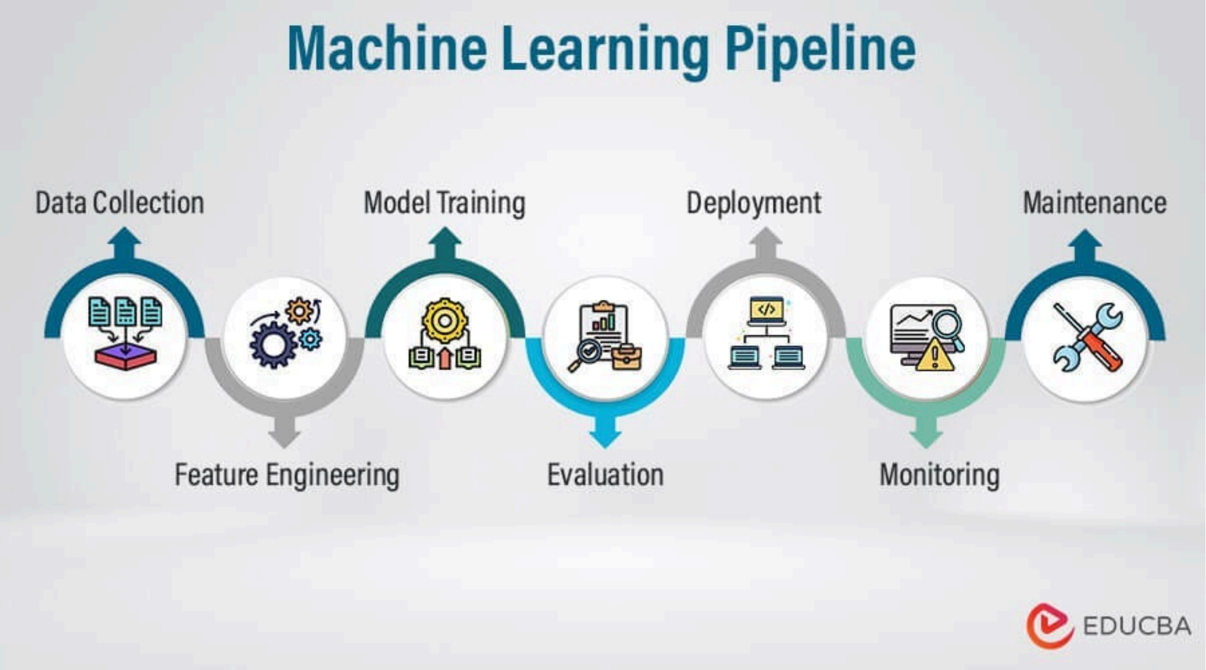
ML Pipeline

Machine Learning Pipeline

Bir **makine öğrenimi hattı (pipeline)**, ham veriden başlayıp dağıtıma hazır bir makine öğrenimi modeline ulaşana kadar geçen tüm adımların sistematik ve otomatize edilmiş bir sıraya konulmasıdır. Bunu, bir fabrikanın üretim hattına benzetebiliriz: her bir istasyon (adım), ürünü (modeli) bir sonraki aşamaya hazır hale getirir.

Amacı, bir makine öğrenimi projesindeki farklı süreçleri (veri ön işleme, özellik mühendisliği, model eğitimi, değerlendirme vb.) birbirine bağlamak ve bu süreci **tutarlı, tekrarlanabilir ve yönetilebilir** hale getirmektir.



Keşifçi Veri Analizi (EDA)

- Research→ Pipeline Diye iki .py dosyası oluşturup çalışıp canlı bir pipeline hazırlanır.
- **Joblib** is a set of tools to provide lightweight pipelining in Python. In particular:

```
import joblib
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, VotingClassifier, AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_validate, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import StandardScaler

# !pip install catboost
# !pip install lightgbm
# !pip install xgboost

from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier

pd.set_option('display.max_columns', None)
pd.set_option('display.width', 500)
```

```
##################################################
# 1. Exploratory Data Analysis
##################################################

def check_df(dataframe, head=5):
    print("##################### Shape #####################")
    print(dataframe.shape)
    print("##################### Types #####################")
    print(dataframe.dtypes)
    print("##################### Head #####################")
    print(dataframe.head(head))
    print("##################### Tail #####################")
```

```
    print(dataframe.tail(head))
    print("##################### NA #####################")
    print(dataframe.isnull().sum())
    print("##################### Quantiles ####################
#")
    #print(dataframe.quantile([0, 0.05, 0.50, 0.95, 0.99, 1]).T)
    numeric_df = dataframe.select_dtypes(include='number')
    print(numeric_df.quantile([0, 0.05, 0.50, 0.95, 0.99, 1]).T)

def cat_summary(dataframe, col_name, plot=False):
    print(pd.DataFrame({col_name: dataframe[col_name].value_counts(),
                        "Ratio": 100 * dataframe[col_name].value_counts() / len(data
frame)}))
    print("##########################################")
    if plot:
        sns.countplot(x=dataframe[col_name], data=dataframe)
        plt.show(block=True)

def num_summary(dataframe, numerical_col, plot=False):
    quantiles = [0.05, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 0.9
5, 0.99]
    print(dataframe[numerical_col].describe(quantiles).T)

    if plot:
        dataframe[numerical_col].hist(bins=20)
        plt.xlabel(numerical_col)
        plt.title(numerical_col)
        plt.show(block=True)

def target_summary_with_num(dataframe, target, numerical_col):
    print(dataframe.groupby(target).agg({numerical_col: "mean"}), end="\n
\n\n")

def target_summary_with_cat(dataframe, target, categorical_col):
    print(pd.DataFrame({"TARGET_MEAN": dataframe.groupby(categorical_c
ol)[target].mean()}), end="\n\n\n")

def correlation_matrix(df, cols):
```

```
    fig = plt.gcf()
    fig.set_size_inches(10, 8)
    plt.xticks(fontsize=10)
    plt.yticks(fontsize=10)
    fig = sns.heatmap(df[cols].corr(), annot=True, linewidths=0.5, annot_kws
={'size': 12}, linecolor='w', cmap='RdBu')
    plt.show(block=True)

def grab_col_names(dataframe, cat_th=10, car_th=20):
    """

    Veri setindeki kategorik, numerik ve kategorik fakat kardinal değişkenleri
n isimlerini verir.
    Not: Kategorik değişkenlerin içerisine numerik görünümlü kategorik deği
şkenler de dahildir.

    Parameters
    ------
        dataframe: dataframe
                Değişken isimleri alınmak istenilen dataframe
        cat_th: int, optional
                numerik fakat kategorik olan değişkenler için sınıf eşik değeri
        car_th: int, optinal
                kategorik fakat kardinal değişkenler için sınıf eşik değeri

    Returns
    ------
        cat_cols: list
                Kategorik değişken listesi
        num_cols: list
                Numerik değişken listesi
        cat_but_car: list
                Kategorik görünümlü kardinal değişken listesi

    Examples
    ------
        import seaborn as sns
        df = sns.load_dataset("iris")
```

```
        print(grab_col_names(df))


    Notes
    ------
        cat_cols + num_cols + cat_but_car = toplam değişken sayısı
        num_but_cat cat_cols'un içerisinde.
        Return olan 3 liste toplamı toplam değişken sayısına eşittir: cat_cols +
num_cols + cat_but_car = değişken sayısı

    """

    # cat_cols, cat_but_car
    cat_cols = [col for col in dataframe.columns if dataframe[col].dtypes ==
"O"]
    num_but_cat = [col for col in dataframe.columns if dataframe[col].nuniqu
e() < cat_th and
                   dataframe[col].dtypes != "O"]
    cat_but_car = [col for col in dataframe.columns if dataframe[col].nunique
() > car_th and
                   dataframe[col].dtypes == "O"]
    cat_cols = cat_cols + num_but_cat
    cat_cols = [col for col in cat_cols if col not in cat_but_car]

    # num_cols
    num_cols = [col for col in dataframe.columns if dataframe[col].dtypes !=
"O"]
    num_cols = [col for col in num_cols if col not in num_but_cat]

    # print(f"Observations: {dataframe.shape[0]}")
    # print(f"Variables: {dataframe.shape[1]}")
    # print(f'cat_cols: {len(cat_cols)}')
    # print(f'num_cols: {len(num_cols)}')
    # print(f'cat_but_car: {len(cat_but_car)}')
    # print(f'num_but_cat: {len(num_but_cat)}')
    return cat_cols, num_cols, cat_but_car
```

