına bakalım.



Yüksek gelen doğruluk oranlarının ardından cross validation ile doğruluk oranlarına bakıp daha objektif bir karşılaştırma yapalım.

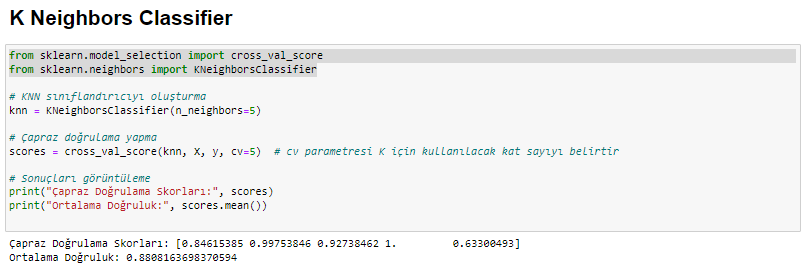




Modellerin f1 skorlarına baktığımızda K Neighbors, Linear SVC, Logistic Regression öne çıkmaktadır.

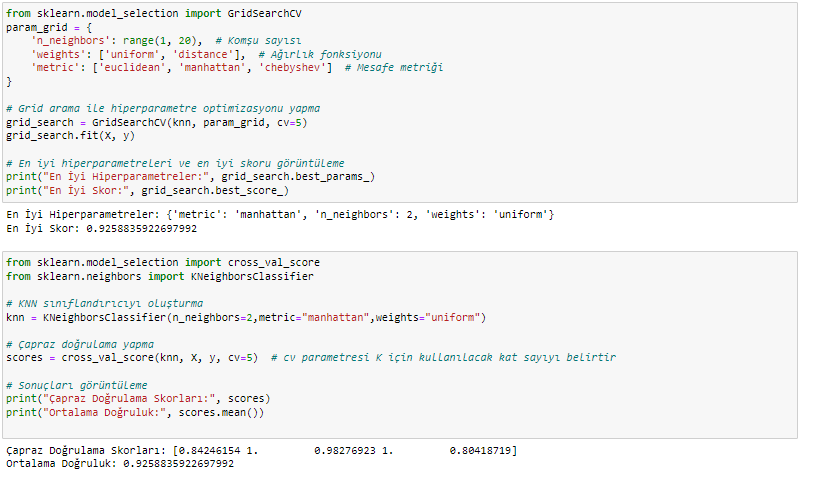
Biz projemizde K Neighbors, Logistic Regresyon ve ensemble metod olarak Random Forest, ek olarak Denetimsiz öğrenme metodu olan KMeans kümeleme algoritmasıyla boyut indirgeme tahmin ve verimizin karşılaştırmalı görselleştirmesini yapacağız.

K Neighbors



Verimizi cross validation ile modellediğimiz zaman doğruluk oranları ve ortalamalarını veren kod ile çıktısı yukardadır.Burada %63 lik kısım ortalamayı oldukça düşürmektedir. Model verinin o kısmında yeterince iyi çalışmamaktadır.

Hiperparametre optimizasyonu

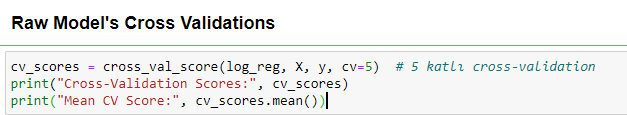


K Neighbors’un parametrelerini deneyip optimal parametreleri bulduk, uzaklık metriği olarak manhattan uzaklığı, komşu sayısı olarak 2 verimizin ağırlığı da uniform olarak belirlenmiştir.

Çapraz doğrulama ile arttığı %4 arttığı, optimizasyonun ne kadar önemli olduğu anlaşılmıştır.

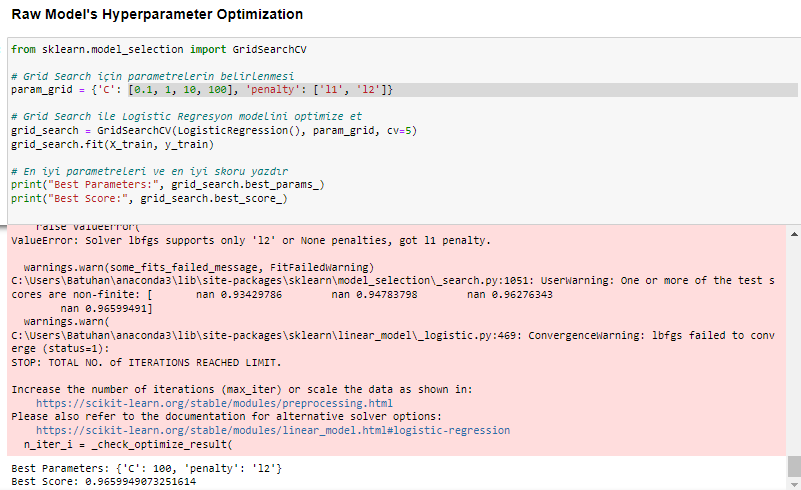
Lojistik Regresyon 

İlk modellememiz sonucu %96,49 doğruluk oranı mevcuttur.



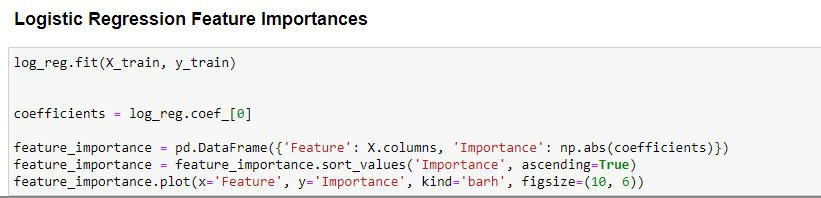
Cross validation yaptığımızda ortalama 84’e yakın gelmekte.

