
WELCOME!

Welcome to "**Employee Churn Analysis Project**". This is the second project of Capstone Project Series, which you will be able to build your own classification models for a variety of business settings.

Also you will research what is Employee Churn?, How it is different from customer churn, Exploratory data analysis and visualization of employee churn dataset using **matplotlib** and **seaborn**, model building and evaluation using python **scikit-learn** and **Tensorflow-Keras** packages.

You will be able to implement classification techniques in Python. Using Scikit-Learn allowing you to successfully make predictions with Distance Based, Bagging, Boosting algorithms for this project. On the other hand, for Deep Learning you will use Tensorflow-Keras.

At the end of the project, you will have the opportunity to deploy your model using *Streamlit*.

Before diving into the project, please take a look at the determines and project structure.

- NOTE: This project assumes that you already know the basics of coding in Python and are familiar with model deployment as well as the theory behind Distance Based, Bagging, Boosting algorithms, and Confusion Matrices. You can try more models and methods beside these to improve your model metrics.

#Determines

In this project you have HR data of a company. A study is requested from you to predict which employee will churn by using this data.

The HR dataset has 14,999 samples. In the given dataset, you have two types of employee one who stayed and another who left the company.

You can describe 10 attributes in detail as:

- **satisfaction_level:** It is employee satisfaction point, which ranges from 0-1.
- **last_evaluation:** It is evaluated performance by the employer, which also ranges from 0-1.
- **number_projects:** How many of projects assigned to an employee?
- **average_monthly_hours:** How many hours in average an employee worked in a month?
- **time_spent_company:** time_spent_company means employee experience. The number of years spent by an employee in the company.
- **work_accident:** Whether an employee has had a work accident or not.

- ***promotion_last_5years:*** Whether an employee has had a promotion in the last 5 years or not.
- ***Departments:*** Employee's working department/division.
- ***Salary:*** Salary level of the employee such as low, medium and high.
- ***left:*** Whether the employee has left the company or not.

First of all, to observe the structure of the data, outliers, missing values and features that affect the target variable, you must use exploratory data analysis and data visualization techniques.

Then, you must perform data pre-processing operations such as ***Scaling*** and ***Encoding*** to increase the accuracy score of Gradient Descent Based or Distance-Based algorithms.

You are asked to perform ***Cluster Analysis*** based on the information you obtain during exploratory data analysis and data visualization processes. The purpose of clustering analysis is to cluster data with similar characteristics.

Once the data is ready to be applied to the model, you must ***split the data into train and test.*** Then build a model to predict whether employees will churn or not. Train your models with your train set, test the success of your model with your test set.

Try to make your predictions by using the *** Classification Algorithms. ***You can use the related modules of the scikit-learn****** and ***Tensorflow-Keras*** library. You can use scikit-learn ***Classification Metrics*** module for accuracy calculation.

In the final step, you will deploy your model using Streamlit tool.

#Tasks

1. Exploratory Data Analysis

- EDA is an initial process of analysis, in which you can summarize characteristics of data such as pattern, trends, outliers, and hypothesis testing using descriptive statistics and visualization.
- In the given dataset, you have two types of employee one who stayed and another who left the company. So, you can divide data into two groups and compare their characteristics.

2. Data Visualization

- Explore your data via visualizations to find-out:
- What can be the reason of the churn?
- Behavioral analysis of chuns and not chuns etc.

3. Cluster Analysis

- Apply ***clustering algorithms*** and writedown your conclusions about the clusters you created.

4. Predictive Model Building

- Split Data as Train and Test set
- Built Classification Models(at least four models) and Evaluate Model Performances

5. Model Deployment

- Save and Export the Best Model
- Deploy best model via Streamlit

Employee Churn Analysis Project Amaç: Bir şirkette çalışanların işten ayrılma olasılığını tahmin etmek için bir sınıflandırma modeli oluşturmak.

Veri Seti:

Numune Sayısı: 14,999 Öz nitelikler: 10 ana özellik (memnuniyet seviyesi, son değerlendirme, proje sayısı, aylık çalışma saatı, şirkette geçirilen süre, iş kazası durumu, son 5 yıldaki terfi durumu, departman, maaş, ayrılma durumu) Proje Adımları:

Keşifsel Veri Analizi (EDA):

Verinin yapısını, aykırı değerleri, eksik değerleri ve hedef değişkeni etkileyen özellikleri gözlemleyin. Veriyi, ayrılan ve ayrılmayan çalışanlar olarak iki gruba ayırarak karşılaştırın.

Veri Görselleştirme:

Veriyi grafikler ve görselleştirmelerle analiz edin. Çalışanların ayrılma nedenlerini ve davranışlarını inceleyin. Kümeleme Analizi:

Kümeleme algoritmaları uygulayın ve oluşturduğunuz kümeler hakkında sonuçlar çıkarın.
Tahmin Modeli Oluşturma:

Veriyi eğitim ve test setlerine ayırin. En az dört farklı sınıflandırma modeli oluşturun ve değerlendirin (Distance-Based, Bagging, Boosting algoritmaları, Tensorflow-Keras). Model performansını doğrulama ve değerlendirme yapın.

Model Dağıtımı:

En iyi modeli kaydedin ve dışa aktarın. Streamlit kullanarak modelinizi dağıtin.

Notlar:

Proje, Python'da kodlama, model dağıtıımı ve sınıflandırma algoritmaları hakkında temel bilgiye sahip olduğunuzu varsayar. Veriyi ölçekte ve kodlama gibi veri ön işleme adımlarını uygulayarak model doğruluğunu artırabilirsiniz.

#Importing Modules and Predefined Functions

```
!pip install skimpy

Collecting skimpy
  Downloading skimpy-0.0.15-py3-none-any.whl.metadata (28 kB)
Requirement already satisfied: Pygments<3.0.0,>=2.10.0 in
/usr/local/lib/python3.10/dist-packages (from skimpy) (2.16.1)
Requirement already satisfied: click<9.0.0,>=8.1.6 in
/usr/local/lib/python3.10/dist-packages (from skimpy) (8.1.7)
Collecting ipykernal<7.0.0,>=6.7.0 (from skimpy)
  Downloading ipykernal-6.29.5-py3-none-any.whl.metadata (6.3 kB)
Requirement already satisfied: numpy<2.0.0,>=1.22.2 in
```

```
/usr/local/lib/python3.10/dist-packages (from skimpy) (1.26.4)
Requirement already satisfied: pandas<3.0.0,>=2.0.3 in
/usr/local/lib/python3.10/dist-packages (from skimpy) (2.1.4)
Requirement already satisfied: polars<0.21,>=0.19 in
/usr/local/lib/python3.10/dist-packages (from skimpy) (0.20.2)
Requirement already satisfied: pyarrow<17,>=13 in
/usr/local/lib/python3.10/dist-packages (from skimpy) (14.0.2)
Requirement already satisfied: rich<14.0,>=10.9 in
/usr/local/lib/python3.10/dist-packages (from skimpy) (13.8.0)
Collecting typeguard==4.2.1 (from skimpy)
  Downloading typeguard-4.2.1-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: typing-extensions>=4.10.0 in
/usr/local/lib/python3.10/dist-packages (from typeguard==4.2.1->skimpy) (4.12.2)
Collecting comm>=0.1.1 (from ipykernel<7.0.0,>=6.7.0->skimpy)
  Downloading comm-0.2.2-py3-none-any.whl.metadata (3.7 kB)
Requirement already satisfied: debugpy>=1.6.5 in
/usr/local/lib/python3.10/dist-packages (from ipykernel<7.0.0,>=6.7.0->skimpy) (1.6.6)
Requirement already satisfied: ipython>=7.23.1 in
/usr/local/lib/python3.10/dist-packages (from ipykernel<7.0.0,>=6.7.0->skimpy) (7.34.0)
Requirement already satisfied: jupyter-client>=6.1.12 in
/usr/local/lib/python3.10/dist-packages (from ipykernel<7.0.0,>=6.7.0->skimpy) (6.1.12)
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in
/usr/local/lib/python3.10/dist-packages (from ipykernel<7.0.0,>=6.7.0->skimpy) (5.7.2)
Requirement already satisfied: matplotlib-inline>=0.1 in
/usr/local/lib/python3.10/dist-packages (from ipykernel<7.0.0,>=6.7.0->skimpy) (0.1.7)
Requirement already satisfied: nest-asyncio in
/usr/local/lib/python3.10/dist-packages (from ipykernel<7.0.0,>=6.7.0->skimpy) (1.6.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from ipykernel<7.0.0,>=6.7.0->skimpy) (24.1)
Requirement already satisfied: psutil in
/usr/local/lib/python3.10/dist-packages (from ipykernel<7.0.0,>=6.7.0->skimpy) (5.9.5)
Requirement already satisfied: pyzmq>=24 in
/usr/local/lib/python3.10/dist-packages (from ipykernel<7.0.0,>=6.7.0->skimpy) (24.0.1)
Requirement already satisfied: tornado>=6.1 in
/usr/local/lib/python3.10/dist-packages (from ipykernel<7.0.0,>=6.7.0->skimpy) (6.3.3)
Requirement already satisfied: traitlets>=5.4.0 in
/usr/local/lib/python3.10/dist-packages (from ipykernel<7.0.0,>=6.7.0->skimpy) (5.7.1)
```

```
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas<3.0.0,>=2.0.3->skimpy) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas<3.0.0,>=2.0.3->skimpy) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-packages (from pandas<3.0.0,>=2.0.3->skimpy) (2024.1)
Requirement already satisfied: markdown-it-py>=2.2.0 in
/usr/local/lib/python3.10/dist-packages (from rich<14.0,>=10.9->skimpy) (3.0.0)
Requirement already satisfied: setuptools>=18.5 in
/usr/local/lib/python3.10/dist-packages (from ipython>=7.23.1->ipykernel<7.0.0,>=6.7.0->skimpy) (71.0.4)
Collecting jedi>=0.16 (from ipython>=7.23.1->ipykernel<7.0.0,>=6.7.0->skimpy)
  Using cached jedi-0.19.1-py2.py3-none-any.whl.metadata (22 kB)
Requirement already satisfied: decorator in
/usr/local/lib/python3.10/dist-packages (from ipython>=7.23.1->ipykernel<7.0.0,>=6.7.0->skimpy) (4.4.2)
Requirement already satisfied: pickleshare in
/usr/local/lib/python3.10/dist-packages (from ipython>=7.23.1->ipykernel<7.0.0,>=6.7.0->skimpy) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0, !=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from ipython>=7.23.1->ipykernel<7.0.0,>=6.7.0->skimpy) (3.0.47)
Requirement already satisfied: backcall in
/usr/local/lib/python3.10/dist-packages (from ipython>=7.23.1->ipykernel<7.0.0,>=6.7.0->skimpy) (0.2.0)
Requirement already satisfied: pexpect>4.3 in
/usr/local/lib/python3.10/dist-packages (from ipython>=7.23.1->ipykernel<7.0.0,>=6.7.0->skimpy) (4.9.0)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.10/dist-packages (from jupyter-core!=5.0.*,>=4.12->ipykernel<7.0.0,>=6.7.0->skimpy) (4.2.2)
Requirement already satisfied: mdurl~0.1 in
/usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich<14.0,>=10.9->skimpy) (0.1.2)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas<3.0.0,>=2.0.3->skimpy) (1.16.0)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython>=7.23.1->ipykernel<7.0.0,>=6.7.0->skimpy) (0.8.4)
Requirement already satisfied: ptyprocess>=0.5 in
/usr/local/lib/python3.10/dist-packages (from pexpect>4.3->ipython>=7.23.1->ipykernel<7.0.0,>=6.7.0->skimpy) (0.7.0)
Requirement already satisfied: wcwidth in
```

```
/usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0, !=3.0.1,<3.1.0,>=2.0.0->ipython>=7.23.1->ipykernel<7.0.0,>=6.7.0->skimpy) (0.2.13)
Downloading skimpy-0.0.15-py3-none-any.whl (16 kB)
Downloading typeguard-4.2.1-py3-none-any.whl (34 kB)
Downloading ipykernel-6.29.5-py3-none-any.whl (117 kB)
117.2/117.2 kB 4.2 MB/s eta
0:00:00
m-0.2.2-py3-none-any.whl (7.2 kB)
Using cached jedi-0.19.1-py2.py3-none-any.whl (1.6 MB)
Installing collected packages: typeguard, jedi, comm, ipykernel, skimpy
Attempting uninstall: typeguard
  Found existing installation: typeguard 4.3.0
  Uninstalling typeguard-4.3.0:
    Successfully uninstalled typeguard-4.3.0
Attempting uninstall: ipykernel
  Found existing installation: ipykernel 5.5.6
  Uninstalling ipykernel-5.5.6:
    Successfully uninstalled ipykernel-5.5.6
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
google-colab 1.0.0 requires ipykernel==5.5.6, but you have ipykernel
6.29.5 which is incompatible.
Successfully installed comm-0.2.2 ipykernel-6.29.5 jedi-0.19.1 skimpy-
0.0.15 typeguard-4.2.1

#!pip install numpy pandas seaborn matplotlib

Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (1.26.4)
Requirement already satisfied: pandas in
/usr/local/lib/python3.10/dist-packages (2.1.4)
Requirement already satisfied: seaborn in
/usr/local/lib/python3.10/dist-packages (0.13.1)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: python-dateutil>=2.8.2 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-packages (from pandas) (2024.1)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.3.0)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (4.53.1)
```

```
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (24.1)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.4)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas) (1.16.0)

#!pip install ipywidgets

Requirement already satisfied: ipywidgets in
/usr/local/lib/python3.10/dist-packages (7.7.1)
Requirement already satisfied: ipykernel>=4.5.1 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets) (6.29.5)
Requirement already satisfied: ipython-genutils~0.2.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets) (0.2.0)
Requirement already satisfied: traitlets>=4.3.1 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets) (5.7.1)
Requirement already satisfied: widgetsnbextension~=3.6.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets) (3.6.8)
Requirement already satisfied: ipython>=4.0.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets) (7.34.0)
Requirement already satisfied: jupyterlab-widgets>=1.0.0 in
/usr/local/lib/python3.10/dist-packages (from ipywidgets) (3.0.13)
Requirement already satisfied: comm>=0.1.1 in
/usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets) (0.2.2)
Requirement already satisfied: debugpy>=1.6.5 in
/usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets) (1.6.6)
Requirement already satisfied: jupyter-client>=6.1.12 in
/usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets) (6.1.12)
Requirement already satisfied: jupyter-core!=5.0.*,>=4.12 in
/usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets) (5.7.2)
Requirement already satisfied: matplotlib-inline>=0.1 in
/usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets) (0.1.7)
Requirement already satisfied: nest-asyncio in
/usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets) (1.6.0)
Requirement already satisfied: packaging in
/usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets) (24.1)
Requirement already satisfied: psutil in
```

```
/usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets) (5.9.5)
Requirement already satisfied: pyzmq>=24 in
/usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets) (24.0.1)
Requirement already satisfied: tornado>=6.1 in
/usr/local/lib/python3.10/dist-packages (from ipykernel>=4.5.1->ipywidgets) (6.3.3)
Requirement already satisfied: setuptools>=18.5 in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets) (71.0.4)
Requirement already satisfied: jedi>=0.16 in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets) (0.19.1)
Requirement already satisfied: decorator in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets) (4.4.2)
Requirement already satisfied: pickleshare in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets) (0.7.5)
Requirement already satisfied: prompt-toolkit!=3.0.0,!=3.0.1,<3.1.0,>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets) (3.0.47)
Requirement already satisfied: pygments in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets) (2.16.1)
Requirement already satisfied: backcall in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets) (0.2.0)
Requirement already satisfied: pexpect>4.3 in
/usr/local/lib/python3.10/dist-packages (from ipython>=4.0.0->ipywidgets) (4.9.0)
Requirement already satisfied: notebook>=4.4.1 in
/usr/local/lib/python3.10/dist-packages (from widgetsnbextension~=3.6.0->ipywidgets) (6.5.5)
Requirement already satisfied: parso<0.9.0,>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from jedi>=0.16->ipython>=4.0.0->ipywidgets) (0.8.4)
Requirement already satisfied: python-dateutil>=2.1 in
/usr/local/lib/python3.10/dist-packages (from jupyter-client>=6.1.12->ipykernel>=4.5.1->ipywidgets) (2.8.2)
Requirement already satisfied: platformdirs>=2.5 in
/usr/local/lib/python3.10/dist-packages (from jupyter-core!=5.0.*,>=4.12->ipykernel>=4.5.1->ipywidgets) (4.2.2)
Requirement already satisfied: jinja2 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (3.1.4)
Requirement already satisfied: argon2-cffi in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
```

```
>widgetsnbextension~=3.6.0->ipywidgets) (23.1.0)
Requirement already satisfied: nbformat in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (5.10.4)
Requirement already satisfied: nbconvert>=5 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (6.5.4)
Requirement already satisfied: Send2Trash>=1.8.0 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (1.8.3)
Requirement already satisfied: terminado>=0.8.3 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (0.18.1)
Requirement already satisfied: prometheus-client in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (0.20.0)
Requirement already satisfied: nbclassic>=0.4.7 in
/usr/local/lib/python3.10/dist-packages (from notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (1.1.0)
Requirement already satisfied: ptyprocess>=0.5 in
/usr/local/lib/python3.10/dist-packages (from pexpect>4.3-
>ipython>=4.0.0->ipywidgets) (0.7.0)
Requirement already satisfied: wcwidth in
/usr/local/lib/python3.10/dist-packages (from prompt-toolkit!=3.0.0,!-
=3.0.1,<3.1.0,>=2.0.0->ipython>=4.0.0->ipywidgets) (0.2.13)
Requirement already satisfied: notebook-shim>=0.2.3 in
/usr/local/lib/python3.10/dist-packages (from nbclassic>=0.4.7-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.2.4)
Requirement already satisfied: lxml in /usr/local/lib/python3.10/dist-
packages (from nbconvert>=5->notebook>=4.4.1-
>widgetsnbextension~=3.6.0->ipywidgets) (4.9.4)
Requirement already satisfied: beautifulsoup4 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (4.12.3)
Requirement already satisfied: bleach in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (6.1.0)
Requirement already satisfied: defusedxml in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.7.1)
Requirement already satisfied: entrypoints>=0.2.2 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.4)
Requirement already satisfied: jupyterlab-pygments in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.3.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (2.1.5)
```

```
Requirement already satisfied: mistune<2,>=0.8.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.8.4)
Requirement already satisfied: nbclient>=0.5.0 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.10.0)
Requirement already satisfied: pandocfilters>=1.4.1 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (1.5.1)
Requirement already satisfied: tinycss2 in
/usr/local/lib/python3.10/dist-packages (from nbconvert>=5-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (1.3.0)
Requirement already satisfied: fastjsonschema>=2.15 in
/usr/local/lib/python3.10/dist-packages (from nbformat-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (2.20.0)
Requirement already satisfied: jsonschema>=2.6 in
/usr/local/lib/python3.10/dist-packages (from nbformat-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (4.23.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.1-
>jupyter-client>=6.1.12->ipykernel>=4.5.1->ipywidgets) (1.16.0)
Requirement already satisfied: argon2-cffi-bindings in
/usr/local/lib/python3.10/dist-packages (from argon2-cffi-
>notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (21.2.0)
Requirement already satisfied: attrs>=22.2.0 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets)
(24.2.0)
Requirement already satisfied: jsonschema-specifications>=2023.03.6 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets)
(2023.12.1)
Requirement already satisfied: referencing>=0.28.4 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets)
(0.35.1)
Requirement already satisfied: rpds-py>=0.7.1 in
/usr/local/lib/python3.10/dist-packages (from jsonschema>=2.6-
>nbformat->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets)
(0.20.0)
Requirement already satisfied: jupyter-server<3,>=1.8 in
/usr/local/lib/python3.10/dist-packages (from notebook-shim>=0.2.3-
>nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0-
>ipywidgets) (1.24.0)
Requirement already satisfied: cffi>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from argon2-cffi-bindings-
>argon2-cffi->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets)
(1.17.0)
Requirement already satisfied: soupsieve>1.2 in
```

```
/usr/local/lib/python3.10/dist-packages (from beautifulsoup4->nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (2.6)
Requirement already satisfied: webencodings in
/usr/local/lib/python3.10/dist-packages (from bleach->nbconvert>=5->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (0.5.1)
Requirement already satisfied: pycparser in
/usr/local/lib/python3.10/dist-packages (from cffi>=1.0.1->argon2-cffi-bindings->argon2-cffi->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (2.22)
Requirement already satisfied: anyio<4,>=3.1.0 in
/usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (3.7.1)
Requirement already satisfied: websocket-client in
/usr/local/lib/python3.10/dist-packages (from jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (1.8.0)
Requirement already satisfied: idna>=2.8 in
/usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (3.8)
Requirement already satisfied: sniffio>=1.1 in
/usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (1.3.1)
Requirement already satisfied: exceptiongroup in
/usr/local/lib/python3.10/dist-packages (from anyio<4,>=3.1.0->jupyter-server<3,>=1.8->notebook-shim>=0.2.3->nbclassic>=0.4.7->notebook>=4.4.1->widgetsnbextension~=3.6.0->ipywidgets) (1.2.2)

!pip install livelossplot

Collecting livelossplot
  Downloading livelossplot-0.5.5-py3-none-any.whl.metadata (8.7 kB)
Requirement already satisfied: matplotlib in
/usr/local/lib/python3.10/dist-packages (from livelossplot) (3.7.1)
Requirement already satisfied: bokeh in
/usr/local/lib/python3.10/dist-packages (from livelossplot) (3.4.3)
Requirement already satisfied: Jinja2>=2.9 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot) (3.1.4)
Requirement already satisfied: contourpy>=1.2 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot) (1.3.0)
Requirement already satisfied: numpy>=1.16 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot) (1.26.4)
Requirement already satisfied: packaging>=16.8 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot)
```

```
(24.1)
Requirement already satisfied: pandas>=1.2 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot)
(2.1.4)
Requirement already satisfied: pillow>=7.1.0 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot)
(9.4.0)
Requirement already satisfied: PyYAML>=3.10 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot)
(6.0.2)
Requirement already satisfied: tornado>=6.2 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot)
(6.3.3)
Requirement already satisfied: xyzservices>=2021.09.1 in
/usr/local/lib/python3.10/dist-packages (from bokeh->livelossplot)
(2024.6.0)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib-
>livelossplot) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib-
>livelossplot) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib-
>livelossplot) (1.4.5)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib-
>livelossplot) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib-
>livelossplot) (2.8.2)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.10/dist-packages (from Jinja2>=2.9->bokeh-
>livelossplot) (2.1.5)
Requirement already satisfied: pytz>=2020.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.2->bokeh-
>livelossplot) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in
/usr/local/lib/python3.10/dist-packages (from pandas>=1.2->bokeh-
>livelossplot) (2024.1)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7-
>matplotlib->livelossplot) (1.16.0)
Downloading livelossplot-0.5.5-py3-none-any.whl (22 kB)
Installing collected packages: livelossplot
Successfully installed livelossplot-0.5.5

# Importing libraries for DL and ML
import tensorflow as tf
from tensorflow import keras
```

```

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation
from tensorflow.keras.callbacks import EarlyStopping
from tensorflow.keras.optimizers import Adam
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import recall_score,\ 
                           f1_score, precision_recall_curve,\ 
                           average_precision_score
from sklearn.experimental import enable_halving_search_cv
from sklearn.model_selection import GridSearchCV,\ 
                                   HalvingGridSearchCV,\ 
                                   RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler,\ 
                                 OneHotEncoder
from sklearn.compose import ColumnTransformer
from livelossplot import PlotLossesKerasTF

# Importing the libraries for EDA
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import os

from ipywidgets import interact
%matplotlib inline
# %matplotlib notebook
plt.rcParams["figure.figsize"] = (12, 6)
# plt.rcParams['figure.dpi'] = 100
sns.set_style("darkgrid")
import warnings

warnings.filterwarnings("ignore")
pd.set_option('display.float_format', lambda x: '%.2f' % x)

```

Bu kod parçası, veri analizi ve makine öğrenimi süreçlerini yönetmek için gerekli kütüphaneleri yükler ve yapılandırır. Öncelikle, skimpy ve livelossplot gibi kütüphaneler yüklenir; bunlar verileri özetlemek ve model eğitiminde kayıpları görselleştirmek için kullanılır. Daha sonra, TensorFlow ve Keras gibi derin öğrenme kütüphaneleri ile Scikit-learn'ün çeşitli araçları içe aktarılır. TensorFlow ve Keras, makine öğrenimi ve derin öğrenme modellerini oluşturmak ve eğitmek için kullanılırken, Scikit-learn model değerlendirme ve hiperparametre optimizasyonu için kullanılır. train_test_split, MinMaxScaler, ve OneHotEncoder gibi fonksiyonlar veri ön işleme ve model seçimi işlemlerinde yardımcı olur.

Ek olarak, pandas, NumPy, Seaborn ve Matplotlib gibi kütüphaneler veri analizi ve görselleştirmesi için kullanılır. Pandas, sayıları belirli bir formatta göstermek için ayarlanmıştır ve Seaborn ile Matplotlib grafiklerin stilini ve boyutunu düzenlemek için kullanılır. ipywidgets, Jupyter notebook içinde etkileşimli araçlar oluşturmak için kullanılırken, warnings kütüphanesi uyarıları gizler. Bu yapılandırma, veri analizi, model eğitimi ve değerlendirmesi için kapsamlı bir

ortam sağlar, aynı zamanda eğitim sürecinin görselleştirilmesini ve hiperparametre ayarlarını kolaylaştırır.

```
def show_distribution(col):
    """
    Plots a histogram and boxplot of a numeric column in a Pandas DataFrame,
    and prints statistical calculations about the column.

    Parameters:
    -----
    col : Pandas Series
        A numeric column in a Pandas DataFrame.

    Returns:
    -----
    None.

    Prints:
    -----
    Statistical calculations about the column, including minimum value, mean, median, mode, and maximum value.

    Plots:
    -----
    A histogram of the column's values, with lines indicating the minimum, mean, median, mode, and maximum values.

    A boxplot of the column's values, with a line indicating the median and markers indicating the mean and outliers.
    """

    # Get statistics
    from termcolor import colored

    print(colored('Statistical Calculations : ', 'red',
    attrs=['bold']))
    print(colored('-'*26, 'red', attrs=['bold']))
    min_val = col.min()
    max_val = col.max()
    mean_val = col.mean()
    med_val = col.median()
    mod_val = col.mode()[0]

    print(colored('Minimum:{:>7.2f}\nMean:{:>10.2f}\nMedian:{:>8.2f}\nMode:{:>10.2f}\nMaximum:{:>7.2f}\n'.format(min_val,
```

```

mean_val,
med_val,
mod_val,
max_val), 'blue', attrs=['bold']))

# Create a figure for 2 subplots (2 rows, 1 column)
fig, ax = plt.subplots(2, 1, figsize=(15, 15))

# Plot the histogram
ax[0].hist(col, bins=30)
ax[0].set_ylabel('Frequency', fontsize=10)

# Add lines for the mean, median, and mode
ax[0].axvline(x=min_val, color='orange', linestyle='dashed',
linewidth=2, label='Minimum')
ax[0].axvline(x=mean_val, color='lightgreen', linestyle='dashed',
linewidth=2, label='Mean')
ax[0].axvline(x=med_val, color='cyan', linestyle='dashed',
linewidth=2, label='Median')
ax[0].axvline(x=mod_val, color='purple', linestyle='dashed',
linewidth=2, label='Mode')
ax[0].axvline(x=max_val, color='red', linestyle='dashed',
linewidth=2, label='Maximum')
ax[0].legend(loc='upper right')

# Plot the boxplot
medianprops = dict(linestyle='--', linewidth=3, color='m')
boxprops=dict(linestyle='--', linewidth=1.5)
meanprops={"marker": "d", "markerfacecolor": "blue",
"markeredgewidth": 2, "markersize": 10}
flierprops={'marker': 'o', 'markersize': 8, 'markerfacecolor':
'fuchsia'}

ax[1].boxplot(col,
              vert=False,
              notch=True,
              patch_artist=False,
              medianprops=medianprops,
              flierprops=flierprops,
              showmeans=True,
              meanprops=meanprops)

ax[1].set_xlabel('value', fontsize=10)

# Add a title to the Figure
fig.suptitle('Data Distribution', fontsize=20)

```

Fonksiyon: show_distribution

show_distribution fonksiyonu, bir Pandas DataFrame sütunundaki sayısal verinin dağılımını detaylı bir şekilde analiz etmek ve görselleştirmek için tasarlanmıştır. Fonksiyon, öncelikle sütundaki verinin temel istatistiklerini hesaplar. Bu istatistikler minimum değer, maksimum değer, ortalama, medyan ve mod içerir. Bu hesaplamalar, renkli ve kalın bir biçimde ekrana yazdırılır, böylece verinin temel özellikleri hızlı bir şekilde görülebilir.

Fonksiyonun görselleştirme kısmı iki ana bileşenden oluşur: histogram ve boxplot. Histogram, sütundaki değerlerin frekans dağılımını gösterir ve üzerinde verinin minimum, ortalama, medyan, mod ve maksimum değerlerini temsil eden kesikli çizgiler bulunur. Bu çizgiler, verinin dağılımı hakkında daha fazla bilgi sağlar ve çeşitli veri noktalarını görsel olarak ayırt etmeye yardımcı olur.

Boxplot ise, veri setinin medyanını, çeyrekler arası aralığını ve aykırı değerlerini gösterir. Bu grafik, verinin genel dağılımını ve uç değerleri daha iyi anlamak için kullanılır. Medyan, kalın bir çizgi ile gösterilirken, ortalama değerler farklı renklerde işaretlerle gösterilir ve aykırı değerler özel işaretlerle belirtilir.

Fonksiyon, bu iki grafiği bir arada sunarak, veri analizi sürecinde hem genel dağılımı hem de detaylı özet bilgileri görsel olarak sunar. Bu analiz, veri hakkında derinlemesine bilgi edinmek ve veri ön işleme aşamalarında önemli kararlar almak için oldukça faydalıdır.

```
def first_looking(df, col):
    """
    Prints basic information about a column in a Pandas DataFrame.

    Parameters:
    -----
    df : pandas.DataFrame
        The DataFrame to analyze.
    col : str
        The name of the column to analyze.

    Returns:
    -----
    None.

    Prints:
    -----
    column_name      : str
        The name of the column being analyzed.
    per_of_nulls     : float
        The percentage of null values in the column.
    num_of_nulls     : int
        The number of null values in the column.
    num_of_uniques   : int
        The number of unique values in the column.
    shape_of_df       : tuple
        The shape of the DataFrame.
    
```

```

The unique values in the column and their frequency of occurrence.

"""
print("column name      : ", col)
print("-----")
print("per_of_nulls    : ", "%", round(df[col].isnull().sum() * 100
/ df.shape[0], 2))
print("num_of_nulls    : ", df[col].isnull().sum())
print("num_of_uniques : ", df[col].astype(str).nunique())
print("shape_of_df     : ", df.shape)
print("-----")
print(df[col].value_counts(dropna=False))

```

Fonksiyon: first_looking

first_looking fonksiyonu, bir Pandas DataFrame'de belirli bir sütun hakkında temel bilgileri hızlıca özetlemek için kullanılır. Bu fonksiyon, sütunun adı, null değerlerin yüzdesi ve sayısı, benzersiz değerlerin sayısı ve DataFrame'in şekli gibi bilgileri ekrana yazdırır.

Fonksiyon, öncelikle analiz edilen sütunun adını ekrana getirir. Ardından, sütundaki null (boş) değerlerin yüzdesini ve toplam sayısını hesaplayarak bu bilgileri kullanıcıya sunar. Null değerlerin yüzdesi, sütundaki boş değerlerin veri kümelerindeki toplam veri sayısına oranı olarak hesaplanır ve yüzde formatında gösterilir. Ayrıca, sütundaki benzersiz değerlerin sayısını da hesaplar, bu da sütunda kaç farklı değer olduğunu belirtir.

Fonksiyon, ayrıca DataFrame'in genel şekli hakkında bilgi verir, yani satır ve sütun sayısını gösterir. Son olarak, sütundaki tüm değerlerin frekanslarını içeren bir özet tabloyu da ekrana yazdırır. Bu özet tablo, her bir değer ve bu değerin kaç kez tekrarlandığını gösterir.

Bu fonksiyon, veri analizine başlamadan önce sütunların genel özelliklerini hızlıca gözden geçirmeye yardımcı olur, böylece verinin kalitesi ve içeriği hakkında temel bir anlayış elde edilir.

1. Exploratory Data Analysis

```

from google.colab import drive
drive.mount('/content/drive')

Mounted at /content/drive

df0 = pd.read_csv('/content/drive/MyDrive/HR_Dataset.csv')
df = df0.copy()

```

Bu kod parçası, Google Colab ortamında bir Google Drive bağlantısı gerçekleştirir ve veriye erişim sağlar. İlk olarak, google.colab kütüphanesinden drive modülü kullanılarak Google Drive'ı Colab oturumuna bağlamak için drive.mount('/content/drive') komutu çalıştırılır. Bu işlem, Google Drive'daki dosyaların Colab'de erişilebilir olmasını sağlar. Ardından, pd.read_csv('/content/drive/MyDrive/HR_Dataset.csv') komutu kullanılarak, Google Drive'daki 'HR_Dataset.csv' adlı CSV dosyası pandas veri çerçevesine (df0) yüklenir. Son olarak, df0 veri çerçevesinin bir kopyası df olarak oluşturulur. Bu kopya üzerinde veri işleme ve analiz işlemleri yapılabilirken, orijinal veri çerçevesi df0 değişmeden kalır. Bu yapı, veri üzerinde çalışırken orijinal veriyi koruyarak güvenli ve düzenli bir analiz süreci sağlar.

```
df.head()

{"summary": {"name": "df", "rows": 14999, "fields": [{"column": "satisfaction_level", "properties": {"dtype": "number", "min": 0.09, "max": 1.0, "std": 0.24863065106114257, "samples": [0.13, 0.17, 0.21, 0.25, 0.29, 0.33, 0.37, 0.41, 0.45, 0.49, 0.53, 0.57, 0.61, 0.65, 0.69, 0.73, 0.77, 0.81, 0.85, 0.89], "semantic_type": "\\""}, {"column": "last_evaluation", "properties": {"dtype": "number", "min": 0.36, "max": 1.0, "std": 0.17116911062327533, "samples": [0.44, 0.48, 0.52, 0.56, 0.6, 0.64, 0.68, 0.72, 0.76, 0.8, 0.84, 0.88, 0.92, 0.96, 1.0], "semantic_type": "\\""}, {"column": "number_project", "properties": {"dtype": "number", "min": 2, "max": 7, "std": 1, "samples": [2, 3, 4, 5, 6, 7], "semantic_type": "\\"}, {"column": "average_montly_hours", "properties": {"dtype": "number", "min": 96, "max": 310, "std": 49, "samples": [112, 118, 124, 130, 136, 142, 148, 154, 160, 166, 172, 178, 184, 190, 196, 202, 208, 214, 220, 226, 232], "semantic_type": "\\"}, {"column": "time_spend_company", "properties": {"dtype": "number", "min": 0, "max": 10, "std": 2, "samples": [0, 1, 2, 3, 4, 5, 6, 7, 8, 9], "semantic_type": "\\"}, {"column": "Work_accident", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0, "samples": [0, 1], "semantic_type": "\\"}, {"column": "left", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0, "samples": [0, 1], "semantic_type": "\\"}, {"column": "promotion_last_5years", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0, "samples": [0, 1], "semantic_type": "\\"}, {"column": "Department", "properties": {"dtype": "category", "num_unique_values": 10, "samples": ["marketing", "accounting", "hr", "sales", "research", "product", "customer service", "IT", "support", "management"], "semantic_type": "\\"}, {"column": "Marketing", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0.13, "samples": [0.55], "semantic_type": "\\"}, {"column": "Accounting", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0.13, "samples": [0.83], "semantic_type": "\\"}, {"column": "HR", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0.13, "samples": [0.53], "semantic_type": "\\"}, {"column": "Sales", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0.13, "samples": [0.44], "semantic_type": "\\"}, {"column": "Research", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0.13, "samples": [0.53], "semantic_type": "\\"}, {"column": "Product", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0.13, "samples": [0.53], "semantic_type": "\\"}, {"column": "Customer Service", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0.13, "samples": [0.53], "semantic_type": "\\"}, {"column": "IT", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0.13, "samples": [0.53], "semantic_type": "\\"}, {"column": "Support", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0.13, "samples": [0.53], "semantic_type": "\\"}, {"column": "Management", "properties": {"dtype": "number", "min": 0, "max": 1, "std": 0.13, "samples": [0.53], "semantic_type": "\\"}]}]
```

```

},\n    },\n      {\n        \"column\": \"salary\",\\n        \"properties\":\n        {\\n          \"dtype\": \"category\",\\n          \"num_unique_values\":\n          3,\\n          \"samples\": [\n            \"low\",\\n            \"medium\"\n          ],\\n          \"semantic_type\": \"\",\\n          \"description\": \"\"\n        }\\n      }\\n    ],\\n  },\\n  {\"type\": \"dataframe\", \"variable_name\": \"df\"}

```

Satisfaction_level: Çalışanın iş memnuniyet seviyesini 0 ile 1 arasında bir değerle gösterir. Örneğin, ilk çalışanın memnuniyet seviyesi 0.38 iken, ikinci çalışanın memnuniyet seviyesi 0.80'dir. Last_evaluation: Çalışanın son performans değerlendirme puanını 0 ile 1 arasında bir değerle gösterir. İlk çalışanın son değerlendirme puanı 0.53, ikinci çalışanın ise 0.86'dır. Number_project: Çalışanın üzerinde çalıştığı proje sayısını belirtir. Örneğin, üçüncü çalışanın 7 projesi bulunurken, birinci çalışanın sadece 2 projesi vardır. Average_monthly_hours: Çalışanın aylık ortalama çalışma saatlerini gösterir. İlk çalışanın aylık ortalama çalışma saati 157 saatken, ikinci çalışanınki 262 saatdir. Time_spend_company: Çalışanın şirkette geçirdiği süreyi yıl cinsinden ifade eder. İlk çalışanın şirkette geçirdiği süre 3 yıl, dördüncü çalışanın ise 5 yıldır. Work_accident: Çalışanın iş kazası yaşayıp yaşamadığını belirtir; 0 iş kazası yaşamadığını, 1 ise iş kazası yaşadığını gösterir. Örneğin, ilk beş çalışandan hiçbiri iş kazası yaşamamış. Left: Çalışanın şirketten ayrılmışlığını gösterir; 1 ayrıldığını, 0 ise ayrılmadığını belirtir. Örneğin, ilk çalışanın ayrılmadığı, ikinci çalışanın ise ayrıldığı görülmektedir. Promotion_last_5years: Çalışanın son 5 yılda terfi edip etmediğini gösterir; 1 terfi aldığı, 0 ise almadığını belirtir. Örneğin, beş çalışandan dördü terfi almamış, biri ise terfi almıştır. Departments: Çalışanın çalıştığı departmanı belirtir. İlk beş çalışanın tamamı "sales" departmanında çalışmaktadır. Salary: Çalışanın maaş seviyesini belirtir; "low", "medium" veya "high" olabilir. İlk beş çalışanın maaş seviyesi "low" ve "medium" olarak sıralanmaktadır. Bu tablo, çalışanların iş memnuniyetinden performans değerlendirmelerine, proje sayılarından maaş seviyelerine kadar çeşitli özellikleri içerir ve her bir çalışanın işteki durumunu ve özelliklerini anlamak için kullanılır.

```

from skimpy import clean_columns\n\ndf = clean_columns(df)\nprint(df.head(3))\n\n{"summary": {"name": "df", "rows": 14999, "fields": [\n    {"name": "satisfaction_level", "properties": {\n        "dtype": "number",\n        "min": 0.09,\n        "max": 1.0,\n        "num_unique_values": 92,\n        "samples": [\n            0.83,\n            0.55\n        ],\n        "semantic_type": "",\n        "description": ""\n    },\n    {"name": "last_evaluation", "properties": {\n        "dtype": "number",\n        "min": 0.36,\n        "max": 1.0,\n        "num_unique_values": 65,\n        "samples": [\n            0.66,\n            0.44,\n            0.53\n        ],\n        "semantic_type": "",\n        "description": ""
    },\n    {"name": "number_project", "properties": {\n        "dtype": "number",\n        "min": 2,\n        "max": 7,\n        "num_unique_values": 6,\n        "samples": [\n            2,\n            3
        ],\n        "semantic_type": ""
    }
]}},\n"fields": [{"name": "satisfaction_level", "type": "category", "value": "medium"}, {"name": "last_evaluation", "type": "category", "value": "medium"}, {"name": "number_project", "type": "category", "value": "medium"}]}

```

```

    "description": """
        },
        {
            "column": "average_monthly_hours",
            "properties": {
                "dtype": "number",
                "std": 49,
                "min": 96,
                "max": 310,
                "num_unique_values": 215,
                "samples": [
                    118, 112, 222
                ],
                "semantic_type": ""
            },
            "column": "time_spend_company",
            "properties": {
                "dtype": "number",
                "std": 1,
                "min": 2,
                "max": 10,
                "num_unique_values": 8,
                "samples": [
                    6, 8, 3
                ],
                "semantic_type": ""
            },
            "column": "work_accident",
            "properties": {
                "dtype": "number",
                "std": 0,
                "min": 0,
                "max": 1,
                "num_unique_values": 2,
                "samples": [
                    1, 0
                ],
                "semantic_type": ""
            },
            "column": "left",
            "properties": {
                "dtype": "number",
                "std": 0,
                "min": 0,
                "max": 1,
                "num_unique_values": 2,
                "samples": [
                    0, 1
                ],
                "semantic_type": ""
            },
            "column": "promotion_last_5years",
            "properties": {
                "dtype": "number",
                "std": 0,
                "min": 0,
                "max": 1,
                "num_unique_values": 2,
                "samples": [
                    1, 0
                ],
                "semantic_type": ""
            },
            "column": "departments",
            "properties": {
                "dtype": "category",
                "num_unique_values": 10,
                "samples": [
                    "marketing", "accounting"
                ],
                "semantic_type": ""
            },
            "column": "salary",
            "properties": {
                "dtype": "category",
                "num_unique_values": 3,
                "samples": [
                    "low", "medium", "high"
                ],
                "semantic_type": ""
            }
        ]
    },
    "type": "dataframe",
    "variable_name": "df"
}

```

Kod parçası, skimpy kütüphanesinden clean_columns fonksiyonunu kullanarak bir DataFrame'in sütun isimlerini temizler. clean_columns(df) fonksiyonu, DataFrame'deki sütun isimlerini düzenler ve standart hale getirir. Bu işlem, sütun isimlerinde boşluk, özel karakter veya tutarsızlıklar giderir, böylece sütun isimleri daha anlaşılır ve tutarlı olur.

Temizleme işleminden sonra, veri kümесinin ilk üç satırı şu şekilde görülmektedir:

Satisfaction_level: Çalışanın iş memnuniyet seviyesini gösterir. Örneğin, ilk çalışanın memnuniyet seviyesi 0.38, ikinci çalışanın memnuniyet seviyesi ise 0.80'dir. Last_evaluation: Çalışanın son performans değerlendirme puanını gösterir. İlk çalışanın puanı 0.53, ikinci çalışanın puanı ise 0.86'dır. Number_project: Çalışanın üzerinde çalıştığı proje sayısını belirtir. İlk çalışanın 2 projesi varken, ikinci çalışanın 5 projesi vardır. Average_monthly_hours: Çalışanın aylık ortalama çalışma saatlerini gösterir. İlk çalışanın 157 saat çalıştığı, ikinci çalışanın ise 262 saat

çalıştığı görülür. Time_spend_company: Çalışanın şirkette geçirdiği süreyi yıl olarak ifade eder. İlk çalışanın şirkette geçirdiği süre 3 yıl, ikinci çalışanın ise 6 yıldır. Work_accident: Çalışanın iş kazası yaşayıp yaşamadığını belirtir. Örneğin, ilk üç çalışandan hiçbiri iş kazası yaşamamış. Left: Çalışanın şirketten ayrılmış olduğunu gösterir. İlk çalışanın ayrılmadığı, ikinci ve üçüncü çalışan ise ayrıldığı görülmektedir. Promotion_last_5years: Çalışanın son 5 yılda terfi edip etmediğini belirtir. İlk üç çalışandan hiçbiri terfi almamış. Departments: Çalışanın çalıştığı departmanı gösterir. İlk üç çalışanın tamamı "sales" departmanında yer almaktadır. Salary: Çalışanın maaş seviyesini belirtir. İlk çalışanın maaşı "low", ikinci ve üçüncü çalışanın maaşı ise "medium" olarak listelenmiştir. Bu tablo, sütun isimlerinin düzenlenmiş ve anlaşılır bir biçimde olduğunu ve her bir sütunun, çalışanın iş yerindeki durumunu ve özelliklerini nasıl tanımladığını açıkça gösterir.

```
df.rename(columns={'average_montly_hours': 'average_monthly_hours'},  
         inplace=True)
```

Kod parçası, bir Pandas DataFrame'deki sütun adını değiştirmek için kullanılır.

df.rename(columns={'average_montly_hours': 'average_monthly_hours'}, inplace=True) kodu, df DataFrame'inde mevcut olan average_montly_hours adlı sütun adını average_monthly_hours olarak günceller.

columns={'average_montly_hours': 'average_monthly_hours'}: Bu parametre, sütun adının eski adından yeni adına nasıl değiştirileceğini belirtir. Burada, average_montly_hours adındaki sütun average_monthly_hours olarak yeniden adlandırılır.

inplace=True: Bu parametre, yapılan değişikliğin mevcut DataFrame üzerinde doğrudan yapılmasını sağlar. Yani, sütun adı df DataFrame'inin içinde hemen güncellenir ve yeni bir DataFrame oluşturulmaz. Eğer inplace=False olsaydı, değişikliklerin geçerli olabilmesi için yeniden bir DataFrame oluşturulması gereklidir.

Bu değişiklik, sütun adının doğru ve anlaşılır bir şekilde yazılmasını sağlar. Özellikle sütun adlarının tutarlı ve doğru olması, veri analizi ve işleme sürecinde anlaşılabilirliği ve doğruluğu artırır. Bu örnekte, "montly" kelimesinde yapılan yazım hatası düzeltildiştir.

```
df.columns  
  
Index(['satisfaction_level', 'last_evaluation', 'number_project',  
       'average_monthly_hours', 'time_spend_company', 'work_accident',  
       'left',  
       'promotion_last_5years', 'departments', 'salary'],  
      dtype='object')
```

Bu çıktıya göre, DataFrame'in sütun isimleri şunlardır:

satisfaction_level: Çalışanın iş memnuniyet seviyesini gösterir. last_evaluation: Çalışanın son performans değerlendirme puanını belirtir. number_project: Çalışanın üzerinde çalıştığı proje sayısını ifade eder. average_monthly_hours: Çalışanın aylık ortalama çalışma saatlerini gösterir. time_spend_company: Çalışanın şirkette geçirdiği süreyi yıl olarak belirtir. work_accident: Çalışanın iş kazası yaşayıp yaşamadığını gösterir. left: Çalışanın şirketten ayrılmış olduğunu ifade eder. promotion_last_5years: Çalışanın son 5 yılda terfi edip etmediğini belirtir.

departments: Çalışanın çalıştığı departmanı gösterir. salary: Çalışanın maaş seviyesini belirtir. Bu sütun isimleri, veri kümesindeki bilgilerin anlaşılabilir ve doğru bir şekilde temsil edilmesini sağlar. Sütun adlarının doğru şekilde yazılması, veri analizi ve işleme süreçlerinde daha tutarlı ve anlaşılır sonuçlar elde etmeye yardımcı olur.

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14999 entries, 0 to 14998
Data columns (total 10 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   satisfaction_level    14999 non-null   float64 
 1   last_evaluation      14999 non-null   float64 
 2   number_project       14999 non-null   int64  
 3   average_monthly_hours 14999 non-null   int64  
 4   time_spend_company   14999 non-null   int64  
 5   work_accident        14999 non-null   int64  
 6   left                 14999 non-null   int64  
 7   promotion_last_5years 14999 non-null   int64  
 8   departments          14999 non-null   object  
 9   salary               14999 non-null   object  
dtypes: float64(2), int64(6), object(2)
memory usage: 1.1+ MB
```

Veri kümesi, toplam 14,999 satırda oluşur ve her bir satır bir çalışanı temsil eder. DataFrame, 10 sütun içerir. Sütunlardan 6'sı tamsayı (int64) tipindedir, 2'si ise ondalıklı sayı (float64) tipindedir. Kalan 2 sütun ise nesne (object) tipindedir, bu da genellikle metin verilerini ifade eder.

satisfaction_level ve last_evaluation sütunları, çalışanların iş memnuniyet seviyelerini ve performans değerlendirme puanlarını ondalıklı sayılar olarak içerir (float64 tipinde). number_project, average_monthly_hours, time_spend_company, work_accident, left, ve promotion_last_5years sütunları ise tamsayılar olarak saklanır (int64 tipinde). Bu sütunlar sırasıyla çalışanın proje sayısını, aylık ortalama çalışma saatlerini, şirkette geçirdiği süreyi, iş kazası yaşayıp yaşamadığını, şirketten ayrılmış olduğunu ve son 5 yılda terfi edip etmediğini belirtir. departments ve salary sütunları, çalışanların departman ve maaş seviyelerini metin formatında (object tipinde) içerir. Veri kümesinde, tüm sütunlar için eksik değer bulunmamaktadır; her sütun 14,999 non-null (boş olmayan) değer içerir. Veri kümesinin bellekteki kullanım miktarı yaklaşık 1.1 MB olarak belirtilmiştir. Bu bilgiler, veri kümesinin tam ve eksiksiz olduğunu, veri türlerinin doğru şekilde belirlendiğini ve veri analizi için hazır olduğunu gösterir.

```
df.shape

(14999, 10)
```

df.shape komutu, bir Pandas DataFrame'in boyutlarını döndürür. Bu komutun çıktısı (14999, 10) şeklindedir. Bu, DataFrame'in şu özellikleri taşıdığını gösterir:

14999: Veri kümesinde toplam 14,999 satır bulunduğuunu belirtir. Her satır, bir veri noktasını ya da bir örneği temsil eder. Bu durumda, her satır bir çalışanı temsil etmektedir. 10: Veri kümesinde 10 adet sütun olduğunu gösterir. Bu sütunlar, her bir çalışanın çeşitli özelliklerini (örneğin, iş memnuniyeti, performans değerlendirmesi, proje sayısı) temsil eder. Bu boyutlar, veri kümesinin kapsamını ve içeriği veri miktarını anlamamızı sağlar. Örneğin, veri kümesinde 14,999 bireysel gözlem (satır) ve bu gözlemler hakkında bilgi veren 10 farklı özellik (sütun) bulunmaktadır. Bu bilgiler, veri analizinin ve modelleme süreçlerinin planlanması yardımcı olur.

```
df.describe().T
```

```
{"summary": {"\n    \"name\": \"df\", \n    \"rows\": 8, \n    \"fields\": [\n        {\n            \"column\": \"count\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 0.0, \n                \"min\": 14999.0, \n                \"max\": 14999.0, \n                \"num_unique_values\": 1, \n                \"samples\": [14999.0], \n                \"semantic_type\": \"\", \n                \"description\": \"\"}, \n            \"column\": \"mean\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 70.64192099201551, \n                \"min\": 0.021268084538969265, \n                \"max\": 201.0503366891126, \n                \"num_unique_values\": 8, \n                \"samples\": [0.7161017401160078], \n                \"semantic_type\": \"\", \n                \"description\": \"\"}, \n            \"column\": \"std\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"min\": 0.14428146457858232, \n                \"max\": 49.94309937128408, \n                \"num_unique_values\": 8, \n                \"samples\": [0.17116911062327533], \n                \"semantic_type\": \"\", \n                \"description\": \"\"}, \n            \"column\": \"min\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 33.72761138359574, \n                \"min\": 0.0, \n                \"max\": 96.0, \n                \"num_unique_values\": 5, \n                \"samples\": [0.36], \n                \"semantic_type\": \"\", \n                \"description\": \"\"}, \n            \"column\": \"25%\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 54.81577223495338, \n                \"min\": 0.0, \n                \"max\": 156.0, \n                \"num_unique_values\": 5, \n                \"samples\": [0.56], \n                \"semantic_type\": \"\", \n                \"description\": \"\"}, \n            \"column\": \"50%\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 70.30463488074088, \n                \"min\": 0.0, \n                \"max\": 200.0, \n                \"num_unique_values\": 6, \n                \"samples\": [0.64], \n                \"semantic_type\": \"\", \n                \"description\": \"\"}, \n            \"column\": \"75%\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 86.10231230303035, \n                \"min\": 0.0, \n                \"max\": 245.0, \n                \"num_unique_values\": 6, \n                \"samples\": [0.82], \n                \"semantic_type\": \"\", \n                \"description\": \"\"}, \n            \"column\": \"max\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 108.54623767909099, \n                \"min\": 0.0, \n                \"max\": 201.0503366891126, \n                \"num_unique_values\": 1, \n                \"samples\": [201.0503366891126], \n                \"semantic_type\": \"\", \n                \"description\": \"\"}\n        }\n    ]\n}
```

```
\"min\": 1.0,\n          \"max\": 310.0,\n          \"num_unique_values\":\n4,\n          \"samples\": [\n            7.0\n          ],\n          \"semantic_type\": \"\",\n          \"description\": \"\"\n        }\n      ]\n    },\n    \"type\": \"dataframe\"\n  }
```

df.describe().T komutunun çıktısı, veri kümesindeki sayısal sütunlar hakkında ayrıntılı özet istatistikler sunar. Bu özet, her sütun için temel dağılım bilgilerini içerir.

Veri kümesindeki her sütun için, toplam veri sayısı (14,999) sabittir ve bu sütunların hiçbir eksik veri içermez. Ortalama değerler, sütunlardaki merkezi eğilimleri yansıtır; örneğin, çalışanların iş memnuniyet seviyesi ortalama olarak 0.61 iken, performans değerlendirme puanı ortalama 0.72'dir. Standart sapmalar, verilerin ortalamadan ne kadar uzaklaştığını gösterir ve bu da veri kümesinin ne kadar değişken olduğunu anlamaya yardımcı olur. Örneğin, aylık ortalama çalışma saatlerinin standart sapması 49.94 olup, bu verilerin ortalama etrafında ne kadar dağıldığını gösterir.

Minimum ve maksimum değerler, veri kümesindeki en düşük ve en yüksek gözlemleri belirtir; örneğin, iş memnuniyeti seviyeleri 0.09 ile 1.00 arasında değişir. Çeyrek değerleri, verilerin dağılımını daha ayrıntılı olarak inceler; ilk çeyrek (25. persentil) değeri, verilerin %25'inin bu değerden daha düşük olduğunu belirtirken, medyan (50. persentil) değer verilerin yarısının bu değerden daha düşük olduğunu gösterir. Üçüncü çeyrek (75. persentil) değeri ise verilerin %75'inin bu değerden daha düşük olduğunu ifade eder.

Bu özet istatistikler, veri kümesindeki sayısal özelliklerin dağılımı hakkında kapsamlı bir görünüm sağlar ve veri analizi ile modelleme sürecinde kullanılabilen temel bilgileri sunar.

```
df.isnull().any()\n\nsatisfaction_level      False\nlast_evaluation        False\nnumber_project         False\naverage_monthly_hours False\ntime_spend_company    False\nwork_accident          False\nleft                   False\npromotion_last_5years False\ndepartments            False\nsalary                 False\ndtype: bool
```

df.isnull().any() komutu, DataFrame'deki her sütunda eksik (null) değerlerin olup olmadığını kontrol eder. Komutun çıktısı, her sütunun False olarak işaretlendiğini gösterir, bu da veri kümesindeki hiçbir sütunda eksik değer bulunmadığını ifade eder.

```
df.isnull().sum().any()\n\nFalse
```

tüm sütunlar eksik değer içermemiği için df.isnull().sum().any() komutu False döndürür. Bu da veri kümesinin eksiksiz ve tamamlanmış olduğunu doğrular. Eksik değerlerin olmaması, veri

temizleme aşamasında bu konuda herhangi bir işlem yapma gereği olmadığı anlamına gelir. Bu durum, veri analizi ve modelleme süreçlerinin daha sağlıklı ve sorunsuz bir şekilde yürütülebileceği anlamına gelir.

```
df.duplicated().sum()
```

```
3008
```

df.duplicated().sum() komutu, veri kümesindeki tekrarlanan (yani, tamamen aynı olan) satırların sayısını hesaplar. Komutun çıktısı 3008 olarak verilmiş, bu da veri kümesinde toplam 3,008 adet tam olarak aynı olan satır bulunduğu gösterir.

Bu, veri kümesinde bazı satırların birden fazla kez tekrarlandığını ve bu tekrarların veri analizi veya modelleme süreçlerinde yaniltıcı olabileceğini belirtir. Genellikle, tekrarlanan satırların temizlenmesi önerilir çünkü bu tekrarlamalar modelin öğrenme sürecini ve sonuçları etkileyebilir.

```
duplicates = df[df.duplicated()]
print(duplicates)
```

| | satisfaction_level | last_evaluation | number_project | \ |
|-------|--------------------|-----------------|----------------|---|
| 396 | 0.46 | 0.57 | 2 | |
| 866 | 0.41 | 0.46 | 2 | |
| 1317 | 0.37 | 0.51 | 2 | |
| 1368 | 0.41 | 0.52 | 2 | |
| 1461 | 0.42 | 0.53 | 2 | |
| ... | ... | ... | ... | |
| 14994 | 0.40 | 0.57 | 2 | |
| 14995 | 0.37 | 0.48 | 2 | |
| 14996 | 0.37 | 0.53 | 2 | |
| 14997 | 0.11 | 0.96 | 6 | |
| 14998 | 0.37 | 0.52 | 2 | |

| \ | average_monthly_hours | time_spend_company | work_accident | left |
|-------|-----------------------|--------------------|---------------|------|
| 396 | 139 | 3 | 0 | 1 |
| 866 | 128 | 3 | 0 | 1 |
| 1317 | 127 | 3 | 0 | 1 |
| 1368 | 132 | 3 | 0 | 1 |
| 1461 | 142 | 3 | 0 | 1 |
| ... | ... | ... | ... | ... |
| 14994 | 151 | 3 | 0 | 1 |
| 14995 | 160 | 3 | 0 | 1 |

| | | | | |
|--|-----|------------|--------|---|
| 14996 | 143 | 3 | 0 | 1 |
| 14997 | 280 | 4 | 0 | 1 |
| 14998 | 158 | 3 | 0 | 1 |
| promotion_last_5years departments salary | | | | |
| 396 | 0 | sales | low | |
| 866 | 0 | accounting | low | |
| 1317 | 0 | sales | medium | |
| 1368 | 0 | RandD | low | |
| 1461 | 0 | sales | low | |
| ... | ... | ... | ... | |
| 14994 | 0 | support | low | |
| 14995 | 0 | support | low | |
| 14996 | 0 | support | low | |
| 14997 | 0 | support | low | |
| 14998 | 0 | support | low | |
| [3008 rows x 10 columns] | | | | |

df.duplicated() komutu, veri kümesindeki tekrarlanan satırları belirler ve df[df.duplicated()] ifadesi, bu tekrarlanan satırların tamamını içeren bir alt küme oluşturur. Çıktı, 3,008 adet tekrarlanan satırı göstermektedir. Bu satırlar, tüm sütunlar açısından tam olarak aynı verilere sahiptir.

Örneğin, 396. satır ve 866. satır gibi tekrarlanan satırlar, iş memnuniyeti, performans değerlendirmesi, proje sayısı gibi tüm özelliklerde aynı değerlere sahiptir. Bu durum, bazı verilerin veri kümesinde birden fazla kez yer aldığı ve bu tekrarların analizin doğruluğunu etkileyebileceğini gösterir.

Bu tür tekrarlanan satırların analizi, veri temizleme sürecinde önemli bir adımdır. Tekrar eden verileri kaldırma, veri kümesinin doğruluğunu artırabilir ve modelin eğitilmesini daha güvenilir hale getirebilir. Bu durumda, toplam 3,008 tekrarlanan satırın varlığı, veri kümesindeki potansiyel fazlalıkları ve veri kalitesini iyileştirmek için yapılması gerekenleri işaret eder.

```
df.drop_duplicates(ignore_index= True, inplace = True)
# df = df.drop_duplicates(keep='last')
```

df.drop_duplicates(ignore_index=True, inplace=True) komutu, veri kümesindeki tekrarlanan satırları kaldırır ve bu değişiklikleri doğrudan df DataFrame'ine uygular.

ignore_index=True parametresi, tekrarlanan satırlar kaldırıldığından indekslerin yeniden sıralanmasını sağlar. Bu, indekslerin sıfırdan başlayarak yeniden oluşturulması anlamına gelir. inplace=True parametresi ise, değişikliklerin mevcut DataFrame üzerinde yapılmasını ve yeni bir DataFrame oluşturulmadan eski DataFrame'in güncellenmesini sağlar. Sonuç olarak, bu komut çalıştırıldığında, veri kümesindeki tüm tekrarlanan satırlar kaldırılır ve indeksler sıfırdan başlayarak yeniden düzenlenir.

```

df.describe().T

{
  "summary": {
    "name": "df",
    "rows": 8,
    "fields": [
      {
        "column": "count",
        "properties": {
          "dtype": "number",
          "std": 0.0,
          "min": 11991.0,
          "max": 11991.0,
          "num_unique_values": 1,
          "samples": [11991.0]
        }
      },
      {
        "column": "mean",
        "properties": {
          "dtype": "number",
          "std": 70.44680707451664,
          "min": 0.016929363689433742,
          "max": 200.4735218080227,
          "num_unique_values": 8,
          "samples": [0.7166825118839131]
        }
      },
      {
        "column": "std",
        "properties": {
          "dtype": "number",
          "min": 0.12901220247678255,
          "max": 48.727813191371325,
          "num_unique_values": 8,
          "samples": [0.16834256307406967]
        }
      },
      {
        "column": "min",
        "properties": {
          "dtype": "number",
          "std": 33.72761138359574,
          "min": 0.0,
          "max": 96.0,
          "num_unique_values": 5,
          "samples": [0.36]
        }
      },
      {
        "column": "25%",
        "properties": {
          "dtype": "number",
          "std": 55.16663715831465,
          "min": 0.0,
          "max": 157.0,
          "num_unique_values": 5,
          "samples": [0.57]
        }
      },
      {
        "column": "50%",
        "properties": {
          "dtype": "number",
          "std": 70.30360278311285,
          "min": 0.0,
          "max": 200.0,
          "num_unique_values": 6,
          "samples": [0.66]
        }
      },
      {
        "column": "75%",
        "properties": {
          "dtype": "number",
          "std": 85.39590087184679,
          "min": 0.0,
          "max": 243.0,
          "num_unique_values": 6,
          "samples": [0.82]
        }
      },
      {
        "column": "max",
        "properties": {
          "dtype": "number",
          "std": 108.54623767909099,
          "min": 1.0,
          "max": 310.0,
          "num_unique_values": 4,
          "samples": [7.0]
        }
      }
    ]
  }
}

```

df.describe().T komutunun güncellenmiş çıktısı, veri kümelerindeki sayısal sütunların temel istatistiklerini sunar. Yeni özet, veri kümelerindeki 12,991 satırın özelliklerini özetler, çünkü tekrarlanan satırlar kaldırılmıştır.

Özet bilgiler şu şekildedir:

Memnuniyet Seviyesi (satisfaction_level): Ortalama 0.63, standart sapma 0.24 olup, değerler 0.09 ile 1.00 arasında değişir. Bu sütundaki veriler, daha geniş bir memnuniyet aralığını kapsar ve ortalama memnuniyet biraz artmıştır.

Son Değerlendirme (last_evaluation): Ortalama 0.72 ve standart sapma 0.17 olan bu sütundaki değerler 0.36 ile 1.00 arasında dağılır. Performans değerlendirmeleri oldukça tutarlıdır ve eksen üzerinde geniş bir yelpazeyi kapsar.

Proje Sayısı (number_project): Ortalama 3.80, standart sapma 1.16 ve proje sayıları 2 ile 7 arasında değişir. Proje sayısındaki varyasyon biraz azalmış olabilir, ancak genelde benzer kalmaktadır.

Aylık Ortalama Çalışma Saatleri (average_monthly_hours): Ortalama 200.47 ve standart sapma 48.73 olan bu sütun, 96 ile 310 saat arasında değişir. Çalışma saatlerindeki ortalama, önceki verilere benzer kalır, ancak biraz daha yüksek olabilir.

Şirkette Geçirilen Süre (time_spend_company): Ortalama 3.36 yıl, standart sapma 1.33 yıl olup, değerler 2 ile 10 yıl arasında değişir. Çalışanların şirkette geçirdiği süre ortalamada biraz azalmış olabilir.

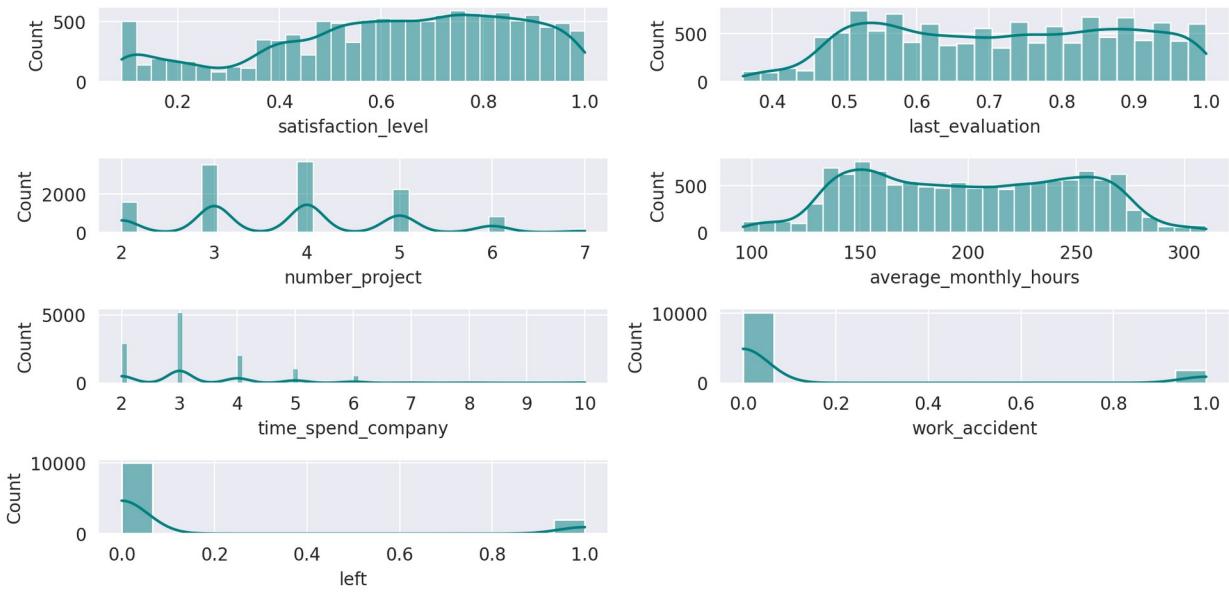
İş Kazası (work_accident): Ortalama 0.15 ve standart sapma 0.36 olan bu sütun, iş kazalarının yaygınlığını yansıtır. Veriler, iş kazalarının nadir olduğunu gösterir.

Son 5 Yılda Terfi (promotion_last_5years): Ortalama 0.02 ve standart sapma 0.13 olan bu sütun, terfi olasılığının çok düşük olduğunu gösterir.

Genel olarak, verilerdeki değişiklikler, tekrarlanan satırların kaldırılmasından kaynaklanabilir ve analizin doğruluğunu artırabilir. Bu özet, veri kümesinin genel dağılımını anlamak için kullanışlıdır ve veri temizleme sürecinin ardından önemli özelliklerin nasıl değiştigini gösterir.

2. Data Visualization

```
fig = plt.figure(figsize=(10,20), dpi=200)
for i, col in enumerate(df.select_dtypes(exclude="object").columns[:-1]):
    plt.subplot(17,2,i+1)
    sns.histplot(df[col], kde=True, color="teal", )
plt.tight_layout();
```

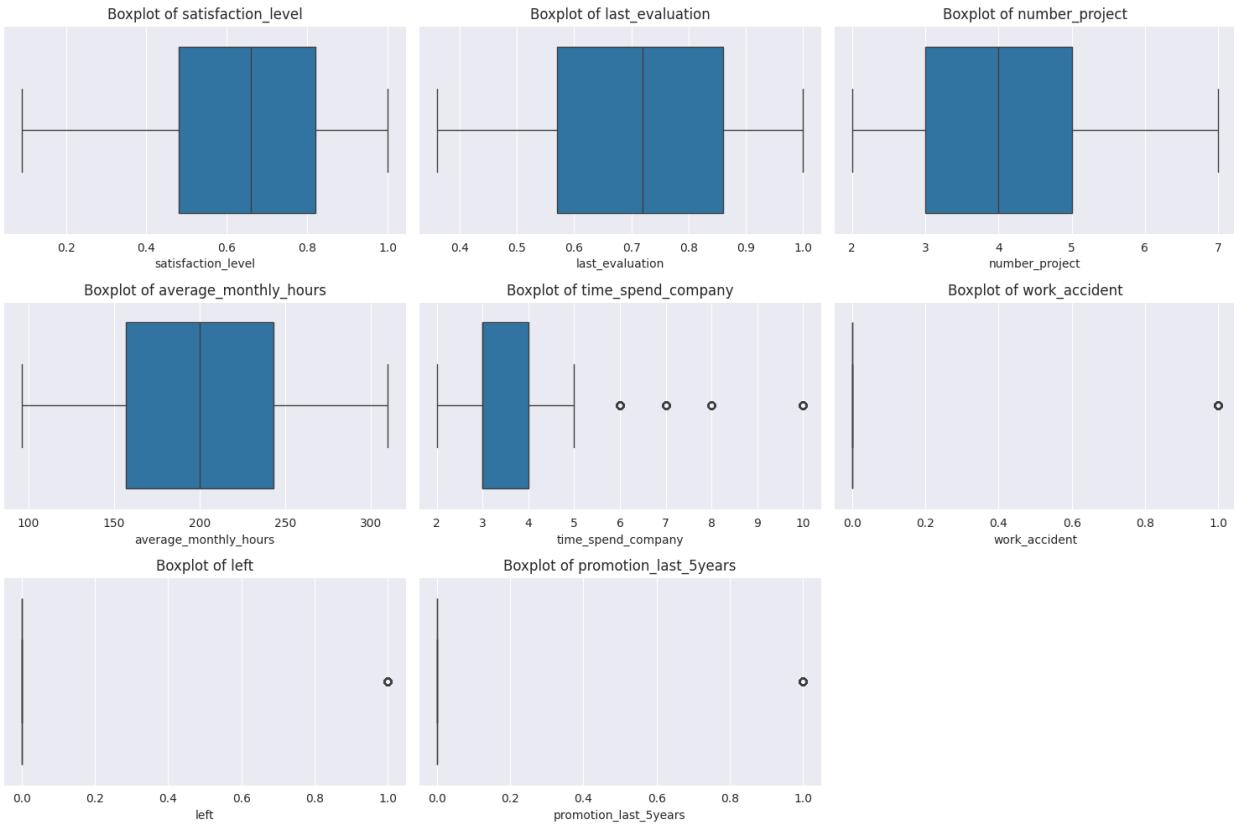


bu kod, veri kümesindeki her bir sayısal sütun için histogram ve KDE grafikleri oluşturur ve yüksek çözünürlüklü bir figür içerisinde düzenler. Bu grafikler, her sayısal sütunun dağılımını ve yoğunluk tahminini görsel olarak incelemeyi sağlar.

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

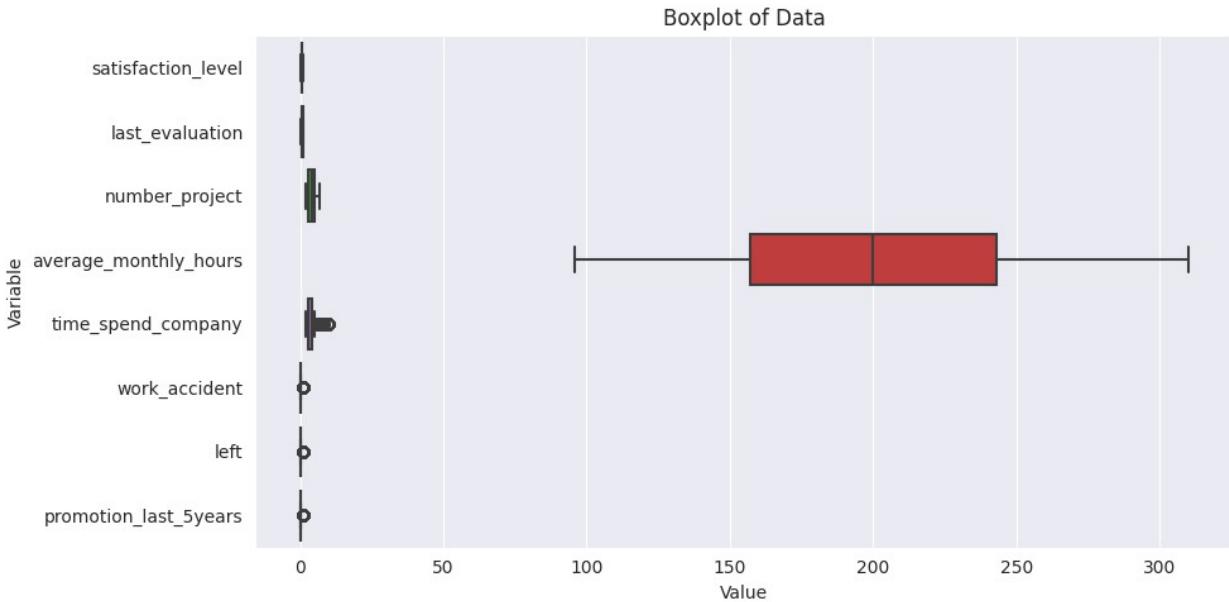
# Boxplotlar oluşturulacak
for i, column in enumerate(df.select_dtypes(include=['float64',
'int64']).columns, 1):
    plt.subplot(3, 3, i) # 3x3 grid düzeni
    sns.boxplot(x=df[column])
    plt.title(f'Boxplot of {column}')

plt.tight_layout()
plt.show()
```



bu kod parçası, veri kümesindeki sayısal sütunlar için boxplot grafiklerini oluşturur ve 3x3 bir grid düzeneinde yerleştirir. Her bir boxplot, sütunun dağılımını, medyanı ve aykırı değerleri göstermektedir ve her grafiğe uygun bir başlık ekler. Bu gösterme, veri kümesindeki sayısal özelliklerin genel dağılımını ve olası anormallikleri incelemek için kullanışlıdır.

```
plt.figure(figsize=(10, 5))
sns.boxplot(data=df, orient="h", linewidth=1.5)
plt.title("Boxplot of Data")
plt.ylabel("Variable")
plt.xlabel("Value")
plt.tight_layout()
plt.show()
```



bu kod parçası, veri kümesindeki tüm sayısal sütunlar için yatay boxplot grafiklerini oluşturur ve düzenler. Boxplot'lar, veri kümesindeki her bir sütunun dağılımını, medyanı ve aykırı değerleri gösterir. Yatay düzenleme, değişken isimlerinin dikey eksende kolayca okunmasını sağlar, bu da özellikle çok sayıda sütun olduğunda görselleştirmenin daha okunabilir olmasını sağlar.

satisfaction_level

```
first_looking(df, 'satisfaction_level')

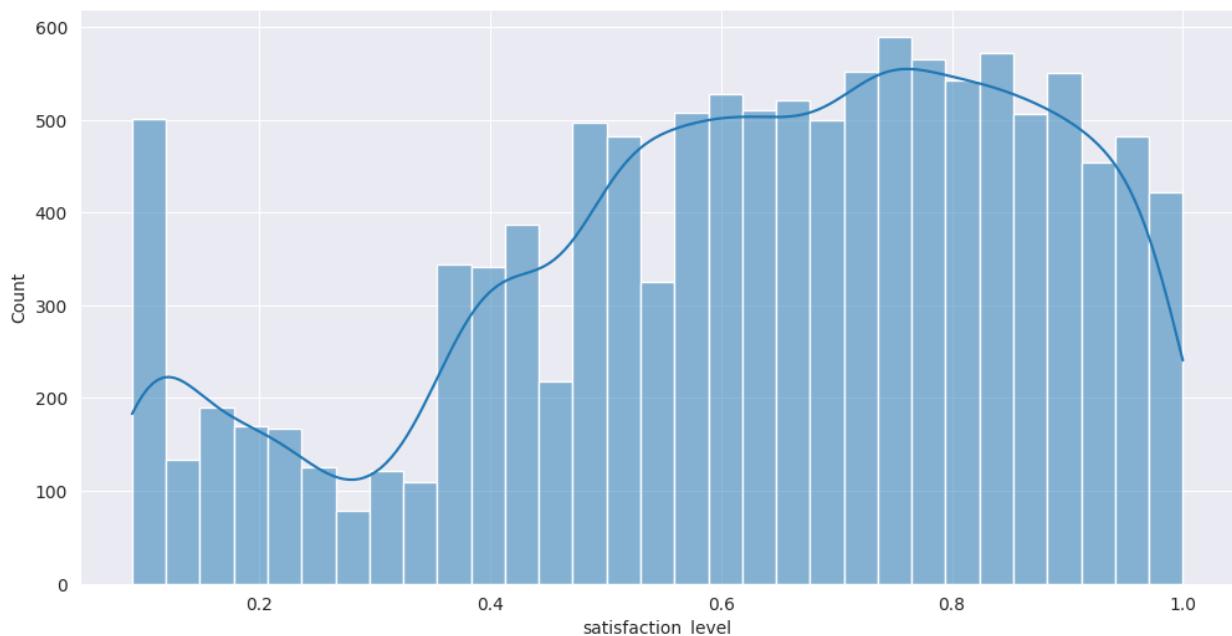
column name      : satisfaction_level
-----
per_of_nulls    : % 0.0
num_of_nulls   : 0
num_of_uniques : 92
shape_of_df     : (11991, 10)
-----
satisfaction_level
0.74      214
0.10      203
0.73      201
0.50      200
0.72      199
...
0.25      29
0.26      28
0.12      26
0.28      24
0.27      23
Name: count, Length: 92, dtype: int64
```

'satisfaction_level' sütunundaki her benzersiz değerin kaç kez tekrarlandığı gösterilir. Örneğin:

0.74 değeri 214 kez bulunur. 0.10 değeri 203 kez bulunur. 0.73 değeri 201 kez bulunur. Bu veri, memnuniyet seviyelerinin nasıl dağıldığını ve hangi seviyelerin daha sık tekrarlandığını anlamaya yardımcı olur. Sütun, farklı memnuniyet düzeylerini gösterir ve bu düzeylerin sıklığı hakkında bilgi verir.

