PARKİNSON HASTALIĞI

Parkinson hastalığı, beyinde dopamin adı verilen, beyin hücrelerinin birbirleriyle haberleşmesini sağlayan maddeyi üreten hücrelerin bozulması sonucu ortaya çıkar. Beyinde dopamini üreten hücreler hareketlerin kontrolünden, uyumundan ve akıcılığından sorumludur. Hareketlerde yavaşlık, dinlenme halindeyken titreme, psikiyatrik rahatsızlıklarla kendini belli eden hastalık, çoğunlukla 60 yaş sonrası kişilerde görülür. Ancak genetik nedenlerle 40'lı yaşlarda da rastlanabilir. Tedavide ilk seçenek ilaçlardır. Ancak bazı kişilerde ilaç tedavisi yetersiz kaldığında ya da yoğun yan etkiler gelişirse "beyin pili" yöntemi tercih edilebilir.

Atipik Parkinson veya Parkinson plus hastalıklar erken dönemlerde Parkinson hastalığını taklit edebilirler. Klasik Parkinson diyebilmek için ana bulgulara ek olarak kısa dönem bile olsa hastanın Levodopa'ya yanıtı iyi olmalıdır. Atipik Parkinson hastalığı denildiğinde multiple sistem atrofisi, ilerleyici supranukleer palsi ve kortikobazal dejenerasyon dediğimiz hastalıkları sayabiliriz. İkincil parkinsonizm nondejeneratif olarak tanımlanabilen farklı nedenlerin sonucunda görülür; bunlar ilaç kullanımı, toksine maruz kalma, beyinde su toplama ya da beyin tümörü olabilir.

PARKİNSON HASTALIĞI NEDENLERİ

Parkinson hastalığı beyinde dopamin üreten bölgedeki hücre kaybı nedeniyle bu maddenin az salınımı sonucu oluşur. Bu hücre kaybına zirai ilaçlar gibi kimi kimyasallar neden olabilmekle beraber, genetik faktörler de sebep olabilmektedir.

ÖZET

Parkinson hastalığı (PH), dopamin üreten beyin hücrelerinin ölmesiyle ya da zarar görmesiyle ortaya çıkan bir beyin hastalığıdır. Böyle bir durumda, beyin normal fonksiyonlarını yerine getiremez. PH, konuşma, yürüme ve yazma gibi insan hareketlerini olumsuz olarak etkiler. Bu hastalığın teşhisinde detaylı tıbbi öykü, geçmiş tedavi öyküsü, fiziksel testler ve bazı kan testleri ile beyin filmleri istenilmektedir. Bu işlemler maliyetli ve meşakkatli olabildiği için daha az maliyetli ve daha kolay yapılabilen teşhis bu noktada önem kazanmaktadır. Bu çalışmada doktorun kararına destek olabilmesi için 252 bireyden alınan ses verileri ile PH'ın teşhis edilebilmesi amaçlanmıştır. Verilerden daha iyi sonuç alabilmek için bazı ön işlemler uygulanmıştır. Verilerde dengeleme işlemi yapılmış ve sistematik örnekleme metodu ile işleme alınacak veriler belirlenmiştir. Öznitelik seçme algoritması ile özniteliklerin etiket üzerindeki etki gücü hesaplanıp bazı veri grupları oluşturulmuştur. Sınıflandırma algoritmalarından Karar ağacı, Destek Vektör Makineleri ve K En Yakın Komşu Algoritması kullanılıp, performans değerlendirme kriterleri - bunlar; Doğruluk Oranı, Duyarlılık, Özgünlük, F-Ölçümü, Kappa, Auc - değerlendirilmiştir. En yüksek performans değerine sahip veri grubu ve kullanılan sınıflandırma algoritması belirlenip model oluşturulmuştur. Model en ilgiliden en ilgisize doğru sıralanmış veri setinin %45'inden ve Destek vektör makineleri algoritması kullanılarak oluşturulmuştur. Performans kriterlerinde %85 doğruluk oranı ve diğer kriterlerde de olumlu sonuçlar elde edilmiştir. Böylece PH olma ihtimali olan bireyin ses kayıtlarından oluşturulan veri seti ve uygulanan model yardımı ile doktora tıbbi karar destek sağlanacağı anlaşılmıştır.

KULLANILAN VERİ SETİ VE ÖZELLİKLERİ

Öncelikli olarak kullanmış olduğum veri setimdeki veriler hastalardan toplanan ses sinyalleri ve ph değerlerinden oluşmaktadır. Veri seti içeriği olarak 195 örneklem ve 24 adet öznitelikten oluşmaktadır.Amacım ise bu 24 öznitelikten sınıflandırma ve tahmin yapmaya yönelik bir öznitelik bulmaktır.Bu öznitelik ise durum yani("status) niteliğidir. Yapılılan bilimsel testler sonucu veri setimde mevcut bulunan kişilerin çeşitli parametlerce ortaya çıkan sonucu şunu göstermektedir: 195 kişiden 147 kişide parkinson teşhisi konulurken, 48 kişinin sağlam olduğu ortaya çıkmıştır.

AMAÇ

Amacım ise bulmuş olduğum ayırt edici yani sınıflandırıcı ve tahmin muhakemesine sahip öznitelik değerinden yararlanarak çeşitli makine öğrenmesi algoritmaları ile bir tahmin modeli ortaya koymak.Daha sonrasıanda bu tahmin modellerine dayanarak geçmiş veriler bazında yeni gelecek değerler karşısında algoritmaların doğruluk ve hata değerlerini saptamaktır. Bu işlemler sayesinde yeni gelecek bir bireyin parkinson ya da parkinson hastalığı taşıyıp taşımadığını doğru bir şekilde teyit etmektir.