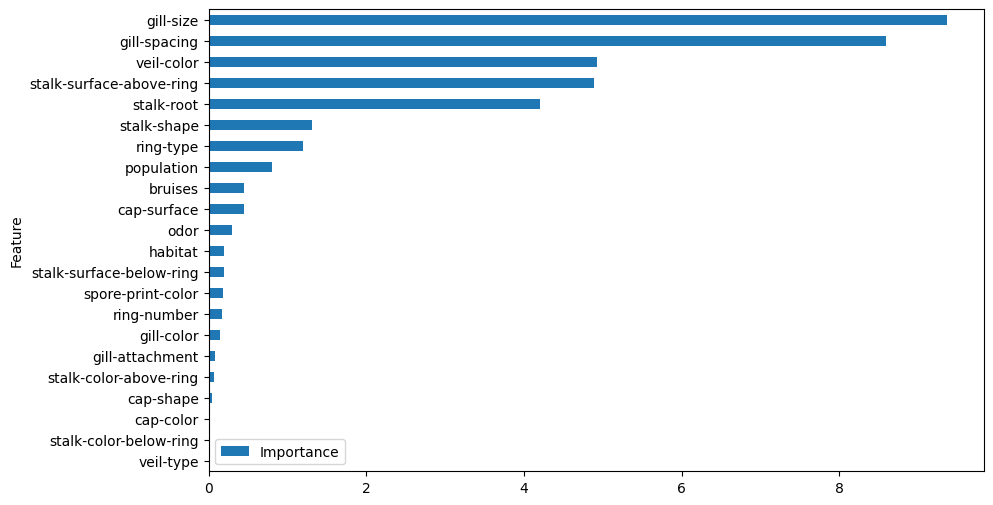
Hiperparametre optimizasyonu



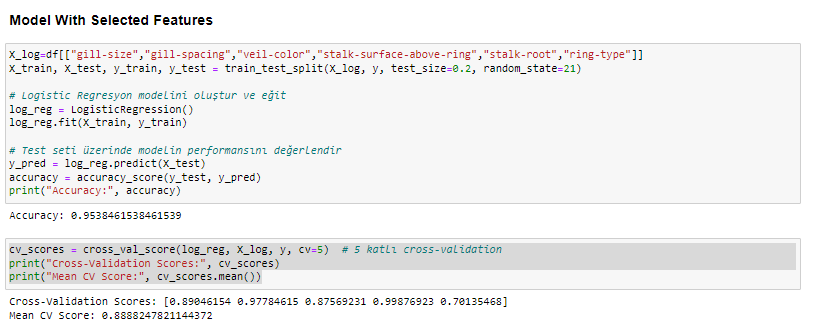
Parametreler denendiği zaman , doğruluğun %97 ye yaklaştığı görülmektedir.

Lojistik Regresyon ile değişken seçimi

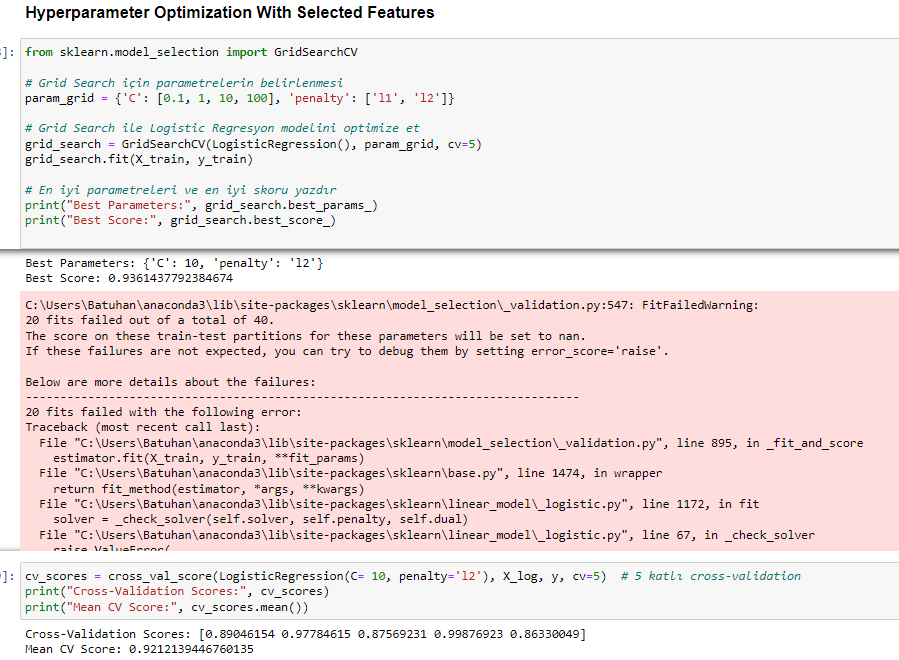




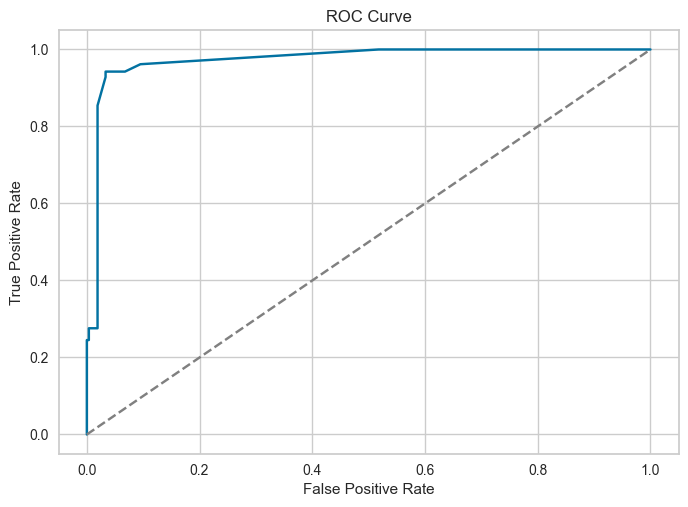
Önem sırasına göre aldığımızda, populationa kadar olan değişkenler seçilmiştir.