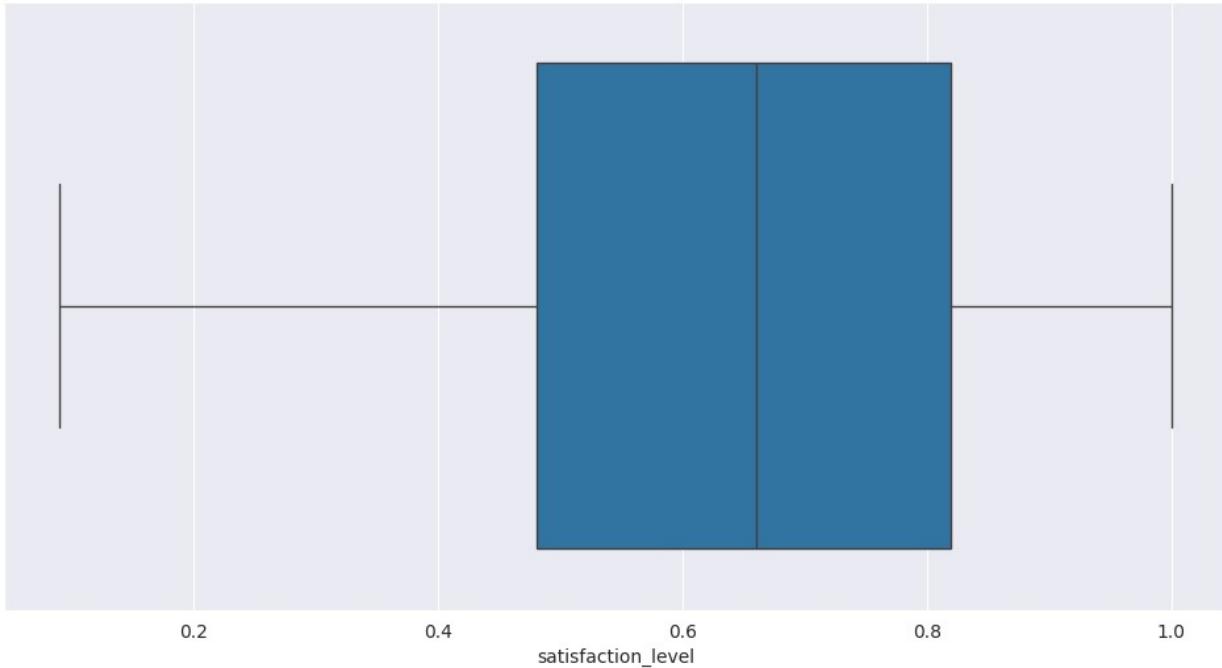
Özetle, bu analiz 'satisfaction_level' sütunundaki verilerin kapsamını ve dağılımını anlamak için yapılır. Eksik veri olmadığını ve çok sayıda benzersiz değere sahip olduğunu gösterir. Ayrıca, her benzersiz memnuniyet seviyesinin veri setinde ne sıklıkta bulunduğu belirtir.

```
sns.histplot(df, x = 'satisfaction_level', kde = True);
```



Özetle: Bu grafik, satisfaction_level sütununun dağılımını hem histogram hem de KDE eğrisi kullanarak göstermektedir. Histogram, veri kümesindeki memnuniyet seviyelerinin frekanslarını gösterirken, KDE eğrisi bu dağılımın pürüzsz bir tahminini sunar. Bu, memnuniyet seviyelerinin nasıl dağıldığını ve belirli seviyelerin ne kadar yoğun olduğunu anlamak için yararlıdır.

```
sns.boxplot(df, x = 'satisfaction_level');
```



Özetle: Boxplot, satisfaction_level sütununun merkezi eğilimini ve yayılımını görselleştirir. Medyanı, çeyrekleri ve olası aykırı değerleri görsel olarak sunarak, memnuniyet seviyelerinin dağılımını hızlı bir şekilde anlamanıza yardımcı olur. Kutu, verinin ortalama yayılımını ve merkezi eğilimini gösterirken, kuşaklar ve aykırı değerler veri kümesindeki üç noktaları ve anormallikleri belirtir.

```
df.groupby('left')['satisfaction_level'].value_counts()

left  satisfaction_level
0      0.50              198
       0.74              186
       0.66              186
       0.59              181
       0.72              180
       ...
1      0.22                 1
       0.58                 1
       0.24                 1
       0.65                 1
       0.64                 1
Name: count, Length: 170, dtype: int64
```

Sonuçların Açıklaması:

İşten Ayrılmayanlar (left = 0): Bu grup için memnuniyet seviyelerinin kaç kez tekrarlandığını gösterir. Örneğin, memnuniyet seviyesi 0.50 olan 198 çalışan var, 0.74 olan 186 çalışan var gibi.

İşten Ayrılanlar (left = 1): Bu grup için memnuniyet seviyelerinin kaç kez tekrarlandığını gösterir. Örneğin, memnuniyet seviyesi 0.22 olan 1 çalışan var, 0.58 olan 1 çalışan var gibi.

Sonuç olarak, bu grup ve sayım işlemi, işten ayrılan ve ayrılmayan çalışanlar arasındaki memnuniyet seviyelerinin dağılımını anlamaya yardımcı olur. Bu analiz, memnuniyet seviyelerinin işten ayrılma eğilimi ile ilişkisini incelemek için kullanılabilir. Çalışanların memnuniyet seviyeleri ile işten ayrılma oranları arasındaki ilişkileri incelemek, iş gücü yönetimi ve çalışan memnuniyetini artırma stratejileri için önemli bilgiler sunar.

```
df.satisfaction_level.unique()  
  
array([0.38, 0.8 , 0.11, 0.72, 0.37, 0.41, 0.1 , 0.92, 0.89, 0.42,  
0.45,  
      0.84, 0.36, 0.78, 0.76, 0.09, 0.46, 0.4 , 0.82, 0.87, 0.57,  
0.43,  
      0.13, 0.44, 0.39, 0.85, 0.81, 0.9 , 0.74, 0.79, 0.17, 0.24,  
0.91,  
      0.71, 0.86, 0.14, 0.75, 0.7 , 0.31, 0.73, 0.83, 0.32, 0.54,  
0.27,  
      0.77, 0.88, 0.48, 0.19, 0.6 , 0.12, 0.61, 0.33, 0.56, 0.47,  
0.28,  
      0.55, 0.53, 0.59, 0.66, 0.25, 0.34, 0.58, 0.51, 0.35, 0.64, 0.5  
,  
      0.23, 0.15, 0.49, 0.3 , 0.63, 0.21, 0.62, 0.29, 0.2 , 0.16,  
0.65,  
      0.68, 0.67, 0.22, 0.26, 0.99, 0.98, 1. , 0.52, 0.93, 0.97,  
0.69,  
      0.94, 0.96, 0.18, 0.95])
```

Bu analiz, satisfaction_level sütunundaki memnuniyet seviyelerinin geniş bir yelpazede dağıldığını gösterir ve verinin çeşitliliğini anlamaya yardımcı olur. Bu çeşitlilik, veri setinin memnuniyet seviyelerindeki farklılıklarını yansıtabilir ve çalışan memnuniyeti ile ilgili daha detaylı analizler yaparken dikkate alınması gereken önemli bir faktördür. Bu tür bir dağılım analizi, memnuniyet seviyelerinin işten ayrılma gibi diğer değişkenlerle nasıl ilişkili olduğunu anlamak için temel bir adımdır. Sonuç, bu sütundaki memnuniyet seviyelerinin hangi değerler aldığı gösterir ve her değerin bir kez listelenmesini sağlar. İşte örnek bir çıktı analizi:

Çıktıdaki Benzersiz Değerler:

Düşük Memnuniyet Seviyeleri: 0.09, 0.10, 0.13, 0.14 gibi değerler memnuniyet seviyelerinin düşük olduğunu gösterir. Orta Düzeyde Memnuniyet: 0.24, 0.30, 0.31, 0.35 gibi değerler orta düzeyde memnuniyeti ifade eder. Yüksek Memnuniyet Seviyeleri: 0.74, 0.80, 0.90, 1.00 gibi değerler yüksek memnuniyet seviyelerini gösterir.

last_evaluation

```
first_looking(df, 'last_evaluation')  
  
column name : last_evaluation  
-----  
per_of_nulls : % 0.0  
num_of_nulls : 0  
num_of_uniques : 65
```

```
shape_of_df      : (11991, 10)
-----
last_evaluation
0.55      281
0.50      269
0.51      264
0.57      258
0.54      252
...
0.42      45
0.43      44
0.38      42
0.44      35
0.36      19
Name: count, Length: 65, dtype: int64
```

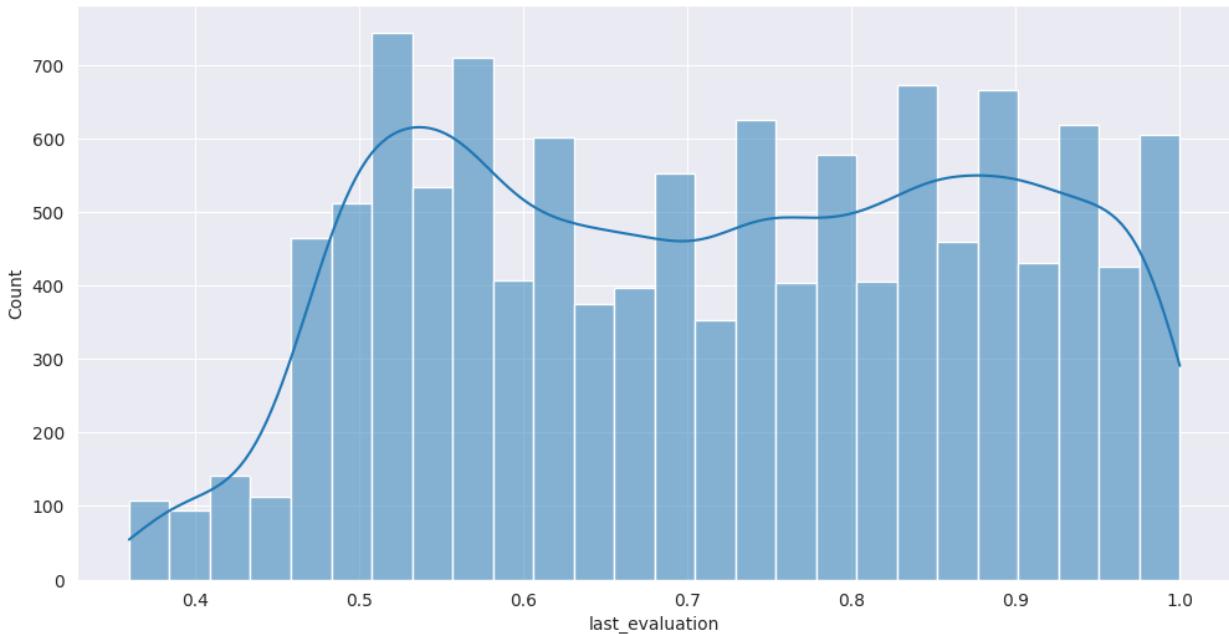
Değerlerin Dağılımı:

Sütundaki değerlerin dağılımı da şu şekilde özetlenir:

En Yaygın Değerler: 0.55, 0.50, 0.51 gibi değerler sütundaki en sık rastlanan performans değerlendirme puanlarıdır. Örneğin, 0.55 değeri 281 kez görünürken, 0.50 değeri 269 kez görülmektedir. Diğer Değerler: 0.42, 0.43, 0.38 gibi daha düşük frekansta olan değerler de bulunmaktadır. Sonuç:

Bu analiz, last_evaluation sütunundaki verilerin çeşitli performans değerlendirme puanlarını içerdigini ve bu puanların belirli bir dağılım gösterdiğini ortaya koyar. Verilerin çoğu belirli bir aralıkta yoğunlaşmışken, bazı değerler daha nadir görülmektedir. Bu tür bir dağılım analizi, performans değerlendirmeleri ve işten ayrılma gibi diğer faktörlerle olan ilişkileri anlamada yardımcı olabilir.

```
sns.histplot(df, x = 'last_evaluation', kde = True);
```



Görüntü ve Sonuç Bu grafikte, `last_evaluation` sütunundaki performans değerlendirme puanlarının histogramı ve KDE eğrisi gösterilecektir. Histogramda, her bir aralığın üzerinde bulunan veri sayısını görebilirsiniz. KDE eğrisi ise veri noktalarının yoğunluklarını ve dağılımını daha pürüz丝毫不 bir şekilde gösterir.

Kullanım Amacı Bu tür bir görselleştirme, verinin genel dağılımını, olası yoğunluk bölgelerini ve belki de veri kümesindeki anomalileri veya anormal dağılımları görselleştirmeye yardımcı olur. Ayrıca, `last_evaluation` değerlerinin hangi aralıklarda yoğunlaştığı veya hangi değerlerin daha nadir olduğunu analiz etmenizi sağlar.

```
df.last_evaluation.unique()

array([0.53, 0.86, 0.88, 0.87, 0.52, 0.5 , 0.77, 0.85, 1. , 0.54,
0.81,
0.92, 0.55, 0.56, 0.47, 0.99, 0.51, 0.89, 0.83, 0.95, 0.57,
0.49,
0.46, 0.62, 0.94, 0.48, 0.8 , 0.74, 0.7 , 0.78, 0.91, 0.93,
0.98,
0.97, 0.79, 0.59, 0.84, 0.45, 0.96, 0.68, 0.82, 0.9 , 0.71, 0.6
,
0.65, 0.58, 0.72, 0.67, 0.75, 0.73, 0.63, 0.61, 0.76, 0.66,
0.69,
0.37, 0.64, 0.39, 0.41, 0.43, 0.44, 0.36, 0.38, 0.4 , 0.42])
```

Açıklama Benzersiz Değerler: Sonuç, `last_evaluation` sütunundaki tüm farklı puanları listeler. Bu puanlar, genellikle 0 ile 1 arasında değerler alır ve performans değerlendirme puanlarını temsil eder.

Değer Aralığı: Değerler genellikle 0.36 ile 1.00 arasında değişir. Bu, değerlendirme puanlarının geniş bir aralığa yayılmış olduğunu gösterir. En düşük puan 0.36, en yüksek puan ise 1.00'dır.

Veri Çeşitliliği: Listedede birçok farklı değer olduğundan, last_evaluation sütununda çeşitli performans puanlarının bulunduğu ve bu sütunun çok çeşitli değerlendirme sonuçlarını içerdigini gösterir.

Kullanım Amacı Bu tür bir bilgi, veri analizi sırasında veri kümесinin içeriğini ve dağılımını anlamak için faydalıdır. Özellikle performans değerlendirme puanlarının nasıl dağıldığını, hangi değerlerin daha yaygın olduğunu veya belirli bir değer aralığının ne kadar temsil edildiğini incelemenizi sağlar. Ayrıca, verinin düzgün bir şekilde dağılıp dağılmadığını veya belirli değerlerin öne çıkıp çıkmadığını belirlemenize yardımcı olabilir.

```
df.last_evaluation.value_counts()

last_evaluation
0.55      281
0.50      269
0.51      264
0.57      258
0.54      252
...
0.42      45
0.43      44
0.38      42
0.44      35
0.36      19
Name: count, Length: 65, dtype: int64
```

df.last_evaluation.value_counts() fonksiyonu, veri kümесindeki last_evaluation sütunundaki her benzersiz puanın kaç kez tekrarlandığını gösterir. Sonuçlar, belirli puanların ne kadar yaygın olduğunu ve hangi puanların daha sık bulunduğu ortaya koyar. Örneğin, 0.55 puanı en sık tekrar eden değer olarak 281 kez gözlemlenmiştir, ardından 0.50 puanı 269 kez, 0.51 puanı ise 264 kez göründüğündür. Bu değerler, last_evaluation puanlarının çoğunlukla bu aralıklarda yoğunlaştığını gösterir. Öte yandan, 0.36 puanı gibi daha az yaygın olan değerler veri kümeseinde yalnızca 19 kez gözlemlenmiştir. Toplamda, 65 farklı puan değeri bulunur, bu da puanların oldukça çeşitli olduğunu ve bazı değerlerin diğerlerinden daha sık tekrarlandığını gösterir. Bu bilgi, veri kümесindeki performans değerlendirmelerinin dağılımını anlamak ve analizlerinizde hangi puanların daha fazla temsil edildiğini belirlemek için kullanılabilir.

number_project

```
first_looking(df, 'number_project')

column name      : number_project
-----
per_of_nulls    : % 0.0
num_of_nulls    : 0
num_of_uniques : 6
shape_of_df     : (11991, 10)
-----
number_project
4      3685
```

```

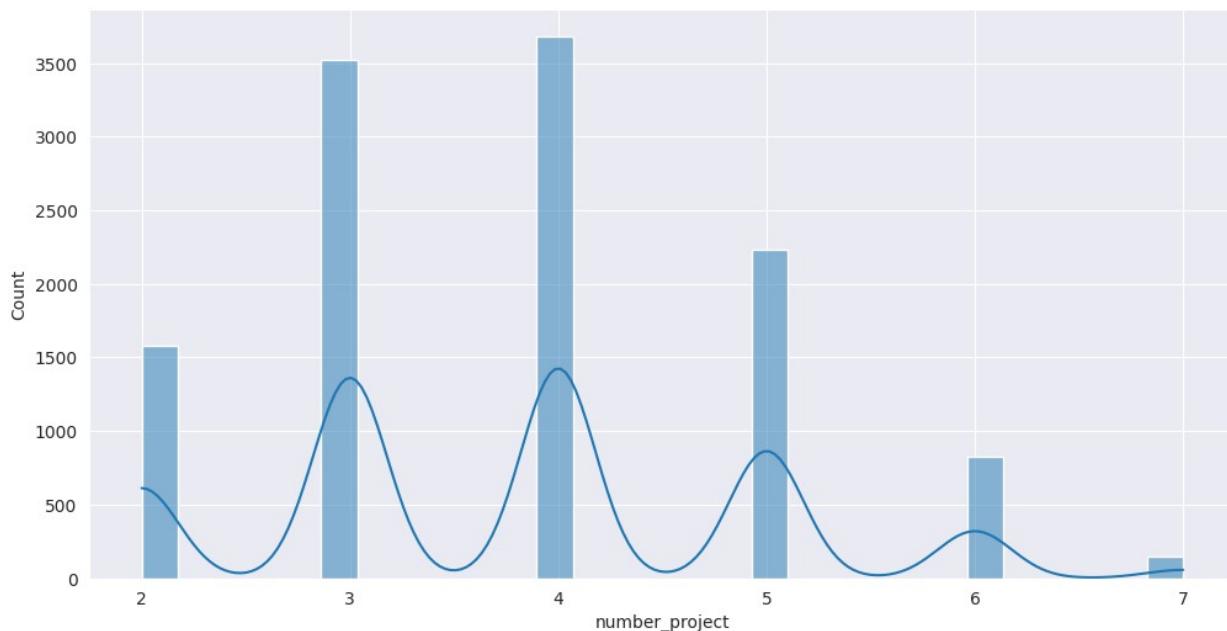
3    3520
5    2233
2    1582
6    826
7    145
Name: count, dtype: int64

```

number_project sütunu, her çalışanın kaç projede yer aldığıni gösteren bir veri içeriyor. Bu sütunda eksik değer bulunmuyor; yani tüm kayıtlar eksiksiz. Toplamda 11.991 veri bulunmaktadır ve 6 farklı proje sayısı mevcut.

Verilerin dağılımı incelediğinde, en yaygın proje sayısının 4 olduğu görülmektedir; bu durumda 3.685 çalışan bulunmaktadır. Bunu, 3 projeye sahip olan 3.520 çalışan takip ediyor. Diğer proje sayıları daha az yaygın; 2 proje için 1.582, 5 proje için 2.233, 6 proje için 826 ve 7 proje için 145 çalışan var. Proje sayıları 2 ile 7 arasında değişmektedir ve bu dağılım, çalışanların projelere göre nasıl dağıldığını göstermektedir.

```
sns.histplot(df, x = 'number_project', kde = True);
```



number_project sütununun histogramını çizmek için kullanılan sns.histplot fonksiyonu, her bir proje sayısının kaç kez tekrarlandığını görsel olarak gösterir. Histogram, bu sütundaki veri dağılımını ve frekansını incelemek için kullanışlıdır. kde=True parametresi, histogramın üzerine bir Kernel Density Estimation (KDE) eğrisi ekler, bu da veri dağılımının daha pürüzsüz bir tahminini sağlar.

Sonuç olarak:

Histogram: Bu grafik, farklı proje sayılarının sıklığını gösterir. X ekseninde proje sayıları yer almaktadır, Y ekseninde her bir proje sayısının kaç kez bulunduğu (frekansı) gösterilir. Histogramın barları, proje sayısının dağılımını ve yoğunluğunu görsel olarak ifade eder.

KDE Eğrisi: KDE eğrisi, veri dağılımının daha pürüzsüz bir tahminini sağlar ve veri noktalarının yoğunluklarını anlamaya yardımcı olur. Bu eğri, histogramın genel şeklini daha net bir şekilde görebilmenizi sağlar.

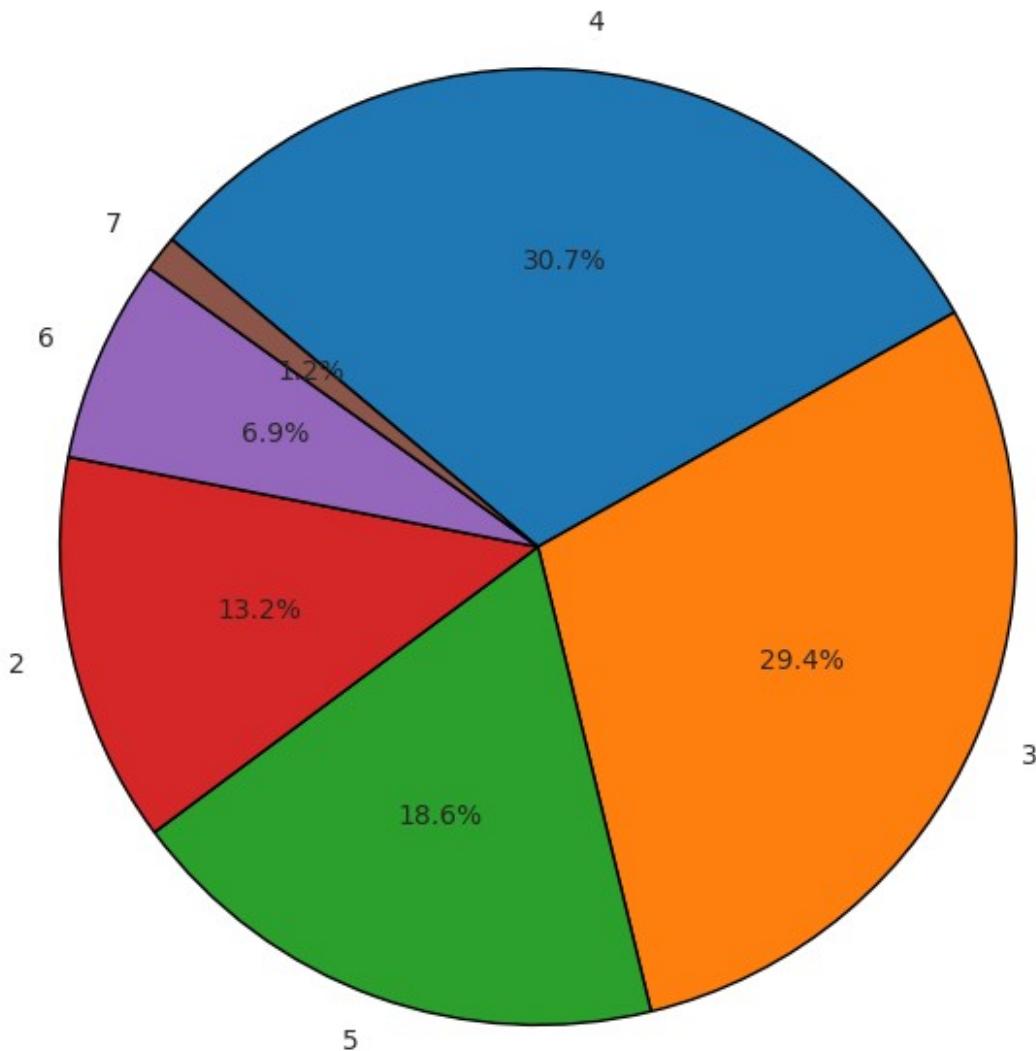
hangi proje sayılarının daha yaygın olduğunu ve hangi proje sayılarının daha nadir olduğunu görebilirsiniz. Örneğin, number_project sütunundaki değerlerin dağılımının çoğunlukla belirli aralıklar içinde yoğunlaştığını veya belirli değerlerde kümelenliğini gözlemleyebilirsiniz.

```
# Verilerin frekanslarını hesapla
project_counts = df['number_project'].value_counts()

# Pasta grafiği oluştur
plt.figure(figsize=(8, 8))
plt.pie(project_counts, labels=project_counts.index, autopct='%.1f%%',
        startangle=140, counterclock=False,
        wedgeprops={'edgecolor': 'black'})

plt.title('Number of Projects Distribution')
plt.show()
```

Number of Projects Distribution



Bu kod, number_project sütunundaki farklı proje sayılarının veri setindeki dağılımını görsel olarak gösteren bir pasta grafiği oluşturur. İlk olarak, number_project sütununda yer alan her proje sayısının ne kadar sıklıkla tekrarlandığını hesaplar ve bu frekansları bir Seri objesi olarak saklar. Ardından, bu verileri kullanarak bir pasta grafiği çizer. Pasta grafiği, her bir proje sayısının veri kümesindeki toplam içindeki yüzdesel dağılımını gösterir. Grafik, proje sayılarının hangi oranda dağıldığını net bir şekilde görmenizi sağlar. Dilimlerin üzerindeki yüzdelik değerler, her proje sayısının toplam projeler içindeki oranını belirtir. Grafiğin estetik bir görünüm kazanması için dilimlerin kenarları siyah renkte çizilir ve grafik 140 derece döndürülmüş olarak başlatılır. Son olarak, grafiğe bir başlık eklenir ve ekran üzerinde görüntülenir. Bu pasta grafiği, proje sayılarına dair genel bir bakış ve dağılım analizi sağlar.

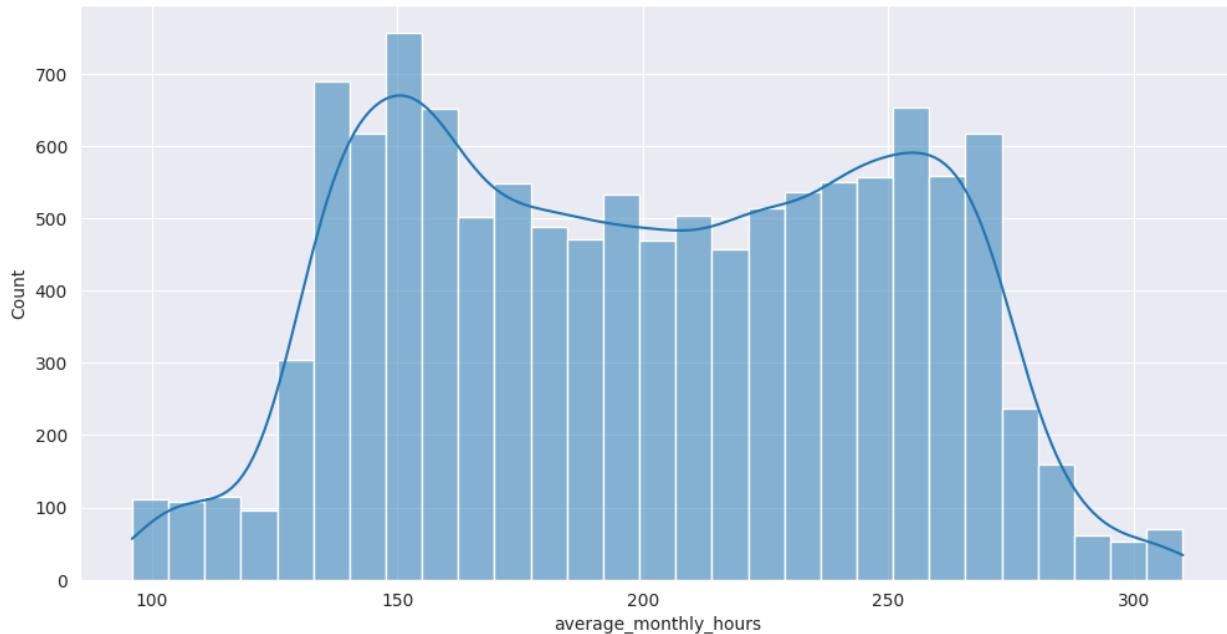
average_monthly_hours

```
first_looking(df, 'average_monthly_hours')

column name      : average_monthly_hours
-----
per_of_nulls    : % 0.0
num_of_nulls    : 0
num_of_uniques  : 215
shape_of_df     : (11991, 10)
-----
average_monthly_hours
156      112
149      112
160      111
151      107
135      104
...
298      5
302      5
297      5
299      5
303      5
Name: count, Length: 215, dtype: int64
```

average_monthly_hours sütunuyla ilgili yapılan analizde, veri kümesinde bu sütundaki her bir değerin frekansları hesaplanmıştır. Analiz sonucunda, sütunda toplam 215 farklı değer olduğu ve her bir değerin kaç kez tekrarlandığı belirtilmiştir. Bu değerler arasında, örneğin 156 saatlik çalışma süresi 112 kez, 149 saatlik çalışma süresi ise 112 kez gözlemlenmiştir. Bu tür detaylı bir frekans analizi, çalışma saatlerinin dağılımını anlamak için oldukça faydalıdır. Ayrıca, veri kümesinde bu sütunda eksik değer bulunmadığı ve tüm 11,991 satırda bu bilginin mevcut olduğu belirtilmiştir. Frekans dağılımı, belirli saat aralıklarında yoğunlaşan ya da nadiren karşılaşılan saatler hakkında bilgi verir ve bu da veri analizi sürecinde önemli bir içgörü sağlar.

```
sns.histplot(df, x = 'average_monthly_hours', kde = True);
```



bu grafik average_monthly_hours sütunundaki çalışma saatlerinin hangi aralıklarda yoğunlaştığını ve nasıl dağıldığını görsel olarak anlamanıza yardımcı olur.

```
#Index(['satisfaction_level', 'last_evaluation', 'number_project',
       '#average_monthly_hours', 'time_spend_company',
       'work_accident', 'left',
       '#promotion_last_5years', 'departments', 'salary'],
      #dtype='object')
```

Sütun İsimleri: Bu Index objesi, DataFrame'deki sütunların adlarını içerir:

'satisfaction_level': Çalışan memnuniyet düzeyini gösteren sütun.

'last_evaluation': Son değerlendirme notunu temsil eden sütun.

'number_project': Çalışanın üstlendiği proje sayısını gösteren sütun.

'average_monthly_hours': Çalışanın aylık ortalama çalışma saatlerini belirten sütun.

'time_spend_company': Şirkette geçirilen süreyi ölçen sütun.

'work_accident': Çalışanın iş kazası geçirme durumunu gösteren sütun.

'left': Çalışanın şirketten ayrılmış olduğunu belirten sütun (genellikle ikili değerler alır: ayrıldıysa 1, ayrılmadıysa 0).

'promotion_last_5years': Son beş yılda terfi alıpmadığını gösteren sütun.

'departments': Çalışanın çalıştığı departmanı belirten sütun.

'salary': Çalışanın maaş seviyesini gösteren sütun.

`dtype='object'`: Bu ifade, Index objesinin veri tipinin object olduğunu belirtir. Pandas'ta object genellikle metin veya karmaşık veri türlerini temsil eder.

Bu sütun isimleri ve veri türleri, DataFrame'in yapısını ve içeriğini anlamak için kullanılır.

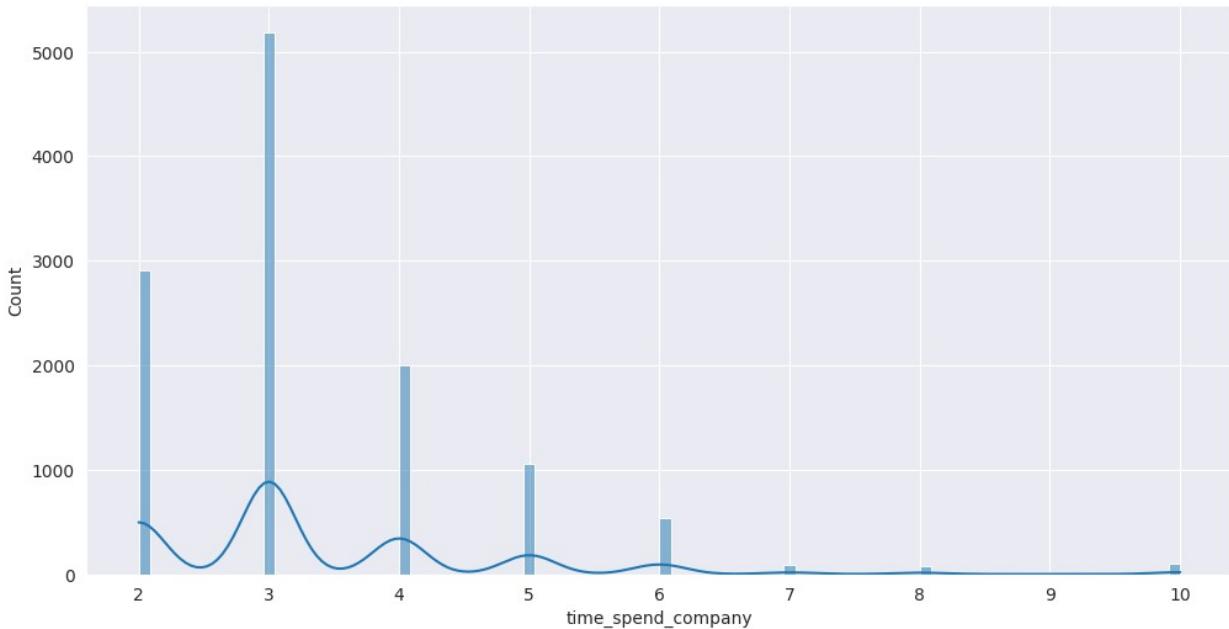
time_spend_company

```
first_looking(df, 'time_spend_company')

column name      : time_spend_company
-----
per_of_nulls    : % 0.0
num_of_nulls    : 0
num_of_uniques : 8
shape_of_df     : (11991, 10)
-----
time_spend_company
3      5190
2      2910
4      2005
5      1062
6       542
10      107
7        94
8        81
Name: count, dtype: int64
```

first_looking fonksiyonunun `time_spend_company` sütunu üzerinde kullanılması, bu sütunun temel özelliklerini ayrıntılı bir şekilde sunar. Öncelikle, bu sütunda hiç eksik değer bulunmadığını belirtir; tüm hücreler doludur. Sütundaki değerler arasında 8 farklı benzersiz yıl sayısı mevcuttur, bu da şirkette geçirilen yıl sayısının geniş bir aralığı kapsadığını gösterir. DataFrame'in genel yapısı içinde bu sütunun yer aldığı toplamda 11,991 satır ve 10 sütun olduğu belirtilir. Verilerin frekanslarına baktığımızda, 3 yıl boyunca şirkette kalan çalışan sayısının en yüksek olduğunu, bunu sırasıyla 2, 4, 5, 6, 10, 7 ve 8 yıl geçiren çalışanların takip ettiğini görüyoruz. Özellikle 3 yıl boyunca çalışan sayısının 5,190 gibi yüksek bir değere ulaşması, bu yıl aralığının en yaygın olduğunu gösterir. Bu dağılım, şirket içindeki çalışanların genel olarak hangi süre zarfında çalıştığını anlamamıza yardımcı olur ve veri analizi sürecinde bu bilgilerin nasıl kullanılacağı konusunda önemli ipuçları sağlar.

```
sns.histplot(df, x = 'time_spend_company', kde = True);
```



`time_spend_company` sütununu `sns.histplot` fonksiyonu ile görselleştirdiğinizde, şirket içinde geçirilen yılların dağılımını görsel olarak inceleyebilirsiniz. Bu histogram ve yoğunluk eğrisi (KDE) grafiği aşağıdaki bilgileri sunar:

Histogram: X ekseninde şirket içinde geçirilen yıl sayısı, Y ekseninde ise bu yıl sayısına sahip çalışanların frekansı yer alır. Her bir çubuğun yüksekliği, belirli bir yıl aralığında kaç çalışan bulunduğu gösterir. Bu şekilde, hangi sürelerin daha yaygın olduğunu ve hangi sürelerin daha az yaygın olduğunu hızlıca görebilirsiniz.

Yoğunluk Eğrisi (KDE): Histogramın üstüne eklenen bu eğri, veri dağılımının daha pürüzsüz bir temsilidir. KDE, belirli bir yıl sayısında çalışanların yoğunluğunu gösterir ve veri kümelerindeki yoğun bölgeleri daha belirgin hale getirir. Eğri, yıl sayıları arasında geçişleri ve olası yoğunluk merkezlerini görselleştirir.

Grafikte genellikle şirket içinde geçirilen yılların çoğunluğunun 2-3 yıl arasında yoğunlaştığı gözlemlenebilir. Yoğunluk eğrisi genellikle bu yıl aralıklarında yüksek bir yoğunluk gösterir, bu da bu sürelerin çalışanlar arasında en sık rastlanan süreler olduğunu ifade eder. Diğer yıl aralıkları ise daha az yoğunluk gösterir. Bu tür bir analiz, şirkette çalışan sürelerinin dağılımını anlamınızı sağlar ve çalışan hareketliliği gibi konuları incelemek için temel bilgiler sunar.

```
df['time_spend_company'].value_counts(normalize = True)*100
```

| time_spend_company | Percentage |
|--------------------|------------|
| 3 | 43.28 |
| 2 | 24.27 |
| 4 | 16.72 |
| 5 | 8.86 |
| 6 | 4.52 |
| 10 | 0.89 |
| 7 | 0.78 |

```
8      0.68
Name: proportion, dtype: float64
```

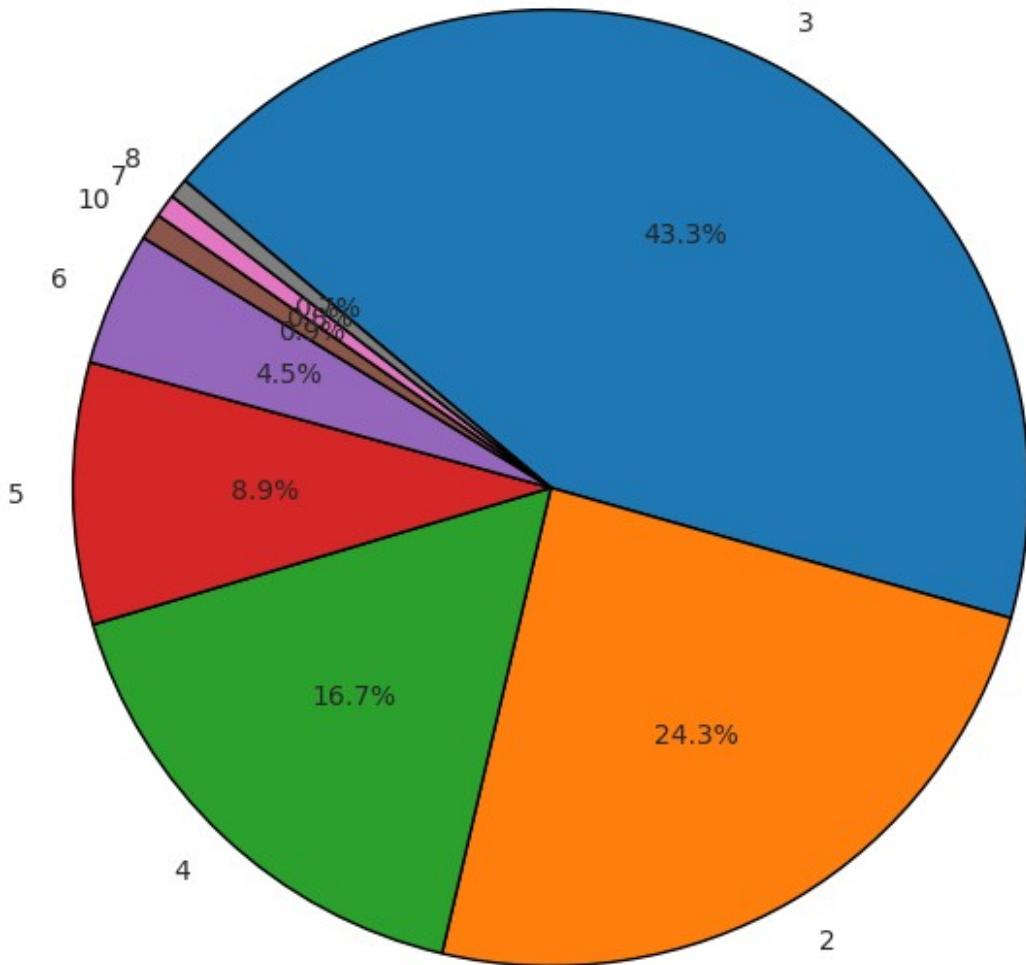
df['time_spend_company'].value_counts(normalize=True) * 100 komutu, verilerdeki time_spend_company sütunundaki her bir benzersiz değerin yüzdesini hesaplar. Sonuçlar, her bir süre aralığının şirket içindeki çalışanlar arasında ne kadar yaygın olduğunu gösterir. Çalışanların büyük çoğunluğu, şirkette geçirdikleri süre bakımından 3 yıl (%43.28) ile en yüksek orana sahiptir. Bunu, 2 yıl süren çalışanlar (%24.27) ve 4 yıl süren çalışanlar (%16.72) takip eder. 5 yıl ve daha uzun süre kalan çalışanların oranı ise belirgin şekilde düşüktür. Örneğin, 6 yıl süren çalışanlar sadece %4.52, 10 yıl sürenler ise %0.89 gibi çok düşük bir oranı temsil eder. Bu dağılım, şirket içinde geçirilen sürelerin büyük ölçüde kısa ve orta vadeli olduğunu, uzun süreli çalışanların ise nadir olduğunu ortaya koyar.

```
# Verilerin frekanslarını hesapla
time_spend_counts = df['time_spend_company'].value_counts()

# Pasta grafiği oluştur
plt.figure(figsize=(8, 8))
plt.pie(time_spend_counts, labels=time_spend_counts.index,
autopct='%1.1f%%', startangle=140, counterclock=False,
wedgeprops={'edgecolor': 'black'})

plt.title('Time Spend of the Company Distribution')
plt.show()
```

Time Spend of the Company Distribution



time_spend_counts verisi, şirket içinde geçirilen sürelerin her bir kategorisinin frekanslarını hesaplar ve bu frekanslar kullanılarak bir pasta grafiği oluşturulur. Grafikte, time_spend_company sütunundaki her bir benzersiz süre kategorisinin, toplam çalışanlar içindeki yüzdesi gösterilmektedir.

Grafik, şirket içinde geçirilen sürelerin dağılımını görsel olarak anlamayı kolaylaştırır. Örneğin, çoğu çalışanın 2 veya 3 yıl süresince şirkette kaldığını, daha uzun süreli çalışanların ise daha az yaygın olduğunu gösterir. Bu, şirketin çalışan devir hızını ve çalışan sürelerinin yayılımını hızlı bir şekilde analiz etmeye yardımcı olur.

work_accident

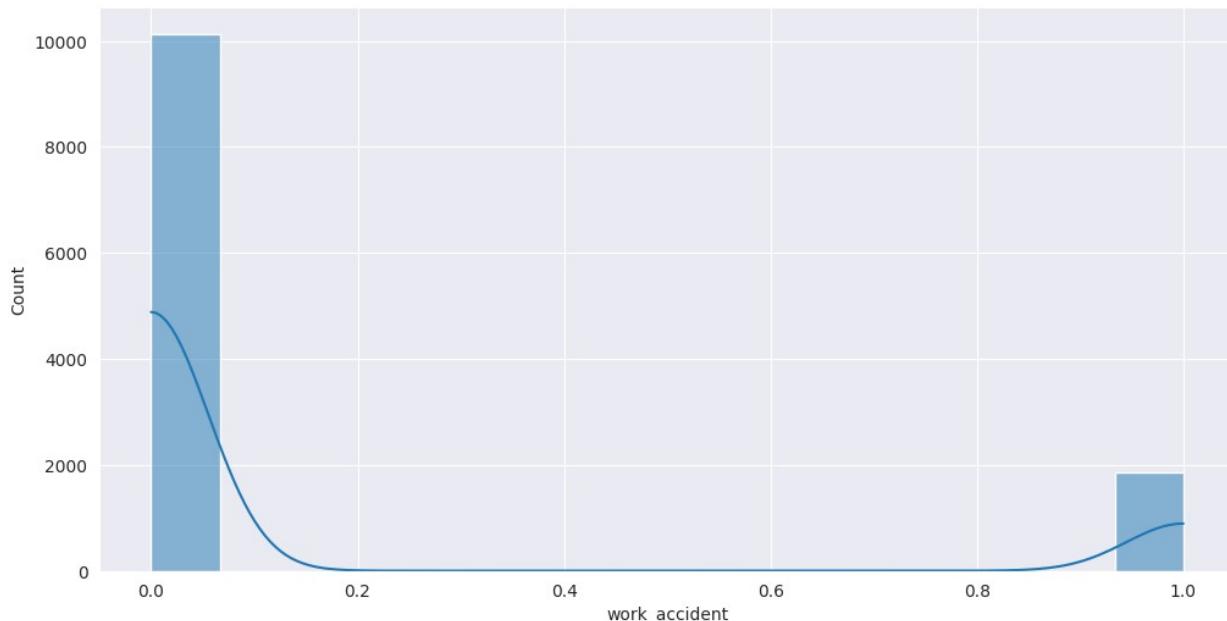
```
first_looking(df, 'work_accident')

column name      : work_accident
-----
per_of_nulls    : % 0.0
num_of_nulls    : 0
num_of_uniques  : 2
shape_of_df     : (11991, 10)
-----
work_accident
0      10141
1      1850
Name: count, dtype: int64
```

Bu sütunla ilgili yapılan incelemeye göre:

Eksik Veri: Sütunda eksik değer bulunmamaktadır; tüm 11,991 satırda veriler tamamlanmıştır.
Benzersiz Değerler: work_accident sütununda iki benzersiz değer vardır: 0 (kaza geçirmemiş) ve 1 (kaza geçirmiştir). Değer Dağılımı: 0 değeri 10,141 kez tekrarlanırken, 1 değeri 1,850 kez gözlemlenmiştir. Bu da, verideki çoğu çalışanın iş yerinde bir kaza yaşamadığını, ancak bir kısmının yaşadığını gösterir. Bu bilgiler, iş yerindeki kaza oranının analizi için önemlidir. Kaza geçirme durumu oldukça düşük bir oranda gözlemlenmektedir ve bu, iş yerinde güvenlik önlemlerinin etkili olduğunu gösterebilir veya kazaların veri toplama sürecinde eksik olduğunu işaret edebilir.

```
sns.histplot(df, x = 'work_accident', kde = True);
```



work_accident sütununu içeren histogram grafiği, iş yerindeki kaza geçirme durumlarının dağılımını görselleştirir. Bu sütun, iki kategorili bir veri sütunudur: 0 (kaza geçirmemiş) ve 1 (kaza geçirmiştir). Grafik, bu iki kategori arasındaki frekanstağı dağılımı ve yoğunluğu gösterir.

Grafikte:

X-Ekseninde: İş kazası durumu (0 veya 1) yer alır. Y-Ekseninde: Her kategori için gözlem sayısının dağılımı gösterilir. Grafikte genellikle şunlar gözlemlenir:

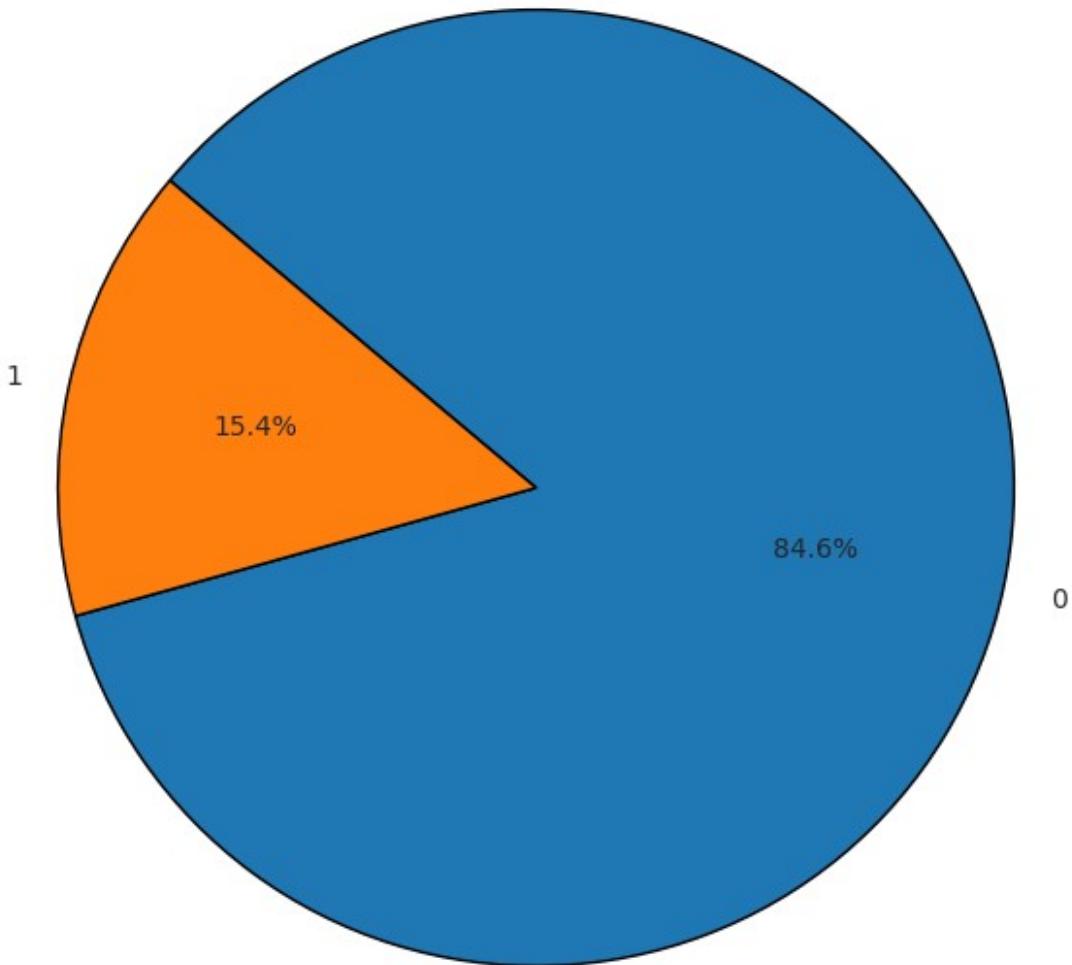
Histogram: 0 ve 1 değerlerinin sıklığını gösterir. 0 (kaza geçirmemiş) genellikle çok daha yüksek bir frekansta olacaktır. Bu, çoğu çalışanın iş yerinde kaza yaşamadığını gösterir. KDE Eğrisi: Kernel Density Estimate (KDE) eğrisi, veri noktalarının dağılımını daha pürüzsüz bir şekilde gösterir. Ancak, iki kategorili bir veri için KDE eğrisi genellikle sınırlı bilgi sağlar ve çoğunlukla histogramın üzerine konur. Bu grafik, iş yerinde kaza geçirme durumunun ne kadar yaygın olduğunu anlamak ve bu olayların veri setinde nasıl dağıldığını görmek için kullanışlıdır. Çoğunlukla 0 değeri yüksek bir sıklığa sahip olacaktır, bu da iş yerindeki kazaların nispeten nadir olduğunu gösterir.

```
accident_counts = df['work_accident'].value_counts()

# Pasta grafiği oluştur
plt.figure(figsize=(8, 8))
plt.pie(accident_counts, labels=accident_counts.index, autopct='%.1f%',
        startangle=140, counterclock=False,
        wedgeprops={'edgecolor': 'black'})

plt.title('Work Accident Distribution')
plt.show()
```

Work Accident Distribution



Grafik, iş kazası geçirme durumlarının veri setinde nasıl dağıldığını ve bu durumların oranlarını hızlı bir şekilde anlamak için kullanışlıdır. Çoğu dilim büyük ihtimalle 0 olacaktır, bu da iş yerindeki kazaların yaygın olmadığını gösterir.

[promotion_last_5years](#)

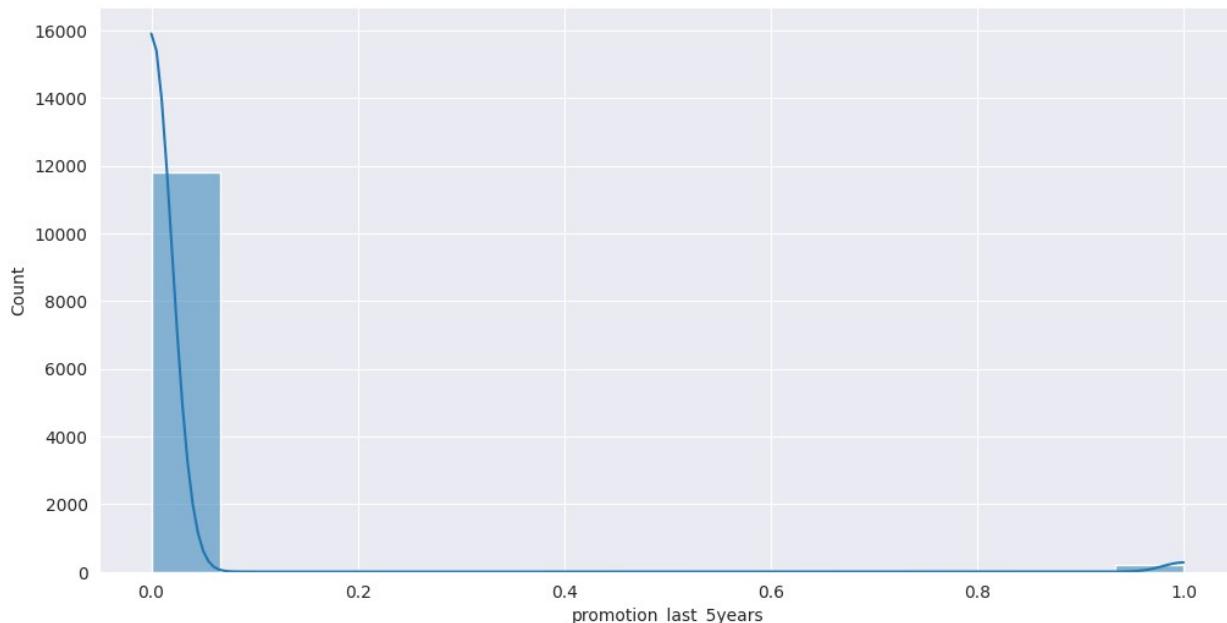
```
first_looking(df, 'promotion_last_5years')  
column name      : promotion_last_5years  
-----  
per_of_nulls    : % 0.0  
num_of_nulls    : 0
```

```
num_of_uniques : 2
shape_of_df    : (11991, 10)
-----
promotion_last_5years
0    11788
1     203
Name: count, dtype: int64
```

`promotion_last_5years` sütunundaki veriler, son beş yıl içinde terfi alıp almadığını belirten iki kategori içerir. Bu sütunda eksik veri bulunmamaktadır, yani tüm 11,991 kayıt için veri mevcuttur. Bu sütunun iki benzersiz değeri vardır:

0: Bu değeri taşıyan 11,788 çalışan son beş yıl içinde herhangi bir terfi almamıştır. 1: Bu değeri taşıyan 203 çalışan ise son beş yıl içinde terfi almıştır. Verilere göre, çoğunlukla çalışanların terfi almadığı görülmektedir. Yani, bu sütunun büyük kısmı 0 değeri ile doldurulmuş ve 1 değeri oldukça düşük bir orandadır. Bu durum, terfi fırsatlarının sınırlı veya nadir olduğunu gösterebilir. Genel olarak, terfi oranı düşük, yani sadece küçük bir grup çalışan terfi almış.

```
sns.histplot(df, x = 'promotion_last_5years', kde = True);
```



`promotion_last_5years` sütunundaki sns.histplot grafiği, son beş yıl içinde terfi alıp almadığını gösteren verilerin dağılımını görselleştirir. Grafikte iki ana kategori bulunmaktadır:

0: Bu değer, terfi almamış çalışanları temsil eder ve histogramda yüksek bir sıklık gösterir. Bu durum, çoğu çalışanın terfi almadığını ve bu grubun verilerde baskın olduğunu gösterir.

1: Bu değer, terfi almış çalışanları temsil eder ve histogramda düşük bir sıklık gösterir. Bu, terfi alan çalışan sayısının oldukça az olduğunu belirtir.

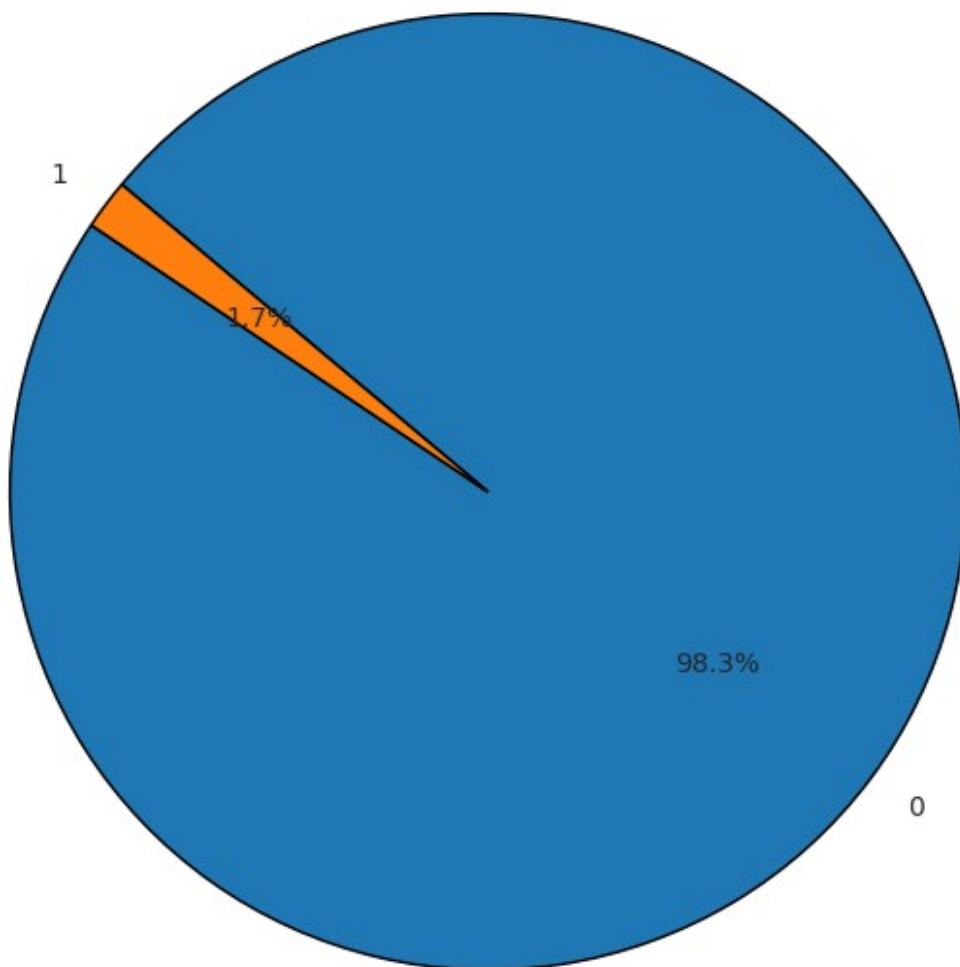
Grafikteki kde (Kernel Density Estimate) eğrisi, veri kümesinin sürekli bir dağılımını tahmin eder. Bu eğri, histogramdaki verilerin yoğunluğunu yumuşatarak daha pürüzsüz bir dağılım sağlar. Ancak, bu sütundaki veri yapısının kategorik olduğunu göz önünde bulundurduğumuzda, kde eğrisi bu tür veriler için pek anlamlı olmayabilir. Yine de, verilerin yoğunluğu ve dağılımı hakkında genel bir fikir verir.

```
promotion_counts = df['promotion_last_5years'].value_counts()

# Pasta grafiği oluştur
plt.figure(figsize=(8, 8))
plt.pie(promotion_counts, labels=promotion_counts.index,
autopct='%.1f%%', startangle=140, counterclock=False,
wedgeprops={'edgecolor': 'black'})

plt.title('Promotion Distribution (Last 5 Years)')
plt.show()
```

Promotion Distribution (Last 5 Years)



`promotion_last_5years` sütununu içeren pasta grafiği, son beş yıl içinde terfi alıp almama durumlarının dağılımını görselleştirir. Grafikte iki ana kategori bulunmaktadır:

Terfi Almayanlar (0): Bu segment, son beş yıl içinde terfi almamış çalışanların oranını gösterir ve pasta grafiğinde büyük bir dilim olarak yer alır. Bu, çalışanların büyük çoğunluğunun terfi almadığını belirtir.

Terfi Alanlar (1): Bu segment, son beş yıl içinde terfi almış çalışanların oranını gösterir ve pasta grafiğinde küçük bir dilim olarak yer alır. Bu, terfi almış çalışan sayısının görece olarak az olduğunu gösterir.

Grafikteki dilimlerin yüzdelik değerleri, her kategoriye düşen oranı açıkça gösterir. Grafiğin tasarımı, her iki kategorinin görsel olarak karşılaştırılmasını sağlar ve terfi durumlarının genel

dağılımını anlamak için kolayca yorumlanabilir. Bu tür bir grafik, veri setindeki terfi durumlarının ne kadar yaygın olduğunu ve bu durumların oranını hızlı bir şekilde değerlendirmeye yardımcı olur.

departments

```
first_looking(df, 'departments')

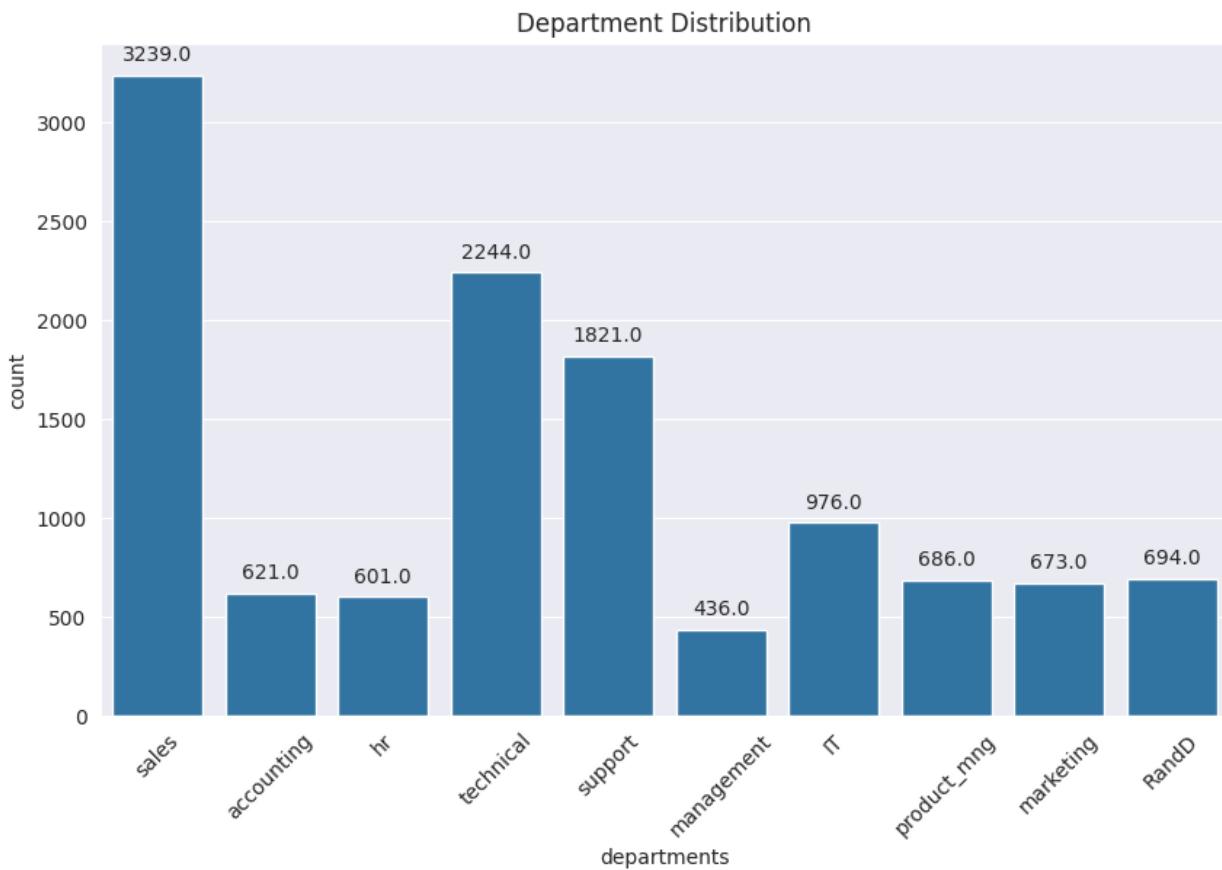
column name      : departments
-----
per_of_nulls     : % 0.0
num_of_nulls    : 0
num_of_uniques  : 10
shape_of_df      : (11991, 10)
-----
departments
sales            3239
technical        2244
support          1821
IT                976
RandD             694
product_mng      686
marketing         673
accounting        621
hr                601
management       436
Name: count, dtype: int64
```

departments sütunu, veri setinde yer alan her çalışanın hangi departmanda görev aldığıni belirten bir kategorik değişkendir. Bu sütunda toplamda 10 farklı departman bulunmaktadır. Çalışan sayısı bakımından en büyük departman Satış (sales) olup, 3,239 çalışanı ile en büyük grubu oluşturur. Bunu sırasıyla Teknik (technical), Destek (support), ve BT (IT) departmanları izler. Her bir departmanın çalışan sayısı ve oranları, şirketin iç işleyiş hakkında bilgi verir ve özellikle iş gücü planlaması, kaynak dağılımı ve departmanlar arası stratejiler geliştirmek için kullanışlıdır. Örneğin, Araştırma ve Geliştirme (RandD) departmanı, Pazarlama (marketing) ve Muhasebe (accounting) gibi departmanlar daha küçük grupları temsil eder. Bu bilgiler, şirketin departman bazında iş gücünü ve organizasyonel yapısını anlamak için önemlidir.

```
plt.figure(figsize=(10, 6))
ax = sns.countplot(data=df, x='departments')
plt.xticks(rotation=45) # Departman isimlerini döndürmek için
plt.title('Department Distribution')

# Her bir çubuğun üzerine sayısal değer ekleme
for p in ax.patches:
    ax.annotate(f'{p.get_height()}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='center',
                xytext=(0, 10),
```

```
textcoords='offset points')  
plt.show()
```



Bu grafik, her bir departmandaki çalışan sayısının görsel bir temsilini sağlar ve departmanlar arasındaki çalışan dağılımını hızlı bir şekilde değerlendirmeye yardımcı olur.

salary

```
first_looking(df, 'salary')  
  
column name      : salary  
-----  
per_of_nulls     : % 0.0  
num_of_nulls    : 0  
num_of_uniques  : 3  
shape_of_df      : (11991, 10)  
-----  
salary  
low      5740  
medium   5261  
high     990  
Name: count, dtype: int64
```

df veri setindeki salary (maaş) sütunuyla ilgili yapılan analiz şu bilgileri verir:

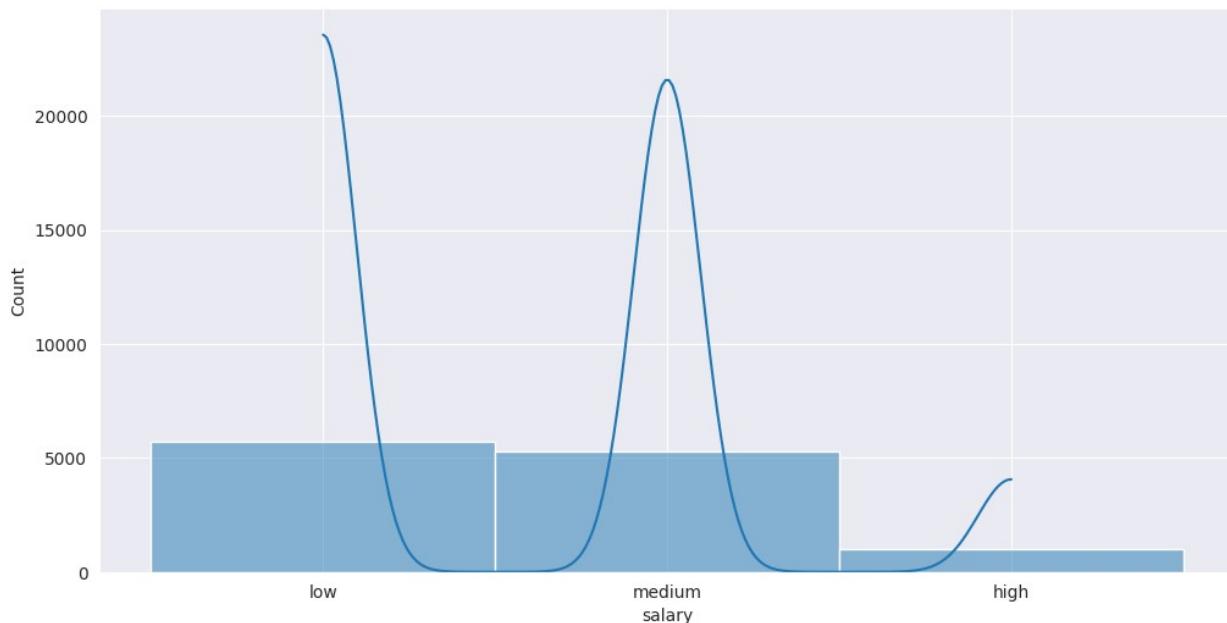
Eksik Veri Yok: per_of_nulls ve num_of_nulls değerleri sıfırdır, yani bu sütunda eksik veri bulunmamaktadır.

Benzersiz Değerler: num_of_uniques değeri 3'tür, bu da maaş kategorilerinin 3 farklı türde olduğunu gösterir: "low" (düşük), "medium" (orta) ve "high" (yüksek).

Veri Dağılımı:

"low" (düşük maaş) kategorisinde 5740 kayıt bulunur. "medium" (orta maaş) kategorisinde 5261 kayıt vardır. "high" (yüksek maaş) kategorisinde ise 990 kayıt mevcuttur. Bu dağılım, veri setindeki maaş kategorilerinin frekansını ve her bir kategoriye ait çalışan sayısını göstermektedir. Örneğin, düşük ve orta maaş kategorilerinin daha yaygın olduğunu, yüksek maaş kategorisinin ise daha az temsil edildiğini görebiliriz. Bu tür bir analiz, maaş dağılıminın genel profilini anlamak ve şirket içindeki maaş dağılımını değerlendirmek için yararlıdır.

```
sns.histplot(df, x = 'salary', kde = True);
```

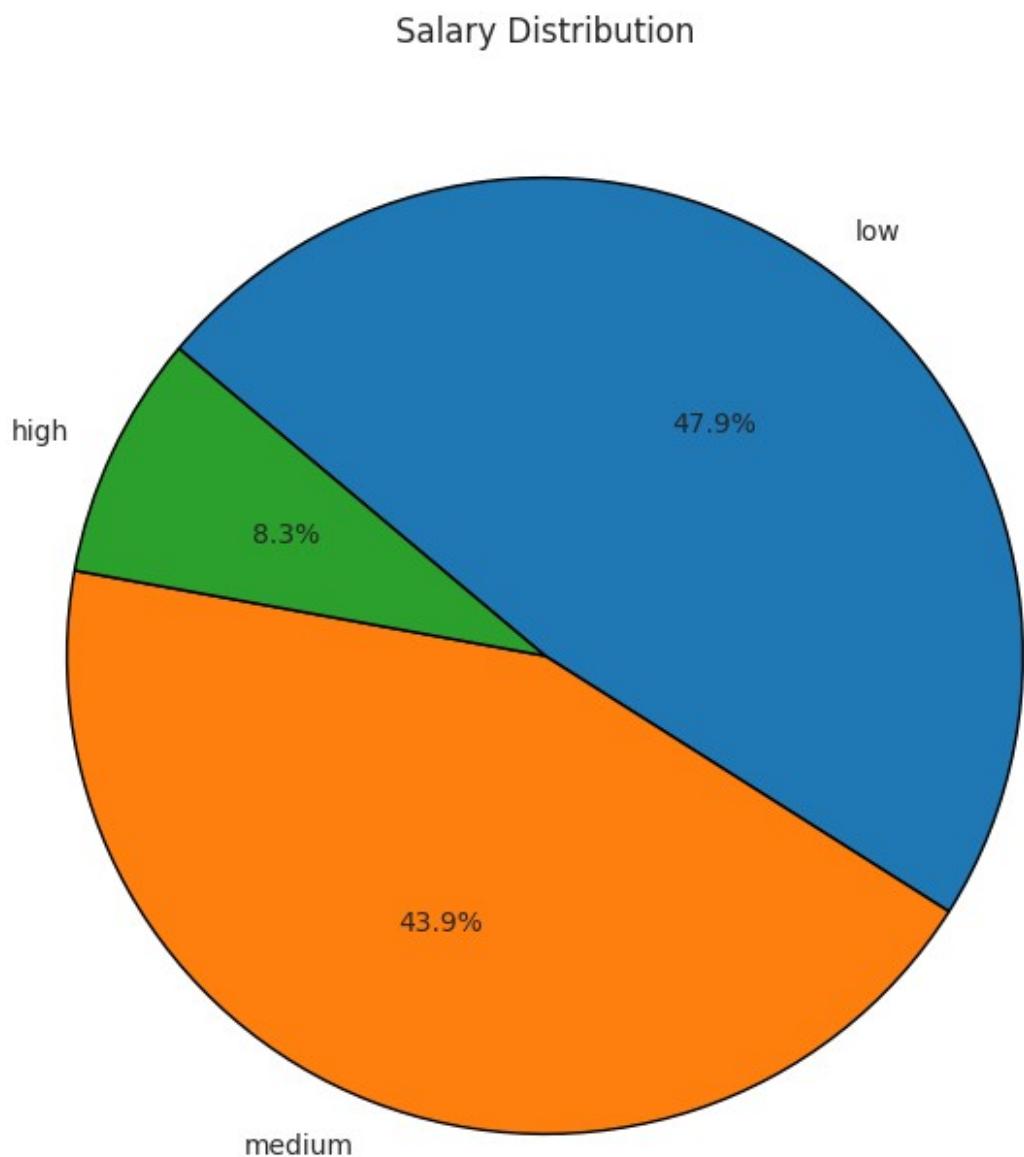


salary sütununa ait oluşturulan histogram ve Kernel Density Estimation (KDE) grafiği, maaş kategorilerinin dağılımını detaylı bir şekilde gösterir. Histogram, "low", "medium" ve "high" olarak üç kategorideki maaşların her birinin ne kadar sıklıkla yer aldığıni çubuklarla ifade eder. Burada "low" maaşlı çalışanların sayısının en yüksek, "high" maaşlı çalışanların ise en düşük olduğunu görebiliriz. KDE eğrisi ise bu verilerin üzerine uygulanan bir yoğunluk tahminidir; ancak, çünkü salary gibi kategorik verilerde KDE eğrisi genellikle belirgin bir yoğunluk göstermez. Bu durumda KDE eğrisi, maaş kategorilerinin sayısal dağılımını pek etkileyemez ve histogramdaki çubuklarla birlikte çok fazla bilgi sunmaz. Sonuç olarak, histogram verilerin kategorilere göre ne kadar sıklıkta dağıldığını gösterirken, KDE eğrisi, bu kategorilerdeki veri yoğunluğunu daha pürüzsüz bir şekilde görselleştirmeye çalışır.

```
salary_counts = df['salary'].value_counts()

# Pasta grafiği oluştur
plt.figure(figsize=(8, 8))
plt.pie(salary_counts, labels=salary_counts.index, autopct='%.1f%%',
        startangle=140, counterclock=False,
        wedgeprops={'edgecolor': 'black'})

plt.title('Salary Distribution')
plt.show()
```



Oluşturulan pasta grafiği, çalışanların maaş dağılımını göstermektedir. Grafikte, "low", "medium" ve "high" maaş kategorilerinin her birinin tüm çalışanlar içerisindeki oranları gösterilmektedir.

Grafikte, her bir dilim, maaş kategorisinin toplam çalışan sayısına olan yüzdesini temsil eder. Grafik, maaş kategorilerinin genel dağılımını net bir şekilde ortaya koyar.

Low (Düşük) maaş kategorisi en büyük dilimi oluşturur ve tüm çalışanların büyük bir kısmını kapsar. Medium (Orta) maaş kategorisi de oldukça büyük bir dilim oluşturur ve önemli bir kısmı temsil eder. High (Yüksek) maaş kategorisi ise en küçük dilimi temsil eder, yani bu kategori en az çalışan sayısına sahiptir. Pasta grafiği, maaş kategorilerinin oranlarını görsel olarak kolayca anlamınızı sağlar ve hangi maaş kategorisinin baskın olduğunu gösterir.