VERİ ÖN İŞLEME

In [1]:

```
import warnings#hataları gözden gelmek adına kullanılan kütüphane
```

In [3]:

```
warnings.filterwarnings('ignore')
```

In [2]:

```
import numpy as np# veri setinde mevcut bulunan verilerin dizi matris işlemlerinin yapı
```

In [3]:

```
import pandas as pd# veri ön işleme,veri seti okuma,veri manipülasyonu için gerekli küt
```

In [6]:

```
from sklearn.preprocessing import MinMaxScaler
```

In [4]:

```
from sklearn.preprocessing import StandardScaler# verileri normalize etmek için kullanı
```

In [5]:

```
from sklearn.model_selection import train_test_split# eğitim ve test verileri için gere
```

In [6]:

```
from sklearn.model_selection import train_test_split, GridSearchCV,cross_val_score# mod
```

In [10]:

```
from sklearn.metrics import mean_squared_error, r2_score
```

In [11]:

```
from sklearn.metrics import accuracy_score
```

In [7]:

```
df = pd.read_csv('veribilimi.csv')# veri setinin sisteme tanıtılması
```

In [8]:

```
df.head()# veri setinin ilk 5 satırının Dataframe olarak gösterelim
```

Out[8]:

	name	MDVP:Fo(Hz)	MDVP:Fhi(Hz)	MDVP:Flo(Hz)	MDVP:Jitter(%)	MDVP:Jitter(Abs)
0	phon_R01_S01_1	119.992	157.302	74.997	0.00784	0.00007
1	phon_R01_S01_2	122.400	148.650	113.819	0.00968	0.00008
2	phon_R01_S01_3	116.682	131.111	111.555	0.01050	0.00009
3	phon_R01_S01_4	116.676	137.871	111.366	0.00997	0.00009
4	phon_R01_S01_5	116.014	141.781	110.655	0.01284	0.00011

5 rows × 24 columns

In [25]:

```
df.shape
```

Out[25]:

```
(195, 24)
```

In [9]:

```
df.isnull().sum()# boş veri yani veri kaybı gözlemlemesi
```

Out[9]:

```
name                0
MDVP:Fo(Hz)         0
MDVP:Fhi(Hz)        0
MDVP:Flo(Hz)        0
MDVP:Jitter(%)      0
MDVP:Jitter(Abs)    0
MDVP:RAP            0
MDVP:PPQ            0
Jitter:DDP          0
MDVP:Shimmer        0
MDVP:Shimmer(dB)    0
Shimmer:APQ3        0
Shimmer:APQ5        0
MDVP:APQ            0
Shimmer:DDA         0
NHR                 0
HNR                 0
status              0
RPDE                0
DFA                 0
spread1             0
spread2             0
D2                  0
PPE                 0
dtype: int64
```

In [27]:

```
df.dtypes
```

Out[27]:

```
name                 object
MDVP:Fo(Hz)         float64
MDVP:Fhi(Hz)        float64
MDVP:Flo(Hz)        float64
MDVP:Jitter(%)      float64
MDVP:Jitter(Abs)    float64
MDVP:RAP            float64
MDVP:PPQ            float64
Jitter:DDP          float64
MDVP:Shimmer        float64
MDVP:Shimmer(dB)    float64
Shimmer:APQ3        float64
Shimmer:APQ5        float64
MDVP:APQ            float64
Shimmer:DDA         float64
NHR                 float64
HNR                 float64
status                int64
RPDE                float64
DFA                 float64
spread1             float64
spread2             float64
D2                  float64
PPE                 float64
dtype: object
```

In [28]:

```
df.columns
```

Out[28]:

```
Index(['name', 'MDVP:Fo(Hz)', 'MDVP:Fhi(Hz)', 'MDVP:Flo(Hz)', 'MDVP:Jitter
(%)',
       'MDVP:Jitter(Abs)', 'MDVP:RAP', 'MDVP:PPQ', 'Jitter:DDP',
       'MDVP:Shimmer', 'MDVP:Shimmer(dB)', 'Shimmer:APQ3', 'Shimmer:APQ5',
       'MDVP:APQ', 'Shimmer:DDA', 'NHR', 'HNR', 'status', 'RPDE', 'DFA',
       'spread1', 'spread2', 'D2', 'PPE'],
      dtype='object')
```

In [29]:

```
for i in df.columns:
    print(i)
```

```
name
MDVP:Fo(Hz)
MDVP:Fhi(Hz)
MDVP:Flo(Hz)
MDVP:Jitter(%)
MDVP:Jitter(Abs)
MDVP:RAP
MDVP:PPQ
Jitter:DDP
MDVP:Shimmer
MDVP:Shimmer(dB)
Shimmer:APQ3
Shimmer:APQ5
MDVP:APQ
Shimmer:DDA
NHR
HNR
status
RPDE
DFA
spread1
spread2
D2
PPE
```

In [10]:

```
df["status"].value_counts()# burada ayırıcı öznitelik olarak durum değişkeni belirlendi
```

Out[10]:

```
1    147
0     48
Name: status, dtype: int64
```

HAM VERİYİ GÖRSELLEŞTİRME

Bu kısımda elimizdeki veri setinde herhangi bir model oluşturulmadan önce avr olan verilerimizi daha iyi anlamak adına birkaç veri görselleştirme metodu kullanıldı bunları yaparken gerekli olan ve sisteme tanıtılan kütüphaneler şunlardır:

```
Matplotlib
Seaborn
```

veri görselleştimesi yapılırken şu grafikler kullanıldı:

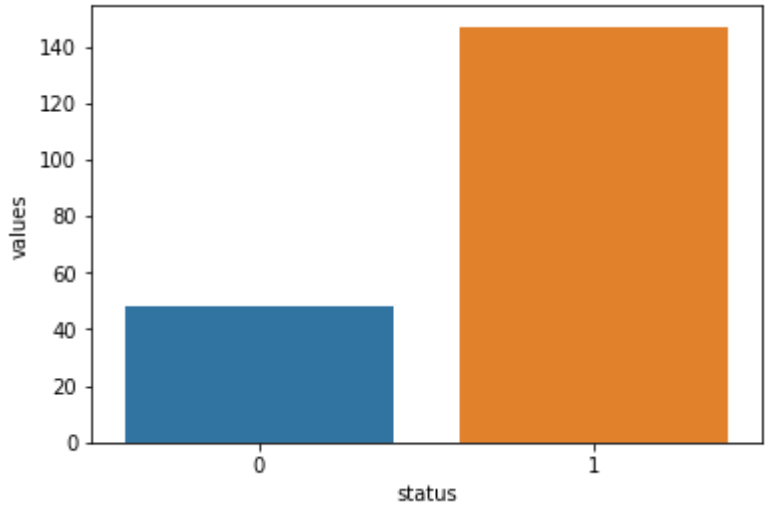
```
Boxplots
Barplots
Heatmaps
Pairplots
Displots
```

In [33]:

```
import matplotlib.pyplot as plt
import seaborn as sns

temp=df["status"].value_counts()
temp_df=pd.DataFrame({'status':temp.index,'values':temp.values})
print(sns.barplot(x='status',y="values",data=temp_df))
```

```
AxesSubplot(0.125,0.125;0.775x0.755)
```