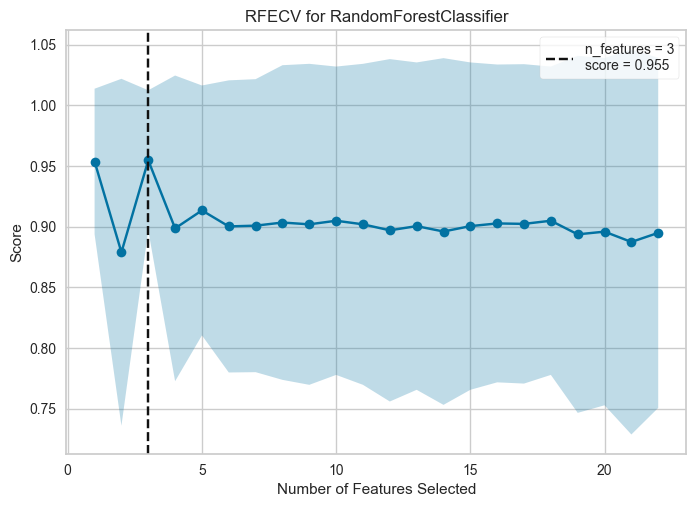
Seçilen değişkenlerle beraber modelleme doğruluğu ve çapraz doğrulama ile skorlara bakıldığında ortalama %88 doğruluğa ulaşılmıştır.

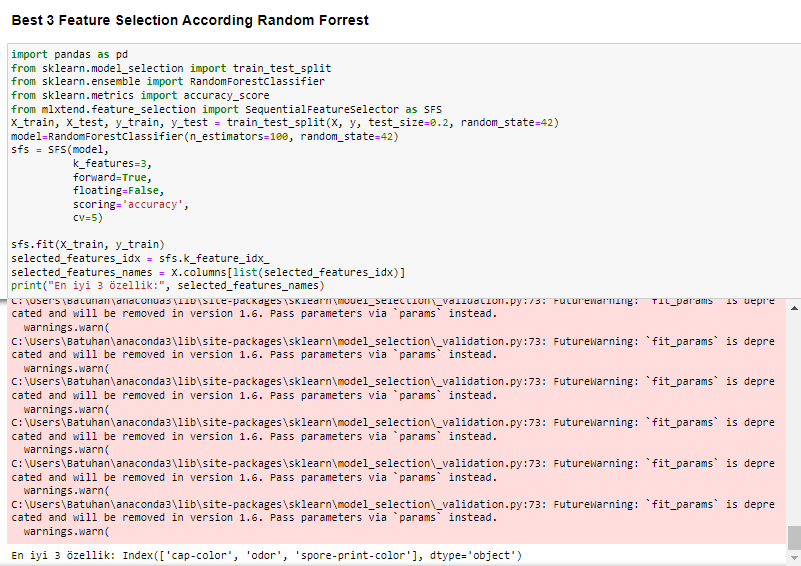


Hiper parametre optimizasyonunun ardından, gerekli parametre ve değişkenler ile ortalama cv doğruluğunun %84 den %92 ye çıktığı görülmüştür.



Roc curvemize baktığımızda alanımız 1 oldukça yakın gelmektedir. Roc skorumuz 0.973 dır.

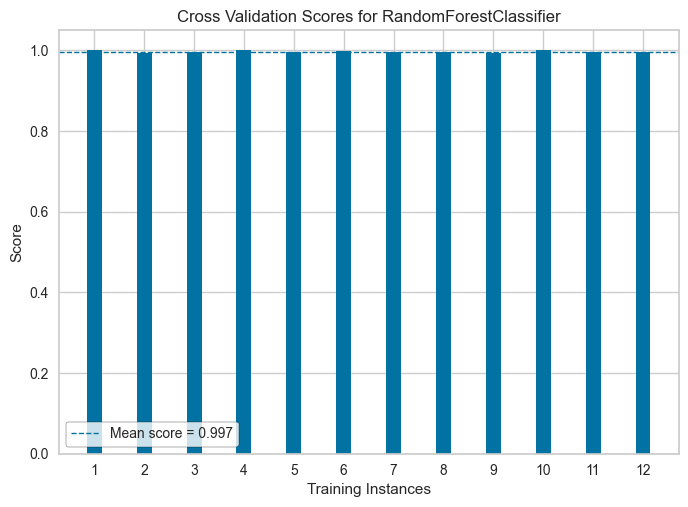
ENSEMBLE METHOD (RANDOM FOREST)

Verimizde önce seçeceğimiz kaç tane değişken seçmemiz gerektiğini Recursive feature elimination metoduyla bulduk (İnternette ilgili kaynaklarda görülmemiştir.)