- **Bunların çoğunu Pipeline dosyasında kullanmayacağız.**

```
df = pd.read_csv("datasets/diabetes.csv")

check_df(df)
```

```
##################### Shape #####################
(768, 9)
##################### Types #####################
Pregnancies                   int64
Glucose                       int64
BloodPressure                 int64
SkinThickness                 int64
Insulin                       int64
BMI                         float64
DiabetesPedigreeFunction    float64
Age                           int64
Outcome                       int64
dtype: object
##################### Head #####################
   Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  Diabe
tesPedigreeFunction  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
##################### Tail #####################
     Pregnancies  Glucose  BloodPressure  SkinThickness  Insulin   BMI  Diab
etesPedigreeFunction  Age  Outcome
763           10      101             76             48      180  32.9                     0.171   63
0
764            2      122             70             27        0  36.8                     0.340   27
```

```
0
765          5     121          72          23     112  26.2                 0.245   30
0
766          1     126          60           0       0  30.1                 0.349   47
1
767          1      93          70          31       0  30.4                 0.315   23
0
##################### NA #####################
Pregnancies                0
Glucose                    0
BloodPressure              0
SkinThickness              0
Insulin                    0
BMI                        0
DiabetesPedigreeFunction    0
Age                        0
Outcome                    0
dtype: int64
##################### Quantiles #####################
                          0.00      0.05      0.50       0.95       0.99    1.00
Pregnancies               0.000   0.00000    3.0000   10.00000   13.00000   17.0
0
Glucose                   0.000  79.00000  117.0000  181.00000  196.00000  199.
00
BloodPressure             0.000  38.70000   72.0000   90.00000  106.00000  1
22.00
SkinThickness             0.000   0.00000   23.0000   44.00000   51.33000   9
9.00
Insulin                   0.000   0.00000   30.5000  293.00000  519.90000  846.
00
BMI                       0.000  21.80000   32.0000   44.39500   50.75900   67.10
DiabetesPedigreeFunction   0.078   0.14035    0.3725    1.13285    1.69833
2.42
Age                      21.000  21.00000   29.0000   58.00000   67.00000   81.00
Outcome                   0.000   0.00000    0.0000    1.00000    1.00000    1.00
```

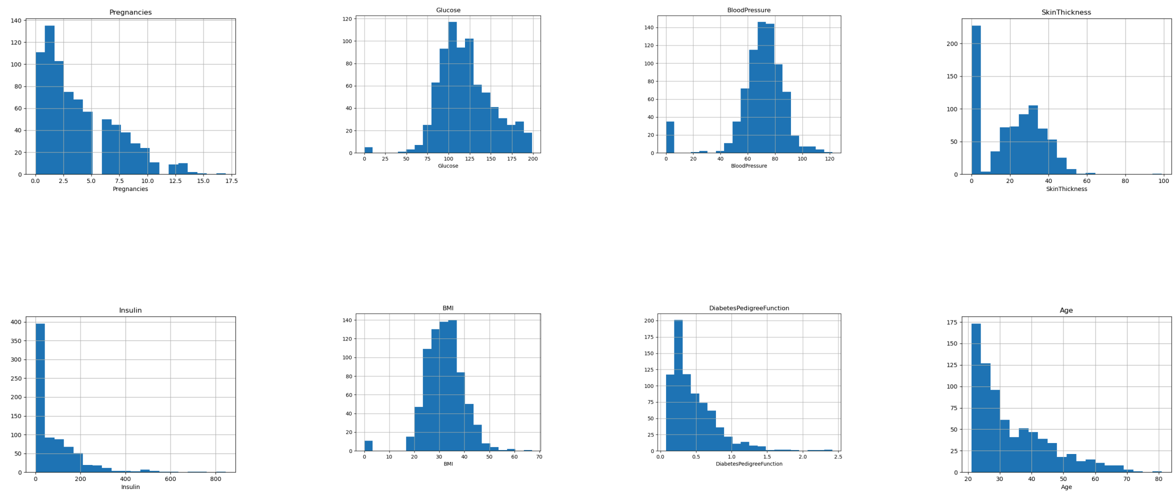
```
# Değişken türlerinin ayrıştırılması
cat_cols, num_cols, cat_but_car = grab_col_names(df, cat_th=5, car_th=20)
```

```
##→ grab_col_names(df, cat_th=5, car_th=20)
#(['Outcome'], ['Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',
'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'], [])

# Kategorik değişkenlerin incelenmesi→ Sade Target verisi
for col in cat_cols:
    cat_summary(df, col)

"""
       Outcome      Ratio
Outcome
0            500  65.104167
1            268  34.895833


"""



# Sayısal değişkenlerin incelenmesi
df[num_cols].describe().T
```

```
                         count       mean        std     min       25%       50%        75%
max
Pregnancies              768.0    3.845052    3.369578   0.000   1.00000    3.00
00    6.00000   17.00
Glucose                  768.0  120.894531   31.972618   0.000  99.00000  117.00
00  140.25000  199.00
BloodPressure            768.0   69.105469   19.355807   0.000  62.00000   7
2.0000   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   30.500
0  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
```

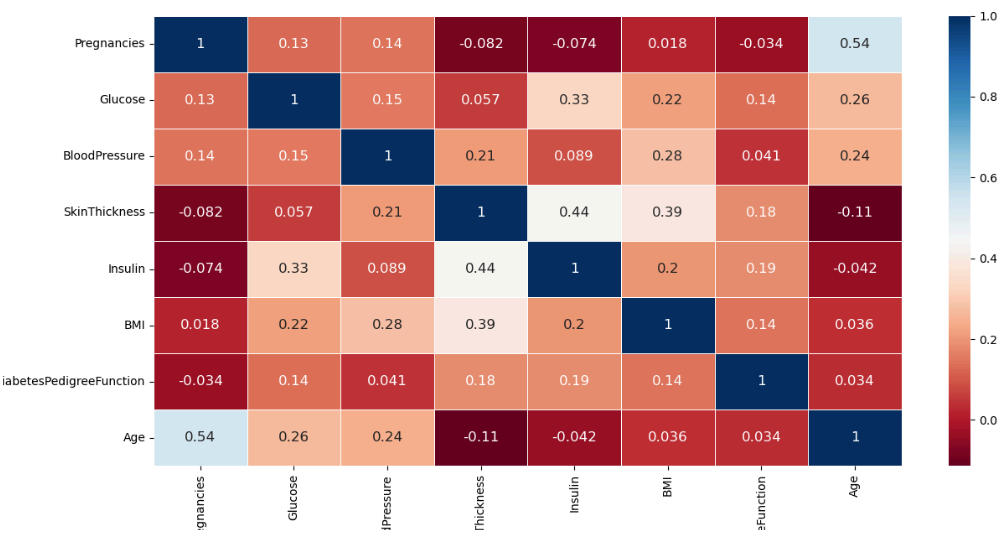
```
Age                    768.0   33.240885   11.760232  21.000  24.00000   29.00
00   41.00000   81.00
```

```
for col in num_cols:
    num_summary(df, col, plot=True)
```




```
# Sayısal değişkenkerin birbirleri ile korelasyonu
correlation_matrix(df, num_cols)

# Target ile sayısal değişkenlerin incelemesi
for col in num_cols:
    target_summary_with_num(df, "Outcome", col)

"""
         Pregnancies
Outcome
0           3.298000
1           4.865672
             Glucose
Outcome
0         109.980000
1         141.257463
          BloodPressure
```

```
Outcome
0          68.184000
1          70.824627
       SkinThickness
Outcome
0          19.664000
1          22.164179
          Insulin
Outcome
0        68.792000
1       100.335821
            BMI
Outcome
0       30.304200
1       35.142537
       DiabetesPedigreeFunction
Outcome
0                    0.429734
1                    0.550500
            Age
Outcome
0       31.190000
1       37.067164
"""
```