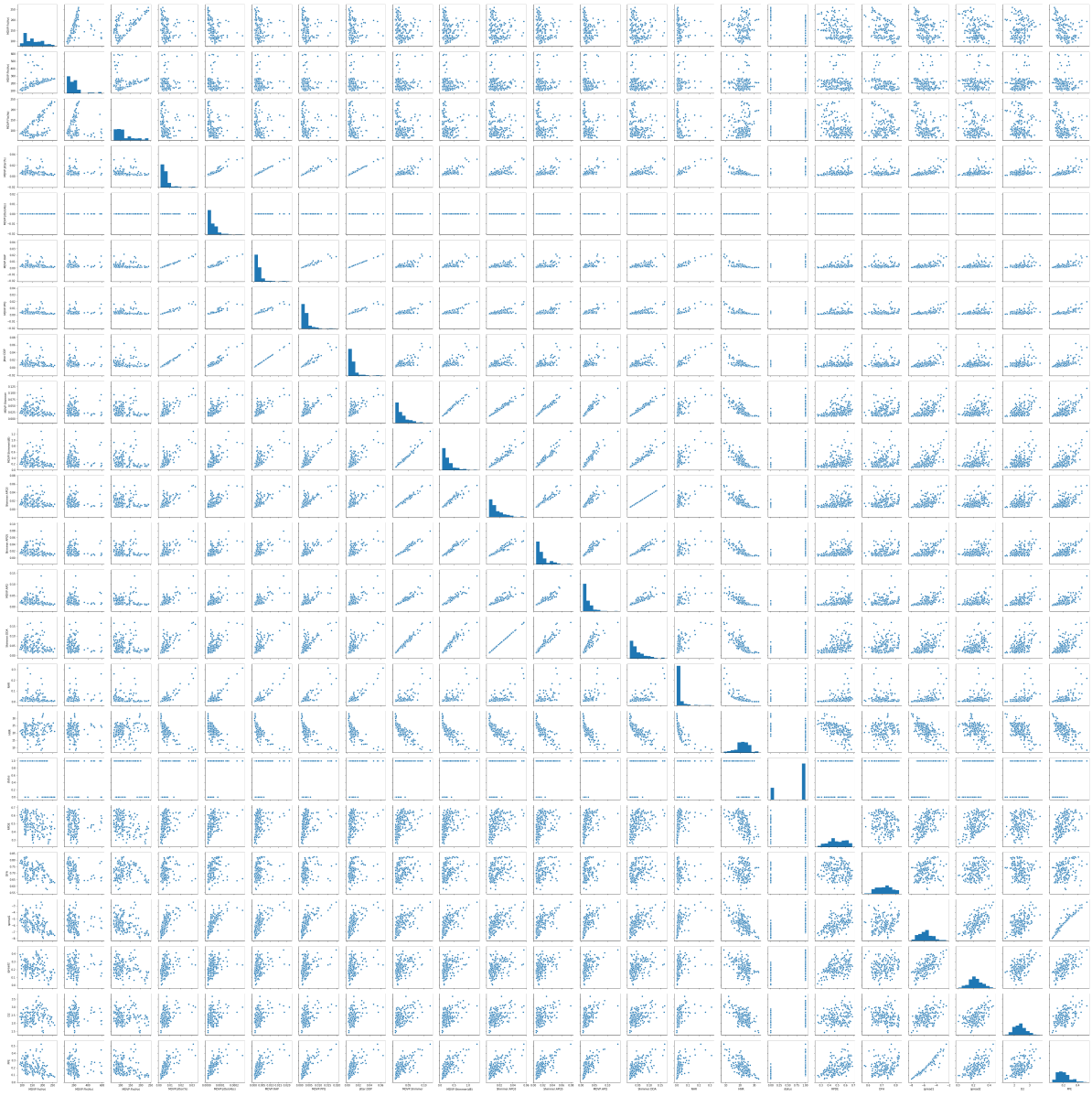


In [32]:

```
sns.pairplot(df)
```

Out[32]:

```
<seaborn.axisgrid.PairGrid at 0x23cfecb5160>
```



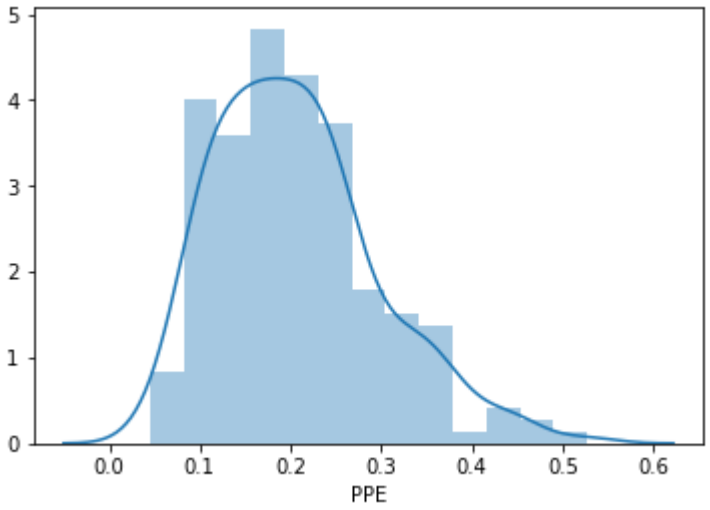
In [34]:

```
sns.distplot(df["PPE"])
```

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\sit
e-packages\matplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kwarg
is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```

Out[34]:

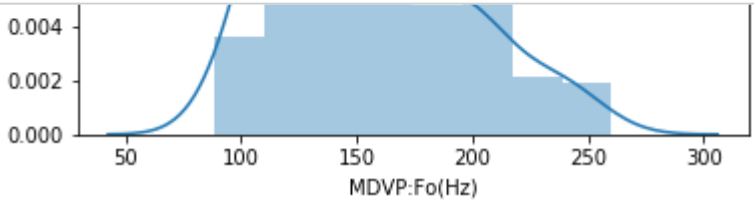
```
<matplotlib.axes._subplots.AxesSubplot at 0x23cd0268f60>
```



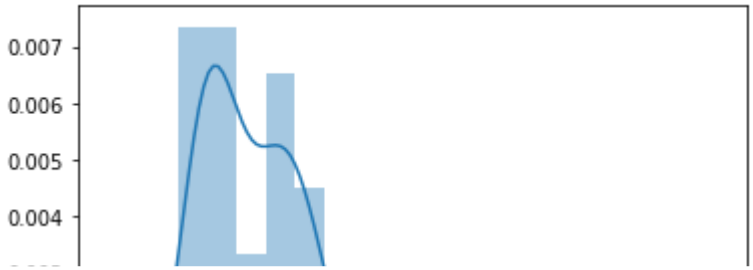
In [35]:

```
# veri dağılımını bulalım.
def distplots(col):
    sns.distplot(df[col])
    plt.show()

for i in list(df.columns[1:]):
    distplots(i)
```



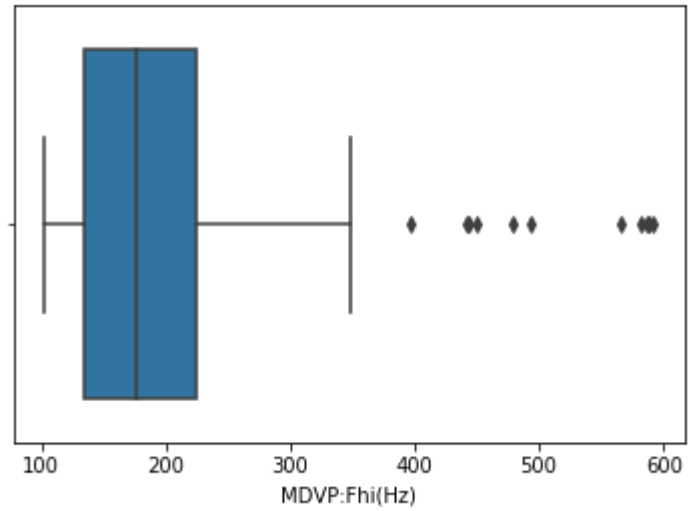

```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Anaconda3_64\lib\s
ite-packages\matplotlib\axes\_axes.py:6462: UserWarning: The 'normed' kw
arg is deprecated, and has been replaced by the 'density' kwarg.
  warnings.warn("The 'normed' kwarg is deprecated, and has been "
```



In [36]:

```
def boxplots(col):
    sns.boxplot(df[col])
    plt.show()

for i in list(df.select_dtypes(exclude=["object"]).columns)[1:]:
    boxplots(i)
```