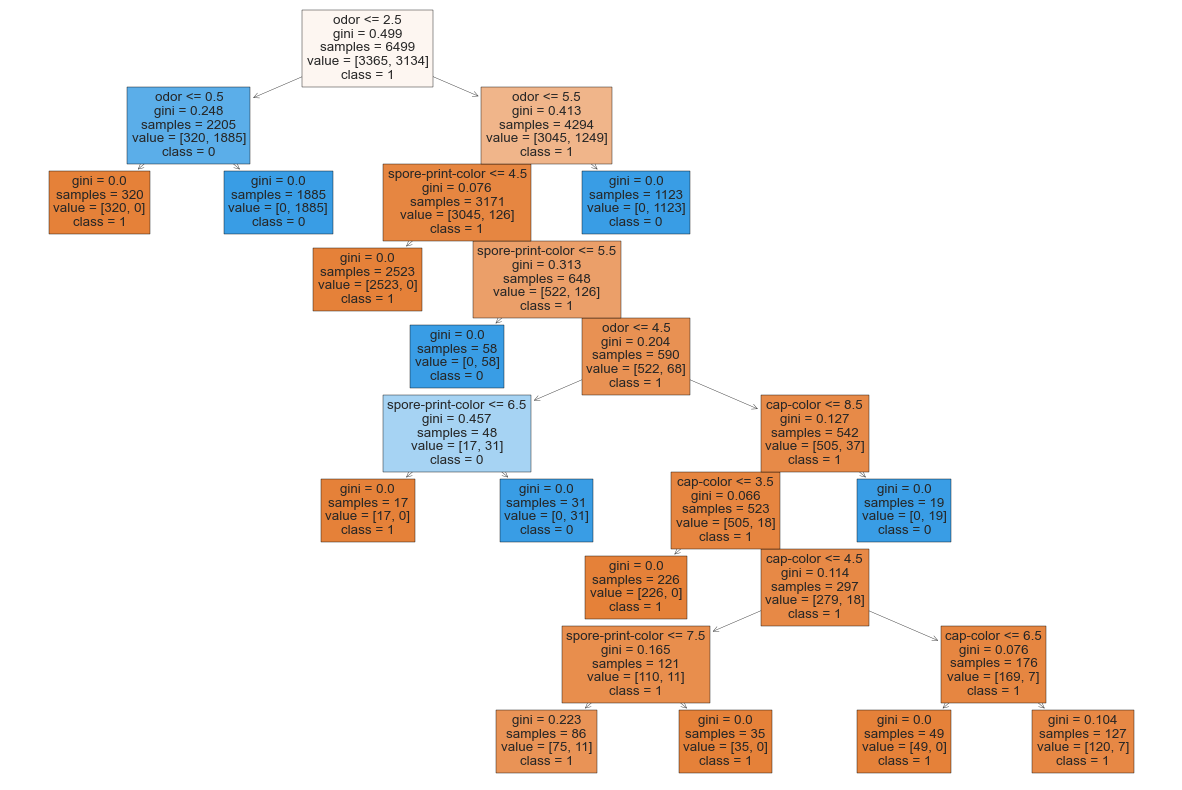
Sequential Feature Selector ile de en iyi 3 değişkeni seçiyoruz.

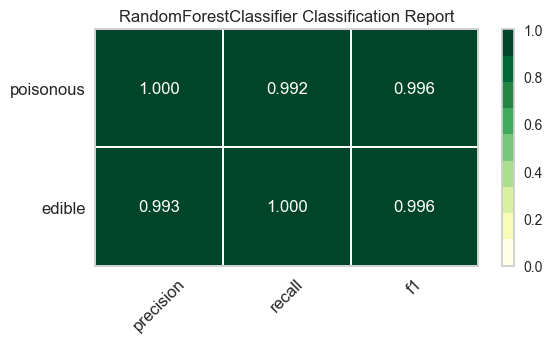


Ardından cross validation optimal parametrele ile neredeyse %100 e yakın doğruluk oranları elde ediliyor.