Veri Ön İşleme (Data Pre-Processing)

```
################################################
# 2. Data Preprocessing & Feature Engineering
################################################

def outlier_thresholds(dataframe, col_name, q1=0.25, q3=0.75):
    quartile1 = dataframe[col_name].quantile(q1)
    quartile3 = dataframe[col_name].quantile(q3)
    interquantile_range = quartile3 - quartile1
    up_limit = quartile3 + 1.5 * interquantile_range
    low_limit = quartile1 - 1.5 * interquantile_range
    return low_limit, up_limit

def replace_with_thresholds(dataframe, variable):
    low_limit, up_limit = outlier_thresholds(dataframe, variable)
    dataframe.loc[(dataframe[variable] < low_limit), variable] = low_limit
    dataframe.loc[(dataframe[variable] > up_limit), variable] = up_limit

def check_outlier(dataframe, col_name, q1=0.25, q3=0.75):
    low_limit, up_limit = outlier_thresholds(dataframe, col_name, q1, q3)
    if dataframe[(dataframe[col_name] > up_limit) | (dataframe[col_name] < l
ow_limit)].any(axis=None):
        return True
    else:
        return False
#Hem iki sınıflı hem de daha fazla sınıflı kolonlar için kullanılır.
def one_hot_encoder(dataframe, categorical_cols, drop_first=False):
    dataframe = pd.get_dummies(dataframe, columns=categorical_cols, dro
p_first=drop_first)
    return dataframe
```

```
# Değişken isimleri büyütmek
df.columns = [col.upper() for col in df.columns]
```

```
# Glucose
df['NEW_GLUCOSE_CAT'] = pd.cut(x=df['GLUCOSE'], bins=[-1, 139, 200], l
abels=["normal", "prediabetes"])

# Age
df.loc[(df['AGE'] < 35), "NEW_AGE_CAT"] = 'young'
df.loc[(df['AGE'] >= 35) & (df['AGE'] <= 55), "NEW_AGE_CAT"] = 'middleag
e'
df.loc[(df['AGE'] > 55), "NEW_AGE_CAT"] = 'old'


# BMI
df['NEW_BMI_RANGE'] = pd.cut(x=df['BMI'], bins=[-1, 18.5, 24.9, 29.9, 10
0],
                             labels=["underweight", "healty", "overweight", "obese"])

# BloodPressure
df['NEW_BLOODPRESSURE'] = pd.cut(x=df['BLOODPRESSURE'], bins=[-1,
79, 89, 123], labels=["normal", "hs1", "hs2"])
```

```
   PREGNANCIES  GLUCOSE  BLOODPRESSURE  SKINTHICKNESS  INSULIN
BMI  DIABETESPEDIGREEFUNCTION  AGE  OUTCOME NEW_GLUCOSE_CA
T NEW_AGE_CAT NEW_BMI_RANGE NEW_BLOODPRESSURE
0            6      148             72             35        0  33.6                     0.627   50
1    prediabetes   middleage        obese          normal
1            1       85             66             29        0  26.6                     0.351   31
0         normal      young    overweight          normal
2            8      183             64              0        0  23.3                     0.672   32
1    prediabetes      young       healty          normal
3            1       89             66             23       94  28.1                     0.167   21
0         normal      young    overweight          normal
4            0      137             40             35      168  43.1                     2.288   33
1         normal      young        obese          normal
```

```
check_df(df)
```

```
##################### Shape #####################
(768, 13)
##################### Types #####################
PREGNANCIES                  int64
GLUCOSE                      int64
BLOODPRESSURE                int64
SKINTHICKNESS                int64
INSULIN                      int64
BMI                        float64
DIABETESPEDIGREEFUNCTION   float64
AGE                          int64
OUTCOME                      int64
NEW_GLUCOSE_CAT           category
NEW_AGE_CAT                 object
NEW_BMI_RANGE             category
NEW_BLOODPRESSURE         category
dtype: object
##################### Head #####################
   PREGNANCIES  GLUCOSE  BLOODPRESSURE  SKINTHICKNESS  INSULIN
BMI  DIABETESPEDIGREEFUNCTION  AGE  OUTCOME NEW_GLUCOSE_CA
T NEW_AGE_CAT NEW_BMI_RANGE NEW_BLOODPRESSURE
0            6      148             72             35        0  33.6                      0.627   50
1    prediabetes   middleage        obese          normal
1            1       85             66             29        0  26.6                      0.351   31
0         normal      young    overweight          normal
2            8      183             64              0        0  23.3                      0.672   32
1    prediabetes      young       healty           normal
3            1       89             66             23       94  28.1                      0.167   21
0         normal      young    overweight          normal
4            0      137             40             35      168  43.1                      2.288   33
1        normal      young        obese          normal
##################### Tail #####################
     PREGNANCIES  GLUCOSE  BLOODPRESSURE  SKINTHICKNESS  INSULI
N   BMI  DIABETESPEDIGREEFUNCTION  AGE  OUTCOME NEW_GLUCOSE_
CAT NEW_AGE_CAT NEW_BMI_RANGE NEW_BLOODPRESSURE
```

```
763          10     101            76            48      180  32.9                 0.171   63
0        normal        old        obese          normal
764           2     122            70            27        0  36.8                 0.340   27
0        normal      young        obese          normal
765           5     121            72            23      112  26.2                 0.245   30
0        normal      young   overweight          normal
766           1     126            60             0        0  30.1                 0.349   47
1        normal   middleage       obese          normal
767           1      93            70            31        0  30.4                 0.315   23
0        normal      young        obese          normal
##################### NA #####################
PREGNANCIES                 0
GLUCOSE                     0
BLOODPRESSURE               0
SKINTHICKNESS               0
INSULIN                     0
BMI                         0
DIABETESPEDIGREEFUNCTION    0
AGE                         0
OUTCOME                     0
NEW_GLUCOSE_CAT             0
NEW_AGE_CAT                 0
NEW_BMI_RANGE               0
NEW_BLOODPRESSURE           0
dtype: int64
##################### Quantiles #####################
                           0.00      0.05      0.50      0.95      0.99    1.00
PREGNANCIES               0.000   0.00000    3.0000  10.00000   13.00000   1
7.00
GLUCOSE                   0.000  79.00000  117.0000  181.00000  196.00000  19
9.00
BLOODPRESSURE             0.000  38.70000   72.0000   90.00000  106.0000
0  122.00
SKINTHICKNESS             0.000   0.00000   23.0000   44.00000   51.33000
99.00
INSULIN                   0.000   0.00000   30.5000  293.00000  519.90000  84
6.00
BMI                       0.000  21.80000   32.0000   44.39500   50.75900   67.10
```

```
DIABETESPEDIGREEFUNCTION   0.078   0.14035    0.3725    1.13285    1.698
33    2.42
AGE                       21.000  21.00000   29.0000   58.00000   67.00000   81.00
OUTCOME                    0.000   0.00000    0.0000    1.00000    1.00000    1.0
0
```

```
cat_cols, num_cols, cat_but_car = grab_col_names(df, cat_th=5, car_th=20)
#(['NEW_AGE_CAT', 'OUTCOME', 'NEW_GLUCOSE_CAT', 'NEW_BMI_RANG
E', 'NEW_BLOODPRESSURE'], ['PREGNANCIES', 'GLUCOSE', 'BLOODPRESS
URE', 'SKINTHICKNESS', 'INSULIN', 'BMI', 'DIABETESPEDIGREEFUNCTION',
'AGE'], [])
```

```
for col in cat_cols:
    cat_summary(df, col)
```

```
            NEW_AGE_CAT      Ratio
NEW_AGE_CAT
young                488  63.541667
middleage            230  29.947917
old                   50   6.510417
##########################################
         OUTCOME      Ratio
OUTCOME
0            500  65.104167
1            268  34.895833
##########################################
                 NEW_GLUCOSE_CAT      Ratio
NEW_GLUCOSE_CAT
normal                       571  74.348958
prediabetes                  197  25.651042
##########################################
```

```
              NEW_BMI_RANGE      Ratio
NEW_BMI_RANGE
obese                  472  61.458333
overweight             179  23.307292
healty                 102  13.281250
underweight             15   1.953125
##########################################
                 NEW_BLOODPRESSURE      Ratio
NEW_BLOODPRESSURE
normal                        563  73.307292
hs1                           145  18.880208
hs2                            60   7.812500
##########################################
```

```
for col in cat_cols:
    target_summary_with_cat(df, "OUTCOME", col)
```

```
 TARGET_MEAN
NEW_AGE_CAT
middleage      0.543478
old            0.340000
young          0.258197
        TARGET_MEAN
OUTCOME
0               0.0
1               1.0

               TARGET_MEAN
NEW_GLUCOSE_CAT
normal             0.232925
prediabetes        0.685279
             TARGET_MEAN
NEW_BMI_RANGE
underweight       0.133333
```

```
healty           0.068627
overweight       0.223464
obese            0.463983
                 TARGET_MEAN
NEW_BLOODPRESSURE
normal              0.316163
hs1                 0.420690
hs2                 0.483333
```

```
cat_cols = [col for col in cat_cols if "OUTCOME" not in col]

#['NEW_AGE_CAT', 'NEW_GLUCOSE_CAT', 'NEW_BMI_RANGE', 'NEW_BLO
ODPRESSURE']
```

```
df = one_hot_encoder(df, cat_cols, drop_first=True)

check_df(df)
```

```
##################### Shape #####################
(768, 17)
##################### Types #####################
PREGNANCIES                      int64
GLUCOSE                          int64
BLOODPRESSURE                    int64
SKINTHICKNESS                    int64
INSULIN                          int64
BMI                            float64
DIABETESPEDIGREEFUNCTION       float64
AGE                              int64
OUTCOME                          int64
NEW_AGE_CAT_OLD                   bool
NEW_AGE_CAT_YOUNG                 bool
NEW_GLUCOSE_CAT_PREDIABETES       bool
NEW_BMI_RANGE_HEALTY              bool
NEW_BMI_RANGE_OVERWEIGHT          bool
NEW_BMI_RANGE_OBESE               bool
```

```
NEW_BLOODPRESSURE_HS1          bool
NEW_BLOODPRESSURE_HS2          bool
dtype: object
##################### Head #####################
  PREGNANCIES  GLUCOSE  BLOODPRESSURE  SKINTHICKNESS  INSULIN
BMI  DIABETESPEDIGREEFUNCTION  AGE  OUTCOME  NEW_AGE_CAT_OLD
NEW_AGE_CAT_YOUNG  NEW_GLUCOSE_CAT_PREDIABETES  NEW_BMI_R
ANGE_HEALTY  NEW_BMI_RANGE_OVERWEIGHT  NEW_BMI_RANGE_OBES
