Sınıflandırma Modelleri

In [40]:
```
import numpy as np
import pandas as pd
import statsmodels.api as sm
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import scale, StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV,cross_val_score
from sklearn.metrics import confusion_matrix, accuracy_score, mean_squared_error,r2_score, roc_auc_score, roc_c
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.neural_network import MLPClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import GradientBoostingClassifier
```

In [4]:
```
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
warnings.filterwarnings("ignore", category=FutureWarning)
```

In [8]:
```
df = pd.read_csv("./diabetes.csv")
```

In [9]:
```
df.head()
```

Out[9]:

	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

Lojistik Regresyon(Logistic Regression)

Model & Tahmin

In [11]:
```
df["Outcome"].value_counts()
```

Out[11]:
```
0    500
1    268
Name: Outcome, dtype: int64
```

In [14]:
```
df.describe().T
```

Out[14]:

	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

In [15]:
```
y = df["Outcome"]
X = df.drop(["Outcome"], axis=1)
```

In [17]:
```
y.head()
```

Out[17]:
```
0    1
1    0
2    1
3    0
4    1
Name: Outcome, dtype: int64
```

In [18]:
```
X.head()
```

Out[18]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	6	148	72	35	0	33.6	0.627	50
1	1	85	66	29	0	26.6	0.351	31
2	8	183	64	0	0	23.3	0.672	32
3	1	89	66	23	94	28.1	0.167	21
4	0	137	40	35	168	43.1	2.288	33

In [20]:
```
loj_model = LogisticRegression(solver = "liblinear").fit(X,y)
```

In [21]:
```
loj_model.intercept_
```

Out[21]:
```
array([-5.88679617])
```

In [22]:
```
loj_model.coef_
```

Out[22]:
```
array([[ 1.16994476e-01,  2.83733435e-02, -1.68981359e-02,
         7.55145090e-04, -6.41407258e-04,  5.97201268e-02,
         6.76128123e-01,  7.23989971e-03]])
```

In [23]:
```
loj_model.predict(X)[0:10]
```

Out[23]:
```
array([1, 0, 1, 0, 1, 0, 0, 1, 1, 0], dtype=int64)
```

In [24]:
```
y[0:10]
```

Out[24]:
```
0    1
1    0
2    1
3    0
4    1
5    0
6    1
7    0
8    1
9    1
Name: Outcome, dtype: int64
```

In [25]:
```
y_pred = loj_model.predict(X)
```

In [26]:
```
confusion_matrix(y,y_pred)
```

Out[26]:
```
array([[448,  52],
       [121, 147]], dtype=int64)
```

In [27]:
```
accuracy_score(y,y_pred)
```

Out[27]:
```
0.7747395833333334
```

In [28]:
```
print(classification_report(y,y_pred))
```

```
              precision    recall  f1-score   support

           0       0.79      0.90      0.84       500
           1       0.74      0.55      0.63       268

    accuracy                           0.77       768
   macro avg       0.76      0.72      0.73       768
weighted avg       0.77      0.77      0.77       768
```

In [30]:
```
loj_model.predict_proba(X)[0:10] #ondalık olarak şekilde tahmin fonksiyonları
```

Out[30]:
```
array([[0.3505852 , 0.6494148 ],
       [0.91692518, 0.08307482],
       [0.22489628, 0.77510372],
       [0.92127453, 0.07872547],
       [0.16759435, 0.83240565],
       [0.79886109, 0.20113891],
       [0.8800353 , 0.1199647 ],
       [0.27795677, 0.72204323],
       [0.32053464, 0.67946536],
       [0.92264521, 0.07735479]])
```

In [31]:
```
# roc eğrisi kontrol yöntemi
logit_roc_auc = roc_auc_score(y, loj_model.predict(X))
fpr, tpr, thresholds = roc_curve(y, loj_model.predict_proba(X)[:,1])
plt.figure()
plt.plot(fpr,tpr,label='AUC (area=%0.2f)'% logit_roc_auc)
plt.plot([0,1],[0,1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver operating characteristic')
plt.legend(loc="lower right")
plt.savefig('Log_ROC')
plt.show()
```

Model Tuning(Model Doğrulama)

In [35]:
```
X_train, X_test, y_train, y_test = train_test_split(
                                                    X,
                                                    y,
                                                    test_size=0.30,
                                                    random_state=42)
```

In [36]:
```
loj_model = LogisticRegression(solver="liblinear").fit(X_train,y_train)
```

In [39]:
```
y_pred = loj_model.predict(X_test)
```

In [40]:
```
print(accuracy_score(y_test,y_pred))
```

```
0.7445887445887446
```

In [41]:
```
cross_val_score(loj_model, X_test,y_test,cv=10).mean()
```

Out[41]:
```
0.7704710144927536
```

K-En Yakın Komşu(KNN)

In [53]:
```
y = df["Outcome"]
X = df.drop(["Outcome"], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(
                                                    X,y,
                                                    test_size = 0.30,
                                                    random_state=42)
```

In [51]:
```
y = df["Outcome"]
X = df.drop(["Outcome"], axis = 1)
X_train, X_test, y_train, y_test = train_test_split(
                                                    X,y,
                                                    test_size = 0.30,
                                                    random_state=42)
```

Model & Tahmin

In [54]:
```
knn_model = KNeighborsClassifier().fit(X_train,y_train)
```

In [56]:
```
y_pred = knn_model.predict(X_test)
accuracy_score(y_test,y_pred)
```

Out[56]:
```
0.6883116883116883
```

Model Tuning(Model Doğrulama)

In [57]:
```
knn = KNeighborsClassifier()
```

In [58]:
```
knn_params = {"n_neighbors":np.arange(1,50)}
```

In [59]:
```
knn_cv_model = GridSearchCV(knn,knn_params,cv=10).fit(X_train,y_train)
```

In [62]:
```
knn_cv_model.best_params_
```

Out[62]:
```
{'n_neighbors': 11}
```

In [27]:
```
#final modeli
knn_tuned = KNeighborsClassifier(n_neighbors= 11).fit(X_train,y_train)
```

In [27]:
```
y_pred = knn_tuned.predict(X_test)
accuracy_score(y_test,y_pred)
```

Out[27]:
```
0.7142857142857143
```

In [27]:
```
knn_tuned.score(X_test,y_test) #daha pratik hesaplama
```

Out[27]:
```
0.7142857142857143
```

Destek Vektör Makineleri(SVM)

In [70]:
```
df.head()
```

Out[70]:

	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

In [71]:
```
X_train, X_test, y_train, y_test = train_test_split(
                                                    X,
                                                    y,
                                                    test_size=0.30,
                                                    random_state=42)
```

Model & Tahmin

In [73]:
```
svm_model = SVC(kernel="linear").fit(X_train,y_train)
```

In [75]:
```
from sklearn import set_config
set_config(print_changed_only=False)
```

In [ ]:

In [79]:
```
svm_model
```

Out[79]:
```
SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='linear',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

In [79]:
```
y_pred = svm_model.predict(X_test)
```

In [80]:
```
accuracy_score(y_test,y_pred)
```

Out[80]:
```
0.7445887445887446
```

Model Tuning

In [82]:
```
svm = SVC()
```

In [83]:
```
svm_params = {"C":np.arange(1,10),
              "kernel":["linear","rbf"]}
```

In [85]:
```
svm_cv_model = GridSearchCV(svm,svm_params,cv=10,n_jobs=-1,verbose=2).fit(X_train,y_train)
```