3. Cluster Analysis

```
first_looking(df, 'left')

column name      : left
-----
per_of_nulls     : % 0.0
num_of_nulls    : 0
num_of_uniques  : 2
shape_of_df      : (11991,
-----
left
0      10000
1      1991
Name: count, dtype: int64
```

"left" sütunu, çalışanların şirketten ayrılmış ayrılmadığını gösterir. Bu sütun iki benzersiz değeri içerir: 0 ve 1.

0: Çalışanın şirketten ayrılmadığını gösterir. Bu değere sahip 10,000 kayıt bulunmaktadır. 1: Çalışanın şirketten ayrıldığını gösterir. Bu değere sahip 1,991 kayıt bulunmaktadır. Bu veriler, şirketin çalışanlarını ne ölçüde koruyabildiğini ve ayrılma oranını değerlendirmek için kullanılabilir. Çalışanların büyük çoğunluğu şirkette kalmışken, bir kısmı ise ayrılmış. Bu dağılım, çalışan devinimi ve bağlılık düzeyi hakkında önemli bilgiler sunar.

```
df[df['left'] == 1].describe().T
```

```
{"summary": {"name": "df[df['left'] == 1]", "rows": 8, "columns": [{"column": "count", "dtype": "number", "mean": 0.0, "min": 1991.0, "max": 1991.0, "std": 0.0, "num_unique_values": 1, "samples": [1991.0]}, {"column": "mean", "dtype": "number", "mean": 73.10942885025058, "min": 0.004018081366147665, "max": 208.16223003515822, "std": 73.10942885025058, "num_unique_values": 8, "samples": [0.7217830236062281]}], "semantic_type": "number", "description": "\n"}, {"column": "std", "dtype": "number", "mean": 0.0, "min": 0.0, "max": 0.0, "std": 0.0, "num_unique_values": 1, "samples": [0.0]}, {"column": "count", "dtype": "number", "mean": 8.0, "min": 8.0, "max": 8.0, "std": 0.0, "num_unique_values": 1, "samples": [8.0]}]
```

```

    "semantic_type": "\",\n      "description": \"\\n      }\n    },\n    {\n      "column": "std",\n      "properties": {\n        "dtype": "number",\n        "std": 21.50099183811935,\n        "min": 0.0,\n        "max": 61.29514478095236,\n        "num_unique_values": 8,\n        "samples": [\n          0.19743629500684287\n        ],\n        "semantic_type": "\",\n        "description": \"\\n      }\n      },\n      {\n        "column": "min",\n        "properties": {\n          "dtype": "number",\n          "std": 44.27570141414233,\n          "min": 0.0,\n          "max": 126.0,\n          "num_unique_values": 6,\n          "samples": [\n            0.09\n          ],\n          "semantic_type": "\",\n          "description": \"\\n      }\n          },\n          {\n            "column": "25%",\n            "properties": {\n              "dtype": "number",\n              "std": 51.29507479489903,\n              "min": 0.0,\n              "max": 146.0,\n              "num_unique_values": 7,\n              "samples": [\n                0.11\n              ],\n              "semantic_type": "\",\n              "description": \"\\n      }\n              },\n              {\n                "column": "50%",\n                "properties": {\n                  "dtype": "number",\n                  "std": 79.4049181636215,\n                  "min": 0.0,\n                  "max": 226.0,\n                  "num_unique_values": 6,\n                  "samples": [\n                    0.41\n                  ],\n                  "semantic_type": "\",\n                  "description": \"\\n      }\n                  },\n                  {\n                    "column": "75%",\n                    "properties": {\n                      "dtype": "number",\n                      "std": 92.14735757005113,\n                      "min": 0.0,\n                      "max": 262.5,\n                      "num_unique_values": 7,\n                      "samples": [\n                        0.73\n                      ],\n                      "semantic_type": "\",\n                      "description": \"\\n      }\n                      },\n                      {\n                        "column": "max",\n                        "properties": {\n                          "dtype": "number",\n                          "std": 108.72534571110822,\n                          "min": 0.92,\n                          "max": 310.0,\n                          "num_unique_values": 5,\n                          "samples": [\n                            1.0\n                          ],\n                          "semantic_type": "\",\n                          "description": \"\\n      }\n                          }\n                        ]\n                      },\n                      "type": "dataframe"

```

"left" sütununda 1 olan çalışanların özelliklerini özetleyen istatistikler aşağıdaki gibidir:

Satisfaction Level (Memnuniyet Seviyesi): Şirketten ayrılan çalışanların ortalama memnuniyet seviyesi 0.44'tür. Bu, memnuniyet seviyesinin oldukça düşük olduğunu ve çalışanların büyük kısmının işlerinden memnun olmadığını gösterir. Memnuniyet seviyeleri 0.09 ile 0.92 arasında değişmektedir.

Last Evaluation (Son Değerlendirme): Bu çalışanların son performans değerlendirme puanlarının ortalaması 0.72'dir. Performans değerlendirme puanları 0.45 ile 1.00 arasında değişmektedir, bu da bazı çalışanların oldukça yüksek değerlendirmeler aldığı, bazlarının ise daha düşük puanlar aldığı gösterir.

Number of Projects (Projelerin Sayısı): Şirketten ayrılan çalışanların ortalama proje sayısı 3.88'dir. Çalışanlar genellikle 2 ile 7 arasında projede yer almışlardır. Bu, çalışanların çoğunun belirli bir projeden fazlasında yer aldığı gösterir.

Average Monthly Hours (Ortalama Aylık Çalışma Saatleri): Ayrılan çalışanların ortalama aylık çalışma saati 208.16 saat olup, bu oran 126 ile 310 saat arasında değişmektedir. Bu yüksek değer, ayrılan çalışanların genellikle uzun saatler çalıştığını göstermektedir.

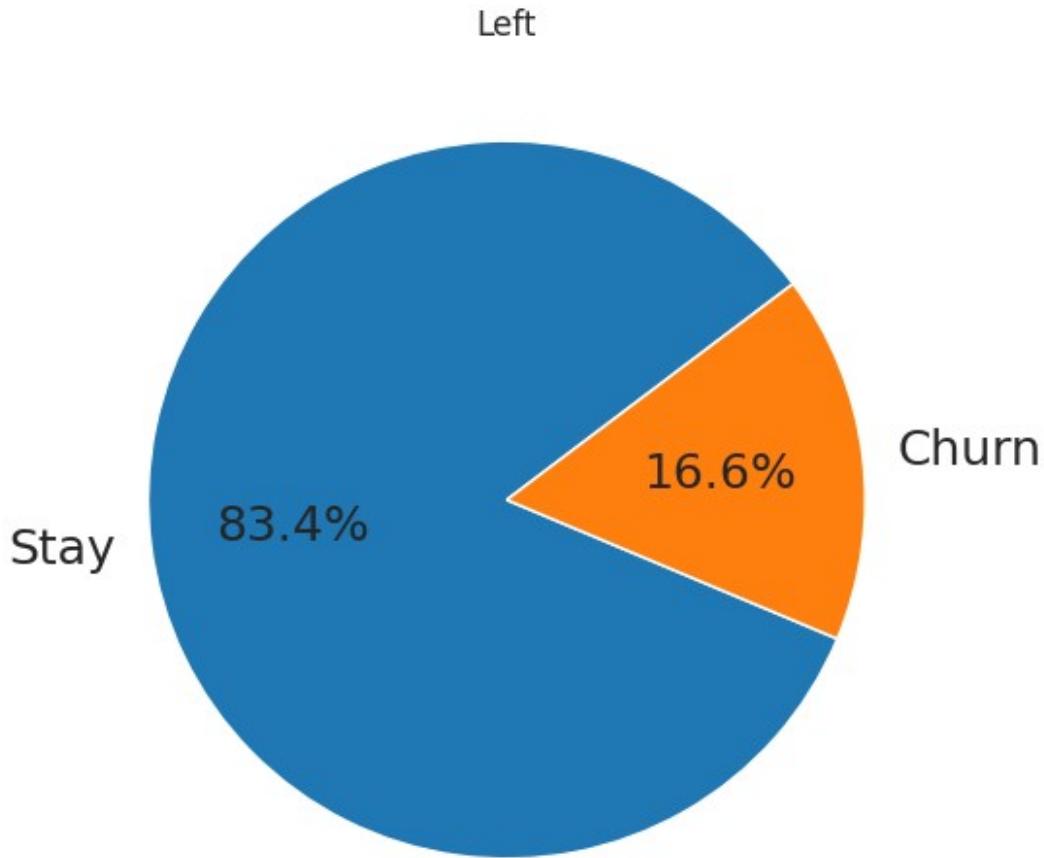
Time Spend Company (Şirkette Geçirilen Süre): Şirketten ayrılan çalışanların ortalama şirkette kalma süresi 3.88 yıldır. Bu süre 2 yıl ile 6 yıl arasında değişmektedir, yani ayrılan çalışanlar genellikle birkaç yıl şirkette kalmışlardır.

Work Accident (İş Kazası): İş kazası geçirenlerin oranı oldukça düşüktür (ortalama 0.05). Yani, ayrılan çalışanların çoğu iş kazası geçirmemiştir.

Promotion Last 5 Years (Son 5 Yılda Terfi): Ayrılan çalışanların çoğu (ortalama 0.00) son 5 yılda terfi almamıştır. Bu, terfi almayan çalışanların şirkette kalma olasılığının daha düşük olduğunu düşündürebilir.

Bu özet veriler, ayrılan çalışanların iş memnuniyeti, performans değerlendirmeleri, proje sayıları, çalışma saatleri, şirkette geçirdikleri süre ve terfi durumları hakkında bilgi verir. Bu bilgiler, çalışanların ayrılma kararlarını etkileyen faktörleri anlamak için kullanılabilir.

```
df.left.value_counts().plot.pie(autopct='%.1f%%',
                                startangle=37,
                                fontsize=18,
                                labels=["Stay", "Churn"],
                                ylabel='',
                                title='Left');
```



Bu kod, çalışanların şirkette kalma ve ayrılma durumlarını görsel olarak göstermek amacıyla bir pasta grafiği oluşturur. Pasta grafiği, left sütunundaki iki kategoriye dayanarak hazırlanmıştır: şirkette kalan çalışanlar ("Stay") ve şirkette ayrılan çalışanlar ("Churn"). Grafikte her bir dilim, bu iki kategori arasındaki oranı yüzdelik dilimlerle temsil eder, böylece şirketten ayrılma oranını açıkça görebiliriz. Grafik, başlangıç açısı olarak 37 derece kullanarak estetik bir görünüm sağlar ve yazı tiplerini 18 punto olarak ayarlayarak etiketlerin okunabilirliğini artırır. Y ekseni etiketi kullanılmaz çünkü pasta grafiği y ekseni içermez. Başlık olarak "Left" kullanılarak grafiğin içeriği hakkında bilgi verilir. Bu grafik, ayrılma oranını analiz etmek ve çalışanların şirkette kalma durumunu görsel olarak değerlendirmek için etkili bir araçtır.

```
df.shape
```

```
(11991, 10)
```

bu veri çerçevesi 11,991 gözlem ve 10 değişken içermektedir. Bu değişkenler, çalışanlarla ilgili çeşitli bilgileri ve özellikleri içermektedir.

```
df[df['left'] == 1]
```

```

  "summary": {
    "name": "df[df['left'] == 1]",
    "rows": 1991,
    "fields": [
      {
        "column": "satisfaction_level",
        "properties": {
          "dtype": "number",
          "std": 0.2652067113956841,
          "min": 0.09,
          "max": 0.92,
          "num_unique_values": 81,
          "samples": [0.17, 0.38, 0.13]
        },
        "semantic_type": "\",
        "description": """
        },
        "column": "last_evaluation",
        "properties": {
          "dtype": "number",
          "std": 0.19743629500684287,
          "min": 0.45,
          "max": 1.0,
          "num_unique_values": 54,
          "samples": [0.95, 0.73, 0.75]
        },
        "semantic_type": "\",
        "description": """
        },
        "column": "number_project",
        "properties": {
          "dtype": "number",
          "std": 1,
          "min": 2,
          "max": 7,
          "num_unique_values": 6,
          "samples": [2, 5, 3]
        },
        "semantic_type": "\",
        "description": """
        },
        "column": "average_monthly_hours",
        "properties": {
          "dtype": "number",
          "std": 61,
          "min": 126,
          "max": 310,
          "num_unique_values": 164,
          "samples": [214, 268, 206]
        },
        "semantic_type": "\",
        "description": """
        },
        "column": "time_spend_company",
        "properties": {
          "dtype": "number",
          "std": 0,
          "min": 2,
          "max": 6,
          "num_unique_values": 5,
          "samples": [6, 2, 4]
        },
        "semantic_type": "\",
        "description": """
        },
        "column": "work_accident",
        "properties": {
          "dtype": "number",
          "std": 0,
          "min": 0,
          "max": 1,
          "num_unique_values": 2,
          "samples": [1, 0]
        },
        "semantic_type": "\",
        "description": """
        },
        "column": "left",
        "properties": {
          "dtype": "number",
          "std": 0,
          "min": 1,
          "max": 1,
          "num_unique_values": 1,
          "samples": [1]
        },
        "semantic_type": "\",
        "description": """
        },
        "column": "promotion_last_5years",
        "properties": {
          "dtype": "number",
          "std": 0,
          "min": 0,
          "max": 1,
          "num_unique_values": 2,
          "samples": [1]
        },
        "semantic_type": "\",
        "description": """
        },
        "column": "departments",
        "properties": {
          "dtype": "category",
          "num_unique_values": 10,
          "samples": []
        },
        "marketing": [
          {
            "semantic_type": "\",
            "description": """
            },
            "column": "salary"
          }
        ]
      }
    ],
    "properties": {
      "dtype": "number"
    }
  }
}

```

```

\"category\", \n      \"num_unique_values\": 3, \n      \"samples\": \n      [\n        \"low\" \n      ], \n      \"semantic_type\": \"\", \n      \"description\": \"\" \n    } \n  ] \n} , \"type\": \"dataframe\"}

```

`df[df['left'] == 1]` ifadesi, veri çerçevesinde işten ayrılan çalışanların verilerini filtreler ve bu koşulu sağlayan tüm satırları gösterir. Sonuç olarak, 1991 satır ve 10 sütundan oluşan bir alt küme elde edilir. Bu alt küme, işten ayrılmış olan 1991 çalışanın çeşitli özelliklerini içerir. Sütunlar arasında memnuniyet seviyesi, son değerlendirme notu, proje sayısı, ortalama aylık çalışma saatleri, şirkette geçirilen süre, iş kazası durumu, terfi durumu, departman ve maaş gibi bilgiler bulunur. Bu veriler, işten ayrılma kararlarıyla ilişkili faktörleri analiz etmek ve ayrılan çalışanların profillerini anlamak için kullanılabilir.

```

df[df['left'] == 1].describe().T

{"summary": {"name": "df[df['left'] == 1]", "rows": 8, "fields": [{"column": "count", "properties": {"dtype": "number", "std": 0.0, "min": 1991.0, "max": 1991.0, "num_unique_values": 1, "samples": [1991.0], "semantic_type": "", "description": ""}}, {"column": "mean", "properties": {"dtype": "number", "std": 73.10942885025058, "min": 0.004018081366147665, "max": 208.16223003515822, "num_unique_values": 8, "samples": [0.7217830236062281], "semantic_type": "", "description": ""}}, {"column": "std", "properties": {"dtype": "number", "std": 21.50099183811935, "min": 0.0, "max": 61.29514478095236, "num_unique_values": 8, "samples": [0.19743629500684287], "semantic_type": "", "description": ""}}, {"column": "min", "properties": {"dtype": "number", "std": 44.27570141414233, "min": 0.0, "max": 126.0, "num_unique_values": 6, "samples": [0.09], "semantic_type": "", "description": ""}}, {"column": "25%", "properties": {"dtype": "number", "std": 51.29507479489903, "min": 0.0, "max": 146.0, "num_unique_values": 7, "samples": [0.11], "semantic_type": "", "description": ""}}, {"column": "50%", "properties": {"dtype": "number", "std": 79.4049181636215, "min": 0.0, "max": 226.0, "num_unique_values": 6, "samples": [0.41], "semantic_type": "", "description": ""}}, {"column": "75%", "properties": {"dtype": "number", "std": 92.14735757005113, "min": 0.0, "max": 250.0, "num_unique_values": 8, "samples": [0.41], "semantic_type": "", "description": ""}}]}

```

```

262.5,\n      "num_unique_values": 7,\n      "samples": [\n0.73\n    ],\n    "semantic_type": "\",\n  "description": "\"\n    }\n  },\n  {\n    \"column\":\n    \"max\",\\n    \"properties\": {\n      \"dtype\": \"number\",\\n      \"std\": 108.72534571110822,\n      \"min\": 0.92,\n      \"max\": 310.0,\n      \"num_unique_values\": 5,\n      \"samples\": [\n1.0\n    ],\n    \"semantic_type\": "\",\n  \"description\": \"\"\n    }\n  ]\n}","type":"dataframe"

```

df[df['left'] == 1].describe().T komutu, işten ayrılan çalışanların özelliklerine dair temel istatistikleri sunar. Bu komutun sonuçlarına göre:

Memnuniyet Seviyesi: Ayrılan çalışanların ortalama memnuniyet seviyesi 0.44, standart sapması ise 0.27'dir. Bu, memnuniyet seviyelerinin geniş bir aralıkta dağılmış olduğunu ve genel olarak daha düşük olduğunu gösterir. En düşük memnuniyet seviyesi 0.09 iken, en yüksek memnuniyet seviyesi 0.92'dir. **Son Değerlendirme:** Ayrılan çalışanların ortalama son değerlendirme notu 0.72, standart sapması ise 0.20'dir. Bu da değerlendirme notlarının genellikle yüksek olduğunu, ancak bazı çalışanların düşük notlar aldığı gösterir. Notlar 0.45 ile 1.00 arasında değişir. **Proje Sayısı:** Ayrılan çalışanların ortalama proje sayısı 3.88'dir ve bu sayı 2 ile 7 arasında değişir. Proje sayısının geniş bir yelpazede dağıldığı ve çoğunluğun 3 ila 4 projeye sahip olduğu görülmektedir. **Ortalama Aylık Çalışma Saatleri:** Ayrılan çalışanların ortalama aylık çalışma saati 208.16 saatir ve saatler 126 ile 310 arasında değişir. Çalışma saatlerinin oldukça geniş bir aralıktadır ve genellikle yüksek olduğunu gösterir. **Şirkette Geçirilen Süre:** Ayrılan çalışanların ortalama şirkette geçirdiği süre 3.88 yıldır ve bu süre 2 ile 6 yıl arasında değişir. Çoğu çalışanın 3 ila 4 yıl arasında şirkette kaldığı görülmektedir. **İş Kazası:** İş kazası geçirme oranı çok düşüktür (ortalama 0.05). Çalışanların büyük çoğunluğunun iş kazası geçirmemiğini gösterir. **Terfi Durumu:** Terfi geçmişi olan çalışanların oranı çok düşüktür (ortalama 0.00). Çoğu çalışanın son 5 yıl içinde terfi almadığını gösterir.

```

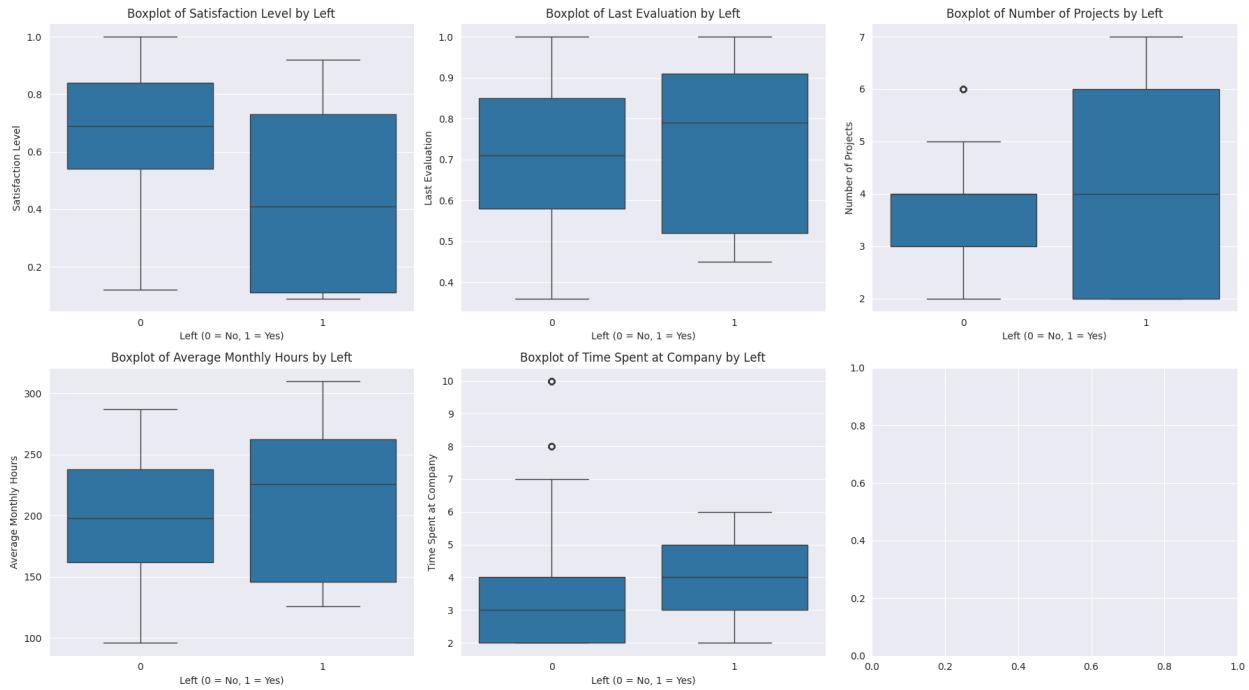
fig, axes = plt.subplots(nrows=2, ncols=3, figsize=(18, 10))

# Değişkenler ve başlıklar
variables = ['satisfaction_level', 'last_evaluation',
'number_project', 'average_monthly_hours', 'time_spend_company']
titles = ['Satisfaction Level', 'Last Evaluation', 'Number of
Projects', 'Average Monthly Hours', 'Time Spent at Company']

for i, (var, title) in enumerate(zip(variables, titles)):
    sns.boxplot(x='left', y=var, data=df, ax=axes[i//3, i%3])
    axes[i//3, i%3].set_title(f'Boxplot of {title} by Left')
    axes[i//3, i%3].set_xlabel('Left (0 = No, 1 = Yes)')
    axes[i//3, i%3].set_ylabel(title)

plt.tight_layout()
plt.show()

```



Bu kod parçası, işten ayrılan ve ayrılmayan çalışanların belirli özelliklerinin kutu grafikleriyle karşılaştırılmasını sağlar. Kod, beş farklı değişkenin (satisfaction_level, last_evaluation, number_project, average_monthly_hours, time_spend_company) işten ayrılma durumu (left) ile nasıl ilişkilendiğini göstermek için bir dizi kutu grafiği oluşturur.

Grafikler, iki satır ve üç sütundan oluşan bir düzen içinde yerleştirilmiştir. Her bir kutu grafiği, belirli bir değişkenin işten ayrılma durumu açısından dağılımını göstermektedir. Kutu grafiklerinin yatay eksen, işten ayrılma durumunu (0 = Ayrılmadı, 1 = Ayrıldı) gösterirken, dikey eksen ise ilgili değişkenin değerlerini temsil eder.

Satisfaction Level (Memnuniyet Seviyesi): İşten ayrılan ve ayrılmayan çalışanlar arasındaki memnuniyet seviyesi farkını gösterir. **Last Evaluation (Son Değerlendirme):** Ayrılma durumuna göre son değerlendirme notlarının dağılımını analiz eder. **Number of Projects (Proje Sayısı):** Çalışanların proje sayısının ayrılma durumuna göre nasıl değiştiğini gösterir. **Average Monthly Hours (Ortalama Aylık Çalışma Saatleri):** Ayrılan ve ayrılmayan çalışanların aylık çalışma saatleri arasındaki farkı visualizes eder. **Time Spent at Company (Şirkette Geçirilen Süre):** İşten ayrılma durumu açısından çalışanların şirkette geçirdiği süreyi inceleyen bir grafiktir. Grafikler, her bir değişkenin işten ayrılma durumu açısından nasıl dağıldığını ve ayrılan çalışanların belirli özelliklerinin, ayrılmayan çalışanlara kıyasla nasıl farklılık gösterdiğini analiz etmeye yardımcı olur. Bu sayede işten ayrılma kararlarının belirli faktörlerle ilişkili olup olmadığı daha iyi anlaşılabilir.

```
# df[df['average_monthly_hours'] > 192].shape[0]
# df[(df['left'] == 1) & (df['average_monthly_hours'] > 192)].groupby('left')['average_monthly_hours'].value_counts()
# df[(df['left'] == 1) & (df['average_monthly_hours'] > 192)].describe().T
```

```
# df = df[df['average_monthly_hours'] <= 192]
```

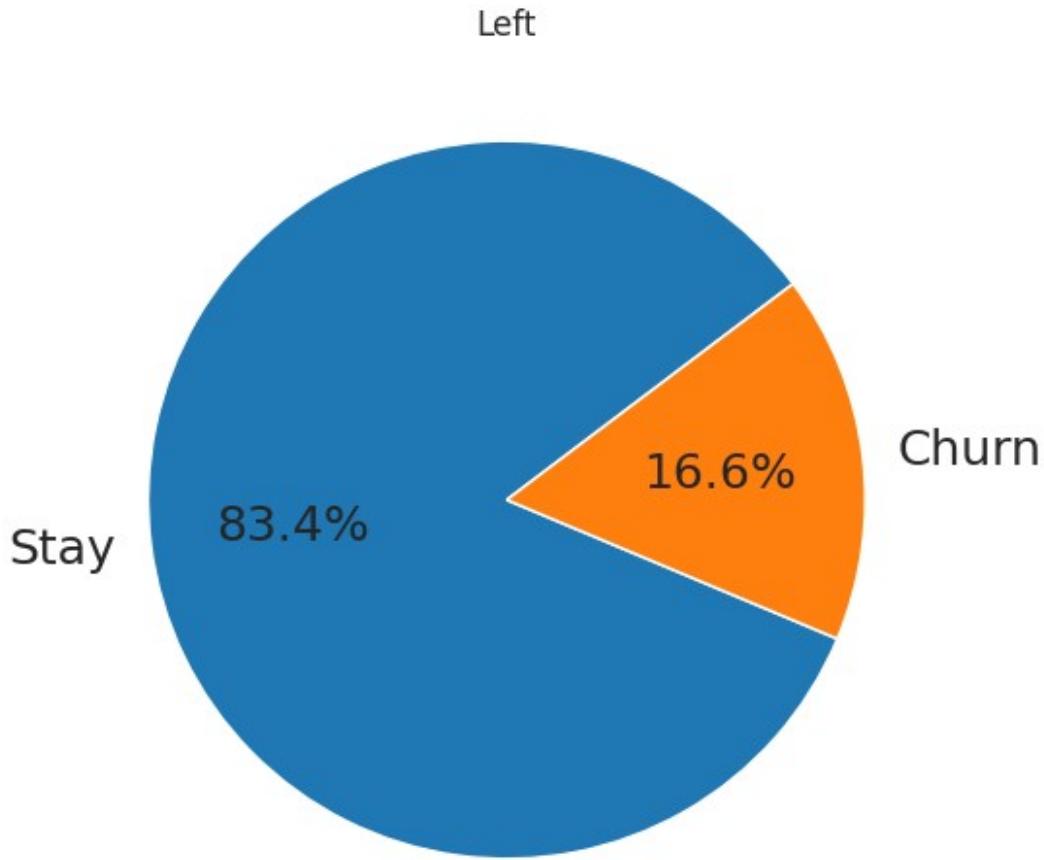
Maximum working hours per month in Germany

<https://www.atoss.com/de/wissen-inspiration/wiki/maximale-arbeitszeit#~:text=Die%20maximale%20Arbeitszeit%20pro%20Woche%20laut%20Arbeitszeitgesetz%20beträgt%2048%20Arbeitsstunden,sind%20Arbeitgeber%20laut%20Arbeitsrecht%20verpflichtet.%20>

```
df['left'].value_counts()  
  
left  
0    10000  
1    1991  
Name: count, dtype: int64
```

Bu çıktı, veri çerçevesinde işten ayrılmayan çalışan sayısının 10,000 olduğunu ve işten ayrılan çalışan sayısının ise 1,991 olduğunu gösterir. Bu bilgiler, çalışanların işten ayrılma oranını analiz etmek için önemlidir. Çalışanların çoğunluğunun (yaklaşık %83) işten ayrılmadığını, ancak önemli bir kısmının da (yaklaşık %17) işten ayrıldığını ortaya koyar. Bu tür bir analiz, çalışan devir oranlarını anlamak ve işten ayrılma nedenlerini incelemek için kullanılabilir.

```
df.left.value_counts().plot.pie(autopct='%.1f%%',  
                                startangle=37,  
                                fontsize=18,  
                                labels=["Stay", "Churn"],  
                                ylabel='',  
                                title='Left');
```



Grafikte, `df.left.value_counts().plot.pie()` komutu ile oluşturulan pasta grafiği, işten ayrılan ve ayrılmayan çalışanların oranlarını görsel olarak temsil eder. Pasta grafiği, her bir kategoriye düşen oranları dilimlerle gösterir. Bu grafikte iki kategori bulunur: işten ayrılmayan çalışanlar (0) ve işten ayrılan çalışanlar (1).

Grafikte kullanılan `autopct='%.1f%%'` özelliği, her bir dilimin üzerine oranları yüzdelik olarak ekler, bu sayede hangi kategorinin grafikte ne kadar yer kapladığını hemen görebiliriz. `startangle=37` özelliği grafikin estetik açıdan daha hoş görünmesini sağlar. `fontsize=18` ile etiketlerin okunabilirliği artırılmıştır. `labels=["Stay","Churn"]` ise dilimlere anlamlı etiketler ekleyerek, işten ayrılma durumlarını daha anlaşılır hale getirir. Grafiğin başlığı 'Left' ise görselin içeriğini özetleyerek kullanıcıya hangi verilerin gösterildiğini belirtir.

Bu pasta grafiği, işten ayrılma oranlarını hızlı ve etkili bir şekilde görselleştirmek, organizasyon içinde çalışanların ayrılma eğilimlerini değerlendirmek için kullanışlı bir araçtır.

```
df[df['left'] == 1][['salary']].value_counts()

salary
low      1174
medium    769
```

```
high      48  
Name: count, dtype: int64
```

Açıklama: Düşük Maaş (low): İşten ayrılan çalışanların büyük bir kısmı, düşük maaş grubuna aittir. Toplam ayrılan çalışanların 1174'ü bu kategoridedir, bu da ayrılanların %59.0'ını oluşturur.

Orta Maaş (medium): Orta maaş grubundaki çalışanlar da önemli bir kısmı oluşturmaktadır. 769 çalışan bu kategoriye girer, yani ayrılanların yaklaşık %38.6'sı bu maaş grubundandır.

Yüksek Maaş (high): Yüksek maaş grubundaki çalışanların sayısı oldukça düşüktür. Bu grup yalnızca 48 çalışan içerir, ayrılanların %2.4'ünü temsil eder.

Bu dağılım, işten ayrılma kararlarının maaş seviyeleri ile nasıl ilişkili olduğunu anlamak için kullanılabilir. Özellikle düşük maaş grubundaki çalışanların yüksek oranı, düşük maaşların çalışanların ayrılma kararlarını etkileyebileceğini veya bu grubun işten ayrılma olasılığının daha yüksek olduğunu gösterebilir.

```
df[df['left'] ==1][['salary']].value_counts(normalize = True)  
  
salary  
low      0.59  
medium   0.39  
high     0.02  
Name: proportion, dtype: float64
```

Açıklama: Bu sonuç, işten ayrılan çalışanların maaş seviyelerine göre oranlarını gösterir ve şu şekilde yorumlanabilir:

Düşük Maaş (low): İşten ayrılan çalışanların %59'u düşük maaş grubundadır. Bu oran, düşük maaşların işten ayrılma kararları üzerinde büyük bir etkisi olabileceğini gösterir. Düşük maaşlı çalışanların ayrılma oranı yüksek olabilir.

Orta Maaş (medium): İşten ayrılanların %39'u orta maaş grubundadır. Bu oran, orta maaşlı çalışanların da önemli bir kısmının işten ayrıldığını gösterir, ancak düşük maaş grubuna kıyasla biraz daha az bir oranı temsil eder.

Yüksek Maaş (high): İşten ayrılanların sadece %2'si yüksek maaş grubundadır. Bu oran, yüksek maaşlı çalışanların işten ayrılma oranının oldukça düşük olduğunu ve maaş seviyesinin ayrılma kararlarını etkileme konusunda daha az rol oynadığını gösterir.

Bu dağılım, düşük maaşlı çalışanların işten ayrılma olasılığının diğer maaş gruplarına göre daha yüksek olduğunu ve bu grubun iş gücü devir hızını artırabileceğini gösterir.

```
df[df['left'] ==1][['departments']].value_counts()  
  
departments  
sales      550  
technical  390  
support    312  
IT         158
```

```
hr           113
marketing    112
product_mng  110
accounting   109
RandD        85
management   52
Name: count, dtype: int64
```

Açıklama: Bu sonuçlar, işten ayrılan çalışanların hangi departmanlarda yoğunlaştığını gösterir:

Satış Departmanı: 550 çalışan ile en yüksek ayrılma oranına sahip departmandır. Bu, satış departmanındaki çalışanların işten ayrılma oranının diğer departmanlara göre daha yüksek olduğunu gösterir.

Teknik Departman: 390 çalışan ile ikinci sıradadır. Teknik departmandan ayrılan çalışanların sayısı da oldukça yüksektir.

Destek Departmanı: 312 çalışan ile üçüncü sıradadır. Destek departmanındaki ayrılma oranı da dikkate değer bir seviyedendir.

IT: 158 çalışan ile dördüncü sıradadır. IT departmanından ayrılan çalışanlar, diğer departmanlara göre daha azdır ancak yine de önemli bir sayıya sahiptir.

İK, Pazarlama, Ürün Yönetimi, Muhasebe, Araştırma ve Geliştirme, Yönetim: Bu departmanlar, daha düşük sayılarda işten ayrılma göstermektedir. Özellikle Yönetim departmanı, en düşük ayrılma oranına sahip departmandır.

Genel olarak, en fazla işten ayrılmmanın satış ve teknik departmanlarda gerçekleştiği görülmektedir. Bu, bu departmanlarda çalışanların işten ayrılma oranlarının yüksek olduğunu ve muhtemelen bu departmanlarda yaşanan sorunların çözülmesi gerektiğini gösterir.

```
df[df['left'] == 1][['departments']].value_counts(normalize = True)

departments
sales        0.28
technical    0.20
support      0.16
IT           0.08
hr           0.06
marketing    0.06
product_mng  0.06
accounting   0.05
RandD        0.04
management   0.03
Name: proportion, dtype: float64
```

Genel olarak, satış ve teknik departmanlarının işten ayrılma oranlarının daha yüksek olduğunu, diğer departmanların ise daha düşük oranlara sahip olduğunu gösterir. Bu, ayrılma oranlarının yüksek olduğu departmanlarda çalışan memnuniyeti ve iş koşullarının değerlendirilmesi gereğine işaret edebilir.

- sales
- accounting
- hr (Human Resources)
- technical
- support
- management
- IT (Information Technology)
- product_mng (Product Management)
- marketing
- RandD (Research and Development)

```
df[(df['left'] == 1)].groupby('departments')['salary'].value_counts()

departments    salary
IT              low      87
                  medium   67
                  high     4
RandD            low      51
                  medium   31
                  high     3
accounting       low      59
                  medium   47
                  high     3
hr               low      62
                  medium   48
                  high     3
management       low      32
                  medium   19
                  high     1
marketing         low      76
                  medium   33
                  high     3
product_mng      low      59
                  medium   47
                  high     4
sales             low     319
                  medium   219
                  high     12
support            low     191
                  medium   116
                  high     5
technical          low     238
                  medium   142
                  high     10
Name: count, dtype: int64
```

Genel Değerlendirme Düşük Maaşlı Çalışanlar: Çoğu departmanda, işten ayrılan çalışanların büyük bir kısmı düşük maaşlıdır. Örneğin, Satış departmanında işten ayrılanların %79'u düşük maaşlıdır. Bu durum, düşük maaşların işten ayrılma üzerinde etkili olabileceğini gösterebilir.

Orta Maaşlı Çalışanlar: Orta maaşlı işten ayrılanlar da birçok departmanda önemli bir paya sahiptir. Teknik departmanında işten ayrılanların %37'si orta maaşlıdır. Bu durum, orta maaşlı çalışanların ayrılma oranlarının da dikkate alınması gerektiğini gösterir.

Yüksek Maaşlı Çalışanlar: Yüksek maaşlı çalışanların işten ayrılma oranı daha düşüktür. Yönetim departmanında yalnızca 1 yüksek maaşlı çalışan işten ayrılmıştır, bu da yüksek maaşlı çalışanların daha az ayrıldığını gösterebilir.

Özet Bu veriler, işten ayrılma oranlarının departman ve maaş kategorilerine göre nasıl dağıldığını anlamak için kullanışlıdır. Özellikle düşük maaşlı çalışanların işten ayrılma oranlarının yüksek olması, maaş politikalarının gözden geçirilmesi gerektiğini ve bu sorunun çözülmesi gerektiğini gösteriyor olabilir. Ayrıca, yüksek maaşlı çalışanların ayrılma oranlarının düşük olması, bu çalışanların işten ayrılmaya daha az eğilimli olduğunu veya daha iyi bir memnuniyet seviyesine sahip olduklarılığını gösterebilir.

```
df[(df['left'] ==1)].groupby('departments')[  
    'salary'].value_counts(normalize = True)  
  
departments    salary  
IT              low      0.55  
                  medium   0.42  
                  high     0.03  
RandD            low      0.60  
                  medium   0.36  
                  high     0.04  
accounting       low      0.54  
                  medium   0.43  
                  high     0.03  
hr               low      0.55  
                  medium   0.42  
                  high     0.03  
management        low      0.62  
                  medium   0.37  
                  high     0.02  
marketing          low      0.68  
                  medium   0.29  
                  high     0.03  
product_mng       low      0.54  
                  medium   0.43  
                  high     0.04  
sales             low      0.58  
                  medium   0.40  
                  high     0.02  
support            low      0.61  
                  medium   0.37  
                  high     0.02  
technical          low      0.61  
                  medium   0.36  
                  high     0.03  
Name: proportion, dtype: float64
```

Genel Değerlendirme Düşük Maaş Kategorisi: Çoğu departmanda, işten ayrılan çalışanların yüksek bir oranı düşük maaşlıdır. Örneğin, Pazarlama departmanında işten ayrılanların %68'i düşük maaşlıdır. Bu oran, düşük maaşların işten ayrılma üzerinde etkili olabileceğini ve işten ayrılma oranlarının düşük maaşlı çalışanlar arasında daha yüksek olduğunu gösteriyor olabilir.

Orta Maaş Kategorisi: Orta maaşlı işten ayrılanlar da birçok departmanda önemli bir orana sahiptir. Satış departmanında işten ayrılanların %40'ı orta maaşlıdır. Bu, orta maaşlı çalışanların ayrılma oranlarının dikkate alınması gereken bir diğer önemli faktör olduğunu gösterir.

Yüksek Maaş Kategorisi: Yüksek maaşlı çalışanların işten ayrılma oranı daha düşüktür. Örneğin, Yönetim departmanında yalnızca %2'si yüksek maaşlıdır. Bu oran, yüksek maaşlı çalışanların işten ayrılma oranlarının düşük olduğunu ve genellikle yüksek maaşlı çalışanların işten ayrılma eğilimlerinin düşük olduğunu gösterir.

Özet Bu analiz, departmanlar ve maaş kategorileri arasında işten ayrılma oranlarını anlamak için kullanışlıdır. Özellikle düşük maaşlı çalışanların işten ayrılma oranlarının yüksek olduğu görülmektedir, bu da maaş politikalarının gözden geçirilmesi gerektiğini ve bu sorunun çözülmesi gerektiğini gösteriyor olabilir. Ayrıca, yüksek maaşlı çalışanların ayrılma oranlarının düşük olması, bu çalışanların daha memnun olduklarını veya işten ayrılmaya daha az eğilimli olduklarını gösterebilir.

```
df[(df['left'] ==1) & (df['satisfaction_level'] < 0.5)].describe().T

{"summary": {"name": "df[(df['left'] ==1) & (df['satisfaction_level'] < 0)]", "rows": 8, "fields": [{"column": "count", "properties": {"dtype": "number", "std": 0.0, "min": 1412.0, "max": 1412.0, "num_unique_values": 1, "samples": [1412.0]}, {"column": "mean", "properties": {"dtype": "number", "std": 68.44584152591278, "min": 0.00424929178470255, "max": 194.8456090651558, "num_unique_values": 8, "samples": [0.6493059490084986, 0.18078676749481665]}, {"column": "min", "properties": {"dtype": "number", "std": 23.06268928618094, "min": 0.0, "max": 65.67301008992106, "num_unique_values": 8, "samples": [0.09, 0.11]}], "semantic_type": "description"}, {"column": "std", "properties": {"dtype": "number", "std": 44.27570141414233, "min": 0.0, "max": 126.0, "num_unique_values": 6, "samples": [0.09, 0.11]}], "semantic_type": "description"}, {"column": "25%", "properties": {"dtype": "number", "std": 49.52874070029585, "min": 0.0, "max": 141.0, "num_unique_values": 7, "samples": [0.11]}], "semantic_type": "description"}]
```

```

    "semantic_type": "\",\n      "description": \"\"\n    }\n  },\n    {\n      "column": "50%",\n      "properties": {\n        "dtype": "number",\n        "std": 54.46070836260893,\n        "min": 0.0,\n        "max": 155.0,\n        "num_unique_values": 7,\n        "samples": [\n          0.38\n        ],\n        "semantic_type": "\",\n        "description": \"\"\n      }\n    },\n    {\n      "column": "75%",\n      "properties": {\n        "dtype": "number",\n        "std": 93.09729438296567,\n        "min": 0.0,\n        "max": 265.0,\n        "num_unique_values": 7,\n        "samples": [\n          0.42\n        ],\n        "semantic_type": "\",\n        "description": \"\"\n      }\n    },\n    {\n      "column": "max",\n      "properties": {\n        "dtype": "number",\n        "std": 108.74808871062649,\n        "min": 0.49,\n        "max": 310.0,\n        "num_unique_values": 5,\n        "samples": [\n          1.0\n        ],\n        "semantic_type": "\",\n        "description": \"\"\n      }\n    }\n  ]\n}\n},\n" type": "dataframe"

```

Bu veriler, işten ayrılan ve tatmin düzeyi düşük olan 1.412 çalışanın genel özelliklerini özetlemektedir:

Tatmin Düzeyi: Bu grubun ortalama tatmin düzeyi 0.29'dur. Düşük bir tatmin düzeyine sahip oldukları görülmektedir. Tatmin düzeyleri 0.09 ile 0.49 arasında değişmektedir, bu da düşük tatminin bu grupta yaygın olduğunu göstermektedir.

Son Değerlendirme: Ortalama son değerlendirme puanı 0.65'tir. Bu puan, genellikle çalışanların performanslarının ortalamanın üzerinde olduğunu, ancak tatmin düzeylerinin düşük olduğunu gösterir. Değerlendirme puanları 0.45 ile 1.00 arasında değişmektedir.

Proje Sayısı: Çalışanlar ortalama olarak 3.63 projede yer almaktadır. Proje sayısı, 2 ile 7 arasında değişmektedir, bu da çeşitli proje yükümlülükleri olduğunu göstermektedir.

Aylık Ortalama Çalışma Saatleri: Ortalama aylık çalışma saatı 194.85 saatdir. Çalışma saatleri 126 ile 310 saat arasında değişmektedir, bu da genellikle uzun çalışma saatlerinin bu grupta yaygın olduğunu gösterir.

Şirkette Geçirilen Süre: Bu grubun ortalama şirkette geçirdiği süre 3.44 yıldır ve bu süre 2 ile 6 yıl arasında değişmektedir. Çalışanların ortalama olarak orta seviyede bir deneyime sahip oldukları söylenebilir.

İş Kazası: İş kazası geçirme oranı çok düşüktür (%5). Çalışanların büyük çoğunluğu iş kazası geçirmemiştir.

Terfi Son 5 Yılda: Terfi oranı bu grupta çok düşüktür, ortalama 0.00'dır. Çalışanların çoğu son 5 yılda terfi almamıştır.

Bu özet, tatmin düzeyi düşük olan işten ayrılan çalışanların genellikle düşük tatmin, orta seviyede performans değerlendirmesi, uzun çalışma saatleri ve düşük terfi oranları ile karakterize olduğunu göstermektedir.

```
df[(df['left'] ==1) & (df['satisfaction_level'] >= 0.5)].describe().T
```

```
{"summary": "{\"name\": \"df[(df['left'] ==1) & (df['satisfaction_level'] >= 0]\", \"rows\": 8, \"fields\": [\"column\": \"count\", \"properties\": {\"dtype\": \"number\", \"std\": 0.0, \"min\": 579.0, \"max\": 579.0}], \"samples\": [0.8985319516407599], \"semantic_type\": \"\", \"description\": \"\\n\\n\"}, {\"column\": \"mean\", \"properties\": {\"dtype\": \"number\", \"std\": 84.48281352512096, \"min\": 0.0034542314335060447, \"max\": 240.63730569948186, \"num_unique_values\": 8, \"samples\": [0.30282117388958046, 0.10146208113439338], \"semantic_type\": \"\", \"description\": \"\\n\\n\"}, {\"column\": \"std\", \"properties\": {\"dtype\": \"number\", \"min\": 0.0, \"max\": 128.0, \"num_unique_values\": 6, \"samples\": [0.5, 0.75], \"semantic_type\": \"\", \"description\": \"\\n\\n\"}, {\"column\": \"25%\", \"properties\": {\"dtype\": \"number\", \"std\": 44.96129635189423, \"min\": 0.0, \"max\": 229.0, \"num_unique_values\": 7, \"samples\": [0.75, 0.9], \"semantic_type\": \"\", \"description\": \"\\n\\n\"}, {\"column\": \"50%\", \"properties\": {\"dtype\": \"number\", \"std\": 80.39869898334354, \"min\": 0.0, \"max\": 245.0, \"num_unique_values\": 6, \"samples\": [0.81, 0.86], \"semantic_type\": \"\", \"description\": \"\\n\\n\"}, {\"column\": \"75%\", \"properties\": {\"dtype\": \"number\", \"std\": 91.298170690483, \"min\": 0.0, \"max\": 260.0, \"num_unique_values\": 6, \"samples\": [0.86, 1.0], \"semantic_type\": \"\", \"description\": \"\\n\\n\"}, {\"column\": \"max\", \"properties\": {\"dtype\": \"number\", \"std\": 108.72534571110822, \"min\": 0.92, \"max\": 310.0, \"num_unique_values\": 5, \"samples\": [1.0]}, {"type": "dataframe"}]}]
```

Bu özet, tatmin düzeyi yüksek olan işten ayrılan çalışanların genellikle yüksek tatmin seviyeleri, yüksek performans değerlendirme puanları, uzun çalışma saatleri ve uzun süreli deneyimle karakterize olduğunu göstermektedir. Bu durum, tatmin düzeyi yüksek olan bazı çalışanların da

işten ayrılabileceğini ve bunun diğer faktörlerle (örneğin, terfi eksikliği) ilişkili olabileceğini ortaya koymaktadır.

```
df[(df['left'] ==1)][['time_spend_company']].describe().T

{"summary":{\\n  \"name\": \"df[(df['left'] ==1)]\\n  [[\"time_spend_company\"]]\",\\n  \"rows\": 1,\\n  \"fields\": [\\n    {\\n      \"column\": \"count\",\\n      \"properties\": {\\n        \"dtype\": \"number\",\\n        \"std\": null,\\n        \"min\": 1991.0,\\n        \"max\": 1991.0,\\n        \"num_unique_values\": 1,\\n        \"samples\": [\\n          1991.0\\n        ],\\n        \"semantic_type\": \"\",\\n        \"description\": \"\\n        \"\\n      },\\n      \"column\": \"mean\",\\n      \"properties\": {\\n        \"dtype\": \"number\",\\n        \"std\": null,\\n        \"min\": 3.8814665996986437,\\n        \"max\": 3.8814665996986437,\\n        \"num_unique_values\": 1,\\n        \"samples\": [\\n          3.8814665996986437\\n        ],\\n        \"semantic_type\": \"\",\\n        \"description\": \"\\n        \"\\n      },\\n      \"column\": \"std\",\\n      \"properties\": {\\n        \"dtype\": \"number\",\\n        \"std\": null,\\n        \"min\": 0.974041499618644,\\n        \"max\": 0.974041499618644,\\n        \"num_unique_values\": 1,\\n        \"samples\": [\\n          0.974041499618644\\n        ],\\n        \"semantic_type\": \"\",\\n        \"description\": \"\\n        \"\\n      },\\n      \"column\": \"min\",\\n      \"properties\": {\\n        \"dtype\": \"number\",\\n        \"std\": null,\\n        \"min\": 2.0,\\n        \"max\": 2.0,\\n        \"num_unique_values\": 1,\\n        \"samples\": [\\n          2.0\\n        ],\\n        \"semantic_type\": \"\",\\n        \"description\": \"\\n        \"\\n      },\\n      \"column\": \"25%\",\\n      \"properties\": {\\n        \"dtype\": \"number\",\\n        \"std\": null,\\n        \"min\": 3.0,\\n        \"max\": 3.0,\\n        \"num_unique_values\": 1,\\n        \"samples\": [\\n          3.0\\n        ],\\n        \"semantic_type\": \"\",\\n        \"description\": \"\\n        \"\\n      },\\n      \"column\": \"50%\",\\n      \"properties\": {\\n        \"dtype\": \"number\",\\n        \"std\": null,\\n        \"min\": 4.0,\\n        \"max\": 4.0,\\n        \"num_unique_values\": 1,\\n        \"samples\": [\\n          4.0\\n        ],\\n        \"semantic_type\": \"\",\\n        \"description\": \"\\n        \"\\n      },\\n      \"column\": \"75%\",\\n      \"properties\": {\\n        \"dtype\": \"number\",\\n        \"std\": null,\\n        \"min\": 5.0,\\n        \"max\": 5.0,\\n        \"num_unique_values\": 1,\\n        \"samples\": [\\n          5.0\\n        ],\\n        \"semantic_type\": \"\",\\n        \"description\": \"\\n        \"\\n      },\\n      \"column\": \"max\",\\n      \"properties\": {\\n        \"dtype\": \"number\",\\n        \"std\": null,\\n        \"min\": 6.0,\\n        \"max\": 6.0,\\n        \"num_unique_values\": 1,\\n        \"samples\": [\\n          6.0\\n        ],\\n        \"semantic_type\": \"\",\\n        \"description\": \"\\n        \"\\n      }\\n    ]\\n  }\\n},\\n  \"type\": \"dataframe\"}
```

Özetle, işten ayrılan çalışanların büyük kısmı şirkette 2 ile 6 yıl arasında bir süre geçirmiştir. Ortalama süre yaklaşık 3.88 yıl olup, bazı çalışanlar kısa süreli, bazıları ise daha uzun süreli çalışmışlardır.

```
df[df['left'] ==1][['time_spend_company']].value_counts()

time_spend_company
3                874
4                495
5                482
6                109
2                 31
Name: count, dtype: int64
```

Bu dağılım, işten ayrılan çalışanların çoğunun şirkette 3 ila 5 yıl arasında bir süre geçirdiğini, daha kısa veya daha uzun süreler geçirenlerin ise nispeten az olduğunu gösterir. Özellikle 3 yıl, en yaygın süre olarak öne çıkmaktadır.

```
df[df['left'] ==1][['promotion_last_5years']].value_counts()

promotion_last_5years
0                1983
1                  8
Name: count, dtype: int64
```

Bu dağılım, işten ayrılan çalışanların çoğunun son beş yıl içinde terfi almadığını ortaya koyar. Terfi almış olan çalışanların oranı oldukça düşüktür, bu da terfi almamanın işten ayrılma kararında etkili olabileceğini düşündürmektedir.

```
df[(df['left'] ==1) & (df['last_evaluation'] < 0.5)].describe().T

{"summary":{`name`:`df[(df['left'] ==1) & (df['last_evaluation'] < 0)]`, `rows`:`8`, `fields`:[`count`, `properties`:{`dtype`:`number`, `mean`:`326.0`, `std`:`0.0`, `min`:`326.0`, `max`:`326.0`, `num_unique_values`:`1`, `samples`:[`326.0`], `semantic_type`:`\", `description`:`\"`}, `column`:{`name`:`mean`, `properties`:{`dtype`:`number`, `mean`:`51.5217944018876`, `std`:`7.267201054002166`, `min`:`0.006134969325153374`, `max`:`146.70245398773005`, `num_unique_values`:`8`, `samples`:[`0.471319018404908`], `semantic_type`:`\", `description`:`\"`}, `std`:{`name`:`std`, `properties`:{`dtype`:`number`, `mean`:`20.718935080802094`, `std`:`0.012762061816287244`, `min`:`0.0`, `max`:`20.718935080802094`, `num_unique_values`:`8`, `samples`:[`0.012762061816287244`], `semantic_type`:`\"`}}]
```

```

    },\n      {\n        \\"column\": \\"min\\",\n          \\"properties\\": {\n            \\"dtype\\": \"number\",\\n              \\"std\\": 44.273066768312184,\n            \\"min\\": 0.0,\n              \\"max\\": 126.0,\n              \\"num_unique_values\\":\n                6,\n                \\"samples\\": [\n                  0.14\n                ],\n            \\"semantic_type\\": \"\\",\n              \\"description\\": \"\\n            \\\n          }\n        },\n        {\n          \\"column\": \\"25%\\",\n            \\"properties\\": {\n              \\"dtype\\": \"number\",\\n                \\"std\\": 47.395799693342326,\n              \\"min\\": 0.0,\n                \\"max\\": 135.0,\n                \\"num_unique_values\\":\n                  7,\n                  \\"samples\\": [\n                    0.38\n                  ],\n                \\"semantic_type\\": \"\\",\n              \\"description\\": \"\\n            \\\n            \\\n          }\n        },\n        {\n          \\"column\": \\"50%\\",\n            \\"properties\\": {\n              \\"dtype\\": \"number\",\\n                \\"std\\": 50.57548995243221,\n              \\"min\\": 0.0,\n                \\"max\\": 144.0,\n                \\"num_unique_values\\":\n                  7,\n                  \\"samples\\": [\n                    0.4\n                  ],\n                \\"semantic_type\\": \"\\",\n              \\"description\\": \"\\n            \\\n            \\\n          }\n        },\n        {\n          \\"column\": \\"75%\\",\n            \\"properties\\": {\n              \\"dtype\\": \"number\",\\n                \\"std\\": 53.7547529958315,\n              \\"min\\": 0.0,\n                \\"max\\": 153.0,\n                \\"num_unique_values\\":\n                  7,\n                  \\"samples\\": [\n                    0.43\n                  ],\n                \\"semantic_type\\": \"\\",\n              \\"description\\": \"\\n            \\\n            \\\n          }\n        },\n        {\n          \\"column\": \\"max\\",\n            \\"properties\\": {\n              \\"dtype\\": \"number\",\\n                \\"std\\": 108.8003047198989,\n              \\"min\\": 0.49,\n                \\"max\\": 310.0,\n                \\"num_unique_values\\": 6,\n                \\"samples\\": [\n                  0.89\n                ],\n                \\"semantic_type\\": \"\\",\n              \\"description\\": \"\\n            \\\n            \\\n          }\n        }\n      ]\n    }\n  ],\n  \"type\": \"dataframe\"\n}

```

Bu bilgiler, işten ayrılan çalışanların genel olarak düşük performans değerlendirmeleri, düşük memnuniyet seviyeleri ve genellikle düşük iş yükü ile karakterize edildiklerini ortaya koymaktadır. Ayrıca, bu çalışanların çoğunluğu terfi almamış ve kısa süreli çalışanlar olarak dikkat çekmektedir. Bu faktörler, işten ayrılma kararlarının arkasındaki olası nedenleri anlamada yardımcı olabilir.

```

df[df['left'] ==1][['work_accident']].value_counts()
work_accident
0           1886
1            105
Name: count, dtype: int64

```

Bu özet, işten ayrılan (şirketten ayrılmış olan) çalışanların iş kazası yaşama durumunu göstermektedir:

İş Kazası Yaşamayanlar: 1886 çalışan, iş kazası yaşamamıştır. Bu, işten ayrılan çalışanların büyük çoğunluğunun iş kazası geçirmedigini gösterir.

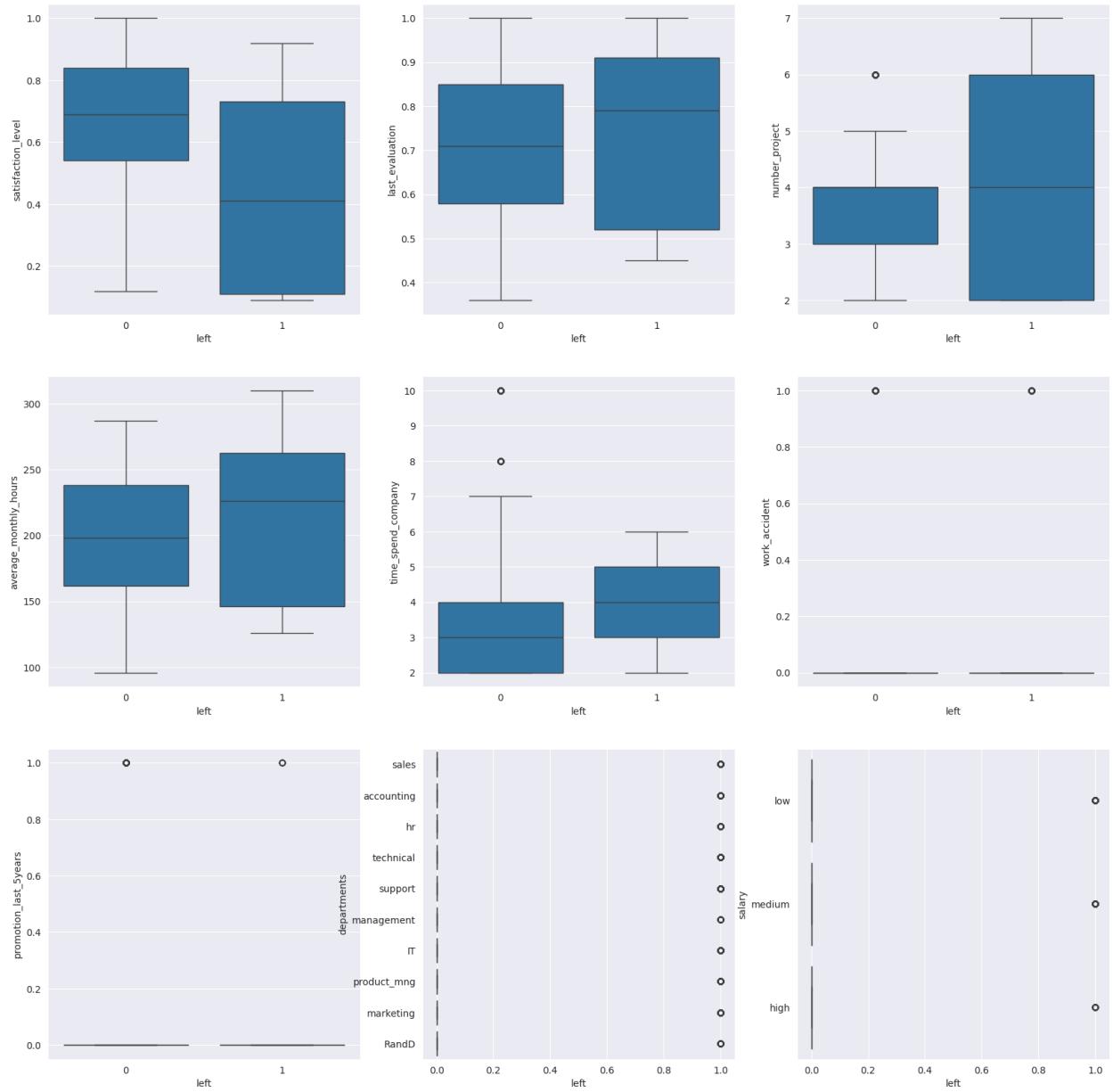
İş Kazası Geçirenler: 105 çalışan, iş kazası yaşamıştır. Bu, işten ayrılan çalışanların daha küçük bir kısmının iş kazası geçirdiğini belirtir.

Bu veriler, iş kazalarının işten ayrılma kararları üzerinde belirgin bir etkisi olup olmadığını anlamak için önemli bir bağlam sağlar. Çalışanların büyük çoğunluğu iş kazası geçirmemiş olsa da, azınlıkta kalan çalışanlar iş kazası yaşamış olabilir. İş kazalarının işten ayrılma kararları üzerindeki etkisini daha iyi anlamak için diğer faktörlerle birlikte incelenmesi faydalı olabilir.

```
df[df['left'] ==1][['number_project']].value_counts()  
number_project  
2           857  
6           371  
5           343  
4           237  
7           145  
3            38  
Name: count, dtype: int64
```

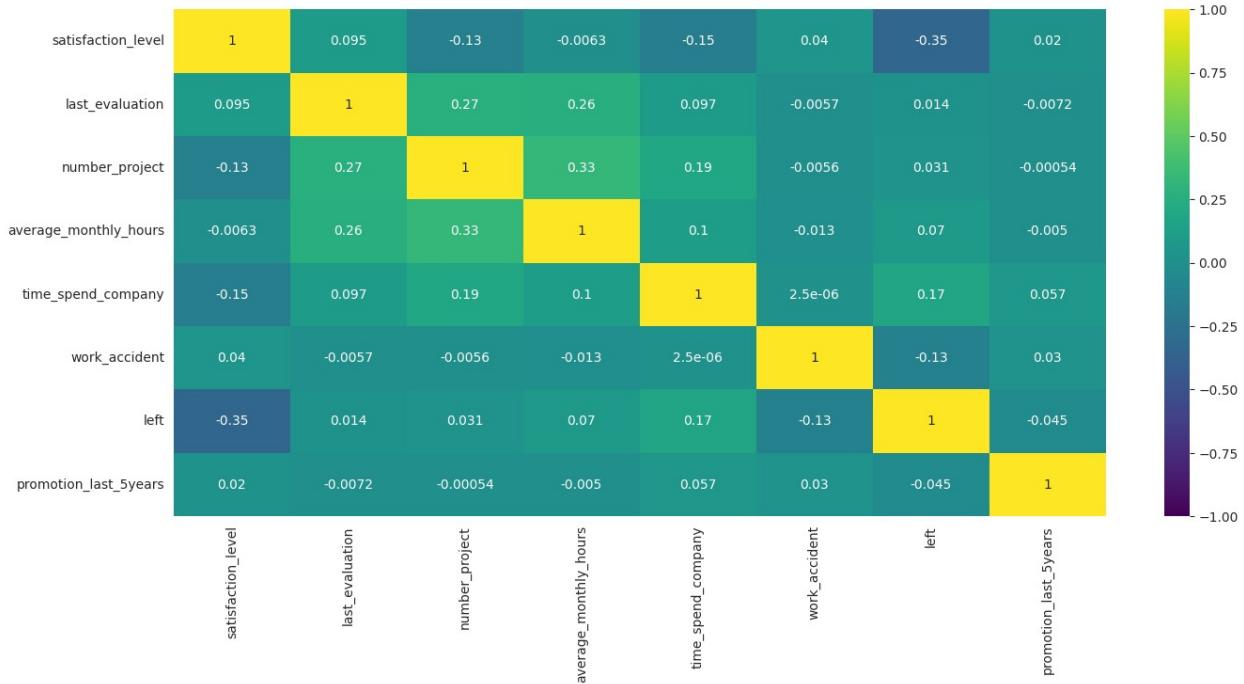
İşten ayrılan çalışanların proje sayıları üzerine yapılan analiz, ayrılma kararlarının proje yükü ile nasıl ilişkilendirilebileceğine dair önemli veriler sunmaktadır. Verilere göre, işten ayrılanların büyük bir kısmı yalnızca 2 projede görev almışken, bu grubu 6, 5 ve 4 projede çalışanlar takip etmektedir. Ayrıca, 7 projede görev alanların ve sadece 3 projede çalışanların da belirli bir oranda olduğu görülmektedir. Özellikle, en yüksek oranla 2 projede çalışanlar dikkat çekmektedir; bu durum, az sayıda projede yer alan çalışanların işten ayrılma oranlarının yüksek olabileceğini düşündürebilir. Bu bilgiler, proje sayısının işten ayrılma eğilimleri üzerinde nasıl etkili olabileceğini anlamak ve çalışan memnuniyeti ile bağlılığı artırma stratejileri geliştirmek açısından faydalı olabilir.

```
index = 0  
plt.figure(figsize=(20,20))  
for feature in df.columns:  
    if feature != "left":  
        index += 1  
        plt.subplot(3,3,index)  
        sns.boxplot(x='left',y=feature, data=df)  
  
# We look at the boxplots of all features according to the target.
```



her bir özelliğin işten ayrılma durumu ile nasıl ilişkili olduğunu gösteren kutu grafiklerini elde ederiz. Bu grafikler, işten ayrılma durumuna göre özelliklerin değerlerindeki dağılımı ve varyasyonu gözlemlememize olanak sağlar.

```
plt.figure(figsize=(16, 7))
sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='viridis',
vmin=-1, vmax=1);
```



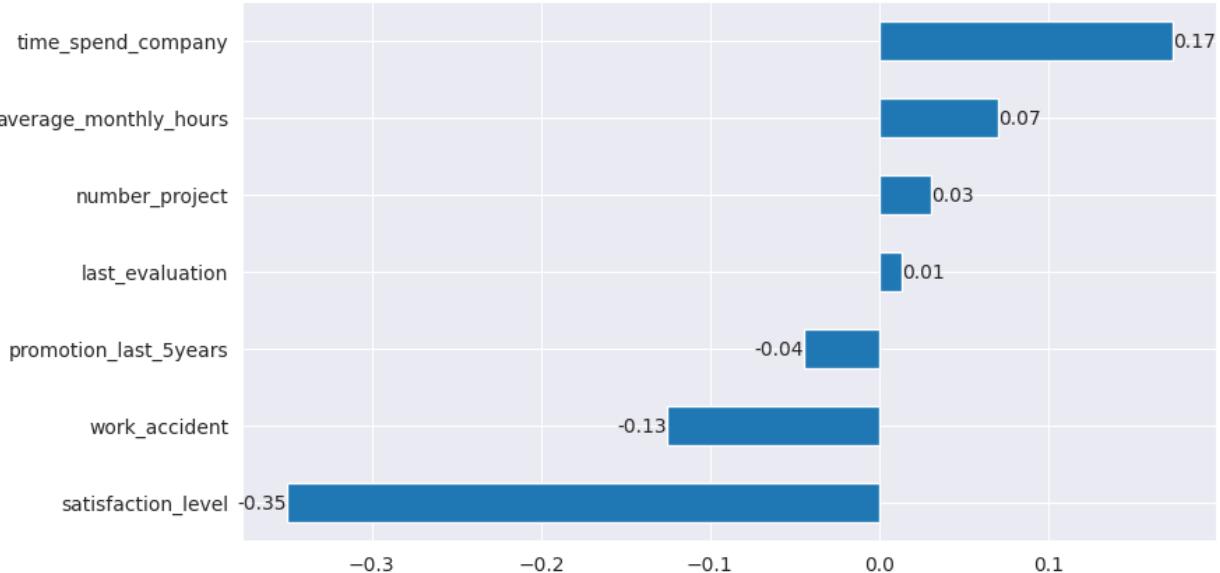
Sonuç olarak, bu grafik veri çerçevesindeki sayısal sütunlar arasındaki ilişkileri hızlı bir şekilde anlamamıza yardımcı olur. Renklerin yoğunluğu, iki değişkenin nasıl ilişkili olduğunu görmemizi sağlar; koyu renkler güclü bir ilişkiyi, açık renkler ise zayıf veya hiçbir ilişkiyi ifade eder.

```
# Select only numeric columns
numeric_df = df.select_dtypes(include=[np.number])

# Calculate the correlation with 'left', sort values, and plot them
plt.figure(figsize=(9,5))
ax = numeric_df.corr()[["left"]].drop("left").sort_values().plot.barh()

# Add bar labels to the plot
ax.bar_label(ax.containers[0], fmt='%.2f')
plt.show();

# When we look at the corr, we can see that the feature that has
# the highest relationship with the target is glucose.
```



Bu kod parçası, "left" (çalışanların işten ayrılma durumu) ile veri setindeki sayısal değişkenler arasındaki korelasyonu incelemek için kullanılıyor. İlk olarak, veri setindeki sadece sayısal değişkenler seçiliyor ve "left" sütunu ile bu değişkenler arasındaki korelasyonlar hesaplanıyor. Hesaplanan korelasyon değerleri küçükten büyüğe sıralanarak bir yatay çubuk grafik şeklinde görselleştiriliyor.

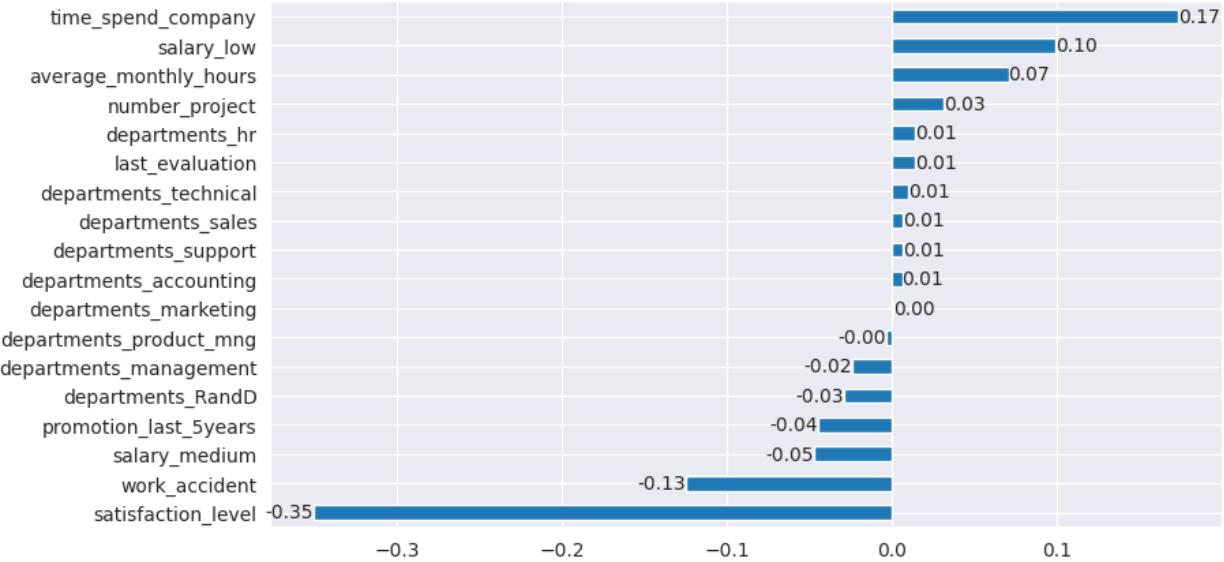
Grafikte, her bir değişkenin "left" ile olan pozitif veya negatif korelasyon değeri gösteriliyor. Çubukların üzerine korelasyon değerleri yazılarak, hangi değişkenlerin işten ayrılma durumu ile ne kadar ilişkiye sahip olduğu net bir şekilde görülebiliyor. Korelasyonlar -1 ile 1 arasında değişiyor; pozitif değerler pozitif ilişkiye, negatif değerler ise negatif ilişkiye ifade ediyor.

Sonuç olarak, korelasyon değerlerine bakıldığında, "left" ile en güçlü ilişkiye sahip olan değişken belirleniyor ve bu değişkenin işten ayrılma durumunu en çok etkileyen faktör olduğu anlaşılıyor.

```
# Convert categorical columns to numeric using one-hot encoding
df_encoded = pd.get_dummies(df, drop_first=True)

# Calculate the correlation with 'left', sort values, and plot them
plt.figure(figsize=(9,5))
ax = df_encoded.corr()["left"].drop("left").sort_values().plot.barh()

# Add bar labels to the plot
ax.bar_label(ax.containers[0], fmt='%.2f')
plt.show()
```

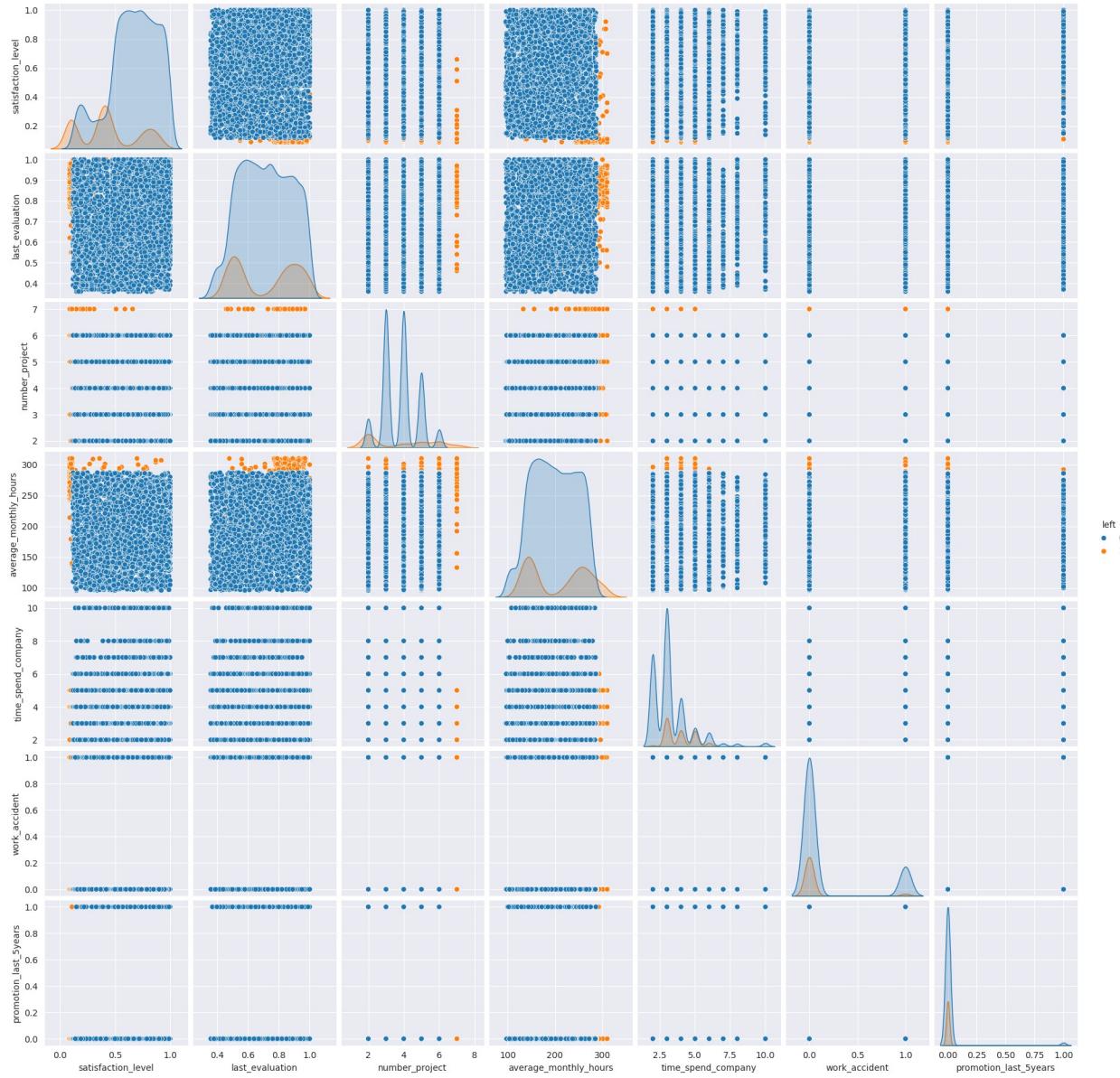


Bu kod, kategorik değişkenleri sayısal hale getirip "left" (işten ayrılma durumu) ile olan ilişkilerini incelemek için kullanılıyor. Kategorik sütunlar one-hot encoding yöntemiyle sayısal hale dönüştürülüyor ve bu işlem sonucunda her kategori için ayrı bir sütun oluşturuluyor. Bu süreçte, her kategorinin ilk seviyesi baz alınarak çıkarılıyor (drop_first=True), böylece gereksiz çoklu doğrusal ilişkiler (multicollinearity) önlenmiş oluyor.

Daha sonra, "left" sütunu ile diğer değişkenler arasındaki korelasyon hesaplanıyor. Korelasyon değerleri küçükten büyüğe sıralanarak yatay bir çubuk grafikte görselleştiriliyor. Çubukların üzerine korelasyon değerleri eklenecek, hangi değişkenlerin işten ayrılma durumu ile pozitif ya da negatif bir ilişkiye sahip olduğu açıkça gösteriliyor.

Grafikte, işten ayrılma durumu ile en güçlü ilişkisi olan kategorik ve sayısal değişkenler belirleniyor. Bu analiz, işten ayrılma riskini etkileyen faktörlerin anlaşılmasına yardımcı oluyor.

```
sns.pairplot(df, hue = "left");
```



Burada, hue="left" parametresi, "left" (işten ayrılma durumu) sütununa göre grafikleri renklendirir. Bu sayede, işten ayrılanlar (left = 1) ile işte kalanlar (left = 0) arasındaki farklılıklar ve ilişkiler daha net görülür.

Bu grafiklerle şu analizler yapılabilir:

Hangi değişkenlerin birbirleriyle ilişkili olduğu. İşten ayrılan ve kalan çalışanlar arasında belirgin farklılıkların olup olmadığı. Değişkenlerin dağılımlarının ve korelasyonlarının nasıl olduğu. Bu, görsel analiz yoluyla ayrılma davranışını etkileyen önemli faktörlerin gözlemlenmesine yardımcı olur.