E  NEW_BLOODPRESSURE_HS1  NEW_BLOODPRESSURE_HS2
0          6     148           72           35       0  33.6                 0.627   50
1          False            False                  True             False                F
alse             True              False            False
1          1      85            66           29       0  26.6                 0.351   31
0          False            True                   False            False
True             False             False            False
2          8     183           64            0       0  23.3                 0.672   32
1          False            True                   True             True                 F
alse             False             False            False
3          1      89            66           23      94  28.1                 0.167   21
0          False            True                   False            False
True             False             False            False
4          0     137           40           35     168  43.1                 2.288   33
1          False            True                   False            False                F
alse             True              False            False
##################### Tail #####################
    PREGNANCIES  GLUCOSE  BLOODPRESSURE  SKINTHICKNESS  INSULI
N   BMI  DIABETESPEDIGREEFUNCTION  AGE  OUTCOME  NEW_AGE_CAT_
OLD  NEW_AGE_CAT_YOUNG  NEW_GLUCOSE_CAT_PREDIABETES  NEW_B
MI_RANGE_HEALTY  NEW_BMI_RANGE_OVERWEIGHT  NEW_BMI_RANGE_
OBESE  NEW_BLOODPRESSURE_HS1  NEW_BLOODPRESSURE_HS2
763          10     101           76           48     180  32.9                 0.171   63
0          True             False                  False            False
False            True              False            False
764           2     122           70           27       0  36.8                 0.340   27
0          False            True                   False            False
False            True              False            False
765           5     121           72           23     112  26.2                 0.245   30
0          False            True                   False            False
```

```
True             False              False              False
766          1     126           60           0       0  30.1                0.349   47
1          False           False                    False             False                 F
alse              True              False              False
767          1      93           70           31      0  30.4                0.315   23
0          False           True                     False             False
False              True              False              False
##################### NA #####################
PREGNANCIES                   0
GLUCOSE                       0
BLOODPRESSURE                 0
SKINTHICKNESS                 0
INSULIN                       0
BMI                           0
DIABETESPEDIGREEFUNCTION      0
AGE                           0
OUTCOME                       0
NEW_AGE_CAT_OLD               0
NEW_AGE_CAT_YOUNG             0
NEW_GLUCOSE_CAT_PREDIABETES   0
NEW_BMI_RANGE_HEALTY          0
NEW_BMI_RANGE_OVERWEIGHT      0
NEW_BMI_RANGE_OBESE           0
NEW_BLOODPRESSURE_HS1         0
NEW_BLOODPRESSURE_HS2         0
dtype: int64
##################### Quantiles #####################
                         0.00      0.05      0.50      0.95      0.99    1.00
PREGNANCIES              0.000   0.00000    3.0000   10.00000   13.00000   1
7.00
GLUCOSE                  0.000  79.00000  117.0000  181.00000  196.00000  19
9.00
BLOODPRESSURE            0.000  38.70000   72.0000   90.00000  106.0000
0  122.00
SKINTHICKNESS            0.000   0.00000   23.0000   44.00000   51.33000
99.00
INSULIN                  0.000   0.00000   30.5000  293.00000  519.90000  84
6.00
```

```
BMI                    0.000  21.80000   32.0000   44.39500   50.75900   67.10
DIABETESPEDIGREEFUNCTION   0.078   0.14035    0.3725    1.13285    1.698
33    2.42
AGE                   21.000  21.00000   29.0000   58.00000   67.00000   81.00
OUTCOME                    0.000   0.00000    0.0000    1.00000    1.00000    1.0
0
```

```
# Son güncel değişken türlerimi tutuyorum.
cat_cols, num_cols, cat_but_car = grab_col_names(df, cat_th=5, car_th=20)
cat_cols = [col for col in cat_cols if "OUTCOME" not in col]

#['NEW_AGE_CAT_OLD', 'NEW_AGE_CAT_YOUNG', 'NEW_GLUCOSE_CAT_P
REDIABETES', 'NEW_BMI_RANGE_HEALTY', 'NEW_BMI_RANGE_OVERWEIG
HT', 'NEW_BMI_RANGE_OBESE', 'NEW_BLOODPRESSURE_HS1', 'NEW_BLO
ODPRESSURE_HS2']
```

```
#Outlier Handling
for col in num_cols:
    print(col, check_outlier(df, col, 0.05, 0.95))

"""
PREGNANCIES False
GLUCOSE False
BLOODPRESSURE False
SKINTHICKNESS False
INSULIN True
BMI False
DIABETESPEDIGREEFUNCTION False
AGE False

"""
#Missing Handling
replace_with_thresholds(df, "INSULIN")
```

```
# Standartlaştırma
X_scaled = StandardScaler().fit_transform(df[num_cols])
df[num_cols] = pd.DataFrame(X_scaled, columns=df[num_cols].columns)
```

```
y = df["OUTCOME"]
X = df.drop(["OUTCOME"], axis=1)

check_df(df)
```

```
##################### Shape #####################
(768, 16)
##################### Types #####################
PREGNANCIES                     float64
GLUCOSE                         float64
BLOODPRESSURE                   float64
SKINTHICKNESS                   float64
INSULIN                         float64
BMI                             float64
DIABETESPEDIGREEFUNCTION        float64
AGE                             float64
NEW_AGE_CAT_OLD                    bool
NEW_AGE_CAT_YOUNG                  bool
NEW_GLUCOSE_CAT_PREDIABETES        bool
NEW_BMI_RANGE_HEALTY               bool
NEW_BMI_RANGE_OVERWEIGHT           bool
NEW_BMI_RANGE_OBESE                bool
NEW_BLOODPRESSURE_HS1              bool
NEW_BLOODPRESSURE_HS2              bool
dtype: object
##################### Head #####################
   PREGNANCIES   GLUCOSE  BLOODPRESSURE  SKINTHICKNESS   INSULI
N      BMI  DIABETESPEDIGREEFUNCTION      AGE  NEW_AGE_CAT_OLD  N
EW_AGE_CAT_YOUNG  NEW_GLUCOSE_CAT_PREDIABETES  NEW_BMI_RA
NGE_HEALTY  NEW_BMI_RANGE_OVERWEIGHT  NEW_BMI_RANGE_OBESE
NEW_BLOODPRESSURE_HS1  NEW_BLOODPRESSURE_HS2
```

```
0    0.639947  0.848324      0.149641      0.907270 -0.787602  0.204013
0.468492  1.425995          False          False                 True
False                 False            True             False             False
1    -0.844885 -1.123396     -0.160546      0.530902 -0.787602 -0.684422
-0.365061 -0.190672          False          True                 False
False                 True            False             False             False
2    1.233880  1.943724     -0.263941     -1.288212 -0.787602 -1.103255
0.604397 -0.105584          False          True                 True
True                 False            False             False             False
3    -0.844885 -0.998208     -0.160546      0.154533  0.217583 -0.494043
-0.920763 -1.041549          False          True                 False
False                 True            False             False             False
4    -1.141852  0.504055     -1.504687      0.907270  1.008900  1.409746
5.484909 -0.020496          False          True                 False
False                 False            True             False             False
##################### Tail #####################
    PREGNANCIES  GLUCOSE  BLOODPRESSURE  SKINTHICKNESS  INSUL
IN      BMI  DIABETESPEDIGREEFUNCTION      AGE  NEW_AGE_CAT_OLD
NEW_AGE_CAT_YOUNG  NEW_GLUCOSE_CAT_PREDIABETES  NEW_BMI_R
ANGE_HEALTY  NEW_BMI_RANGE_OVERWEIGHT  NEW_BMI_RANGE_OBES
E  NEW_BLOODPRESSURE_HS1  NEW_BLOODPRESSURE_HS2
763     1.827813 -0.622642      0.356432      1.722735  1.137221  0.115169
-0.908682  2.532136            True          False                 False
False                 False            True             False             False
764    -0.547919  0.034598      0.046245      0.405445 -0.787602  0.61015
4              -0.398282 -0.531023          False          True                 Fa
lse              False                 False            True             False
False
765     0.342981  0.003301      0.149641      0.154533  0.410066 -0.735190
-0.685193 -0.275760          False          True                 False
False                 True            False             False             False
766    -0.844885  0.159787     -0.470732     -1.288212 -0.787602 -0.24020
5              -0.371101  1.170732          False          False                 Fals
e              False                 False            True             False
False
767    -0.844885 -0.873019      0.046245      0.656358 -0.787602 -0.2021
29              -0.473785 -0.871374          False          True                 F
alse              False                 False            True             False
```

```
False
##################### NA #####################
PREGNANCIES                 0
GLUCOSE                     0
BLOODPRESSURE               0
SKINTHICKNESS               0
INSULIN                     0
BMI                         0
DIABETESPEDIGREEFUNCTION    0
AGE                         0
NEW_AGE_CAT_OLD             0
NEW_AGE_CAT_YOUNG           0
NEW_GLUCOSE_CAT_PREDIABETES    0
NEW_BMI_RANGE_HEALTY        0
NEW_BMI_RANGE_OVERWEIGHT    0
NEW_BMI_RANGE_OBESE         0
NEW_BLOODPRESSURE_HS1       0
NEW_BLOODPRESSURE_HS2       0
dtype: int64
##################### Quantiles #####################
                              0.00      0.05      0.50      0.95      0.99      1.00
PREGNANCIES              -1.141852 -1.141852 -0.250952  1.827813  2.718712
3.906578
GLUCOSE                  -3.783654 -1.311179 -0.121888  1.881130  2.350587  2.
444478
BLOODPRESSURE            -3.572597 -1.571894  0.149641  1.080200  1.90736
4  2.734528
SKINTHICKNESS            -1.288212 -1.288212  0.154533  1.471822  1.931619
4.921866
INSULIN                  -0.787602 -0.787602 -0.461451  2.345582  2.614256
2.614256
BMI                      -4.060474 -1.293634  0.000942  1.574106  2.381820  4.4
55807
DIABETESPEDIGREEFUNCTION -1.189553 -1.001249 -0.300128  1.996219
3.704036  5.883565
AGE                      -1.041549 -1.041549 -0.360847  2.106697  2.872487  4.06
3716
```