```
Fitting 10 folds for each of 18 candidates, totalling 180 fits
```

In [86]:
```
svm_cv_model.best_params_
```

Out[86]:
```
{'C': 5, 'kernel': 'linear'}
```

In [87]:
```
svm_cv_model.best_score_
```

Out[87]:
```
0.7765199161425576
```

In [89]:
```
#final modeli
svm_tuned = SVC(C = 5,kernel="linear").fit(X_train,y_train)
```

In [90]:
```
y_pred = svm_tuned.predict(X_test)
accuracy_score(y_test,y_pred)
```

Out[90]:
```
0.7532467532467533
```

Yapay Sinir Ağları(Çok Katmanlı Algılayıcılar)

In [90]:
```
df.head()
```

Out[90]:

	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

In [91]:
```
X_train, X_test, y_train, y_test = train_test_split(
                                                    X,
                                                    y,
                                                    test_size=0.30,
                                                    random_state=42)
```

In [92]:
```
scaler = StandardScaler()
```

In [93]:
```
scaler.fit(X_train)
X_train = scaler.transform(X_train)
```

In [94]:
```
scaler.fit(X_test)
X_test = scaler.transform(X_test)
```

Model & Tahmin

In [95]:
```
mlpc_model = MLPClassifier().fit(X_train,y_train)
```

In [96]:
```
mlpc_model.coefs_
```

Out[96]:
```
[array([[ 5.42355354e-02, -5.25820014e-02,  1.9774272
         -2.41522845e-01,  1.94805718e-01, -2.3297733
[illegible]
         -1.62126033e-01,  8.31314923e-01, -1.6136617
         -1.61795934e-01]]),
 array([[-0.26770184],
        [-0.14297492],
        [-0.07260625],
        [-0.15933408],
        [-0.16886065],
        [-0.16420581],
        [-0.07700961],
        [ 0.01356935],
        [-0.07686238],
        [-0.07264475],
        [ 0.0378327 ],
        [-0.08895292],
        [-0.09513202],
        [-0.17260469],
        [-0.12738644],
        [-0.12418463],
        [-0.16132337],
        [-0.1026658 ],
        [-0.14511881],
        [-0.00810054],
        [-0.16855502],
        [ 0.03271507],
        [-0.00880956],
        [-0.05042421],
        [-0.07640099],
        [-0.0639059 ],
        [-0.13707362],
        [-0.00907015],
        [ 0.02830125],
        [ 0.21325384],
        [ 0.20409576],
        [-0.14906526],
        [-0.21498059],
        [-0.04350504],
        [ 0.08456575],
        [-0.04924399],
        [ 0.10255272],
        [ 0.15937255],
        [ 0.14231882],
        [ 0.1987358 ],
        [-0.2337773 ],
        [ 0.13068615],
        [-0.15611906],
        [ 0.17138895],
        [-0.01882415],
        [-0.00819618],
        [-0.24519773],
        [ 0.09375999],
        [ 0.22730303],
        [-0.07349724],
        [-0.0103241 ],
        [ 0.1332542 ],
        [ 0.15557379],
        [ 0.11820532],
        [ 0.08160935],
        [ 0.05416512],
        [ 0.18759014],
        [ 0.03896452],
        [ 0.2282426 ],
        [ 0.01045031],
        [ 0.12430945],
        [ 0.08893485],
        [-0.12326561],
        [ 0.11459842],
        [-0.08729522],
        [-0.07519269],
        [-0.29606837],
        [-0.22597746],
        [ 0.21598228],
        [ 0.01560926],
        [ 0.01978656],
        [ 0.02482352],
        [-0.05432876],
        [-0.21256489],
        [ 0.02841869],
        [ 0.201019  ],
        [-0.0305588 ],
        [ 0.0030465 ],
        [ 0.14268175],
        [ 0.1313516 ],
        [ 0.03653729],
        [ 0.01550928],
        [ 0.09208959],
        [ 0.15819998],
        [-0.2272455 ],
        [ 0.04841202],
        [ 0.17015557],
        [ 0.0709495 ],
        [-0.16772744],
        [ 0.05886519],
        [-0.18189664],
        [ 0.1206471 ],
        [ 0.10204795],
        [-0.01494255],
        [-0.23696368],
        [ 0.02051311],
        [ 0.20455949],
        [ 0.09233236],
        [ 0.00742044],
        [-0.18363907]])]
```

In [97]:
```
#mlpc_model
```

```
Type:           MLPClassifier
String form:
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2 <...>?              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
File:           c:\users\lenovo\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py
Docstring:
Multi-layer Perceptron classifier.

This model optimizes the log-loss function using LBFGS or stochastic
gradient descent.

.. versionadded:: 0.18

Parameters
----------
hidden_layer_sizes : tuple, length = n_layers - 2, default=(100,)
    The ith element represents the number of neurons in the ith
    hidden layer.

activation : {'identity', 'logistic', 'tanh', 'relu'}, default='relu'
    Activation function for the hidden layer.

    - 'identity', no-op activation, useful to implement linear bottleneck,
      returns f(x) = x

    - 'logistic', the logistic sigmoid function,
      returns f(x) = 1 / (1 + exp(-x)).

    - 'tanh', the hyperbolic tan function,
      returns f(x) = tanh(x).

    - 'relu', the rectified linear unit function,
      returns f(x) = max(0, x)

solver : {'lbfgs', 'sgd', 'adam'}, default='adam'
    The solver for weight optimization.

    - 'lbfgs' is an optimizer in the family of quasi-Newton methods.

    - 'sgd' refers to stochastic gradient descent.

    - 'adam' refers to a stochastic gradient-based optimizer proposed
      by Kingma, Diederik, and Jimmy Ba

    Note: The default solver 'adam' works pretty well on relatively
    large datasets (with thousands of training samples or more) in terms of
    both training time and validation score.
    For small datasets, however, 'lbfgs' can converge faster and perform
    better.

alpha : float, default=0.0001
    L2 penalty (regularization term) parameter.

batch_size : int, default='auto'
    Size of minibatches for stochastic optimizers.
    If the solver is 'lbfgs', the classifier will not use minibatch.
    When set to "auto", `batch_size=min(200, n_samples)`

learning_rate : {'constant', 'invscaling', 'adaptive'}, default='constant'
    Learning rate schedule for weight updates.

    - 'constant' is a constant learning rate given by
      'learning_rate_init'.

    - 'invscaling' gradually decreases the learning rate at each
      time step 't' using an inverse scaling exponent of 'power_t'.
      effective_learning_rate = learning_rate_init / pow(t, power_t)

    - 'adaptive' keeps the learning rate constant to
      'learning_rate_init' as long as training loss keeps decreasing.
      Each time two consecutive epochs fail to decrease training loss by at
      least tol, or fail to increase validation score by at least tol if
      'early_stopping' is on, the current learning rate is divided by 5.

    Only used when ``solver='sgd'``.

learning_rate_init : double, default=0.001
    The initial learning rate used. It controls the step-size
    in updating the weights. Only used when solver='sgd' or 'adam'.

power_t : double, default=0.5
    The exponent for inverse scaling learning rate.
    It is used in updating effective learning rate when the learning_rate
    is set to 'invscaling'. Only used when solver='sgd'.

max_iter : int, default=200
    Maximum number of iterations. The solver iterates until convergence
    (determined by 'tol') or this number of iterations. For stochastic
    solvers ('sgd', 'adam'), note that this determines the number of epochs
    (how many times each data point will be used), not the number of
    gradient steps.

shuffle : bool, default=True
    Whether to shuffle samples in each iteration. Only used when
    solver='sgd' or 'adam'.

random_state : int, RandomState instance, default=None
    Determines random number generation for weights and bias
    initialization, train-test split if early stopping is used, and batch
    sampling when solver='sgd' or 'adam'.
    Pass an int for reproducible results across multiple function calls.
    See :term:`Glossary <random_state>`.

tol : float, default=1e-4
    Tolerance for the optimization. When the loss or score is not improving
    by at least ``tol`` for ``n_iter_no_change`` consecutive iterations,
    unless ``learning_rate`` is set to 'adaptive', convergence is
    considered to be reached and training stops.

verbose : bool, default=False
    Whether to print progress messages to stdout.

warm_start : bool, default=False
    When set to True, reuse the solution of the previous
    call to fit as initialization, otherwise, just erase the
    previous solution. See :term:`the Glossary <warm_start>`.