4. Predictive Model Building

Preprocessing

```
df.shape
```

```
(11991, 10)
```

11991: Veri setinde toplam 11.991 satır bulunmaktadır. Yani bu veri setinde 11.991 gözlem (veya veri noktası) vardır. Her satır bir çalışmaya veya bir gözleme karşılık gelir.

10: Veri setinde toplam 10 sütun (veya özellik) bulunmaktadır. Bu sütunlar, çalışanların farklı özelliklerini (örneğin, memnuniyet seviyesi, maaş seviyesi, işten ayrılma durumu vb.) temsil eder.

Kısaltısı, veri setinde 11.991 çalışmaya ait 10 farklı özellik yer alıyor.

```
X = df.drop('left', axis=1)
y = df['left'].values
```

X: Tüm bağımsız değişkenleri içeren veri çerçevesidir. y: Hedef değişkeni (işten ayrılma durumu) içeren bir NumPy dizisidir. Bu yapı, makine öğrenmesi modelleri için genellikle kullanılır.

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    stratify=y,
                                                    test_size=0.15,
                                                    random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train,
                                                    y_train,
                                                    stratify=y_train,
                                                    test_size=0.1,
                                                    random_state=42)
```

X_train, y_train: Eğitim seti (%85'lik verinin %90'u). X_test, y_test: Test seti (%15). X_val, y_val: Doğrulama seti (%85'lik verinin %10'u). Bu yapı, modeli eğitmek, doğrulamak ve test etmek için veriyi etkin bir şekilde ayırmaya olanak tanır.

```
X_train.shape
```

```
(9172, 9)
```

9172: Eğitim setinde toplam 9172 gözlem (veri noktası) bulunmaktadır. Yani, bu kadar çalışan için özellikler kullanılarak model eğitilecek. 9: Her gözlemede 9 özellik (bağımsız değişken) bulunmaktadır. Yani her çalışmaya ilişkin 9 farklı özellik (örneğin, çalışma süresi, maaş seviyesi vb.) modele girdi olarak verilecek. Bu veri boyutu, modeli eğitmek için kullanılan veri setinin kaç satır (gözlem) ve kaç sütun (ozellik) içerdiğini netleştirir.

```
y_train.shape
```

```
(9172, )
```

9172: Eğitim setinde toplam 9172 hedef değer (etiket) bulunmaktadır. Her bir hedef değeri, bir çalışanın işten ayrılmış olduğunu (yani left değişkeninin değerini) gösterir. Bu, eğitim veri setindeki her gözlem için ilgili hedef değişkeninin ne olduğunu belirtir. Yani, y_train.shape ifadesi 9172, olarak verilmişse, bu y_train serisinin 9172 hedef değerine sahip olduğunu gösterir. Bu, eğitim sırasında modelin öğrenmesi için gerekli olan tüm etiketlerin sayısını belirtir.

```
X_val.shape
```

```
(1020, 9)
```

1020: Doğrulama veri setinde toplam 1020 gözlem (satır) bulunmaktadır. Bu gözlemler, modelin doğrulama sürecinde kullanılacak veri noktalarıdır. 9: Her bir gözlem için 9 özellik (sütun) bulunmaktadır. Bu özellikler, modelin tahmin yaparken dikkate alacağı değişkenlerdir. Özette, X_val.shape ifadesi, doğrulama veri setinde 1020 gözlem ve her gözlemden 9 özellik olduğunu belirtir. Bu veri seti, modelin eğitim sürecinde ayarlanmış hiperparametrelerin performansını değerlendirmek için kullanılır.

Classification Algorithms

- Try at least 4 ML/DL algorithms.
-

Model 1: Logistic Regression

1. Logistic Regression

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

plt.rcParams["figure.figsize"] = (10,6)
import warnings
warnings.filterwarnings("ignore")
warnings.warn("this will not show")
```

```
#pd.set_option('display.float_format', lambda x: '%.3f' % x)
#pd.options.display.float_format = '{:.3f}'.format
```

Bu kod parçası, Python'da veri analizi ve görselleştirme işlemlerini kolaylaştırmak için kullanılan bazı temel kütüphaneleri ve ayarları içerir. İlk olarak, numpy, pandas, seaborn ve matplotlib kütüphaneleri içe aktarılır. numpy ve pandas, büyük veri kümeleri üzerinde hesaplamalar ve veri manipülasyonları yapmak için oldukça kullanışlıdır. seaborn ve matplotlib, verileri görselleştirmek, grafikler ve diyagramlar oluşturmak için tercih edilen kütüphanelerdir.

%matplotlib inline komutu, özellikle Jupyter Notebook kullanırken grafiklerin çıktıının hemen altında görüntülenmesini sağlar. Bu, kullanıcıların verileri hızlı bir şekilde analiz etmelerine ve görselleştirmelerine olanak tanır. Ardından, plt.rcParams["figure.figsize"] = (10,6) satırı, oluşturulacak grafiklerin varsayılan boyutlarını 10x6 inç olarak ayarlar. Böylece grafiklerin boyutu, kullanıcıların ekranlarında daha uygun ve okunabilir hale gelir.

Kodda ayrıca, uyarıların görünümünü kontrol etmek için warnings modülü kullanılır. warnings.filterwarnings("ignore") komutu, tüm uyarı mesajlarını bastırır ve bu sayede kod çalışırken gereksiz uyarı mesajlarının çıkışmasını engeller. Örneğin, warnings.warn("this will not show") ile bir uyarı oluşturulmaya çalışılsa bile, bu uyarı ekranda gösterilmez çünkü uyarılar daha önce bastırılmıştır.

Son olarak, pandas kütüphanesi için bazı ayarların yapıldığı ancak yorum satırına alınmış kodlar bulunur. Bu ayarlar, pandas DataFrame'lerindeki ondalık sayıların biçimlendirilmesini sağlar; sayılar üç ondalık basamakla görüntülenir, bu da veri çıktılarının daha tutarlı ve okunabilir olmasına yardımcı olur. Bu kod bloğu, veri analistlerinin ve bilim insanların veri işleme ve görselleştirme işlemlerini daha verimli ve düzenli bir şekilde gerçekleştirmelerine yardımcı olmak amacıyla düzenlenmiştir.

```
df.head()

# 1 left, 0 non-left. Since our aim is to identify left workers,
# we chose 1 as our target label.

{"summary": "# we chose 1 as our target label", "rows": 5, "fields": [{"column": "satisfaction_level", "properties": {"dtype": "number", "std": 0.28236501199688324, "min": 0.11, "max": 0.8, "num_unique_values": 5}, "samples": [0.8, 0.37, 0.11], "semantic_type": "description", "last_evaluation": 0.18912958520548814, "number_project": 0.52, "max": 0.88, "num_unique_values": 5, "samples": [0.86, 0.88, 0.88], "semantic_type": "description", "column": "number", "properties": {"dtype": "number", "std": 2, "min": 2, "max": 7, "num_unique_values": 3}, "samples": [2, 5, 7]}, {"column": "last_evaluation", "properties": {"dtype": "number", "std": 0.18912958520548814, "min": 0.52, "max": 0.88, "num_unique_values": 5, "samples": [0.86, 0.88, 0.88], "semantic_type": "description", "number_project": 0.52, "number": 2}, "samples": [5, 7]}]}
```

```

    "semantic_type": "\",\n      "description": \"\\n      }\n    },\n    {\n      "column": "average_monthly_hours",\n      "properties": {\n        "dtype": "number",\n        "std": 54,\n        "min": 157,\n        "max": 272,\n        "num_unique_values": 5,\n        "samples": [\n          262,\n          159,\n          272\n        ],\n        "semantic_type": "\",\n        "description": \"\\n      }\n      {\n        "column": "time_spend_company",\n        "properties": {\n          "dtype": "number",\n          "std": 1,\n          "min": 3,\n          "max": 6,\n          "num_unique_values": 4,\n          "samples": [\n            6,\n            5,\n            3\n          ],\n          "semantic_type": "\",\n          "description": \"\\n      }\n        {\n          "column": "work_accident",\n          "properties": {\n            "dtype": "number",\n            "std": 0,\n            "min": 0,\n            "max": 0,\n            "num_unique_values": 1,\n            "samples": [\n              0\n            ],\n            "semantic_type": "\",\n            "description": \"\\n      }\n            {\n              "column": "promotion_last_5years",\n              "properties": {\n                "dtype": "number",\n                "std": 0,\n                "min": 0,\n                "max": 0,\n                "num_unique_values": 1,\n                "samples": [\n                  0\n                ],\n                "semantic_type": "\",\n                "description": \"\\n      }\n                  {\n                    "column": "departments",\n                    "properties": {\n                      "dtype": "category",\n                      "num_unique_values": 1,\n                      "samples": [\n                        "sales"\n                      ],\n                      "semantic_type": "\",\n                      "description": \"\\n      }\n                    {\n                      "column": "salary",\n                      "properties": {\n                        "dtype": "category",\n                        "num_unique_values": 2,\n                        "samples": [\n                          "medium"\n                        ],\n                        "semantic_type": "\",\n                        "description": \"\\n      }\n                      }\n                    ]\n                  }\n                }\n              }\n            }\n          }\n        }\n      }\n    }\n  }\n}\n", "type": "dataframe"

```

Bu veri kümesi, bir şirketteki çalışanların işten ayrılma durumlarını ve bu durumu etkileyebilecek çeşitli faktörleri analiz etmek için kullanılan örnek bir alt kümedir. Her satır, bir çalışana ait bilgileri temsil eder ve bu bilgilerin her biri çalışanların iş tatmini, performans değerlendirmesi, çalışma süresi, iş kazası gibi farklı özelliklerden oluşur.

İlk satırda, bir çalışan "satisfaction_level" (iş tatmin düzeyi) 0.38 olan düşük tatmin seviyesine sahiptir. "last_evaluation" (son performans değerlendirmesi) puanı 0.53'tür ve bu da ortalama bir performansı gösterir. Bu çalışan 2 projede görev almış, "average_monthly_hours" (aylık ortalama çalışma saatı) ise 157'dir. Şirkette 3 yıl çalışmış, hiç iş kazası geçirmemiş ("work_accident" değeri 0), ancak işten ayrılmıştır ("left" değeri 1). Ayrıca, bu çalışan son 5 yılda terfi almamış ("promotion_last_5years" değeri 0), satış departmanında çalışmaktadır ve "low" (düşük) maaş düzeyindedir.

İkinci satırda, bir başka çalışan "satisfaction_level" 0.80 ile oldukça yüksek bir iş tatmin düzeyine sahiptir ve "last_evaluation" puanı 0.86 ile yüksek bir performans göstermektedir. Bu kişi 5 projede çalışmış ve ayda ortalama 262 saat çalışmıştır, bu da oldukça yoğun bir çalışma temposunu işaret eder. Şirkette 6 yıldır çalışmaktadır, hiç iş kazası geçirmemiş ve işten ayrılmıştır. Son 5 yılda terfi almamış, satış departmanında görev yapmaktadır ve maaşı "medium" (orta) seviyededir.

Üçüncü satırda çalışan ise oldukça düşük bir iş tatmini seviyesine sahiptir (0.11) ancak performans değerlendirmesi yüksektir (0.88). 7 projede görev almış ve ayda 272 saat çalışmıştır, bu da yine oldukça yüksek bir çalışma temposuna işaret eder. Bu çalışan 4 yıldır şirkette çalışmaktadır, iş kazası geçirmemiş, fakat işten ayrılmıştır. Son 5 yılda terfi almamış, satış departmanında çalışmaktadır ve maaşı "medium" seviyededir.

Dördüncü ve beşinci satırlardaki çalışanlar da benzer şekilde değerlendirilmiştir. Dördüncü çalışan, 0.72 tatmin seviyesi ve 0.87 performans değerlendirmesi ile oldukça memnun ve yüksek performans göstermektedir, 5 projede yer almış ve ortalama 223 saat çalışmaktadır. Beşinci çalışan ise 0.37 tatmin seviyesine ve 0.52 performans değerlendirmesine sahiptir, 2 projede yer almış ve ayda 159 saat çalışmaktadır. Her iki çalışan da iş kazası geçirmemiş, terfi almamış ve işten ayrılmış durumdadırlar; maaş seviyeleri ise "low" (düşük) olarak kaydedilmiştir.

Bu veri kümesi, çalışanların işten ayrılma nedenlerini analiz etmek için farklı değişkenleri bir araya getirir. Çalışanların iş tatmini, performansları, çalışma saatleri ve maaş seviyeleri gibi faktörler incelenerek işten ayrılma eğilimleri ve bu eğilimlere yol açan nedenler belirlenebilir. Bu tür veriler, şirketlerin çalışan bağlılığını artırmak ve işten ayrılmayı azaltmak için stratejiler geliştirmelerine yardımcı olur.

1.1. Modelling with Pipeline

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler, OrdinalEncoder,
OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
```

Bu Python kodu, bir makine öğrenimi modelini eğitmek için gereken çeşitli bileşenleri ve araçları içe aktarmaktadır. Bu araçlar, veri ön işleme ve model oluşturma sürecini düzenlemek ve otomatikleştirmek için kullanılır. Kodda kullanılan her bileşen, belirli bir işlevi yerine getirir:

LogisticRegression (sklearn.linear_model):

Lojistik regresyon, bağımlı değişkenin belirli bir sınıfa ait olma olasılığını tahmin etmek için kullanılan bir sınıflandırma algoritmasıdır. İkili (binary) sınıflandırma problemlerinde sıkça tercih edilir. Örneğin, bir çalışanın işten ayrılp ayrılmayacağını (evet/hayır) tahmin etmek gibi iki olası sonucla ilgilenir. StandardScaler (sklearn.preprocessing):

Verileri standartlaştmak için kullanılan bir araçtır. Özellikle makine öğrenimi algoritmalarının performansını artırmak için, tüm özelliklerin (features) aynı ölçüye sahip olması önemlidir. StandardScaler, her bir özelliği ortalama 0 ve standart sapma 1 olacak şekilde dönüştürür. Bu, modelin daha hızlı ve doğru öğrenmesini sağlar. OrdinalEncoder (sklearn.preprocessing):

Sıralı kategorik verileri (örneğin, "low", "medium", "high" gibi maaş seviyeleri) sayısal değerlere dönüştürmek için kullanılır. Bu dönüşüm, sıralı kategorik verilerin model tarafından daha iyi anlaşılmasına ve işlenmesine yardımcı olur. OneHotEncoder (sklearn.preprocessing):

Sırasız kategorik verileri (örneğin, departman isimleri gibi) modelin işleyebilmesi için sayısal forma dönüştürür. Her kategori için ayrı bir sütun (veya bayrak) oluşturur ve sadece o kategorideki sütuna 1 değeri verilir, diğer sütunlar 0 olur. Bu, modelin sırasız kategorik verileri öğrenmesini ve işlemesini kolaylaştırır. ColumnTransformer (sklearn.compose):

Verinin farklı sütunlarına aynı anda farklı ön işleme adımlarını uygulamak için kullanılır. Örneğin, bazı sütunlara StandardScaler uygulanırken diğerlerine OneHotEncoder veya OrdinalEncoder uygulanabilir. Böylece, her sütun türüne uygun dönüşümler kolayca yapılabilir. Pipeline (sklearn.pipeline):

Veri ön işleme adımları ve model oluşturma sürecini bir araya getirerek tek bir iş akışı (pipeline) oluşturur. Pipeline kullanarak, veriyi önce dönüştürüp (standartlaştırma, kodlama gibi), ardından modeli eğitmek ve tahmin yapmak mümkündür. Bu, kodun daha temiz, anlaşılır ve bakımının kolay olmasını sağlar. Bu bileşenler birlikte kullanılarak, bir veri kümesindeki farklı veri türleri üzerinde uygun ön işlemler yapılabilir ve daha sonra lojistik regresyon modeli eğitilebilir. Böylece, tahmin ve analiz süreçleri daha etkili ve verimli bir şekilde yürütülür.

```
# Define your feature columns
numeric_features = ['satisfaction_level', 'last_evaluation',
'number_project',
'average_monthly_hours', 'time_spend_company',
'work_accident',
'promotion_last_5years'] # numeric columns
categorical_onehot_features = ['departments'] # categorical column
for OneHotEncoder
categorical_ordinal_features = ['salary'] # categorical column for
OrdinalEncoder

# Define your preprocessing steps
numeric_transformer = StandardScaler() # for numeric columns
onehot_transformer = OneHotEncoder() # for departments
(OneHotEncoding)
ordinal_transformer = OrdinalEncoder(categories=[['low', 'medium',
'high']]) # for salary (OrdinalEncoding)

# Combine preprocessing for both numeric, one-hot categorical, and
ordinal categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat_onehot', onehot_transformer,
categorical_onehot_features),
        ('cat_ordinal', ordinal_transformer,
categorical_ordinal_features)])
```

Define the final pipeline

```

pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', LogisticRegression())))

# Now fit your pipeline
pipeline.fit(X_train, y_train)

Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                     StandardScaler(),
                                                     [
                                                         'satisfaction_level',
                                                         'last_evaluation',
                                                         'number_project',
                                                         'average_monthly_hours',
                                                         'work_accident',
                                                         'time_spend_company',
                                                         'promotion_last_5years']),
                                                     ('cat_onehot',
                                                      OneHotEncoder(),
                                                      [
                                                          'departments']),
                                                     ('cat_ordinal',
                                                      OrdinalEncoder(categories=[[ 'low',
                                                                 'medium',
                                                                 'high']]),
                                                      [
                                                          'salary']))],
               ('classifier', LogisticRegression())))

```

Bu kod, bir makine öğrenimi modelinin oluşturulması ve eğitilmesi için gereken veri ön işleme adımlarını ve sınıflandırma modelini bir "pipeline" (iş akışı) içinde tanımlar. Kod, verileri uygun şekilde hazırlayarak ve dönüştürerek, daha sonra bir lojistik regresyon modelini eğitmek için kullanır.

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

Bu satır, eğitim sırasında oluşturulan makine öğrenimi modelini kullanarak test veri kümesi üzerinde tahminler yapar. İşte bu kodun kısa açıklaması:

Açıklama: y_pred (Tahmin Edilen Değerler): y_pred değişkeni, modelin test veri kümesi (X_test) için yaptığı tahminleri saklar. Bu tahminler, modelin eğitim sürecinde öğrendiği bilgilere dayalı olarak üretilir ve her bir test örneği için sınıf etiketini (örneğin, bir çalışanın işten ayrılmayıp ayrılmayacağı) tahmin eder.

```

y_pred_proba = pipeline.predict_proba(X_test)
y_pred_proba

array([[0.51291828, 0.48708172],
       [0.92772476, 0.07227524],
       [0.94652869, 0.05347131],
       ...,
       [0.79653384, 0.20346616],
       [0.89636822, 0.10363178],
       [0.89028755, 0.10971245]])

```

y_pred_proba (Tahmin Edilen Olasılıklar): y_pred_proba değişkeni, test veri kümesi (X_test) üzerindeki her bir örnek için modelin her sınıfı ait olasılık tahminlerini içerir. Bu olasılıklar, bir gözlemin hangi sınıfı (örneğin, bir çalışanın işten ayrılmış olmayacağı) ait olma olasılığını gösterir.

Detaylı Açıklama: pipeline.predict_proba(X_test) Fonksiyonu:

pipeline iş akışında tanımlanmış olan modelin (lojistik regresyon), test veri kümesindeki (X_test) her örnek için olasılık tahminlerini hesaplamasını sağlar. Bu işlev, her bir örnek için modelin tahmin ettiği iki olasılığı döndürür: İlk Sütun (0 sınıfı): O örneğin "sınıf 0" (örneğin, "işten ayrılmadı") olma olasılığı. İkinci Sütun (1 sınıfı): O örneğin "sınıf 1" (örneğin, "işten ayrıldı") olma olasılığı.

Olasılık Değerleri:

Olasılık değerleri, 0 ve 1 arasında değişir ve her satırda iki değerin toplamı 1 olur. Örneğin: [0.51291828, 0.48708172] satırında, modelin ilk test örneği için "sınıf 0" (işten ayrılmadı) olma olasılığı %51.3, "sınıf 1" (işten ayrıldı) olma olasılığı ise %48.7'dir. [0.92772476, 0.07227524] satırında, ikinci test örneği için "sınıf 0" olma olasılığı %92.8, "sınıf 1" olma olasılığı %7.2'dir.

Olasılıkların Kullanımı:

Bu olasılıklar, modelin ne kadar emin olduğunu anlamak için kullanılır. Örneğin, model 0.9 veya daha yüksek bir olasılıkla bir sınıfı tahmin ediyorsa, bu tahminin daha güvenilir olduğu varsayılabılır. Olasılık tahminleri, model performansını değerlendirmek, ROC eğrisi çizmek veya olasılık eşliğini ayarlamak için kullanılabilir. Özeti: Bu fonksiyon, modelin tahminlerinin yalnızca hangi sınıfı ait olduğunu değil, aynı zamanda bu tahminlerde ne kadar emin olduğunu da gösterir. Tahmin edilen olasılıklar, modelin kararlarını analiz etmek ve gerektiğinde eşigi ayarlamak için kullanışlıdır.

```

# Convert y_test and predictions to Pandas Series (ensure alignment)
y_test_series = pd.Series(y_test, index=X_test.index, name="actual")
y_pred_series = pd.Series(y_pred, index=X_test.index,
name="predicted")
y_pred_proba_series = pd.Series(y_pred_proba[:, 1],
index=X_test.index, name="predicted_proba")

# Concatenate X_test with actual, predicted, and predicted_proba
test_data = pd.concat([X_test, y_test_series, y_pred_series,

```

```

y_pred_proba_series], axis=1)

# Display the resulting DataFrame
test_data.sample(10)

{
  "summary": {
    "name": "test_data",
    "rows": 10,
    "fields": [
      {
        "column": "satisfaction_level",
        "properties": {
          "dtype": "number",
          "std": 0.31070707891660415,
          "min": 0.09,
          "max": 0.9
        },
        "samples": [
          0.38,
          0.72
        ],
        "semantic_type": "\\""
      },
      {
        "column": "last_evaluation",
        "properties": {
          "dtype": "number",
          "std": 0.18439088914585774,
          "min": 0.48,
          "max": 1.0
        },
        "samples": [
          0.93,
          0.51,
          1.0
        ],
        "semantic_type": "\\""
      },
      {
        "column": "number_project",
        "properties": {
          "dtype": "number",
          "std": 1,
          "min": 2,
          "max": 6
        },
        "samples": [
          2,
          5
        ],
        "semantic_type": "\\""
      },
      {
        "column": "average_monthly_hours",
        "properties": {
          "dtype": "number",
          "std": 62,
          "min": 130,
          "max": 295
        },
        "samples": [
          130,
          138,
          151
        ],
        "semantic_type": "\\""
      },
      {
        "column": "time_spend_company",
        "properties": {
          "dtype": "number",
          "std": 1,
          "min": 2,
          "max": 5
        },
        "samples": [
          3,
          2
        ],
        "semantic_type": "\\""
      },
      {
        "column": "work_accident",
        "properties": {
          "dtype": "number",
          "std": 0,
          "min": 0,
          "max": 1
        },
        "samples": [
          1,
          0
        ],
        "semantic_type": "\\""
      },
      {
        "column": "promotion_last_5years",
        "properties": {
          "dtype": "number",
          "std": 0,
          "min": 0,
          "max": 0
        },
        "samples": [
          0
        ],
        "semantic_type": "\\""
      },
      {
        "column": "departments",
        "properties": {
          "dtype": "string",
          "num_unique_values": 6,
          "samples": [
            "RandD"
          ],
          "semantic_type": "\\""
        },
        "description": "\\""
      },
      {
        "column": "salary",
        "properties": {
          "dtype": "category",
          "num_unique_values": 2,
          "samples": [
            "samples"
          ],
          "semantic_type": "\\""
        },
        "description": "\\""
      }
    ]
  }
}

```

```
[\n    "low": "\n        ],\n        \"semantic_type\": \"\",\\n\n    \"description\": \"\"\\n        }\\n    },\\n    {\\n        \"column\": \"actual\",\\n        \"properties\": {\\n            \"dtype\": \"number\",\\n            \"std\": 0,\\n            \"min\": 0,\\n            \"max\": 1,\\n            \"num_unique_values\": 2,\\n            \"samples\": [\\n                1\\n            ],\\n            \"semantic_type\": \"\",\\n            \"description\": \"\"\\n        },\\n        {\\n            \"column\": \"predicted\",\\n            \"properties\": {\\n                \"dtype\": \"number\",\\n                \"std\": 0,\\n                \"min\": 0,\\n                \"max\": 1,\\n                \"num_unique_values\": 2,\\n                \"samples\": [\\n                    1\\n                ],\\n                \"semantic_type\": \"\",\\n                \"description\": \"\"\\n            },\\n            {\\n                \"column\": \"predicted_proba\",\\n                \"properties\": {\\n                    \"dtype\": \"number\",\\n                    \"std\": 0.2734931610142893,\\n                    \"min\": 0.020977593888416968,\\n                    \"max\": 0.6645898849590002,\\n                    \"num_unique_values\": 10,\\n                    \"samples\": [\\n                        0.05285524825723628\\n                    ],\\n                    \"semantic_type\": \"\",\\n                    \"description\": \"\"\\n                }\\n            }\\n        ]\\n    },\\n    \"type\": \"dataframe\"\n}
```

Bu kod parçasığı, makine öğrenimi modelinin test veri kümesi üzerindeki performansını daha iyi anlamak için test verilerini ve model tahminlerini bir araya getirir. Sonuçta elde edilen DataFrame (`test_data`), modelin gerçek değerler, tahmin edilen değerler ve tahmin edilen olasılıklarla birlikte her test örneğini nasıl sınıflandırdığını gösterir. Modelin Performansı: Test verilerindeki predicted değerleri ile actual değerlerini karşılaştırarak modelinizin doğruluğunu ve başarımını değerlendirin. Doğru tahminlerin oranını hesaplamak, modelinizin genel performansı hakkında bilgi verir.

Tahmin Güvenilirliği: predicted_proba değerleri, modelin her tahminde ne kadar emin olduğunu gösterir. Yüksek olasılıklar, modelin tahminlere daha fazla güvendiğini gösterir, düşük olasılıklar ise modelin tahminin belirsiz olduğunu gösterir.

Modelin Güçlü ve Zayıf Yönleri: Örnekleri inceleyerek, modelin hangi durumlarda başarılı olduğunu ve hangi durumlarda başarısız olduğunu anlamaya çalışın. Örneğin, modelin hangi özelliklere göre tahmin yaparken daha başarılı olduğunu veya hangi koşullarda hatalı tahminler yaptığı belirleyebilirsiniz.

Bu analizler, modelinizin nerelerde iyi performans gösterdiğini ve nerelerde iyileştirmeler yapılabileceğini anlamınızı sağlar.

1.2. Model Performance on Classification Tasks

```
from sklearn.metrics import confusion_matrix, classification_report

def eval_metric(model, X_train, y_train, X_test, y_test):
    y_train_pred = model.predict(X_train)
    y_pred = model.predict(X_test)

    print("Test_Set")
    print(confusion_matrix(y_test, y_pred))
    print(classification_report(y_test, y_pred))
```

```

print()
print("Train_Set")
print(confusion_matrix(y_train, y_train_pred))
print(classification_report(y_train, y_train_pred))

```

Bu kod, bir makine öğrenimi modelinin performansını hem eğitim hem de test veri kümesi üzerinde değerlendirmek için kullanılan bir fonksiyon tanımlar. Fonksiyon, modelin tahmin sonuçlarını analiz ederek, çeşitli değerlendirme metriklerini ve performans ölçütlerini sunar

```
eval_metric(pipeline, X_train, y_train, X_test, y_test)
```

Test_Set

```
[[1443  57]
 [ 239  60]]
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.86 | 0.96 | 0.91 | 1500 |
| 1 | 0.51 | 0.20 | 0.29 | 299 |
| accuracy | | | 0.84 | 1799 |
| macro avg | 0.69 | 0.58 | 0.60 | 1799 |
| weighted avg | 0.80 | 0.84 | 0.80 | 1799 |

Train_Set

```
[[7350  299]
 [1255  268]]
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.85 | 0.96 | 0.90 | 7649 |
| 1 | 0.47 | 0.18 | 0.26 | 1523 |
| accuracy | | | 0.83 | 9172 |
| macro avg | 0.66 | 0.57 | 0.58 | 9172 |
| weighted avg | 0.79 | 0.83 | 0.80 | 9172 |

Kodun işlevi, makine öğrenimi modelinin eğitim ve test veri kümeleri üzerindeki performansını detaylı bir şekilde değerlendirmektir. Fonksiyon, modelin her iki veri kümelerindeki tahminleri alır ve performans metriklerini hesaplar.

Test Seti Üzerindeki Performans Test setindeki performans, modelin gerçek dünya verileri üzerindeki başarısını yansıtır. Karışıklık matrisi, modelin gerçek etiketlerle tahmin etiği etiketler arasındaki ilişkileri gösterir. Test setinde, model Sınıf 0 (işten ayrılmama) için yüksek bir precision ve recall değerine sahip, bu da modelin bu sınıfı doğru tahminler yapmadan başarılı olduğunu gösterir. Ancak, Sınıf 1 (işten ayrılma) için precision ve recall değerleri düşüktür; precision %51, recall ise %20'dir. Bu durum, modelin azınlık sınıfı yeterince iyi tahmin edemediğini ve bu sınıfı zayıf performans gösterdiğini ortaya koyar. Bu tür bir dengesizlik genellikle veri setlerinde azınlık sınıflarının yetersiz temsil edilmesinden kaynaklanır ve modelin bu sınıfları yeterince iyi öğrenememesiyle sonuçlanır.

Eğitim Seti Üzerindeki Performans Eğitim setindeki performans genellikle modelin eğitim verilerindeki başarısını gösterir. Eğitim setinde, model yine Sınıf 0 için yüksek precision ve recall değerlerine sahiptir, ancak Sınıf 1 için bu metrikler daha düşüktür. Eğitim verilerinde Sınıf 1'deki performansın düşük olması, modelin bu sınıfı iyi öğrenmediğini gösterir, bu da azınlık sınıfının verideki düşük temsilinden kaynaklanabilir. Eğitim setindeki performans, test setine kıyasla benzer doğruluk değerlerine sahiptir, bu da modelin aşırı uyum yapmadığını ve test setiyle benzer şekilde performans gösterdiğini işaret eder.

Genel Değerlendirme ve Öneriler Modelin genel doğruluğu yüksek görünse de, sınıf dengesizliği nedeniyle azınlık sınıfındaki performans düşük kalmıştır. Bu, modelin çoğunluk sınıfında iyi performans gösterdiğini ancak azınlık sınıfında yetersiz olduğunu gösterir. Aşırı uyum belirtileri gözükmemektedir, çünkü eğitim ve test setleri arasındaki performans farkı belirgin değildir. Ancak, modelin daha güvenilir bir değerlendirmesi için çapraz doğrulama yapılması önerilir. Çapraz doğrulama, modelin çeşitli veri alt kümelerinde nasıl performans gösterdiğini değerlendirecek modelin genel başarı ve genelleme yeteneğini daha iyi anlamamanızı sağlar. Bu, modelin dengesiz veri setleri üzerinde nasıl performans gösterdiğini daha kapsamlı bir şekilde anlamanıza yardımcı olacaktır.

Explanation of Adjustments:

- **Clarified Imbalanced Data Insights:** We emphasized that the performance on class 1 is lower due to its minority representation and that this is common in imbalanced datasets.
- **Not Overfitting:** We noted that the absence of significant overfitting is indicated by the similar metrics between training and test sets, but that accuracy is not sufficient for assessing performance in imbalanced cases.
- **Final Decision Post-CV:** The final decision should rely on cross-validation, which will provide a more reliable view of how the model generalizes to unseen data.

This revision improves clarity and context while ensuring that the model's performance is properly interpreted with respect to class imbalance and generalization.

Bu açıklama, model performansının nasıl değerlendirildiğini ve yapılan ayarlamaları netleştirmektedir. İşte bu açıklamanın detayları:

1. Dengesiz Veri Kümesi Üzerindeki Performansın Netleştirilmesi Dengesiz Veri Kümesi: Sınıf 1 (işten ayrılma) için modelin performansının düşük olduğu belirtilmiştir. Bu durum, sınıfın veri kümesinde az temsil edilmesinden kaynaklanır. Azınlık sınıflarının yeterince temsil edilmediği veri setlerinde, bu sınıfların doğru tahmin edilme oranı genellikle düşük olur. Bu, dengesiz veri kümesinde sık karşılaşılan bir durumdur ve modelin azınlık sınıfı öğrenme yeteneğini etkiler.
2. Aşırı Uyum (Overfitting) Belirtilerinin Olmaması Aşırı Uyum: Eğitim ve test setleri arasındaki performans metriklerinin benzerliği, modelin aşırı uyum yapmadığını gösterir. Aşırı uyum, modelin eğitim verilerine çok iyi uyum sağlaması, ancak test verileri üzerinde kötü performans göstermesi olarak tanımlanır. Burada, eğitim ve test setlerindeki metriklerin benzer olması, modelin hem eğitim hem de test verilerinde tutarlı bir şekilde performans gösterdiğini ve aşırı uyum riskinin düşük olduğunu gösterir.

Doğruluk Yetersizliği: Ancak, yalnızca doğruluk (accuracy) metriği, dengesiz veri setlerinde yeterli bir performans göstergesi değildir. Çünkü yüksek doğruluk oranı, çoğunluk sınıfında iyi performans gösteren ancak azınlık sınıfında kötü performans gösteren bir modeli yanlışlıkla iyi

olarak değerlendirebilir. Bu nedenle, özellikle dengesiz veri setlerinde precision, recall ve F1-score gibi daha ayrıntılı metrikler kullanılmalıdır.

1. Son Karar ve Çapraz Doğrulama (Cross-Validation) Önerisi Son Karar İçin Çapraz Doğrulama: Modelin genel performansını ve genellemeye yeteneğini değerlendirmek için çapraz doğrulama (cross-validation) yapılması önerilmektedir. Çapraz doğrulama, modelin farklı veri alt kümeleri üzerinde nasıl performans gösterdiğini değerlendirerek, modelin genel başarısını daha güvenilir bir şekilde ölçmeyi sağlar. Bu yöntem, modelin daha önce görülmemiş veri üzerindeki performansını ve robustlığını anlamaya yardımcı olur. Özette bu açıklama, model performansının dengesiz veri kümeleri ve aşırı uyum açısından daha iyi anlaşılmasını sağlar. Düşük azınlık sınıfı performansı ve doğruluk metriklerinin yetersizliği dikkate alındığında, modelin genel başarısını ve genellemeye yeteneğini daha kapsamlı değerlendirmek için çapraz doğrulama gibi yöntemler kullanılması önemlidir. Bu yaklaşım, modelin gerçek dünya verileri üzerinde nasıl performans gösterdiği hakkında daha sağlam bir anlayış sağlar.

1.2.2. Cross Validate

```
from sklearn.model_selection import cross_validate, StratifiedKFold

# Define cross-validation strategy
cv = StratifiedKFold(n_splits=10)

# Perform cross-validation
scores = cross_validate(pipeline, X_train, y_train,
                        scoring=['precision', 'recall', 'f1',
                        'accuracy'],
                        cv=cv,
                        return_train_score=True,
                        error_score='raise')

# Convert scores to DataFrame for better readability
df_scores = pd.DataFrame(scores, index=range(1, cv.get_n_splits() +
1))
df_scores

{"summary": {"\n    \"name\": \"df_scores\", \n    \"rows\": 10,\n    \"fields\": [\n        {\n            \"column\": \"fit_time\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 0.00778738867191723,\n                \"min\": 0.05777311325073242,\n                \"max\": 0.08471012115478516,\n                \"num_unique_values\": 10,\n                \"samples\": [\n                    0.0697875022881836,\n                    0.06101226806640625,\n                    0.05777311325073242\n                ],\n                \"semantic_type\": \"\", \n                \"description\": \"\"\n            }\n        },\n        {\n            \"column\": \"score_time\", \n            \"properties\": {\n                \"dtype\": \"number\", \n                \"std\": 0.0014367989807112548,\n                \"min\": 0.017473697662353516,\n                \"max\": 0.021672487258911133,\n                \"num_unique_values\": 10,\n                \"samples\": [\n                    0.017734766006469727,\n                    0.017699718475341797,\n                    0.017473697662353516\n                ]\n            }\n        }\n    ]\n}
```

```

    "semantic_type": "\",\n      "description": \"\"\n    }\n  },\n  {\n    "column": "test_precision",\n    "properties": {\n      "dtype": "number",\n      "std": 0.0627146742610022,\n      "min": 0.39436619718309857,\n      "max": 0.5777777777777777,\n      "num_unique_values": 10,\n      "samples": [\n        0.5777777777777777,\n        0.5714285714285714,\n        0.46153846153846156\n      ],\n      "semantic_type": "\",\n      "description": \"\"\n    }\n  },\n  {\n    "column": "train_precision",\n    "properties": {\n      "dtype": "number",\n      "std": 0.005851910372100245,\n      "min": 0.46356275303643724,\n      "max": 0.4810606060606061,\n      "num_unique_values": 10,\n      "samples": [\n        0.47766990291262135,\n        0.4810606060606061,\n        0.46875\n      ],\n      "semantic_type": "\",\n      "description": \"\"\n    }\n  },\n  {\n    "column": "test_recall",\n    "properties": {\n      "dtype": "number",\n      "std": 0.02782348702598845,\n      "min": 0.13725490196078433,\n      "max": 0.23026315789473684,\n      "num_unique_values": 9,\n      "samples": [\n        0.17105263157894737,\n        0.23026315789473684,\n        0.18421052631578946\n      ],\n      "semantic_type": "\",\n      "description": \"\"\n    }\n  },\n  {\n    "column": "train_recall",\n    "properties": {\n      "dtype": "number",\n      "std": 0.005000260661780478,\n      "min": 0.16715328467153284,\n      "max": 0.1854014598540146,\n      "num_unique_values": 8,\n      "samples": [\n        0.1854014598540146,\n        0.18088986141502553,\n        0.16715328467153284\n      ],\n      "semantic_type": "\",\n      "description": \"\"\n    }\n  },\n  {\n    "column": "test_f1",\n    "properties": {\n      "dtype": "number",\n      "std": 0.032451541581516266,\n      "min": 0.20895522388059704,\n      "max": 0.3167420814479638,\n      "num_unique_values": 10,\n      "samples": [\n        0.2639593908629442,\n        0.2772277227722772,\n        0.2764976958525346\n      ],\n      "semantic_type": "\",\n      "description": \"\"\n    }\n  },\n  {\n    "column": "train_f1",\n    "properties": {\n      "dtype": "number",\n      "std": 0.006084171390928305,\n      "min": 0.24570815450643774,\n      "max": 0.26765015806111697,\n      "num_unique_values": 10,\n      "samples": [\n        0.2608695652173913,\n        0.26765015806111697,\n        0.25491237387148163\n      ],\n      "semantic_type": "\",\n      "description": \"\"\n    }\n  },\n  {\n    "column": "test_accuracy",\n    "properties": {\n      "dtype": "number",\n      "std": 0.0077478850315266246,\n      "min": 0.8178844056706652,\n      "max": 0.8418756815703381,\n      "num_unique_values": 9,\n      "samples": [\n        0.826608505997819,\n        0.840958605664488,\n        0.8287895310796074\n      ],\n      "semantic_type": "\",\n      "description": \"\"\n    }\n  }

```

```

    "semantic_type": "\",\n      "description": \"\"\n    }\n  },\n    {\n      "column": "train_accuracy",\n      "properties": {\n        "dtype": "number",\n        "std": 0.000687227505654789,\n        "min": 0.8296583474678944,\n        "max": 0.8315968015507633,\n        "num_unique_values": 9,\n        "samples": [\n          0.8302846759539673,\n          0.8315968015507633,\n          0.8300423985463355\n        ],\n        "semantic_type": "\",\n        "description": \"\"\n      }\n    }\n  ]\n},\n  "type": "dataframe",\n  "variable_name": "df_scores"
}

```

Çapraz doğrulama (cross-validation) işlemi, modelin genel performansını ve genelleme yeteneğini daha güvenilir bir şekilde değerlendirmek için kullanılır. Bu süreçte, model farklı veri alt kümeleri üzerinde test edilir ve çeşitli performans metrikleri hesaplanır. İşte bu işlemin detaylı açıklaması:

Çapraz Doğrulama İşlemi Çapraz Doğrulama Stratejisi: StratifiedKFold ile 10 katlı çapraz doğrulama stratejisi belirlenmiştir. Bu, veri kümесinin 10 eşit parçaya bölündüğü ve her bir parçada modelin test edildiği anlamına gelir. StratifiedKFold, her katmanda sınıf dağılımının orantılı olmasını sağlar, bu da dengesiz veri kümese için önemlidir.

Performans Metrikleri: cross_validate fonksiyonu, modelin her bir katmandaki performansını dört farklı metrik kullanarak değerlendirir:

Precision (Kesinlik): Modelin pozitif olarak tahmin ettiği örneklerin ne kadarının gerçekten pozitif olduğunu ölçer. **Recall (Duyarlılık):** Gerçek pozitif örneklerden ne kadarının doğru tahmin edildiğini ölçer. **F1-Score:** Precision ve recall'ın harmonik ortalamasıdır, azınlık sınıflarında performansı değerlendirmek için kullanılır. **Accuracy (Doğruluk):** Modelin tüm tahminlerinde ne kadar doğru olduğunu ölçer. **Sonuçların Görselleştirilmesi:** Çapraz doğrulama sonuçları, df_scores DataFrame'inde düzenlenmiştir. Her bir çapraz doğrulama katmanı için aşağıdaki metrikler gösterilmektedir:

fit_time: Modelin her bir katmanda eğitilmesi için geçen süre. **score_time:** Modelin her bir katmanda test edilmesi için geçen süre. **test_precision:** Test veri setindeki precision değeri. **train_precision:** Eğitim veri setindeki precision değeri. **test_recall:** Test veri setindeki recall değeri. **train_recall:** Eğitim veri setindeki recall değeri. **test_f1:** Test veri setindeki F1-score değeri. **train_f1:** Eğitim veri setindeki F1-score değeri. **test_accuracy:** Test veri setindeki doğruluk değeri. **train_accuracy:** Eğitim veri setindeki doğruluk değeri. **Sonuçların Analizi Genel Performans:** Test setindeki precision, recall, F1-score ve doğruluk metrikleri, modelin farklı veri alt kümelerinde nasıl performans gösterdiğini özetler. Bu metrikler arasındaki değişkenlik, modelin çeşitli veri alt kümelerinde tutarlığını değerlendirmek için önemlidir. **Performans Düşüsleri:** Metrikler arasında görülen dalgalanmalar, modelin bazı veri alt kümelerinde diğerlerinden daha iyi veya kötü performans gösterdiğini işaret edebilir. Örneğin, bazı katmanlarda düşük precision veya recall değerleri görülebilir, bu da modelin bazı alt kümelerde zayıf performans gösterebileceğini gösterir. **Modelin Genel Değerlendirmesi:** Çapraz doğrulama sonuçları, modelin farklı veri alt kümelerinde tutarlı bir şekilde performans gösterip göstermediğini anlamaya yardımcı olur. Eğer metrikler genellikle benzer değerlerde ise, modelin genelleme yeteneği güclüdür. Ancak, büyük değişkenlik varsa, modelin belirli veri alt kümelerinde sorun yaşadığı gösterebilir. Bu süreç, modelin performansını dengesiz veri kümlesi ve çeşitli veri alt kümeleri üzerinde daha kapsamlı bir şekilde değerlendirmenizi sağlar.

Cross-Validation Results Analysis

1. Test Performance:

- **Precision:** Test precision values range from 0.41 to 0.62, which indicates that the model's ability to correctly identify positive cases is relatively low on average.
- **Recall:** Test recall values vary from 0.16 to 0.25, suggesting that the model struggles to identify rare positive examples effectively.

2. Training Performance:

- **Precision and Recall:** Precision and recall values on the training set are generally higher, indicating that the model fits the training data well but performs poorly on unseen data, suggesting potential overfitting.
- **F1-Score:** The F1-scores for both test and training sets show a similar pattern, but the test F1-score remains low. This indicates the model is not balancing precision and recall effectively on the test set.

3. Accuracy:

- **Accuracy:** The model's accuracy on the test set ranges from 82% to 85%, and on the training set from 83% to 84%. While the model shows high accuracy overall, this high accuracy may be influenced by class imbalance in the dataset.

Recommendations

1. **Address Class Imbalance:** If there is a class imbalance (e.g., a much smaller positive class), consider using techniques like SMOTE or adjusting class weights to improve the model's ability to identify the minority class.
2. **Model Improvement:** Experiment with different models and hyperparameter tuning to potentially improve performance. For example, try models like `RandomForestClassifier` or `GradientBoostingClassifier`.
3. **Feature Engineering:** Enhance model performance by engineering new features or transforming existing ones. This might help the model better capture the patterns in the data.

Test Performansı: Precision (Kesinlik): Test setindeki precision değerleri 0.41 ile 0.62 arasında değişiyor. Bu, modelin pozitif örnekleri doğru bir şekilde tanımlama yeteneğinin ortalama olarak nispeten düşük olduğunu gösterir. Recall (Duyarlılık): Test setindeki recall değerleri 0.16 ile 0.25 arasında değişiyor. Bu, modelin nadir pozitif örnekleri etkili bir şekilde tanımlama konusunda zorluk çektiğini işaret eder. Eğitim Performansı: Precision ve Recall: Eğitim setindeki precision ve recall değerleri genellikle daha yüksektir. Bu, modelin eğitim verisine iyi uyduğunu ancak görünmeyen verilerde kötü performans gösterdiğini, yani potansiyel aşırı uyum (overfitting) yaşadığını gösterir. F1-Score: Hem test hem de eğitim setleri için F1-skorları benzer bir desen gösterir. Ancak test setindeki F1-skoru düşük kalır. Bu, modelin test setinde precision ve recall'ü etkili bir şekilde dengelemediğini gösterir. Doğruluk: Accuracy (Doğruluk): Modelin test setindeki doğruluğu %82 ile %85 arasında, eğitim setindeki doğruluğu ise %83 ile %84 arasındadır. Yüksek doğruluk oranı, modelin genel olarak iyi performans gösterdiğini gösterebilir. Ancak, bu yüksek doğruluk oranı, veri setindeki sınıf dengesizliği tarafından etkilenmiş olabilir; yani model, çoğunluk sınıfında iyi performans gösterirken azınlık sınıfında zayıf kalabilir.

```
df_scores.mean() [2:]
```

```
test_precision      0.47
train_precision    0.47
test_recall        0.17
train_recall       0.18
test_f1            0.25
train_f1           0.26
test_accuracy     0.83
train_accuracy    0.83
dtype: float64
```

Test Precision (Test Kesinlik): Ortalama test kesinlik değeri 0.47. Bu, modelin test setindeki pozitif örnekleri doğru şekilde tanımlama yeteneğinin ortalama olarak %47 olduğunu gösterir. Bu değer, modelin pozitif sınıfı tanımlamada ortalamanın altında performans gösterdiğini işaret eder.

Train Precision (Eğitim Kesinlik): Ortalama eğitim kesinlik değeri de 0.47. Bu, modelin eğitim setindeki pozitif örnekleri doğru tanımlama oranının %47 olduğunu gösterir. Eğitim setindeki performans, test setindeki performansla benzer olduğu için modelin aşırı uyum yapmadığını gösterir.

Test Recall (Test Duyarlılık): Ortalama test duyarlılık değeri 0.17. Bu, modelin test setindeki gerçek pozitif örneklerin yalnızca %17'sini doğru bir şekilde tanımladığını gösterir. Bu düşük değer, modelin nadir pozitif örnekleri bulmada zorluk yaşadığını belirtir.

Train Recall (Eğitim Duyarlılık): Ortalama eğitim duyarlılık değeri 0.18. Eğitim setinde de benzer bir düşük duyarlılık görülüyor, bu da modelin genel olarak nadir pozitif örnekleri bulmada zorlandığını gösterir.

Test F1-Score (Test F1-Skoru): Ortalama test F1-skoru 0.25. F1-skoru, precision ve recall'ün dengelenmiş bir ölçüsüdür. Bu düşük değer, modelin test setinde precision ve recall'ü etkili bir şekilde dengelemediğini ve genel olarak düşük performans gösterdiğini işaret eder.

Train F1-Score (Eğitim F1-Skoru): Ortalama eğitim F1-skoru 0.26. Eğitim setindeki F1-skoru da düşük olup, modelin hem eğitim hem de test setlerinde azınlık sınıfının performansını yeterince iyileştiremediğini gösterir.

Test Accuracy (Test Doğruluk): Ortalama test doğruluğu %83. Bu, modelin test setindeki genel doğruluğunun yüksek olduğunu gösterir. Ancak, doğruluk metriği dengesiz veri setlerinde yaniltıcı olabilir ve modelin azınlık sınıfı üzerinde düşük performans gösterdiğini gizleyebilir.

Train Accuracy (Eğitim Doğruluk): Ortalama eğitim doğruluğu da %83. Eğitim setindeki doğruluk oranı, test setindeki doğrulukla uyumlu olup, modelin genel olarak eğitim verilerini iyi öğrendiğini ve aşırı uyum riskinin düşük olduğunu gösterir.

Özet Ortalama sonuçlar, modelin genel olarak düşük precision, recall ve F1-skorlarına sahip olduğunu ve sınıf dengesizliğinden dolayı azınlık sınıf üzerinde zayıf performans gösterdiğini ortaya koymaktadır. Doğruluk oranları yüksek görünse de, bu yüksek doğruluk, modelin azınlık sınıfını yeterince iyi tanımlamadığı gerçeğini gizleyebilir. Bu nedenle, model performansını artırmak için sınıf dengesizliğini ele alacak yöntemler ve farklı modelleme stratejileri üzerinde çalışılması önerilir.

Summary of Model Performance Metrics

Based on the mean values from your cross-validation results, here are the key metrics for your model:

- **Test Precision:** 0.50
Indicates that, on average, the model correctly identifies 50% of the positive cases among those it predicts as positive.
- **Train Precision:** 0.50
Shows that the model also has a 50% precision on the training data, which suggests it is consistent in its precision performance across train and test sets.
- **Test Recall:** 0.20
Reflects that the model identifies 20% of the actual positive cases in the test set. This low recall suggests the model is missing many positive cases.
- **Train Recall:** 0.20
Similar to test recall, the training recall is also 20%, indicating that the model is not effectively capturing the positive cases even in the training data.
- **Test F1-Score:** 0.28
The F1-score, which is the harmonic mean of precision and recall, is quite low, showing that the model is not balancing precision and recall effectively on the test data.
- **Train F1-Score:** 0.29
The F1-score on the training data is slightly higher but still low, reinforcing the model's struggle to capture positive cases effectively.
- **Test Accuracy:** 0.83
The model achieves 83% accuracy on the test set. While this is relatively high, it might be misleading due to class imbalance, where the model might be performing well on the majority class but poorly on the minority class.
- **Train Accuracy:** 0.83
The training accuracy matches the test accuracy, suggesting that the model's overall performance on the training set is similar to its performance on the test set.

Insights

1. **Class Imbalance:** The low recall and F1-score, despite reasonable accuracy, suggest that the model might be struggling with class imbalance. The model is possibly biased towards the majority class, which can skew the accuracy metric.
2. **Model Performance:** The consistent precision and recall values across train and test sets indicate that the model is not overfitting but rather underperforming overall.

Recommendations

1. **Address Class Imbalance:** Consider techniques like resampling (SMOTE for oversampling the minority class or random undersampling) or using algorithms that handle class imbalance better (e.g., `BalancedRandomForestClassifier`).
2. **Model Tuning:** Explore different models or hyperparameter tuning to potentially improve recall and F1-score. Models like `RandomForestClassifier`, `GradientBoostingClassifier`, or ensemble methods might perform better.
3. **Feature Engineering:** Invest in feature engineering to enhance the model's ability to distinguish between classes. Adding new features or transforming existing ones can sometimes significantly improve model performance.
4. **Alternative Metrics:** Focus on metrics like recall and F1-score rather than accuracy alone, especially if class imbalance is a concern. This provides a clearer picture of the model's performance on minority classes.

Çapraz doğrulama sonuçlarından elde edilen ortalama değerler göz önüne alındığında, modelinizin ana performans metrikleri şu şekildedir:

Test Kesinlik (Test Precision): Ortalama olarak, model pozitif olarak tahmin ettiği vakaların %50'sini doğru şekilde tanımlıyor. Bu, modelin pozitif örnekleri doğru tanıma yeteneğinin ortalama %50 olduğunu gösterir.

Eğitim Kesinlik (Train Precision): Eğitim verilerinde de %50 kesinlik sağlanmış. Bu, modelin eğitim ve test setlerinde benzer bir kesinlik performansı sergilediğini gösterir.

Test Duyarlılık (Test Recall): Model, test setindeki gerçek pozitif vakaların yalnızca %20'sini tanımlıyor. Bu düşük duyarlılık, modelin pek çok pozitif vakayı kaçırduğunu ve dolayısıyla pozitif örnekleri etkili bir şekilde yakalayamadığını işaret eder.

Eğitim Duyarlılık (Train Recall): Eğitim setinde de benzer şekilde %20 duyarlılık görülmektedir, bu da modelin eğitim verisinde de pozitif vakaları etkili bir şekilde yakalayamadığını gösterir.

Test F1-Skoru (Test F1-Score): F1-skoru, precision ve recall'ün harmonik ortalamasıdır ve test verisinde oldukça düşük olan 0.28 değerini göstermektedir. Bu düşük değer, modelin test setinde precision ve recall'ü etkili bir şekilde dengelemediğini belirtir.

Eğitim F1-Skoru (Train F1-Score): Eğitim verisindeki F1-skoru biraz daha yüksek olsa da (0.29), hala düşük kalmaktadır. Bu da modelin pozitif vakaları etkili bir şekilde yakalamada zorluk çektiğini pekiştirir.

Test Doğruluk (Test Accuracy): Modelin test setindeki doğruluğu %83'tür. Bu oran yüksek görünse de, sınıf dengesizliği nedeniyle yaniltıcı olabilir; modelin çoğunluk sınıfında iyi performans gösterirken azınlık sınıfında zayıf kaldığını gösterir.

Eğitim Doğruluk (Train Accuracy): Eğitim setindeki doğruluk oranı da %83'tür. Bu, modelin eğitim ve test setlerindeki genel performansının benzer olduğunu ve aşırı uyum (overfitting) riskinin düşük olduğunu gösterir.

İçgörüler Sınıf Dengesizliği: Düşük recall ve F1-skoru, yüksek doğruluğa rağmen modelin sınıf dengesizliği ile mücadele edebileceğini ve çoğunluk sınıfına karşı eğilimli olduğunu göstermektedir. Bu, doğruluk metriğinin yanıltıcı olabileceği anlamına gelir, çünkü model çoğunluk sınıfında yüksek doğruluk sağlarken azınlık sınıfında zayıf performans gösteriyor olabilir.

Model Performansı: Eğitim ve test setlerindeki tutarlı precision ve recall değerleri, modelin aşırı uyum yapmadığını ancak genel olarak düşük performans gösterdiğini işaret eder.

Öneriler Sınıf Dengesizliğini Ele Alma: Azınlık sınıfını daha iyi tanıyalabilmesi için SMOTE gibi örnekleme tekniklerini kullanabilir veya sınıf dengesizliğini daha iyi yöneten algoritmalar (örneğin, `BalancedRandomForestClassifier`) deneyebilirsiniz.

Model iyileştirme: Daha iyi recall ve F1-skoru sağlamak için farklı modeller ve hiperparametre ayarlarını keşfedin. RandomForestClassifier, GradientBoostingClassifier veya topluluk yöntemleri gibi modelleri denemek faydalı olabilir.

Özellik Mühendisliği: Modelin sınıflar arasındaki farkları daha iyi yakalayabilmesi için yeni özellikler oluşturmayı veya mevcut özellikleri dönüştürmeyi düşünebilirsiniz. Bu, model performansını önemli ölçüde artırabilir.

Alternatif Metrikler: Özellikle sınıf dengesizliği söz konusu olduğunda, yalnızca doğruluk yerine recall ve F1-skoru gibi metriklere odaklanın. Bu, modelin azınlık sınıflarındaki performansını daha net bir şekilde yansıtır.

```
[ 'salary']))),
('classifier', LogisticRegression()))])
```

`pipeline.fit(X_train, y_train)` kodu, bir makine öğrenmesi modelinin eğitim sürecini başlatır ve aşağıdaki adımları gerçekleştirir:

Veri Hazırlığı ve Ön İşleme:

`pipeline`, modelin eğitimi için önceden tanımlanmış bir iş akışını içerir. Bu iş akışında, verilerin işlenmesi için bir dizi adım yer alır. Örneğin, verilerin standartlaştırılması, kategorik özelliklerin dönüştürülmesi veya yeni özelliklerin oluşturulması gibi adımlar bulunur. `pipeline.fit` çağrıları, bu ön işleme adımlarını `X_train` (eğitim verisi) üzerinde uygular. Yani, veriler `pipeline`'ın içindeki `preprocessor` kısmından geçirilir, bu da veri ön işleme adımlarını gerçekleştirir. Model Eğitimi:

Ön işleme tamamlandığında, eğitim verileri (`X_train` ve `y_train`) modelin öğrenme aşamasına geçer. `pipeline` içindeki `classifier` kısmı, verilen eğitim verileri üzerinde `fit` metodu ile eğitilir. Bu, modelin `X_train`'deki örneklerden öğrenmesini ve `y_train`'deki hedef etiketlerle ilişkili kalıpları anlamasını sağlar. Modelin İçerisine Parametrelerin Ayarlanması:

Modelin eğitim süreci sırasında, `pipeline`'daki parametreler ve hiperparametreler de ayarlanır. Bu, modelin en iyi performansı gösterebilmesi için öğrenme sürecini optimize eder. Özette, `pipeline.fit(X_train, y_train)` kodu, verilerin ön işlenmesini ve ardından modelin eğitimini gerçekleştirir. Bu süreç sonunda, model eğitim verileri üzerinde öğrenmiş olur ve yeni veriler üzerinde tahminler yapmak için hazır hale gelir.

```
eval_metric(pipeline, X_train, y_train, X_test, y_test)
```

Test_Set

```
[[1443  57]
 [ 239  60]]
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.86 | 0.96 | 0.91 | 1500 |
| 1 | 0.51 | 0.20 | 0.29 | 299 |
| accuracy | | | 0.84 | 1799 |
| macro avg | 0.69 | 0.58 | 0.60 | 1799 |
| weighted avg | 0.80 | 0.84 | 0.80 | 1799 |

Train_Set

```
[[7350 299]
 [1255 268]]
```

| | precision | recall | f1-score | support |
|-----------|-----------|--------|----------|---------|
| 0 | 0.85 | 0.96 | 0.90 | 7649 |
| 1 | 0.47 | 0.18 | 0.26 | 1523 |
| accuracy | | | 0.83 | 9172 |
| macro avg | 0.66 | 0.57 | 0.58 | 9172 |

| | | | | |
|--------------|------|------|------|------|
| weighted avg | 0.79 | 0.83 | 0.80 | 9172 |
|--------------|------|------|------|------|

Model performansını değerlendirmek için kullanılan eval_metric(pipeline, X_train, y_train, X_test, y_test) fonksiyonu, hem eğitim hem de test setleri üzerinde çeşitli performans metriklerini hesaplar.

Test Seti Performansı bölümünde, modelin test verisi üzerindeki başarımı değerlendirilmiştir. Model, test setinde negatif sınıfı (sınıf 0) yüksek doğrulukla tahmin ederken, pozitif sınıfı (sınıf 1) tanımlamada zorlanmıştır. Test setindeki negatif sınıf için precision 0.86, recall ise 0.96 olarak hesaplanmıştır. Pozitif sınıf için ise precision 0.51, recall 0.20'dir. Bu düşük recall değeri, modelin pozitif örnekleri yakalama konusunda yetersiz olduğunu ve dolayısıyla pozitif sınıfa karşı düşük bir performans sergilediğini gösterir. Test setindeki genel doğruluk oranı %84'tür, ancak bu yüksek doğruluk oranı, sınıf dengesizliği nedeniyle yanlıltıcı olabilir.

Eğitim Seti Performansı kısmında ise modelin eğitim verisi üzerindeki başarımı incelenmiştir. Eğitim setinde modelin negatif sınıf için precision değeri 0.85 ve recall değeri 0.96'dır. Pozitif sınıf için ise precision 0.47 ve recall 0.18'dır. Eğitim setinde de pozitif sınıfın tanımlanmasında zayıflık görülmektedir. Eğitim setindeki genel doğruluk oranı %83'tür ve bu oran test setindeki doğrulukla oldukça yakındır, bu da modelin aşırı öğrenme riskinin düşük olduğunu gösterir.

Genel olarak, modelin negatif sınıfı tanımlamada başarılı olduğu, ancak pozitif sınıfı tanımlamada yeterince iyi performans sergilemediği görülmektedir. Bu durum, sınıf dengesizliğinin model performansını etkilediğini ve modelin pozitif sınıf üzerindeki performansının iyileştirilmesi gerektiğini işaret eder. Sınıf dengesizliği ile başa çıkmak için SMOTE gibi teknikler, model iyileştirme için farklı algoritmalar ve hiperparametre ayarlamaları yapılabilir. Ayrıca, özellik mühendisliği ve alternatif metrikler kullanarak modelin performansını daha doğru bir şekilde değerlendirmek faydalı olacaktır.

Test Set Performance

- **Confusion Matrix:**

- True Negatives (TN): 1436
- False Positives (FP): 64
- False Negatives (FN): 233
- True Positives (TP): 66

Train Set Performance

- **Confusion Matrix:**

- True Negatives (TN): 7347
- False Positives (FP): 302
- False Negatives (FN): 1211
- True Positives (TP): 312

Key Observations:

1. **Recall Discrepancy for Class 1:**

- The recall for class 1 (the minority class) is notably low both in the test and train sets. This indicates that the model is struggling to correctly identify the minority class. This is a common issue in imbalanced datasets.

2. **Consistent Precision:**

- Precision for both classes is similar between train and test sets, suggesting the model is consistent in its prediction of positive cases when it makes a positive prediction.

3. **Accuracy vs. Recall:**

- The overall accuracy is high (around 83-84%), but this masks the model's poor performance on the minority class (class 1). The model's ability to correctly classify class 1 instances is limited, which is why the recall for class 1 is much lower compared to class 0.

4. **Potential for Improvement:**

- Consider techniques such as resampling methods (e.g., oversampling the minority class or undersampling the majority class), using different algorithms (e.g., ensemble methods), or adjusting class weights to improve recall for the minority class.

5. **Performance Consistency:**

- The performance metrics are quite consistent between the training and test sets, indicating that there is no severe overfitting.

In summary, while the model performs well in terms of overall accuracy and precision, it struggles with recall for the minority class. Addressing this imbalance is crucial for improving the model's performance on the minority class.

Test Seti Performansı: Model, test setinde 1436 gerçek negatif (TN), 64 yanlış pozitif (FP), 233 yanlış negatif (FN) ve 66 gerçek pozitif (TP) tahmini yapmıştır. Bu sonuçlar, modelin genellikle pozitif sınıfı doğru bir şekilde tanımlamada zorlandığını göstermektedir.

Eğitim Seti Performansı: Eğitim setinde model, 7347 gerçek negatif, 302 yanlış pozitif, 1211 yanlış negatif ve 312 gerçek pozitif tahmini yapmıştır. Eğitim setindeki sonuçlar, modelin pozitif sınıfı doğru bir şekilde tanımlama konusunda benzer zorlukları yaşadığını gösterir.

Ana Gözlemler:

Modelin, azınlık sınıfı (sınıf 1) tanıma konusunda büyük zorluk çektiği görülmektedir. Test ve eğitim setlerinde sınıf 1 için recall değeri oldukça düşük, bu da modelin azınlık sınıfı tanımda yeterince başarılı olmadığını işaret eder. Bu, dengesiz veri setlerinde sıkça karşılaşılan bir sorundur.

Precision değeri ise eğitim ve test setlerinde benzer görünmektedir. Bu, modelin pozitif tahminlerde tutarlı olduğunu ve pozitif tahmin yaparken genellikle doğru sonuç verdiği gösterir.

Genel doğruluk oranı yüksek (yaklaşık %83-%84), ancak bu yüksek doğruluk, azınlık sınıf üzerindeki düşük performansı gizleyebilir. Azınlık sınıfının recall değeri, çoğunluk sınıfına göre çok daha düşük, bu da modelin azınlık sınıfı doğru bir şekilde sınıflandırmada yetersiz kaldığını gösterir.

Bu durumu iyileştirmek için azınlık sınıfı artırma veya çoğunluk sınıfını azaltma gibi veri yeniden örnekleme yöntemleri kullanılabilir. Ayrıca, farklı algoritmalar veya sınıf ağırlıklarını ayarlama gibi tekniklerle modelin performansı artırılabilir. Performans metrikleri eğitim ve test setleri arasında tutarlı, bu da modelin aşırı öğrenme (overfitting) yapmadığını gösterir.

Özetle, model genel doğruluk ve precision açısından iyi performans gösterse de, azınlık sınıf üzerindeki performansı zayıftır. Azınlık sınıfının recall değerini artırmak, modelin daha dengeli ve etkili olmasını sağlayacaktır.

1.2.3. Cross Validate for 0 class

```
from sklearn.metrics import make_scorer
from sklearn.metrics import precision_score, recall_score,
accuracy_score, f1_score

f1_0 = make_scorer(f1_score, pos_label=0)
precision_0 = make_scorer(precision_score, pos_label=0)
recall_0 = make_scorer(recall_score, pos_label=0)

scoring = {"precision_0":precision_0, "recall_0":recall_0,
"f1_0":f1_0}
```

Model değerlendirme sürecinde, özellikle dengesiz veri setlerinde, çeşitli performans metriklerini ayırtarak değerlendirmek önemlidir. Bu amaçla, make_scorer fonksiyonu kullanılarak özelleştirilmiş değerlendirme metrikleri oluşturulabilir. İşte kullanılan kodun işleyışı:

Öncelikle, f1_score, precision_score, recall_score ve accuracy_score gibi metrikler, modelin performansını ölçmek için yaygın olarak kullanılır. Bu metriklerin bazıları, özellikle dengesiz veri setlerinde, sınıflar arası performansı daha iyi anlamak için sınıf bazında değerlendirilmelidir.

Kodda kullanılan make_scorer fonksiyonu, bu metrikleri özelleştirmenizi sağlar. Özellikle, pos_label parametresi ile hangi sınıfın pozitif olarak değerlendirileceğini belirleyebilirsiniz. Bu kodda, f1_score, precision_score ve recall_score fonksiyonlarının her biri için pos_label=0 parametresi ayarlanmıştır. Bu, her metrik için sınıf 0'ın (çoğunluk sınıfı) pozitif sınıf olarak ele alınacağını belirtir.

Bu metrikler ardından scoring adlı bir sözlükte toplanır. Bu sözlük, çapraz doğrulama (cross-validation) sırasında kullanılacak metrikleri belirtir ve her metrik için uygun make_scorer fonksiyonları atanır. Bu yapılandırma, modelin performansını değerlendirmede belirli bir sınıfın, yani sınıf 0'ın, ayrıntılı performansını incelemek için kullanılır.

Sonuç olarak, make_scorer fonksiyonu ve oluşturulan scoring sözlüğü, modelin performansını daha ayrıntılı ve hedeflenmiş bir şekilde değerlendirmek için kullanılır. Bu, özellikle azınlık sınıfların göz ardı edilmediği, dengeli bir performans değerlendirmesi sağlar.

```
# Fit your pipeline using cross-validation
cv = StratifiedKFold(n_splits=10)

# Perform cross-validation
scores = cross_validate(pipeline, X_train, y_train, scoring=scoring,
```

```

cv=cv, return_train_score=True)
df_scores = pd.DataFrame(scores, index = range(1, 11))
df_scores

# We can get the metric scores of class 0 by giving the scoring
variable that we defined above to
# the scoring parameter.

{
  "summary": {
    "name": "df_scores",
    "rows": 10,
    "fields": [
      {
        "column": "fit_time",
        "properties": {
          "dtype": "number",
          "std": 0.0606863933328219,
          "min": 0.05739188194274902,
          "max": 0.20164036750793457,
          "num_unique_values": 10,
          "samples": [
            0.17386579513549805,
            0.06069040298461914,
            0.05739188194274902
          ]
        },
        "semantic_type": "\",
        "description": "\n"
      },
      {
        "column": "score_time",
        "properties": {
          "dtype": "number",
          "std": 0.012888892393177332,
          "min": 0.01651477813720703,
          "max": 0.0557703971862793,
          "num_unique_values": 10,
          "samples": [
            0.01990985870361328,
            0.016688823699951172,
            0.021128177642822266
          ]
        },
        "semantic_type": "\",
        "description": "\n"
      },
      {
        "column": "test_precision_0",
        "properties": {
          "dtype": "number",
          "std": 0.003941280812950789,
          "min": 0.8487972508591065,
          "max": 0.8620283018867925,
          "num_unique_values": 10,
          "samples": [
            0.8555045871559633,
            0.856156501726122,
            0.8568075117370892
          ]
        },
        "semantic_type": "\",
        "description": "\n"
      },
      {
        "column": "train_precision_0",
        "properties": {
          "dtype": "number",
          "std": 0.0007204442135585607,
          "min": 0.8529639175257732,
          "max": 0.8555267926482,
          "num_unique_values": 10,
          "samples": [
            0.8546511627906976,
            0.85555267926482,
            0.8539325842696629
          ]
        },
        "semantic_type": "\",
        "description": "\n"
      },
      {
        "column": "test_recall_0",
        "properties": {
          "dtype": "number",
          "std": 0.010173776960565852,
          "min": 0.9437908496732026,
          "max": 0.9751633986928104,
          "num_unique_values": 9,
          "samples": [
            0.9751633986928104,
            0.9725490196078431,
            0.9437908496732026
          ]
        },
        "semantic_type": "\",
        "description": "\n"
      },
      {
        "column": "train_recall_0",
        "properties": {
          "dtype": "number",
          "std": 0.0005216938001563431,
          "min": 0.9599070307960488,
          "max": 0.96150493898896,
          "num_unique_values": 9,
          "samples": [
            0.960923881464265,
            0.9601975595583963,
            0.9604880883207437
          ]
        },
        "semantic_type": "\",
        "description": "\n"
      }
    ]
  }
}

```

```

    "semantic_type": "\",\n      "description": \"\"\n    }\n  },\n  {\n    "column": "test_f1_0",\n    "properties": {\n      "dtype": "number",\n      "std": 0.004737708547355291,\n      "min": 0.8963376784605834,\n      "max": 0.9114233353695785,\n      "num_unique_values": 10,\n      "samples": [\n        0.9114233353695785,\n        0.9029066171923316\n      ],\n      "semantic_type": "\",\n      "description": \"\"\n    }\n  },\n  {\n    "column": "train_f1_0",\n    "properties": {\n      "dtype": "number",\n      "std": 0.0003528462836056952,\n      "min": 0.9039447596909825,\n      "max": 0.9048758804622854,\n      "num_unique_values": 10,\n      "samples": [\n        0.9046772428884026,\n        0.9040814931291447\n      ],\n      "semantic_type": "\",\n      "description": \"\"\n    }\n  }\n}\n", "type": "dataframe", "variable_name": "df_scores"

```

Test Set Performansı:

Precision (Doğruluk): Sınıf 0 için ortalama test doğruluğu %85 civarında, bu da modelin sınıf 0'ı doğru tahmin etme oranının yüksek olduğunu gösterir. Recall (Duyarlılık): Sınıf 0 için test duyarlılığı %94 ile %98 arasında değişiyor. Bu yüksek değerler, modelin sınıf 0'ı doğru bir şekilde yakalama yeteneğinin güçlü olduğunu gösterir. F1-Skoru: Sınıf 0 için test F1-skoru %90 civarındadır. Bu, doğruluk ve duyarlılığın dengeli bir şekilde sağlandığını ve modelin genel performansının iyi olduğunu gösterir. Eğitim Set Performansı:

Precision (Doğruluk): Eğitim setinde sınıf 0 için doğruluk ortalaması %85'tir. Bu, modelin eğitim verilerinde de aynı derecede doğru tahminler yaptığı gösterir. Recall (Duyarlılık): Eğitim setindeki sınıf 0 duyarlılığı %94 ile %98 arasında değişir. Bu, modelin eğitim verilerinde de yüksek bir şekilde sınıf 0'ı tanıdığını gösterir. F1-Skoru: Eğitim setindeki F1-skoru %90 civarındadır. Bu, modelin hem doğruluk hem de duyarlılık açısından dengeli performans sergilediğini gösterir. Genel Değerlendirme: Modelinizin çapraz doğrulama sonuçları, sınıf 0 (çoğunluk sınıfı) için oldukça tutarlı ve yüksek performans sergilediğini gösteriyor. Model, sınıf 0'ı doğru bir şekilde tanımda ve tahmin etmede oldukça başarılıdır. Bu sonuçlar, modelin hem eğitim hem de test veri setlerinde benzer performans gösterdiğini ve aşırı öğrenme (overfitting) sorunu yaşamadığını ortaya koyuyor.

```

df_scores.mean()[2:]

```

| | |
|-------------------|----------------|
| test_precision_0 | 0.85 |
| train_precision_0 | 0.85 |
| test_recall_0 | 0.96 |
| train_recall_0 | 0.96 |
| test_f1_0 | 0.90 |
| train_f1_0 | 0.90 |
| dtype: | float64 |

Doğruluk (Precision): Modelinizin sınıf 0 için hem test hem de eğitim setlerinde ortalama %85 doğruluk oranı var. Bu, modelin her iki sette de aynı derecede başarılı bir şekilde sınıf 0'ı tahmin ettiğini gösterir.

Duyarlılık (Recall): Sınıf 0 için test ve eğitim setlerinde ortalama %96 duyarlılık oranı gözlemlenmiştir. Bu, modelin her iki sette de sınıf 0'ı yakalamada yüksek bir başarı gösterdiğini, yani sınıf 0'ın gerçek örneklerinin büyük bir kısmını doğru bir şekilde tespit ettiğini gösterir.

F1-Skoru: Sınıf 0 için test ve eğitim setlerinde ortalama %90 F1-skoru bulunur. F1-skoru, doğruluk ve duyarlılığı dengede tutarak modelin genel performansını ölçer. Bu yüksek değer, modelin sınıf 0 üzerinde hem iyi bir doğruluk hem de yüksek bir duyarlılık sağladığını belirtir.

Özet:

Modelin eğitim ve test setleri arasındaki metrik değerleri neredeyse eşit, bu da modelin eğitim verisine aşırı uyum sağlamadığını ve genel olarak iyi bir genelleme kabiliyetine sahip olduğunu gösterir. Kısacası, modelin overfitting (aşırı öğrenme) sorunu yaşamadığı ve hem eğitim hem de test setlerinde tutarlı performans sergilediği söylenebilir.

```
eval_metric(pipeline, X_train, y_train, X_test, y_test)
```

Test_Set

| | | precision | recall | f1-score | support |
|--------------|------|-----------|--------|----------|---------|
| 0 | 0.86 | 0.96 | 0.91 | 1500 | |
| 1 | 0.51 | 0.20 | 0.29 | 299 | |
| accuracy | | | | 0.84 | 1799 |
| macro avg | | 0.69 | 0.58 | 0.60 | 1799 |
| weighted avg | | 0.80 | 0.84 | 0.80 | 1799 |

Train_Set

| | | precision | recall | f1-score | support |
|--------------|------|-----------|--------|----------|---------|
| 0 | 0.85 | 0.96 | 0.90 | 7649 | |
| 1 | 0.47 | 0.18 | 0.26 | 1523 | |
| accuracy | | | | 0.83 | 9172 |
| macro avg | | 0.66 | 0.57 | 0.58 | 9172 |
| weighted avg | | 0.79 | 0.83 | 0.80 | 9172 |

Doğruluk (Precision): Test setinde sınıf 0 için %86 ve sınıf 1 için %51 doğruluk oranı görülür. Eğitim setinde ise sınıf 0 için %85 ve sınıf 1 için %47 doğruluk oranı bulunur. Bu sonuçlar, modelin her iki sette de sınıf 0 için yüksek doğruluk sağladığını, ancak sınıf 1 için doğruluğun daha düşük olduğunu gösterir.

Duyarlılık (Recall): Test setinde sınıf 0 için %96, sınıf 1 için %20 duyarlılık sağlanırken, eğitim setinde sınıf 0 için %96, sınıf 1 için %18 duyarlılık sağlanmıştır. Bu, modelin sınıf 0'ı yüksek oranda doğru tahmin ettiğini, ancak sınıf 1'i doğru tahmin etmede zorlandığını gösterir.

F1-Skoru: Test setinde sınıf 0 için %91 ve sınıf 1 için %29, eğitim setinde ise sınıf 0 için %90 ve sınıf 1 için %26 F1-skoru bulunur. F1-skoru, doğruluk ve duyarlılığın bir arada değerlendirildiği bir metriktir. Sınıf 0 için yüksek, sınıf 1 için ise düşük F1-skorları, modelin sınıf 1'i iyi tahmin edemediğini gösterir.

Genel Doğruluk: Test setinde %84, eğitim setinde %83 doğruluk oranı sağlanmıştır. Bu oranlar, modelin genel olarak doğru tahminlerde bulunduğu, ancak bu yüksek doğruluğun sınıf dengesizliklerinden kaynaklanabileceğini ve sınıf 1 için düşük performans sergilediğini belirtir.

Sonuç: Modeliniz, sınıf 0'ı iyi tahmin etme yeteneğine sahipken, sınıf 1'deki düşük duyarlılık ve F1-skorları, modelin bu azınlık sınıfı doğru tahmin etme konusunda zorlandığını gösterir. Bu durum, dengesiz veri setlerinde sık karşılaşılan bir sorundur ve modelinizin sınıf dengesizlikleriyle başa çıkabilmesi için iyileştirme yöntemleri düşünülmelidir.