Functionalization

```
def diabetes_data_prep(dataframe):
    dataframe.columns = [col.upper() for col in dataframe.columns]

    # Glucose
    dataframe['NEW_GLUCOSE_CAT'] = pd.cut(x=dataframe['GLUCOSE'], bins=[-1, 139, 200], labels=["normal", "prediabetes"])

    # Age
    dataframe.loc[(dataframe['AGE'] < 35), "NEW_AGE_CAT"] = 'young'
    dataframe.loc[(dataframe['AGE'] >= 35) & (dataframe['AGE'] <= 55), "NEW_AGE_CAT"] = 'middleage'
    dataframe.loc[(dataframe['AGE'] > 55), "NEW_AGE_CAT"] = 'old'

    # BMI
    dataframe['NEW_BMI_RANGE'] = pd.cut(x=dataframe['BMI'], bins=[-1, 18.5, 24.9, 29.9, 100],
                                        labels=["underweight", "healty", "overweight", "obese"])

    # BloodPressure
    dataframe['NEW_BLOODPRESSURE'] = pd.cut(x=dataframe['BLOODPRESSURE'], bins=[-1, 79, 89, 123],
                                            labels=["normal", "hs1", "hs2"])

    cat_cols, num_cols, cat_but_car = grab_col_names(dataframe, cat_th=5, car_th=20)

    cat_cols = [col for col in cat_cols if "OUTCOME" not in col]

    df = one_hot_encoder(dataframe, cat_cols, drop_first=True)

    df.columns = [col.upper() for col in df.columns]

    cat_cols, num_cols, cat_but_car = grab_col_names(df, cat_th=5, car_th=20)
```

```
    cat_cols = [col for col in cat_cols if "OUTCOME" not in col]

    replace_with_thresholds(df, "INSULIN")

    X_scaled = StandardScaler().fit_transform(df[num_cols])
    df[num_cols] = pd.DataFrame(X_scaled, columns=df[num_cols].column
s)

    y = df["OUTCOME"]
    X = df.drop(["OUTCOME"], axis=1)

    return X, y
```