momentum : float, default=0.9
    Momentum for gradient descent update. Should be between 0 and 1. Only
    used when solver='sgd'.

nesterovs_momentum : bool, default=True
    Whether to use Nesterov's momentum. Only used when solver='sgd' and
    momentum > 0.

early_stopping : bool, default=False
    Whether to use early stopping to terminate training when validation
    score is not improving. If set to true, it will automatically set
    aside 10% of training data as validation and terminate training when
    validation score is not improving by at least tol for
    ``n_iter_no_change`` consecutive epochs. The split is stratified,
    except in a multilabel setting.
    Only effective when solver='sgd' or 'adam'

validation_fraction : float, default=0.1
    The proportion of training data to set aside as validation set for
    early stopping. Must be between 0 and 1.
    Only used if early_stopping is True

beta_1 : float, default=0.9
    Exponential decay rate for estimates of first moment vector in adam,
    should be in [0, 1). Only used when solver='adam'

beta_2 : float, default=0.999
    Exponential decay rate for estimates of second moment vector in adam,
    should be in [0, 1). Only used when solver='adam'

epsilon : float, default=1e-8
    Value for numerical stability in adam. Only used when solver='adam'

n_iter_no_change : int, default=10
    Maximum number of epochs to not meet ``tol`` improvement.
    Only effective when solver='sgd' or 'adam'

    .. versionadded:: 0.20

max_fun : int, default=15000
    Only used when solver='lbfgs'. Maximum number of loss function calls.
    The solver iterates until convergence (determined by 'tol'), number
    of iterations reaches max_iter, or this number of loss function calls.
    Note that number of loss function calls will be greater than or equal
    to the number of iterations for the `MLPClassifier`.

    .. versionadded:: 0.22

Attributes
----------
classes_ : ndarray or list of ndarray of shape (n_classes,)
    Class labels for each output.

loss_ : float
    The current loss computed with the loss function.

best_loss_ : float
    The minimum loss reached by the solver throughout fitting.

loss_curve_ : list of shape (`n_iter_`,)
    The ith element in the list represents the loss at the ith iteration.

t_ : int
    The number of training samples seen by the solver during fitting.

coefs_ : list of shape (n_layers - 1,)
    The ith element in the list represents the weight matrix corresponding
    to layer i.

intercepts_ : list of shape (n_layers - 1,)
    The ith element in the list represents the bias vector corresponding to
    layer i + 1.

n_iter_ : int
    The number of iterations the solver has ran.

n_layers_ : int
    Number of layers.

n_outputs_ : int
    Number of outputs.

out_activation_ : str
    Name of the output activation function.

Examples
--------
>>> from sklearn.neural_network import MLPClassifier
>>> from sklearn.datasets import make_classification
>>> from sklearn.model_selection import train_test_split
>>> X, y = make_classification(n_samples=100, random_state=1)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y,
...                                                     random_state=1)
>>> clf = MLPClassifier(random_state=1, max_iter=300).fit(X_train, y_train)
>>> clf.predict_proba(X_test[:1])
array([[0.038..., 0.961...]])
>>> clf.predict(X_test[:5, :])
array([1, 0, 1, 0, 1])
>>> clf.score(X_test, y_test)
0.8...

Notes
-----
MLPClassifier trains iteratively since at each time step
the partial derivatives of the loss function with respect to the model
parameters are computed to update the parameters.

It can also have a regularization term added to the loss function
that shrinks model parameters to prevent overfitting.

This implementation works with data represented as dense numpy arrays or
sparse scipy arrays of floating point values.

References
----------
Hinton, Geoffrey E.
    "Connectionist learning procedures." Artificial intelligence 40.1
    (1989): 185-234.

Glorot, Xavier, and Yoshua Bengio. "Understanding the difficulty of
    training deep feedforward neural networks." International Conference
    on Artificial Intelligence and Statistics. 2010.

He, Kaiming, et al. "Delving deep into rectifiers: Surpassing human-level
    performance on imagenet classification." arXiv preprint
    arXiv:1502.01852 (2015).

Kingma, Diederik, and Jimmy Ba. "Adam: A method for stochastic
    optimization." arXiv preprint arXiv:1412.6980 (2014).
```

In [98]:
```
y_pred = mlpc_model.predict(X_test)
```

In [99]:
```
accuracy_score(y_test,y_pred)
```

Out[99]: 0.7056277056277056

Model Tuning

In [100]:
```
mlpc = MLPClassifier(solver="lbfgs",activation="logistic",max_iter=500)
```

In [101]:
```
mlpc_params = {"alpha":[1,5,0.0001],
               "hidden_layer_sizes":[(100,100),(3,5)]}
```

In [102]:
```
mlpc_cv_model = GridSearchCV(mlpc,mlpc_params,cv=10,n_jobs=-1,verbose=2).fit(X_train,y_train)
```

Fitting 10 folds for each of 6 candidates, totalling 60 fits

In [134]:
```
mlpc_cv_model.best_score_
```

Out[134]: 0.7747300469484692

In [135]:
```
mlpc_cv_model.best_params_
```

Out[135]: {'alpha': 1, 'hidden_layer_sizes': (3, 5)}

In [276]:
```
mlpc_tuned = MLPClassifier(solver="lbfgs",activation="logistic",alpha=1,hidden_layer_sizes=(3,5)).fit(X_train,
```

In [277]:
```
mlpc_tuned.score(X_test,y_test)
```

Out[277]: 0.7532467532467533

CART (Classification and Regression Tree)

In [136]:
```
df.head()
```

Out[136]:

	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

In [138]:
```
X_train, X_test, y_train, y_test = train_test_split(
                                                    X,
                                                    y,
                                                    test_size=0.30,
                                                    random_state=42)
```

Model & Tahmin

In [140]:
```
cart_model = DecisionTreeClassifier().fit(X_train,y_train)
```

In [141]:
```
cart_model
```

Out[141]:
```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       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, random_state=None,
                       splitter='best')
```

In [142]:
```
cart_model.score(X_test,y_test)
```

Out[142]: 0.70995670995671

Model Tuning

In [144]:
```
cart = DecisionTreeClassifier()
```

In [148]:
```
cart_params = {"max_depth":[1,3,5,8,10],
               "min_samples_split":[2,3,5,10,20]}
```

In [149]:
```
cart_cv_model = GridSearchCV(cart,cart_params,cv=10,n_jobs=-1,verbose=2).fit(X_train,y_train)
```

Fitting 10 folds for each of 25 candidates, totalling 250 fits

In [150]:
```
cart_cv_model.best_score_
```

Out[150]: 0.7634870701977638

In [151]:
```
cart_cv_model.best_params_
```

Out[151]: {'max_depth': 5, 'min_samples_split': 20}

In [278]:
```
#final model
cart_tuned = DecisionTreeClassifier(max_depth=5, min_samples_split=20).fit(X_train,y_train)
```

In [279]:
```
y_pred =cart_tuned.predict(X_test)
```

In [280]:
```
accuracy_score(y_test,y_pred)
```

Out[280]: 0.7532467532467533

Random Forest

In [137]:
```
df.head()
```

Out[137]:

	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

In [139]:
```
X_train, X_test, y_train, y_test = train_test_split(
                                                    X,
                                                    y,
                                                    test_size=0.30,
                                                    random_state=42)
```

Model & Tahmin

In [159]:
```
rf_model = RandomForestClassifier().fit(X_train,y_train)
```

In [160]:
```
rf_model
```

Out[160]:
```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='gini', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=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=100,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [161]:
```
y_pred = rf_model.predict(X_test)
```

In [162]:
```
accuracy_score(y_test,y_pred)
```

Out[162]: 0.7359307359307359

Model Tuning

In [163]:
```
rf = RandomForestClassifier()
```

In [164]:
```
rf_params = {"n_estimators":[100,200,500,1000],
             "max_features":[3,5,7],
             "min_samples_split":[2,5,10,20]}
```