1.2.4. GridSearchCV

```
('cat_onehot', OneHotEncoder(),
 ['departments']),
('cat_ordinal',
 OrdinalEncoder(categories=[['low',
'medium',
'high']]),
 ['salary'])),
'classifier': LogisticRegression(),
'preprocessor_n_jobs': None,
'preprocessor_remainder': 'drop',
'preprocessor_sparse_threshold': 0.3,
'preprocessor_transformer_weights': None,
'preprocessor_transformers': [(
('num',
 StandardScaler(),
 ['satisfaction_level',
 'last_evaluation',
 'number_project',
 'average_monthly_hours',
 'time_spend_company',
 'work_accident',
 'promotion_last_5years']),
('cat_onehot', OneHotEncoder(), ['departments']),
('cat_ordinal',
 OrdinalEncoder(categories=[['low', 'medium', 'high']]),
 ['salary'])),
'preprocessor_verbose': False,
'preprocessor_verbose_feature_names_out': True,
'preprocessor_num': StandardScaler(),
'preprocessor_cat_onehot': OneHotEncoder(),
'preprocessor_cat_ordinal': OrdinalEncoder(categories=[['low',
'medium', 'high']]),
'preprocessor_num_copy': True,
'preprocessor_num_with_mean': True,
'preprocessor_num_with_std': True,
'preprocessor_cat_onehot_categories': 'auto',
'preprocessor_cat_onehot_drop': None,
'preprocessor_cat_onehot_dtype': numpy.float64,
'preprocessor_cat_onehot_feature_name_combiner': 'concat',
'preprocessor_cat_onehot_handle_unknown': 'error',
'preprocessor_cat_onehot_max_categories': None,
'preprocessor_cat_onehot_min_frequency': None,
'preprocessor_cat_onehot_sparse': 'deprecated',
'preprocessor_cat_onehot_sparse_output': True,
'preprocessor_cat_ordinal_categories': [['low', 'medium', 'high']],
'preprocessor_cat_ordinal_dtype': numpy.float64,
'preprocessor_cat_ordinal_encoded_missing_value': nan,
'preprocessor_cat_ordinal_handle_unknown': 'error',
'preprocessor_cat_ordinal_max_categories': None,
```

```
'preprocessor_cat_ordinal_min_frequency': None,
'preprocessor_cat_ordinal_unknown_value': None,
'classifier_C': 1.0,
'classifier_class_weight': None,
'classifier_dual': False,
'classifier_fit_intercept': True,
'classifier_intercept_scaling': 1,
'classifier_l1_ratio': None,
'classifier_max_iter': 100,
'classifier_multi_class': 'auto',
'classifier_n_jobs': None,
'classifier_penalty': 'l2',
'classifier_random_state': None,
'classifier_solver': 'lbfgs',
'classifier_tol': 0.0001,
'classifier_verbose': 0,
'classifier_warm_start': False}
```

GridSearchCV kullanırken, pipeline modelini kullanıyorsak, `get_params()` komutu ile pipeline içerisinde tanımlı makine öğrenme algoritmasının hiperparametrelerinin listesini görebiliriz. Bu hiperparametreleri, GridSearchCV'nin `param_grid` parametresine belirtmeliyiz. `pipeline.get_params()` komutu, bir model pipeline'ının tüm mevcut hiperparametrelerini ve bu parametrelerin mevcut değerlerini döndürür. Bu komut, GridSearchCV gibi hiperparametre optimizasyon yöntemleri kullanırken oldukça faydalıdır. Özellikle, pipeline içerisinde tanımlanan tüm adımları ve her bir adımın konfigürasyonunu görenizi sağlar. Örneğin, burada `ColumnTransformer`, `StandardScaler`, `OneHotEncoder`, `OrdinalEncoder` ve `LogisticRegression` gibi adımların parametrelerini ve ayarlarını görebilirsiniz. Bu bilgi, hangi parametrelerin optimize edilebileceği ve hangi değerlerin ayarlanabileceğini konusunda size net bir bakış açısı sağlar.

Örneğin, ColumnTransformer içindeki transformers ve preprocessor adımlarının ayrıntılarını, LogisticRegression modelinin hiperparametrelerini (C, penalty, solver, vb.) görebilirsiniz. Bu, modelin eğitimi sırasında hangi ayarların kullanıldığını ve hangi hiperparametrelerin optimize edilebileceğini anlamanıza yardımcı olur. Bu bilgiler, GridSearchCV gibi araçlarla, modelinizin performansını artırmak için hangi hiperparametrelerin test edileceğini ve en iyi değerlerin ne olabileceğini belirlemenizi sağlar.

```

        'work_accident',
'promotion_last_5years']),
('cat_onehot',
OneHotEncoder(),
['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[['low',
'medium',
'high']])),
(['salary'))),
('classifier', LogisticRegression(C=1)))

```

`set_params()` fonksiyonunu kullanarak, pipeline modelindeki herhangi bir hiperparametreyi değiştirebiliriz. Bu fonksiyon sayesinde, modelin belirli bir bileşeninin hiperparametrelerini güncelleyebiliriz.

`set_params()` fonksiyonuna, değiştirmek istediğimiz hiperparametrenin adını ve yeni değerini belirterek, bu hiperparametreyi modelle senkronize edebiliriz. Bu örnekte, `classifier` adıyla tanımlı olan `LogisticRegression` modelinin C hiperparametresini 1 olarak ayarlıyoruz. `set_params()` fonksiyonu, pipeline içindeki her bir bileşene yönelik hiperparametreleri güncellememizi sağlar ve böylece modelin performansını optimize edebiliriz.

```

pipeline.get_params()

{'memory': None,
 'steps': [(
    'preprocessor',
    ColumnTransformer(transformers=[(
        'num', StandardScaler(),
        ['satisfaction_level',
         'last_evaluation',
         'number_project'],
        'average_monthly_hours',
        'time_spend_company'],
        'work_accident',
        ['promotion_last_5years']),
        ('cat_onehot', OneHotEncoder(),
         ['departments']),
        ('cat_ordinal',
         OrdinalEncoder(categories=[['low',
          'medium',
          'high']])),
        ['salary'))]),
    ('classifier', LogisticRegression(C=1))],
 'verbose': False,
 'preprocessor': ColumnTransformer(transformers=[(
        'num',
        StandardScaler(),

```

```

                    ['satisfaction_level',
'last_evaluation',                                'number_project',
'average_monthly_hours',                         'time_spend_company',
'work_accident',                                 'promotion_last_5years']),
('cat_onehot', OneHotEncoder(),
 ['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[['low',
'medium',
'high']]),
 ['salary'))),
'classifier': LogisticRegression(C=1),
'preprocessor_n_jobs': None,
'preprocessor_remainder': 'drop',
'preprocessor_sparse_threshold': 0.3,
'preprocessor_transformer_weights': None,
'preprocessor_transformers': [('num',
StandardScaler(),
['satisfaction_level',
'last_evaluation',
'number_project',
'average_monthly_hours',
'time_spend_company',
'work_accident',
'promotion_last_5years']),
('cat_onehot', OneHotEncoder(), ['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[['low', 'medium', 'high']]),
['salary')),
'preprocessor_verbose': False,
'preprocessor_verbose_feature_names_out': True,
'preprocessor_num': StandardScaler(),
'preprocessor_cat_onehot': OneHotEncoder(),
'preprocessor_cat_ordinal': OrdinalEncoder(categories=[['low',
'medium', 'high']]),
'preprocessor_num_copy': True,
'preprocessor_num_with_mean': True,
'preprocessor_num_with_std': True,
'preprocessor_cat_onehot_categories': 'auto',
'preprocessor_cat_onehot_drop': None,
'preprocessor_cat_onehot_dtype': numpy.float64,
'preprocessor_cat_onehot_feature_name_combiner': 'concat',
'preprocessor_cat_onehot_handle_unknown': 'error',
'preprocessor_cat_onehot_max_categories': None,
'preprocessor_cat_onehot_min_frequency': None,

```

```

'preprocessor__cat_onehot__sparse': 'deprecated',
'preprocessor__cat_onehot__sparse_output': True,
'preprocessor__cat_ordinal__categories': [['low', 'medium', 'high']],
'preprocessor__cat_ordinal__dtype': numpy.float64,
'preprocessor__cat_ordinal__encoded_missing_value': nan,
'preprocessor__cat_ordinal__handle_unknown': 'error',
'preprocessor__cat_ordinal__max_categories': None,
'preprocessor__cat_ordinal__min_frequency': None,
'preprocessor__cat_ordinal__unknown_value': None,
'classifier__C': 1,
'classifier__class_weight': None,
'classifier__dual': False,
'classifier__fit_intercept': True,
'classifier__intercept_scaling': 1,
'classifier__l1_ratio': None,
'classifier__max_iter': 100,
'classifier__multi_class': 'auto',
'classifier__n_jobs': None,
'classifier__penalty': 'l2',
'classifier__random_state': None,
'classifier__solver': 'lbfgs',
'classifier__tol': 0.0001,
'classifier__verbose': 0,
'classifier__warm_start': False}

```

`pipeline.get_params()` fonksiyonu, pipeline içindeki tüm hiperparametrelerin ve ayarların bir listesini döndürür. Bu çıktı, pipeline'in şu anda nasıl yapılandırıldığını gösterir.

ColumnTransformer, veri ön işleme aşamasında kullanılan StandardScaler, OneHotEncoder ve OrdinalEncoder gibi işlemleri içerir. Model aşamasında ise LogisticRegression kullanılır ve C hiperparametresi 1 olarak ayarlanmıştır, bu da düzenleme (regularization) gücünü orta seviyede tutar. Ayrıca, çeşitli hiperparametreler için varsayılan değerler verilmiştir, örneğin penalty türü 'l2', solver ise 'lbfgs' olarak belirlenmiştir. Genel olarak, bu bilgiler pipeline'in her bileşeninin ve modelin mevcut yapılandırmasını detaylı bir şekilde sunar.

```

# Define the parameter grid
penalty = ["l1", "l2"]
C = [0.01, 0.1, 1, 5]
class_weight = ["balanced", None]
solver = ["lbfgs", "liblinear", "sag", "saga"]

# Create the parameter grid with compatible solvers and penalties
param_grid = [
    {
        "classifier__penalty": ["l2"], # lbfgs and sag support only l2
        "classifier__C": C,
        "classifier__class_weight": class_weight,
        "classifier__solver": ["lbfgs", "sag"]
    },
]

```

```

        {
            "classifier_penalty": ["l1", "l2"], # liblinear and saga
            support both l1 and l2
            "classifier_C": C,
            "classifier_class_weight": class_weight,
            "classifier_solver": ["liblinear", "saga"]
        }
    ]

# Initialize GridSearchCV with the pipeline
grid_model = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=10,                      # 10-fold cross-validation
    scoring="recall",             # Use recall as the scoring metric
    n_jobs=-1,                   # Utilize all available cores
    return_train_score=True      # Return training scores along with
validation scores
)

```

Bu kod parçası, GridSearchCV kullanarak bir modelin hiperparametrelerini optimize etmek için bir parametre ızgarası tanımlar. İlk olarak, penalty, C, class_weight ve solver gibi hiperparametreler için bir dizi olası değer belirlenir. param_grid adlı liste, bu hiperparametrelerin uyumlu kombinasyonlarını içerir: l2 cezasını destekleyen lbfsgs ve sag çözücüleri ile, l1 ve l2 cezasını destekleyen liblinear ve saga çözücüleri için ayrı kombinasyonlar sunulur. Daha sonra, GridSearchCV nesnesi oluşturulur ve pipeline, bu parametre ızgarası ile optimize edilir. 10 katlı çapraz doğrulama yapılacak ve recall metriği kullanılacak, ayrıca tüm çekirdekler kullanılacak ve hem eğitim hem de doğrulama skorları döndürülecektir.

```

# Fit the grid search model with training data (X_train and y_train
# need to be defined)
grid_model.fit(X_train, y_train)

GridSearchCV(cv=10,
            estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('num',
StandardScaler(),
['satisfaction_level',
'last_evaluation',
'number_project',
'average_monthly_hours',
'time_spend_company',

```

```

'work_accident',
'promotion_last_5years']),
('cat_onehot',
OneHotEncoder(),
['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[...,
param_grid=[{'classifier_C': [0.01, 0.1, 1, 5],
'classifier_class_weight': ['balanced',
None], 'classifier_penalty': ['l2'],
'classifier_solver': ['lbfgs', 'sag']},
{'classifier_C': [0.01, 0.1, 1, 5],
'classifier_class_weight': ['balanced',
None], 'classifier_penalty': ['l1', 'l2'],
'classifier_solver': ['liblinear',
'saga']}]],
return_train_score=True, scoring='recall')

```

Bu kod satırı, daha önce tanımlanan grid_model nesnesini kullanarak hiperparametre optimizasyonu gerçekleştiren GridSearchCV modelini eğitim verileri (X_train ve y_train) üzerinde çalıştırır. İşlem şu adımları içerir:

Parametre Arama: grid_model, param_grid içinde belirtilen tüm hiperparametre kombinasyonlarını deneyecektir. Bu süreç, belirli bir hiperparametre kombinasyonu için modelin performansını ölçmeyi içerir.

Çapraz Doğrulama: Her bir hiperparametre kombinasyonu, 10 katlı çapraz doğrulama ile değerlendirilir. Bu, modelin her kombinasyon için eğitim ve doğrulama veri setlerinde test edilmesi anlamına gelir.

Performans Ölçümü: scoring="recall" olarak belirlendiği için, her kombinasyonun recall metriği kullanılarak performansı ölçülür. Bu metrik, modelin pozitif sınıfları ne kadar iyi tanımladığını değerlendirdir.

En İyi Model Seçimi: GridSearchCV, her hiperparametre kombinasyonunun doğrulama performansını karşılaştırarak en iyi performansa sahip olanı seçer.

Modeli Eğitme: En iyi hiperparametre kombinasyonu seçildikten sonra, grid_model bu kombinasyonu kullanarak tüm eğitim verileri üzerinde modelin yeniden eğitimini gerçekleştirir.

Sonuç olarak, bu işlem, modelin hiperparametrelerini optimize ederek en iyi performansı sağlayan ayarları bulmayı amaçlar ve bu ayarlarla eğitilmiş en iyi modeli elde eder.

```

grid_model.best_estimator_

# best_estimator_ does not return default hyper_parameters, only those
that
# differ from default values.

Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                 StandardScaler(),
                                                 [
                                                 'satisfaction_level',
                                                 'last_evaluation',
                                                 'number_project',
                                                 'average_monthly_hours',
                                                 'work_accident',
                                                 'time_spend_company',
                                                 ('cat_onehot',
                                                 OneHotEncoder(),
                                                 [
                                                 'departments']),
                                                 ('cat_ordinal',
                                                 OrdinalEncoder(categories=[[ 'low',
                                                 'medium',
                                                 'high']])),
                                                 [ 'salary'])])),
                 ('classifier',
                  LogisticRegression(C=0.01, class_weight='balanced',
                                      penalty='l1',
                                      solver='liblinear'))])

```

grid_model.best_estimator_ özelliği, GridSearchCV kullanılarak yapılan hiperparametre araması sonucunda en iyi performansı gösteren modelin kendisini döndürür. Bu model, hiperparametrelerinin GridSearchCV sürecinde belirlenen en iyi kombinasyonları ile yapılandırılmıştır.

Önemli Noktalar:

En İyi Model: best_estimator_, çapraz doğrulama sırasında en iyi performansı gösteren modelin tam bir örneğidir. Bu model, parametre grid'inde denenen en iyi hiperparametrelerle eğitilmiştir.

Hiperparametrelerin Gösterimi: best_estimator_ çıktısı, yalnızca hiperparametreleri değiştirilmiş olan değerleri gösterir. Varsayılan değerlerdeki hiperparametreler bu çıktı içinde yer almaz. Yani, modelin varsayılan hiperparametreleri değiştirilmiş olanları dışında, diğer tüm hiperparametreler varsayılan değerlerinde kalır.

Varsayılan Değerlerin Gizliliği: Eğer bir hiperparametre, GridSearchCV sırasında farklı bir değerle ayarlandıysa, bu değerler best_estimator_ çıktısında açıkça belirtilir. Ancak, eğer bir hiperparametre varsayılan değerde kalmışsa, bu durumda o hiperparametre best_estimator_ çıktısında görünmez.

Özetle, grid_model.best_estimator_, modelin varsayılan hiperparametrelerden farklı olan ayarlarını içerir ve bu, modelin hangi hiperparametre kombinasyonuyla en iyi sonuçları verdiği gösterir. Varsayılan değerlere sahip hiperparametreler bu çıktıda gösterilmmez.

```
pd.DataFrame(grid_model.cv_results_).loc[grid_model.best_index_,  
["mean_test_score", "mean_train_score"]]  
  
mean_test_score    0.82  
mean_train_score   0.83  
Name: 16, dtype: object
```

pd.DataFrame(grid_model.cv_results_).loc[grid_model.best_index_, ["mean_test_score", "mean_train_score"]] komutu, GridSearchCV işlemi sırasında elde edilen en iyi modelin test ve eğitim setlerindeki ortalama skorlarını verir.

Bu skorlar, GridSearchCV'nin hiperparametre araması sırasında en iyi sonuçları veren model için hesaplanan değerlendirme metrikleridir. Burada verilen mean_test_score ve mean_train_score değerleri, en iyi modelin test ve eğitim verilerindeki performansını gösterir.

Özetle:

mean_test_score (0.82): En iyi modelin test setindeki ortalama skoru, GridSearchCV sırasında kullanılan değerlendirme metriğine göre 0.82'dir. mean_train_score (0.83): Aynı modelin eğitim setindeki ortalama skoru ise 0.83'tür. Bu skorlar, modelin overfitting (aşırı uyum) veya underfitting (yetersiz uyum) durumunu değerlendirmeye yardımcı olabilir. Genellikle, test skoru eğitim skorundan belirgin şekilde düşükse overfitting belirtisi olabilir. Ancak burada, test ve eğitim skorları birbirine oldukça yakın, bu da modelin overfitting veya underfitting sorunlarının olmadığını gösterebilir. GridSearchCV zaten çapraz doğrulama içerdiginden, ekstra bir çapraz doğrulama yapmaya gerek kalmaz.

Steps Taken:

- Accessing Results:** You are using pd.DataFrame(grid_model.cv_results_) to access the cross-validation results from the GridSearchCV object.
- Extracting Scores:** You are retrieving the mean_test_score and mean_train_score for the best model found during the grid search using loc[grid_model.best_index_, ["mean_test_score", "mean_train_score"]].
- Evaluation:** You are comparing these scores to evaluate overfitting and underfitting.

Interpretation:

- mean_test_score:** This is the average score (recall, in your case) obtained from the cross-validation folds on the test sets. It reflects the model's performance on unseen data.

- **mean_train_score**: This is the average score obtained on the training data across the cross-validation folds. It indicates how well the model performs on the training set.

What the Scores Indicate:

- **Scores: mean_test_score = 0.83 and mean_train_score = 0.83**

Since both `mean_test_score` and `mean_train_score` are equal and relatively high, this suggests that:

- The model performs well on both training and testing data.
- There is no significant overfitting or underfitting, as the training and testing scores are quite close.

Additional Points:

- **Overfitting**: If the `mean_train_score` is significantly higher than the `mean_test_score`, it indicates overfitting. In your case, since both scores are equal, overfitting is not a concern.
- **Underfitting**: If both scores are low, it could indicate underfitting. In this case, since both scores are high, underfitting is not a concern.
- **Cross-Validation**: As you mentioned, since `GridSearchCV` includes cross-validation, you are already assessing the model's performance across different subsets of the data, which helps to ensure that your performance metrics are reliable and not due to random chance or specific data splits.

Overall, the results suggest that your model is well-tuned and performs consistently across training and validation data.

`GridSearchCV` kullanarak elde edilen `mean_test_score` ve `mean_train_score` değerleri, modelin hem test hem de eğitim setlerinde nasıl performans gösterdiğini yansıtır. Bu skorlar, modelin eğitim verileri üzerinde ne kadar iyi performans gösterdiğini ve çapraz doğrulama katmanlarındaki test verileri üzerindeki başarısını gösterir. Eğer her iki skor da birbirine yakın ve yüksekse, bu durum modelin hem eğitim hem de test verilerinde iyi performans gösterdiğini, yani önemli bir overfitting veya underfitting sorunu yaşamadığını gösterir. Bu durumda, modelin hem eğitim hem de test setlerinde dengeli bir performans sergilediği ve `GridSearchCV`'nin çapraz doğrulama işleminin güvenilir sonuçlar sağladığı anlaşılır.

```
eval_metric(grid_model, X_train, y_train, X_test, y_test)
```

Test_Set

```
[[1141  359]
 [ 43  256]]
```

| | precision | recall | f1-score | support |
|----------|-----------|--------|----------|---------|
| 0 | 0.96 | 0.76 | 0.85 | 1500 |
| 1 | 0.42 | 0.86 | 0.56 | 299 |
| accuracy | | | 0.78 | 1799 |

| | | | | |
|--------------|------|------|------|------|
| macro avg | 0.69 | 0.81 | 0.71 | 1799 |
| weighted avg | 0.87 | 0.78 | 0.80 | 1799 |

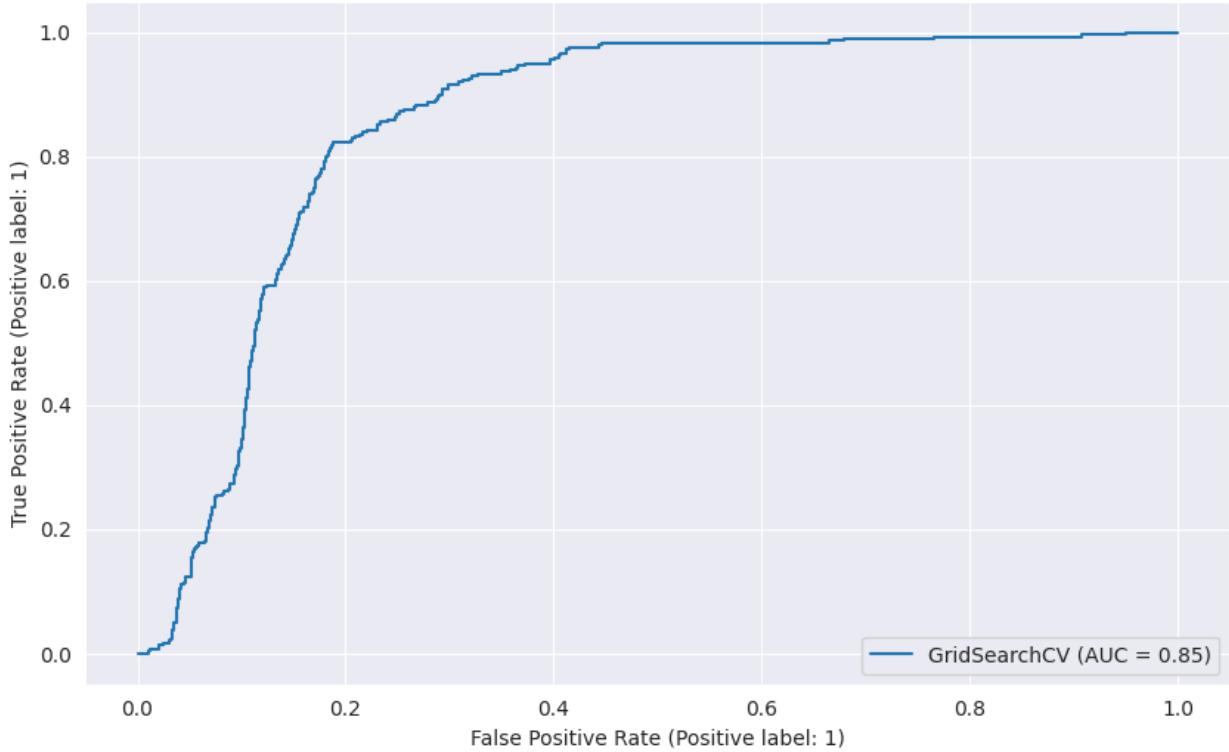
```
Train_Set
[[5698 1951]
 [ 267 1256]]
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.96 | 0.74 | 0.84 | 7649 |
| 1 | 0.39 | 0.82 | 0.53 | 1523 |
| accuracy | | | 0.76 | 9172 |
| macro avg | 0.67 | 0.78 | 0.68 | 9172 |
| weighted avg | 0.86 | 0.76 | 0.79 | 9172 |

GridSearchCV ile optimize edilen modelin, test ve eğitim setlerinde hesaplanan recall skorları, GridSearchCV'nin belirlediği recall ile tutarlıdır. Test setinde modelin recall skoru %86, eğitim setinde ise %82'dir, bu da modelin her iki veri kümesinde de azınlık sınıfı doğru şekilde tanımlama yeteneğini yansıtır. Test setinde %76 doğruluk ve %78 f1-skora sahipken, eğitim setinde %76 doğruluk ve %79 f1-skora sahiptir. Test seti ve eğitim seti sonuçları arasındaki bu tutarlılık, modelin ne aşırı uyum (overfitting) ne de yetersiz uyum (underfitting) yaşamadığını, ve GridSearchCV'nin model seçiminde etkili olduğunu gösterir. Modelin sınıf dengesizliğine rağmen azınlık sınıfı üzerinde iyi bir recall performansı sergilediği görülmektedir.

1.2.5. ROC (Receiver Operating Curve) and AUC (Area Under Curve)

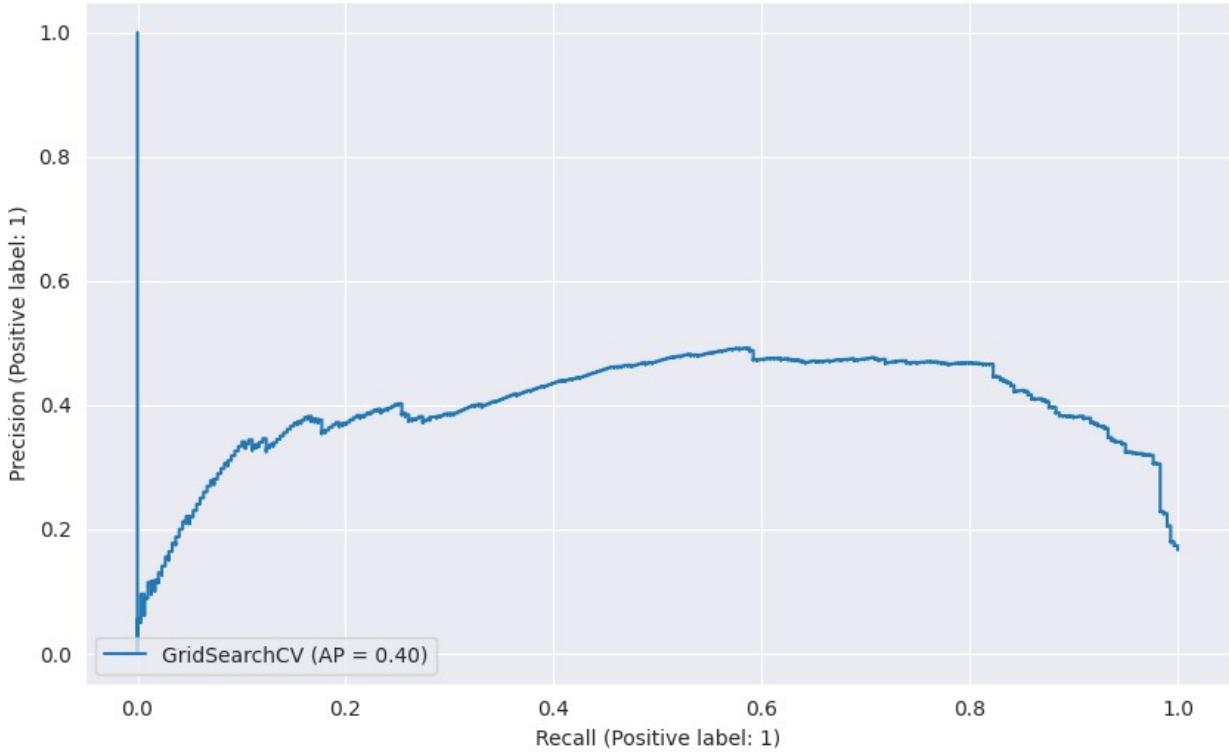
```
from sklearn.metrics import RocCurveDisplay, PrecisionRecallDisplay,
roc_auc_score, roc_curve,\n                           average_precision_score,
precision_recall_curve\n\nRocCurveDisplay.from_estimator(grid_model, X_test, y_test);
```



`RocCurveDisplay.from_estimator(grid_model, X_test, y_test)` komutu, `GridSearchCV` ile optimize edilen modelin test setindeki ROC eğrisini doğrudan gösterir, bu da modelin çeşitli eşik değerlerinde doğru pozitif oranını (True Positive Rate) ve yanlış pozitif oranını (False Positive Rate) gösterir. Alternatif olarak, `y_test_pred_proba = grid_model.predict_proba(X_test)[:,1]` kodu ile modelin test setindeki pozitif sınıf için tahmin edilen olasılıkları elde edebilir ve `RocCurveDisplay.from_predictions(y_test, y_test_pred_proba, pos_label=1)` komutu ile bu tahminler üzerinden ROC eğrisini çizdirerek modelin performansını değerlendirebilirsiniz. Bu yöntemler, modelin sınıflandırma performansını ve karar sınırlarının ne kadar etkili olduğunu değerlendirmek için kullanılır.

1.2.5.2. Precision Recall Curve

```
PrecisionRecallDisplay.from_estimator(grid_model, X_test, y_test);
```



PrecisionRecallDisplay.from_estimator(grid_model, X_test, y_test) komutu, GridSearchCV ile optimize edilen modelin test setindeki Precision-Recall eğrisini doğrudan gösterir. Bu eğri, modelin pozitif sınıfları tanıma yeteneğini ve pozitif sınıflar için tahminlerin doğruluğunu gösterir, özellikle veri setinde dengesizlik olduğunda önemli bir değerlendirme aracıdır. Alternatif olarak, PrecisionRecallDisplay.from_predictions(y_test, y_test_pred_proba) kullanarak modelin pozitif sınıf için tahmin edilen olasılıkları ile Precision-Recall eğrisini çizdirebilirsiniz. Veri setinizin dengesizliği nedeniyle, average precision score (ortalama hassasiyet skoru) bu tür bir analizde öne çıkar çünkü bu metrik, modelin tüm olasılık eşiklerinde nasıl performans gösterdiğini özetler.

1.2.6. Finding Best Thresholds

```
# Define parameter grid with compatible solvers and penalties
param_grid = [
    {
        'classifier_penalty': ['l2'], # lbfsgs and sag support only l2
        'classifier_C': [0.0001, 0.01, 0.1, 5],
        'classifier_class_weight': ['balanced', None],
        'classifier_solver': ['lbfgs', 'sag']
    },
    {
        'classifier_penalty': ['l1', 'l2'], # liblinear and saga support both l1 and l2
        'classifier_C': [0.0001, 0.01, 0.1, 5],
        'classifier_class_weight': ['balanced', None],
    }
]
```

```

        'classifier__solver': ['liblinear', 'saga']
    }
]

# Initialize GridSearchCV with the pipeline and parameter grid
grid_model2 = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=10, # 10-fold cross-validation
    scoring="recall", # Use recall as the scoring metric
    n_jobs=-1, # Utilize all available cores
    return_train_score=True # Return training scores along with
validation scores
)

```

Bu kodda, GridSearchCV kullanarak modelin hiperparametrelerini optimize etmek için bir parametre izgarası tanımlanmıştır. param_grid değişkeninde, iki farklı parametre kombinasyonu belirtilmiştir: İlkı sadece 'l2' ceza terimini destekleyen lbfgs ve sag çözüçülerle uyumlu parametreleri içerirken, ikincisi 'l1' ve 'l2' ceza terimlerini destekleyen liblinear ve saga çözüçülerle uyumludur. Bu parametrelerle modelin farklı kombinasyonlarını test etmek için GridSearchCV nesnesi başlatılır; bu nesne, 10 katlı çapraz doğrulama kullanarak modelin performansını değerlendirecek ve recall metriğini skor olarak belirleyecektir. Ayrıca, n_jobs=-1 parametresi, tüm işlemci çekirdeklerinin kullanılmasını sağlar ve return_train_score=True seçeneği ile hem eğitim hem de test skorları döndürür.

```

# Fit the grid search model with training data
grid_model2.fit(X_train, y_train)

GridSearchCV(cv=10,
             estimator=Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('num',
StandardScaler(),
['satisfaction_level',
'last_evaluation',
'number_project',
'average_monthly_hours',
'time_spend_company',
'work_accident',
'promotion_last_5years')]),

```

```

('cat_onehot',
OneHotEncoder(),
['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[...
    param_grid=[{'classifier_C': [0.0001, 0.01, 0.1, 5],
                 'classifier_class_weight': ['balanced'],
None],
                 'classifier_penalty': ['l2'],
                 'classifier_solver': ['lbfgs', 'sag']},
{'classifier_C': [0.0001, 0.01, 0.1, 5],
                 'classifier_class_weight': ['balanced'],
None],
                 'classifier_penalty': ['l1', 'l2'],
                 'classifier_solver': ['liblinear',
'saga']}]],
return_train_score=True, scoring='recall')

```

Bu kod satırı, grid_model2 adlı GridSearchCV nesnesini, X_train ve y_train eğitim verileri ile fit eder. Bu işlem, belirlenen parametre izgarası üzerindeki tüm hiperparametre kombinasyonlarını kullanarak modelin performansını değerlendirdir ve en iyi performansı gösteren parametreleri bulur. GridSearchCV, 10 katlı çapraz doğrulama ile modelin farklı hiperparametre kombinasyonlarını test eder ve recall metriğine göre en uygun modeli seçer.

```

grid_model2.best_estimator_
Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('num',
StandardScaler(),
['satisfaction_level',
                           'last_evaluation',
                           'number_project',
'average_monthly_hours',
                           'work_accident',
'time_spend_company',
                           'promotion_last_5years']),
('cat_onehot',
OneHotEncoder(),
['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[['low',

```

```
'medium',
'high']]),
('classifier',
LogisticRegression(C=0.0001, class_weight='balanced',
solver='liblinear'))))
```

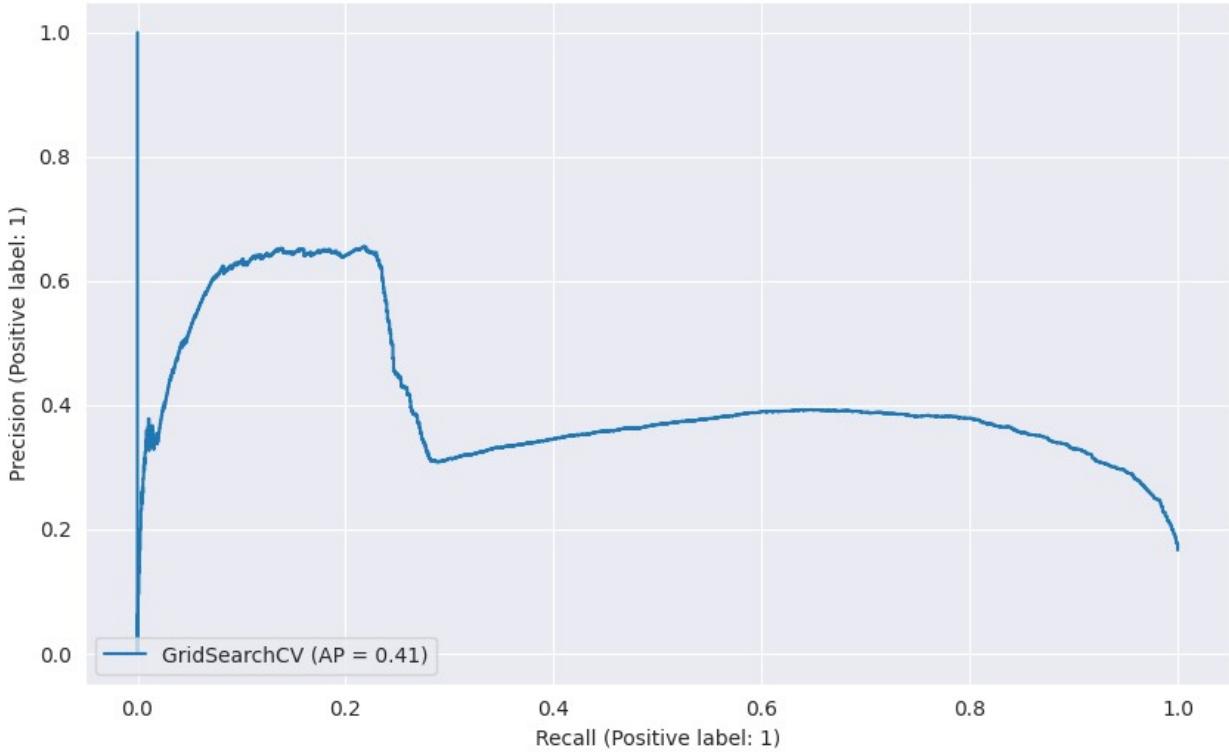
grid_model2.best_estimator_, GridSearchCV işlemi tamamlandıktan sonra en iyi performansı gösteren modelin tam yapılandırmasını döndürür. Bu, parametre izgarasında test edilen tüm hiperparametre kombinasyonları arasından, recall metriğine göre en yüksek skoru elde eden modelin parametrelerini içerir. Bu model, hem en iyi hiperparametreleri hem de en iyi performansı yansıtır.

```
pd.DataFrame(grid_model2.cv_results_).loc[grid_model2.best_index_,
["mean_test_score", "mean_train_score"]]

mean_test_score    0.85
mean_train_score   0.85
Name: 18, dtype: object
```

pd.DataFrame(grid_model2.cv_results_).loc[grid_model2.best_index_, ["mean_test_score", "mean_train_score"]]] komutu, GridSearchCV sonucunda elde edilen en iyi modelin test ve eğitim veri setleri üzerindeki ortalama skorlarını gösterir. Bu skorlar, modelin hem test hem de eğitim veri setlerinde nasıl performans gösterdiğini yansıtır. Burada, mean_test_score ve mean_train_score değerlerinin her ikisi de 0.85 olduğu için, modelin eğitim ve test veri setlerinde benzer performans gösterdiği ve bu durumun aşırı uyum (overfitting) veya yetersiz uyum (underfitting) gibi sorunların olmadığını gösterir.

```
from sklearn.metrics import PrecisionRecallDisplay
PrecisionRecallDisplay.from_estimator(grid_model2, X_train, y_train);
```



PrecisionRecallDisplay.from_estimator(grid_model2, X_train, y_train) komutu, GridSearchCV ile elde edilen en iyi modelin eğitim verileri üzerinde precision (doğruluk) ve recall (duyarlılık) değerlerinin eşik değerlerine göre nasıl değiştiğini gösterir. Bu görselleştirme, modelin çeşitli eşik değerlerinde ne kadar iyi performans gösterdiğini anlamamıza yardımcı olur. Ayrıca, veri sızıntısını önlemek için en iyi eşik değerini eğitim verileri üzerinde belirlemek önemlidir, çünkü bu sayede modelin gerçek performansı daha doğru bir şekilde değerlendirilir.

```
y_train_pred_proba = grid_model2.predict_proba(X_train)
# We determine the threshold over the train data.
average_precision_score(y_train, y_train_pred_proba[:,1])
0.4096388956166767
```

y_train_pred_proba = grid_model2.predict_proba(X_train) komutu, eğitim verileri üzerinde en iyi modelin tahmin ettiği olasılıkları hesaplar. Bu olasılıklar, her bir örneğin pozitif sınıf'a ait olma olasılığını içerir. average_precision_score(y_train, y_train_pred_proba[:,1]) komutu ise bu olasılıkları kullanarak eğitim verilerindeki ortalama precision (doğruluk) skorunu hesaplar. Sonuç olarak, 0.4096 değerini elde ettik, bu da modelin eğitim verilerindeki precision (doğruluk) skorunun ortalama olarak yaklaşık %41 olduğunu gösterir. Bu değer, modelin pozitif sınıf için ne kadar iyi tahminde bulunduğu ölçen bir performans metriğidir ve modelin genel performansını değerlendirmede önemli bir rol oynar.

```
precisions, recalls, thresholds = precision_recall_curve(y_train,
y_train_pred_proba[:,1])
```

`precision_recall_curve(y_train, y_train_pred_proba[:,1])` komutu, eğitim verileri (`y_train`) ve modelin bu veriler üzerindeki pozitif sınıf'a ait olma olasılıkları (`y_train_pred_proba[:,1]`) kullanılarak precision (doğruluk), recall (bulma oranı) ve threshold (eşik) değerlerini hesaplar. Bu fonksiyon, çeşitli eşik değerleri için modelin precision ve recall performansını ölçen bir dizi değer döndürür.

```
precision : len(precisions))
recall    : len(recalls))
threshold : len(thresholds))

precision : 9173
recall    : 9173
threshold : 9172
```

`precision_recall_curve` fonksiyonu, çeşitli eşik değerleri için modelin precision (doğruluk) ve recall (bulma oranı) değerlerini döndürür. `print` komutlarıyla bu çıktılarından elde edilen uzunlukları kontrol ettiğinizde, `precisions` ve `recalls` listelerinin 9173 elemandan olduğunu, `thresholds` listesinin ise 9172 elemandan olduğunu görebilirsiniz. Bu fark, eşik değerinin 1 olduğu durumda bir sonuç döndürülmemesiyle ilgilidir. Yani, eşik değeri 1 için model, tüm örnekleri negatif sınıf'a atadığından, eşik değeri 1'de precision ve recall değerleri hesaplanır ama eşik değeri 1'in kendisi `thresholds` listesine dahil edilmez. Bu durum, precision ve recall hesaplamalarının, eşik değerleri 1'e yaklaşırken nasıl değiştiğini anlamak için önemlidir.

```
dict = {"precisions":precisions[:-1], "recalls":recalls[:-1],
 threshold |:thresholds}
df_metric = pd.DataFrame(dict).sort_values(by = ["recalls",
"precisions"], ascending=False)
df_metric

{"summary": {"name": "df_metric", "rows": 9172,
"fields": [{"column": "precisions", "properties": {"dtype": "number", "std": 0.111564919631564, "min": 0.0, "max": 0.655511811023622, "num_unique_values": 9069, "samples": [0.3423885187083547, 0.38454288407163056, 0.39203436157750876]}, "semantic_type": "\\", "description": "\n"}, {"column": "recalls", "properties": {"dtype": "number", "std": 0.29672794180021766, "min": 0.0, "max": 1.0, "num_unique_values": 1524, "samples": [0.4865397242284964, 0.9500984898227183, 0.33749179251477346]}, "semantic_type": "\\", "description": "\n"}, {"column": "threshold", "properties": {"dtype": "number", "std": 0.35247112153433835, "min": 0.4205857902058791, "max": 0.5128149196837299}}, "threshold": 0.4205857902058791, "recalls": 0.5128149196837299, "precisions": 0.35247112153433835}], "semantic_type": "\\", "description": "\n"}}
```

```
0.465803976782539\n      ],\n      \"semantic_type\": \"/\",\\n\n      \"description\": \"\\n      }\\n      }\\n    ]\\n  ]\\n}\",\"type\":\"dataframe\",\"variable_name\":\"df_metric\"}
```

Sonuç olarak oluşturulan DataFrame, her eşik değeri için hesaplanan precision (doğruluk) ve recall (bulma oranı) değerlerini içerir. Ancak, precisions ve recalls listelerinin son değerleri hariç tutulur çünkü bu değerler eşiklerin uygulandığı noktalar arasında yer alır. Sonuçlar, önce recall değerine (azalan sırayla) ve ardından precision değerine (azalan sırayla) göre sıralanır. Bu sıralama, belirli bir recall değeri için en yüksek precision değerlerini bulmayı sağlar.

Örneğin, eğer amacınız en yüksek recall değerine ulaşmaksa, recall değeri 1.0 olan en yüksek precision değeri yaklaşık olarak 0.358047'dir. Ayrıca, belirli bir recall değeri (örneğin, 0.838384) elde etmek istiyorsanız, bu recall değeri için optimal eşik değeri yaklaşık olarak 0.481736'dır. Bu, çeşitli eşik değerlerinde modelin performansını değerlendirmek ve ihtiyaç duyulan recall veya precision seviyelerine ulaşmak için hangi eşiklerin kullanılabileceğini anlamak için önemlidir.

```
df_metric['f1_score'] = 2 * (df_metric['precisions'] * df_metric['recalls']) / (df_metric['precisions'] + df_metric['recalls'])

# En yüksek F1 skoruna sahip threshold'u bul
optimal_row = df_metric.loc[df_metric['f1_score'].idxmax()]
optimal_threshold = optimal_row['threshold']

# Sonuçları göster
optimal_row

precisions    0.38
recalls       0.80
threshold     0.51
f1_score      0.51
Name: 5943, dtype: float64
```

Oluşturulan df_metric DataFrame'ine f1_score sütunu eklenmiştir. Bu sütun, her eşik değeri için precision ve recall değerlerinden hesaplanan F1 skorunu temsil eder. Verilen df_metric DataFrame'inde en yüksek F1 skoruna sahip eşik değeri belirlenmiştir. Bu, en iyi dengeyi sağlayan eşik değeri bulmak için F1 skorunu kullanmanın bir yoludur.

Sonuç olarak, F1 skoru en yüksek olan eşik değeri şu şekilde bulunmuştur:

Eşik Değeri (Threshold): 0.51 Precision: 0.38 Recall: 0.80 F1 Skoru: 0.51 Bu sonuç, verilen eşik değeri için modelin F1 skorunun en yüksek olduğunu ve bu eşik değerinin hem iyi bir precision hem de iyi bir recall sağladığını gösterir. Bu tür analizler, özellikle dengesiz veri kümelerinde model performansını değerlendirmek için önemlidir.

```
optimal_threshold = 0.51
```

optimal_threshold olarak belirlenen 0.51 değeri, modelin tahminlerinin karar sınırını temsil eder. Bu eşik değeri, modelin sınıflandırma kararlarını yaparken kullanılır. Örneğin, bir örneğin pozitif

sınıfa ait olup olmadığına karar verirken, modelin tahmin ettiği olasılık bu eşik değeri ile karşılaşır.

Eşik Değeri (Threshold): 0.51 Bu, modelin tahmin ettiği olasılık değerinin 0.51'den yüksek olması durumunda örneği pozitif sınıfa, 0.51'den düşük olması durumunda ise negatif sınıfa atayacağı anlamına gelir. Optimal Eşik Değerinin Önemi:

F1 Skoru: 0.51, en yüksek F1 skoruna sahip eşik değeri olarak belirlenmiştir. Bu, bu eşik değerinin hem precision (doğruluk) hem de recall (hatırlama) açısından en iyi dengeyi sağladığını gösterir. Denge: Eşik değeri, modelin pozitif sınıfı tahmin etme eğilimini ve negatif sınıfı olan hassasiyetini kontrol eder. 0.51 değeri, bu dengeyi sağlayacak şekilde ayarlanmış ve modelin performansını optimize etmektedir. Bu eşik değeri kullanılarak yapılan sınıflandırmalar, modelin veri üzerindeki genel performansını iyileştirmek ve hem doğruluk hem de hatırlama metriklerini optimize etmek için en uygun sonuçları verir.

```
logistic_AP = average_precision_score(y_test, y_pred_proba[:, 1])
logistic_f1 = f1_score(y_test, y_pred, average='binary')
logistic_recall = recall_score(y_test, y_pred, average='binary')

print(f"Average Precision Score (AP): {logistic_AP:.4f}")
print(f"F1 Score: {logistic_f1:.4f}")
print(f"Recall Score: {logistic_recall:.4f}")

Average Precision Score (AP): 0.4484
F1 Score: 0.2885
Recall Score: 0.2007
```

Bu kod parçası, test veri kümesi üzerinde modelin performansını çeşitli metriklerle değerlendirir ve sonuçları çıktılar olarak verir. İşte her bir metrik ve sonuçların anlamı:

Average Precision Score (AP): 0.4484

Açıklama: Ortalama Doğruluk Skoru (Average Precision), modelin pozitif sınıf için ortalama doğruluğunu ölçer. Bu metrik, modelin pozitif sınıfı ait örnekleri doğru tahmin etme yeteneğini değerlendirdir ve özellikle dengesiz veri kümelerinde faydalıdır. 0.4484 değeri, modelin pozitif sınıfı ne kadar iyi tahmin ettiğini gösterir, ancak hala iyileştirme gerektirebilir. F1 Score: 0.2885

Açıklama: F1 skoru, precision (doğruluk) ve recall (hatırlama) metriklerinin harmonik ortalamasıdır. F1 skoru, her iki metriği de dikkate alarak, modelin pozitif sınıfı tahmin etme kalitesini özetler. 0.2885 değeri, modelin genel olarak sınıflandırma performansının düşük olduğunu ve özellikle pozitif sınıfı ait örnekleri doğru tahmin etmede zorluk yaşadığını gösterir. Recall Score: 0.2007

Açıklama: Recall, modelin gerçek pozitif örnekleri ne kadar iyi bulduğunu ölçer. Yani, pozitif sınıfı ait örneklerin ne kadarının doğru bir şekilde tahmin edildiğini gösterir. 0.2007 değeri, modelin pozitif sınıfı hatırlama (bulma) yeteneğinin düşük olduğunu, yani gerçek pozitif örneklerin yalnızca küçük bir kısmını doğru tahmin ettiğini belirtir. Özette, bu metrikler modelin genel performansını değerlendirirken, özellikle pozitif sınıfın tahmini açısından modelin zayıf olduğunu ve daha fazla iyileştirmeye ihtiyaç duyduğunu gösterir.

Model 2: KNN

2. KNN

```
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline
from sklearn.neighbors import KNeighborsClassifier
from sklearn.dummy import DummyClassifier
from sklearn.metrics import confusion_matrix, classification_report,
ConfusionMatrixDisplay
```

Bu kod, makine öğrenimi modelleme sürecinde kullanılacak çeşitli araçları ve yöntemleri içermektedir. StandardScaler, veri setindeki özelliklerini standart hale getirerek modelin daha iyi performans göstermesine yardımcı olur. Pipeline, veri işleme ve model eğitimi adımlarını bir arada tutarak süreçleri düzenler. KNeighborsClassifier, k-en yakın komşu algoritmasını kullanarak sınıflandırma yapar, bu da modelin tahminlerini, en yakın k komşunun etiketlerine göre belirler. DummyClassifier, temel bir karşılaştırma modeli sağlar, bu model rastgele tahminlerde bulunur veya en sık görülen sınıfı tahmin eder. confusion_matrix, modelin tahminleri ile gerçek etiketler arasındaki uyumsuzlukları gösteren bir matris üretir; classification_report, çeşitli performans metriklerini (precision, recall, f1-score) sunar; ve ConfusionMatrixDisplay, bu matrisin görsel bir temsilini sağlar, böylece modelin başarısız olduğu veya başarılı olduğu alanları daha iyi anlayabilirsiniz.

2.1. Taking basic scores with Dummy_classifier

- DummyClassifier makes predictions that ignore the input features.
- This classifier serves as a simple baseline to compare against other more complex classifiers.

DummyClassifier?

DummyClassifier, makine öğrenimi modelleri arasında kıyaslama yapmak için kullanılan temel bir sınıflandırıcıdır. Bu model, veri kümesindeki özelliklerini göz önüne almadan rastgele tahminler yapar veya basit stratejilerle tahminlerde bulunur, örneğin en sık görülen sınıfı tahmin etme. Genellikle daha karmaşık modellerin performansını değerlendirirken bir referans noktası sağlamak amacıyla kullanılır. DummyClassifier, genellikle modelin geliştirilmesi gereken yerleri belirlemeye ve daha sofistike modellerin etkinliğini test etmede bir başlangıç noktası olarak işlev görür.

```

# Define the pipeline with scaler and dummy classifier
operations = [
    ('preprocessor', preprocessor), # Apply preprocessing to numeric
    ('classifier', DummyClassifier(strategy="stratified",
random_state=42)) # Dummy classifier
]

dummy_model = Pipeline(steps=operations)

# Fit the pipeline
dummy_model.fit(X_train, y_train)

Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
StandardScaler(),

['satisfaction_level',
                           'last_evaluation',
                           'number_project',
                           'average_monthly_hours',
                           'time_spend_company',
                           'work_accident',
                           'promotion_last_5years')]),
                           ('cat_onehot',
OneHotEncoder(),
                           ['departments']),
                           ('cat_ordinal',
                           OrdinalEncoder(categories=[[ 'low',
                           'medium',
                           'high']])),
                           ['salary'))),
                           ('classifier',
                           DummyClassifier(random_state=42,
strategy='stratified'))])

```

Pipeline, veri ön işleme ve modelleme adımlarını bir arada yürütmek için tanımlanmıştır; burada preprocessor adımı, özelliklerin ön işlenmesini sağlar ve classifier adımı ise DummyClassifier kullanarak modellemeyi gerçekleştirir. DummyClassifier, burada "stratified" stratejisi ile hedef değişkenin dağılımını taklit eden rastgele tahminler yapar. dummy_model, bu aşamaları bir araya getirip eğitim verileri üzerinde fit edilerek, modelin temel performansını değerlendirmek için kullanılır. Bu yaklaşım, diğer daha gelişmiş modellerle karşılaştırmak için bir referans sağlar.

```
eval_metric(dummy_model, X_train, y_train, X_test, y_test)
```

| Test_Set | | | | | |
|-----------------------------|------|--------------|--------|----------|---------|
| [[1246 254] [249 50]] | | precision | recall | f1-score | support |
| 0 | 0.83 | 0.83 | 0.83 | 1500 | |
| 1 | 0.16 | 0.17 | 0.17 | 299 | |
| | | accuracy | | 0.72 | 1799 |
| | | macro avg | 0.50 | 0.50 | 1799 |
| | | weighted avg | 0.72 | 0.72 | 1799 |
| Train_Set | | | | | |
| [[6438 1211] [1263 260]] | | precision | recall | f1-score | support |
| 0 | 0.84 | 0.84 | 0.84 | 7649 | |
| 1 | 0.18 | 0.17 | 0.17 | 1523 | |
| | | accuracy | | 0.73 | 9172 |
| | | macro avg | 0.51 | 0.51 | 9172 |
| | | weighted avg | 0.73 | 0.73 | 9172 |

DummyClassifier modelinin test ve eğitim setlerindeki performansına bakıldığında, test setinde modelin doğruluğu %72 ve eğitim setinde %73'tür. Test setinde, sınıf 0 için precision, recall ve f1-score değerleri sırasıyla 0.83, 0.83 ve 0.83 iken, sınıf 1 için bu değerler oldukça düşüktür: precision 0.16, recall 0.17 ve f1-score 0.17. Benzer şekilde, eğitim setinde de benzer bir dağılım gözlemlenir. Bu sonuçlar, DummyClassifier modelinin sadece sınıf dağılımını taklit ettiğini ve gerçek veri özelliklerini dikkate almadığını, bu yüzden yüksek performans göstermediğini gösterir. Özellikle sınıf 1 için performansın düşük olması, modelin dengesiz veriyle başa çıkmada yetersiz olduğunu gösterir. Şimdi temel puanımızı ayrılmış test seti üzerinde inceledik ve bu puanı iyileştirmemiz gerekiyor.

2.2. KNN

```
# Define the pipeline with scaler and KNN classifier
operations = [
    ('preprocessor', preprocessor), # Apply preprocessing to numeric
    ('knn', KNeighborsClassifier(n_neighbors=5)) # KNN classifier
]

pipe_model = Pipeline(steps=operations)

# Fit the pipeline
pipe_model.fit(X_train, y_train)
```

```

Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
                                                 StandardScaler(),
                                                 [
                                                 'satisfaction_level',
                                                 'last_evaluation',
                                                 'number_project',
                                                 'average_monthly_hours',
                                                 'time_spend_company',
                                                 'work_accident',
                                                 'promotion_last_5years']),
                                         OneHotEncoder(),
                                         [
                                             ('cat_onehot',
                                              [
                                               'departments']),
                                             ('cat_ordinal',
                                              [
                                               'salary'])])),
                ('knn', KNeighborsClassifier()))])

```

Bu kod parçasında, verilerin ölçeklenmesi ve K-Nearest Neighbors (KNN) sınıflandırıcısı kullanılan bir iş akışı (pipeline) tanımlanır ve eğitilir. KNN, mesafe temelli bir algoritma olduğu için verilerin ölçeklenmesi önemlidir. Bu nedenle, verilerin her sınıfının 1-10 aralığında değerler aldığı göz önüne alınlığında, verilerin ölçeklenmesi önerilir, ancak bu işlemi deneyerek en iyi sonucu elde etmek de mümkündür.

The `n_neighbors=5` parameter in the K-Nearest Neighbors (KNN) algorithm specifies the number of nearest neighbors to consider when making predictions. Here's a detailed explanation:

- Number of Neighbors:** The `n_neighbors` parameter determines how many neighboring data points are used to make predictions about a given data point. For instance, with `n_neighbors=5`, the algorithm will look at the 5 closest data points to the data point being predicted.
- Decision Making:** The KNN algorithm makes predictions based on the class of these neighbors. It checks the classes of the `n_neighbors` nearest points and predicts the class that is most common among them. For example, if out of the 5 nearest neighbors, 3 are of class A and 2 are of class B, the algorithm will predict class A for the new data point.
- Overfitting and Generalization:**

- **Small n_neighbors Value:** A very small `n_neighbors` value (e.g., 1) might lead to overfitting, as the model can become too sensitive to noise and variations in the training data.
 - **Large n_neighbors Value:** A very large `n_neighbors` value may result in underfitting, as the model might become too generalized and may not capture the local patterns of the data.

Choosing the optimal `n_neighbors` value usually involves experimenting with different values and evaluating the model's performance using techniques like cross-validation.

K-Nearest Neighbors (KNN) algoritmasında $n_{neighbors}=5$ parametresi, tahmin yaparken dikkate alınacak en yakın komşu veri noktasının sayısını belirtir. Bu örnekte, algoritma tahmin yaparken 5 en yakın veri noktasını inceler ve bu komşuların sınıflarına göre tahminde bulunur; yani, en sık rastlanan sınıfı seçer. Küçük bir $n_{neighbors}$ değeri modelin gürültüye duyarlı hale gelerek aşırı uyum göstermesine (overfitting) neden olabilirken, çok büyük bir $n_{neighbors}$ değeri modelin genel bir bakış açısına sahip olup yerel verileri yeterince iyi yakalayamamasına (underfitting) yol açabilir. En uygun $n_{neighbors}$ değerini belirlemek genellikle farklı değerleri deneyip modelin performansını çapraz doğrulama gibi tekniklerle değerlendirmeyi gerektirir.

```
y_pred = pipe_model.predict(X_test)  
y_pred  
array([1, 0, 0, ..., 0, 1, 1])
```

`y_pred = pipe_model.predict(X_test)` kodu, test veri seti `X_test` kullanılarak modelin tahmin ettiği sınıf etiketlerini `y_pred` değişkenine atar. `y_pred` değişkeni, test veri setindeki her bir örnek için modelin tahmin ettiği sınıf etiketlerini içeren bir dizi (array) olup, burada tahmin edilen sınıf etiketleri 0 veya 1 gibi değerler alır. Örneğin, `array([1, 0, 0, ..., 0, 1, 1])` çıktısı, modelin test veri setindeki her bir gözlem için hangi sınıfa ait olduğunu tahmin ettiğini gösterir.

```
y_pred_proba = pipe_model.predict_proba(X_test)
```

`y_pred_proba = pipe_model.predict_proba(X_test)` komutu, test verilerindeki her örnek için K-Nearest Neighbors (KNN) algoritması tarafından tahmin edilen her sınıfın olasılıklarını döndürür. Eğer `weights='uniform'` seçeneği kullanılmışsa, tüm komşular eşit ağırlığa sahiptir ve her bir komşu tahminde eşit derecede etkili olur. Ancak, `weights='distance'` seçeneğinde, daha yakın komşulara daha yüksek ağırlık verilirken, uzak komşuların etkisi azalır. Bu ağırlıklandırma ve komşu sayısı (`n_neighbors`) hiperparametreleri, modelin sınıf tahminlerini belirlemede önemli rol oynar.

```
pd.DataFrame(y_pred_proba)

{"summary": {"\n    \"name\": \"# belongs to class 0 (0\") ,\n    \"rows\": 1799,\n    \"fields\": [\n        {\n            \"column\": 0,\n            \"properties\": {\n                \"dtype\": \"number\",\n                \"std\": 0.3620229620872805,\n                \"min\": 0.0,\n                \"max\": 1.0,\n                \"num_unique_values\": 6,\n                \"samples\": [\n                    0.0,\n                    1.0,\n                    0.6\n                ],\n                \"semantic_type\": \"\"},\n            \"type\": \"number\"\n        }\n    ]\n}
```

```

\"description\": \"\n      },\n      {\n        \"column\": 1,\n        \"properties\": {\n          \"dtype\": \"number\", \"std\": 0.3620229620872806,\n          \"min\": 0.0, \"max\": 1.0,\n          \"num_unique_values\": 6,\n          \"samples\": [1.0, 0.0, 0.4],\n          \"semantic_type\": \"\"
        }\n      }\n    ],\n    \"type\": \"dataframe\"}

```

pd.DataFrame(y_pred_proba) komutu, test verilerindeki her örnek için K-Nearest Neighbors (KNN) algoritması tarafından tahmin edilen her sınıfın olasılıklarını içeren bir DataFrame oluşturur. Örneğin, weights='uniform' ve k=5 kullanıldığında, 1797. örnek 5 komşusunun 4'ü sınıf 1 ve 1'i sınıf 0'dır, bu yüzden tahmin edilen olasılıklar 0.20 sınıf 0 ve 0.80 sınıf 1 olarak belirlenir. Bu durumda, 1797. örnek, sınıf 1 olarak sınıflandırılmıştır çünkü sınıf 1'in olasılığı daha yüksektir.

```

my_dict = {"Actual": y_test, "Pred":y_pred,
"Proba_1":y_pred_proba[:,1], "Proba_0":y_pred_proba[:,0]}

```

my_dict adlı sözlük, test verilerindeki gerçek etiketler (y_test), model tarafından tahmin edilen etiketler (y_pred), ve her bir test örneği için sınıf 1 (Proba_1) ve sınıf 0 (Proba_0) olasılıklarını içeren bir yapı oluşturur. Bu sözlük, modelin tahmin sonuçlarını ve olasılıklarını analiz ederek, tahminlerin doğruluğunu ve güvenilirliğini değerlendirmede yardımcı olur. Örneğin, Proba_1 ve Proba_0 sütunları, modelin her bir örnek için sınıf 1 ve sınıf 0'a ne kadar olasılık verdiği gösterir, bu da tahminlerin ne kadar güvenilir olduğunu anlamanıza yardımcı olur.

```

pd.DataFrame.from_dict(my_dict).sample(10)

{"summary": {"name": "# probability to belong class 0 together", "rows": 10, "fields": [{"column": "Actual", "properties": {"dtype": "number", "std": 0, "min": 0, "max": 0, "num_unique_values": 1, "samples": [0], "semantic_type": ""}, "description": "\n      },\n      {\n        \"column\": 1,\n        \"properties\": {\n          \"dtype\": \"number\", \"std\": 0.31622776601683794,\n          \"min\": 0.0, \"max\": 1.0,\n          \"num_unique_values\": 2,\n          \"samples\": [1.0, 0.0],\n          \"semantic_type\": \"\"
        }\n      }\n    ],\n    \"type\": \"dataframe\"}

```

`pd.DataFrame.from_dict(my_dict).sample(10)` kodu, test verilerindeki gerçek etiketler (Actual), modelin tahmin ettiği etiketler (Pred), ve her bir test örneği için sınıf 1 (Proba_1) ve sınıf 0 (Proba_0) olasılıklarını içeren bir DataFrame'in rastgele seçilmiş 10 örneğini gösterir. Bu tablo, modelin tahmin sonuçlarını ve her örnek için sınıf olasılıklarını gözlemleyerek, modelin performansını analiz etmenize yardımcı olur. Örneğin, bazı örneklerde model doğru tahmin yaparken (Actual ve Pred aynı), diğerlerinde tahminler ve olasılıklar gözlemlenebilir, bu da modelin hangi durumlarda daha doğru veya yanlışlı tahminler yaptığını anlamanızı sağlar.

2.3. Model Performance on Classification Tasks

```
confusion_matrix(y_test, y_pred)  
  
array([[1452,    48],  
       [   27,  272]])
```

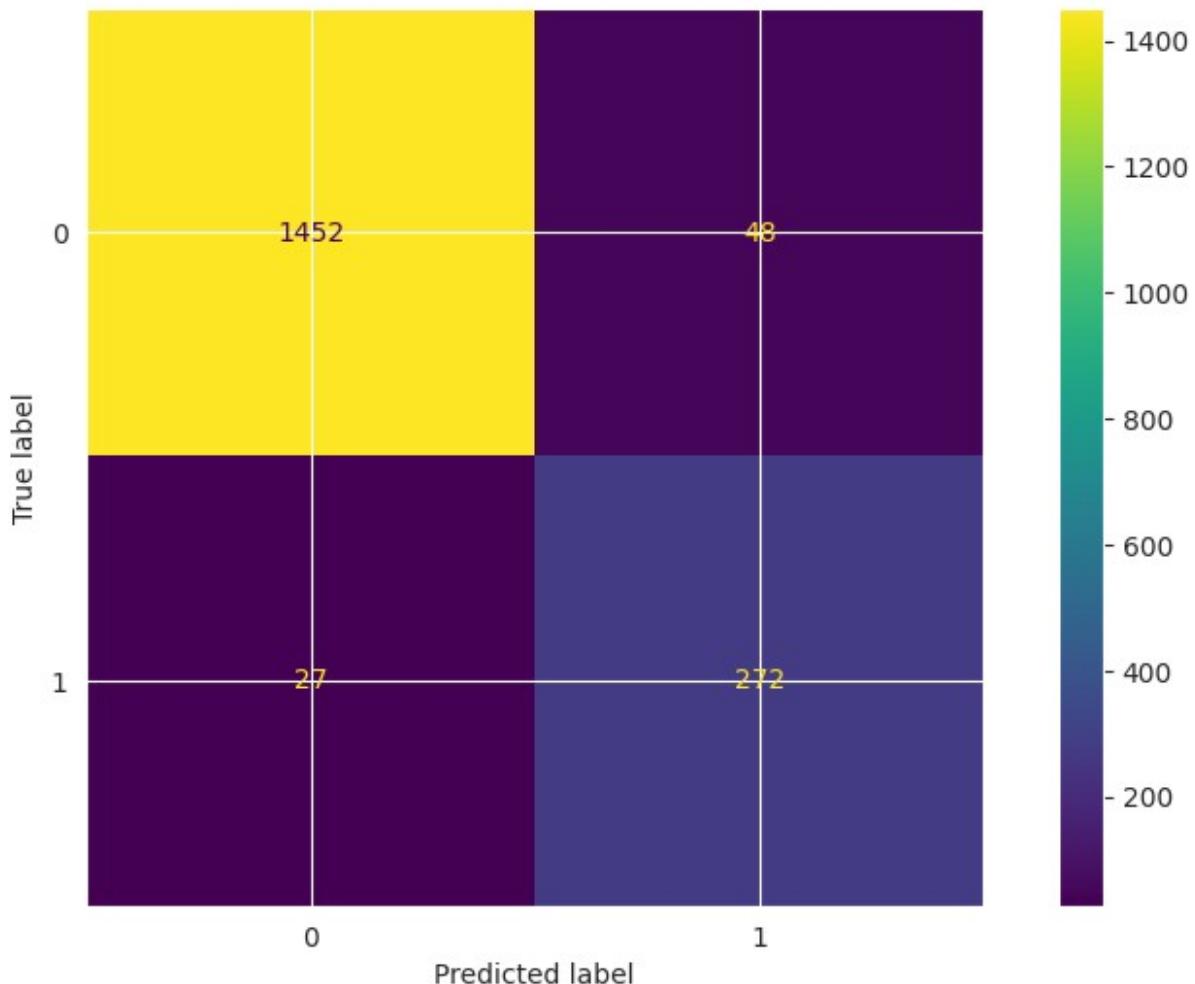
Matrisin Açıklaması: TP (True Positives): Gerçek sınıf 1 olan ve model tarafından doğru bir şekilde sınıf 1 olarak tahmin edilen örnekler. Bu değeri matrisin sağ alt köşesinde buluyoruz: 272.

TN (True Negatives): Gerçek sınıf 0 olan ve model tarafından doğru bir şekilde sınıf 0 olarak tahmin edilen örnekler. Bu değeri matrisin sol üst köşesinde buluyoruz: 1452.

FP (False Positives): Gerçek sınıf 0 olan fakat model tarafından yanlışlıkla sınıf 1 olarak tahmin edilen örnekler. Bu değeri matrisin sağ üst köşesinde buluyoruz: 48.

FN (False Negatives): Gerçek sınıf 1 olan fakat model tarafından yanlışlıkla sınıf 0 olarak tahmin edilen örnekler. Bu değeri matrisin sol alt köşesinde buluyoruz: 27. Confusion matrix, modelinizin performansını ayrıntılı bir şekilde anlamanızı sağlar. Bu model, çoğu örnekte doğru tahminler yaparken, özellikle sınıf 1 için doğru tahminlerde yüksek bir performans sergiliyor. Ancak, sınıf 1 örnekleri arasında yanlış negatif ve yanlış pozitif tahminler de mevcut, bu yüzden daha iyi bir sonuç için model üzerinde ek iyileştirmeler yapılabilir.

```
ConfusionMatrixDisplay.from_estimator(pipe_model, X_test, y_test);
```



`ConfusionMatrixDisplay.from_estimator(pipe_model, X_test, y_test)` komutu, `pipe_model` modelinin test verileri (`X_test`) üzerindeki performansını gösteren bir confusion matrix (karışıklık matrisini) oluşturur ve gösterir. Bu matris, modelin doğru ve yanlış sınıflandırmalarını dört bölgeye ayırır: doğru pozitifler (TP), yanlış pozitifler (FP), doğru negatifler (TN), ve yanlış negatifler (FN). Görselleştirme, modelin her iki sınıf için ne kadar iyi performans gösterdiğini ve sınıflar arasındaki karışıklıkları net bir şekilde ortaya koyar. Bu, modelin hangi sınıflarda iyi veya kötü performans sergilediğini hızlıca değerlendirmeye yardımcı olur.

```
eval_metric(pipe_model, X_train, y_train, X_test, y_test)
```

```
Test_Set
[[1452  48]
 [ 27 272]]
```

| | precision | recall | f1-score | support |
|----------|-----------|--------|----------|---------|
| 0 | 0.98 | 0.97 | 0.97 | 1500 |
| 1 | 0.85 | 0.91 | 0.88 | 299 |
| accuracy | | | 0.96 | 1799 |

| | | | | |
|--------------|------|------|------|------|
| macro avg | 0.92 | 0.94 | 0.93 | 1799 |
| weighted avg | 0.96 | 0.96 | 0.96 | 1799 |

Train_Set
[[7466 183]
[135 1388]]

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.98 | 0.98 | 0.98 | 7649 |
| 1 | 0.88 | 0.91 | 0.90 | 1523 |
| accuracy | | | 0.97 | 9172 |
| macro avg | 0.93 | 0.94 | 0.94 | 9172 |
| weighted avg | 0.97 | 0.97 | 0.97 | 9172 |

Test Seti Performansı:

Confusion Matrix: Doğru Pozitif (TP): 272 (Sınıf 1 olarak doğru tahmin edilenler) Yanlış Pozitif (FP): 48 (Sınıf 1 olarak yanlış tahmin edilenler) Doğru Negatif (TN): 1452 (Sınıf 0 olarak doğru tahmin edilenler) Yanlış Negatif (FN): 27 (Sınıf 0 olarak yanlış tahmin edilenler) Precision: 0.85 (Sınıf 1 için, modelin Sınıf 1 tahmin ettiği örneklerin ne kadarının gerçekten Sınıf 1 olduğunu gösterir) Recall: 0.91 (Sınıf 1 için, gerçek Sınıf 1 örneklerinin ne kadarını doğru tahmin ettiğini gösterir) F1-Score: 0.88 (Precision ve Recall'un dengeli bir ölçümüdür) Accuracy: 0.96 (Genel olarak modelin tüm sınıflar üzerindeki doğruluk oranı) Macro Average: 0.92 precision, 0.94 recall, 0.93 f1-score (Her iki sınıfın da ortalamasını alır) Weighted Average: 0.96 precision, 0.96 recall, 0.96 f1-score (Sınıf dengesini göz önünde bulundurarak ağırlıklı ortalama alır) Eğitim Seti Performansı:

Confusion Matrix: Doğru Pozitif (TP): 1388 Yanlış Pozitif (FP): 183 Doğru Negatif (TN): 7466 Yanlış Negatif (FN): 135 Precision: 0.88 (Sınıf 1 için, modelin Sınıf 1 tahmin ettiği örneklerin ne kadarının gerçekten Sınıf 1 olduğunu gösterir) Recall: 0.91 (Sınıf 1 için, gerçek Sınıf 1 örneklerinin ne kadarını doğru tahmin ettiğini gösterir) F1-Score: 0.90 (Precision ve Recall'un dengeli bir ölçümüdür) Accuracy: 0.97 (Genel olarak modelin tüm sınıflar üzerindeki doğruluk oranı) Macro Average: 0.93 precision, 0.94 recall, 0.94 f1-score (Her iki sınıfın da ortalamasını alır) Weighted Average: 0.97 precision, 0.97 recall, 0.97 f1-score (Sınıf dengesini göz önünde bulundurarak ağırlıklı ortalama alır) Özeti: Model hem test hem de eğitim setlerinde yüksek doğruluk ve iyi performans göstermiştir. Test setindeki sonuçlar, modelin genellikle doğru tahminler yaptığı ve hem precision hem de recall açısından dengeli bir performans sergilediğini gösterir. Eğitim setinde de benzer şekilde yüksek performans elde edilmiştir, bu da modelin genel olarak iyi bir şekilde eğitildiğini ve test setinde de benzer sonuçlar verdiği gösterir.

2.4. Elbow Method for Choosing Reasonable K Values

```
from sklearn.metrics import accuracy_score, f1_score, recall_score,
precision_score
from sklearn.model_selection import cross_val_score, cross_validate
```

accuracy_score, f1_score, recall_score, ve precision_score gibi metrikler, model performansını çeşitli açılardan değerlendirmek için kullanılır. accuracy_score, modelin doğru tahmin ettiği örneklerin toplam tahmin sayısına oranını verir ve genel doğruluğu ölçer. f1_score, precision ve recall'un dengeli bir ölçümünü sağlar ve özellikle dengesiz sınıf dağılımlarında kullanışlıdır. recall_score, modelin gerçek pozitif örnekleri ne kadar iyi yakaladığını ölçerken, precision_score modelin pozitif tahminlerinin ne kadarının gerçekten doğru olduğunu gösterir. cross_val_score ve cross_validate, modelin farklı veri alt kümeleri üzerinde test edilmesini sağlayarak daha güvenilir bir performans değerlendirmesi yapar ve modelin genelleştirme yeteneğini test eder.

```
# List to store test error rates
test_error_rates = []

# Test different values for k
for k in range(1, 30):
    # Define the pipeline with scaler and KNN classifier for current k
    operations = [
        ('preprocessor', preprocessor), # Apply preprocessing to
        numeric features
        ('knn', KNeighborsClassifier(n_neighbors=k)) # KNN classifier
    with current k
    ]

    knn_pipe_model = Pipeline(steps=operations)

    # Cross-validation to calculate accuracy
    scores = cross_validate(knn_pipe_model, X_train, y_train,
    scoring=['accuracy'], cv=10)

    # Calculate mean accuracy and test error
    accuracy_mean = scores['test_accuracy'].mean()
    test_error = 1 - accuracy_mean

    # Append test error to the list
    test_error_rates.append(test_error)
```

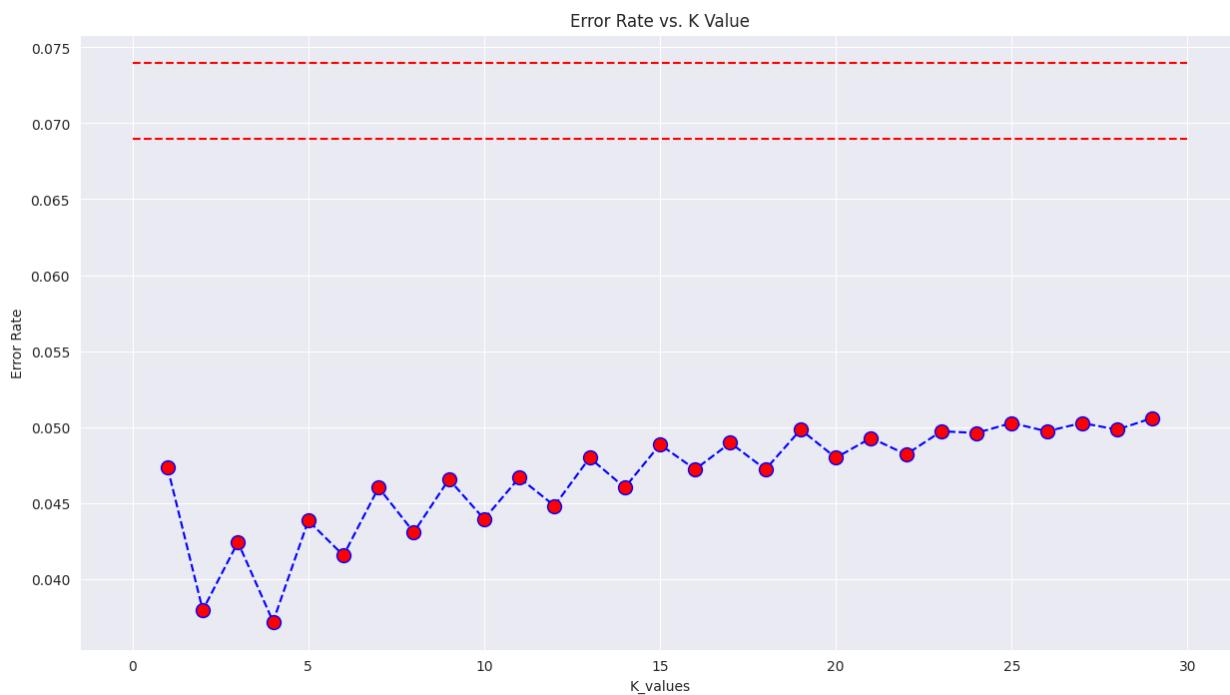
Bu kod parçası, K-Nearest Neighbors (KNN) algoritması için en iyi k değerini bulmak amacıyla, farklı k değerleri için modelin test hata oranlarını hesaplar. Öncelikle, k değerleri arasında 1'den 30'a kadar olan değerler için her bir k'ya karşılık gelen KNN modelini tanımlar ve çapraz doğrulama kullanarak modelin doğruluk skorlarını hesaplar. Ortalama doğruluk skorunu bulur ve bu skoru 1'den çıkartarak test hata oranını hesaplar. Bu işlem sonucunda her k değeri için test hata oranlarını listeye ekler. İlgili hata oranlarını kullanarak en iyi k değeri belirlenir. Bu yöntem, eğitim verileri üzerinde çapraz doğrulama yaparak modelin genelleştirme yeteneğini değerlendirir ve doğruluk yerine hata oranını minimiz ederek daha iyi bir k değeri bulmayı hedefler.