Base Models

- Pipeline'da yer almaz sadece sonuçlarınız gözlemlemek için yapılır.

```
def base_models(X, y, scoring="roc_auc"):
    print("Base Models....")
    classifiers = [('LR', LogisticRegression()),
                   ('KNN', KNeighborsClassifier()),
                   ("SVC", SVC()),
                   ("CART", DecisionTreeClassifier()),
                   ("RF", RandomForestClassifier()),
                   ('Adaboost', AdaBoostClassifier()),
                   ('GBM', GradientBoostingClassifier()),
                   ('XGBoost', XGBClassifier(use_label_encoder=False, eval_metric
='logloss')),
                   ('LightGBM', LGBMClassifier()),
                   # ('CatBoost', CatBoostClassifier(verbose=False))
                   ]

    for name, classifier in classifiers:
        cv_results = cross_validate(classifier, X, y, cv=3, scoring=scoring)
        print(f"{scoring}: {round(cv_results['test_score'].mean(), 4)} ({name})
")
```

```
base_models(X, y, scoring="accuracy")
```

```
Base Models....
accuracy: 0.7604 (LR)
accuracy: 0.7617 (KNN)
accuracy: 0.7656 (SVC)
accuracy: 0.6784 (CART)
accuracy: 0.7656 (RF)
accuracy: 0.7578 (Adaboost)
accuracy: 0.7487 (GBM)
accuracy: 0.7487 (XGBoost)
accuracy: 0.7383 (LightGBM)
```

```
base_models(X, y, scoring="roc_auc")
```

```
Base Models....
roc_auc: 0.841 (LR)
roc_auc: 0.791 (KNN)
roc_auc: 0.8355 (SVC)
roc_auc: 0.6473 (CART)
roc_auc: 0.8279 (RF)
roc_auc: 0.8196 (Adaboost)
roc_auc: 0.8226 (GBM)
roc_auc: 0.7938 (XGBoost)
roc_auc: 0.807 (LightGBM)
```

Automated Hyperparameter Optimization

```
knn_params = {"n_neighbors": range(2, 50)}

cart_params = {'max_depth': range(1, 20),
             "min_samples_split": range(2, 30)}
```

```
rf_params = {"max_depth": [8, 15, None],
             "max_features": [5, 7, "auto"],
             "min_samples_split": [15, 20],
             "n_estimators": [200, 300]}

xgboost_params = {"learning_rate": [0.1, 0.01],
                  "max_depth": [5, 8],
                  "n_estimators": [100, 200]}

lightgbm_params = {"learning_rate": [0.01, 0.1],
                   "n_estimators": [300, 500]}



classifiers = [('KNN', KNeighborsClassifier(), knn_params),
               ("CART", DecisionTreeClassifier(), cart_params),
               ("RF", RandomForestClassifier(), rf_params),
               ('XGBoost', XGBClassifier(use_label_encoder=False, eval_metric
='logloss'), xgboost_params),
               ('LightGBM', LGBMClassifier(), lightgbm_params)]
```

```
def hyperparameter_optimization(X, y, cv=3, scoring="roc_auc"):
    print("Hyperparameter Optimization....")
    best_models = {}
    for name, classifier, params in classifiers:
        print(f"########## {name} ##########")
        cv_results = cross_validate(classifier, X, y, cv=cv, scoring=scoring)
        print(f"{scoring} (Before): {round(cv_results['test_score'].mean(), 4)}")

        gs_best = GridSearchCV(classifier, params, cv=cv, n_jobs=-1, verbose
=False).fit(X, y)
        final_model = classifier.set_params(**gs_best.best_params_)

        cv_results = cross_validate(final_model, X, y, cv=cv, scoring=scoring)
        print(f"{scoring} (After): {round(cv_results['test_score'].mean(), 4)}")
        print(f"{name} best params: {gs_best.best_params_}", end="\n\n")
        best_models[name] = final_model
    return best_models
```

```
best_models = hyperparameter_optimization(X, y)
```

```
Hyperparameter Optimization....
########## KNN ##########
roc_auc (Before): 0.791
roc_auc (After): 0.8211
KNN best params: {'n_neighbors': 20}
########## CART ##########
roc_auc (Before): 0.6433
roc_auc (After): 0.7943
########## RF ##########
roc_auc (Before): 0.8269
roc_auc (After): 0.8372
RF best params: {'max_depth': 8, 'max_features': 5, 'min_samples_split': 15,
'n_estimators': 200}
########## XGBoost ##########
roc_auc (Before): 0.7938
roc_auc (After): 0.8139
XGBoost best params: {'learning_rate': 0.01, 'max_depth': 5, 'n_estimators':
100}
########## LightGBM ##########
roc_auc (Before): 0.807
roc_auc (After): 0.8185
LightGBM best params: {'learning_rate': 0.01, 'n_estimators': 300}
```

Stacking & Ensemble Learning

- **Ensemble Learning,** makine öğrenmesinde **birden fazla modelin tahminlerini birleştirerek** tek bir modelden daha iyi, daha kararlı ve daha genellenebilir tahminler elde etmeyi amaçlayan genel bir şemsiye terimdir.

  Tek bir "uzman" yerine, bir "uzmanlar komitesi"nin daha doğru kararlar vereceği fikrine dayanır. Her modelin farklı güçlü yönleri ve zayıf yönleri olabilir. Bu modellerin birleştirilmesiyle, bireysel modellerin hataları birbirini dengeleyebilir ve genel performansı artırabilir.

- **Stacking** (Stacked Generalization): **Stacking**, Ensemble Learning teknikleri arasında özellikle **farklı türdeki modellerin (heterojen modeller)** güçlü yönlerini birleştirmek için kullanılan güçlü bir yöntemdir. Diğer ensemble yöntemlerinden (Bagging, Boosting) temel farkı, **modellerin tahminlerini birleştirmek için başka bir model (meta-model) kullanmasıdır.**

```
############################################################
# 5. Stacking & Ensemble Learning
############################################################

def voting_classifier(best_models, X, y):
    print("Voting Classifier...")
    voting_clf = VotingClassifier(estimators=[('KNN', best_models["KNN"]),
                                              ('RF', best_models["RF"]),
                                              ('LightGBM', best_models["LightGBM"])],
                                  voting='soft').fit(X, y)
    #    voting : {'hard', 'soft'}, default='hard' If 'hard', uses predicted class la
bels for majority rule voting.
    #    Else if 'soft', predicts the class label based on the argmax of the sum
s of the predicted probabilities, which is recommended for
    #    an ensemble of well-calibrated classifiers.

    cv_results = cross_validate(voting_clf, X, y, cv=3, scoring=["accuracy", "f
1", "roc_auc"])
    print(f"Accuracy: {cv_results['test_accuracy'].mean()}")
    print(f"F1Score: {cv_results['test_f1'].mean()}")
    print(f"ROC_AUC: {cv_results['test_roc_auc'].mean()}")
    return voting_clf

voting_clf = voting_classifier(best_models, X, y)
```

```
Accuracy: 0.7708333333333334
F1Score: 0.6350140611304488
ROC_AUC: 0.8358954098181207
```