In [165]:
```
rf_cv_model = GridSearchCV(rf,rf_params,cv=10,n_jobs=-1,verbose=2).fit(X_train,y_train)
```

Fitting 10 folds for each of 64 candidates, totalling 640 fits

In [166]:
```
rf_cv_model.best_score_
```

Out[166]: 0.7875960866526904

In [167]:
```
rf_cv_model.best_params_
```

Out[167]: {'max_features': 8, 'min_samples_split': 5, 'n_estimators': 1000}

In [281]:
```
rf_tuned = RandomForestClassifier(max_features=8,min_samples_split=5,n_estimators=1000).fit(X_train,y_train)
```

In [282]:
```
y_pred = rf_tuned.predict(X_test)
```

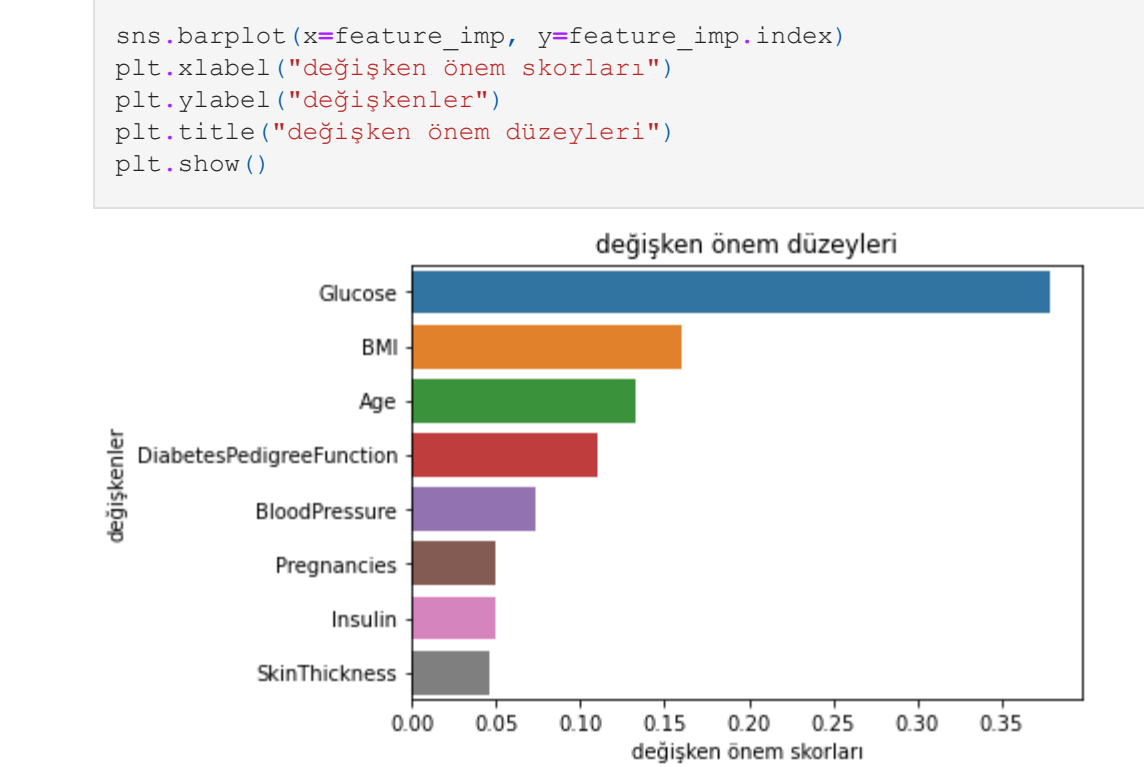
In [283]:
```
accuracy_score(y_test,y_pred)
```

Out[283]: 0.7619047619047619

In [172]:
```
#değişken önem düzeyi
```

In [173]:
```
feature_imp = pd.Series(rf_tuned.feature_importances_,
                        index=X_train.columns).sort_values(ascending=False)

sns.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel("değişken önem skorları")
plt.ylabel("değişkenler")
plt.title("değişken önem düzeyleri")
plt.show()
```



Gradient Boosting Machines

Model & Tahmin

In [174]:
```
gbm_model = GradientBoostingClassifier().fit(X_train,y_train)
```

In [175]:
```
gbm_model.score(X_test,y_test)
```

Out[175]: 0.7445887445887446

Model Tuning

In [176]:
```
gbm = GradientBoostingClassifier()
```

In [177]:
```
gbm_params = {"learning_rate":[0.1,0.01,0.001,0.05,0.001],
              "n_estimators":[100,300,500,1000],
              "max_depth":[2,3,5,8]}
```

In [178]:
```
gbm_cv_model = GridSearchCV(gbm,gbm_params,cv=10,n_jobs=-1).fit(X_train,y_train)
```

In [179]:
```
gbm_cv_model.best_params_
```

Out[179]: {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators': 300}

In [284]:
```
#final model
gbm_tuned = GradientBoostingClassifier(learning_rate= 0.01,n_estimators=300,max_depth=5).fit(X_train,y_train)
```

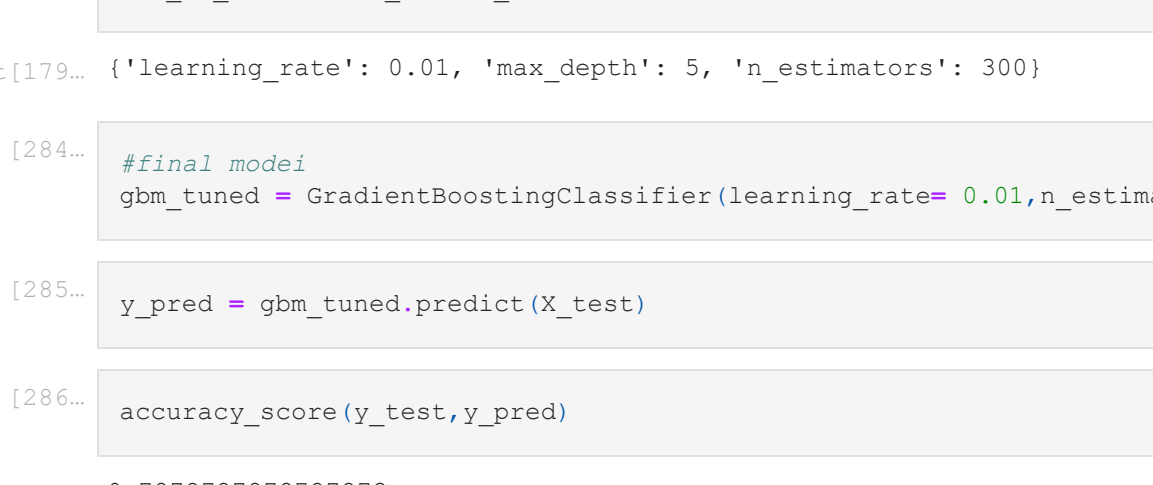
In [285]:
```
y_pred = gbm_tuned.predict(X_test)
```

In [286]:
```
accuracy_score(y_test,y_pred)
```

Out[286]: 0.7272727272727273

In [182]:
```
feature_imp = pd.Series(gbm_tuned.feature_importances_,
                        index=X_train.columns).sort_values(ascending=False)

sns.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel("değişken önem skorları")
plt.ylabel("değişkenler")
plt.title("değişken önem düzeyleri")
plt.show()
```



XGBoost

Model & Tahmin

In [191]:
```
!pip install xgboost
```

```
Requirement already satisfied: xgboost in c:\users\lenovo\anaconda3\lib\site-packages (1.5.0)
Requirement already satisfied: scipy in c:\users\lenovo\anaconda3\lib\site-packages (from xgboost) (1.6.2)
Requirement already satisfied: numpy in c:\users\lenovo\anaconda3\lib\site-packages (from xgboost) (1.20.1)
```