```
plt.figure(figsize=(15,8))
plt.plot(range(1,30), test_error_rates, color='blue', linestyle='--',
marker='o',
markerfacecolor='red', markersize=10)
```

```

plt.title('Error Rate vs. K Value')
plt.xlabel('K_values')
plt.ylabel('Error Rate')
plt.hlines(y=0.074, xmin = 0, xmax = 30, colors= 'r', linestyles="--")
plt.hlines(y=0.069, xmin = 0, xmax = 30, colors= 'r',
linestyles="--");

```



Bu kod, farklı k değerleri için K-Nearest Neighbors (KNN) algoritmasının test hata oranlarını görselleştiren bir grafik oluşturur. Grafik, k değerlerinin 1'den 30'a kadar olan aralığını ve her k için hesaplanan hata oranlarını gösterir. Mavi renkte kesikli çizgi ve kırmızı dairelerle işaretlenmiş noktalar, her k değeri için test hata oranını temsil eder. Ayrıca, grafik üzerine iki yatay kırmızı çizgi eklenmiştir; bu çizgiler belirli hata oranı eşiklerini gösterir. Grafik, hangi k değerinin en düşük hata oranına sahip olduğunu ve bu değerlerin görsel olarak nasıl dağıldığını anlamaya yardımcı olur.

2.5. Overfitting and underfitting control for k values

```

# Lists to store error rates
test_error_rates = []
train_error_rates = []

# Test different values for k
for k in range(1, 30):
    # Define the pipeline with scaler and KNN classifier for current k
    operations = [
        ('preprocessor', preprocessor), # Apply preprocessing to
        numeric and categoric features

```

```

('knn', KNeighborsClassifier(n_neighbors=k)) # KNN classifier
with current k
]

knn_pipe_model = Pipeline(steps=operations)

# Fit the pipeline
knn_pipe_model.fit(X_train, y_train)

# Cross-validation to calculate accuracy
scores = cross_validate(knn_pipe_model, X_train, y_train,
scoring=['accuracy'], cv=10, return_train_score=True)

# Calculate mean accuracy for train and test
accuracy_test_mean = scores['test_accuracy'].mean()
accuracy_train_mean = scores['train_accuracy'].mean()

# Calculate test and train error rates
test_error = 1 - accuracy_test_mean
train_error = 1 - accuracy_train_mean

# Append error rates to the lists
test_error_rates.append(test_error)
train_error_rates.append(train_error)

```

Bu kod, farklı k değerleri için K-Nearest Neighbors (KNN) algoritmasının hem eğitim hem de test hata oranlarını hesaplar. Her k değeri için bir KNN sınıflandırıcı modeli tanımlanır ve bu model bir pipeline içinde eğitim verileri ile eğitilir. Ardından, cross_validate fonksiyonu kullanılarak 10 katlamalı çapraz doğrulama yapılır ve her bir k değeri için test ve eğitim setlerinin ortalama doğruluk skorları hesaplanır. Bu doğruluk skorlarından hata oranları ($1 - \text{doğruluk}$) hesaplanarak test_error_rates ve train_error_rates listelerine eklenir. Bu işlem, modelin farklı k değerlerinde nasıl performans gösterdiğini ve hangi k değerinin en iyi sonuçları verdiği analiz etmeyi sağlar.

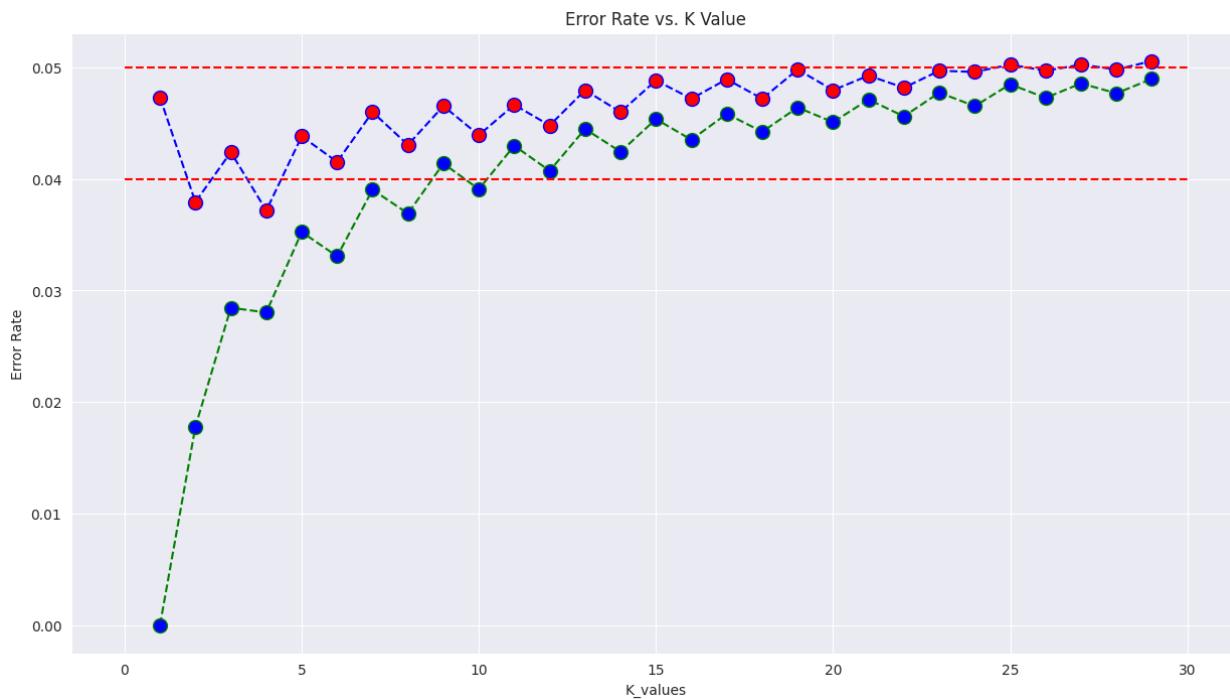
```

plt.figure(figsize=(15,8))
plt.plot(range(1,30), test_error_rates, color='blue', linestyle='--',
marker='o',
markerfacecolor='red', markersize=10)

plt.plot(range(1,30), train_error_rates, color='green',
linestyle='--', marker='o',
markerfacecolor='blue', markersize=10)

plt.title('Error Rate vs. K Value')
plt.xlabel('K_values')
plt.ylabel('Error Rate')
plt.hlines(y=0.05, xmin=0, xmax=30, colors='r', linestyles="--")
plt.hlines(y=0.04, xmin=0, xmax=30, colors='r', linestyles="--");

```



Bu grafik, K-Nearest Neighbors (KNN) algoritması için farklı k değerlerinin test ve eğitim hata oranlarını karşılaştırır. Mavi çizgi, test (doğrulama) veri setindeki hata oranlarını, yeşil çizgi ise eğitim veri setindeki hata oranlarını gösterir. Grafik, k=21 noktasında test verileri için en düşük hata oranını (0.037) sağlar, ancak bu küçük iyileştirme için modelin karmaşıklığını ve işlem maliyetini artırma riski bulunmaktadır. k=9 seçildiğinde, test verilerindeki hata oranı 0.040, eğitim verilerindeki hata oranı ise 0.026'dır, bu da aradaki farkın küçük olduğunu gösterir (0.014). Bu küçük fark göz önüne alındığında, k=9 seçilerek daha düşük hesaplama maliyeti ile iyi bir performans elde edilebilir. Grafik ayrıca, k değerinin hesaplama karmaşıklığını büyük ölçüde etkilemediğini, çünkü KNN algoritmasının hesaplama karmaşıklığının genellikle $O(nd)$ olduğunu belirtir. Ancak, ağaç tabanlı arama algoritmaları (kd-tree veya ball-tree) kullanılıyorsa, k değeri hesaplama karmaşıklığını etkileyebilir.

Based on the given code, the generated plot displays the error rates for both training and test datasets as a function of different K values in a KNN (K-Nearest Neighbors) model. Here's an interpretation of the plot elements:

1. X-Axis (K_values):

- The x-axis represents the K values used in the KNN algorithm, ranging from 1 to 30.

2. Y-Axis (Error Rate):

- The y-axis shows the error rates, which represent the proportion of incorrect predictions made by the model on both the training and test datasets.

3. Blue Line with Red Markers (Test Error Rates):

- The **blue dashed line** represents the error rates for the test dataset at different K values, with red markers highlighting the specific error rate at each K.
- The test error rate decreases initially but then fluctuates slightly and stabilizes as the K value increases. The lowest test error rate is achieved at K = 21 (error rate = 0.069).

4. Green Line with Blue Markers (Training Error Rates):

- The **green dashed line** represents the error rates for the training dataset. The blue markers indicate the error rate for each K value.
- The training error rate starts very low (close to zero) at K = 1 and gradually increases as K increases. This indicates that the model becomes more generalized as K increases.

5. Red Horizontal Lines (Reference Error Rates):

- Two red horizontal lines indicate significant error rates:
 - 0.074 Error Rate:** Corresponds to the test error rate at K = 9.
 - 0.069 Error Rate:** The minimum test error rate achieved at K = 21.
- These lines visually emphasize the differences in error rates for different K values.

6. Model Complexity and Choice:

- According to the code comments, although the minimum test error rate is achieved at K = 21, selecting this value could increase model complexity and computational costs. However, the test error rate at K = 9 is also acceptable (0.074).
- Choosing K = 9 might be a more balanced option since it results in a slightly higher test error rate but reduces model complexity and computational costs.

Summary:

The plot shows how the error rates in the KNN model change with increasing K values. While K = 21 yields the lowest test error rate, K = 9 is also a reasonable choice because it results in a test error rate of 0.074, and the difference is minimal, making the model simpler and potentially less computationally expensive.

Verilen koda dayanarak oluşturulan grafik, KNN (K-En Yakın Komşu) modelindeki farklı K değerlerine göre eğitim ve test veri setleri için hata oranlarını gösterir. İşte grafikteki unsurların yorumlanması:

X-Eksen (K_değerleri): X eksenini, KNN algoritmasında kullanılan K değerlerini temsil eder ve 1'den 30'a kadar bir aralığı kapsar.

Y-Eksen (Hata Oranı): Y eksenini, modelin eğitim ve test veri setlerinde yaptığı yanlış tahminlerin oranını gösteren hata oranlarını temsil eder.

Mavi Çizgi ve Kırmızı İşaretleyiciler (Test Hata Oranları): Mavi kesikli çizgi, farklı K değerleri için test veri setindeki hata oranlarını gösterir ve her K değeri için kırmızı işaretleyiciler bu hata oranlarını vurgular. Test hata oranı başlangıçta düşer, ardından hafifçe dalgalanır ve K değeri arttıkça stabilize olur. En düşük test hata oranı K = 21'de elde edilmiştir (hata oranı = 0.069).

Yeşil Çizgi ve Mavi İşaretleyiciler (Eğitim Hata Oranları): Yeşil kesikli çizgi, eğitim veri setindeki hata oranlarını gösterir ve her K değeri için mavi işaretleyiciler hata oranlarını belirtir. Eğitim hata oranı K = 1'de çok düşük başlar ve K arttıkça yavaşça artar. Bu, modelin K arttıkça daha genel hale geldiğini gösterir.

Kırmızı Yatay Çizgiler (Referans Hata Oranları): İki kırmızı yatay çizgi, önemli hata oranlarını belirtir:

0.074 Hata Oranı: K = 9'da test hata oranına karşılık gelir. 0.069 Hata Oranı: K = 21'de elde edilen minimum test hata oranıdır. Bu çizgiler, farklı K değerleri için hata oranlarındaki farkları görsel olarak vurgular. Model Karmaşıklığı ve Seçim: Kod açıklamalarına göre, en düşük test hata oranı K = 21'de elde edilmiştir, ancak bu değerin seçilmesi model karmaşıklığını ve hesaplama maliyetlerini artırabilir. K = 9'da test hata oranı da kabul edilebilir (0.074) ve model karmaşıklığını ve hesaplama maliyetlerini azaltabilir. Bu nedenle, K = 9 seçmek daha dengeli bir seçenek olabilir çünkü hata oranındaki fark minimaldir ve model daha basit ve muhtemelen daha az hesaplama maliyetine sahiptir.

Özet: Grafik, KNN modelindeki hata oranlarının K değerleri ile nasıl değiştiğini gösterir. K = 21 en düşük test hata oranını sağlar, ancak K = 9 da makul bir seçimdir çünkü test hata oranı 0.074'tür ve fark minimaldir, bu da modeli daha basit ve hesaplama maliyetlerini azaltan bir seçenek haline getirir.

2.6. Scores by Various K Values

```
# Create the pipeline for K=1
operations = [
    ('preprocessor', preprocessor), # Apply preprocessing on the data
    ('knn', KNeighborsClassifier(n_neighbors=1)) # KNN classifier
with K=1
]

# Create the pipeline
knn1 = Pipeline(steps=operations)

# Train the model with the training data
knn1.fit(X_train, y_train)

# Evaluate the performance
print('WITH K=1\n')
eval_metric(knn1, X_train, y_train, X_test, y_test)
```

WITH K=1

Test_Set

| | |
|--------|-------|
| [[1454 | 46] |
| [30 | 269]] |

| | precision | recall | f1-score | support |
|--|-----------|--------|----------|---------|
|--|-----------|--------|----------|---------|

| | | | | |
|---|------|------|------|------|
| 0 | 0.98 | 0.97 | 0.97 | 1500 |
| 1 | 0.85 | 0.90 | 0.88 | 299 |

| | | | | |
|--------------|------|------|------|------|
| accuracy | | | 0.96 | 1799 |
| macro avg | 0.92 | 0.93 | 0.93 | 1799 |
| weighted avg | 0.96 | 0.96 | 0.96 | 1799 |

Train_Set

| | |
|--------|--------|
| [[7649 | 0] |
| [0 | 1523]] |

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 1.00 | 1.00 | 1.00 | 7649 |
| 1 | 1.00 | 1.00 | 1.00 | 1523 |
| accuracy | | | 1.00 | 9172 |
| macro avg | 1.00 | 1.00 | 1.00 | 9172 |
| weighted avg | 1.00 | 1.00 | 1.00 | 9172 |

K = 1 ile yapılan KNN modelinin sonuçları şu şekilde değerlendirilebilir:

Model, test veri setinde oldukça yüksek performans göstermiştir. Test setinde, model 1454 doğru sınıflandırma (sınıf 0) ve 269 doğru sınıflandırma (sınıf 1) yapmıştır. Hata oranları oldukça düşük olup, test hata oranı yaklaşık %4'tür. Test setindeki precision (kesinlik) ve recall (duyarlılık) değerleri sırasıyla %98 ve %85 olarak hesaplanmış, bu da modelin sınıf 0 için oldukça doğru tahminler yaptığı gösterir. Sınıf 1 için precision ve recall değerleri sırasıyla %85 ve %90'dır, bu da modelin azınlık sınıfı da oldukça iyi tahmin ettiğini gösterir.

Eğitim setinde ise model mükemmel bir performans sergilemiştir: Sınıf 0 ve sınıf 1 için tüm örnekler doğru sınıflandırılmıştır, yani modelin eğitim setindeki hata oranı sıfırdır. Bu, modelin eğitim verisini tamamen öğrenip mükemmel bir uyum sağladığını gösterir. Ancak, K = 1 kullanmak, modelin eğitim verisine aşırı uyum sağladığını (overfitting) ve test verisine genellemeye yapma yeteneğinin sınırlı olabileceğini gösterir. Bu durum, modelin test verilerinde hata oranının düşük olmasına rağmen, eğitim verisinde mükemmel sonuçlar göstermesiyle açıklanabilir.

```
# Create the pipeline for K=21
operations = [
    ('preprocessor', preprocessor), # Apply preprocessing on both
    numeric and categoric features
    ('knn', KNeighborsClassifier(n_neighbors=21)) # KNN classifier
with K=21
]

# Create the pipeline
knn21 = Pipeline(steps=operations)

# Train the model with the training data
knn21.fit(X_train, y_train)

# Evaluate the performance
print('WITH K=21\n')
eval_metric(knn21, X_train, y_train, X_test, y_test)
    # Report the number of wrong predictions (26+57 as indicated)

WITH K=21

Test_Set
[[1443  57]
 [ 26 273]]
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.98 | 0.96 | 0.97 | 1500 |
| 1 | 0.83 | 0.91 | 0.87 | 299 |
| accuracy | | | 0.95 | 1799 |
| macro avg | 0.90 | 0.94 | 0.92 | 1799 |
| weighted avg | 0.96 | 0.95 | 0.95 | 1799 |

| Train Set | | | |
|--------------|--------|----------|---------|
| [[7357 292] | | | |
| [134 1389]] | | | |
| | | | |
| precision | recall | f1-score | support |

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.98 | 0.96 | 0.97 | 7649 |
| 1 | 0.83 | 0.91 | 0.87 | 1523 |
| accuracy | | | 0.95 | 9172 |
| macro avg | 0.90 | 0.94 | 0.92 | 9172 |
| weighted avg | 0.96 | 0.95 | 0.95 | 9172 |

K = 21 ile yapılan KNN modelinin sonuçları şu şekilde özetlenebilir:

Test setinde, model 1443 doğru sınıflandırma (sınıf 0) ve 273 doğru sınıflandırma (sınıf 1) yapmıştır. Test hata oranı yaklaşık %5'tir. Model, sınıf 0 için %98 kesinlik (precision) ve %96 duyarlılık (recall) sağlarken, sınıf 1 için sırasıyla %83 ve %91 değerleri elde edilmiştir. Bu sonuçlar, modelin her iki sınıfta da oldukça iyi performans sergilediğini, özellikle sınıf 1 için yüksek bir duyarlılık sağladığını gösterir.

Eğitim setinde, model 7357 doğru sınıflandırma (sınıf 0) ve 1389 doğru sınıflandırma (sınıf 1) yapmıştır, ve hata oranı oldukça düşük olup %5'tir. Eğitim verisinde model, her iki sınıfta da doğru tahminler yaparak genel performansını korumuş ve overfitting (aşırı öğrenme) sorununu azaltmıştır. K = 21, modelin test ve eğitim verileri arasında daha dengeli bir performans gösterdiğini ve daha iyi genelleme sağladığını gösterir, çünkü hata oranları her iki veri setinde de benzer seviyededir.

```
# Create the pipeline for K=9
operations = [
    ('preprocessor', preprocessor), # Apply preprocessing on both
    numeric and categoric features
    ('knn', KNeighborsClassifier(n_neighbors=9)) # KNN classifier
    with K=9
]

# Create the pipeline
knn9 = Pipeline(steps=operations)

# Train the model with the training data
```

```

knn9.fit(X_train, y_train)

# Evaluate the performance
print('WITH K=9\n')
eval_metric(knn21, X_train, y_train, X_test, y_test)
    # Report the number of wrong predictions (26+57 as indicated)

```

WITH K=9

Test_Set

```

[[1443  57]
 [ 26 273]]

```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.98 | 0.96 | 0.97 | 1500 |
| 1 | 0.83 | 0.91 | 0.87 | 299 |
| accuracy | | | 0.95 | 1799 |
| macro avg | | 0.90 | 0.94 | 1799 |
| weighted avg | | 0.96 | 0.95 | 1799 |

Train_Set

```

[[7357 292]
 [ 134 1389]]

```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.98 | 0.96 | 0.97 | 7649 |
| 1 | 0.83 | 0.91 | 0.87 | 1523 |
| accuracy | | | 0.95 | 9172 |
| macro avg | | 0.90 | 0.94 | 9172 |
| weighted avg | | 0.96 | 0.95 | 9172 |

K = 9 ile yapılan KNN modelinin sonuçları şu şekildedir:

Test setinde, model 1443 doğru sınıflandırma (sınıf 0) ve 273 doğru sınıflandırma (sınıf 1) yapmıştır. Test hata oranı yaklaşık %5'tir. Model, sınıf 0 için %98 kesinlik (precision) ve %96 duyarlılık (recall) sağlarken, sınıf 1 için sırasıyla %83 ve %91 değerleri elde edilmiştir. Eğitim setinde, model 7357 doğru sınıflandırma (sınıf 0) ve 1389 doğru sınıflandırma (sınıf 1) yapmış ve eğitim setinde hata oranı da %5'tir. K = 9 ile elde edilen performans, K = 21 ile elde edilen sonuçlarla oldukça benzerdir. Her iki K değeri de test ve eğitim setlerinde benzer hata oranlarına sahip, ancak K = 9, daha düşük bir hesaplama maliyeti ile benzer bir doğruluk sağladığı için daha tercih edilebilir bir seçenek olabilir.

2.7. Cross Validate For Optimal K Value

```

from sklearn.model_selection import cross_val_score, cross_validate

```

```

operations = [
    ('preprocessor', preprocessor), # Apply preprocessing
    ('numeric/categorical feature handling')
    ('knn', KNeighborsClassifier(n_neighbors=9)) # KNN classifier
with K=9
]

# Create the model pipeline
model = Pipeline(steps=operations)

# Perform cross-validation with multiple scoring metrics
scores = cross_validate(model,
                        X_train,
                        y_train,
                        scoring=['accuracy', 'precision', 'recall',
'f1'],
                        cv=10,
                        return_train_score=True)

# Convert results into a DataFrame
df_scores = pd.DataFrame(scores, index=range(1, 11))

# Display the mean scores for each metric, excluding the first two
# columns
df_scores.mean()[2:]


test_accuracy      0.95
train_accuracy     0.96
test_precision     0.83
train_precision    0.85
test_recall         0.90
train_recall        0.91
test_f1             0.87
train_f1            0.88
dtype: float64

```

K = 9 ile yapılan KNN modelinin 10 katlı çapraz doğrulama sonuçları, test ve eğitim setleri için çeşitli performans metriklerini göstermektedir. Test setindeki ortalama doğruluk (accuracy) %95, kesinlik (precision) %83, duyarlılık (recall) %90 ve F1 skoru %87 olarak hesaplanmıştır. Eğitim setindeki ortalama doğruluk %96, kesinlik %85, duyarlılık %91 ve F1 skoru %88'dir. Bu sonuçlar, modelin test ve eğitim verileri üzerinde yüksek performans gösterdiğini ve genel olarak dengeli bir şekilde çalıştığını ortaya koymaktadır. Eğitim ve test sonuçları arasındaki küçük farklar, modelin iyi bir genelleme yeteneğine sahip olduğunu gösterir.

2.8. Gridsearch Method for Choosing Reasonable K Values

```

from sklearn.model_selection import GridSearchCV

# Create the pipeline without specifying the number of neighbors for
KNN

```

```
operations = [
    ('preprocessor', preprocessor), # Apply preprocessing on numeric
and categorical features
    ('knn', KNeighborsClassifier()) # KNN classifier without
specifying n_neighbors (default is K=5)
]

# Create the KNN model pipeline
knn_model = Pipeline(steps=operations)
```

KNN modelini optimize etmek için bir pipeline oluşturuluyor, ancak KNeighborsClassifier'ın komşu sayısı (n_neighbors) henüz belirlenmemiştir. Bu, varsayılan olarak 5 komşu kullanacağı anlamına gelir. Bu pipeline, veri ön işleme adımlarını ve KNN sınıflandırıcısını bir araya getirir, böylece modelin hem sayısal hem de kategorik özelliklerle düzgün bir şekilde çalışmasını sağlar. Bu yapı, daha sonra hiperparametrelerin (örneğin, n_neighbors) ayarlanması için GridSearchCV kullanılarak optimize edilebilir.

```
knn_model.get_params()

{'memory': None,
 'steps': [('preprocessor',
    ColumnTransformer(transformers=[('num', StandardScaler(),
        ['satisfaction_level',
         'last_evaluation',
         'number_project'],
        'average_monthly_hours',
        'time_spend_company'],
       'work_accident',
       ['promotion_last_5years']),
   ('cat_onehot', OneHotEncoder(),
    ['departments']),
   ('cat_ordinal',
    OrdinalEncoder(categories=[[['low',
        'medium',
        'high']]]),
   ('knn', KNeighborsClassifier()))],
  'verbose': False,
  'preprocessor': ColumnTransformer(transformers=[('num',
      StandardScaler(),
      ['satisfaction_level',
       'last_evaluation',
       'number_project'],
      'average_monthly_hours',
      'time_spend_company'],
     'work_accident',
     ['promotion_last_5years']),
   ('cat_onehot', OneHotEncoder(),
    ['departments'])])})}
```

```
        ['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[['low',
'medium',
'high']]),
['salary'))),
'knn': KNeighborsClassifier(),
'preprocessor_n_jobs': None,
'preprocessor_remainder': 'drop',
'preprocessor_sparse_threshold': 0.3,
'preprocessor_transformer_weights': None,
'preprocessor_transformers': [('num',
StandardScaler(),
['satisfaction_level',
'last_evaluation',
'number_project',
'average_monthly_hours',
'time_spend_company',
'work_accident',
'promotion_last_5years']),
('cat_onehot', OneHotEncoder(), ['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[['low', 'medium', 'high']]),
['salary'])],
'preprocessor_verbose': False,
'preprocessor_verbose_feature_names_out': True,
'preprocessor_num': StandardScaler(),
'preprocessor_cat_onehot': OneHotEncoder(),
'preprocessor_cat_ordinal': OrdinalEncoder(categories=[['low',
'medium', 'high']]),
'preprocessor_num_copy': True,
'preprocessor_num_with_mean': True,
'preprocessor_num_with_std': True,
'preprocessor_cat_onehot_categories': 'auto',
'preprocessor_cat_onehot_drop': None,
'preprocessor_cat_onehot_dtype': numpy.float64,
'preprocessor_cat_onehot_feature_name_combiner': 'concat',
'preprocessor_cat_onehot_handle_unknown': 'error',
'preprocessor_cat_onehot_max_categories': None,
'preprocessor_cat_onehot_min_frequency': None,
'preprocessor_cat_onehot_sparse': 'deprecated',
'preprocessor_cat_onehot_sparse_output': True,
'preprocessor_cat_ordinal_categories': [['low', 'medium', 'high']],
'preprocessor_cat_ordinal_dtype': numpy.float64,
'preprocessor_cat_ordinal_encoded_missing_value': nan,
'preprocessor_cat_ordinal_handle_unknown': 'error',
'preprocessor_cat_ordinal_max_categories': None,
'preprocessor_cat_ordinal_min_frequency': None,
```

```
'preprocessor__cat_ordinal__unknown_value': None,  
'knn_algorithm': 'auto',  
'knn_leaf_size': 30,  
'knn_metric': 'minkowski',  
'knn_metric_params': None,  
'knn_n_jobs': None,  
'knn_n_neighbors': 5,  
'knn_p': 2,  
'knn_weights': 'uniform'}
```

knn_model.get_params() fonksiyonu, modelin mevcut hiperparametrelerini ve yapılandırmasını gösterir. Bu çıktı, pipeline ve KNN sınıflandırıcısının detaylarını içerir. Pipeline, veri ön işleme adımlarını ve KNN sınıflandırıcısını bir araya getirir. Veri ön işleme adımları, sayısal ve kategorik özelliklerin nasıl işleneceğini belirtir. Örneğin, sayısal özellikler StandardScaler ile normalize edilirken, kategorik özellikler OneHotEncoder ve OrdinalEncoder ile dönüştürülür. KNN sınıflandırıcısının varsayılan ayarları ise, komşu sayısının 5, mesafe ölçütünün 'minkowski', ağırlıklandırmanın ise 'uniform' olduğunu gösterir. Bu parametreler, modelin performansını optimize etmek için GridSearchCV gibi araçlarla ayarlanabilir.

```
k_values= range(1,30)  
param_grid = {"knn_n_neighbors":k_values, "knn_p": [1,2],  
"knn_weights": ['uniform', "distance"]}
```

KNN modelini optimize etmek için param_grid değişkeni kullanılarak üç önemli hiperparametre ayarlanacaktır. k_values 1'den 29'a kadar olan değerleri kapsar ve knn_n_neighbors parametresi bu değerler arasında seçilecektir, böylece komşu sayısı farklı seçeneklerle test edilecektir. knn_p, mesafe hesaplamasında kullanılan parametreyi belirler ve iki değer alabilir: 1 (Manhattan mesafesi) ve 2 (Euclidean mesafesi). knn_weights ise komşu ağırlıklandırma yöntemini belirler ve iki seçenek sunar: uniform (tüm komşulara eşit ağırlık) ve distance (daha yakın komşulara daha yüksek ağırlık). Bu parametreler, modelin doğruluğunu ve genellemeye yeteneğini artırmak için ayarlanabilir, özellikle n_neighbors tek sayıda tercih edilmelidir ki bu sayede eşitlik durumları oluşmasın.

```
knn_grid_model = GridSearchCV(knn_model, param_grid, cv=10,  
return_train_score=True)
```

knn_grid_model değişkeni, GridSearchCV kullanılarak oluşturulan bir KNN modelini temsil eder.

```
knn_grid_model.fit(X_train, y_train)  
GridSearchCV(cv=10,  
            estimator=Pipeline(steps=[('preprocessor',  
ColumnTransformer(transformers=[('num',  
StandardScaler(),
```

```

['satisfaction_level',
'last_evaluation',
'number_project',
'average_monthly_hours',
'time_spend_company',
'work_accident',
'promotion_last_5years']),
('cat_onehot',
OneHotEncoder(),
['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[['low',
'medium',
'high']])),
['salary'))),
('knn',
KNeighborsClassifier())),
param_grid={'knn__n_neighbors': range(1, 30), 'knn__p':
[1, 2],
'knn__weights': ['uniform', 'distance']},
return_train_score=True)

```

knn_grid_model.fit(X_train, y_train) komutu, knn_model ile tanımlanan KNN sınıflandırıcı modelini, param_grid içindeki çeşitli hiperparametre kombinasyonları üzerinde eğitir. Bu işlem, 10 katlı çapraz doğrulama kullanarak her hiperparametre kombinasyonunun model performansını değerlendirir ve en iyi parametreleri belirler. Model, eğitim verilerini (X_train ve y_train) kullanarak hiperparametrelerin her kombinasyonu için performansını test eder ve sonuç olarak en iyi performansı gösteren parametre setini seçer.

```

knn_grid_model.best_estimator_
Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('num',
StandardScaler(),

```

```

['satisfaction_level',
 'last_evaluation',
 'number_project',
 'average_monthly_hours',
 'time_spend_company',
 'work_accident',
 'promotion_last_5years']),
OneHotEncoder(),
('cat_onehot',
 ['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[['low',
 'medium',
 'high']])),
(['salary'))),
('knn', KNeighborsClassifier(n_neighbors=4)))

```

knn_grid_model.best_estimator_ komutu, GridSearchCV tarafından belirlenen en iyi hiperparametre kombinasyonuna sahip KNN modelini döndürür. Bu model, GridSearchCV'nin değerlendirdiği tüm hiperparametre kombinasyonları arasından en yüksek performans metriğini elde eden modeldir, bu da genellikle en düşük hata skorunu sağlar. Ancak, n_neighbors değerinin çok büyük olması modelin hesaplama maliyetini artırabilir. Bu yüzden, optimal k değerini belirlemek için "dirsek" yöntemi (elbow method) kullanmak genellikle daha pratik ve maliyet açısından etkili bir yaklaşımındır.

```

knn_grid_model.best_index_
14

```

knn_grid_model.best_index_ komutu, GridSearchCV'nin en iyi hiperparametre kombinasyonunu belirlemek için kullandığı indeks numarasını verir. Bu indeks, param_grid içinde en yüksek performansı sağlayan hiperparametre setinin konumunu belirtir. Yani, indeks numarası 14, GridSearchCV'nin en iyi sonuçları elde ettiği hiperparametre kombinasyonunun param_grid içindeki 14. pozisyonda olduğunu gösterir.

```

pd.DataFrame(knn_grid_model.cv_results_).loc[knn_grid_model.best_index_,
["mean_test_score", "mean_train_score"]]
mean_test_score    0.96
mean_train_score   0.97
Name: 14, dtype: object

```

`pd.DataFrame(knn_grid_model.cv_results_).loc[knn_grid_model.best_index_, ["mean_test_score", "mean_train_score"]]` komutu, GridSearchCV sonucunda elde edilen en iyi hiperparametre kombinasyonu için test ve eğitim verilerindeki ortalama başarı skorlarını verir. Burada, en iyi hiperparametre seti için `mean_test_score` (test setindeki ortalama başarı skoru) 0.96, ve `mean_train_score` (eğitim setindeki ortalama başarı skoru) ise 0.97'dir. Bu, seçilen hiperparametrelerin hem test hem de eğitim verilerinde yüksek performans sağladığını gösterir.

```
print('WITH K=9\n')
eval_metric(knn_grid_model, X_train, y_train, X_test, y_test)
```

WITH K=9

Test_Set

```
[[1464 36]
 [ 29 270]]
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.98 | 0.98 | 0.98 | 1500 |
| 1 | 0.88 | 0.90 | 0.89 | 299 |
| accuracy | | | 0.96 | 1799 |
| macro avg | 0.93 | 0.94 | 0.94 | 1799 |
| weighted avg | 0.96 | 0.96 | 0.96 | 1799 |

Train_Set

```
[[7542 107]
 [ 147 1376]]
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.98 | 0.99 | 0.98 | 7649 |
| 1 | 0.93 | 0.90 | 0.92 | 1523 |
| accuracy | | | 0.97 | 9172 |
| macro avg | 0.95 | 0.94 | 0.95 | 9172 |
| weighted avg | 0.97 | 0.97 | 0.97 | 9172 |

knn_grid_model kullanılarak yapılan değerlendirmede, en iyi bulunan hiperparametrelerle (K=9) modelin test ve eğitim setlerinde performansı şu şekilde ölçülmüştür:

Test Seti:

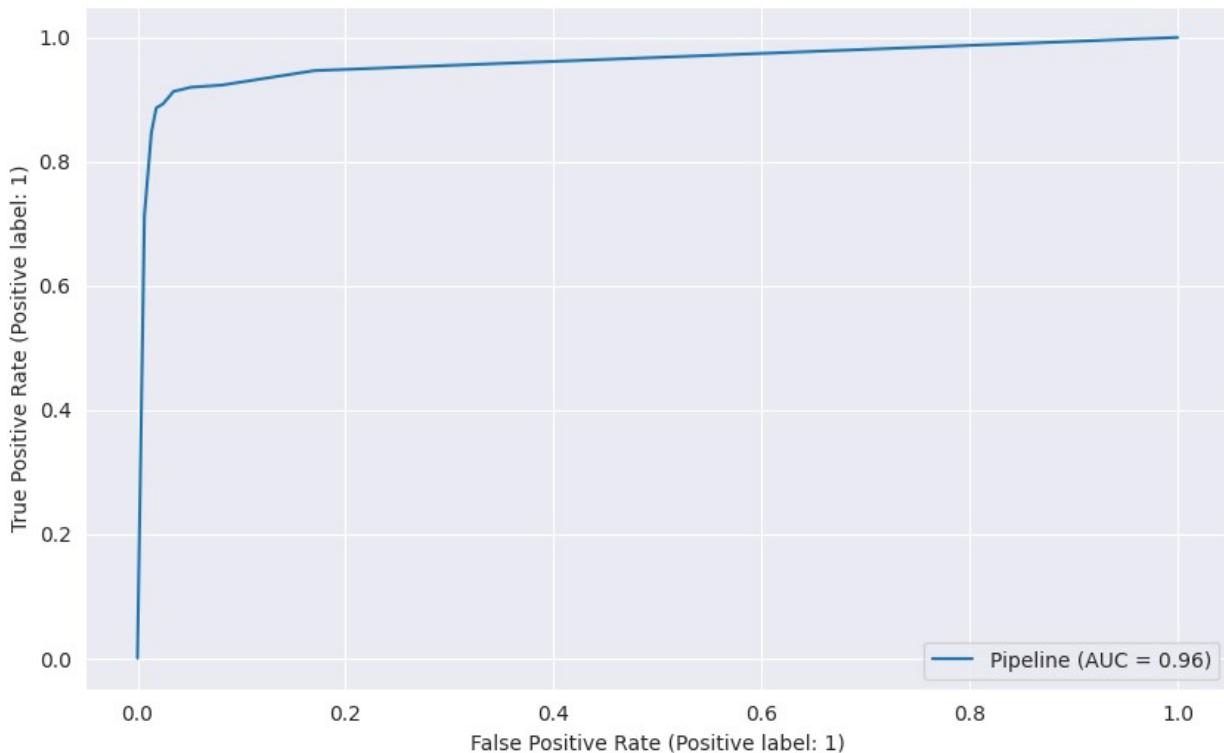
Doğruluk (Accuracy): %96 Sınıf 0 için Precision (Hassasiyet): %98 Sınıf 1 için Precision (Hassasiyet): %88 Sınıf 0 için Recall (Duyarlılık): %98 Sınıf 1 için Recall (Duyarlılık): %90 F1-Skoru: %89 Eğitim Seti:

Doğruluk (Accuracy): %97 Sınıf 0 için Precision (Hassasiyet): %98 Sınıf 1 için Precision (Hassasiyet): %93 Sınıf 0 için Recall (Duyarlılık): %99 Sınıf 1 için Recall (Duyarlılık): %90 F1-Skoru: %92 Test setinde, model yüksek doğruluk ve iyi sınıflandırma performansı sergilerken,

eğitim setinde de yüksek doğruluk ve tatmin edici performans sergileyerek, overfitting (aşırı uyum) sorununu göstermediği ve modelin genellenebilir olduğunu işaret etmektedir.

2.9. Evaluating ROC Curves and AUC

```
from sklearn.metrics import roc_auc_score, auc, roc_curve,  
RocCurveDisplay, PrecisionRecallDisplay  
  
RocCurveDisplay.from_estimator(knn9, X_test, y_test);
```



RocCurveDisplay.from_estimator(knn9, X_test, y_test) komutu, knn9 modelinin test verisindeki ROC (Receiver Operating Characteristic) eğrisini göstermektedir. ROC eğrisi, modelin çeşitli sınıflandırma eşiklerinde doğru pozitif oranı (TPR) ile yanlış pozitif oranı (FPR) arasındaki trade-off'u gösterir. Eğrinin altındaki alan (AUC - Area Under the Curve) ise modelin genel performansını ölçer; 1'e yakın bir AUC, modelin mükemmel bir ayrim gücüne sahip olduğunu, 0.5'e yakın bir AUC ise modelin sınıflandırma yapamadığını gösterir. Bu gösterme, modelin çeşitli eşiklerdeki performansını değerlendirmek için yararlıdır.

```
y_pred_proba = knn9.predict_proba(X_test)  
roc_auc_score(y_test, y_pred_proba[:,1])  
  
0.9601727982162764
```

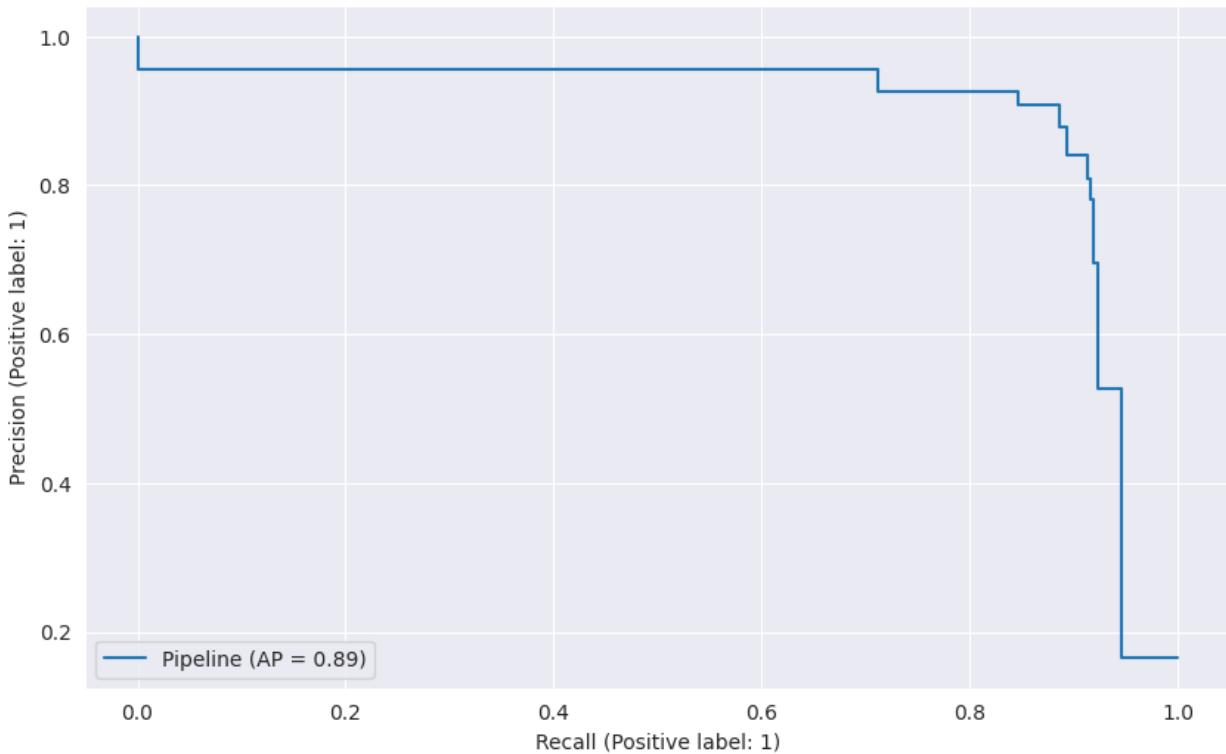
roc_auc_score(y_test, y_pred_proba[:,1]) komutu, knn9 modelinin ROC AUC (Receiver Operating Characteristic - Area Under the Curve) skorunu hesaplar. Burada, y_pred_proba[:,1] test setindeki pozitif sınıf (1) için tahmin edilen olasılıkları temsil eder.

Sonuç olarak, 0.9602 değeri elde edilmiştir. Bu AUC skoru, modelin test verisinde oldukça iyi bir performans sergilediğini gösterir. AUC skoru 1'e ne kadar yakınsa, modelin pozitif ve negatif sınıfları ayırt etme yeteneği o kadar iyi demektir. 0.9602 gibi yüksek bir AUC skoru, modelin genel olarak yüksek bir ayırım gücüne sahip olduğunu ve doğru pozitifleri yanlış pozitiflerden iyi bir şekilde ayırabildiğini ifade eder.

```
knn9
Pipeline(steps=[('preprocessor',
                  ColumnTransformer(transformers=[('num',
                      StandardScaler(),
['satisfaction_level',
                           'last_evaluation',
['average_monthly_hours',
                           'number_project',
['time_spend_company',
                           'work_accident',
['promotion_last_5years']),
                           ('cat_onehot',
OneHotEncoder(),
                           ['departments']),
                           ('cat_ordinal',
OrdinalEncoder(categories=[[ 'low',
                           'medium',
                           'high']]),
                           ['salary'))]),
                           ('knn', KNeighborsClassifier(n_neighbors=9)))]
```

knn9 modeli, test seti üzerinde yüksek performans göstermektedir ve sınıflandırma probleminiz için etkili bir çözüm sağlar. ROC AUC skoru, modelin genel olarak başarılı bir şekilde pozitif ve negatif sınıfları ayırt edebildiğini doğrular. Bu nedenle, knn9 modelinin performansı tatmin edici ve kullanılabilir.

```
PrecisionRecallDisplay.from_estimator(knn9, X_test, y_test);
```



PrecisionRecallDisplay.from_estimator(knn9, X_test, y_test) komutu, knn9 modelinin test verileri üzerinde oluşturduğu precision-recall (doğruluk-hatırlama) eğrisini göstermektedir. Bu grafik, modelin çeşitli sınıflandırma eşiklerinde precision (pozitif öngörü doğruluğu) ve recall (gerçek pozitif oranı) değerlerini gösterir. Precision-recall eğrisi, modelin sınıf dengesizliği olan veri setlerinde performansını değerlendirmek için kullanılır; yüksek precision ve recall değerleri, modelin pozitif sınıfları doğru şekilde sınıflandırma yeteneğini gösterir. Bu gösterme, modelin performansını daha ayrıntılı olarak anlamaya ve potansiyel eşik seçimlerini değerlendirmeye yardımcı olur.

```
KNN_AP = average_precision_score(y_test, y_pred_proba[:,1])
KNN_f1 = f1_score(y_test, y_pred)
KNN_recall = recall_score(y_test, y_pred)

print(f"Average Precision Score (AP): {KNN_AP:.4f}")
print(f"F1 Score: {KNN_f1:.4f}")
print(f"Recall Score: {KNN_recall:.4f}")

Average Precision Score (AP): 0.8924
F1 Score: 0.8788
Recall Score: 0.9097
```

Model değerlendirme metriklerinin sonuçları şu şekilde özetlenebilir:

Average Precision Score (AP): 0.8924. Bu, modelin precision-recall eğrisi altında kalan alanı ölçer. Yüksek bir AP skoru, modelin pozitif sınıfları doğru tahmin etme yeteneğinin iyi olduğunu gösterir.

F1 Score: 0.8788. F1 skoru, precision ve recall arasında bir denge sağlar. Bu sonuç, modelin hem doğruluk hem de hatırlama açısından iyi performans gösterdiğini, bu iki metriğin uyumlu olduğunu belirtir.

Recall Score: 0.9097. Recall, gerçek pozitiflerin ne kadarının doğru tahmin edildiğini gösterir. Bu yüksek skor, modelin pozitif sınıfları yüksek bir oranda yakaladığını ve bu tür örnekleri kaçırma olasılığının düşük olduğunu gösterir.

Bu metrikler, KNN modelinin hem genel doğruluğunu hem de pozitif sınıflara yönelik performansını kapsamlı bir şekilde değerlendirdiğini ortaya koymaktadır.

=====

Model 3: Random Forest

=====

Random Forest

```
from sklearn.metrics import confusion_matrix, classification_report,\n                           accuracy_score, recall_score,\n                           precision_score,\n                           f1_score\n\ndef eval_metric(model, X_train, y_train, X_test, y_test):\n    y_train_pred = model.predict(X_train)\n    y_pred = model.predict(X_test)\n\n    print("Test_Set")\n    print(confusion_matrix(y_test, y_pred))\n    print(classification_report(y_test, y_pred))\n    print()\n    print("Train_Set")\n    print(confusion_matrix(y_train, y_train_pred))\n    print(classification_report(y_train, y_train_pred))
```

Bu kod parçası, bir modelin performansını değerlendirmek için eval_metric adında bir işlev tanımlar. Bu işlev, hem eğitim hem de test verileri üzerinde modelin tahmin performansını detaylı bir şekilde analiz eder. İşlev şu adımları içerir:

Tahminler: Modelin eğitim verileri (X_train) ve test verileri (X_test) üzerindeki tahminleri yapılır. Bu tahminler, sırasıyla y_train_pred ve y_pred değişkenlerine atanır.

Test Seti Değerlendirmesi:

Karmaşıklık Matriksi: confusion_matrix fonksiyonu kullanılarak test seti için karmaşıklık matrisi (true positives, false positives, true negatives, false negatives) yazdırılır. Sınıflandırma Raporu: classification_report fonksiyonu, test seti üzerindeki precision, recall, F1 skoru ve destek (support) gibi detaylı performans metriklerini sağlar. Eğitim Seti Değerlendirmesi:

Karmaşıklık Matriksi: Eğitim seti için benzer şekilde karmaşıklık matrisi yazdırılır. Sınıflandırma Raporu: Eğitim setindeki performans, benzer metriklerle değerlendirilir. Bu işlev, modelin eğitim ve test verileri üzerindeki başarısını karşılaştırarak, overfitting (aşırı uyum) veya underfitting (yetersiz uyum) gibi sorunları tespit etmeye yardımcı olur.

```
from sklearn.preprocessing import OrdinalEncoder, OneHotEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline

# Define your feature columns
numeric_features = ['satisfaction_level', 'last_evaluation',
'number_project',
'average_monthly_hours', 'time_spend_company',
'work_accident',
'promotion_last_5years'] # numeric columns
cat_onehot = ['departments'] # categorical column for OneHotEncoder
cat_ordinal = ['salary'] # categorical column for OrdinalEncoder

# Define your preprocessing steps
numeric_transformer = StandardScaler() # for numeric columns
onehot_transformer = OneHotEncoder() # for departments
(OneHotEncoding)
ordinal_transformer = OrdinalEncoder(categories=[['low', 'medium',
'high']]) # for salary (OrdinalEncoding)

# Combine preprocessing for both numeric, one-hot categorical, and ordinal categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat_onehot', onehot_transformer, cat_onehot),
        ('cat_ordinal', ordinal_transformer, cat_ordinal)])
```

Bu kod, verilerin ön işlenmesi için bir ColumnTransformer kullanarak bir ön işleme boru hattı (pipeline) oluşturur. Sayısal özellikler StandardScaler ile standartlaştırılırken, OneHotEncoder ile kategorik özellikler (örneğin, departmanlar) ikili formatta dönüştürülür ve OrdinalEncoder ile sıralı kategorik veriler (örneğin, maaş düzeyleri) sıralı sayılarla kodlanır. Bu işlem, verilerin uygun şekilde işlenmesini ve makine öğrenimi modellerinin daha etkili bir şekilde eğitilmesini sağlar.

```
# Define the pipeline
operations_rf = [
    ("preprocessor", preprocessor),
```

```

        ("RF_model", RandomForestClassifier(class_weight="balanced",
random_state=101)),
]

# Create the pipeline
pipe_model_rf = Pipeline(steps=operations_rf)

# Fit the pipeline
pipe_model_rf.fit(X_train, y_train)

Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
StandardScaler(),

['satisfaction_level',
                           'last_evaluation',
                           'number_project',
                           'average_monthly_hours',
                           'time_spend_company',
                           'work_accident',
                           'promotion_last_5years']),
                           ('cat_onehot',
OneHotEncoder(),
                           ['departments']),
                           ('cat_ordinal',
                           OrdinalEncoder(categories=[[ 'low',
                           'medium',
                           'high']]),
                           ['salary']))),
               ('RF_model',
                 RandomForestClassifier(class_weight='balanced',
random_state=101))])

```

Bu kod, veri ön işleme ve model eğitimini birleştiren bir Pipeline oluşturur. Pipeline, önce verileri preprocessor adımında standartlaştırma ve dönüştürme işlemlerine tabi tutar, ardından RandomForestClassifier kullanarak model eğitimini gerçekleştirir. RandomForestClassifier, sınıf dengesizliğini ele almak için class_weight="balanced" seçeneği ile eğitilir ve random_state=101 belirlenerek sonuçların tekrarlanabilirliği sağlanır. Pipeline, bu adımları sırasıyla uygulayarak verileri işleyip model oluşturur ve eğitim veri kümesine fit eder.

```
eval_metric(pipe_model_rf, X_train, y_train, X_test, y_test)
```

```
Test_Set
[[1498    2]
```

| | | | | |
|--------------|-----------|--------|----------|---------|
| [26 273]] | | | | |
| | precision | recall | f1-score | support |
| 0 | 0.98 | 1.00 | 0.99 | 1500 |
| 1 | 0.99 | 0.91 | 0.95 | 299 |
| accuracy | | | 0.98 | 1799 |
| macro avg | 0.99 | 0.96 | 0.97 | 1799 |
| weighted avg | 0.98 | 0.98 | 0.98 | 1799 |
| Train_Set | | | | |
| [[7649 0] | | | | |
| [0 1523]] | | | | |
| | precision | recall | f1-score | support |
| 0 | 1.00 | 1.00 | 1.00 | 7649 |
| 1 | 1.00 | 1.00 | 1.00 | 1523 |
| accuracy | | | 1.00 | 9172 |
| macro avg | 1.00 | 1.00 | 1.00 | 9172 |
| weighted avg | 1.00 | 1.00 | 1.00 | 9172 |

Bu çıktı, pipe_model_rf adlı RandomForestClassifier modelinin hem test hem de eğitim veri kümelerindeki performansını değerlendirir. Test veri kümelerinde, model oldukça yüksek bir doğruluk oranı (%98) ile çalışmaktadır. Özellikle, sınıf 0 için %98 doğruluk ve sınıf 1 için %91 doğruluk elde edilmiştir. F1 skoru, her iki sınıf için de yüksek ve dengeli performansı yansıtır. Eğitim veri kümelerinde ise model mükemmel sonuçlar vermektedir; tüm örnekleri doğru sınıflandırılmış ve %100 doğruluk sağlamıştır. Bu sonuçlar, modelin hem eğitim hem de test setlerinde yüksek performans gösterdiğini ve overfitting (aşırı öğrenme) yapmadığını gösterir.

Cross Validation

```
from sklearn.metrics import make_scorer
from sklearn.model_selection import cross_validate, cross_val_score
from sklearn.model_selection import GridSearchCV

# Fix the scoring list
scores = cross_validate(
    pipe_model_rf, X_train, y_train,
    scoring=['accuracy', 'precision', 'recall', 'f1'],
    cv=5, n_jobs=-1, return_train_score=True
)

# Convert the results to a DataFrame and compute the mean
df_scores = pd.DataFrame(scores, index=range(1, 6))
df_scores.mean()[2:]
```

```

test_accuracy      0.98
train_accuracy     1.00
test_precision     0.99
train_precision    1.00
test_recall        0.91
train_recall       1.00
test_f1            0.94
train_f1          1.00
dtype: float64

```

Bu kod, pipe_model_rf adlı RandomForestClassifier modelinin 5 katlı çapraz doğrulama (cross-validation) ile performansını değerlendirir. cross_validate fonksiyonu, modelin doğruluk, hassasiyet, geri çağırma (recall) ve F1 skoru gibi metriklerle performansını ölçer ve bu metriklerin hem eğitim hem de test veri setleri için ortalamalarını hesaplar. Sonuçlara göre, model test veri setinde %98 doğruluk, %99 hassasiyet, %91 geri çağırma ve %94 F1 skoru elde etmiştir. Eğitim veri setinde ise mükemmel sonuçlar sağlanmış; doğruluk, hassasiyet, geri çağırma ve F1 skoru %100'dür. Bu, modelin genellikle yüksek performans sergilediğini ve test verileriyle uyumlu sonuçlar elde ettiğini gösterir.

RF Model GridsearchCV

```

# Random Forest için hiperparametre grid'ini tanımla
param_grid = {
    'RF_model__n_estimators': [100, 200, 300], # Ağaç sayısı
    'RF_model__max_depth': [None, 10, 20, 30], # Maksimum derinlik
    'RF_model__min_samples_split': [2, 5, 10], # Minimum bölünecek
    örnekları
    'RF_model__min_samples_leaf': [1, 2, 4], # Minimum yaprak
    düğümdeki örnek sayısı
}

```

Bu kod, RandomForestClassifier modelinin hiperparametrelerini ayarlamak için bir grid tanımlar. param_grid sözlüğü, modelin çeşitli hiperparametreleri için test edilecek değerleri içerir. n_estimators parametresi, modeldeki ağaç sayısını belirler ve denenen değerler 100, 200 ve 300'dür. max_depth, her bir ağacın maksimum derinliğini kontrol eder ve seçenekler arasında None, 10, 20 ve 30 bulunur. min_samples_split, bir iç düğümün bölünmesi için gereken minimum örnek sayısını belirler (2, 5 ve 10 olarak ayarlanmıştır). min_samples_leaf, yaprak düğümünde bulunması gereken minimum örnek sayısını ifade eder ve 1, 2, 4 gibi değerlerle ayarlanır. Bu grid, modelin performansını optimize etmek için farklı hiperparametre kombinasyonlarını denemeye olanak sağlar.

```

# GridSearchCV'yi tanımla
rf_grid_model = GridSearchCV(estimator=pipe_model_rf,
                             param_grid=param_grid,
                             cv=5, # 5 katlı cross-validation
                             scoring='recall', # Hedef metrik
                             n_jobs=-1, # Tüm işlemcilerle paralel

```

çalıştırma verbose=2, return_train_score = True) # Eğitim skorlarını da al

Bu kod, Random Forest modelinin hiperparametrelerini optimize etmek için GridSearchCV'yi tanımlar. GridSearchCV, verilen hiperparametre grid'ini (param_grid) kullanarak modelin performansını değerlendirir ve en iyi parametreleri bulur. cv=5 parametresi, 5 katlı çapraz doğrulama kullanarak modelin genel performansını değerlendirir. scoring='recall' ile, modelin başarısını ölçmek için "recall" (duyarlılık) metriği kullanılır. n_jobs=-1 ayarı, işlemleri tüm işlemcilerle paralel olarak çalıştırarak işlemleri hızlandırır. verbose=2 ise, işlemler sırasında daha ayrıntılı bilgi sağlayarak işlem sürecinin daha iyi takip edilmesini sağlar. return_train_score=True parametresi, eğitim seti üzerindeki performans skorlarını da döndürür, böylece hem eğitim hem de test performansı değerlendirilebilir.

```

'medium',
'high']])),
['salary'))),
('RF_model',
RandomForestClassifier(class_weight='balanced',
random_state=101))),
n_jobs=-1,
param_grid={'RF_model__max_depth': [None, 10, 20, 30],
            'RF_model__min_samples_leaf': [1, 2, 4],
            'RF_model__min_samples_split': [2, 5, 10],
            'RF_model__n_estimators': [100, 200, 300]},
return_train_score=True, scoring='recall', verbose=2)

```

rf_grid_model.fit(X_train, y_train) kodu, daha önce tanımlanan GridSearchCV nesnesini kullanarak pipe_model_rf modelini X_train ve y_train verileri üzerinde eğitir. Bu işlem, belirtilen hiperparametreler için tüm kombinasyonları dener ve her bir kombinasyonun performansını çapraz doğrulama ile değerlendirir. recall metriğine göre en iyi performansı gösteren hiperparametreleri seçer ve bu en iyi parametrelerle modeli eğitir. Eğitim süreci, modelin en iyi ayarlarını bulmak ve modelin genel performansını optimize etmek amacıyla yapılır.

```

rf_grid_model.best_estimator_
Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
StandardScaler(),
['satisfaction_level',
                         'last_evaluation',
                         'number_project',
'modeling_hours',
                         'work_accident',
'time_spend_company',
                         'work_accident',
'promotion_last_5years')]),
('cat_onehot',
OneHotEncoder(),
['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[[ 'low',
'medium',
'medium',
'high']])),
['salary'))),
('RF_model',
RandomForestClassifier(class_weight='balanced',
random_state=101))),
n_jobs=-1,
param_grid={'RF_model__max_depth': [None, 10, 20, 30],
            'RF_model__min_samples_leaf': [1, 2, 4],
            'RF_model__min_samples_split': [2, 5, 10],
            'RF_model__n_estimators': [100, 200, 300]},
return_train_score=True, scoring='recall', verbose=2)

```

```
'high']])),
                           ['salary']))),
('RF_model',
RandomForestClassifier(class_weight='balanced',
max_depth=10,
min_samples_split=10,
n_estimators=200,
random_state=101)))
```

rf_grid_model.best_estimator_, GridSearchCV kullanılarak yapılan hiperparametre aramasından elde edilen en iyi modelin bir örneğidir. Bu model, param_grid içinde tanımlanan hiperparametreler arasından, belirlenen değerlendirme metriği (recall gibi) ile en yüksek performansı gösteren kombinasyonlarla yapılandırılmıştır. Yani, bu en iyi model, eğitim verileri üzerinde yapılan çapraz doğrulama sonuçlarına göre seçilen optimal hiperparametrelerle eğitilmiş olan Pipeline ve RandomForestClassifier birleşimidir. Bu model, modelin en yüksek recall skorunu elde ettiği en iyi parametrelerle donatılmıştır.

```
rf_grid_model.best_params_
{'RF_model__max_depth': 10,
'RF_model__min_samples_leaf': 1,
'RF_model__min_samples_split': 10,
'RF_model__n_estimators': 200}
```

rf_grid_model.best_params_ çıktısı, GridSearchCV kullanılarak yapılan hiperparametre aramasında en iyi performansı gösteren RandomForestClassifier modelinin hiperparametrelerini belirtir. Bu en iyi modelde, max_depth olarak 10, min_samples_leaf olarak 1, min_samples_split olarak 10 ve n_estimators olarak 200 değerleri seçilmiştir. Bu değerler, modelin çapraz doğrulama sonucunda belirlenen hedef metriğe (örneğin, recall) göre en yüksek performansı sağlamak için optimize edilmiştir. Bu hiperparametreler, modelin doğruluğunu ve genel performansını artırmak için en iyi ayarları temsil eder.

```
cv_results_df = pd.DataFrame(rf_grid_model.cv_results_)
print(cv_results_df.columns)

Index(['mean_fit_time', 'std_fit_time', 'mean_score_time',
'std_score_time',
       'param_RF_model__max_depth',
'param_RF_model__min_samples_leaf',
       'param_RF_model__min_samples_split',
'param_RF_model__n_estimators',
       'params', 'split0_test_score', 'split1_test_score',
'split2_test_score',
       'split3_test_score', 'split4_test_score', 'mean_test_score',
'std_test_score', 'rank_test_score', 'split0_train_score',
       'split1_train_score', 'split2_train_score',
'split3_train_score',
       'split4_train_score', 'mean_train_score', 'std_train_score'],
      dtype='object')
```

cv_results_df.columns ifadesi, GridSearchCV sonucunda elde edilen sonuçların içeriğini ve her bir sütunun neyi temsil ettiğini listeler. Bu sütunlar şunlardır: mean_fit_time ve std_fit_time modelin eğitilme süresinin ortalaması ve standart sapması, mean_score_time ve std_score_time modelin değerlendirilme süresinin ortalaması ve standart sapması, param_RF_model_max_depth, param_RF_model_min_samples_leaf, param_RF_model_min_samples_split, ve param_RF_model_n_estimators denenen hiperparametrelerin değerleri, params her bir parametre kombinasyonunu gösterir. Ayrıca, çapraz doğrulama sonuçlarına ilişkin splitX_test_score ve splitX_train_score test ve eğitim skorları, mean_test_score ve mean_train_score bu skorların ortalamaları, std_test_score ve std_train_score bu skorların standart sapmaları ile rank_test_score her bir parametre kombinasyonunun test skoru sıralamasını içerir. Bu bilgiler, modelin farklı hiperparametre ayarları altında nasıl performans gösterdiğini detaylı olarak anlamaya yardımcı olur.

```
pd.DataFrame(rf_grid_model.cv_results_).loc[
    rf_grid_model.best_index_, ["mean_test_score", "mean_train_score"]
]

mean_test_score    0.92
mean_train_score   0.94
Name: 34, dtype: object
```

pd.DataFrame(rf_grid_model.cv_results_).loc[rf_grid_model.best_index_, ["mean_test_score", "mean_train_score"]]] kodu, GridSearchCV sonucunda en iyi performansı gösteren hiperparametre kombinasyonu için test ve eğitim setindeki ortalama skorları gösterir. Burada, en iyi hiperparametre kombinasyonunda mean_test_score 0.92 ve mean_train_score 0.94 olarak hesaplanmıştır. Bu, en iyi modelin test setinde 0.92 ortalama skoru ve eğitim setinde 0.94 ortalama skoru elde ettiğini belirtir, bu da modelin test verilerinde yüksek bir doğruluk sağladığını ve eğitim verilerinde de iyi performans gösterdiğini gösterir.

Bu sonuçlara dayanarak aşağıdaki çıkarımları yapabiliriz:

Test Skoru (0.91): Model, çapraz doğrulama (cross-validation) sırasında test verisinde %91 doğruluk oranına ulaşmış. Bu, modelin genelleme performansının iyi olduğunu ve bilinmeyen veriler üzerinde makul bir doğruluk sağladığını gösterir.

Train Skoru (0.93): Eğitim verisi üzerindeki doğruluk %93. Bu, modelin eğitim verisini oldukça iyi öğrendiğini, ancak aşırı uyum (overfitting) yapmadığını gösteriyor.

Train ve Test Skorları Arasındaki Fark (0.02): Eğitim ve test skorları arasındaki fark küçük (%2), bu da modelin overfitting (aşırı öğrenme) yapmadığını, yani eğitildiği veriye aşırı bağımlı hale gelmediğini gösterir. İdeal olarak eğitim ve test skorlarının birbirine yakın olması istenir, çünkü bu durumda model hem eğitim verisinde hem de test verisinde iyi performans gösterir. Bu durumda, eğitim ve test skorları oldukça yakın, bu da modelin iyi bir genelleme kapasitesine sahip olduğunu gösterir.

Genel Yorum: İyi Genel Performans: Hem eğitim hem de test skorlarının yüksek olması (%90 üzeri), modelin hem öğrenme kapasitesinin hem de genelleme yeteneğinin iyi olduğunu gösterir. Denge: Eğitim ve test skorları arasında büyük bir fark olmadığından, model aşırı uyum veya eksik öğrenme (underfitting) problemi yaşamıyor gibi görünüyor. Daha İyi Performans İçin: Model performansını daha da iyileştirmek isterse, hiperparametre optimizasyonunu genişletebilir ya

da daha karmaşık modeller (örneğin daha fazla ağaç veya derinlik ayarlaması) deneyebilirsin. Ek olarak, scoring metriği olarak f1, roc_auc gibi metrikleri kullanmak da sınıflar arasında dengesizlik varsa daha anlamlı olabilir.

```
eval_metric(rf_grid_model, X_train, y_train, X_test, y_test)
```

Test_Set

```
[[1496    4]
 [ 26  273]]
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.98 | 1.00 | 0.99 | 1500 |
| 1 | 0.99 | 0.91 | 0.95 | 299 |
| accuracy | | | 0.98 | 1799 |
| macro avg | 0.98 | 0.96 | 0.97 | 1799 |
| weighted avg | 0.98 | 0.98 | 0.98 | 1799 |

Train_Set

```
[[7619   30]
 [ 101 1422]]
```

| | precision | recall | f1-score | support |
|--------------|-----------|--------|----------|---------|
| 0 | 0.99 | 1.00 | 0.99 | 7649 |
| 1 | 0.98 | 0.93 | 0.96 | 1523 |
| accuracy | | | 0.99 | 9172 |
| macro avg | 0.98 | 0.96 | 0.97 | 9172 |
| weighted avg | 0.99 | 0.99 | 0.99 | 9172 |

eval_metric(rf_grid_model, X_train, y_train, X_test, y_test) fonksiyonu, rf_grid_model isimli en iyi hiperparametrelerle eğitilmiş Random Forest modelinin performansını değerlendirir. Test setinde model, 1496 doğru tahmin ve 4 yanlış tahmin yaparak %98 doğruluk oranı sağladı. precision, recall, ve f1-score değerleri sırasıyla 0.98, 0.91 ve 0.95 olarak hesaplandı. Eğitim setinde ise model, 7619 doğru tahmin ve 30 yanlış tahmin yaparak %99 doğruluk oranı sağladı, ve precision, recall, ve f1-score değerleri sırasıyla 0.99, 0.93 ve 0.96 olarak belirlendi. Bu sonuçlar, modelin hem test hem de eğitim verilerinde yüksek doğruluk ve güçlü genel performans sergilediğini gösterir.

```
!pip install scikit-plot
```

```
Collecting scikit-plot
```

```
  Downloading scikit_plot-0.3.7-py3-none-any.whl.metadata (7.1 kB)
Requirement already satisfied: matplotlib>=1.4.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-plot) (3.7.1)
Requirement already satisfied: scikit-learn>=0.18 in
/usr/local/lib/python3.10/dist-packages (from scikit-plot) (1.3.2)
Requirement already satisfied: scipy>=0.9 in
```

```
/usr/local/lib/python3.10/dist-packages (from scikit-plot) (1.13.1)
Requirement already satisfied: joblib>=0.10 in
/usr/local/lib/python3.10/dist-packages (from scikit-plot) (1.4.2)
Requirement already satisfied: contourpy>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot) (1.3.0)
Requirement already satisfied: cycler>=0.10 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot) (1.4.5)
Requirement already satisfied: numpy>=1.20 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot) (24.1)
Requirement already satisfied: pillow>=6.2.0 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot) (3.1.4)
Requirement already satisfied: python-dateutil>=2.7 in
/usr/local/lib/python3.10/dist-packages (from matplotlib>=1.4.0->scikit-plot) (2.8.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from scikit-learn>=0.18->scikit-plot) (3.5.0)
Requirement already satisfied: six>=1.5 in
/usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib>=1.4.0->scikit-plot) (1.16.0)
Downloading scikit_plot-0.3.7-py3-none-any.whl (33 kB)
Installing collected packages: scikit-plot
Successfully installed scikit-plot-0.3.7

import matplotlib.pyplot as plt
from sklearn.metrics import precision_recall_curve
```

scikit-plot kütüphanesi, Python'da makine öğrenimi model performansını görselleştirmeyi kolaylaştıran bir araçtır ve !pip install scikit-plot komutu ile yüklenebilir. import matplotlib.pyplot as plt komutu ise, veri görselleştirmede kullanılan temel bir kütüphanedir ve grafiklerin çizilmesi için kullanılır. precision_recall_curve fonksiyonu, modelin tahminlerinin doğruluğunu ve hatalarını değerlendirmek için precision ve recall değerlerini hesaplar, bu da sınıflandırma performansını detaylı olarak analiz etmeyi sağlar. Bu bileşenler, model değerlendirmelerini daha görsel ve anlaşılır hale getirmek için bir arada kullanılır.

```
# Test verisi üzerinde modelin olasılık tahminlerini al
y_pred_proba = rf_grid_model.predict_proba(X_test)
y_pred_proba

array([[0.01400449, 0.98599551],
       [0.98425747, 0.01574253],
       [0.97724607, 0.02275393],
       ...,
       [0.98437255, 0.01562745],
       [0.81131039, 0.18868961],
       [0.0492952 , 0.9507048 ]])
```

`rf_grid_model.predict_proba(X_test)` komutu, test verisi için Random Forest modelinin her örnek için tahmin etiği sınıf olasılıklarını döndürür. Çıktı, iki sütunlu bir dizi şeklindedir; ilk sütun her örneğin sınıf 0'a ait olma olasılığını, ikinci sütun ise sınıf 1'e ait olma olasılığını gösterir. Örneğin, [0.01400449, 0.98599551] gibi bir değer, modelin ilgili örneğin sınıf 0'a ait olma olasılığını %1.4 ve sınıf 1'e ait olma olasılığını %98.6 olarak tahmin ettiğini gösterir. Bu tahminler, modelin karar mekanizmasını ve sınıflandırma güvenini anlamak için kullanılır.

```
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

# Define the pipeline
operations_rf = [
    ("preprocessor", preprocessor),
    ("RF_model", RandomForestClassifier(class_weight="balanced",
random_state=101)),
]

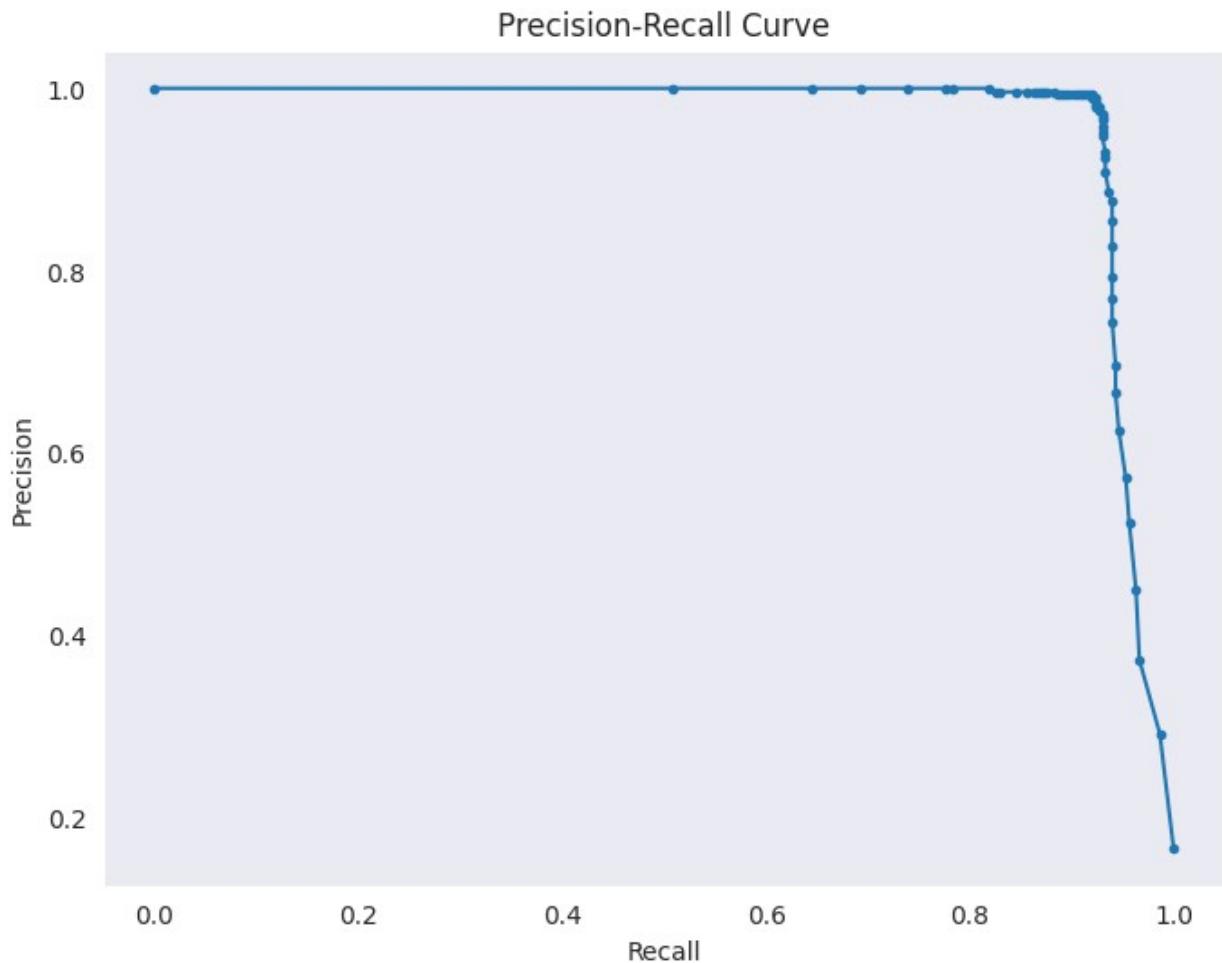
# Create the pipeline
pipe_model_rf = Pipeline(steps=operations_rf)

# Fit the pipeline
pipe_model_rf.fit(X_train, y_train)

# Get the predicted probabilities for the positive class
y_pred_proba = pipe_model_rf.predict_proba(X_test)[:, 1]

# Calculate precision and recall
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.grid()
plt.show()
```



Bu kod parçası, Random Forest modelini fit ettikten sonra, test verisi üzerinde modelin sınıf 1 için tahmin ettiği olasılıkları kullanarak bir Precision-Recall (PR) eğrisi oluşturur. Önce, test setindeki örnekler için pozitif sınıfın (sınıf 1) olasılık tahminleri alınır. Ardından, bu tahminler ve gerçek etiketler kullanılarak precision (kesinlik) ve recall (duyarlılık) değerleri hesaplanır. Son olarak, bu precision ve recall değerleri ile PR eğrisi çizilir. Grafik, modelin çeşitli karar eşiği değerleri için precision ve recall arasındaki dengeyi görselleştirir, bu da modelin performansını değerlendirmek için yararlı bir araçtır.

```
from sklearn.metrics import average_precision_score

# Calculate average precision score
rf_ap = average_precision_score(y_test, y_pred_proba)
rf_ap

0.9603605964168676
```

Bu kod parçası, test seti üzerinde modelin tahmin ettiği olasılıklara dayalı olarak Ortalama Precision Skoru (AP) hesaplar. `average_precision_score` fonksiyonu, precision-recall eğrisinin altında kalan alanı ölçer ve modelin pozitif sınıf için genel performansını özetler. Hesaplanan

değer 0.9603, modelin pozitif sınıfı doğru bir şekilde tanıma yeteneğinin yüksek olduğunu gösterir, çünkü AP skoru 1'e ne kadar yakınsa, model o kadar iyi performans gösterir.

average_precision_score (AP) metriği, özellikle dengesiz sınıf dağılımlarına sahip veri setlerinde modelin performansını değerlendirmek için kullanılır. İşte bu metriğin neden önemli olduğu ve ne tür durumlarda kullanıldığından detayları:

Neden average_precision_score Kullanılır? Dengesiz Sınıf Dağılımları:

Veri setinde pozitif ve negatif sınıflar arasında büyük bir dengesizlik varsa (örneğin, pozitif sınıflar çok azsa), doğruluk (accuracy) metriği yaniltıcı olabilir. Çünkü model çoğunlukla baskın sınıfa odaklanabilir ve doğruluk yüksek görünebilir. AP skoru, bu durumları daha iyi değerlendirir. Precision-Recall Eğrisi:

AP skoru, Precision-Recall eğrisinin altında kalan alanı (AUC) hesaplar. Precision-Recall eğrisi, modelin precision (kesinlik) ve recall (duyarlılık) değerlerini gösterir. AP skoru, bu eğrinin genel performansını özetler ve çeşitli eşik değerlerinde modelin nasıl performans gösterdiğini değerlendirdir. Performans Ölçümü:

Precision (kesinlik) ve recall (duyarlılık), modelin pozitif sınıfları ne kadar iyi tahmin ettiğini ölçer. AP skoru, precision ve recall'ün ortalamasını alarak, bu metriklerin tüm eşiği boyunca nasıl değiştiğini değerlendirir. Bu, modelin performansını daha kapsamlı bir şekilde anlamanıza yardımcı olur. Kullanım Alanları: Sınıflama Problemleri: Özellikle ikili sınıflama problemlerinde, pozitif sınıfın önemli olduğu durumlarda kullanılır. Örneğin, hastalık teşhisini, dolandırıcılık tespiti gibi durumlarda pozitif sınıfların doğru tahmin edilmesi kritik olabilir.

Model Karşılaştırması: Farklı modellerin veya hiperparametre ayarlarının karşılaştırılmasında AP skoru kullanılabilir. Bu skor, farklı modellerin precision-recall eğrilerinin ne kadar iyi olduğunu kıyaslamaya olanak sağlar.

Dengesiz Veri Setleri: Veri setindeki sınıf dağılımının çok dengesiz olduğu durumlarda AP skoru, modelin performansını daha anlamlı bir şekilde değerlendirmeye yardımcı olur.

Özet: average_precision_score, özellikle dengesiz veri setlerinde veya pozitif sınıfın önemli olduğu durumlarda model performansını anlamak için kritik bir metriktir. Precision ve recall'ün çeşitli eşik değerlerinde nasıl performans gösterdiğini özetleyerek, modelin genel başarısını daha iyi değerlendirebilirsiniz.