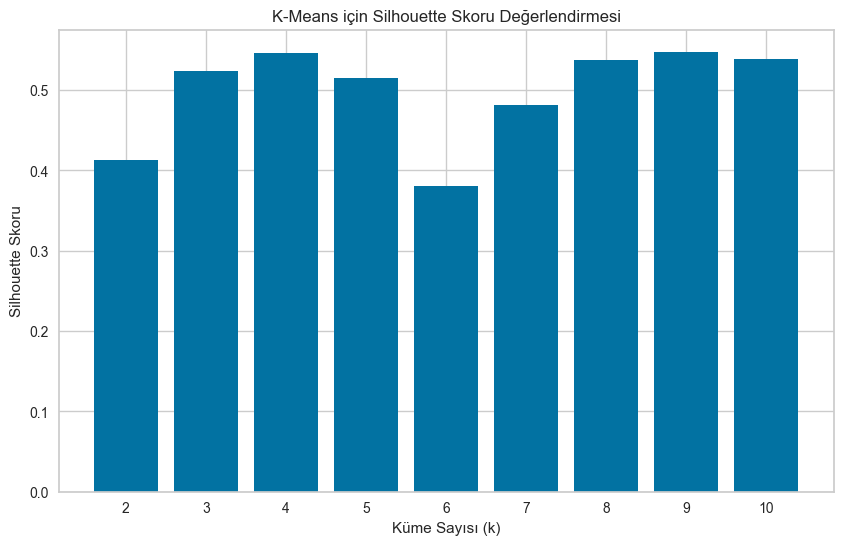


Karar ağacının görselleştirmesi

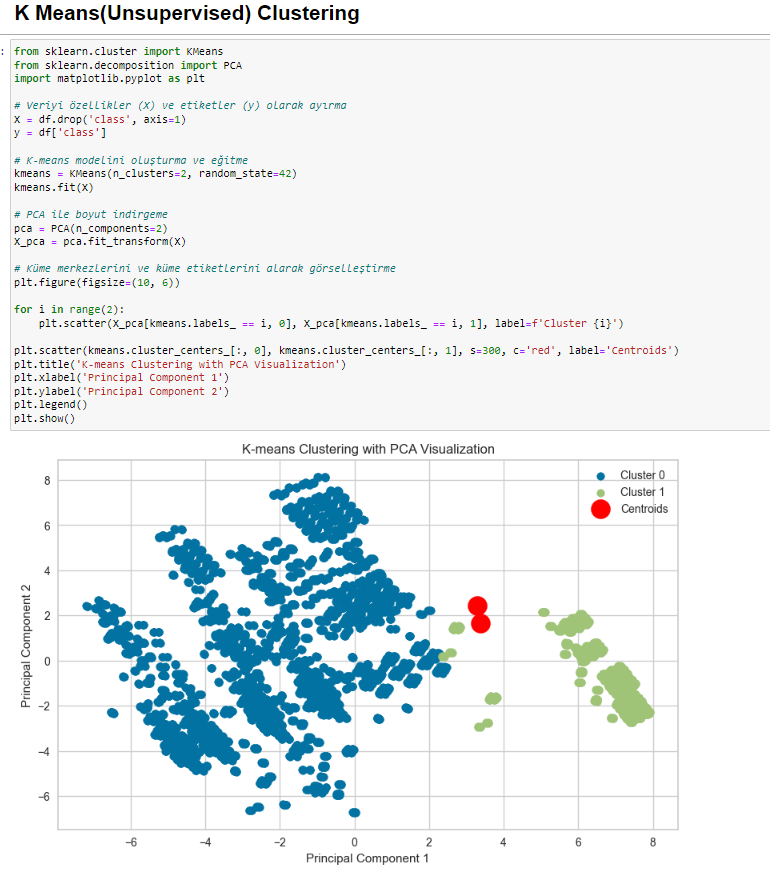




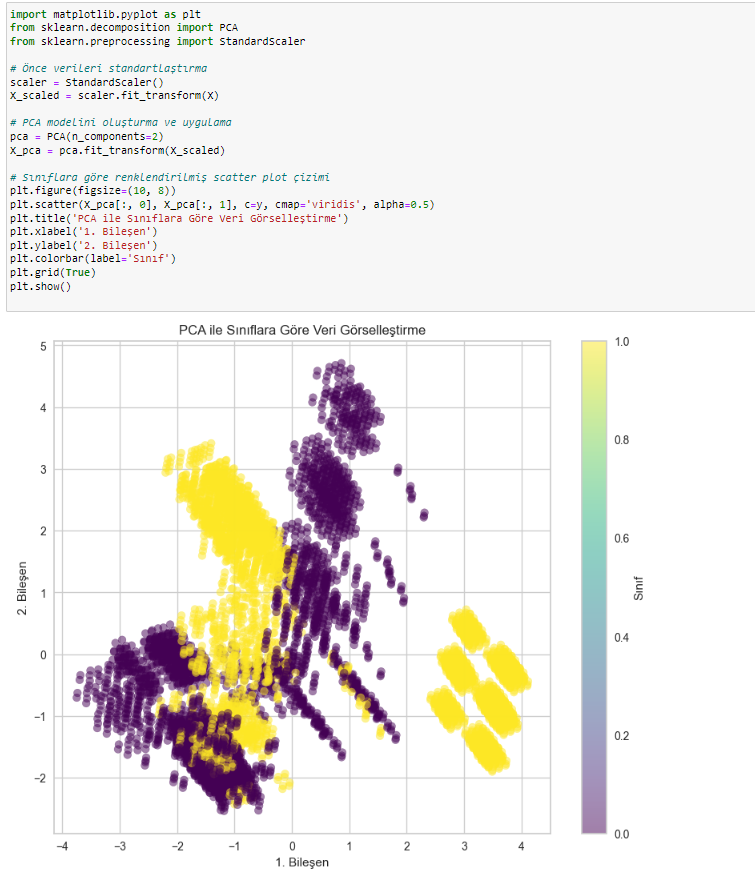
Yenilebilir mantarların %99.3 luk kısmını doğru tahmin etmekteyiz, f1 scorelarımız da 1 e çok yakın olduğu için modelimizi anlamlı ve %100 e yakın olduğunu kabul edebiliriz.

K Means ve PCA ile görselleştirme

Küme sayımız olan 2 için yaklaşık %40 doğruluk oranı görünmektedir.



Verimizi k ortalamalar algoritmasına göre sınıflandırıp, boyut 22 olan boyutumuzu 2 boyuta indirgeyip, tahmin edilen sınıfları görselleştirdik.



Ancak asıl verimizi görselleştirdiğimizde büyük bir kısmı yanlış sınıflandırdığımız görülmektedir.

Tüm Kodlar

import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, f1\_score

from sklearn.preprocessing import OneHotEncoder, LabelEncoder

from sklearn.pipeline import Pipeline

from sklearn.svm import LinearSVC, NuSVC, SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.linear\_model import LogisticRegressionCV, LogisticRegression, SGDClassifier

from sklearn.ensemble import BaggingClassifier, ExtraTreesClassifier, RandomForestClassifier

from yellowbrick.classifier import ClassificationReport

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import cross\_val\_score

from sklearn.tree import DecisionTreeClassifier, plot\_tree

from sklearn.cluster import KMeans

df=pd.read\_csv("mushrooms.csv")

df.head()

df.info()

df.dtypes

df.isna().sum()

df.describe()

plt.figure(figsize=(16, 20))

for i, column in enumerate(df.columns[1:], 1):

plt.subplot(6, 4, i)

sns.countplot(x=column, data=df, palette="viridis")

plt.title(f"{column} Countplot")

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

categorical\_counts = df.apply(lambda col: col.value\_counts().shape[0])

# Sütun grafiği oluşturalım

plt.figure(figsize=(10, 6))

categorical\_counts.plot(kind='bar', color='skyblue')

plt.title('Her Sütundaki Kategorik Değişkenlerin Sayısı')

plt.xlabel('Sütunlar')

plt.ylabel('Değer Sayısı')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

def cramers\_v\_matrix(data):

categorical\_vars = data.select\_dtypes(include=['object', 'category']).columns

n\_vars = len(categorical\_vars)

matrix = np.zeros((n\_vars, n\_vars))

for i, var1 in enumerate(categorical\_vars):

for j, var2 in enumerate(categorical\_vars):

if i <= j:

confusion\_matrix = pd.crosstab(data[var1], data[var2])

chi2 = chi2\_contingency(confusion\_matrix)[0]

n = confusion\_matrix.sum().sum()

phi2 = chi2 / n

r, k = confusion\_matrix.shape

phi2corr = max(0, phi2 - ((k - 1) \* (r - 1)) / (n - 1))

rcorr = r - ((r - 1) \*\* 2) / (n - 1)

kcorr = k - ((k - 1) \*\* 2) / (n - 1)

matrix[i, j] = np.sqrt(phi2corr / min((kcorr - 1), (rcorr - 1)))

matrix[j, i] = matrix[i, j]

return pd.DataFrame(matrix, index=categorical\_vars, columns=categorical\_vars)