In [192]:
```
from xgboost import XGBClassifier
```

In [197]:
```
xgb_model = XGBClassifier().fit(X_train,y_train)
```

```
[00:06:26] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

In [198]:
```
xgb_model
```

Out[198]:
```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, enable_categorical=False,
              gamma=0, gpu_id=-1, importance_type=None,
              interaction_constraints='', learning_rate=0.300000012,
              max_delta_step=0, max_depth=6, min_child_weight=1, missing=nan,
              monotone_constraints='()', n_estimators=100, n_jobs=4,
              num_parallel_tree=1, objective='binary:logistic',
              predictor='auto', random_state=0, reg_alpha=0, reg_lambda=1,
              scale_pos_weight=1, subsample=1, tree_method='exact',
              use_label_encoder=True, validate_parameters=1, verbosity=None)
```

In [199]:
```
y_pred = xgb_model.predict(X_test)
```

In [200]:
```
accuracy_score(y_test,y_pred)
```

Out[200]: 0.7359307359307359

Model Tuning

In [201]:
```
xgb = XGBClassifier()
```

In [202]:
```
xgb_params = {"n_estimators":[100,500,1000],
              "subsample":[0.6,0.8,1],
              "max_depth":[3,5,7],
              "learning_rate":[0.1,0.001,0.01]}
```

In [203]:
```
xgb_cv_model = GridSearchCV(xgb,xgb_params,cv=5,n_jobs=-1,verbose=2).fit(X_train,y_train)
```

```
Fitting 5 folds for each of 81 candidates, totalling 405 fits
C:\Users\Lenovo\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in X
GBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as
integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[00:14:42] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

In [204]:
```
xgb_cv_model.best_params_
```

Out[204]: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 500, 'subsample': 1}

In [207]:
```
xgb_tuned = XGBClassifier(learning_rate= 0.01, max_depth = 3, n_estimators = 500, subsample= 1).fit(X_train,y_t
```

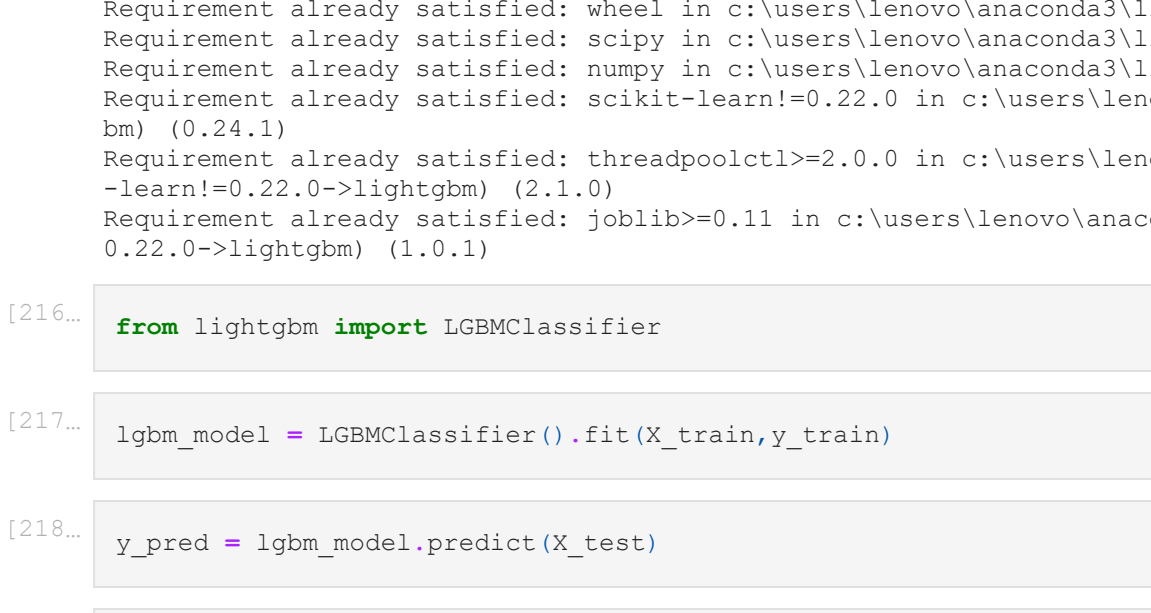
```
C:\Users\Lenovo\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in X
GBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as
integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[01:48:30] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

In [208]:
```
xgb_tuned.score(X_test,y_test)
```

Out[208]: 0.7359307359307359

In [210]:
```
feature_imp = pd.Series(xgb_tuned.feature_importances_,
                        index=X_train.columns).sort_values(ascending=False)

sns.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel("değişken önem skorları")
plt.ylabel("değişkenler")
plt.title("değişken önem düzeyleri")
plt.show()
```



Light GBM

Model & Tahmin

In [214]:
```
!pip install lightgbm
```

```
Requirement already satisfied: lightgbm in c:\users\lenovo\anaconda3\lib\site-packages (3.3.1)
Requirement already satisfied: wheel in c:\users\lenovo\anaconda3\lib\site-packages (from lightgbm) (0.36.2)
Requirement already satisfied: scipy in c:\users\lenovo\anaconda3\lib\site-packages (from lightgbm) (1.6.2)
Requirement already satisfied: numpy in c:\users\lenovo\anaconda3\lib\site-packages (from lightgbm) (1.20.1)
Requirement already satisfied: scikit-learn!=0.22.0 in c:\users\lenovo\anaconda3\lib\site-packages (from lightg
bm) (0.24.1)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\lenovo\anaconda3\lib\site-packages (from scikit
-learn!=0.22.0->lightgbm) (2.1.0)
Requirement already satisfied: joblib>=0.11 in c:\users\lenovo\anaconda3\lib\site-packages (from scikit-learn!=
0.22.0->lightgbm) (1.0.1)
```

In [216]:
```
from lightgbm import LGBMClassifier
```

In [217]:
```
lgbm_model = LGBMClassifier().fit(X_train,y_train)
```

In [218]:
```
y_pred = lgbm_model.predict(X_test)
```

In [219]:
```
accuracy_score(y_test,y_pred)
```

Out[220]: 0.7229437229437229

Model Tuning

In [221]:
```
lgbm = LGBMClassifier()
```

In [222]:
```
lgbm_params = {"n_estimators":[200,500,1000],
               "max_depth":[1,2,3,5],
               "learning_rate":[0.1,0.001,0.01]}
```

In [223]:
```
lgbm_cv_model = GridSearchCV(lgbm,lgbm_params,cv=10,n_jobs=-1,verbose=2).fit(X_train,y_train)
```

Fitting 10 folds for each of 36 candidates, totalling 360 fits

In [ ]:

In [224]:
```
lgbm_cv_model.best_params_
```

Out[224]: {'learning_rate': 0.01, 'max_depth': 1, 'n_estimators': 500}

In [225]:
```
lgbm_tuned = XGBClassifier(learning_rate= 0.01, max_depth =1 , n_estimators = 500).fit(X_train,y_train)
```

```
C:\Users\Lenovo\anaconda3\lib\site-packages\xgboost\sklearn.py:1224: UserWarning: The use of label encoder in X
GBClassifier is deprecated and will be removed in a future release. To remove this warning, do the following:
1) Pass option use_label_encoder=False when constructing XGBClassifier object; and 2) Encode your labels (y) as
integers starting with 0, i.e. 0, 1, 2, ..., [num_class - 1].
  warnings.warn(label_encoder_deprecation_msg, UserWarning)
[01:48:38] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.0/src/learner.cc:1115: Starting
in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'err
or' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.
```

In [226]:
```
lgbm_tuned.score(X_test,y_test)
```

Out[226]: 0.7575757575757576

In [229]:
```
feature_imp = pd.Series(lgbm_tuned.feature_importances_,
                        index=X_train.columns).sort_values(ascending=False)

sns.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel("değişken önem skorları")
plt.ylabel("değişkenler")
plt.title("değişken önem düzeyleri")
plt.show()
```

CatBoost

Model ve Tahmin

In [229]:
```
!pip install catboost
```