1. Hard Voting (Sert Oylama)

- **Nasıl Çalışır?** Her bir temel modelin (örneğin KNN, Random Forest, LightGBM) **doğrudan tahmin ettiği sınıf etiketleri** (yani 'evet' veya 'hayır', 0 veya 1 gibi) alınır. Nihai tahmin, **çoğunluk oylaması** ile belirlenir. En çok oy alan sınıf, nihai sınıf olarak atanır.
- **Ne Zaman Kullanılır?** Modellerin güven puanları veya olasılık tahminleri önemli olmadığında veya modellerin iyi kalibre edilmediği durumlarda tercih edilebilir.
- **Örnek:** Eğer 3 modelden 2'si sınıf A'yı, 1'i sınıf B'yi tahmin ederse, nihai tahmin A olur.

2. Soft Voting (Yumuşak Oylama)

- **Nasıl Çalışır?** Her bir temel modelin **tahmin ettiği sınıf olasılıkları** (veya güven puanları) alınır. Bu olasılıklar, her sınıf için toplanır ve **en yüksek toplam olasılığa sahip sınıf**, nihai tahmin olarak seçilir. `VotingClassifier` dokümantasyonunda da belirtildiği gibi, **iyi kalibre edilmiş modeller için** genellikle önerilen yöntemdir. Modellerin kendi içlerindeki güven derecelerini hesaba katarak daha incelikli bir karar verir.
- **Ne Zaman Kullanılır?** Modellerin sınıf olasılıkları üretebildiği ve bu olasılıkların güvenilir olduğu durumlarda daha iyi performans gösterebilir. Genellikle daha iyi sonuçlar verir çünkü modellerin "ne kadar emin" olduklarını da dikkate alır.
- **Örnek:**
    - Model 1: Sınıf A için 0.9, Sınıf B için 0.1
    - Model 2: Sınıf A için 0.4, Sınıf B için 0.6
    - Model 3: Sınıf A için 0.2, Sınıf B için 0.8
    - **Toplam Olasılıklar:** Sınıf A için (0.9 + 0.4 + 0.2) = 1.5; Sınıf B için (0.1 + 0.6 + 0.8) = 1.5
    - Bu örnekte, toplam olasılıklar eşit çıktı, ancak gerçek senaryoda Soft Voting ile yüksek olasılığa sahip sınıf kazanır. Eğer A: 1.6, B: 1.4 olsaydı, A kazanırdı.

New Observation

```
###########################################################
# 6. Prediction for a New Observation
###########################################################

X.columns
random_user = X.sample(1, random_state=45)
voting_clf.predict(random_user)

#Bu modeli kaydetmek için
joblib.dump(voting_clf, "voting_clf2.pkl")

new_model = joblib.load("voting_clf2.pkl")
new_model.predict(random_user)
```

Pipeline

```
####################################################
# End-to-End Diabetes Machine Learning Pipeline II
####################################################

import joblib
import pandas as pd
from lightgbm import LGBMClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingCl
assifier, VotingClassifier, AdaBoostClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_validate, GridSearchCV
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from xgboost import XGBClassifier
```

- "Helper functions" (Yardımcı fonksiyonlar), programlamada çok yaygın ve kullanışlı bir kavramdır. Adından da anlaşılacağı gibi, **başka bir ana fonksiyonun veya program parçasının belirli bir görevi yerine getirmesine yardımcı olan** fonksiyonlardır.