# Örnek kullanım

# Veri setiniz 'data' adlı bir DataFrame ise:

# data = pd.read\_csv("veri.csv")

cramer\_v\_results = cramers\_v\_matrix(df)

plt.figure(figsize=(10, 8))

sns.heatmap(cramer\_v\_results, annot=True, cmap='coolwarm', fmt=".2f")

plt.title("Cramer's V Skorları Heatmap")

plt.xlabel("Değişkenler")

plt.ylabel("Değişkenler")

plt.show()

# Grafikleri yan yana sığdırmak için subplotlar oluşturalım

fig, axes = plt.subplots(nrows=6, ncols=4, figsize=(20, 20))

# Her bir değişken için gruplanmış verileri çizelim

for i, ax in enumerate(axes.flatten()):

if i < len(df.columns):

grouped = df.groupby([df.columns[i], 'class']).size().unstack(fill\_value=0)

colors = ["#DA70D6", "#778899"]

grouped.plot(kind='bar', color=colors, stacked=True, ax=ax)

ax.set\_title('{}'.format(df.columns[i]))

ax.set\_xlabel('')

ax.set\_ylabel('Count')

ax.legend(title='Class')

# Eksenlerin üst üste gelmemesi için layout ayarlayalım

plt.tight\_layout()

plt.show()

grouped = df.groupby(['odor', 'spore-print-color', 'gill-color','class']).size().unstack(fill\_value=0)

colors = ["#DA70D6", "#778899"]

grouped.plot(kind='bar',color=colors, stacked=True)

plt.title('High Correlation Variables According Class')

plt.xlabel('Habitat and Population')

plt.ylabel('Count')

plt.xticks([]) # X ekseni etiketlerini kaldırma

plt.legend(title='Class')

plt.show()

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

df = df.apply(label\_encoder.fit\_transform)

X = df.drop('class', axis=1) # 'class' sütununu hedef değişken olarak kabul ediyoruz

y = df['class']

models = [

SVC(gamma='auto'), NuSVC(gamma='auto'), LinearSVC(),

SGDClassifier(max\_iter=100, tol=1e-3), KNeighborsClassifier(),

LogisticRegression(solver='lbfgs'), LogisticRegressionCV(cv=3),

BaggingClassifier(), ExtraTreesClassifier(n\_estimators=300),

RandomForestClassifier(n\_estimators=300)

]

def score\_model(X, y, estimator, \*\*kwargs):

"""

Test various estimators.

"""

y = LabelEncoder().fit\_transform(y)

model = Pipeline([

('one\_hot\_encoder', OneHotEncoder()),

('estimator', estimator)

])

# Instantiate the classification model and visualizer

model.fit(X, y, \*\*kwargs)

expected = y

predicted = model.predict(X)

# Compute and return F1 (harmonic mean of precision and recall)

print("{}: {}".format(estimator.\_\_class\_\_.\_\_name\_\_, f1\_score(expected, predicted)))

for model in models:

score\_model(X, y, model)

models = [

("SVC", SVC()),

("NuSVC", NuSVC()),

("LinearSVC", LinearSVC()),

("SGDClassifier", SGDClassifier()),

("KNeighborsClassifier", KNeighborsClassifier()),

("LogisticRegression", LogisticRegression()),

("LogisticRegressionCV", LogisticRegressionCV(cv=3)),

("BaggingClassifier", BaggingClassifier()),

("ExtraTreesClassifier", ExtraTreesClassifier()),

("RandomForestClassifier", RandomForestClassifier())

]

# Modelleri değerlendir ve çapraz doğrulama skorlarını topla

cv\_results = []

model\_names = []

for name, model in models:

scores = cross\_val\_score(model, X, y, scoring='f1', cv=5)

cv\_results.extend(scores)

model\_names.extend([name] \* len(scores))

# Skorları içeren bir veri çerçevesi oluştur

df\_scores = pd.DataFrame({"Model": model\_names, "F1 Score": cv\_results})

# Kutu grafiğiyle görselleştir

plt.figure(figsize=(12, 6))

sns.boxplot(data=df\_scores, x="Model", y="F1 Score")

plt.title('Cross-Validation F1 Scores of Classification Models')

plt.xlabel('Classification Models')

plt.ylabel('F1 Score')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

from sklearn.model\_selection import cross\_val\_score

models = [

("SVC", SVC()),

("NuSVC", NuSVC()),

("LinearSVC", LinearSVC()),

("SGDClassifier", SGDClassifier()),

("KNeighborsClassifier", KNeighborsClassifier()),

("LogisticRegression", LogisticRegression()),

("LogisticRegressionCV", LogisticRegressionCV(cv=3)),

("BaggingClassifier", BaggingClassifier()),

("ExtraTreesClassifier", ExtraTreesClassifier()),

("RandomForestClassifier", RandomForestClassifier())

]

# Modelleri değerlendir ve çapraz doğrulama skorlarını topla

cv\_results = []

model\_names = []

for name, model in models:

scores = cross\_val\_score(model, X, y, scoring='accuracy', cv=5)

cv\_results.extend(scores)

model\_names.extend([name] \* len(scores))

# Skorları içeren bir veri çerçevesi oluştur

df\_scores = pd.DataFrame({"Model": model\_names, "Accuracy": cv\_results})