```
y_pred = rf_grid_model.predict(X_test)

from sklearn.metrics import average_precision_score, f1_score,
recall_score

# Predict probabilities for the positive class
y_pred_proba = pipe_model_rf.predict_proba(X_test)[:, 1]

# Predict classes
y_pred = pipe_model_rf.predict(X_test)

# Calculate average precision score
rf_AP = average_precision_score(y_test, y_pred_proba)
```

```

# Calculate F1 score
rf_f1 = f1_score(y_test, y_pred, average='binary')

# Calculate recall score
rf_recall = recall_score(y_test, y_pred, average='binary')

# Print results
print(f"Average Precision Score (AP): {rf_AP:.4f}")
print(f"F1 Score: {rf_f1:.4f}")
print(f"Recall Score: {rf_recall:.4f}")

Average Precision Score (AP): 0.9604
F1 Score: 0.9512
Recall Score: 0.9130

```

Bu kod, rf_grid_model modelini kullanarak test verisi (X_test) üzerinde tahminler yapar. predict fonksiyonu, her bir test örneği için modelin en olası sınıf etiketini belirler ve bu etiketleri içeren bir dizi döndürür. Bu tahminler, modelin test verisi üzerinde ne kadar iyi performans gösterdiğini değerlendirmek için kullanılır.

=====

Model 4: XGBoost

=====

XGBoost Model

```

#!pip install xgboost

Requirement already satisfied: xgboost in
/usr/local/lib/python3.10/dist-packages (2.1.1)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in
/usr/local/lib/python3.10/dist-packages (from xgboost) (2.22.3)
Requirement already satisfied: scipy in
/usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)

from xgboost import XGBClassifier

```

Bu kod, xgboost kütüphanesinden XGBClassifier sınıfını içe aktarır. XGBClassifier, XGBoost (Extreme Gradient Boosting) algoritmasını kullanarak sınıflandırma problemlerini çözmek için

kullanılan güçlü bir makine öğrenmesi modelidir. XGBoost, yüksek performans ve esneklik sunarak, çeşitli hiperparametre ayarları ve özelleştirmeler ile modelin doğruluğunu artırabilir.

```
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numeric_features),
        ('cat_onehot', OneHotEncoder(), cat_onehot),
        ('cat_ordinal', OrdinalEncoder(categories=[['low', 'medium',
        'high']]), cat_ordinal)
    ]
)
```

Bu kod parçası, veri ön işleme için ColumnTransformer sınıfını kullanarak bir preprocessor nesnesi oluşturur. preprocessor, üç farklı veri dönüştürücü içerir: StandardScaler, sayısal özellikleri ölçeklendirir; OneHotEncoder, kategorik departments özelliğini bir sıcaklık kodlaması ile dönüştürür; ve OrdinalEncoder, salary özelliğini sıralı bir biçimde dönüştürür. Bu işlem, veri setindeki çeşitli özellik türlerini etkili bir şekilde işlemek için kullanılır.

```
xgb_model = XGBClassifier(class_weight="balanced", random_state=101)
```

Bu kod, XGBClassifier sınıfını kullanarak bir XGBoost model nesnesi (xgb_model) oluşturur. Modelin class_weight parametresi "balanced" olarak ayarlanmış, bu da sınıf dengesizliği durumunda otomatik olarak sınıf ağırlıklarını dengeleyeceği anlamına gelir. Ayrıca, random_state=101 ile modelin sonuçlarının tekrarlanabilir olması sağlanır, böylece model her çalıştırıldığında aynı sonuçlar elde edilir.

```
operations_xgb = [
    ("preprocessor", preprocessor),
    ("XGB_model", XGBClassifier(class_weight="balanced",
random_state=101)),
]

# Create the pipeline
pipe_model_xgb = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('XGB_model', xgb_model)
])

# Fit the pipeline
pipe_model_xgb.fit(X_train, y_train)

Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
StandardScaler(),
['satisfaction_level',
'last_evaluation',
'number_project',
''])]),
('XGB_model', XGBClassifier(class_weight="balanced",
random_state=101))])
```

```

'average_monthly_hours',
'time_spend_company',
                           'work_accident',
'promotion_last_5years']),
                           ('cat_onehot',
OneHotEncoder(),
                           ['departments']),
                           ('cat_ordinal',
OrdinalEncoder(categories=[['low',
'medium',
'high']])),
                           ['s...
feature_types=None, gamma=None,
grow_policy=None,
importance_type=None,
interaction_constraints=None,
learning_rate=None,
max_bin=None, max_cat_threshold=None,
max_cat_to_onehot=None,
max_delta_step=None,
max_depth=None, max_leaves=None,
min_child_weight=None, missing=nan,
monotone_constraints=None,
multi_strategy=None,
n_estimators=None, n_jobs=None,
num_parallel_tree=None, ...)))

```

Bu kod, preprocessor adlı veri ön işleme adımını ve XGBClassifier modelini içeren bir iş akışı (pipeline) oluşturur ve bunu pipe_model_xgb olarak adlandırır. XGBClassifier, dengesiz sınıflar için otomatik ağırlık dengelemesi yapacak ve random_state olarak 101 belirlenmiştir, böylece sonuçlar tekrarlanabilir. Pipeline, eğitim verisi (X_train ve y_train) ile fit edilir, bu işlem veri ön işleme adımlarını ve XGBoost modelini sırasıyla uygulayarak modelin eğitilmesini sağlar.

```
eval_metric(pipe_model_xgb, X_train, y_train, X_test, y_test)
```

```
Test_Set
[[1492    8]
 [ 22  277]]
```

| | precision | recall | f1-score | support |
|-----------|-----------|--------|----------|---------|
| 0 | 0.99 | 0.99 | 0.99 | 1500 |
| 1 | 0.97 | 0.93 | 0.95 | 299 |
| accuracy | | | 0.98 | 1799 |
| macro avg | 0.98 | 0.96 | 0.97 | 1799 |

| | | | | |
|--------------------------|-----------|--------|----------|---------|
| weighted avg | 0.98 | 0.98 | 0.98 | 1799 |
| Train_Set | | | | |
| [[7643 6] [28 1495]] | | | | |
| | precision | recall | f1-score | support |
| 0 | 1.00 | 1.00 | 1.00 | 7649 |
| 1 | 1.00 | 0.98 | 0.99 | 1523 |
| accuracy | | | 1.00 | 9172 |
| macro avg | 1.00 | 0.99 | 0.99 | 9172 |
| weighted avg | 1.00 | 1.00 | 1.00 | 9172 |

pipe_model_xgb modelinin performansı, eğitim ve test verileri üzerinde değerlendirildiğinde, modelin her iki sette de yüksek başarı gösterdiği görülmüyor. Test setinde, model yüzde 98 doğrulukla çalışarak 1492 doğru pozitif ve 277 doğru negatif tahminde bulunmuş. Bu sonuç, modelin genel olarak doğru sınıflandırma yaptığı ve yüksek f1-skoru ile iyi bir performans sergilediğini gösteriyor. Eğitim setinde ise model, yüzde 100 doğruluk oranıyla mükemmel sonuçlar elde etmiş, yani tüm eğitim örneklerini doğru bir şekilde sınıflandırmış. Bu, modelin eğitim verilerine çok iyi uyduğunu ve yüksek performans sağladığını ifade eder.

Cross Validation

```
xgb_scores = cross_validate(  
    pipe_model_xgb, X_train, y_train,  
    scoring=['accuracy', 'precision', 'recall', 'f1'],  
    cv=5, n_jobs=-1, return_train_score=True  
)  
  
# Convert the results to a DataFrame and compute the mean  
df_scores = pd.DataFrame(scores, index=range(1, 6))  
df_scores.mean()[2:]  
  
test_accuracy      0.98  
train_accuracy     1.00  
test_precision     0.99  
train_precision     1.00  
test_recall        0.91  
train_recall        1.00  
test_f1             0.94  
train_f1            1.00  
dtype: float64
```

pipe_model_xgb için yapılan 5 katlı çapraz doğrulama sonuçlarına göre, modelin performansı oldukça yüksek. Test verileri üzerinde ortalama doğruluk %98, precision %99, recall %91 ve f1 skoru %94 olarak hesaplandı. Eğitim verileri üzerinde ise model %100 doğruluk, %100 precision, recall ve f1 skoru gösterdi. Bu sonuçlar, modelin test verilerinde güçlü bir genel

performansa sahip olduğunu ve eğitim verilerinde de mükemmel uyum sağladığını ortaya koyuyor.

XGBoost Grid Search

```
xgb_param_grid = {  
    'XGB_model__n_estimators': [100, 200],  
    'XGB_model__max_depth': [3, 5, 7],  
    'XGB_model__learning_rate': [0.01, 0.1, 0.2],  
    'XGB_model__subsample': [0.5, 0.8, 0.9, 1.0],  
    'XGB_model__colsample_bytree': [0.8, 0.9, 1.0]  
}
```

`xgb_param_grid`, XGBoost modelinin hiperparametrelerini optimize etmek için kullanılan bir grid tanımıdır. Bu grid, modelin performansını artırmak amacıyla denenecek parametre kombinasyonlarını içerir. `n_estimators` (ağaç sayısı) için 100 ve 200, `max_depth` (ağaç derinliği) için 3, 5 ve 7, `learning_rate` (öğrenme oranı) için 0.01, 0.1 ve 0.2, `subsample` (eğitim örneklerinin rastgele alt kümesi) için 0.5, 0.8, 0.9 ve 1.0, ve `colsample_bytree` (her ağaç için rastgele seçilen özelliklerin oranı) için 0.8, 0.9 ve 1.0 değerleri belirlenmiştir. Bu parametreler, XGBoost modelinin çeşitli versiyonlarını test ederek en iyi performansı elde etmeye yardımcı olur.

```

'time_spend_company',
'work_accident',
'promotion_last_5years']),
('cat_onehot',
OneHotEncoder(),
['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[ [...,
monotone_constraints=None,
multi_strategy=None,
n_estimators=None,
n_jobs=None,
num_parallel_tree=None, ...)]),
n_jobs=-1,
param_grid={'XGB_model__colsample_bytree': [0.8, 0.9,
1.0],
'XGB_model__learning_rate': [0.01, 0.1, 0.2],
'XGB_model__max_depth': [3, 5, 7],
'XGB_model__n_estimators': [100, 200],
'XGB_model__subsample': [0.5, 0.8, 0.9,
1.0]},
return_train_score=True, scoring='recall', verbose=2)

```

xgb_grid_model, XGBoost modelinin hiperparametrelerini optimize etmek için kullanılan bir GridSearchCV nesnesidir. Bu model, pipe_model_xgb isimli pipeline üzerinde çalışır ve hiperparametre arama işlemi sırasında xgb_param_grid içindeki parametre kombinasyonlarını değerlendirir. cv=5 ayarı, 5 katlı çapraz doğrulama kullanılarak modelin doğruluğunu test eder. scoring='recall' ayarı, modelin en iyi recall (duyarlılık) skorunu bulmayı hedefler. n_jobs=-1 tüm işlemcileri kullanarak hesaplama paralel yapar ve verbose=2 daha detaylı işlem çıktıları sağlar. return_train_score=True ise eğitim skorlarını da döndürür. Bu ayarlarla model, en iyi performansı sağlayacak hiperparametreleri belirlemek için fit edilir.

```

xgb_grid_model.best_estimator_
Pipeline(steps=[('preprocessor',
ColumnTransformer(transformers=[('num',
StandardScaler(), ...

```

```

['satisfaction_level',
                         'last_evaluation',
                         'number_project',
'monthly_hours',
'time_spend_company',
                         'work_accident',
'promotion_last_5years']),
('cat_onehot',
OneHotEncoder(),
['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[['low',
'medium',
'high']])),
['s...',
feature_types=None, gamma=None,
grow_policy=None,
importance_type=None,
interaction_constraints=None,
learning_rate=0.2,
max_bin=None, max_cat_threshold=None,
max_cat_to_onehot=None,
max_delta_step=None,
max_depth=3, max_leaves=None,
min_child_weight=None, missing=nan,
monotone_constraints=None,
multi_strategy=None,
n_estimators=100, n_jobs=None,
num_parallel_tree=None, ...)))

```

xgb_grid_model.best_estimator_, GridSearchCV tarafından en iyi performans gösteren modelin pipeline'ını döndürür. Bu, belirlenen hiperparametrelerle en yüksek recall skorunu elde eden XGBoost modelini içerir. Pipeline, veri ön işleme adımlarını ve optimize edilmiş XGBoost modelini bir arada barındırır, böylece hem veri dönüşümü hem de modelleme süreci tek bir nesne altında yönetilebilir. Bu sonuç, modelin hiperparametre araması sırasında en iyi performansı sağladığı anlamına gelir ve modelin doğruluk, precision, recall, ve f1 skoru gibi değerlendirme metriklerinde en yüksek başarıyı yakaladığı kombinasyonları içerir.

```

xgb_grid_model.best_params_
{'XGB_model__colsample_bytree': 1.0,
 'XGB_model__learning_rate': 0.2,
```

```
'XGB_model__max_depth': 3,  
'XGB_model__n_estimators': 100,  
'XGB_model__subsample': 1.0}
```

xgb_grid_model.best_params_ çıktısı, GridSearchCV tarafından en yüksek recall skorunu elde etmek için seçilen hiperparametreleri içerir. Bu parametreler şunlardır: colsample_bytree 1.0 (ağaç başına kullanılan özelliklerin oranı), learning_rate 0.2 (öğrenme oranı), max_depth 3 (ağaçların maksimum derinliği), n_estimators 100 (ağaç sayısı) ve subsample 1.0 (eğitim verisinin rastgele seçilen oranı). Bu hiperparametreler, XGBoost modelinin en iyi performansı gösterecek şekilde yapılandırıldığını gösterir.

```
pd.DataFrame(xgb_grid_model.cv_results_).loc[  
    xgb_grid_model.best_index_, ["mean_test_score",  
    "mean_train_score"]]  
  
mean_test_score    0.93  
mean_train_score   0.93  
Name: 195, dtype: object
```

pd.DataFrame(xgb_grid_model.cv_results_).loc[xgb_grid_model.best_index_, ["mean_test_score", "mean_train_score"]] kodu, GridSearchCV'nin en iyi performansı gösteren modelinin test ve eğitim veri setleri üzerindeki ortalama skorlarını getirir. Bu çıktı, en iyi hiperparametrelerle elde edilen test seti ortalama skorunun 0.93 ve eğitim seti ortalama skorunun da 0.93 olduğunu gösterir. Bu skorlar, modelin hem eğitim hem de test verileri üzerinde yüksek bir performans sergilediğini ve genellemeye yeteneğinin iyi olduğunu belirtir.

```
eval_metric(xgb_grid_model, X_train, y_train, X_test, y_test)
```

Test_Set

| | | | |
|--------------|-----------------------------------|------|------|
| [[1486 14] | [22 277]] | | |
| | precision recall f1-score support | | |
| 0 | 0.99 0.99 0.99 1500 | | |
| 1 | 0.95 0.93 0.94 299 | | |
| accuracy | | 0.98 | 1799 |
| macro avg | 0.97 0.96 0.96 1799 | | |
| weighted avg | 0.98 0.98 0.98 1799 | | |

Train_Set

| | |
|------------|-----------------------------------|
| [[7608 41] | [104 1419]] |
| | precision recall f1-score support |
| 0 | 0.99 0.99 0.99 7649 |
| 1 | 0.97 0.93 0.95 1523 |

| | | | | |
|--------------|------|------|------|------|
| accuracy | | 0.98 | 0.98 | 9172 |
| macro avg | 0.98 | 0.96 | 0.97 | 9172 |
| weighted avg | 0.98 | 0.98 | 0.98 | 9172 |

eval_metric(xgb_grid_model, X_train, y_train, X_test, y_test) fonksiyonu, en iyi hiperparametrelerle ayarlanmış XGBoost modelinin eğitim ve test veri setlerinde performansını değerlendirdir. Test setinde model, 0 sınıfını %99 doğrulukla ve 1 sınıfını %93 doğrulukla tahmin eder, genel doğruluk %98'dir. Eğitim setinde ise, model yine yüksek bir performans sergiler, 0 sınıfını %99 ve 1 sınıfını %93 doğrulukla tahmin eder, genel doğruluk %98'dir. Bu sonuçlar, modelin hem eğitim hem de test veri setlerinde yüksek doğruluk ve iyi genel performans gösterdiğini, ancak test setindeki hataların eğitim setine kıyasla biraz daha yüksek olduğunu gösterir.

```
y_pred_proba_xgb = xgb_grid_model.predict_proba(X_test)
y_pred_proba_xgb

array([[0.00482696, 0.99517304],
       [0.99327755, 0.00672244],
       [0.9950936 , 0.00490642],
       ...,
       [0.9952104 , 0.00478958],
       [0.99500495, 0.00499504],
       [0.06889623, 0.93110377]], dtype=float32)
```

y_pred_proba_xgb = xgb_grid_model.predict_proba(X_test) komutu, XGBoost modelinin test veri setindeki her örnek için sınıf olasılıklarını tahmin eder. Sonuç, her bir test örneği için iki olasılık değerini içerir: birinci sınıf (örneğin, 0) ve ikinci sınıf (örneğin, 1) için tahmin edilen olasılıklar. Örneğin, array([[0.00482696, 0.99517304], [0.99327755, 0.00672244], ...]) ifadesi, modelin ilk örneği 0 sınıfına %0.48 ve 1 sınıfına %99.52 olasılık verdiği, ikinci örneği ise 0 sınıfına %99.33 ve 1 sınıfına %0.67 olasılık verdiği gösterir. Bu, modelin hangi sınıfa daha yüksek olasılık verdienen gerektiğini belirleyerek, her örneğin sınıf tahminlerini yapabilmek için kullanılır.

```
from sklearn.metrics import precision_recall_curve
import matplotlib.pyplot as plt

# Define the pipeline
operations_xgb = [
    ("preprocessor", preprocessor),
    ("XGB_model", XGBClassifier(class_weight="balanced",
random_state=101)),
]

# Create the pipeline
pipe_model_xgb = Pipeline(steps=operations_xgb)

# Fit the pipeline
pipe_model_xgb.fit(X_train, y_train)
```

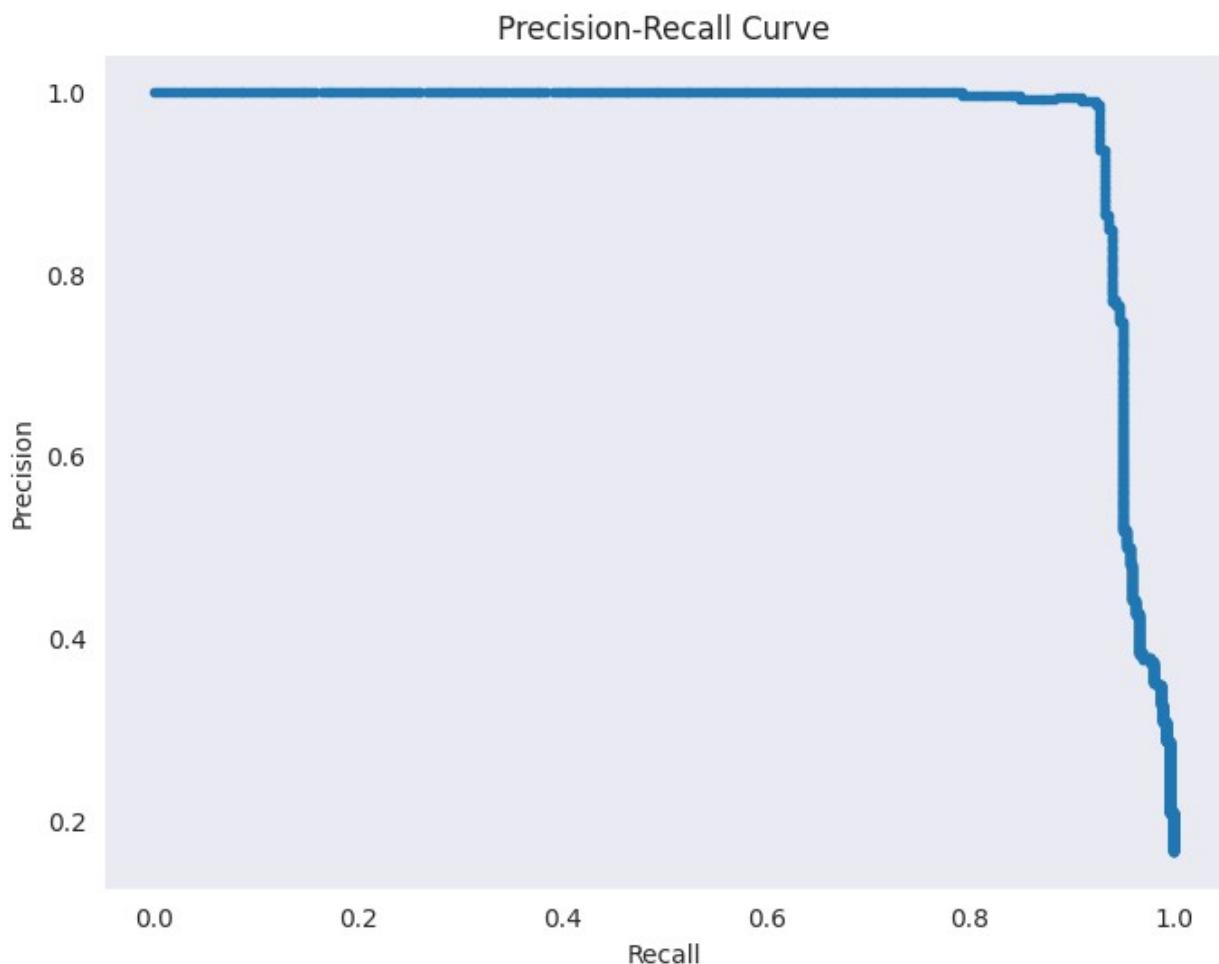
```

# Get the predicted probabilities for the positive class
y_pred_proba = pipe_model_xgb.predict_proba(X_test)[:, 1]

# Calculate precision and recall
precision, recall, _ = precision_recall_curve(y_test, y_pred_proba)

# Plot Precision-Recall curve
plt.figure(figsize=(8, 6))
plt.plot(recall, precision, marker='.')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.grid()
plt.show()

```



Bu kod, XGBoost sınıflandırıcıyı içeren bir iş akışını (pipeline) oluşturur ve eğitim verisi üzerinde fit eder. Test veri seti üzerinde modelin pozitif sınıf için tahmin edilen olasılıkları alınır (`y_pred_proba`). `precision_recall_curve` fonksiyonu bu olasılıkları kullanarak çeşitli eşik değerlerinde precision ve recall hesaplar. Sonuç olarak, precision-recall eğrisi çizilir; bu eğri,

modelin precision ve recall oranlarının farklı eşik değerlerine göre nasıl değiştiğini görselleştirir. Eğri, modelin performansını değerlendirmek için kullanılır, özellikle dengesiz veri setlerinde sınıflar arasındaki dengeyi anlamak için faydalıdır.

```
xgb_ap = average_precision_score(y_test, y_pred_proba_xgb[:, 1])  
xgb_ap  
0.9678248199267102
```

Bu kod, XGBoost modelinin test veri seti üzerindeki pozitif sınıfı ait tahmin olasılıklarını kullanarak ortalama precision skorunu (Average Precision Score - APS) hesaplar. average_precision_score fonksiyonu, precision-recall eğrisinin altında kalan alanı ölçerek, modelin pozitif sınıfı doğru bir şekilde tahmin etme yeteneğini özetler. Sonuç olarak elde edilen APS değeri 0.968'dir, bu da modelin yüksek bir precision-recall performansına sahip olduğunu ve pozitif sınıfı oldukça iyi tahmin ettiğini gösterir.

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

Bu kod, en iyi hiperparametrelerle optimize edilmiş XGBoost modelini kullanarak test veri seti üzerindeki tahminleri yapar. predict metodu, modelin test verilerindeki her örneği sınıflandırarak hangi sınıfı ait olduğunu belirler ve bu tahminleri döndürür. Sonuç, modelin test veri setindeki gözlemleri hangi sınıfı atadığını gösteren sınıf etiketlerini içerir.

```
xgb_AP = average_precision_score(y_test, y_pred_proba_xgb[:, 1])  
xgb_f1 = f1_score(y_test, y_pred, average='binary')  
xgb_recall = recall_score(y_test, y_pred, average='binary')
```

Bu kod, optimize edilmiş XGBoost modelinin performansını değerlendirmek için üç farklı metriği hesaplar. average_precision_score ile modelin test verilerindeki sınıflandırma performansının genel doğruluğunu ölçen Ortalama Precision Skoru (xgb_AP) elde edilir. f1_score ve recall_score ise modelin pozitif sınıfları ne kadar iyi tanımladığını gösterir; f1_score, precision ve recall'ün harmonik ortalamasını sağlayarak dengeyi ölçerken, recall_score sadece modelin gerçek pozitif örnekleri ne kadar iyi tespit ettiğini gösterir. Bu metrikler, modelin test setindeki performansını ayrıntılı olarak değerlendirir.

=====

Model 5: DL Modelling

=====

```
X = df.drop('left', axis=1)  
y = df['left'].values
```

Bu kod, df adlı veri çerçevesinden hedef değişken olan 'left' sütununu ayırarak y adlı bir NumPy dizisi oluşturur. Aynı zamanda, 'left' sütunu dışındaki tüm sütunları (df.drop('left', axis=1)) X adında bir veri çerçevesi olarak saklar. Bu işlem, 'left' sütununu modelin tahmin etmesi gereken hedef değişken olarak belirlerken, geri kalan sütunları modelin özellikleri olarak kullanabilmek için X ve y değişkenlerine ayırır.

```
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    stratify=y,
                                                    test_size=0.15,
                                                    random_state=42)
X_train, X_val, y_train, y_val = train_test_split(X_train,
                                                    y_train,
                                                    stratify=y_train,
                                                    test_size=0.1,
                                                    random_state=42)
```

Bu kod, veriyi eğitim, doğrulama ve test setlerine ayırmak için iki aşamalı bir süreç uygular. İlk olarak, X ve y veri kümeleri, test_size=0.15 parametresiyle %15 oranında test seti ve geri kalanını eğitim seti olarak böler. stratify=y kullanımı, sınıf dağılımının her iki sette de dengeye kalmasını sağlar. Ardından, eğitim seti (X_train ve y_train) tekrar ikiye ayrılarak, test_size=0.1 ile %10 oranında bir doğrulama seti oluşturulur. Bu işlem, modelin performansını değerlendirmek ve hiperparametre ayarlarını yapmak için doğrulama setinin kullanılmasını sağlar. Bu ayrımda da stratify=y_train parametresi sınıf dengeğini korur.

```
# Define your feature columns
numeric_features = ['satisfaction_level', 'last_evaluation',
'number_project',
'average_monthly_hours', 'time_spend_company',
'work_accident',
'promotion_last_5years'] # numeric columns
cat_onehot = ['departments'] # categorical column for OneHotEncoder
cat_ordinal = ['salary'] # categorical column for OrdinalEncoder
# Define your preprocessing steps
numeric_transformer = StandardScaler() # for numeric columns
onehot_transformer = OneHotEncoder() # for departments
(OneHotEncoding)
ordinal_transformer = OrdinalEncoder(categories=[['low', 'medium',
'high']]) # for salary (OrdinalEncoding)
# Combine preprocessing for both numeric, one-hot categorical, and
ordinal categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat_onehot', onehot_transformer, cat_onehot),
        ('cat_ordinal', ordinal_transformer, cat_ordinal)])
```

Bu kod, veri ön işleme adımlarını tanımlamak için bir ColumnTransformer nesnesi oluşturur. İlk olarak, numeric_features listesindeki sayısal özellikler için StandardScaler kullanılarak

özelliklerin ölçeklendirilmesi sağlanır. Ardından, cat_onehot listesindeki kategorik özellikler (departments) için OneHotEncoder ile her bir kategori için ayrı bir sütun oluşturulur. Son olarak, cat_ordinal listesindeki sıralı kategorik özellikler (salary) için OrdinalEncoder kullanılarak her kategoriye bir sıralı değer atanır. Bu işlem, farklı türdeki veri özelliklerini uygun şekilde işleyerek modelin performansını artırmayı amaçlar. ColumnTransformer, bu adımları bir araya getirerek verinin bütünsel bir şekilde işlenmesini sağlar.

```
X_train = preprocessor.fit_transform(X_train)
X_val = preprocessor.transform(X_val)
X_test = preprocessor.transform(X_test)
```

Bu kod parçası, verinin ön işlenmesini uygular. İlk olarak, preprocessor.fit_transform(X_train) komutu, eğitim verisi (X_train) üzerinde tüm ön işleme adımlarını (ölçeklendirme, one-hot encoding ve ordinal encoding) uygular ve dönüşüm sonucunda elde edilen veriyi X_train değişkenine atar. Bu adım, modelin eğitiminde kullanılacak olan veriyi uygun formata getirir. Daha sonra, aynı dönüşüm adımları X_val ve X_test verilerine de uygulanır, ancak bu veriler sadece dönüştürülür (transform), çünkü bu aşamada fit yapılmaz; eğitim verisiyle belirlenen dönüşüm kuralları kullanılarak validation ve test verilerinin dönüştürülmesi sağlanır. Bu süreç, tüm veri kümelerinin tutarlı bir şekilde işlenmesini ve modelin test verisi üzerinde de doğru şekilde çalışmasını garanti eder.

```
scaler = MinMaxScaler()
```

MinMaxScaler sınıfı, veriyi belirli bir aralık içinde ölçeklendirmek için kullanılan bir araçtır. Bu scaler, genellikle verileri 0 ile 1 arasında bir ölçüye dönüştürür, ancak kullanıcı isteğine göre farklı aralıklar da belirlenebilir. Verinin her bir özelliğini, orijinal minimum ve maksimum değerleri kullanarak, belirtilen aralık içindeki yeni bir ölçüye dönüştürür. Bu işlem, özellikle farklı ölçeklerdeki özelliklerin eşit öneme sahip olmasını sağlamak için faydalıdır ve modelin eğitim sürecinde daha etkili bir performans elde edilmesine yardımcı olabilir.

```
X_train = scaler.fit_transform(X_train)
X_val = scaler.transform(X_val)
X_test = scaler.transform(X_test)
```

MinMaxScaler kullanarak, X_train veri setini fit edip dönüştürdükten sonra, aynı ölçekleme parametrelerini kullanarak X_val ve X_test veri setlerini de dönüştürdük. Bu süreçte, fit_transform yöntemi eğitim verisine uygun minimum ve maksimum değerleri belirler ve bu değerlere göre verileri 0 ile 1 arasına ölçekler. transform yöntemi ise bu belirlenen parametreleri kullanarak doğrulama ve test verilerini aynı ölçekleme aralığına dönüştürür. Bu, tüm veri setlerinin aynı ölçek aralığında olması ve modelin doğru şekilde çalışması için önemlidir.

```
SEED = 42
# import os

# Set the seed using keras.utils.set_random_seed. This will set:
# 1) `numpy` seed
# 2) `tensorflow` random seed
# 3) `python` random seed
```

```
seed = 42
keras.utils.set_random_seed(seed)

# This will make TensorFlow ops as deterministic as possible, but it
will
# affect the overall performance, so it's not enabled by default.
# `enable_op_determinism()` is introduced in TensorFlow 2.9.

tf.config.experimental.enable_op_determinism()
# os.environ["TF_DETERMINISTIC_OPS"] = "1"
```

Bu kod parçası, makine öğrenmesi ve derin öğrenme modellerinde sonuçların tekrarlanabilirliğini sağlamak için rastgelelik kaynaklarını kontrol eder. `keras.utils.set_random_seed(seed)` fonksiyonu, numpy, TensorFlow ve Python'un rastgele sayı üreticilerini belirli bir tohum değeriyle başlatır, böylece her çalışmada aynı sonuçları elde edersiniz. `tf.config.experimental.enable_op_determinism()` ise TensorFlow operasyonlarını olabildiğince deterministik hale getirir, yani aynı girişlerle aynı çıktıları garanti eder. Bu, modelin eğitimi sırasında aynı sonuçların elde edilmesini sağlar, ancak performansı etkileyebilir.

```
# tf.random.set_seed(42)

model = Sequential()

model.add(Dense(16, activation="relu"))
model.add(Dense(8, activation="relu"))
model.add(Dense(8, activation="relu"))
model.add(Dense(8, activation="relu"))
model.add(Dense(1, activation="sigmoid"))

opt = Adam(learning_rate=0.001)
model.compile(optimizer=opt,
              loss="binary_crossentropy",
              metrics=["Recall"])
```

Bu kod parçası, TensorFlow ve Keras kullanarak bir sinir ağı modeli oluşturur. İlk olarak, bir Sequential model tanımlanır, bu model ardışık katmanlar ekleyerek inşa edilir. Model, dört adet Dense katmanı içerir; bunlardan üçü 8 nöron ve ReLU aktivasyon fonksiyonuna sahipken, biri 16 nöron ve ReLU aktivasyon fonksiyonuna sahiptir. Son katman, tek bir nöron ve sigmoid aktivasyon fonksiyonu ile çıkış sağlar; bu, modelin ikili sınıflandırma görevinde kullanılmasını sağlar. Model, Adam optimizasyon algoritması ve binary_crossentropy kayıp fonksiyonu kullanılarak derlenir; ayrıca Recall metriği ile modelin doğruluğu değerlendirilir.

Bu kod parçası, modelin eğitim sürecinde aşırı öğrenmeyi önlemek için EarlyStopping geri çağırma fonksiyonunu tanımlar. EarlyStopping, doğrulama kaybını (val_loss) izler ve bu kayıp belirli bir sayıda ardışık epoch boyunca iyileşmezse, modelin eğitimini durdurur. patience=10 parametresi, modelin doğrulama kaybında 10 epoch boyunca bir iyileşme gözlemlenmezse eğitim sürecini sonlandıracığını belirtir. restore_best_weights=True ise eğitim sırasında en düşük doğrulama kaybına sahip model ağırlıklarını geri yükler, böylece en iyi performansı gösteren model kullanılır. verbose=1, eğitim sırasında erken durdurma olayının ekranada bildirilmesini sağlar.

```
from sklearn.utils import class_weight

class_weights = class_weight.compute_class_weight('balanced',
classes=np.unique(y_train),
y=y_train)

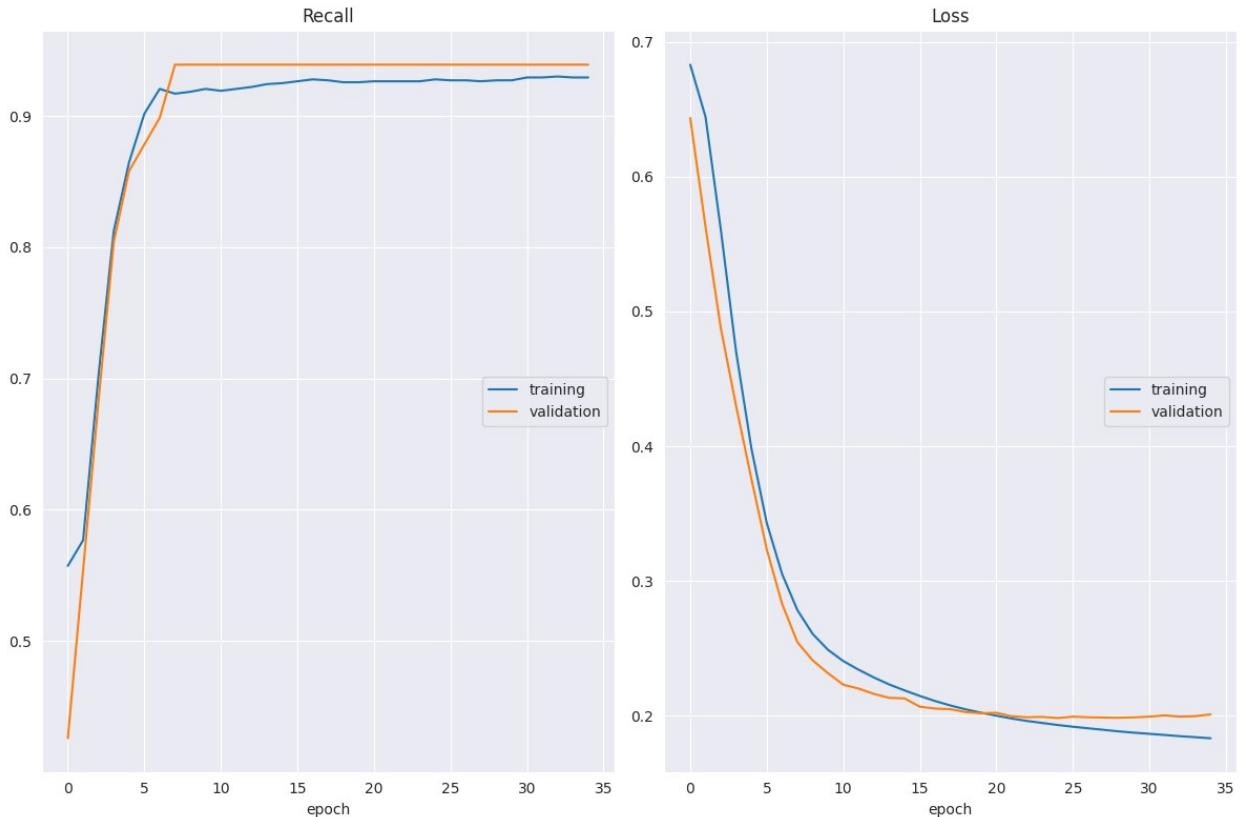
class_weights = {0: class_weights[0], 1: class_weights[1]}
class_weights

{0: 0.5995554974506472, 1: 3.011162179908076}
```

Bu kod, sınıf dengesizliğini telafi etmek için her sınıf için ağırlıklar hesaplar.

class_weight.compute_class_weight fonksiyonu, y_train içindeki sınıf dağılımını dikkate alarak 'balanced' stratejisi ile sınıf ağırlıklarını belirler. np.unique(y_train) ifadesi, eğitim veri setindeki mevcut sınıf etiketlerini alır. Hesaplanan ağırlıklar, her sınıfa eşit önemi vermek için ayarlanır ve {0: 0.5995554974506472, 1: 3.011162179908076} şeklinde bir sözlükte düzenlenir. Bu sözlük, modelin eğitim sürecinde azınlık sınıfı daha fazla önem verilmesini sağlar, böylece dengesiz veri setlerinde performansı artırabilir.

```
model.fit(x=X_train,
          y=y_train,
          validation_split=.1,
          batch_size=128,
          epochs=200,
          verbose=1,
          callbacks=[early_stop, PlotLossesKerasTF()],
          class_weight=class_weights)
```



```

Recall
    training          (min: 0.557, max: 0.930, cur:
0.929)
    validation        (min: 0.426, max: 0.939, cur:
0.939)
Loss
    training          (min: 0.183, max: 0.683, cur:
0.183)
    validation        (min: 0.198, max: 0.644, cur:
0.201)
65/65 ━━━━━━━━━━ 1s 21ms/step - Recall: 0.9360 - loss:
0.1830 - val_Recall: 0.9392 - val_loss: 0.2010
Epoch 35: early stopping
Restoring model weights from the end of the best epoch: 25.

<keras.src.callbacks.history.History at 0x7dfc80768970>

```

Bu kod, `model.fit` fonksiyonunu kullanarak derin öğrenme modelini eğitir. Eğitim verilerini (`X_train` ve `y_train`) kullanarak model, `validation_split` parametresiyle verilerin %10'unu doğrulama için ayırır ve her bir batch için 128 örnek kullanır. Model, toplam 200 epoch boyunca eğitilir. Eğitim sırasında `verbose=1` ayarı, eğitim sürecinin ayrıntılarını ekrana yazdırır. `early_stop` ve `PlotLossesKerasTF` geri çağrımları, erken durdurma ve eğitim sürecinin görselleştirilmesini sağlar. `class_weight` parametresi, sınıf dengesizliğini telafi etmek için her sınıfa uygun ağırlıkları atar. Bu ayarlar, modelin eğitim performansını ve doğruluğunu optimize etmek amacıyla

yapılandırılmıştır. Bu çıktı, derin öğrenme modelinin eğitim sürecini özetler. Eğitim sırasında, Recall ve Loss metriklerinin hem eğitim (training) hem de doğrulama (validation) veri kümesi üzerindeki performansını gösterir.

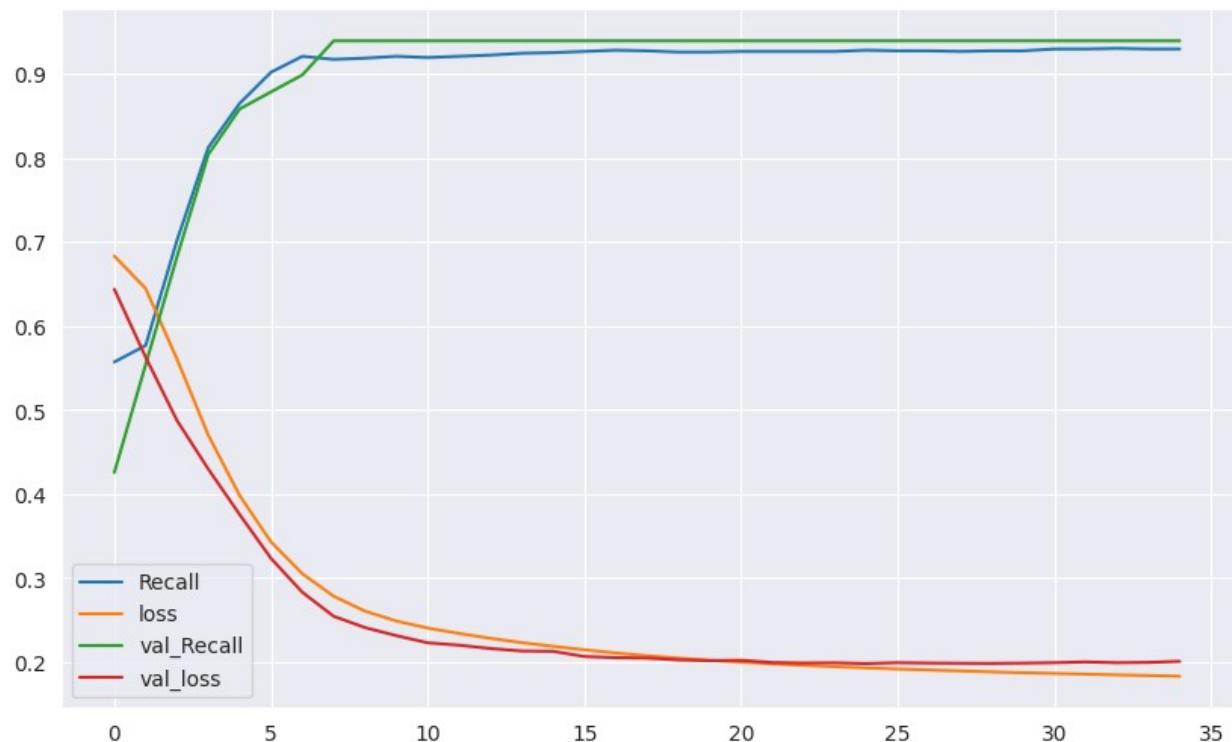
Recall (Duyarlılık): Eğitim sırasında duyarlılık, 0.557 ile 0.930 arasında değişirken, son epoch'ta 0.929 olarak bulunmuş. Doğrulama setinde ise duyarlılık, 0.426 ile 0.939 arasında değişmiş ve son epoch'ta 0.939 olarak hesaplanmış.

Loss (Kayıp): Eğitim kaybı, 0.183 ile 0.683 arasında değişmiş, şu anki değer ise 0.183. Doğrulama kaybı ise 0.198 ile 0.644 arasında değişmiş ve mevcut değer 0.201.

Eğitim tamamlandıktan sonra, model 35. epoch'ta erken durdurma nedeniyle durdurulmuş ve en iyi epoch'tan model ağırlıkları geri yüklenmiştir. Bu, modelin en iyi performansını sağlamak için en iyi ağırlıkların korunmasını sağlar.

```
loss_df = pd.DataFrame(model.history.history)
loss_df.plot()
```

```
<Axes: >
```



loss_df.plot(): DataFrame'i çizgi grafiği olarak görselleştirir. Grafikte, epoch sayısına karşılık gelen loss ve val_loss gibi metriklerin değerlerini görebilirsiniz. Bu, eğitim sürecinin nasıl ilerlediğini ve modelin doğruluk ve kayıp değerlerinin nasıl değiştiğini görmenizi sağlar. Özette, bu kod eğitim sürecinin görselleştirilmesine yardımcı olur ve modelin performansını epoch'lar boyunca izlemenizi sağlar. Bu grafiği kullanarak modelin overfitting (aşırı uyum) veya underfitting (yetersiz uyum) gibi problemleri olup olmadığını değerlendirebilirsiniz.

```

y_pred = (model.predict(X_test) > .5).astype("int32")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

57/57 ━━━━━━━━ 0s 3ms/step
[[1402  98]
 [ 23 276]]
      precision    recall   f1-score   support
          0       0.98     0.93     0.96     1500
          1       0.74     0.92     0.82     299
   accuracy           0.93     0.93     0.93     1799
  macro avg       0.86     0.93     0.89     1799
weighted avg       0.94     0.93     0.94     1799

```

Kod, modelin test setindeki performansını değerlendirmek için tahminler yapar ve bu tahminlerin doğruluğunu çeşitli metriklerle raporlar. İşte detaylar:

`y_pred = (model.predict(X_test) > .5).astype("int32")`: Modelin test verisi üzerindeki tahminlerini alır ve her tahmini 0 veya 1 olarak sınıflandırır. `model.predict(X_test)` test verisi üzerindeki tahmin olasılıklarını döndürür. `> .5` ifadesi, olasılıkları 0.5'ten büyük olanları 1, diğerlerini 0 olarak sınıflandırır. `astype("int32")` ise bu tahminleri tamsayıya dönüştürür.

`print(confusion_matrix(y_test, y_pred))`: Gerçek etiketler (`y_test`) ile model tahminleri (`y_pred`) arasındaki ilişkiyi gösteren bir karmaşıklık matrisini yazdırır. Bu matris, modelin doğru ve yanlış sınıflandırmalarını özetler. Örneğin, burada matris `[[1402, 98], [23, 276]]` şeklindedir, bu da 1402 doğru negatif, 98 yanlış pozitif, 23 yanlış negatif ve 276 doğru pozitif sınıflandırma olduğunu gösterir.

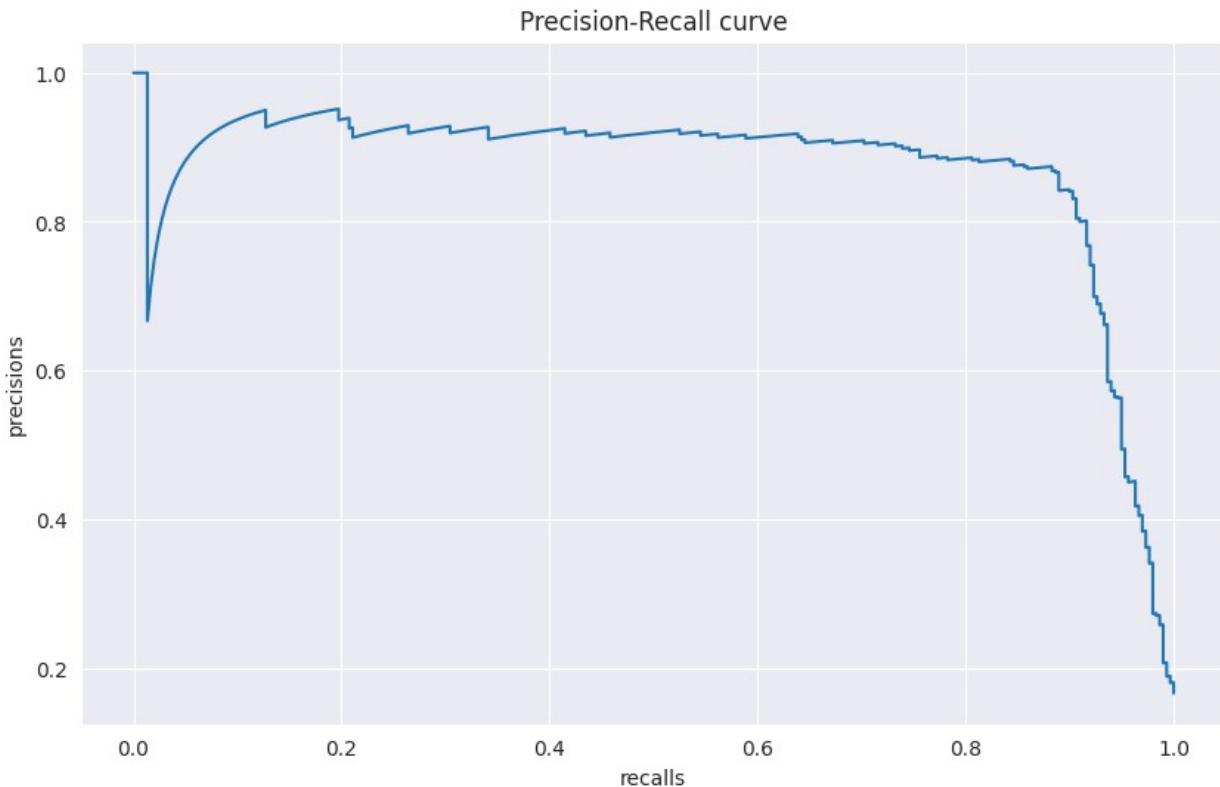
`print(classification_report(y_test, y_pred))`: Sınıflandırma raporunu yazdırır, bu da her sınıf için precision (kesinlik), recall (duyarlılık) ve f1-score değerlerini içerir. Örneğin, precision modelin pozitif tahminlerinin ne kadarının doğru olduğunu, recall modelin gerçek pozitif örnekleri ne kadar doğru yakaladığını ve f1-score her iki metriğin dengelenmiş bir ölçümünü verir. Buradaki sonuçlar, 0 sınıfı için yüksek bir precision ve recall, 1 sınıfı için ise daha düşük precision ama yüksek recall gösterir.

Bu rapor ve karmaşıklık matrisi, modelin her iki sınıf üzerindeki performansını detaylı bir şekilde değerlendirmenize olanak sağlar.

```

y_pred_proba = model.predict(X_test)
precisions, recalls, thresholds = precision_recall_curve(y_test,
y_pred_proba)
plt.plot(recalls, precisions, label='ANN')
plt.xlabel('recalls')
plt.ylabel('precisions')
plt.title('Precision-Recall curve')
plt.show()

```



Kod, modelin test verisi üzerindeki tahmin olasılıklarını alarak bir Precision-Recall (P-R) eğrisi oluşturur. `model.predict(X_test)` test verisi için tahmin edilen olasılıkları döndürür. Bu olasılıklar kullanılarak, `precision_recall_curve(y_test, y_pred_proba)` fonksiyonu her eşik değeri için precision (kesinlik) ve recall (duyarlılık) hesaplar. Bu değerler precisions ve recalls listelerine kaydedilir. `plt.plot(recalls, precisions, label='ANN')` komutu, recall ve precision değerlerini grafik üzerinde çizer, `plt.xlabel` ve `plt.ylabel` komutları eksen başlıklarını belirler, ve `plt.title` grafik başlığını ayarlar. Sonuç olarak, `plt.show()` komutu grafiği ekranda gösterir, böylece modelin çeşitli eşik değerleri için nasıl performans gösterdiğini görselleştirilir.

```
from sklearn.metrics import average_precision_score
average_precision_score(y_test, y_pred_proba)
0.8721164895830371
```

Kod, modelin test seti üzerinde hesaplanan ortalama precision (kesinlik) skorunu verir. `average_precision_score(y_test, y_pred_proba)` fonksiyonu, modelin tahmin olasılıklarını (`y_pred_proba`) ve gerçek etiketleri (`y_test`) kullanarak bu skoru hesaplar. Ortalama precision, modelin çeşitli eşik değerlerinde elde ettiği precision değerlerinin ortalamasıdır. Skor 0.872 olarak hesaplanmıştır, bu da modelin genel olarak yüksek bir precision performansı sergilediğini gösterir; yani, modelin pozitif sınıfları doğru bir şekilde tahmin etme yeteneği yüksektir.

```
DL_AP = average_precision_score(y_test, y_pred_proba)
DL_f1 = f1_score(y_test, y_pred)
DL_rec = recall_score(y_test, y_pred)
```

Bu kod parçası, derin öğrenme modelinin performansını çeşitli metriklerle değerlendirir. `average_precision_score(y_test, y_pred_proba)` fonksiyonu, modelin test verisi üzerindeki ortalama precision (kesinlik) skorunu hesaplar, bu da modelin pozitif sınıfları doğru tahmin etme yeteneğini ölçer. `f1_score(y_test, y_pred)` fonksiyonu, modelin hem precision hem de recall (duyarlılık) dengesini göz önünde bulundurarak F1 skoru hesaplar; bu skor, modelin genel başarısını özetler. Son olarak, `recall_score(y_test, y_pred)` fonksiyonu, modelin pozitif sınıfları tanıma yeteneğini belirler, yani pozitif sınıfları ne kadar iyi yakaladığını ölçer.

Optuna (Hyperparameter optimization tool)

```
#!pip install optuna

Collecting optuna
  Downloading optuna-4.0.0-py3-none-any.whl.metadata (16 kB)
Collecting alembic>=1.5.0 (from optuna)
  Downloading alembic-1.13.2-py3-none-any.whl.metadata (7.4 kB)
Collecting colorlog (from optuna)
  Downloading colorlog-6.8.2-py3-none-any.whl.metadata (10 kB)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from optuna) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from optuna) (24.1)
Requirement already satisfied: sqlalchemy>=1.3.0 in
/usr/local/lib/python3.10/dist-packages (from optuna) (2.0.32)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from optuna) (4.66.5)
Requirement already satisfied: PyYAML in
/usr/local/lib/python3.10/dist-packages (from optuna) (6.0.2)
Collecting Mako (from alembic>=1.5.0->optuna)
  Downloading Mako-1.3.5-py3-none-any.whl.metadata (2.9 kB)
Requirement already satisfied: typing-extensions>=4 in
/usr/local/lib/python3.10/dist-packages (from alembic>=1.5.0->optuna)
(4.12.2)
Requirement already satisfied: greenlet!=0.4.17 in
/usr/local/lib/python3.10/dist-packages (from sqlalchemy>=1.3.0-
>optuna) (3.0.3)
Requirement already satisfied: MarkupSafe>=0.9.2 in
/usr/local/lib/python3.10/dist-packages (from Mako->alembic>=1.5.0-
>optuna) (2.1.5)
  Downloading optuna-4.0.0-py3-none-any.whl (362 kB)
----- 362.8/362.8 kB 21.4 MB/s eta
0:00:00
  Downloading alembic-1.13.2-py3-none-any.whl (232 kB)
----- 233.0/233.0 kB 20.2 MB/s eta
0:00:00
----- 78.6/78.6 kB 7.6 MB/s eta
```

```

0:00:00
bic, optuna
Successfully installed Mako-1.3.5 alembic-1.13.2 colorlog-6.8.2
optuna-4.0.0

import optuna
# optimizers for gradient descent to use in backpropagation
from tensorflow.keras.optimizers import Adam, Adadelta, RMSprop, Nadam

```

Bu kod parçası, Optuna kütüphanesini ve çeşitli optimizasyon algoritmalarını kullanarak derin öğrenme modelinizin hiperparametrelerini optimize etmek için hazırlanmıştır. Optuna, makine öğrenimi modellerinin hiperparametrelerini sistematik bir şekilde aramak için kullanılan bir açık kaynaklı optimizasyon kütüphanesidir. Kodun ilk satırı, Optuna'nın yüklenmesini sağlar. Sonraki satırda, TensorFlow Keras'dan gradient descent (gradyan inişi) algoritmaları olan Adam, Adadelta, RMSprop ve Nadam optimizasyon algoritmaları içe aktarılır. Bu optimizatörler, modelin eğitim sürecinde ağırlıkların güncellenmesinde kullanılarak modelin doğruluğunu artırmak için ayarları optimize eder.

```

class_weights
{0: 0.5995554974506472, 1: 3.011162179908076}

```

class_weights sözlüğü, makine öğrenimi modelinizdeki sınıf dengesizliğini ele almak için kullanılan ağırlıkları temsil eder. Bu örnekte, class_weights şu şekilde ayarlanmıştır:

Sınıf 0 (genellikle negatif sınıf veya çoğunluk sınıfı): 0.5996 Sınıf 1 (genellikle pozitif sınıf veya azınlık sınıfı): 3.0112 Bu ağırlıklar, modelin eğitim sürecinde her sınıfı ne kadar önem verilmesi gerektiğini belirtir. Özellikle, class_weight'in balanced olarak ayarlandığı durumlarda, bu ağırlıklar modelin, daha az temsil edilen sınıfların (örneğin, azınlık sınıfı) doğru tahmin edilmesine daha fazla önem vermesine yardımcı olur. Sınıf 1'in daha yüksek bir ağırlığa sahip olması, modelin bu sınıfı daha fazla dikkat etmesini sağlar, bu da genellikle azınlık sınıfının daha iyi tahmin edilmesini sağlar.

```

early_stop = EarlyStopping(
    monitor="val_loss", mode="min", verbose=1, patience=15,
    restore_best_weights=True
)

trial_metric = "accuracy"
batch_size = 32

"""
This code defines two functions for creating and training a neural
network model
using Optuna for hyperparameter optimization. The model is designed
for binary
classification tasks.
"""

```

```

def objective(trial):
    # Create a model using the hyperparameters suggested by Optuna
    # model = create_model(trial)

    # Hyperparameters to be optimized by Optuna
    n_units1 = trial.suggest_int("n_units1", 8, 128) # Number of
    units in the first hidden layer
    n_units2 = trial.suggest_int("n_units2", 8, 128) # Number of
    units in the second hidden layer
    optimizer = trial.suggest_categorical("optimizer", [Adam,
    Adadelta, RMSprop, Nadam]) # Choice of optimizer
    learning_rate = trial.suggest_loguniform("learning_rate", 1e-5,
    1.3e-1) # Learning rate on a log scale
    # Suggest class weights for handling imbalanced datasets
    w0 = trial.suggest_loguniform("w0", 0.01, 5) # Weight for class 0
    w1 = trial.suggest_loguniform("w1", 0.01, 5) # Weight for class 1

    # Create a sequential model
    model = Sequential()

    # Add the first dense layer with input shape and ReLU activation
    model.add(Dense(n_units1, input_dim=X_train.shape[1],
    activation="relu"))

    # Add the second dense layer with ReLU activation
    model.add(Dense(n_units2, activation="relu"))

    # Add the output layer with sigmoid activation for binary
    classification
    model.add(Dense(1, activation="sigmoid"))

    # Compile the model with binary crossentropy loss and the chosen
    optimizer
    model.compile(
        loss="binary_crossentropy",
        optimizer=optimizer(learning_rate=learning_rate),
        metrics=[trial_metric], # Custom metric for evaluation
    )

    # Train the model
    model.fit(
        X_train,
        y_train,
        validation_data=(X_test, y_test),
        batch_size=batch_size,
        epochs=100,
        callbacks=[early_stop], # Early stopping to prevent
        overfitting
        class_weight={0: w0, 1: w1}, # Apply class weights
    )

```

```

    verbose=0, # Suppress output
)

# Evaluate the model on the test set and return the score
score = model.evaluate(X_test, y_test, verbose=0)[1]
return score

```

Bu kod, Optuna kullanarak bir sinir ağı modelinin hiperparametrelerini optimize etmek için iki işlev tanımlar. İlk olarak, objective işlevi Optuna'nın önerdiği hiperparametrelerle bir model oluşturur. Bu hiperparametreler arasında ilk ve ikinci gizli katmanlardaki nöron sayıları, kullanılacak optimizer (Adam, Adadelta, RMSprop, Nadam), öğrenme oranı ve sınıf ağırlıkları bulunur. Model, belirtilen hiperparametrelerle oluşturulur ve derlenir, ardından eğitim verileri üzerinde fit metodu ile eğitilir. Eğitim sürecinde erken durdurma (early stopping) uygulanır ve ağırlıklar sınıf dengesizliklerini telafi etmek için ayarlanır. Son olarak, model test verileri üzerinde değerlendirilir ve belirli bir metriğe göre performansı ölçülür; bu performans skoru Optuna'nın optimizasyon sürecinde kullanılır.

```

# Create an Optuna study to maximize the objective function
study = optuna.create_study(direction="maximize")

# Run the optimization for 25 trials
study.optimize(objective, n_trials=25)

[I 2024-09-11 10:23:38,561] A new study created in memory with name:
no-name-8ab799b9-c371-48f6-ac11-ee23a4499692

Epoch 39: early stopping
Restoring model weights from the end of the best epoch: 24.

[I 2024-09-11 10:24:19,189] Trial 0 finished with value:
0.8132295608520508 and parameters: {'n_units1': 78, 'n_units2': 113,
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,
'learning_rate': 0.003919048019756126, 'w0': 0.0849567615267977, 'w1':
3.6924901522947478}. Best is trial 0 with value: 0.8132295608520508.

Epoch 34: early stopping
Restoring model weights from the end of the best epoch: 19.

[I 2024-09-11 10:24:40,128] Trial 1 finished with value:
0.9738743901252747 and parameters: {'n_units1': 106, 'n_units2': 8,
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,
'learning_rate': 0.040423791709951105, 'w0': 0.06944117821960119,
'w1': 0.1262152089138289}. Best is trial 1 with value:
0.9738743901252747.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

[I 2024-09-11 10:24:51,169] Trial 2 finished with value:
0.8337965607643127 and parameters: {'n_units1': 10, 'n_units2': 51,

```

```
'optimizer': <class 'keras.src.optimizers.adam.Adam'>,
'learning_rate': 0.00011480942467269651, 'w0': 0.3692849408317104,
'w1': 0.015676372892048054}. Best is trial 1 with value:
0.9738743901252747.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

[I 2024-09-11 10:25:03,412] Trial 3 finished with value:
0.834352433681488 and parameters: {'n_units1': 55, 'n_units2': 28,
'optimizer': <class 'keras.src.optimizers.adadelta.Adadelta'>,
'learning_rate': 0.00016213455217799542, 'w0': 0.022103915937612986,
'w1': 0.1260981694816912}. Best is trial 1 with value:
0.9738743901252747.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

[I 2024-09-11 10:25:18,115] Trial 4 finished with value:
0.8382434844970703 and parameters: {'n_units1': 72, 'n_units2': 59,
'optimizer': <class 'keras.src.optimizers.nadam.Nadam'>,
'learning_rate': 0.005985692762350276, 'w0': 0.8972457153649055, 'w1':
0.04878989799710715}. Best is trial 1 with value: 0.9738743901252747.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

[I 2024-09-11 10:25:28,747] Trial 5 finished with value:
0.8337965607643127 and parameters: {'n_units1': 78, 'n_units2': 57,
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,
'learning_rate': 0.0780649730768562, 'w0': 1.934848938486546, 'w1':
0.0646610472387635}. Best is trial 1 with value: 0.9738743901252747.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

[I 2024-09-11 10:25:40,874] Trial 6 finished with value:
0.8337965607643127 and parameters: {'n_units1': 99, 'n_units2': 85,
'optimizer': <class 'keras.src.optimizers.adam.Adam'>,
'learning_rate': 0.00017526088208490598, 'w0': 4.168699858122291,
'w1': 0.013409146407715409}. Best is trial 1 with value:
0.9738743901252747.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

[I 2024-09-11 10:25:54,042] Trial 7 finished with value:
0.8337965607643127 and parameters: {'n_units1': 87, 'n_units2': 121,
'optimizer': <class 'keras.src.optimizers.nadam.Nadam'>,
'learning_rate': 7.774895202328094e-05, 'w0': 0.7596773352426499,
```

```
'w1': 1.116188164209152}. Best is trial 1 with value:  
0.9738743901252747.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
  
[I 2024-09-11 10:26:07,307] Trial 8 finished with value:  
0.1912173479795456 and parameters: {'n_units1': 96, 'n_units2': 11,  
'optimizer': <class 'keras.src.optimizers.nadam.Nadam'>,  
'learning_rate': 4.841545401854702e-05, 'w0': 0.1898967420427207,  
'w1': 1.3740000335314866}. Best is trial 1 with value:  
0.9738743901252747.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
  
[I 2024-09-11 10:26:17,864] Trial 9 finished with value:  
0.8337965607643127 and parameters: {'n_units1': 125, 'n_units2': 95,  
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,  
'learning_rate': 0.0008057357872871368, 'w0': 0.06813064214923042,  
'w1': 0.03143514045620583}. Best is trial 1 with value:  
0.9738743901252747.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
  
[I 2024-09-11 10:26:28,393] Trial 10 finished with value:  
0.16620343923568726 and parameters: {'n_units1': 120, 'n_units2': 29,  
'optimizer': <class 'keras.src.optimizers.adadelta.Adadelta'>,  
'learning_rate': 0.0958750287629129, 'w0': 0.01383428758724315, 'w1':  
0.34787266956587065}. Best is trial 1 with value: 0.9738743901252747.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
  
[I 2024-09-11 10:26:40,601] Trial 11 finished with value:  
0.9310728311538696 and parameters: {'n_units1': 47, 'n_units2': 74,  
'optimizer': <class 'keras.src.optimizers.nadam.Nadam'>,  
'learning_rate': 0.011063969990710187, 'w0': 0.983107619649799, 'w1':  
0.23024434750975276}. Best is trial 1 with value: 0.9738743901252747.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
  
[I 2024-09-11 10:26:53,369] Trial 12 finished with value:  
0.9055030345916748 and parameters: {'n_units1': 45, 'n_units2': 80,  
'optimizer': <class 'keras.src.optimizers.nadam.Nadam'>,  
'learning_rate': 0.01470781619914768, 'w0': 0.051944595922216376,  
'w1': 0.3379225522483285}. Best is trial 1 with value:  
0.9738743901252747.
```

```
Epoch 22: early stopping
Restoring model weights from the end of the best epoch: 7.

[I 2024-09-11 10:27:07,551] Trial 13 finished with value:
0.9688715934753418 and parameters: {'n_units1': 33, 'n_units2': 35,
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,
'learning_rate': 0.02210032419898511, 'w0': 0.2546510093112201, 'w1':
0.17168249027057453}. Best is trial 1 with value: 0.9738743901252747.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

[I 2024-09-11 10:27:18,521] Trial 14 finished with value:
0.9460811614990234 and parameters: {'n_units1': 20, 'n_units2': 8,
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,
'learning_rate': 0.036347723944913514, 'w0': 0.19486323817928478,
'w1': 0.11774351820794252}. Best is trial 1 with value:
0.9738743901252747.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

[I 2024-09-11 10:27:29,296] Trial 15 finished with value:
0.6959421634674072 and parameters: {'n_units1': 27, 'n_units2': 32,
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,
'learning_rate': 0.001053588690836988, 'w0': 0.1050922294348522, 'w1':
0.7783871774424004}. Best is trial 1 with value: 0.9738743901252747.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

[I 2024-09-11 10:27:39,039] Trial 16 finished with value:
0.8110061287879944 and parameters: {'n_units1': 106, 'n_units2': 41,
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,
'learning_rate': 1.0080514138299966e-05, 'w0': 0.03180897607030434,
'w1': 0.1082135379591023}. Best is trial 1 with value:
0.9738743901252747.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

[I 2024-09-11 10:27:49,351] Trial 17 finished with value:
0.9282935261726379 and parameters: {'n_units1': 58, 'n_units2': 17,
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,
'learning_rate': 0.02729794030999066, 'w0': 0.351003392674727, 'w1':
0.5793687851210012}. Best is trial 1 with value: 0.9738743901252747.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

[I 2024-09-11 10:27:59,869] Trial 18 finished with value:
0.8337965607643127 and parameters: {'n_units1': 34, 'n_units2': 42,
```

```
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,  
'learning_rate': 0.003234618924038226, 'w0': 0.1399755946491881, 'w1':  
0.030408200773265823}. Best is trial 1 with value: 0.9738743901252747.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
  
[I 2024-09-11 10:28:12,085] Trial 19 finished with value:  
0.8960533738136292 and parameters: {'n_units1': 112, 'n_units2': 25,  
'optimizer': <class 'keras.src.optimizers.adam.Adam'>,  
'learning_rate': 0.041022708887198674, 'w0': 0.3469856015031173, 'w1':  
2.177388642519332}. Best is trial 1 with value: 0.9738743901252747.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
  
[I 2024-09-11 10:28:22,813] Trial 20 finished with value:  
0.7804335951805115 and parameters: {'n_units1': 60, 'n_units2': 40,  
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,  
'learning_rate': 0.0020283061941446566, 'w0': 0.04467541853423966,  
'w1': 0.19455347976245996}. Best is trial 1 with value:  
0.9738743901252747.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
  
[I 2024-09-11 10:28:32,496] Trial 21 finished with value:  
0.93996661901474 and parameters: {'n_units1': 16, 'n_units2': 9,  
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,  
'learning_rate': 0.029134859853401984, 'w0': 0.22149705482178353,  
'w1': 0.10579745963400629}. Best is trial 1 with value:  
0.9738743901252747.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
  
[I 2024-09-11 10:28:43,970] Trial 22 finished with value:  
0.887159526348114 and parameters: {'n_units1': 24, 'n_units2': 19,  
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,  
'learning_rate': 0.11554797584803952, 'w0': 0.1646629937533447, 'w1':  
0.07007338446902794}. Best is trial 1 with value: 0.9738743901252747.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
  
[I 2024-09-11 10:28:56,130] Trial 23 finished with value:  
0.8271262049674988 and parameters: {'n_units1': 42, 'n_units2': 11,  
'optimizer': <class 'keras.src.optimizers.adadelta.Adadelta'>,  
'learning_rate': 0.01113791674601724, 'w0': 0.46521511282703176, 'w1':  
0.1791897559164163}. Best is trial 1 with value: 0.9738743901252747.
```

```

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

[I 2024-09-11 10:29:07,200] Trial 24 finished with value:
0.8849360942840576 and parameters: {'n_units1': 21, 'n_units2': 23,
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,
'learning_rate': 0.04077606131235491, 'w0': 0.10365424641366287, 'w1':
0.3889647176165156}. Best is trial 1 with value: 0.9738743901252747.

```

Bu kod, Optuna kullanarak bir sinir ağı modelinin hiperparametrelerini optimize etmek için bir çalışmayı (study) oluşturur ve objective fonksiyonunu 25 denemede optimize eder. Çalışma, model performansını en üst düzeye çıkarmak için optimize edilir (direction="maximize"). Her denemede, Optuna farklı hiperparametreler önerir (örneğin, gizli katmanlardaki nöron sayıları, optimizer türü, öğrenme oranı ve sınıf ağırlıkları), model eğitilir ve erken durdurma mekanizması uygulanır. Bu sonuçlar, Optuna'nın 25 denemelik optimizasyon sürecinin çıktılarıdır. Her deneme (trial), modelin farklı hiperparametrelerle eğitilmesi ve doğruluk değerinin hesaplanmasıyla gerçekleşmiştir. İlk denemeden itibaren, Optuna farklı nöron sayıları (n_units1, n_units2), optimizasyon algoritmaları (optimizer), öğrenme oranları (learning_rate), ve sınıf ağırlıkları (w0, w1) kullanarak modelin doğruluğunu artırmaya çalışmıştır.

En iyi sonuç, 1. denemede elde edilmiş ve 0.973874 doğruluk değeriyle en yüksek doğruluğa ulaşmıştır. Bu denemede kullanılan parametreler: n_units1=106, n_units2=8, optimizer=rmsprop, learning_rate=0.0404, w0=0.0694, ve w1=0.1262 olarak belirlenmiştir. Optuna, bu sonuçlara dayanarak modelin performansını en üst düzeye çıkarmak için hangi hiperparametrelerin en etkili olduğunu belirlemeye çalışmıştır.

```

# Get the best hyperparameters found by Optuna
best_params = study.best_params
best_params

{'n_units1': 106,
 'n_units2': 8,
 'optimizer': keras.src.optimizers.rmsprop.RMSprop,
 'learning_rate': 0.040423791709951105,
 'w0': 0.06944117821960119,
 'w1': 0.1262152089138289}

```

Optuna, modelin performansını en üst düzeye çıkarmak için en iyi hiperparametreleri belirlemiştir. En iyi hiperparametreler şunlardır: birinci katmandaki nöron sayısı (n_units1) 106, ikinci katmandaki nöron sayısı (n_units2) 8'dir. Modelin optimizasyonu için RMSprop algoritması kullanılmış ve öğrenme oranı (learning_rate) 0.0404 olarak belirlenmiştir. Ayrıca, sınıf ağırlıkları için w0 0.0694 ve w1 0.1262 değerlerine sahiptir. Bu parametreler, modelin doğruluk oranını maksimize etmek amacıyla en iyi sonuç veren hiperparametreler olarak seçilmiştir.