  Kodumuzdaki `grab_col_names`, `outlier_thresholds`, `replace_with_thresholds`, `one_hot_encoder`, `diabetes_data_prep` gibi fonksiyonlar bu kategoriye girer.

```
################################################
# Helper Functions
################################################

# utils.py
# helpers.py

# Data Preprocessing & Feature Engineering
def grab_col_names(dataframe, cat_th=10, car_th=20):
    """

    Veri setindeki kategorik, numerik ve kategorik fakat kardinal değişkenleri
n isimlerini verir.
    Not: Kategorik değişkenlerin içerisine numerik görünümlü kategorik deği
şkenler de dahildir.

    Parameters
    ------
        dataframe: dataframe
                Değişken isimleri alınmak istenilen dataframe
        cat_th: int, optional
                numerik fakat kategorik olan değişkenler için sınıf eşik değeri
        car_th: int, optinal
                kategorik fakat kardinal değişkenler için sınıf eşik değeri

    Returns
    ------
        cat_cols: list
                Kategorik değişken listesi
        num_cols: list
                Numerik değişken listesi
```

```
    cat_but_car: list
        Kategorik görünümlü kardinal değişken listesi

  Examples
  ------
    import seaborn as sns
    df = sns.load_dataset("iris")
    print(grab_col_names(df))


  Notes
  ------
    cat_cols + num_cols + cat_but_car = toplam değişken sayısı
    num_but_cat cat_cols'un içerisinde.
    Return olan 3 liste toplamı toplam değişken sayısına eşittir: cat_cols +
num_cols + cat_but_car = değişken sayısı

  """

  # cat_cols, cat_but_car
  cat_cols = [col for col in dataframe.columns if dataframe[col].dtypes ==
"O"]
  num_but_cat = [col for col in dataframe.columns if dataframe[col].nuniqu
e() < cat_th and
           dataframe[col].dtypes != "O"]
  cat_but_car = [col for col in dataframe.columns if dataframe[col].nunique
() > car_th and
           dataframe[col].dtypes == "O"]
  cat_cols = cat_cols + num_but_cat
  cat_cols = [col for col in cat_cols if col not in cat_but_car]

  # num_cols
  num_cols = [col for col in dataframe.columns if dataframe[col].dtypes !=
"O"]
  num_cols = [col for col in num_cols if col not in num_but_cat]

  # print(f"Observations: {dataframe.shape[0]}")
  # print(f"Variables: {dataframe.shape[1]}")
```

```
    # print(f'cat_cols: {len(cat_cols)}')
    # print(f'num_cols: {len(num_cols)}')
    # print(f'cat_but_car: {len(cat_but_car)}')
    # print(f'num_but_cat: {len(num_but_cat)}')
    return cat_cols, num_cols, cat_but_car

def outlier_thresholds(dataframe, col_name, q1=0.25, q3=0.75):
    quartile1 = dataframe[col_name].quantile(q1)
    quartile3 = dataframe[col_name].quantile(q3)
    interquantile_range = quartile3 - quartile1
    up_limit = quartile3 + 1.5 * interquantile_range
    low_limit = quartile1 - 1.5 * interquantile_range
    return low_limit, up_limit

def replace_with_thresholds(dataframe, variable):
    low_limit, up_limit = outlier_thresholds(dataframe, variable)
    dataframe.loc[(dataframe[variable] < low_limit), variable] = low_limit
    dataframe.loc[(dataframe[variable] > up_limit), variable] = up_limit

def one_hot_encoder(dataframe, categorical_cols, drop_first=False):
    dataframe = pd.get_dummies(dataframe, columns=categorical_cols, dro
p_first=drop_first)
    return dataframe

def diabetes_data_prep(dataframe):
    dataframe.columns = [col.upper() for col in dataframe.columns]

    # Glucose
    dataframe['NEW_GLUCOSE_CAT'] = pd.cut(x=dataframe['GLUCOSE'], bi
ns=[-1, 139, 200], labels=["normal", "prediabetes"])

    # Age
    dataframe.loc[(dataframe['AGE'] < 35), "NEW_AGE_CAT"] = 'young'
    dataframe.loc[(dataframe['AGE'] >= 35) & (dataframe['AGE'] <= 55), "NE
W_AGE_CAT"] = 'middleage'
    dataframe.loc[(dataframe['AGE'] > 55), "NEW_AGE_CAT"] = 'old'

    # BMI
```

```
    dataframe['NEW_BMI_RANGE'] = pd.cut(x=dataframe['BMI'], bins=[-1, 1
8.5, 24.9, 29.9, 100],
                                        labels=["underweight", "healty", "overweight", "obe
se"])

    # BloodPressure
    dataframe['NEW_BLOODPRESSURE'] = pd.cut(x=dataframe['BLOODPRE
SSURE'], bins=[-1, 79, 89, 123],
                                            labels=["normal", "hs1", "hs2"])

    cat_cols, num_cols, cat_but_car = grab_col_names(dataframe, cat_th=5,
car_th=20)

    cat_cols = [col for col in cat_cols if "OUTCOME" not in col]

    df = one_hot_encoder(dataframe, cat_cols, drop_first=True)

    cat_cols, num_cols, cat_but_car = grab_col_names(df, cat_th=5, car_th=2
0)

    replace_with_thresholds(df, "INSULIN")

    X_scaled = StandardScaler().fit_transform(df[num_cols])
    df[num_cols] = pd.DataFrame(X_scaled, columns=df[num_cols].column
s)

    y = df["OUTCOME"]
    X = df.drop(["OUTCOME"], axis=1)

    return X, y

# Base Models
def base_models(X, y, scoring="roc_auc"):
    print("Base Models....")
    classifiers = [('LR', LogisticRegression()),
                   ('KNN', KNeighborsClassifier()),
                   ("SVC", SVC()),
                   ("CART", DecisionTreeClassifier()),
```

```
                ("RF", RandomForestClassifier()),
                ('Adaboost', AdaBoostClassifier()),
                ('GBM', GradientBoostingClassifier()),
                ('XGBoost', XGBClassifier(use_label_encoder=False, eval_metric
='logloss')),
                ('LightGBM', LGBMClassifier()),
                # ('CatBoost', CatBoostClassifier(verbose=False))
                ]

    for name, classifier in classifiers:
        cv_results = cross_validate(classifier, X, y, cv=3, scoring=scoring)
        print(f"{scoring}: {round(cv_results['test_score'].mean(), 4)} ({name})
")

# Hyperparameter Optimization

# config.py

knn_params = {"n_neighbors": range(2, 50)}

cart_params = {'max_depth': range(1, 20),
               "min_samples_split": range(2, 30)}

rf_params = {"max_depth": [8, 15, None],
             "max_features": [5, 7, "auto"],
             "min_samples_split": [15, 20],
             "n_estimators": [200, 300]}

xgboost_params = {"learning_rate": [0.1, 0.01],
                  "max_depth": [5, 8],
                  "n_estimators": [100, 200],
                  "colsample_bytree": [0.5, 1]}

lightgbm_params = {"learning_rate": [0.01, 0.1],
                   "n_estimators": [300, 500],
                   "colsample_bytree": [0.7, 1]}

classifiers = [('KNN', KNeighborsClassifier(), knn_params),
```

```
            ("CART", DecisionTreeClassifier(), cart_params),
            ("RF", RandomForestClassifier(), rf_params),
            ('XGBoost', XGBClassifier(use_label_encoder=False, eval_metric
='logloss'), xgboost_params),
            ('LightGBM', LGBMClassifier(), lightgbm_params)]

def hyperparameter_optimization(X, y, cv=3, scoring="roc_auc"):
    print("Hyperparameter Optimization....")
    best_models = {}
    for name, classifier, params in classifiers:
        print(f"########## {name} ##########")
        cv_results = cross_validate(classifier, X, y, cv=cv, scoring=scoring)
        print(f"{scoring} (Before): {round(cv_results['test_score'].mean(), 4)}")

        gs_best = GridSearchCV(classifier, params, cv=cv, n_jobs=-1, verbose
=False).fit(X, y)
        final_model = classifier.set_params(**gs_best.best_params_)

        cv_results = cross_validate(final_model, X, y, cv=cv, scoring=scoring)
        print(f"{scoring} (After): {round(cv_results['test_score'].mean(), 4)}")
        print(f"{name} best params: {gs_best.best_params_}", end="\n\n")
        best_models[name] = final_model
    return best_models

# Stacking & Ensemble Learning
def voting_classifier(best_models, X, y):
    print("Voting Classifier...")
    voting_clf = VotingClassifier(estimators=[('KNN', best_models["KNN"]),
('RF', best_models["RF"]),
                                              ('LightGBM', best_models["LightGBM"])],
                                  voting='soft').fit(X, y)
    cv_results = cross_validate(voting_clf, X, y, cv=3, scoring=["accuracy", "f
1", "roc_auc"])
    print(f"Accuracy: {cv_results['test_accuracy'].mean()}")
    print(f"F1Score: {cv_results['test_f1'].mean()}")
    print(f"ROC_AUC: {cv_results['test_roc_auc'].mean()}")
    return voting_clf
```

- `# utils.py`: Bu dosya genellikle, projenin farklı yerlerinde kullanılabilecek **genel amaçlı "utility" (yardımcı/araç) fonksiyonlar** içerir. Bu fonksiyonlar genellikle belirli bir etki alanına (domain) özel değildir; daha çok genel veri işleme, matematiksel hesaplamalar, string manipülasyonları gibi görevleri yerine getirirler.
    - **Örnek Fonksiyonlar:** Veri okuma/yazma, tarih/saat formatlama, hata loglama, genel veri temizleme yardımcıları.
- `# helpers.py`: Benzer şekilde, projenin başka bir yerinde `helpers.py` adında bir Python dosyası olması muhtemeldir. Bu dosya da yardımcı fonksiyonlar içerir, ancak "utils" ile arasındaki ayrım bazen ince olabilir. Genellikle `helpers.py`, **daha belirli bir bağlama veya belirli bir modüle özel yardımcı fonksiyonlar** barındırır. Yani, "utils" daha genelken, "helpers" biraz daha projenin o anki bölümüne veya belirli bir probleme daha yakın olabilir.
    - **Örnek Fonksiyonlar:** Diyabet projenizde gördüğünüz `grab_col_names`, `outlier_thresholds`, `replace_with_thresholds`, `one_hot_encoder` gibi fonksiyonlar, aslında `helpers.py` veya `utils.py` gibi bir dosyada gruplandırılmaya çok uygun "helper functions"dır. Zaten kodunuzda da bu fonksiyonların altında belirtilmiş.
- `config.py` dosyası, Python projelerinde **yapılandırma (configuration) bilgilerini** depolamak için kullanılan yaygın bir adlandırma geleneğidir. Projenizin çalışması için gereken, ancak kodun doğrudan mantığına ait olmayan tüm ayarları ve sabit değerleri burada tutarsınız.

Prediction

```
################################################
# End-to-End Diabetes Machine Learning Pipeline III
################################################

import joblib
import pandas as pd

df = pd.read_csv("datasets/diabetes.csv")

random_user = df.sample(1, random_state=45)
```

```
new_model = joblib.load("voting_clf.pkl")

new_model.predict(random_user) # Her değer almıyor.

#Prediction veya scoring işlemi de denir
from diabetes_pipeline import diabetes_data_prep

X, y = diabetes_data_prep(df)

random_user = X.sample(1, random_state=1)

new_model = joblib.load("voting_clf.pkl")

new_model.predict(random_user)
```