```
Requirement already satisfied: catboost in c:\users\lenovo\anaconda3\lib\site-packages (1.0.3)
Requirement already satisfied: matplotlib in c:\users\lenovo\anaconda3\lib\site-packages (from catboost) (3.3.
4)
Requirement already satisfied: six in c:\users\lenovo\anaconda3\lib\site-packages (from catboost) (1.15.0)
Requirement already satisfied: scipy in c:\users\lenovo\anaconda3\lib\site-packages (from catboost) (1.6.2)
Requirement already satisfied: graphviz in c:\users\lenovo\anaconda3\lib\site-packages (from catboost) (0.16)
Requirement already satisfied: numpy>=1.16.0 in c:\users\lenovo\anaconda3\lib\site-packages (from catboost) (1.
20.1)
Requirement already satisfied: plotly in c:\users\lenovo\anaconda3\lib\site-packages (from catboost) (5.3.1)
Requirement already satisfied: pandas>=0.24.0 in c:\users\lenovo\anaconda3\lib\site-packages (from catboost)
(1.2.4)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\lenovo\anaconda3\lib\site-packages (from pand
as>=0.24.0->catboost) (2.8.1)
Requirement already satisfied: pytz>=2017.3 in c:\users\lenovo\anaconda3\lib\site-packages (from pandas>=0.24.0
->catboost) (2021.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\lenovo\anaconda3\lib\site-packages (from matplotlib->c
atboost) (8.2.0)
Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.3 in c:\users\lenovo\anaconda3\lib\site-p
ackages (from matplotlib->catboost) (2.4.7)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\lenovo\anaconda3\lib\site-packages (from matplotli
b->catboost) (1.3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\lenovo\anaconda3\lib\site-packages (from matplotlib->ca
tboost) (0.10.0)
Requirement already satisfied: tenacity>=6.2.0 in c:\users\lenovo\anaconda3\lib\site-packages (from plotly->cat
boost) (8.0.1)
```

In [230]:
```
from catboost import CatBoostClassifier
```

In [231]:
```
catb_model = CatBoostClassifier().fit(X_train,y_train,verbose=False)
```

In [233]:
```
y_pred = catb_model.predict(X_test)
```

In [234]:
```
accuracy_score(y_test,y_pred)
```

Out[234]: 0.7402597402597403

Model Tuning

In [235]:
```
catb = CatBoostClassifier()
```

In [236]:
```
catb_params = {"iterations":[200,500,1000],
               "learning_rate":[0.01,0.03,0.1],
               "depth":[4,5,8]}
```

In [237]:
```
catb_cv_model = GridSearchCV(catb,catb_params,cv=10,n_jobs=-1,verbose=2).fit(X_train,y_train)
```