# Kutu grafiğiyle görselleştir

plt.figure(figsize=(12, 6))

sns.boxplot(data=df\_scores, x="Model", y="Accuracy")

plt.title('Cross-Validation Accuracy Scores of Classification Models')

plt.xlabel('Classification Models')

plt.ylabel('Accuracy')

plt.xticks(rotation=45)

plt.tight\_layout()

plt.show()

from sklearn.model\_selection import cross\_val\_score

from sklearn.neighbors import KNeighborsClassifier

# KNN sınıflandırıcıyı oluşturma

knn = KNeighborsClassifier(n\_neighbors=5)

# Çapraz doğrulama yapma

scores = cross\_val\_score(knn, X, y, cv=5) # cv parametresi K için kullanılacak kat sayıyı belirtir

# Sonuçları görüntüleme

print("Çapraz Doğrulama Skorları:", scores)

print("Ortalama Doğruluk:", scores.mean())

from sklearn.model\_selection import GridSearchCV

param\_grid = {

'n\_neighbors': range(1, 20), # Komşu sayısı

'weights': ['uniform', 'distance'], # Ağırlık fonksiyonu

'metric': ['euclidean', 'manhattan', 'chebyshev'] # Mesafe metriği

}

# Grid arama ile hiperparametre optimizasyonu yapma

grid\_search = GridSearchCV(knn, param\_grid, cv=5)

grid\_search.fit(X, y)

# En iyi hiperparametreleri ve en iyi skoru görüntüleme

print("En İyi Hiperparametreler:", grid\_search.best\_params\_)

print("En İyi Skor:", grid\_search.best\_score\_)

from sklearn.model\_selection import cross\_val\_score

from sklearn.neighbors import KNeighborsClassifier

# KNN sınıflandırıcıyı oluşturma

knn = KNeighborsClassifier(n\_neighbors=2,metric="manhattan",weights="uniform")

# Çapraz doğrulama yapma

scores = cross\_val\_score(knn, X, y, cv=5) # cv parametresi K için kullanılacak kat sayıyı belirtir

# Sonuçları görüntüleme

print("Çapraz Doğrulama Skorları:", scores)

print("Ortalama Doğruluk:", scores.mean())

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=21)

# Logistic Regresyon modelini oluştur ve eğit

log\_reg = LogisticRegression()

log\_reg.fit(X\_train, y\_train)

# Test seti üzerinde modelin performansını değerlendir

y\_pred = log\_reg.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

cv\_scores = cross\_val\_score(log\_reg, X, y, cv=5) # 5 katlı cross-validation

print("Cross-Validation Scores:", cv\_scores)

print("Mean CV Score:", cv\_scores.mean())

from sklearn.model\_selection import GridSearchCV

# Grid Search için parametrelerin belirlenmesi

param\_grid = {'C': [0.1, 1, 10, 100], 'penalty': ['l1', 'l2']}

# Grid Search ile Logistic Regresyon modelini optimize et

grid\_search = GridSearchCV(LogisticRegression(), param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

# En iyi parametreleri ve en iyi skoru yazdır

print("Best Parameters:", grid\_search.best\_params\_)

print("Best Score:", grid\_search.best\_score\_)

log\_reg.fit(X\_train, y\_train)

coefficients = log\_reg.coef\_[0]

feature\_importance = pd.DataFrame({'Feature': X.columns, 'Importance': np.abs(coefficients)})

feature\_importance = feature\_importance.sort\_values('Importance', ascending=True)

feature\_importance.plot(x='Feature', y='Importance', kind='barh', figsize=(10, 6))

X\_log=df[["gill-size","gill-spacing","veil-color","stalk-surface-above-ring","stalk-root","ring-type"]]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_log, y, test\_size=0.2, random\_state=21)

# Logistic Regresyon modelini oluştur ve eğit

log\_reg = LogisticRegression()

log\_reg.fit(X\_train, y\_train)

# Test seti üzerinde modelin performansını değerlendir

y\_pred = log\_reg.predict(X\_test)

accuracy = accuracy\_score(y\_test, y\_pred)

print("Accuracy:", accuracy)

cv\_scores = cross\_val\_score(log\_reg, X\_log, y, cv=5) # 5 katlı cross-validation

print("Cross-Validation Scores:", cv\_scores)

print("Mean CV Score:", cv\_scores.mean())

# Grid Search için parametrelerin belirlenmesi

param\_grid = {'C': [0.1, 1, 10, 100], 'penalty': ['l1', 'l2']}

# Grid Search ile Logistic Regresyon modelini optimize et

grid\_search = GridSearchCV(LogisticRegression(), param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

# En iyi parametreleri ve en iyi skoru yazdır

print("Best Parameters:", grid\_search.best\_params\_)

print("Best Score:", grid\_search.best\_score\_)

cv\_scores = cross\_val\_score(LogisticRegression(C= 10, penalty='l2'), X\_log, y, cv=5) # 5 katlı cross-validation

print("Cross-Validation Scores:", cv\_scores)

print("Mean CV Score:", cv\_scores.mean())

log\_reg = LogisticRegression(C= 10, penalty='l2')

log\_reg.fit(X\_train, y\_train)

# Test seti üzerinde tahmin yap

y\_pred\_proba = log\_reg.predict\_proba(X\_test)[:,1] # Sadece pozitif sınıfın olasılıklarını al

# ROC eğrisini çizdir

fpr, tpr, thresholds = roc\_curve(y\_test, y\_pred\_proba)

plt.plot(fpr, tpr)

plt.plot([0, 1], [0, 1], '--', color='gray') # Rastgele tahmin çizgisi

plt.xlabel('False Positive Rate')

plt.ylabel('True Positive Rate')

plt.title('ROC Curve')

plt.show()

# ROC AUC skorunu hesapla

roc\_auc = roc\_auc\_score(y\_test, y\_pred\_proba)

print("ROC AUC Score:", roc\_auc)

cv = StratifiedKFold(5)

visualizer = RFECV(RandomForestClassifier(), cv=cv, scoring='accuracy')

visualizer.fit(X, y)

visualizer.show()