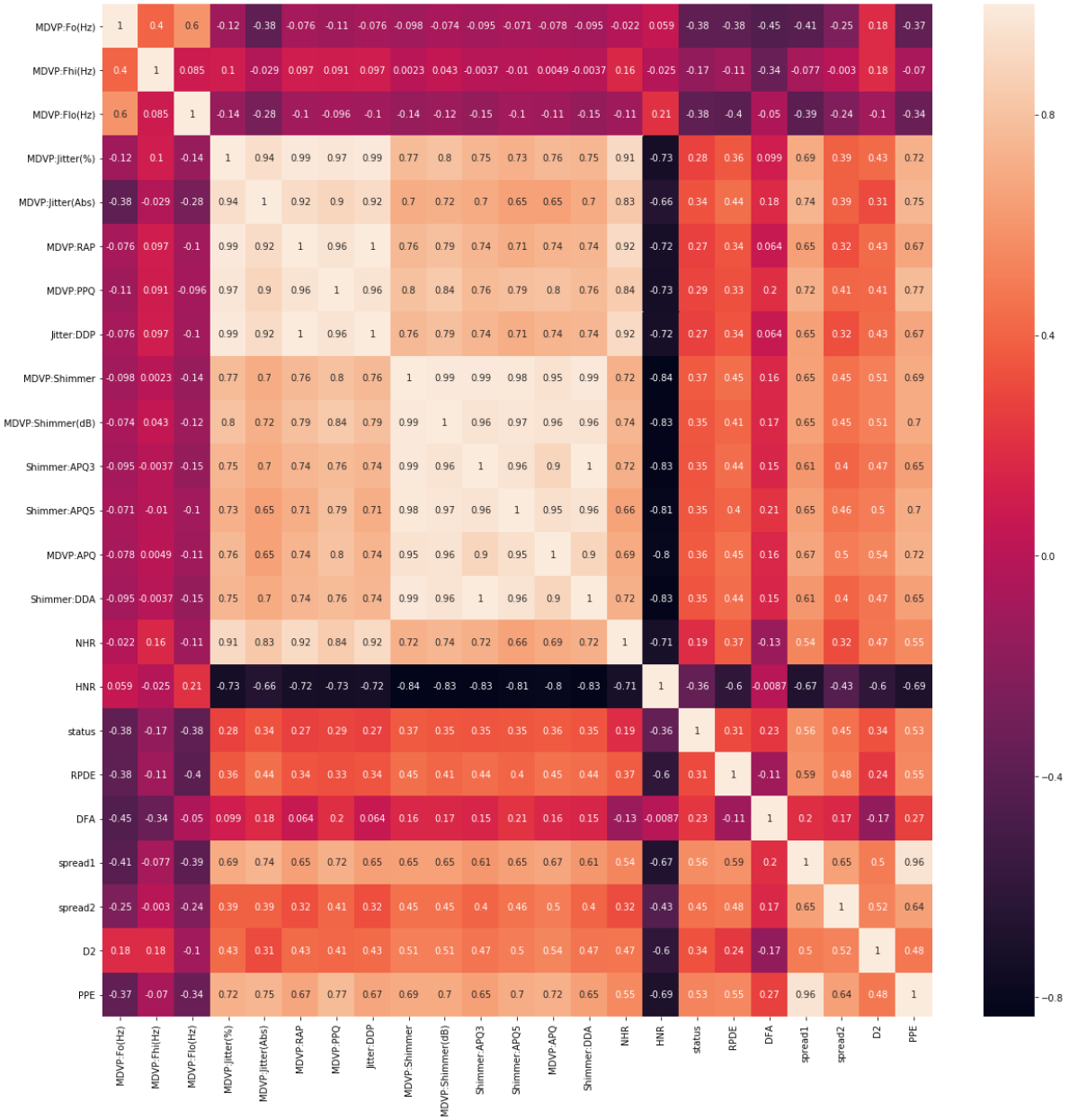


In [37]:

```
plt.figure(figsize=(20,20))
corr=df.corr()
sns.heatmap(corr,annot=True)
```

Out[37]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x23cda614f60>
```



Bağımlı ve Bağımsız Değişkenleri Ayırma

In [38]:

```
y=df["status"]
x=df.drop(["status","name"],axis=1)
```

In [39]:

```
print(x)
```

```
6        120.267       137.244       114.820         0.00333
7        107.332       113.840       104.315         0.00290
8         95.730       132.068        91.754         0.00551
9         95.056       120.103        91.226         0.00532
10        88.333       112.240        84.072         0.00505
11        91.904       115.871        86.292         0.00540
12       136.926       159.866       131.276         0.00293
13       139.173       179.139        76.556         0.00390
14       152.845       163.305        75.836         0.00294
15       142.167       217.455        83.159         0.00369
16       144.188       349.259        82.764         0.00544
17       168.778       232.181        75.603         0.00718
18       153.046       175.829        68.623         0.00742
19       156.405       189.398       142.822         0.00768
20       153.848       165.738        65.782         0.00840
21       153.880       172.860        78.128         0.00480
22       167.930       193.221        79.068         0.00442
23       173.917       192.735        86.180         0.00476
24       163.656       200.841        76.779         0.00742
25       104.400       206.002        77.968         0.00633
```

In [40]:

```
print(y)
```

```
0      1
1      1
2      1
3      1
4      1
5      1
6      1
7      1
8      1
9      1
10     1
11     1
12     1
13     1
14     1
15     1
16     1
17     1
18     1
19     1
20     1
21     1
22     1
23     1
24     1
25     1
26     1
27     1
28     1
29     1
      ..
165    0
166    0
167    0
168    0
169    0
170    0
171    0
172    0
173    0
174    0
175    0
176    0
177    1
178    1
179    1
180    1
181    1
182    1
183    0
184    0
185    0
186    0
187    0
188    0
189    0
190    0
```

```
191    0
192    0
193    0
194    0
Name: status, Length: 195, dtype: int64
```

KULLANILAN MODELLEME YÖNTEMLERİ

1-) SVR MODELİ

Bu modelimizde öncelikli olarak sistmemize destek vektör makineleri konulu kütüphanesini tensorflow yardımı ile tanıttık. Daha sonra ise veri setimizi eğitim ve test olmak üzere iki kısıma ayırdık ve elimizdeki bağımsız eğitim değişkenlerini bir normalizasyon işlemine tabi tuttuk.Daha sonrasında ise normalize edilmiş eğitim verileri ve elimizde var olan test verileri ile bir destek vektör regrestonu modeli oluşturduk.Ve başarı oranımızı elde ettik.

In [16]:

```
from sklearn.svm import SVR
```

In [17]:

```
x_train, x_test, y_train, y_test=train_test_split(x,y,test_size=0.2,random_state=42)
```

In [19]:

```
print(x.shape,x_train.shape,x_test.shape)
```

```
(195, 22) (156, 22) (39, 22)
```

In [188]:

```
scaler=StandardScaler()
```

In [189]:

```
scaler.fit(x_train)
```

Out[189]:

```
StandardScaler(copy=True, with_mean=True, with_std=True)
```

In [190]:

```
x_train=scaler.transform(x_train)
x_test=scaler.transform(x_test)
```

In [191]:

```
model=svm.SVC(kernel='linear')
```

In [192]:

```
model.fit(x_train,y_train)
```

Out[192]:

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='linear',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

In [193]:

```
x_train_prediction=model.predict(x_train)
train_data_accuracy=accuracy_score(y_train,x_train_prediction)
```

In [194]:

```
print('model_basarisi', train_data_accuracy)#svm model basarisi
```

```
model_basarisi 0.9038461538461539
```

2-)ANN MODELİ

Yapay sinir ağları modelimizde öncelikli olarak sisteme gerekli kütüphaneyi tensorflow yardımı ile tanıttık. Daha sonra veri setimizi sisteme okutup,bağımlı ve bağımsız değişkenelri atadıktan sonra verilerimizi eğitim ve test verisi olarak böldük. ardından bağımsız test ve bağımsız eğitim değişkenlerimizi bir normalizasyon işlemine tabi tuttuk. Bir yapay sinir ağı modeli oluşturduk ve el yordamı ile ara katman parametrelerini belirledik,hata kareler ortalaması yöntemi ile hata oranı tespit edilerek modelin daha optimal sonnuç değerlerine ulaşabilmesi adına bir iyileştirme hedeflendi. İyileştirme yapabilmek adına k katlı crossvalidation ve grid search algoritmaları kullamılarak belirli parametrelerce bir sözlük yapaısı oluşturuldu ve en iyi parametrelerin bulunması sağlandı. Daha sonra bulunan en iyi parametreler ile bir dönüştürülmüş model tasarlandı ve tam bu modelde tam başarı sağlandı.

In [41]:

```
from sklearn.neural_network import MLPClassifier
```

In [44]:

```
par = pd.read_csv('veribilimi.csv')
df = par.copy()
df = df.dropna()
y=df["status"]
x=df.drop(["status","name"],axis=1)

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=
```

In [224]:

```
#değişken standartlaştırması
scaler = StandardScaler()
scaler.fit(X_train)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

In [225]:

```
mlp_model = MLPRegressor(hidden_layer_sizes = (15,6)).fit(X_train_scaled, y_train)
print(mlp_model)
print(mlp_model.n_layers_)
print(mlp_model.hidden_layer_sizes)
```

```
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.
9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(15, 6), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
4
(15, 6)
```

In [226]:

```
X_test_scaled[0:5]
```

Out[226]:

```
array([[-0.99491236, -0.74459136, -0.29897103, -0.27526346, -0.09893719,
        -0.27934417, -0.18212835, -0.27844768,  1.30979308,  1.09849109,
         1.38673413,  1.32743041,  1.08758352,  1.38706468,  0.0082092 ,
        -0.97387207,  0.58262298,  1.3602683 , -0.38472133,  0.08546895,
         0.96450045,  0.10664994,  0.05355063],
       [-0.21458024,  1.77944465, -0.81790045, -0.14069761, -0.09893719,
        -0.36139892, -0.1755773 , -0.36252804, -0.46479189, -0.43929921,
        -0.5515292 , -0.46859336, -0.20296759, -0.55123673, -0.16419469,
         0.06791309,  0.58262298,  0.66229819, -1.43440194,  0.26147527,
         0.16703424, -0.28608523,  0.14967735],
       [-0.8573791 , -0.73620247, -0.50431927,  0.05180632,  0.16708178,
         0.07926547, -0.14937307,  0.07813409,  0.04843624, -0.04183956,
         0.08502563, -0.00301467, -0.02488566,  0.08502309,  0.008657  ,
         0.08846774,  0.58262298,  1.24029505, -1.29867568,  1.94097775,
        -0.21635141,  0.15827968,  0.02210831],
       [ 0.15002579, -0.26501885,  0.69154653, -0.51822958, -0.63097515,
        -0.46168806, -0.52933436, -0.46079063,  0.19352189,  0.10484197,
         0.22063948,  0.16656698,  0.19608797,  0.22033209, -0.38966314,
         0.13712773,  0.58262298, -1.24523378, -0.53233035, -0.62282674,
        -0.37629003,  0.24365755, -0.56224744],
       [-0.23057816, -0.34094965, -1.23037282,  0.90031877,  0.96513871,
         0.96363332,  0.39764017,  0.96351041,  1.18197954,  0.91395625,
         1.46238267,  0.47335559,  0.51261775,  1.46209967,  0.51265849,
        -0.35052085,  0.58262298,  0.14860016,  0.17128808, -0.12730457,
        -0.89163978, -0.65662195, -0.1993913 ]])
```

In [227]:

```
y_pred = mlp_model.predict(X_test_scaled)
np.sqrt(mean_squared_error(y_test, y_pred))
```

Out[227]:

```
1.0254479322190861
```

In [228]:

```
mlp_model
```

Out[228]:

```
MLPRegressor(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.
9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(15, 6), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

In [229]:

```
mlp_params = {'alpha': [0.1, 0.01,0.02,0.005],
              'hidden_layer_sizes': [(18,11),(64,50,8),(128,64,32),(10,7,1)],
              'activation': ['relu','logistic','tanh','sigmoid']}
```

In [230]:

```
mlp_cv_model = GridSearchCV(mlp_model, mlp_params, cv = 10)
mlp_cv_model.fit(X_train_scaled, y_train)
```

Out[230]:

```
GridSearchCV(cv=10, error_score='raise',
       estimator=MLPRegressor(activation='relu', alpha=0.0001, batch_size
='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(15, 6), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'alpha': [0.1, 0.01, 0.02, 0.005], 'hidden_layer_size
s': [(18, 11), (64, 50, 8), (128, 64, 32), (10, 7, 1)], 'activation': ['re
lu', 'logistic', 'tanh']},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [232]:

```
mlp_cv_model.best_params_
```

Out[232]:

```
{'activation': 'relu', 'alpha': 0.1, 'hidden_layer_sizes': (128, 64, 32)}
```

In [233]:

```
mlp_tuned = MLPRegressor(activation=mlp_cv_model.best_params_['activation'],
                         alpha = mlp_cv_model.best_params_['alpha'],
                         hidden_layer_sizes = mlp_cv_model.best_params_['hidden_layer_s
print(mlp_tuned)
print(mlp_tuned.activation)
```

```
MLPRegressor(activation='relu', alpha=0.1, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(128, 64, 32), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
relu
```

In [234]:

```
mlp_tuned.fit(X_train_scaled, y_train)
```

Out[234]:

```
MLPRegressor(activation='relu', alpha=0.1, batch_size='auto', beta_1=0.9,
       beta_2=0.999, early_stopping=False, epsilon=1e-08,
       hidden_layer_sizes=(128, 64, 32), learning_rate='constant',
       learning_rate_init=0.001, max_iter=200, momentum=0.9,
       nesterovs_momentum=True, power_t=0.5, random_state=None,
       shuffle=True, solver='adam', tol=0.0001, validation_fraction=0.1,
       verbose=False, warm_start=False)
```

In [235]:

```
y_pred = mlp_tuned.predict(X_test_scaled)
np.sqrt(mean_squared_error(y_test, y_pred))
```

Out[235]:

```
0.09804710804433382
```

In [236]:

```
mlpc = MLPClassifier().fit(X_train_scaled, y_train)
```

In [237]:

```
y_pred = mlpc.predict(X_test_scaled)
accuracy_score(y_test, y_pred)
```

Out[237]:

```
1.0
```

3-) KNN MODELİ

üçüncü olarak kullandığım bu modelde öncelikli oalrak sistemimize k en yakın komşu algoritması kütüphanesi tensorflow yardımı ile sisteme tanıtıldı.Daha sonra tıpkı ANN modelde yapıldığı gibi veri seti tanıtıldı,veri setinin bir kopyası alındı,boş veriler veri setinden atıldı,bağımlı ve bağımsız değişkenler belirlenerek "x" ve "y" ye

atandı ardından bu değişkenler eğitim ve test seti olarak belirli oranlarda parçalandı. Bğımsız eğitim ve bağımlı eğitim verileri kullanılarak bir KNN modeli oluşturuldu ve modelin başarı oranı maksimum olarak gözlemlendi. Daha sonra k katlı cross-validation ve GridSearch algoritmalrı yardımı ile en iyi komşu parametresi belirlenerek bir dönüştürülmüş model tasarlandı ve bu dönüştürülmüş modelin başarı oranı aynı şekilde maksimum olarak gözlemlendi.

In [240]:

```
from sklearn.neighbors import KNeighborsRegressor
```

In [243]:

```
from sklearn.neighbors import KNeighborsClassifier
```

In [45]:

```
par = pd.read_csv('veribilimi.csv')
df = par.copy()
df = df.dropna()
y=df["status"]
x=df.drop(['status'],axis=1)
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.25,random_state=4
```

In [249]:

```
knn = KNeighborsClassifier()
knn_model = knn.fit(X_train, y_train)
knn_model
```

Out[249]:

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
           metric_params=None, n_jobs=1, n_neighbors=5, p=2,
           weights='uniform')
```

In [250]:

```
y_pred = knn_model.predict(X_test)3
```

In [251]:

```
accuracy_score(y_test, y_pred)
```

Out[251]:

```
1.0
```

In [253]:

```
y_pred = knn_model.predict(X_test)

print("test hatası:" , np.sqrt(mean_squared_error(y_test, y_pred)))

RMSE = []
#cross validation yapmadan hatalara bir bakalim
for k in range(10):
    k = k+1
    knn_model = KNeighborsRegressor(n_neighbors = k).fit(X_train, y_train)
    y_pred = knn_model.predict(X_train)
    rmse = np.sqrt(mean_squared_error(y_train,y_pred))
    RMSE.append(rmse)
    print("k =" , k , "için RMSE değeri: ", rmse)
```

```
test hatası: 0.0
k = 1 için RMSE değeri:  0.0
k = 2 için RMSE değeri:  0.0
k = 3 için RMSE değeri:  0.03901371573204352
k = 4 için RMSE değeri:  0.05852057359806528
k = 5 için RMSE değeri:  0.04965635331614208
k = 6 için RMSE değeri:  0.04973292445386293
k = 7 için RMSE değeri:  0.04423739552038088
k = 8 için RMSE değeri:  0.045093130418711276
k = 9 için RMSE değeri:  0.06029971929844841
k = 10 için RMSE değeri:  0.08066504394760157
```

In [180]:

```
#GridSearchCV ile optimum k sayisinin belirlenmesi
knn_params = {'n_neighbors': np.arange(1,30,1)}
knn = KNeighborsRegressor()
knn_cv_model = GridSearchCV(knn, knn_params, cv = 10)
knn_cv_model.fit(X_train, y_train)
```

Out[180]:

```
GridSearchCV(cv=10, error_score='raise',
       estimator=KNeighborsRegressor(algorithm='auto', leaf_size=30, metri
c='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=5, p=2,
          weights='uniform'),
       fit_params=None, iid=True, n_jobs=1,
       param_grid={'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,
9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29])},
       pre_dispatch='2*n_jobs', refit=True, return_train_score='warn',
       scoring=None, verbose=0)
```

In [181]:

```
knn_cv_model.best_params_["n_neighbors"]
```

Out[181]:

```
1
```

In [182]:

```
RMSE = []
RMSE_CV = []
for k in range(10):
    k = k+1
    knn_model = KNeighborsRegressor(n_neighbors = k).fit(X_train, y_train)
    y_pred = knn_model.predict(X_train)
    rmse = np.sqrt(mean_squared_error(y_train,y_pred))
    rmse_cv = np.sqrt(-1*cross_val_score(knn_model, X_train, y_train, cv=10,
                                         scoring = "neg_mean_squared_error").mean())
    #cross validation olmadan hatalar
    RMSE.append(rmse)
    #cross validation kullanilarak alindan hatalar
    RMSE_CV.append(rmse_cv)
    print("k =" , k , "için RMSE değeri: ", rmse, "RMSE_CV değeri: ", rmse_cv )
```

```
k = 1 için RMSE değeri:  0.0 RMSE_CV değeri:  -0.0
k = 2 için RMSE değeri:  0.0 RMSE_CV değeri:  0.058756965139300316
k = 3 için RMSE değeri:  0.03901371573204352 RMSE_CV değeri:  0.0783426201
8573375
k = 4 için RMSE değeri:  0.05852057359806528 RMSE_CV değeri:  0.0622016689
4101491
k = 5 için RMSE değeri:  0.04965635331614208 RMSE_CV değeri:  0.0620291790
0165251
k = 6 için RMSE değeri:  0.04973292445386293 RMSE_CV değeri:  0.0570435645
27194945
k = 7 için RMSE değeri:  0.04423739552038088 RMSE_CV değeri:  0.0677989367
2606979
k = 8 için RMSE değeri:  0.045093130418711276 RMSE_CV değeri:  0.082104028
08980463
k = 9 için RMSE değeri:  0.06029971929844841 RMSE_CV değeri:  0.0955506335
6340458
k = 10 için RMSE değeri:  0.08066504394760157 RMSE_CV değeri:  0.105062339
77070585
```

In [183]:

```
knn_tuned = KNeighborsRegressor(n_neighbors = knn_cv_model.best_params_["n_neighbors"])
knn_tuned.fit(X_train, y_train)
```

Out[183]:

```
KNeighborsRegressor(algorithm='auto', leaf_size=30, metric='minkowski',
          metric_params=None, n_jobs=1, n_neighbors=1, p=2,
          weights='uniform')
```

In [184]:

```
np.sqrt(mean_squared_error(y_test, knn_tuned.predict(X_test)))
```

Out[184]:

```
0.0
```

4-) RANDOM FOREST MODELİ

In [195]:

```
from sklearn.ensemble import RandomForestRegressor
```

In [198]:

```
par = pd.read_csv('veribilimi.csv')
df = par.copy()
df = df.dropna()
y=df["status"]
x=df.drop(["status","name"],axis=1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25,random_state=4
```

In [199]:

```
rf_model = RandomForestRegressor(random_state = 42)
rf_model.fit(X_train, y_train)
```

Out[199]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=None,
           max_features='auto', 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, n_estimators=10, n_jobs=1,
           oob_score=False, random_state=42, verbose=0, warm_start=False)
```

In [200]:

```
# Tahmin
rf_model.predict(X_test)[0:5]
```

Out[200]:

```
array([1., 1., 1., 1., 1.])
```

In [201]:

```
y_pred = rf_model.predict(X_test)
```

In [202]:

```
np.sqrt(mean_squared_error(y_test, y_pred))
```

Out[202]:

```
0.0
```

In [203]:

```
#model tunning
rf_params = {'max_depth': list(range(1,10)),
             'max_features': [3,5,10,15],
             'n_estimators' : [100, 200, 500, 1000, 2000]}
rf_model = RandomForestRegressor(random_state = 42)
rf_cv_model = GridSearchCV(rf_model, rf_params,cv = 10,n_jobs = -1)
```

In [204]:

```
rf_cv_model.fit(X_train, y_train)
rf_cv_model.best_params_
```

Out[204]:

```
{'max_depth': 6, 'max_features': 15, 'n_estimators': 500}
```

In [205]:

```
rf_tuned = RandomForestRegressor(max_depth  = 6, max_features = 15, n_estimators =500)
```

In [206]:

```
rf_tuned.fit(X_train, y_train)
```

Out[206]:

```
RandomForestRegressor(bootstrap=True, criterion='mse', max_depth=6,
           max_features=15, 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,
           n_estimators=500, n_jobs=1, oob_score=False, random_state=None,
           verbose=0, warm_start=False)
```

In [207]:

```
y_pred = rf_tuned.predict(X_test)
np.sqrt(mean_squared_error(y_test, y_pred))
```

Out[207]:

```
0.023280016831468776
```

SONUÇ

Kullanmış olduğum maine öğrenmesi algortimalarındaki başarı oranı şahsım tarafınca analiz edilerek en başaralı modellemeler şu algoritmalarda elde edilmiştir:

K En Yakın Komşu Algoritması
Yapay Sinir Ağları

Bu iki algoritmada oldukça iyi sonuçlar vermekte olup,aynı zamanda hızlı ve doğru sonuçlar üretmektedir. K en yakın komşu algoritması modelimde herhangi bir iyileştirme yapılmadığı halde başarı oranı maksimum olmuştur. Yapay sinir ağları modelimde ise iyileştirme yapılan modelin başarı oranı maksimum noktaya ulaşmıştır.

In [ ]:

```
```