```
Fitting 10 folds for each of 27 candidates, totalling 270 fits
0:	learn: 0.6863161	total: 7.08ms	remaining: 1.41s
1:	learn: 0.6811157	total: 13.5ms	remaining: 1.33s
2:	learn: 0.6752970	total: 19.5ms	remaining: 1.28s
3:	learn: 0.6697432	total: 25.5ms	remaining: 1.25s
4:	learn: 0.6643796	total: 34.1ms	remaining: 1.33s
5:	learn: 0.6576656	total: 40.3ms	remaining: 1.3s
6:	learn: 0.6532870	total: 45.7ms	remaining: 1.26s
7:	learn: 0.6465564	total: 51.2ms	remaining: 1.23s
8:	learn: 0.6404452	total: 57.1ms	remaining: 1.21s
9:	learn: 0.6357725	total: 63.6ms	remaining: 1.21s
10:	learn: 0.6299942	total: 69.2ms	remaining: 1.19s
11:	learn: 0.6240480	total: 75.3ms	remaining: 1.18s
12:	learn: 0.6190330	total: 80.4ms	remaining: 1.16s
13:	learn: 0.6148715	total: 85.5ms	remaining: 1.14s
14:	learn: 0.6092626	total: 91.3ms	remaining: 1.13s
15:	learn: 0.6046764	total: 96.6ms	remaining: 1.11s
16:	learn: 0.5993471	total: 102ms	remaining: 1.1s
17:	learn: 0.5938521	total: 108ms	remaining: 1.09s
18:	learn: 0.5903240	total: 115ms	remaining: 1.1s
19:	learn: 0.5866024	total: 125ms	remaining: 1.12s
20:	learn: 0.5813390	total: 134ms	remaining: 1.14s
21:	learn: 0.5787381	total: 142ms	remaining: 1.15s
22:	learn: 0.5749016	total: 150ms	remaining: 1.15s
23:	learn: 0.5703199	total: 159ms	remaining: 1.17s
24:	learn: 0.5662368	total: 167ms	remaining: 1.17s
25:	learn: 0.5628682	total: 174ms	remaining: 1.2s
26:	learn: 0.5589732	total: 189ms	remaining: 1.21s
27:	learn: 0.5543910	total: 196ms	remaining: 1.23s
28:	learn: 0.5503864	total: 210ms	remaining: 1.24s
29:	learn: 0.5470574	total: 219ms	remaining: 1.24s
30:	learn: 0.5441267	total: 225ms	remaining: 1.23s
31:	learn: 0.5396280	total: 231ms	remaining: 1.21s
32:	learn: 0.5356312	total: 240ms	remaining: 1.21s
33:	learn: 0.5315397	total: 247ms	remaining: 1.21s
34:	learn: 0.5279947	total: 254ms	remaining: 1.21s
35:	learn: 0.5248380	total: 261ms	remaining: 1.19s
36:	learn: 0.5223885	total: 268ms	remaining: 1.18s
37:	learn: 0.5198269	total: 273ms	remaining: 1.16s
38:	learn: 0.5167931	total: 278ms	remaining: 1.15s
39:	learn: 0.5138833	total: 284ms	remaining: 1.14s
40:	learn: 0.5101434	total: 290ms	remaining: 1.13s
41:	learn: 0.5066102	total: 295ms	remaining: 1.11s
42:	learn: 0.5035974	total: 302ms	remaining: 1.1s
43:	learn: 0.5005898	total: 308ms	remaining: 1.09s
44:	learn: 0.4987003	total: 313ms	remaining: 1.08s
45:	learn: 0.4959251	total: 318ms	remaining: 1.07s
46:	learn: 0.4933997	total: 324ms	remaining: 1.05s
47:	learn: 0.4905975	total: 330ms	remaining: 1.04s
48:	learn: 0.4878359	total: 336ms	remaining: 1.03s
49:	learn: 0.4850755	total: 340ms	remaining: 1.02s
50:	learn: 0.4825507	total: 347ms	remaining: 1.01s
51:	learn: 0.4798298	total: 353ms	remaining: 1s
52:	learn: 0.4776108	total: 359ms	remaining: 994ms
53:	learn: 0.4754405	total: 367ms	remaining: 992ms
54:	learn: 0.4733501	total: 373ms	remaining: 984ms
55:	learn: 0.4706076	total: 380ms	remaining: 977ms
56:	learn: 0.4682500	total: 385ms	remaining: 967ms
57:	learn: 0.4656740	total: 391ms	remaining: 961ms
58:	learn: 0.4637415	total: 398ms	remaining: 951ms
59:	learn: 0.4615248	total: 404ms	remaining: 943ms
60:	learn: 0.4595795	total: 413ms	remaining: 941ms
61:	learn: 0.4583385	total: 415ms	remaining: 923ms
62:	learn: 0.4561042	total: 421ms	remaining: 915ms
63:	learn: 0.4540942	total: 428ms	remaining: 910ms
64:	learn: 0.4529922	total: 430ms	remaining: 893ms
65:	learn: 0.4512090	total: 436ms	remaining: 884ms
66:	learn: 0.4489739	total: 442ms	remaining: 877ms
67:	learn: 0.4467865	total: 448ms	remaining: 869ms
68:	learn: 0.4449514	total: 453ms	remaining: 860ms
69:	learn: 0.4426120	total: 459ms	remaining: 853ms
70:	learn: 0.4404611	total: 465ms	remaining: 845ms
71:	learn: 0.4380696	total: 470ms	remaining: 836ms
72:	learn: 0.4367157	total: 477ms	remaining: 829ms
73:	learn: 0.4353223	total: 482ms	remaining: 821ms
74:	learn: 0.4339963	total: 488ms	remaining: 814ms
75:	learn: 0.4315641	total: 494ms	remaining: 806ms
76:	learn: 0.4298680	total: 500ms	remaining: 798ms
77:	learn: 0.4285275	total: 506ms	remaining: 791ms
78:	learn: 0.4265791	total: 512ms	remaining: 784ms
79:	learn: 0.4244024	total: 517ms	remaining: 775ms
80:	learn: 0.4228361	total: 524ms	remaining: 769ms
81:	learn: 0.4214212	total: 529ms	remaining: 761ms
82:	learn: 0.4200662	total: 534ms	remaining: 753ms
83:	learn: 0.4184320	total: 540ms	remaining: 746ms
84:	learn: 0.4167366	total: 549ms	remaining: 741ms
85:	learn: 0.4156404	total: 554ms	remaining: 734ms
86:	learn: 0.4140656	total: 560ms	remaining: 727ms
87:	learn: 0.4124223	total: 566ms	remaining: 721ms
88:	learn: 0.4107329	total: 572ms	remaining: 713ms
89:	learn: 0.4090922	total: 579ms	remaining: 706ms
90:	learn: 0.4075416	total: 588ms	remaining: 705ms
91:	learn: 0.4062399	total: 594ms	remaining: 697ms
92:	learn: 0.4044658	total: 601ms	remaining: 692ms
93:	learn: 0.4030179	total: 607ms	remaining: 685ms
94:	learn: 0.4010536	total: 613ms	remaining: 677ms
95:	learn: 0.3994677	total: 619ms	remaining: 671ms
96:	learn: 0.3983236	total: 625ms	remaining: 664ms
97:	learn: 0.3969830	total: 632ms	remaining: 658ms
98:	learn: 0.3955620	total: 638ms	remaining: 651ms
99:	learn: 0.3946122	total: 644ms	remaining: 644ms
100:	learn: 0.3936863	total: 650ms	remaining: 637ms
101:	learn: 0.3920744	total: 656ms	remaining: 630ms
102:	learn: 0.3911707	total: 662ms	remaining: 623ms
103:	learn: 0.3895752	total: 667ms	remaining: 616ms
104:	learn: 0.3880906	total: 673ms	remaining: 609ms
105:	learn: 0.3867508	total: 679ms	remaining: 602ms
106:	learn: 0.3853061	total: 685ms	remaining: 595ms
107:	learn: 0.3841015	total: 691ms	remaining: 589ms
108:	learn: 0.3825276	total: 696ms	remaining: 581ms
109:	learn: 0.3818795	total: 702ms	remaining: 574ms
110:	learn: 0.3809667	total: 711ms	remaining: 570ms
111:	learn: 0.3790017	total: 716ms	remaining: 563ms
112:	learn: 0.3779553	total: 726ms	remaining: 559ms
113:	learn: 0.3762372	total: 732ms	remaining: 552ms
114:	learn: 0.3750418	total: 738ms	remaining: 546ms
115:	learn: 0.3735203	total: 744ms	remaining: 539ms
116:	learn: 0.3727671	total: 751ms	remaining: 533ms
117:	learn: 0.3713847	total: 756ms	remaining: 526ms
118:	learn: 0.3705588	total: 762ms	remaining: 519ms
119:	learn: 0.3694386	total: 795ms	remaining: 527ms
120:	learn: 0.3687438	total: 805ms	remaining: 523ms
121:	learn: 0.3679429	total: 814ms	remaining: 520ms
122:	learn: 0.3669140	total: 840ms	remaining: 526ms
123:	learn: 0.3654787	total: 851ms	remaining: 521ms
124:	learn: 0.3645385	total: 861ms	remaining: 517ms
125:	learn: 0.3636195	total: 870ms	remaining: 511ms
126:	learn: 0.3625194	total: 876ms	remaining: 504ms
127:	learn: 0.3609378	total: 887ms	remaining: 499ms
128:	learn: 0.3595979	total: 893ms	remaining: 492ms
129:	learn: 0.3583702	total: 901ms	remaining: 485ms
130:	learn: 0.3569225	total: 909ms	remaining: 479ms
131:	learn: 0.3558535	total: 915ms	remaining: 472ms
132:	learn: 0.3548512	total: 922ms	remaining: 464ms
133:	learn: 0.3535439	total: 947ms	remaining: 466ms
134:	learn: 0.3524239	total: 954ms	remaining: 459ms
135:	learn: 0.3517958	total: 960ms	remaining: 452ms
136:	learn: 0.3504998	total: 967ms	remaining: 445ms
137:	learn: 0.3497339	total: 976ms	remaining: 438ms
138:	learn: 0.3488863	total: 984ms	remaining: 432ms
139:	learn: 0.3482265	total: 990ms	remaining: 424ms
140:	learn: 0.3470478	total: 995ms	remaining: 416ms
141:	learn: 0.3458237	total: 1s	remaining: 409ms
142:	learn: 0.3443470	total: 1.01s	remaining: 402ms
143:	learn: 0.3430377	total: 1.01s	remaining: 394ms
144:	learn: 0.3420655	total: 1.02s	remaining: 386ms
145:	learn: 0.3417233	total: 1.02s	remaining: 379ms
146:	learn: 0.3407561	total: 1.03s	remaining: 372ms
147:	learn: 0.3397798	total: 1.04s	remaining: 364ms
148:	learn: 0.3390108	total: 1.04s	remaining: 357ms
149:	learn: 0.3381272	total: 1.05s	remaining: 350ms
150:	learn: 0.3374740	total: 1.05s	remaining: 342ms
151:	learn: 0.3365500	total: 1.06s	remaining: 335ms
152:	learn: 0.3356689	total: 1.07s	remaining: 327ms
153:	learn: 0.3346096	total: 1.07s	remaining: 320ms
154:	learn: 0.3338222	total: 1.08s	remaining: 313ms
155:	learn: 0.3330435	total: 1.09s	remaining: 307ms
156:	learn: 0.3323983	total: 1.1s	remaining: 301ms
157:	learn: 0.3312172	total: 1.1s	remaining: 294ms
158:	learn: 0.3299437	total: 1.11s	remaining: 288ms
159:	learn: 0.3286515	total: 1.12s	remaining: 280ms
160:	learn: 0.3275985	total: 1.13s	remaining: 273ms
161:	learn: 0.3266739	total: 1.14s	remaining: 267ms
162:	learn: 0.3258996	total: 1.15s	remaining: 260ms
163:	learn: 0.3249133	total: 1.15s	remaining: 252ms
164:	learn: 0.3241400	total: 1.16s	remaining: 245ms
165:	learn: 0.3237404	total: 1.16s	remaining: 238ms
166:	learn: 0.3225717	total: 1.17s	remaining: 231ms
167:	learn: 0.3218020	total: 1.17s	remaining: 224ms
168:	learn: 0.3209179	total: 1.18s	remaining: 216ms
169:	learn: 0.3200101	total: 1.19s	remaining: 209ms
170:	learn: 0.3188994	total: 1.19s	remaining: 202ms
171:	learn: 0.3178642	total: 1.2s	remaining: 195ms
172:	learn: 0.3172455	total: 1.21s	remaining: 188ms
173:	learn: 0.3166261	total: 1.21s	remaining: 181ms
174:	learn: 0.3157302	total: 1.22s	remaining: 174ms
175:	learn: 0.3149761	total: 1.22s	remaining: 167ms
176:	learn: 0.3142430	total: 1.23s	remaining: 160ms
177:	learn: 0.3138263	total: 1.23s	remaining: 153ms
178:	learn: 0.3133229	total: 1.24s	remaining: 146ms
179:	learn: 0.3124656	total: 1.25s	remaining: 139ms
180:	learn: 0.3114844	total: 1.25s	remaining: 131ms
181:	learn: 0.3109799	total: 1.26s	remaining: 125ms
182:	learn: 0.3105137	total: 1.27s	remaining: 118ms
183:	learn: 0.3094123	total: 1.27s	remaining: 111ms
184:	learn: 0.3087045	total: 1.28s	remaining: 104ms
185:	learn: 0.3079423	total: 1.29s	remaining: 96.9ms
186:	learn: 0.3071831	total: 1.29s	remaining: 89.9ms
187:	learn: 0.3061123	total: 1.3s	remaining: 82.9ms
188:	learn: 0.3054302	total: 1.3s	remaining: 76ms
189:	learn: 0.3048047	total: 1.31s	remaining: 69.1ms
190:	learn: 0.3042241	total: 1.32s	remaining: 62.2ms
191:	learn: 0.3036602	total: 1.32s	remaining: 55.2ms
192:	learn: 0.3028507	total: 1.33s	remaining: 48.3ms
193:	learn: 0.3019733	total: 1.34s	remaining: 41.4ms
194:	learn: 0.3009586	total: 1.34s	remaining: 34.4ms
195:	learn: 0.3001835	total: 1.35s	remaining: 27.5ms
196:	learn: 0.2999266	total: 1.35s	remaining: 20.6ms
197:	learn: 0.2993811	total: 1.36s	remaining: 13.7ms
198:	learn: 0.2986293	total: 1.36s	remaining: 6.86ms
199:	learn: 0.2980044	total: 1.37s	remaining: 0us
```

In [238]:

```
catb_cv_model.best_params_
```

Out[238]:

```
{'depth': 8, 'iterations': 200, 'learning_rate': 0.01}
```

In [291]:

```
catb_tuned = CatBoostClassifier(iterations=200,learning_rate=0.01,depth=8).fit(X_train,y_train,verbose=False)
```

In [292]:

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

In [293]:

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

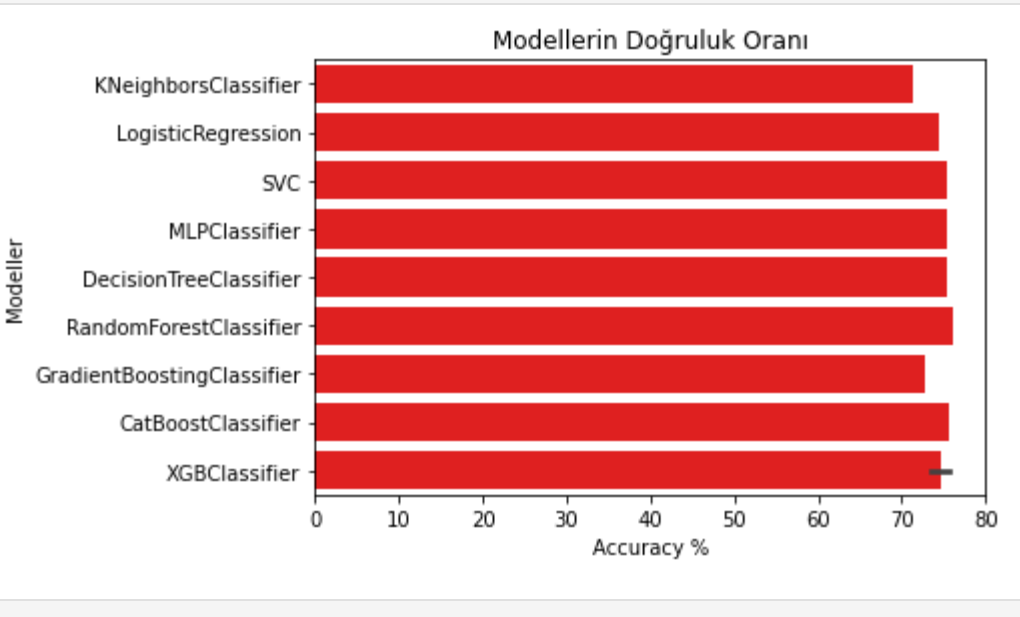
Out[293]:

```
0.7575757575757576
```

In [242]:

```
feature_imp = pd.Series(catb_tuned.feature_importances_,
                        index=X_train.columns).sort_values(ascending=False)

sns.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel("değişken önem skorları")
plt.ylabel("değişkenler")
plt.title("değişken önem düzeyleri")
plt.show()
```



Tüm Modellerin Karşılaştırılması

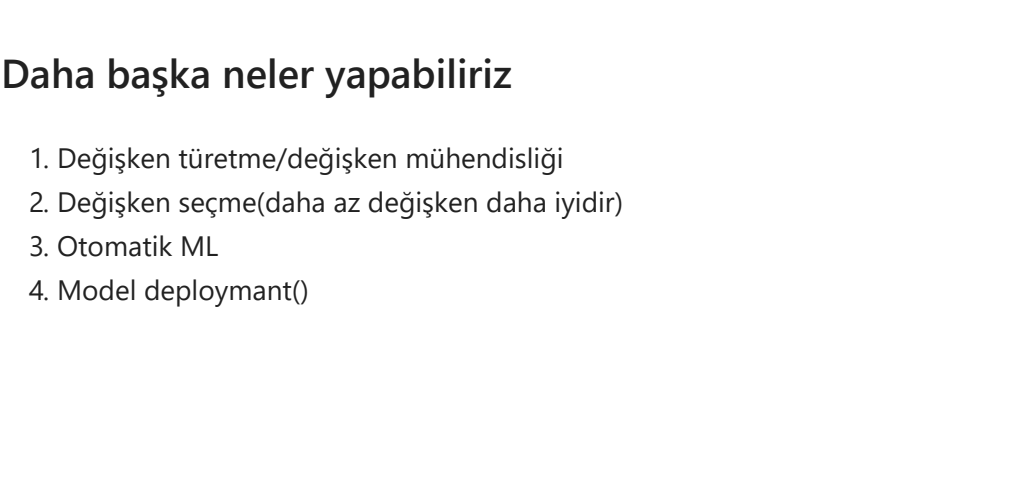
In [294]:

```
Modeller = [
    knn_tuned,
    loj_model,
    svm_tuned,
    mlpc_tuned,
    cart_tuned,
    rf_tuned,
    gbm_tuned,
    catb_tuned,
    lgbm_tuned,
    xgb_tuned
]
sonuc = []
sonuclar = pd.DataFrame(columns = ["Modeller","Accuracy"])

for model in Modeller:
    isimler = model.__class__.__name__
    y_pred = model.predict(X_test)
    dogruluk = accuracy_score(y_test,y_pred)
    sonuc = pd.DataFrame([[isimler,dogruluk*100]],columns = ["Modeller","Accuracy"])
    sonuclar = sonuclar.append(sonuc)
```

In [295]:

```
sns.barplot(x ='Accuracy', y ='Modeller',data=sonuclar,color="r")
plt.xlabel('Accuracy %')
plt.title('Modellerin Doğruluk Oranı');
```



In [296]:

```
sonuclar
```

Out[296]:

	Modeller	Accuracy
0	KNeighborsClassifier	71.428571
0	LogisticRegression	74.458874
0	SVC	75.324675
0	MLPClassifier	75.324675
0	DecisionTreeClassifier	75.324675
0	RandomForestClassifier	76.190476
0	GradientBoostingClassifier	72.727273
0	CatBoostClassifier	75.757576
0	XGBClassifier	75.757576
0	XGBClassifier	73.593074

Daha başka neler yapabiliriz

1. Değişken türetme/değişken mühendisliği
2. Değişken seçme(daha az değişken daha iyidir)
3. Otomatik ML
4. Model deployment()