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

model=RandomForestClassifier(n\_estimators=100, random\_state=42)

sfs = SFS(model,

k\_features=3,

forward=True,

floating=False,

scoring='accuracy',

cv=5)

sfs.fit(X\_train, y\_train)

selected\_features\_idx = sfs.k\_feature\_idx\_

selected\_features\_names = X.columns[list(selected\_features\_idx)]

print("En iyi 3 özellik:", selected\_features\_names)

from sklearn.model\_selection import GridSearchCV

# Hiperparametre aralıkları

param\_grid = {

'n\_estimators': [50, 100, 200],

'max\_depth': [None, 10, 20],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4]

}

# GridSearchCV kullanarak optimizasyon

rf = RandomForestClassifier(random\_state=42)

grid\_search = GridSearchCV(estimator=model, param\_grid=param\_grid, cv=3, scoring='accuracy')

grid\_search.fit(X\_train, y\_train)

print("En iyi parametreler:", grid\_search.best\_params\_)

print("En iyi doğruluk:", grid\_search.best\_score\_)

X=df[['cap-color', 'odor', 'spore-print-color']]

y=df["class"]

# Create a cross-validation strategy

cv = StratifiedKFold(n\_splits=12, shuffle=True, random\_state=42)

visualizer = CVScores(RandomForestClassifier(max\_depth= None, min\_samples\_leaf= 1, min\_samples\_split= 2, n\_estimators= 50), cv=cv, scoring='f1\_weighted')

visualizer.fit(X, y)

visualizer.show()

selected\_features = ['cap-color', 'odor', 'spore-print-color']

X\_selected = df[selected\_features]

y = df['class']

# One-hot encoding ile veriyi işleme

X\_encoded = pd.get\_dummies(X\_selected)

# Eğitim ve test veri setlerini ayırma

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X\_encoded, y, test\_size=0.2, random\_state=42)

# Tek bir ağaç oluşturma ve eğitme

single\_tree\_model = DecisionTreeClassifier(random\_state=42)

single\_tree\_model.fit(X\_train, y\_train)

# Karar ağacını çizme

# Karar ağacını çizme

plt.figure(figsize=(15, 10))

plot\_tree(single\_tree\_model, feature\_names=X\_encoded.columns, class\_names=y.unique().astype(str), filled=True)

plt.show()

from yellowbrick.classifier import ClassificationReport

pipeline = Pipeline([

('estimator', model)

])

# Create the classification report visualizer

visualizer = ClassificationReport(

pipeline, classes=['edible', 'poisonous'], cmap="YlGn", size=(600, 360)

)

# Fit the visualizer and generate the classification report

visualizer.fit(X\_train, y\_train)

visualizer.score(X\_test, y\_test)

# Display the visualizer

visualizer.show()

X = df.drop('class', axis=1)

y = df['class']

# K-means modelini oluşturma ve eğitme

kmeans = KMeans(n\_clusters=2, random\_state=42)

kmeans.fit(X)

# PCA ile boyut indirgeme

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X)

# Küme merkezlerini ve küme etiketlerini alarak görselleştirme

plt.figure(figsize=(10, 6))

for i in range(2):

plt.scatter(X\_pca[kmeans.labels\_ == i, 0], X\_pca[kmeans.labels\_ == i, 1], label=f'Cluster {i}')

plt.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s=300, c='red', label='Centroids')

plt.title('K-means Clustering with PCA Visualization')

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.legend()

plt.show()

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# PCA modelini oluşturma ve uygulama

pca = PCA(n\_components=2)

X\_pca = pca.fit\_transform(X\_scaled)

# Sınıflara göre renklendirilmiş scatter plot çizimi

plt.figure(figsize=(10, 8))

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y, cmap='viridis', alpha=0.5)

plt.title('PCA ile Sınıflara Göre Veri Görselleştirme')

plt.xlabel('1. Bileşen')

plt.ylabel('2. Bileşen')

plt.colorbar(label='Sınıf')

plt.grid(True)

plt.show()

import matplotlib.pyplot as plt

import numpy as np

from sklearn.cluster import KMeans

from sklearn.metrics import silhouette\_score

k\_values = range(2, 11) # 2'den 10'a kadar küme sayılarını deneyin

# Her bir küme sayısı için Silhouette skorunu hesaplayın

silhouette\_scores = []

for k in k\_values:

kmeans = KMeans(n\_clusters=k, random\_state=42)

cluster\_labels = kmeans.fit\_predict(X)

silhouette\_avg = silhouette\_score(X, cluster\_labels)

silhouette\_scores.append(silhouette\_avg)

# Silhouette skorlarını görselleştirin

plt.figure(figsize=(10, 6))

plt.bar(k\_values, silhouette\_scores)

plt.xlabel('Küme Sayısı (k)')

plt.ylabel('Silhouette Skoru')

plt.title('K-Means için Silhouette Skoru Değerlendirmesi')

plt.xticks(np.arange(min(k\_values), max(k\_values)+1, 1))

plt.show()

Sonuç

Verimiz çok düzgün ayarlanmış olup neredeyse tüm sınıflandırma modellemelerinde %100 e yakın doğruluk oranları verip, gerekli analizlerle bile modelsiz iyi tahminlemeler yapılabilir.Ancak ensemble metodlar daha dengeli tahminler yaptığından Random forest algoritması tercih edilmiştir.

Kaynakça

<https://www.kaggle.com/datasets/uciml/mushroom-classification>

<https://archive.ics.uci.edu/dataset/73/mushroom>

<https://www.datacamp.com>

<https://medium.com/analytics-vidhya/building-classification-model-with-python-9bdfc13faa4b>

<https://github.com/kanchitank/Mushroom-Classification>