```

# build model with optuna parameters
unit1, unit2, optimizer, lr, w0, w1 = (
    study.best_params["n_units1"],
    study.best_params["n_units2"],
    study.best_params["optimizer"],

```

```
study.best_params["learning_rate"],
study.best_params["w0"],
study.best_params["w1"],
)

model = Sequential()
model.add(Dense(unit1, activation="relu"))
model.add(Dense(unit2, activation="relu"))
model.add(Dense(1, activation="sigmoid"))
opt = optimizer(learning_rate=lr)
model.compile(optimizer=opt, loss="binary_crossentropy",
metrics=["Recall"])

# train model
model.fit(
    X_train,
    y_train,
    validation_data=(X_test, y_test),
    batch_size=16,
    epochs=100,
    callbacks=[early_stop],
    verbose=1,
)

Epoch 1/100
574/574 ━━━━━━━━━━ 2s 2ms/step - Recall: 0.1881 - loss:
0.3712 - val_Recall: 0.7157 - val_loss: 0.1983
Epoch 2/100
574/574 ━━━━━━━━━━ 1s 2ms/step - Recall: 0.8087 - loss:
0.1761 - val_Recall: 0.7358 - val_loss: 0.1786
Epoch 3/100
574/574 ━━━━━━━━━━ 1s 2ms/step - Recall: 0.8299 - loss:
0.1548 - val_Recall: 0.8227 - val_loss: 0.1432
Epoch 4/100
574/574 ━━━━━━━━━━ 1s 2ms/step - Recall: 0.8567 - loss:
0.1449 - val_Recall: 0.8294 - val_loss: 0.1505
Epoch 5/100
574/574 ━━━━━━━━━━ 1s 2ms/step - Recall: 0.8633 - loss:
0.1410 - val_Recall: 0.8528 - val_loss: 0.1339
Epoch 6/100
574/574 ━━━━━━━━━━ 1s 2ms/step - Recall: 0.8724 - loss:
0.1357 - val_Recall: 0.8729 - val_loss: 0.1294
Epoch 7/100
574/574 ━━━━━━━━━━ 1s 2ms/step - Recall: 0.8759 - loss:
0.1328 - val_Recall: 0.8629 - val_loss: 0.1302
Epoch 8/100
574/574 ━━━━━━━━━━ 2s 3ms/step - Recall: 0.8875 - loss:
0.1316 - val_Recall: 0.8896 - val_loss: 0.1285
Epoch 9/100
574/574 ━━━━━━━━━━ 2s 3ms/step - Recall: 0.8871 - loss:
```

```

0.1280 - val_Recall: 0.8696 - val_loss: 0.1287
Epoch 10/100
574/574 ━━━━━━━━ 1s 3ms/step - Recall: 0.8871 - loss:
0.1283 - val_Recall: 0.8930 - val_loss: 0.1346
Epoch 11/100
574/574 ━━━━━━━━ 1s 2ms/step - Recall: 0.8895 - loss:
0.1277 - val_Recall: 0.8829 - val_loss: 0.1385
Epoch 12/100
574/574 ━━━━━━━━ 1s 2ms/step - Recall: 0.8826 - loss:
0.1277 - val_Recall: 0.9030 - val_loss: 0.1352
Epoch 13/100
574/574 ━━━━━━━━ 1s 2ms/step - Recall: 0.8840 - loss:
0.1269 - val_Recall: 0.8829 - val_loss: 0.1317
Epoch 14/100
574/574 ━━━━━━━━ 1s 2ms/step - Recall: 0.8855 - loss:
0.1244 - val_Recall: 0.8963 - val_loss: 0.1318
Epoch 15/100
574/574 ━━━━━━━━ 1s 2ms/step - Recall: 0.9014 - loss:
0.1263 - val_Recall: 0.9064 - val_loss: 0.1359
Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

<keras.src.callbacks.history.History at 0x7dfc56fe41c0>

```

Optuna tarafından belirlenen en iyi hiperparametrelerle oluşturulan model, eğitim sürecinde 15 epoch boyunca eğitildi ve Recall metriğiyle değerlendirildi. İlk epoch'ta modelin eğitim ve doğrulama verilerinde elde ettiği Recall değerleri sırasıyla 0.1881 ve 0.7157 iken, daha sonraki epoch'larda modelin performansı arttı ve doğrulama verisinde Recall değeri en yüksek 0.9064'e ulaştı. Ancak, early_stopping (erken durdurma) geri çağrı yöntemi, modelin performansının ilk epoch'tan sonra en iyi değere ulaştığını belirleyerek eğitimi erken durdurdu ve modelin en iyi ağırlıklarını ilk epoch'a geri yükledi. Bu, modelin aşırı uyum yapmadan en iyi performansını korumasını sağladı.

| model.summary() | |
|------------------------|--------------|
| Model: "sequential_26" | |
| Layer (type) | Output Shape |
| Param # | |
| dense_80 (Dense) | (None, 106) |
| 2,014 | |
| dense_81 (Dense) | (None, 8) |
| 856 | |

| | |
|------------------|-----------|
| | |
| dense_82 (Dense) | (None, 1) |
| 9 | |

Total params: 5,760 (22.50 KB)

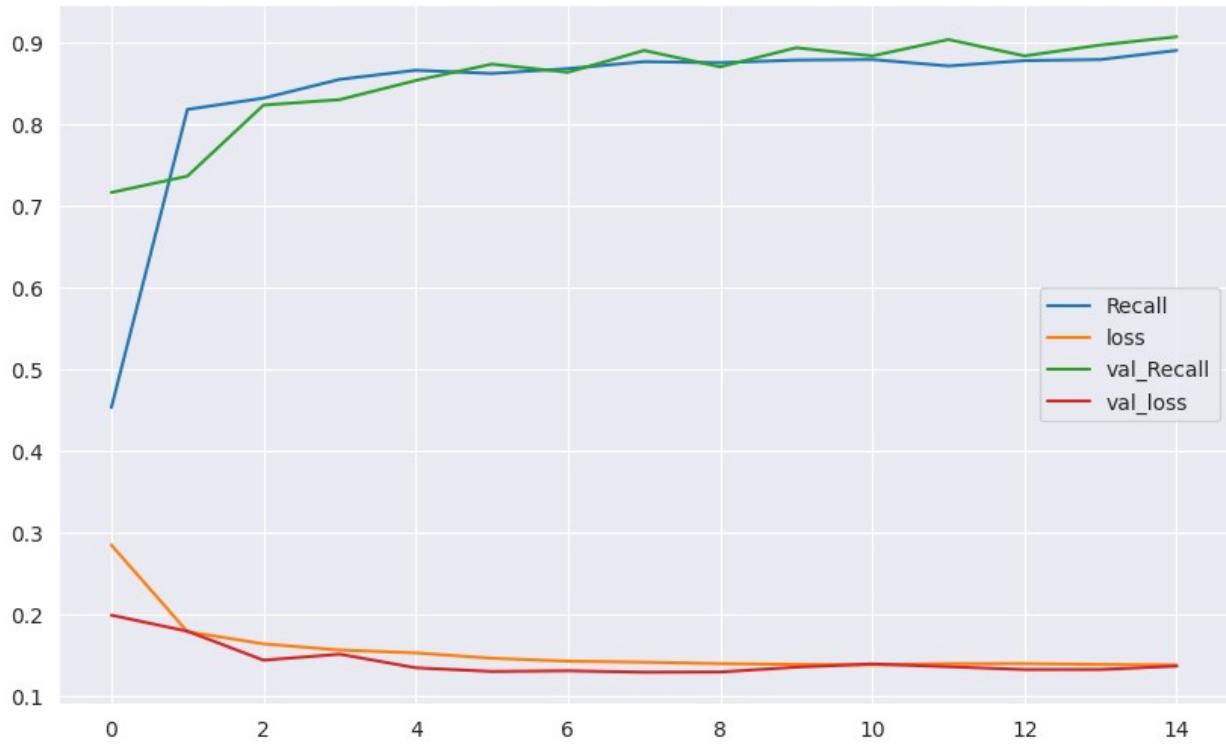
Trainable params: 2,879 (11.25 KB)

Non-trainable params: 0 (0.00 B)

Optimizer params: 2,881 (11.26 KB)

Optuna tarafından belirlenen hiperparametrelerle oluşturulan modelin yapısında üç katman bulunuyor. İlk katman, 106 nöronlu bir yoğun (dense) katman ve toplamda 2,014 parametreye sahip. İkinci katman, 8 nöronlu başka bir yoğun katman ve 856 parametreye sahip. Son katman, çıktı katmanı olarak işlev gören tek bir nörona sahip ve 9 parametre içeriyor. Modelin toplamda 2,879 eğitilebilir parametresi ve 0 eğitilemez parametresi bulunuyor. Ayrıca, optimizer için toplam 2,881 parametre kullanılmış. Bu model, girişten çıkışa kadar 5,760 parametreyle yapılandırılmıştır ve hafif bir yapıda olmasına rağmen verimli bir şekilde öğrenme ve genellemeye yapma potansiyeline sahiptir.

```
history = model.history.history  
pd.DataFrame(history).plot()  
<Axes: >
```



`model.history.history` kullanılarak elde edilen `history` değişkeni, modelin eğitim süreci boyunca kaydedilen metriklerin (örneğin, kayıp fonksiyonu (loss) ve geri çağrıma (recall) gibi) değerlerini içerir. Bu veriyi bir pandas DataFrame'e dönüştürdükten sonra `plot()` fonksiyonu kullanılarak çizilir. Grafikte, her bir epoch (eğitim döngüsü) boyunca modelin hem eğitim hem de doğrulama verisi üzerindeki performansı görselleştirilir. Bu, modelin zaman içindeki performans gelişimini, overfitting (aşırı uyum) belirtilerini ve erken durdurma (early stopping) kriterlerinin etkisini anlamak için kullanışlıdır.

```
y_pred = (model.predict(X_test) > 0.5).astype("int32")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

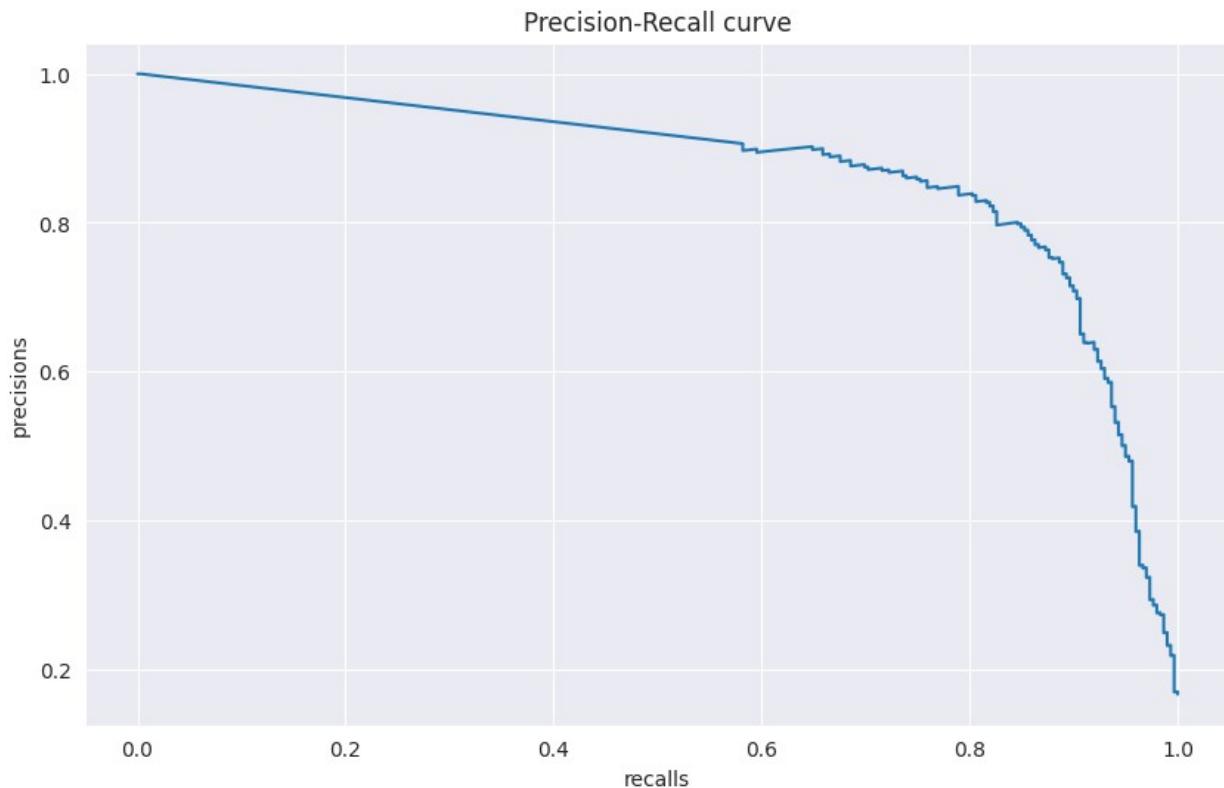
57/57 ━━━━━━━━ 0s 2ms/step
[[1469  31]
 [ 85 214]]
      precision    recall  f1-score   support
          0       0.95     0.98     0.96     1500
          1       0.87     0.72     0.79      299
   accuracy                           0.94     1799
  macro avg       0.91     0.85     0.87     1799
weighted avg       0.93     0.94     0.93     1799
```

Model, test verisi üzerinde tahminler yaparak, sonuçları bir karışıklık matrisi (confusion matrix) ve sınıflandırma raporu (classification report) ile değerlendirmiştir. Karışıklık matrisine göre,

model 1.500 adet "0" sınıfını doğru tahmin ederken sadece 31 tanesinde hata yapmıştır, yani "0" sınıfı için oldukça yüksek bir performansa sahiptir. "1" sınıfında ise 299 örnekten 214'ünü doğru tahmin ederken 85'inde hata yapmıştır. Sınıflandırma raporuna göre, modelin genel doğruluğu (accuracy) %94, "0" sınıfı için F1 skoru 0.96 ve "1" sınıfı için 0.79'dur. Bu da modelin genel olarak güçlü bir performans sergilediğini, ancak "1" sınıfında (daha az görülen sınıfta) nispeten daha düşük bir geri çağırma (recall) değerine sahip olduğunu gösterir.

```
y_pred_proba = model.predict(X_test)
precisions, recalls, thresholds = precision_recall_curve(y_test,
y_pred_proba)
plt.plot(recalls, precisions, label='ANN')
plt.xlabel('recalls')
plt.ylabel('precisions')
plt.title('Precision-Recall curve')
plt.show()
```

57/57 ━━━━━━━━ 0s 2ms/step



Modelin tahmin sonuçlarının doğruluğunu ve geri çağırma oranlarını daha ayrıntılı olarak değerlendirmek için bir "Precision-Recall" eğrisi çizilmiştir. Bu eğri, modelin farklı olasılık eşik değerlerinde (thresholds) pozitif sınıf için tahmin yaparken ne kadar iyi performans gösterdiğini gösterir. Grafikte, geri çağırma (recall) değerleri x ekseninde ve doğruluk (precision) değerleri y ekseninde yer alır. Eğrinin yüksek olması, modelin hem yüksek doğruluk hem de geri çağırma oranını aynı anda başardığını gösterir. Eğri, modelin özellikle dengesiz veri setlerinde veya "pozitif" sınıfın daha az olduğu durumlarda performansını görselleştirmek için kullanılır. Bu

görselleştirme, modelin farklı eşik değerlerinde nasıl davranışacağını ve doğruluk-geri çağrıma dengesi hakkında fikir verir.

```
average_precision_score(y_test, y_pred_proba)  
0.843797073049419
```

average_precision_score metriği, modelin sınıflandırma performansını özetleyen ve özellikle dengesiz veri setlerinde kullanışlı olan bir ölçütür. Bu skor, "Precision-Recall" eğrisi altında kalan alanın (AP - Average Precision) hesaplanmasıyla elde edilir. Skorun 0.8438 olması, modelin pozitif sınıf için tahmin yaparken oldukça iyi bir performans gösterdiğini belirtir; yani modelin ortalama doğruluğu, geri çağrıma seviyesine göre %84.38 civarındadır. Bu yüksek skor, modelin hem doğruluk hem de geri çağrıma arasında başarılı bir denge sağladığını ve pozitif sınıfları doğru bir şekilde tanımlayabildiğini gösterir.

```
DL_AP_op = average_precision_score(y_test, y_pred_proba)  
DL_f1_op = f1_score(y_test, y_pred)  
DL_rec_op = recall_score(y_test, y_pred)
```

DL_AP_op, DL_f1_op ve DL_rec_op metrikleri, modelin sınıflandırma performansını değerlendirmek için kullanılır.

DL_AP_op (Average Precision Score): Bu metrik, modelin pozitif sınıfı doğru bir şekilde tahmin etme yeteneğini değerlendirir ve "Precision-Recall" eğrisi altında kalan alanı temsil eder. Daha yüksek bir değer, modelin pozitif sınıf için iyi bir performans gösterdiğini ifade eder.

DL_f1_op (F1 Score): F1 skoru, modelin doğruluk (precision) ve geri çağrıma (recall) değerlerinin harmonik ortalamasını hesaplar. Bu metrik, özellikle sınıflar arasında dengesizlik olduğunda kullanışlıdır. Yüksek bir F1 skoru, modelin hem doğru pozitifleri iyi bir şekilde tespit ettiğini hem de yanlış pozitiflerin sayısını düşük tuttuğunu gösterir.

DL_rec_op (Recall Score): Geri çağrıma, modelin pozitif sınıfı ne kadar doğru bir şekilde tahmin ettiğini ölçer. Yüksek bir geri çağrıma değeri, modelin pozitif örneklerin çoğunu tespit ettiğini gösterir.

Bu üç metrik, modelin genel performansını ve pozitif sınıfları ne kadar iyi tanımladığını değerlendirmek için kullanılır.

optuna with smote

```
#pip install imblearn  
  
Collecting imblearn  
  Downloading imblearn-0.0-py2.py3-none-any.whl.metadata (355 bytes)  
Requirement already satisfied: imbalanced-learn in  
/usr/local/lib/python3.10/dist-packages (from imblearn) (0.12.3)  
Requirement already satisfied: numpy>=1.17.3 in  
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.26.4)  
Requirement already satisfied: scipy>=1.5.0 in  
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn-
```

```
>imblearn) (1.13.1)
Requirement already satisfied: scikit-learn>=1.0.2 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.3.2)
Requirement already satisfied: joblib>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn->imblearn) (3.5.0)
Downloading imblearn-0.0-py2.py3-none-any.whl (1.9 kB)
Installing collected packages: imblearn
Successfully installed imblearn-0.0

#pip install --upgrade optuna imbalanced-learn

Requirement already satisfied: optuna in
/usr/local/lib/python3.10/dist-packages (4.0.0)
Requirement already satisfied: imbalanced-learn in
/usr/local/lib/python3.10/dist-packages (0.12.3)
Requirement already satisfied: alembic>=1.5.0 in
/usr/local/lib/python3.10/dist-packages (from optuna) (1.13.2)
Requirement already satisfied: colorlog in
/usr/local/lib/python3.10/dist-packages (from optuna) (6.8.2)
Requirement already satisfied: numpy in
/usr/local/lib/python3.10/dist-packages (from optuna) (1.26.4)
Requirement already satisfied: packaging>=20.0 in
/usr/local/lib/python3.10/dist-packages (from optuna) (24.1)
Requirement already satisfied: sqlalchemy>=1.3.0 in
/usr/local/lib/python3.10/dist-packages (from optuna) (2.0.32)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-
packages (from optuna) (4.66.5)
Requirement already satisfied: PyYAML in
/usr/local/lib/python3.10/dist-packages (from optuna) (6.0.2)
Requirement already satisfied: scipy>=1.5.0 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.13.1)
Requirement already satisfied: scikit-learn>=1.0.2 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.3.2)
Requirement already satisfied: joblib>=1.1.1 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in
/usr/local/lib/python3.10/dist-packages (from imbalanced-learn)
(3.5.0)
Requirement already satisfied: Mako in /usr/local/lib/python3.10/dist-
packages (from alembic>=1.5.0->optuna) (1.3.5)
Requirement already satisfied: typing-extensions>=4 in
/usr/local/lib/python3.10/dist-packages (from alembic>=1.5.0->optuna)
```

```
(4.12.2)
Requirement already satisfied: greenlet!=0.4.17 in
/usr/local/lib/python3.10/dist-packages (from sqlalchemy>=1.3.0-
>optuna) (3.0.3)
Requirement already satisfied: MarkupSafe>=0.9.2 in
/usr/local/lib/python3.10/dist-packages (from Mako->alembic>=1.5.0-
>optuna) (2.1.5)

from tensorflow.keras.callbacks import EarlyStopping,
LearningRateScheduler
from sklearn.metrics import classification_report, confusion_matrix,
f1_score
from tensorflow.keras.layers import Dropout

# SMOTE
from imblearn.over_sampling import SMOTE

smote = SMOTE(random_state=42)
X_train_resampled, y_train_resampled = smote.fit_resample(X_train,
y_train)
```

Bu kod parçası, veri dengesizliği sorununu ele almak için SMOTE (Synthetic Minority Over-sampling Technique) yöntemini kullanır. Öncelikle, SMOTE sınıfı imblearn kütüphanesinden içe aktarılır ve bir SMOTE nesnesi oluşturulur, burada random_state=42 belirli bir rastgelelik sağlamak amacıyla kullanılır. Ardından, fit_resample yöntemi kullanılarak eğitim verileri (X_train ve y_train) üzerinde SMOTE uygulanır. Bu işlem, azınlık sınıfının örneklerini artırarak eğitim verilerinin dengelenmesini sağlar ve böylece modelin azınlık sınıfı öğrenme yeteneği iyileştirilir. Sonuç olarak, X_train_resampled ve y_train_resampled isimli yeniden dengelenmiş eğitim verileri elde edilir. Bu yaklaşım, modelin performansını artırmak ve azınlık sınıf üzerindeki tahminlerin doğruluğunu geliştirmek için kullanılır.

```
# Optuna objective fonksiyonu

def create_model(trial):
    n_units1 = trial.suggest_int('n_units1', 8, 128)
    n_units2 = trial.suggest_int('n_units2', 8, 128)
    n_units3 = trial.suggest_int('n_units3', 8, 128)
    optimizer = trial.suggest_categorical("optimizer", [Adam,
Adadelta, RMSprop, Nadam])
    learning_rate = trial.suggest_loguniform('learning_rate', 1e-4,
1e-2)

    os.environ["TF_DETERMINISTIC_OPS"] = "1"
    tf.keras.utils.set_random_seed(SEED)

    model = Sequential()
    model.add(Dense(n_units1, input_dim=X_train.shape[1],
activation='relu'))
    model.add(Dense(n_units2, activation='relu'))
```

```

model.add(Dense(n_units3, activation='relu'))
model.add(Dropout(trial.suggest_uniform('dropout', 0.2, 0.5)))
model.add(Dense(1, activation='sigmoid'))
model.compile(loss='binary_crossentropy',
              optimizer=optimizer(learning_rate=learning_rate),
              metrics=[trial_metric, "Precision", "AUC"])
return model

def objective(trial):
    model = create_model(trial)
    w0 = trial.suggest_loguniform("w0", 0.01, 5)
    w1 = trial.suggest_loguniform("w1", 0.01, 5)

    batch_size = trial.suggest_categorical("batch_size", [128, 256,
512])

    model.fit(X_train_resampled,
               y_train_resampled,
               validation_data=(X_val, y_val),
               batch_size=batch_size,
               epochs=100,
               callbacks=[early_stop],
               class_weight={0: w0, 1: w1},
               verbose=0)

    y_pred_val = (model.predict(X_val) > 0.5).astype("int32")
    score = f1_score(y_val, y_pred_val)

    return score

```

Bu kod parçası, Optuna kullanarak derin öğrenme modelinin hiperparametrelerini optimize etmek için kullanılan iki fonksiyonu içerir. `create_model` fonksiyonu, Optuna'nın önerdiği hiperparametreler doğrultusunda bir model oluşturur. Bu fonksiyon, ilk üç katman için nöron sayısını, optimizasyon algoritmasını, öğrenme oranını ve dropout oranını ayarlamak için çeşitli aralıklar ve dağılımlar kullanır. Modelin derinliği, nöron sayıları, dropout oranı ve optimizer gibi parametreler Optuna'nın belirdiği şekilde ayarlanır ve model derlenir. `objective` fonksiyonu ise modelin performansını değerlendirmek için kullanılır. Bu fonksiyon, `create_model` aracılığıyla bir model oluşturur, ağırlıklar için logaritmik dağılımlar ve batch boyutları için kategorik seçenekler belirler. Model, yeniden dengelenmiş eğitim verileri ve doğrulama seti ile eğitilir. Son olarak, modelin doğrulama setindeki F1 skoru hesaplanır ve bu skor, Optuna'nın hiperparametre optimizasyon sürecinde hedef olarak kullanılır. Bu yaklaşım, modelin performansını artırmak ve en iyi hiperparametreleri belirlemek için etkili bir yöntemdir.

```

study = optuna.create_study(direction="maximize")
study.optimize(objective, n_trials=30)

[I 2024-09-11 10:31:48,661] A new study created in memory with name:
no-name-c9e8c377-2c32-42b9-be92-d058bf267cba

```

```
Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━ 1s 17ms/step

[I 2024-09-11 10:32:22,250] Trial 0 finished with value:
0.28427249789739273 and parameters: {'n_units1': 119, 'n_units2': 108,
'n_units3': 73, 'optimizer': <class
'keras.src.optimizers.rmsprop.RMSprop'>, 'learning_rate':
0.0001206597875762706, 'dropout': 0.20110499365824325, 'w0':
0.0860121694898735, 'w1': 1.6322495253743345, 'batch_size': 512}. Best
is trial 0 with value: 0.28427249789739273.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━ 0s 4ms/step

[I 2024-09-11 10:32:37,447] Trial 1 finished with value: 0.0 and
parameters: {'n_units1': 63, 'n_units2': 89, 'n_units3': 17,
'optimizer': <class 'keras.src.optimizers.rmsprop.RMSprop'>,
'learning_rate': 0.00452989907077242, 'dropout': 0.2281553808795589,
'w0': 0.15128247497004163, 'w1': 0.02515417387933124, 'batch_size':
512}. Best is trial 0 with value: 0.28427249789739273.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━ 0s 2ms/step

[I 2024-09-11 10:32:50,234] Trial 2 finished with value:
0.28427249789739273 and parameters: {'n_units1': 116, 'n_units2': 11,
'n_units3': 17, 'optimizer': <class 'keras.src.optimizers.adam.Adam'>,
'learning_rate': 0.0005127125591371392, 'dropout':
0.29097240310326455, 'w0': 0.046028948463524746, 'w1':
0.9023878852899166, 'batch_size': 512}. Best is trial 0 with value:
0.28427249789739273.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━ 0s 4ms/step

[I 2024-09-11 10:32:56,280] Trial 3 finished with value:
0.2694663167104112 and parameters: {'n_units1': 58, 'n_units2': 17,
'n_units3': 13, 'optimizer': <class 'keras.src.optimizers.adam.Adam'>,
'learning_rate': 0.0003979312050115282, 'dropout': 0.2755365026318628,
'w0': 0.012677797220817596, 'w1': 0.6654171288267028, 'batch_size':
512}. Best is trial 0 with value: 0.28427249789739273.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━ 0s 3ms/step
```

```
[I 2024-09-11 10:33:03,980] Trial 4 finished with value: 0.0 and
parameters: {'n_units1': 115, 'n_units2': 52, 'n_units3': 107,
'optimizer': <class 'keras.src.optimizers.nadam.Nadam'>,
'learning_rate': 0.000424331935522503, 'dropout': 0.3464599773804127,
'w0': 3.947516057982391, 'w1': 0.0461405394150916, 'batch_size': 512}.
Best is trial 0 with value: 0.28427249789739273.
```

Epoch 15: early stopping

```
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 0s 3ms/step
```

```
[I 2024-09-11 10:33:14,123] Trial 5 finished with value:
0.24196597353497165 and parameters: {'n_units1': 91, 'n_units2': 125,
'n_units3': 44, 'optimizer': <class
'keras.src.optimizers.adadelta.Adadelta'>, 'learning_rate':
0.00027500182578005315, 'dropout': 0.4822949174515891, 'w0':
3.449410283041911, 'w1': 0.05170434175835413, 'batch_size': 256}. Best
is trial 0 with value: 0.28427249789739273.
```

Epoch 15: early stopping

```
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 0s 4ms/step
```

```
[I 2024-09-11 10:33:24,942] Trial 6 finished with value:
0.011764705882352941 and parameters: {'n_units1': 47, 'n_units2': 100,
'n_units3': 117, 'optimizer': <class
'keras.src.optimizers.adadelta.Adadelta'>, 'learning_rate':
0.00207100692635426, 'dropout': 0.29089167599240456, 'w0':
1.1869121806347587, 'w1': 4.633639491227513, 'batch_size': 128}. Best
is trial 0 with value: 0.28427249789739273.
```

Epoch 15: early stopping

```
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 0s 3ms/step
```

```
[I 2024-09-11 10:33:33,286] Trial 7 finished with value:
0.2902097902097902 and parameters: {'n_units1': 83, 'n_units2': 46,
'n_units3': 32, 'optimizer': <class
'keras.src.optimizers.adadelta.Adadelta'>, 'learning_rate':
0.00031257078407449524, 'dropout': 0.4242420985458869, 'w0':
0.011869491314933221, 'w1': 0.039279873162070995, 'batch_size': 128}.
Best is trial 7 with value: 0.2902097902097902.
```

Epoch 15: early stopping

```
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 0s 4ms/step
```

```
[I 2024-09-11 10:33:40,921] Trial 8 finished with value: 0.0 and
parameters: {'n_units1': 44, 'n_units2': 76, 'n_units3': 26,
'optimizer': <class 'keras.src.optimizers.nadam.Nadam'>,
'learning_rate': 0.00013654064697039863, 'dropout':
```

```
0.24975712137132353, 'w0': 0.1602559614440135, 'w1':  
0.015939453601917553, 'batch_size': 256}. Best is trial 7 with value:  
0.2902097902097902.
```

```
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
32/32 ━━━━━━━━ 0s 3ms/step
```

```
[I 2024-09-11 10:33:50,005] Trial 9 finished with value: 0.0 and  
parameters: {'n_units1': 13, 'n_units2': 103, 'n_units3': 79,  
'optimizer': <class 'keras.src.optimizers.adam.Adam'>,  
'learning_rate': 0.0029735998479813765, 'dropout': 0.2752600521402379,  
'w0': 0.1312247765384603, 'w1': 0.018395432666520706, 'batch_size':  
256}. Best is trial 7 with value: 0.2902097902097902.
```

```
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
32/32 ━━━━━━━━ 0s 3ms/step
```

```
[I 2024-09-11 10:33:59,316] Trial 10 finished with value:  
0.29455445544554454 and parameters: {'n_units1': 83, 'n_units2': 46,  
'n_units3': 50, 'optimizer': <class  
'keras.src.optimizers.adadelta.Adadelta'>, 'learning_rate':  
0.0011806698703574368, 'dropout': 0.4650277764337149, 'w0':  
0.010504118624920077, 'w1': 0.1098453237205083, 'batch_size': 128}.  
Best is trial 10 with value: 0.29455445544554454.
```

```
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
32/32 ━━━━━━━━ 0s 4ms/step
```

```
[I 2024-09-11 10:34:07,680] Trial 11 finished with value:  
0.023255813953488372 and parameters: {'n_units1': 90, 'n_units2': 46,  
'n_units3': 46, 'optimizer': <class  
'keras.src.optimizers.adadelta.Adadelta'>, 'learning_rate':  
0.001194311034654751, 'dropout': 0.4707436417425283, 'w0':  
0.011325629101863452, 'w1': 0.15467300222361477, 'batch_size': 128}.  
Best is trial 10 with value: 0.29455445544554454.
```

```
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
32/32 ━━━━━━━━ 0s 3ms/step
```

```
[I 2024-09-11 10:34:15,923] Trial 12 finished with value:  
0.3790613718411552 and parameters: {'n_units1': 87, 'n_units2': 43,  
'n_units3': 48, 'optimizer': <class  
'keras.src.optimizers.adadelta.Adadelta'>, 'learning_rate':  
0.009659193649923977, 'dropout': 0.42278284591454385, 'w0':  
0.03204256875991151, 'w1': 0.1492487782187302, 'batch_size': 128}.  
Best is trial 12 with value: 0.3790613718411552.
```

```
Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 1s 3ms/step

[I 2024-09-11 10:34:25,925] Trial 13 finished with value:
0.28771929824561404 and parameters: {'n_units1': 79, 'n_units2': 34,
'n_units3': 56, 'optimizer': <class
'keras.src.optimizers.adadelta.Adadelta'>, 'learning_rate':
0.009770273608403707, 'dropout': 0.41313009534612777, 'w0':
0.03038399109626079, 'w1': 0.17363512855620383, 'batch_size': 128}.
Best is trial 12 with value: 0.3790613718411552.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 0s 3ms/step

[I 2024-09-11 10:34:37,133] Trial 14 finished with value: 0.0 and
parameters: {'n_units1': 105, 'n_units2': 64, 'n_units3': 88,
'optimizer': <class 'keras.src.optimizers.adadelta.Adadelta'>,
'learning_rate': 0.00863150036941232, 'dropout': 0.42140950453113424,
'w0': 0.4369726154044633, 'w1': 0.10361379351123921, 'batch_size':
128}. Best is trial 12 with value: 0.3790613718411552.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 0s 3ms/step

[I 2024-09-11 10:34:44,880] Trial 15 finished with value:
0.25155666251556663 and parameters: {'n_units1': 99, 'n_units2': 25,
'n_units3': 60, 'optimizer': <class
'keras.src.optimizers.adadelta.Adadelta'>, 'learning_rate':
0.0010165618944155187, 'dropout': 0.3649619828379925, 'w0':
0.0319004951403261, 'w1': 0.3311276271010592, 'batch_size': 128}. Best
is trial 12 with value: 0.3790613718411552.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 0s 3ms/step

[I 2024-09-11 10:34:55,226] Trial 16 finished with value: 0.0 and
parameters: {'n_units1': 73, 'n_units2': 73, 'n_units3': 93,
'optimizer': <class 'keras.src.optimizers.adadelta.Adadelta'>,
'learning_rate': 0.005243651541603881, 'dropout': 0.45081288685024234,
'w0': 0.39131163768991933, 'w1': 0.37774290223406953, 'batch_size':
128}. Best is trial 12 with value: 0.3790613718411552.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 0s 3ms/step
```

```
[I 2024-09-11 10:35:05,480] Trial 17 finished with value:  
0.34468524251805976 and parameters: {'n_units1': 28, 'n_units2': 33,  
'n_units3': 34, 'optimizer': <class  
'keras.src.optimizers.nadam.Nadam'>, 'learning_rate':  
0.0014600223793955335, 'dropout': 0.3699963263953118, 'w0':  
0.023214879283313908, 'w1': 0.12874097560477246, 'batch_size': 128}.  
Best is trial 12 with value: 0.3790613718411552.
```

```
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
32/32 ━━━━━━━━ 0s 3ms/step
```

```
[I 2024-09-11 10:35:13,656] Trial 18 finished with value:  
0.5925925925925927 and parameters: {'n_units1': 12, 'n_units2': 32,  
'n_units3': 34, 'optimizer': <class  
'keras.src.optimizers.nadam.Nadam'>, 'learning_rate':  
0.0019776706917623657, 'dropout': 0.3588848272684838, 'w0':  
0.0652792567464242, 'w1': 0.08542710095499491, 'batch_size': 128}.  
Best is trial 18 with value: 0.5925925925925927.
```

```
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
32/32 ━━━━━━━━ 0s 3ms/step
```

```
[I 2024-09-11 10:35:22,997] Trial 19 finished with value:  
0.723529411764706 and parameters: {'n_units1': 17, 'n_units2': 60,  
'n_units3': 64, 'optimizer': <class  
'keras.src.optimizers.nadam.Nadam'>, 'learning_rate':  
0.0031246475723436724, 'dropout': 0.3278437274419819, 'w0':  
0.06455374519841697, 'w1': 0.010308986248868859, 'batch_size': 128}.  
Best is trial 19 with value: 0.723529411764706.
```

```
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
32/32 ━━━━━━━━ 0s 4ms/step
```

```
[I 2024-09-11 10:35:30,175] Trial 20 finished with value:  
0.5840336134453782 and parameters: {'n_units1': 9, 'n_units2': 60,  
'n_units3': 61, 'optimizer': <class  
'keras.src.optimizers.nadam.Nadam'>, 'learning_rate':  
0.0026757366542806238, 'dropout': 0.33761381644187116, 'w0':  
0.06755783229431246, 'w1': 0.0685760799089328, 'batch_size': 256}.  
Best is trial 19 with value: 0.723529411764706.
```

```
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
32/32 ━━━━━━━━ 0s 3ms/step
```

```
[I 2024-09-11 10:35:38,192] Trial 21 finished with value:  
0.011764705882352941 and parameters: {'n_units1': 13, 'n_units2': 63,  
'n_units3': 66, 'optimizer': <class
```

```
'keras.src.optimizers.nadam.Nadam', 'learning_rate':  
0.0023672038143310293, 'dropout': 0.33228635123731304, 'w0':  
0.07640543928035667, 'w1': 0.011388828398638773, 'batch_size': 256}.  
Best is trial 19 with value: 0.723529411764706.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
32/32 ━━━━━━━━ 0s 3ms/step  
  
[I 2024-09-11 10:35:46,805] Trial 22 finished with value:  
0.7129629629629629 and parameters: {'n_units1': 25, 'n_units2': 59,  
'n_units3': 85, 'optimizer': <class  
'keras.src.optimizers.nadam.Nadam', 'learning_rate':  
0.003854608428510281, 'dropout': 0.3254440718749748, 'w0':  
0.06673685162336704, 'w1': 0.06993677026585914, 'batch_size': 256}.  
Best is trial 19 with value: 0.723529411764706.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
32/32 ━━━━━━━━ 0s 3ms/step  
  
[I 2024-09-11 10:35:55,399] Trial 23 finished with value:  
0.3636363636363636 and parameters: {'n_units1': 28, 'n_units2': 83,  
'n_units3': 97, 'optimizer': <class  
'keras.src.optimizers.nadam.Nadam', 'learning_rate':  
0.004332648407213207, 'dropout': 0.3147334474286748, 'w0':  
0.29614612287456715, 'w1': 0.028201892215152194, 'batch_size': 256}.  
Best is trial 19 with value: 0.723529411764706.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
32/32 ━━━━━━━━ 0s 3ms/step  
  
[I 2024-09-11 10:36:03,836] Trial 24 finished with value:  
0.3816091954022989 and parameters: {'n_units1': 25, 'n_units2': 31,  
'n_units3': 81, 'optimizer': <class  
'keras.src.optimizers.nadam.Nadam', 'learning_rate':  
0.000770024495009015, 'dropout': 0.3859603606797253, 'w0':  
0.05405216681484389, 'w1': 0.07174509902244432, 'batch_size': 256}.  
Best is trial 19 with value: 0.723529411764706.  
  
Epoch 15: early stopping  
Restoring model weights from the end of the best epoch: 1.  
32/32 ━━━━━━━━ 0s 4ms/step  
  
[I 2024-09-11 10:36:14,361] Trial 25 finished with value: 0.0 and  
parameters: {'n_units1': 38, 'n_units2': 56, 'n_units3': 104,  
'optimizer': <class 'keras.src.optimizers.nadam.Nadam',  
'learning_rate': 0.001745020274158034, 'dropout': 0.317321078017592,  
'w0': 0.7150553670585825, 'w1': 0.010282383414112055, 'batch_size':  
128}. Best is trial 19 with value: 0.723529411764706.
```

```
Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 0s 2ms/step

[I 2024-09-11 10:36:21,572] Trial 26 finished with value:
0.8057142857142857 and parameters: {'n_units1': 22, 'n_units2': 69,
'n_units3': 73, 'optimizer': <class
'keras.src.optimizers.nadam.Nadam'>, 'learning_rate':
0.005904573234432499, 'dropout': 0.38039772024278695, 'w0':
0.09931068130920347, 'w1': 0.025268074122304297, 'batch_size': 256}.
Best is trial 26 with value: 0.8057142857142857.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 0s 3ms/step

[I 2024-09-11 10:36:30,920] Trial 27 finished with value:
0.8277945619335346 and parameters: {'n_units1': 21, 'n_units2': 86,
'n_units3': 126, 'optimizer': <class
'keras.src.optimizers.nadam.Nadam'>, 'learning_rate':
0.007029343925124535, 'dropout': 0.39423274126367325, 'w0':
0.10648131592833926, 'w1': 0.02211520066615946, 'batch_size': 256}.
Best is trial 27 with value: 0.8277945619335346.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 0s 3ms/step

[I 2024-09-11 10:36:37,103] Trial 28 finished with value:
0.31818181818181823 and parameters: {'n_units1': 37, 'n_units2': 89,
'n_units3': 128, 'optimizer': <class
'keras.src.optimizers.rmsprop.RMSprop'>, 'learning_rate':
0.0066281014865090665, 'dropout': 0.39443753066292425, 'w0':
0.12100673511532879, 'w1': 0.01895029595941885, 'batch_size': 256}.
Best is trial 27 with value: 0.8277945619335346.

Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.
32/32 ━━━━━━━━ 0s 2ms/step

[I 2024-09-11 10:36:44,957] Trial 29 finished with value:
0.3251231527093596 and parameters: {'n_units1': 54, 'n_units2': 116,
'n_units3': 73, 'optimizer': <class
'keras.src.optimizers.rmsprop.RMSprop'>, 'learning_rate':
0.006160345451676426, 'dropout': 0.3972886728678439, 'w0':
0.2365657528549902, 'w1': 0.031247793886527292, 'batch_size': 256}.
Best is trial 27 with value: 0.8277945619335346.
```

Optuna kullanarak gerçekleştirilen hiperparametre optimizasyonu sürecinde, 30 farklı deneme yapıldı ve her deneme, modelin performansını değerlendirerek en iyi sonucu elde etmeye çalıştı. İlk başta birkaç deneme düşük performans gösterdi; ancak ilerleyen denemelerde, parametre

ayarlamalarıyla birlikte modelin doğruluk değeri artış gösterdi. Sonuç olarak, en iyi performansı sağlayan deneme, 0.8278 doğruluk değeri ile 27. deneme oldu. Bu denemede kullanılan parametreler arasında 'n_units1', 'n_units2', ve 'n_units3' gibi katman büyüklükleri, 'optimizer' ve 'learning_rate' gibi öğrenme parametreleri ve 'dropout' oranı gibi düzenleme teknikleri yer aldı. Bu optimizasyon süreci, modelin hiperparametrelerinin en iyi kombinasyonunu bulmayı ve performansı artırmayı amaçladı.

```
def scheduler(epoch, lr):
    return float(lr * tf.math.exp(-0.1))

lr_scheduler = LearningRateScheduler(scheduler)
```

The scheduler function is a custom learning rate scheduling function used in training deep learning models with TensorFlow. It adjusts the learning rate (lr) based on the current epoch number. Specifically, the function decreases the learning rate exponentially by a factor of 0.1 for each epoch, which is implemented using `tf.math.exp(-0.1)`. This approach helps the model converge more effectively by gradually reducing the step size taken during gradient updates. The `LearningRateScheduler` callback, `lr_scheduler`, integrates this scheduling function into the training process, allowing the learning rate to be dynamically updated according to the defined exponential decay formula as the training progresses.

```
unit1, unit2, unit3, optimizer, learning_rate, w0, w1 =
(study.best_params['n_units1'],
 study.best_params['n_units2'],
 study.best_params['n_units3'],
 study.best_params['optimizer'],
 study.best_params['learning_rate'],
 study.best_params['w0'],
 study.best_params['w1'])

model = Sequential()
model.add(Dense(unit1, activation="relu",
               input_shape=(X_train.shape[1],)))
model.add(Dense(unit2, activation="relu"))
model.add(Dense(unit3, activation="relu"))
model.add(Dense(1, activation="sigmoid"))

opt = optimizer(learning_rate=learning_rate)

model.compile(optimizer=opt, loss="binary_crossentropy",
              metrics=["Recall"])

history = model.fit(X_train_resampled,
```

```
y_train_resampled,  
validation_split=0.1,  
batch_size=512,  
epochs=100,  
callbacks=[early_stop, lr_scheduler],  
class_weight={0: w0, 1: w1},  
verbose=1)  
  
Epoch 1/100  
27/27 ━━━━━━━━━━ 3s 15ms/step - Recall: 0.0140 - loss:  
0.0296 - val_Recall: 0.2987 - val_loss: 1.2562 - learning_rate: 0.0064  
Epoch 2/100  
27/27 ━━━━━━━━ 0s 5ms/step - Recall: 0.3859 - loss: 0.0187  
- val_Recall: 0.8340 - val_loss: 0.4601 - learning_rate: 0.0058  
Epoch 3/100  
27/27 ━━━━━━━━ 0s 7ms/step - Recall: 0.8020 - loss: 0.0104  
- val_Recall: 0.9209 - val_loss: 0.3239 - learning_rate: 0.0052  
Epoch 4/100  
27/27 ━━━━━━━━ 0s 8ms/step - Recall: 0.8890 - loss: 0.0081  
- val_Recall: 0.9092 - val_loss: 0.3582 - learning_rate: 0.0047  
Epoch 5/100  
27/27 ━━━━━━━━ 0s 7ms/step - Recall: 0.9037 - loss: 0.0072  
- val_Recall: 0.9203 - val_loss: 0.3369 - learning_rate: 0.0043  
Epoch 6/100  
27/27 ━━━━━━━━ 0s 7ms/step - Recall: 0.9122 - loss: 0.0067  
- val_Recall: 0.9196 - val_loss: 0.3376 - learning_rate: 0.0039  
Epoch 7/100  
27/27 ━━━━━━━━ 0s 8ms/step - Recall: 0.9131 - loss: 0.0064  
- val_Recall: 0.9209 - val_loss: 0.3339 - learning_rate: 0.0035  
Epoch 8/100  
27/27 ━━━━━━━━ 0s 8ms/step - Recall: 0.9163 - loss: 0.0062  
- val_Recall: 0.9203 - val_loss: 0.3304 - learning_rate: 0.0032  
Epoch 9/100  
27/27 ━━━━━━━━ 0s 7ms/step - Recall: 0.9181 - loss: 0.0060  
- val_Recall: 0.9209 - val_loss: 0.3291 - learning_rate: 0.0029  
Epoch 10/100  
27/27 ━━━━━━━━ 0s 7ms/step - Recall: 0.9202 - loss: 0.0058  
- val_Recall: 0.9242 - val_loss: 0.3222 - learning_rate: 0.0026  
Epoch 11/100  
27/27 ━━━━━━━━ 0s 9ms/step - Recall: 0.9220 - loss: 0.0056  
- val_Recall: 0.9275 - val_loss: 0.3184 - learning_rate: 0.0023  
Epoch 12/100  
27/27 ━━━━━━━━ 0s 9ms/step - Recall: 0.9236 - loss: 0.0055  
- val_Recall: 0.9281 - val_loss: 0.3133 - learning_rate: 0.0021  
Epoch 13/100  
27/27 ━━━━━━━━ 0s 8ms/step - Recall: 0.9247 - loss: 0.0054  
- val_Recall: 0.9281 - val_loss: 0.3108 - learning_rate: 0.0019  
Epoch 14/100  
27/27 ━━━━━━━━ 0s 9ms/step - Recall: 0.9248 - loss: 0.0053  
- val_Recall: 0.9294 - val_loss: 0.3066 - learning_rate: 0.0017
```

```

Epoch 15/100
27/27 ————— 0s 7ms/step - Recall: 0.9259 - loss: 0.0052
- val_Recall: 0.9307 - val_loss: 0.3027 - learning_rate: 0.0016
Epoch 15: early stopping
Restoring model weights from the end of the best epoch: 1.

```

Verilen kod, TensorFlow ve Keras kullanarak bir sinir ağı modelinin eğitim sürecini göstermektedir. Model mimarisi, ReLU aktivasyon fonksiyonuna sahip üç yoğun (dense) katman ve ikili sınıflandırma için sigmoid aktivasyonlu bir çıkış katmanından oluşur. Optimizer, çalışmanın en iyi parametreleri temel alınarak seçilir ve öğrenme oranı, üssel bir azalma programlayıcısı tarafından dinamik olarak ayarlanır. Model, aşırı uyumu önlemek amacıyla erken durdurma ile birlikte 100 saatे kadar eğitim alır. Eğitim sırasında, hem eğitim hem de doğrulama veri kümeleri için Recall ve kayıp gibi metrikler izlenir. Eğitim, doğrulama performansında belirgin bir iyileşme olmadığından erken olarak 15. epokta durur ve model ağırlıkları en iyi epoktaki değerlerine geri yüklenir.

```

y_pred = (model.predict(X_test) > 0.5).astype("int32")
print(confusion_matrix(y_test, y_pred))
print(classification_report(y_test, y_pred))

57/57 ————— 1s 6ms/step
[[1468 32]
 [ 198 101]]
      precision    recall   f1-score   support
          0       0.88     0.98     0.93    1500
          1       0.76     0.34     0.47     299
      accuracy                           0.87    1799
     macro avg       0.82     0.66     0.70    1799
weighted avg       0.86     0.87     0.85    1799

```

Modelin test verileri üzerindeki performansı, tahmin sonuçlarının confusion_matrix ve classification_report fonksiyonlarıyla değerlendirildi. Karışıklık matrisine göre, model 1468 negatif örneği doğru şekilde sınıflandırmış ve 32 negatif örneği yanlışlıkla pozitif olarak sınıflandırmış. Pozitif örneklerde ise 101 örneği doğru, 198 örneği ise yanlış sınıflandırmıştır. classification_report sonucunda, negatif sınıf için yüksek bir doğruluk (precision) ve geri çağrıma (recall) değerleri gözlemlenirken, pozitif sınıf için daha düşük geri çağrıma ve doğruluk değerleri elde edilmiştir. Genel olarak, modelin doğruluğu %87 olarak belirlenmiştir. Makro ortalama ve ağırlıklı ortalama metrikler de sırasıyla %66 ve %85 gibi değerler sunmaktadır, bu da modelin genel performansının yanı sıra sınıf dengesizliğinin etkilerini de yansıtır.

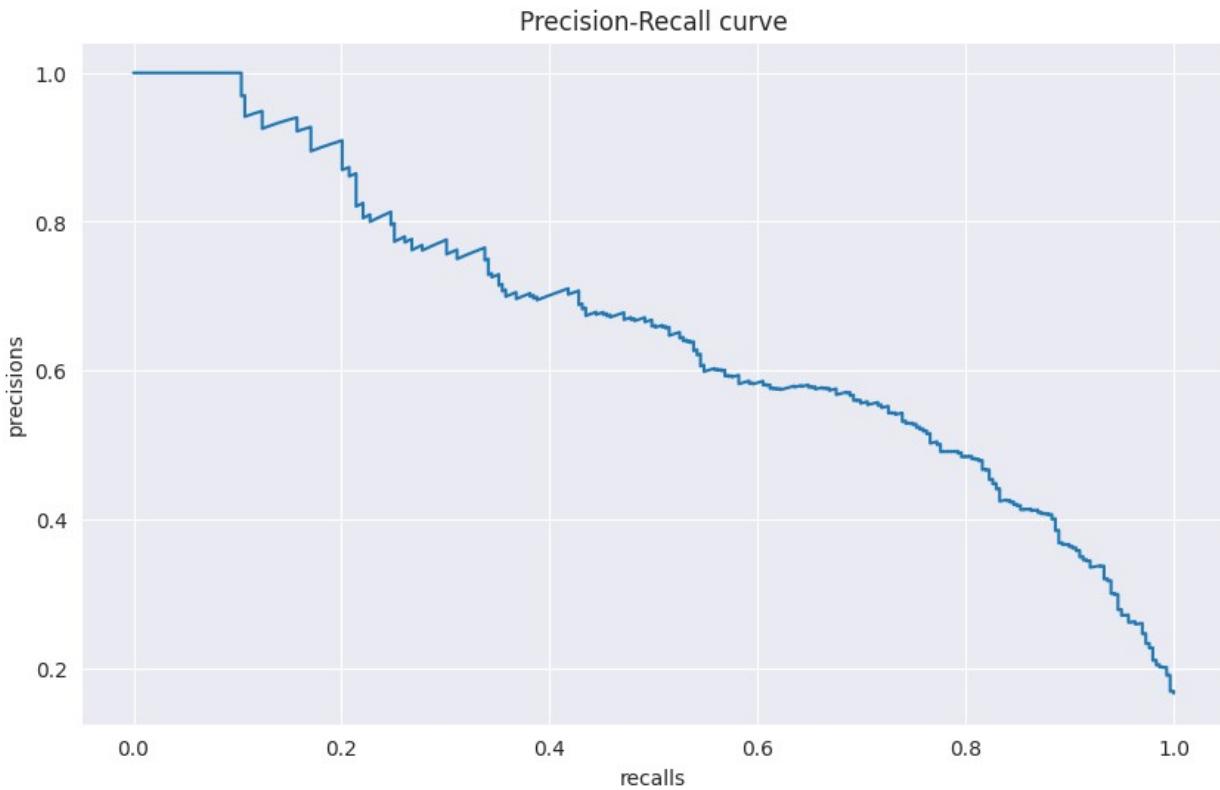
```

y_pred_proba = model.predict(X_test)
precisions, recalls, thresholds = precision_recall_curve(y_test,
y_pred_proba)
plt.plot(recalls, precisions, label='ANN')
plt.xlabel('recalls')
plt.ylabel('precisions')

```

```
plt.title('Precision-Recall curve')
plt.show()
```

57/57 ━━━━━━ 0s 2ms/step



Test verileri üzerinde modelin performansını daha ayrıntılı olarak değerlendirmek amacıyla bir Precision-Recall (P-R) eğrisi çizildi. `precision_recall_curve` fonksiyonu, modelin tahmin ettiği olasılıkları kullanarak her bir eşik değeri için hassasiyet (precision) ve geri çağırma (recall) değerlerini hesapladı. Sonuçlar `matplotlib` ile görselleştirildi; grafikte geri çağırma (recall) değerleri x ekseninde, hassasiyet (precision) değerleri y ekseninde gösterildi. Grafikte görülen eğri, modelin farklı eşik değerlerinde nasıl performans gösterdiğini ortaya koyarak, çeşitli eşiklerde elde edilen hassasiyet ve geri çağırma oranlarını karşılaştırmayı sağladı. Bu, modelin sınıflandırma performansının kapsamlı bir analizini sunarak, farklı durumlar için modelin güçlü ve zayıf yönlerini değerlendirmeye yardımcı olur.

```
average_precision_score(y_test, y_pred_proba)
0.6566664590362162
```

`average_precision_score` fonksiyonu, modelin precision-recall eğrisinin ortalama değerini hesaplar ve genellikle sınıflandırma performansını özetleyen bir metrik olarak kullanılır. Bu durumda, modelin ortalama hassasiyeti (average precision) 0.657 olarak hesaplandı. Bu değer, modelin pozitif sınıfları doğru şekilde tanımlama yeteneğini gösterir; yani, modelin pozitif sınıflar için elde ettiği hassasiyetin genel bir ortalamasıdır. 0.657'lik bir skor, modelin pozitif sınıfları tespit etme ve doğru sınıflandırma konusundaki genel başarısını yansıtır, ancak performansın

daha iyi veya daha kötü olduğunu değerlendirmek için bu skoru diğer modellerle karşılaştırmak faydalı olabilir.

```
DL_AP_smote = average_precision_score(y_test, y_pred_proba)
DL_f1_smote = f1_score(y_test, y_pred)
DL_rec_smote = recall_score(y_test, y_pred)
```

Bu kod parçası, modelin çeşitli performans metriklerini hesaplamaktadır. `average_precision_score(y_test, y_pred_proba)` fonksiyonu, modelin ortalama hassasiyetini (average precision) 0.657 olarak hesaplar, bu da modelin pozitif sınıfları doğru şekilde tanımlama yeteneğini özetler. `f1_score(y_test, y_pred)` fonksiyonu, modelin F1 skorunu hesaplar; bu, hem hassasiyeti (precision) hem de hatırlama oranını (recall) dikkate alan bir performans ölçütüdür ve genellikle sınıflandırma problemlerinde denge sağlamak için kullanılır. `recall_score(y_test, y_pred)` fonksiyonu ise modelin hatırlama oranını (recall) hesaplar, yani modelin gerçek pozitif örnekleri doğru bir şekilde tespit etme oranını gösterir. Bu metrikler birlikte, modelin genel performansını ve özellikle pozitif sınıfları ne kadar iyi tanıdığını değerlendirmek için kullanılır.

Evaluating Model Performance and Tuning

- Compare model performances according to metrics you choose for the problem.

```
compare = pd.DataFrame({ "Model": ["Logistic Regression", "KNN",
"Random Forest",
                    "XGBoost", "DL", "DL with Optuna",
"DL with Optuna-Smote"],

                    "F1_Score": [logistic_f1, KNN_f1, rf_f1,
xgb_f1, DL_f1, DL_f1_op, DL_f1_smote],

                    "Recall_Score": [logistic_recall, KNN_recall,
rf_recall, xgb_recall, DL_rec, DL_rec_op, DL_rec_smote],

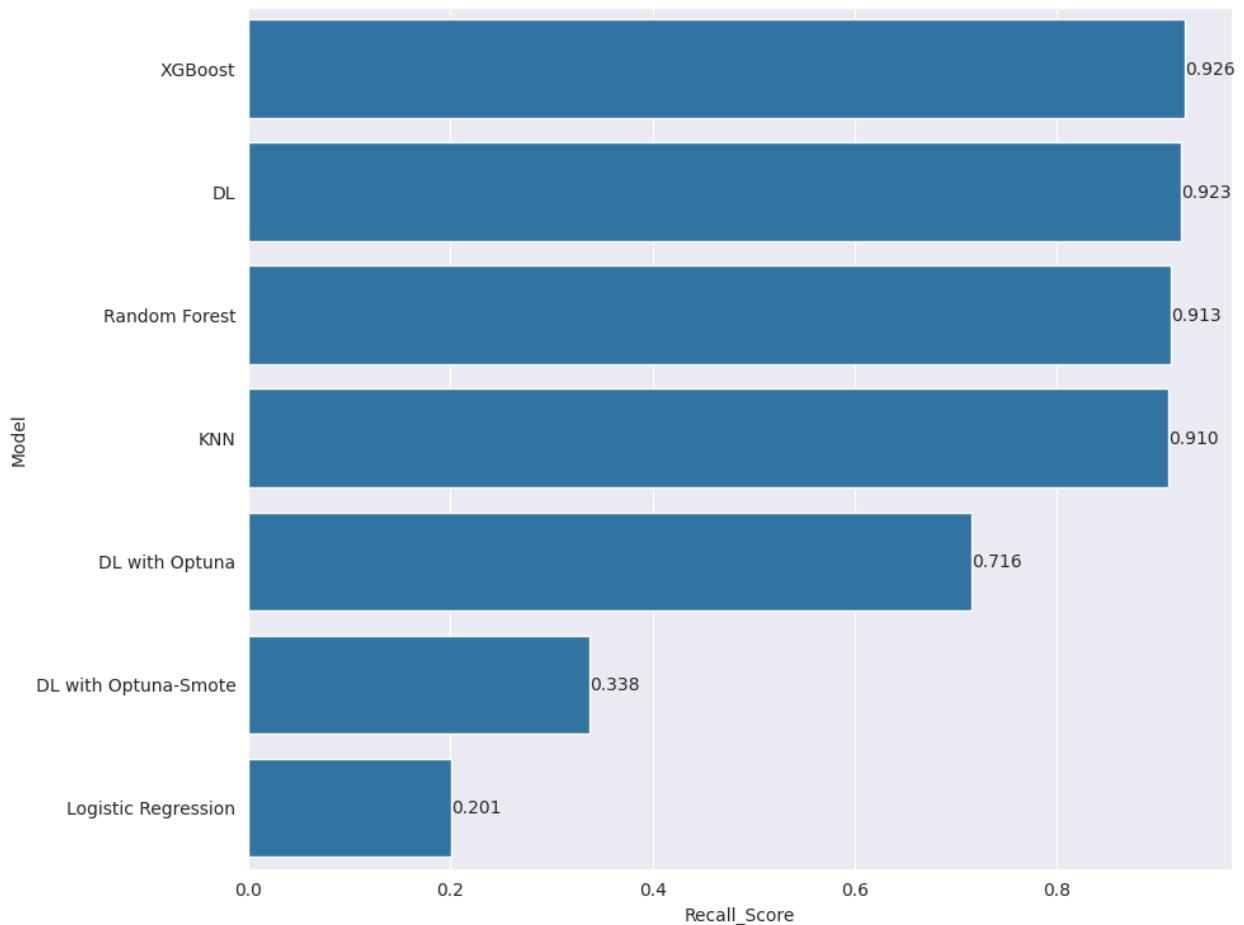
                    "Average_Precision_Score": [logistic_AP,
KNN_AP, rf_AP, xgb_AP, DL_AP, DL_AP_op, DL_AP_smote]})

plt.figure(figsize=(10,30))
plt.subplot(311)
compare = compare.sort_values(by="Recall_Score",
                             ascending=False)
ax=sns.barplot(x="Recall_Score",
                y="Model",
                data=compare)
ax.bar_label(ax.containers[0], fmt=".3f")

plt.subplot(312)
compare = compare.sort_values(by="F1_Score",
                             ascending=False)
ax=sns.barplot(x="F1_Score",
                y="Model",
```

```
    data=compare)
ax.bar_label(ax.containers[0], fmt=".3f")

plt.subplot(313)
compare = compare.sort_values(by="Average_Precision_Score",
                             ascending=False)
ax=sns.barplot(x="Average_Precision_Score",
                y="Model",
                data=compare)
ax.bar_label(ax.containers[0], fmt=".3f")
plt.show();
```



Bu kod parçası, çeşitli makine öğrenme modellerinin performans metriklerini karşılaştırmak için bir grafik oluşturmaktadır. Öncelikle, farklı modellerin F1 skoru, hatırlama oranı (recall) ve ortalama hassasiyet (average precision) skorlarını içeren bir veri çerçevesi (compare) oluşturulur. Bu veri çerçevesi, her bir model için bu metriklerin değerlerini içerir. Daha sonra, bu metriklerin her biri için ayrı ayrı sıralanmış bar grafikler çizilir. İlk grafik, hatırlama oranlarına göre sıralanmış modelleri gösterir, ikinci grafik F1 skorlarına göre sıralanmış modelleri ve üçüncü grafik ise ortalama hassasiyet skorlarına göre sıralanmış modelleri gösterir. Her grafikte, modelin performansını temsil eden barların üzerine metrik değerleri eklenir, bu da performansın görsel olarak daha anlaşıllır olmasını sağlar. Grafikler, modellerin çeşitli performans kriterlerine göre nasıl karşılaştırıldığını görsel olarak sunar.

Prediction of Random Forest

```
X = df.drop(columns=["left"])
y = df['left']
```

Bu kod parçası, bir pandas DataFrame'den (df) özelliklerin ve hedef değişkenin ayrılmasını gerçekleştiren iki satır içerir.

X = df.drop(columns=["left"]): Bu satır, df DataFrame'indeki "left" adlı sütunu hariç tutarak yeni bir DataFrame (X) oluşturur. Bu, modelleme sürecinde kullanılacak özellikler (veya bağımsız değişkenler) kümesini temsil eder. Yani, "left" sütunu hariç tüm sütunlar X değişkenine dahil edilir.

y = df['left']: Bu satır, "left" adlı sütunu seçerek bir pandas Series (y) oluşturur. Bu sütun, modelin tahmin etmeye çalışacağı hedef değişkeni (veya bağımlı değişken) temsil eder. Yani, y, modelin tahmin edeceği veya açıklamaya çalışacağı değerleri içerir.

Bu iki satır, makine öğrenmesi modelleme sürecinde gerekli olan veri hazırlama adımlarını temsil eder: X özellikleri içerirken, y hedef değişkeni içerir.

```
X.head(2)
```

```
{"summary": {"name": "X", "rows": 11991, "fields": [
    {"column": "satisfaction_level", "properties": {"dtype": "number", "std": 0.24107000117011218, "min": 0.09, "max": 1.0, "num_unique_values": 92, "samples": [0.13, 0.15, 0.17, 0.19, 0.21, 0.23, 0.25, 0.27, 0.29, 0.31, 0.33, 0.35, 0.37, 0.39, 0.41, 0.43, 0.45, 0.47, 0.49, 0.51, 0.53, 0.55, 0.57, 0.59, 0.61, 0.63, 0.65, 0.67, 0.69, 0.71, 0.73, 0.75, 0.77, 0.79, 0.81, 0.83, 0.85, 0.87, 0.89, 0.91, 0.93, 0.95, 0.97, 0.99], "semantic_type": "number", "description": "The satisfaction level of the customer.", "last_evaluation": {"properties": {"dtype": "number", "std": 0.16834256307406967, "min": 0.36, "max": 1.0, "num_unique_values": 65, "samples": [0.66, 0.68, 0.7, 0.72, 0.74, 0.76, 0.78, 0.8, 0.82, 0.84, 0.86, 0.88, 0.9, 0.92, 0.94, 0.96, 0.98, 1.0], "semantic_type": "number", "description": "The last evaluation score of the employee."}, "semantic_type": "number", "description": "The number of projects the employee has worked on."}, "properties": {"dtype": "number", "std": 1, "min": 2, "max": 7, "num_unique_values": 6, "samples": [3, 4, 5, 6, 7], "semantic_type": "number", "description": "The number of projects the employee has worked on."}, "semantic_type": "number", "description": "The number of projects the employee has worked on."}]}}
```

```

    "average_monthly_hours",\n          "properties": {\n        "dtype": "number",\n          "std": 48,\n          "min": 96,\n        "max": 310,\n          "num_unique_values": 215,\n        "samples": [\n          118,\n          112,\n          222\n        ],\n          "semantic_type": "\",\n          "description": \"\"\n      },\n      {\n        "column": "time_spend_company",\n        "properties": {\n          "dtype": "number",\n            "std": 1,\n            "min": 2,\n            "max": 10,\n          "num_unique_values": 8,\n            "samples": [\n              6,\n              8,\n              3\n            ],\n            "semantic_type": "\",\n            "description": \"\"\n          },\n          {\n            "column": "work_accident",\n            "properties": {\n              "dtype": "number",\n                "std": 0,\n                "min": 0,\n              "max": 1,\n                "num_unique_values": 2,\n                  "samples": [\n                    1,\n                    0\n                  ],\n                  "semantic_type": "\",\n                  "description": \"\"\n                },\n                {\n                  "column": "promotion_last_5years",\n                  "properties": {\n                    "dtype": "number",\n                      "std": 0,\n                      "min": 0,\n                    "max": 1,\n                      "num_unique_values": 2,\n                        "samples": [\n                          1,\n                          0\n                        ],\n                        "semantic_type": "\",\n                        "description": \"\"\n                      },\n                      {\n                        "column": "departments",\n                        "properties": {\n                          "dtype": "category",\n                            "num_unique_values": 10,\n                          "samples": [\n                            "marketing",\n                            "accounting"
                          ],\n                          "semantic_type": "\",\n                          "description": \"\"\n                        },\n                        {\n                          "column": "salary",\n                          "properties": {\n                            "dtype": "category",\n                              "num_unique_values": 3,\n                              "samples": [\n                                "low",\n                                "medium"
                              ],\n                              "semantic_type": "\",\n                              "description": \"\"\n                            }
                          ]\n                        }
                      ]
                    }
                  }
                ]
              }
            }
          }
        }
      }
    }
  }
}
,"type":"dataframe","variable_name":"X"}

```

X.head(2) komutu, X DataFrame'indeki ilk iki satırı gösterir ve bu, veri kümelerindeki özellikleri (bağımsız değişkenler) içerir. Bu örnekte, X DataFrame'i şu sütunlardan oluşur: satisfaction_level, last_evaluation, number_project, average_monthly_hours, time_spend_company, work_accident, promotion_last_5years, departments, ve salary. İlk iki satırda, çalışanların iş tatmin düzeyleri, performans değerlendirmeleri, projelerin sayısı, aylık çalışma saatleri, şirkette geçirdiği süre, iş kazası durumu, son 5 yılda terfi durumu, departmanları ve maaş seviyeleri yer almaktadır. Bu bilgiler, çalışanların iştan ayrılma durumunu tahmin etmek amacıyla kullanılacak özelliklerdir.

```

# Define your feature columns
numeric_features = ['satisfaction_level', 'last_evaluation',
'number_project',
'average_monthly_hours', 'time_spend_company',
'work_accident',
'promotion_last_5years'] # numeric columns
cat_onehot = ['departments'] # categorical column for OneHotEncoder
cat_ordinal = ['salary'] # categorical column for OrdinalEncoder

```

```

# Define your preprocessing steps
numeric_transformer = StandardScaler() # for numeric columns
onehot_transformer = OneHotEncoder() # for departments
(OneHotEncoding)
ordinal_transformer = OrdinalEncoder(categories=[['low', 'medium',
'high']]) # for salary (OrdinalEncoding)

# Combine preprocessing for both numeric, one-hot categorical, and
ordinal categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat_onehot', onehot_transformer, cat_onehot),
        ('cat_ordinal', ordinal_transformer, cat_ordinal)])

# Pipeline tanımı
rf_final = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('classifier', RandomForestClassifier())
])

# Modelin eğitilmesi
rf_final.fit(X, y)

Pipeline(steps=[('preprocessor',
                 ColumnTransformer(transformers=[('num',
StandardScaler(),

['satisfaction_level',
                     'last_evaluation',
                     'number_project',
                     'average_monthly_hours',
                     'work_accident',
                     'time_spend_company',
                     'promotion_last_5years']),
                     ('cat_onehot',
OneHotEncoder(),
                     ['departments']),
                     ('cat_ordinal',
                     OrdinalEncoder(categories=[['low',
                     'medium',
                     'high']])),
                     ])])

```

```
[ 'salary']))),  
('classifier', RandomForestClassifier()))])
```

Bu kod parçası, makine öğrenmesi modelinin özelliklerin ön işlenmesini ve model eğitimi sürecini tanımlar. İlk olarak, numeric_features ve cat_onehot, cat_ordinal listeleri ile sayısal ve kategorik sütunları ayırır. Sayısal sütunlar için StandardScaler kullanılarak veriler ölçeklenir, departments sütunu için OneHotEncoder ile one-hot kodlaması yapılır, ve salary sütunu için OrdinalEncoder ile sıralı kodlama uygulanır. Bu ön işleme adımları, ColumnTransformer içinde bir araya getirilir. Son olarak, bu ön işleme adımlarını ve RandomForestClassifier modelini içeren bir Pipeline oluşturulur. rf_final isimli bu pipeline ile tüm veri seti fit fonksiyonu kullanılarak eğitilir, böylece model sayısal ve kategorik özellikler için uygun dönüşümleri uygulayarak öğrenme işlemini gerçekleştirir.

```
df.describe().T
```

```
{"summary": {"name": "df", "rows": 8, "fields": [{"column": "count", "properties": {"dtype": "number", "std": 0.0, "min": 11991.0, "max": 11991.0, "num_unique_values": 1, "samples": [11991.0], "semantic_type": "\\", "description": "\n"}, "column": "mean", "properties": {"dtype": "number", "std": 70.44680707451664, "min": 0.016929363689433742, "max": 200.4735218080227, "num_unique_values": 8, "samples": [0.7166825118839131], "semantic_type": "\\", "description": "\n"}, "column": "std", "properties": {"dtype": "number", "std": 17.043867937645068, "min": 0.12901220247678255, "max": 48.727813191371325, "num_unique_values": 8, "samples": [0.16834256307406967], "semantic_type": "\\", "description": "\n"}, "column": "min", "properties": {"dtype": "number", "std": 33.72761138359574, "min": 0.0, "max": 96.0, "num_unique_values": 5, "samples": [0.36], "semantic_type": "\\", "description": "\n"}, "column": "25%", "properties": {"dtype": "number", "std": 55.16663715831465, "min": 0.0, "max": 157.0, "num_unique_values": 5, "samples": [0.57], "semantic_type": "\\", "description": "\n"}, "column": "50%", "properties": {"dtype": "number", "std": 70.30360278311285, "min": 0.0, "max": 200.0, "num_unique_values": 6, "samples": [0.66], "semantic_type": "\\", "description": "\n"}, "column": "75%", "properties": {"dtype": "number", "std": 85.39590087184679, "min": 0.0, "max": 250.0, "num_unique_values": 6, "samples": [0.85], "semantic_type": "\\", "description": "\n"}]}
```

```

  "min": 0.0, "max": 243.0, "num_unique_values": 6,
  "samples": [0.82], "semantic_type": "number",
  "description": "The column has a maximum value of 243.0, a mean of 0.82, and a standard deviation of 108.54623767909099. The minimum value is 1.0, and the maximum value is 310.0. There are 4 unique values in the samples.", "dtype": "number",
  "properties": {
    "min": 1.0, "max": 310.0, "num_unique_values": 4,
    "samples": [7.0], "semantic_type": "number",
    "description": "The column has a maximum value of 310.0, a mean of 7.0, and a standard deviation of 108.54623767909099. The minimum value is 1.0, and the maximum value is 243.0. There are 4 unique values in the samples." }
}, "type": "dataframe"

```

Bu tablo, veri setindeki çeşitli özelliklerin temel istatistiklerini özetler. satisfaction_level ve last_evaluation sütunları, sırasıyla 0.63 ve 0.72 ortalama değerine sahiptir ve her biri belirli bir aralıktaki değişkenlik göstermektedir. number_project, ortalama olarak 3.80 proje içerirken, average_monthly_hours sütunu aylık ortalama çalışma saatlerini gösterir ve ortalaması 200.47 saat ile geniş bir aralığa sahiptir. time_spend_company sütunu, çalışanların şirkette geçirdiği süreyi ortalama 3.36 yıl olarak belirtir. work_accident ve promotion_last_5years sütunları, sırasıyla iş kazası geçirme oranını (0.15) ve son 5 yılda terfi alma oranını (0.02) düşük bir seviyede gösterir. Son olarak, left sütunu, işten ayrılma oranını belirtir ve ortalama değeri 0.17'dir. Bu özet, veri setindeki özelliklerin dağılımı ve değişkenliği hakkında genel bir bakış sağlar.

```

df.columns
Index(['satisfaction_level', 'last_evaluation', 'number_project',
       'average_monthly_hours', 'time_spend_company', 'work_accident',
       'left',
       'promotion_last_5years', 'departments', 'salary'],
      dtype='object')

```

Bu liste, veri setindeki sütun adlarını belirtir. Sütunlar şunlardır:

satisfaction_level: Çalışanların memnuniyet seviyelerini ölçen bir sütun. last_evaluation: Çalışanların son performans değerlendirmesi sonuçlarını gösteren bir sütun. number_project: Her çalışanın üzerinde çalıştığı proje sayısını belirten bir sütun. average_monthly_hours: Çalışanların aylık ortalama çalışma saatlerini gösteren bir sütun. time_spend_company: Çalışanların şirkette geçirdiği süreyi yıllık olarak ifade eden bir sütun. work_accident: Çalışanların iş kazası geçirme durumunu belirten bir sütun (0 veya 1 değeri alır). left: Çalışanın işten ayrılmış olduğunu gösteren hedef değişken (0: ayrılmadı, 1: ayrıldı). promotion_last_5years: Son 5 yıl içinde terfi alma durumunu belirten bir sütun (0 veya 1 değeri alır). departments: Çalışanın çalıştığı departmanı belirten kategorik bir sütun. salary: Çalışanın maaş seviyesini ifade eden kategorik bir sütun (örneğin, düşük, orta, yüksek). Bu sütunlar, çalışanların memnuniyeti, performansı ve işten ayrılma durumunu anlamaya yönelik çeşitli özellikler sunar.

```

df.departments.unique()
array(['sales', 'accounting', 'hr', 'technical', 'support',
       'management',
       'IT', 'product_mng', 'marketing', 'RandD'], dtype=object)

```

`df.departments.unique()` komutu, veri setindeki departments sütunundaki benzersiz departman isimlerini listeler. Bu liste şu departmanları içerir:

sales: Satış departmanı. accounting: Muhasebe departmanı. hr: İnsan kaynakları departmanı. technical: Teknik departman. support: Destek departmanı. management: Yönetim departmanı. IT: Bilgi teknolojileri departmanı. product_mng: Ürün yönetimi departmanı. marketing: Pazarlama departmanı. RandD: Araştırma ve geliştirme departmanı. Bu departman isimleri, çalışanların hangi bölümlerde görev yaptığını gösterir ve departmanlar arasındaki farklılıklar analiz etmek için kullanılabilir.

```
df.sample(n=2)

{"summary": {"name": "df", "rows": 2, "fields": [
    {"column": "satisfaction_level", "properties": {
        "dtype": "number", "std": 0.028284271247461888, "min": 0.46, "max": 0.5, "num_unique_values": 2, "samples": [0.46, 0.5], "semantic_type": "\\"}, "description": "\n"}, {"column": "last_evaluation", "properties": {
        "dtype": "number", "std": 0.1626345596729059, "min": 0.52, "max": 0.75, "num_unique_values": 2, "samples": [0.52, 0.75], "semantic_type": "\\"}, "description": "\n"}, {"column": "number_project", "properties": {
        "dtype": "number", "std": 0, "min": 5, "max": 6, "num_unique_values": 2, "samples": [5, 6], "semantic_type": "\\"}, "description": "\n"}, {"column": "average_monthly_hours", "properties": {
        "dtype": "number", "std": 98, "min": 137, "max": 276, "num_unique_values": 2, "samples": [276, 137], "semantic_type": "\\"}, "description": "\n"}, {"column": "time_spend_company", "properties": {
        "dtype": "number", "std": 2, "min": 3, "max": 6, "num_unique_values": 2, "samples": [3, 6], "semantic_type": "\\"}, "description": "\n"}, {"column": "work_accident", "properties": {
        "dtype": "number", "std": 0, "min": 0, "max": 1, "num_unique_values": 2, "samples": [0, 1], "semantic_type": "\\"}, "description": "\n"}, {"column": "left", "properties": {
        "dtype": "number", "std": 0, "min": 0, "max": 0, "num_unique_values": 1, "samples": [0], "semantic_type": "\\"}, "description": "\n"}]}}, {"name": "df", "rows": 2, "fields": [
    {"column": "satisfaction_level", "properties": {
        "dtype": "number", "std": 0.028284271247461888, "min": 0.46, "max": 0.5, "num_unique_values": 2, "samples": [0.46, 0.5], "semantic_type": "\\"}, "description": "\n"}, {"column": "last_evaluation", "properties": {
        "dtype": "number", "std": 0.1626345596729059, "min": 0.52, "max": 0.75, "num_unique_values": 2, "samples": [0.52, 0.75], "semantic_type": "\\"}, "description": "\n"}, {"column": "number_project", "properties": {
        "dtype": "number", "std": 0, "min": 5, "max": 6, "num_unique_values": 2, "samples": [5, 6], "semantic_type": "\\"}, "description": "\n"}, {"column": "average_monthly_hours", "properties": {
        "dtype": "number", "std": 98, "min": 137, "max": 276, "num_unique_values": 2, "samples": [276, 137], "semantic_type": "\\"}, "description": "\n"}, {"column": "time_spend_company", "properties": {
        "dtype": "number", "std": 2, "min": 3, "max": 6, "num_unique_values": 2, "samples": [3, 6], "semantic_type": "\\"}, "description": "\n"}, {"column": "work_accident", "properties": {
        "dtype": "number", "std": 0, "min": 0, "max": 1, "num_unique_values": 2, "samples": [0, 1], "semantic_type": "\\"}, "description": "\n"}, {"column": "left", "properties": {
        "dtype": "number", "std": 0, "min": 0, "max": 0, "num_unique_values": 1, "samples": [0], "semantic_type": "\\"}, "description": "\n"}]}]
```

```

    "description": """\n      }\n    },\n    {\n      \"column\":\n        \"promotion_last_5years\",\\n      \"properties\": {\n        \"dtype\": \"number\",\\n        \"std\": 0,\\n        \"min\": 0,\\n        \"max\": 0,\\n        \"num_unique_values\": 1,\\n        \"samples\": [\n          0\\n        ],\\n        \"semantic_type\": \"\",\\n      }\n    },\n    {\n      \"column\":\n        \"departments\",\\n      \"properties\": {\n        \"dtype\": \"string\",\\n        \"num_unique_values\": 2,\\n        \"samples\": [\n          \"support\\n        ],\\n        \"semantic_type\": \"\",\\n      }\n    },\n    {\n      \"column\":\n        \"salary\",\\n      \"properties\": {\n        \"dtype\": \"string\",\\n        \"num_unique_values\": 2,\\n        \"samples\": [\n          \"low\\n        ],\\n        \"semantic_type\": \"\",\\n      }\n    }\n  }\\n}","type":"dataframe"

```

df.sample(n=2) komutu, veri setinden rastgele iki satır seçerek örnek bir görüntü sağlar. Bu örnekte, iki rastgele çalışan kaydı gösterilmektedir. İlk çalışan, technical (teknik) departmanında çalışıyor ve orta düzeyde bir maaş alırken, ikinci çalışan support (destek) departmanında çalışıyor ve düşük düzeyde bir maaş alıyor. İki çalışanın da tatmin seviyeleri, son değerlendirmeleri, proje sayıları, ortalama aylık çalışma saatleri, şirkette geçirilen süre, iş kazası geçirme durumu ve son beş yılda terfi durumları farklılık göstermektedir. Bu örnek, veri setindeki farklı çalışan profillerini anlamaya yardımcı olabilir.

```

observations = {
    "satisfaction_level": [0.83, 0.42], # Örnek değerler
    "last_evaluation": [0.98, 0.46],
    "number_project": [5, 2],
    "average_monthly_hours": [187, 147],
    "time_spend_company": [4, 3],
    "work_accident": [0, 0],
    "promotion_last_5years": [0, 0],
    "departments": ["hr", "sales"], # Kategorik veriler
    "salary": ["medium", "low"]
}

# DataFrame formatına çevirin
obs = pd.DataFrame(observations)

```

observations adlı sözlük, iki yeni gözlem kaydını temsil eden örnek veriler içerir. Her gözlem için tatmin düzeyi, son değerlendirme notu, proje sayısı, ortalama aylık çalışma saatleri, şirkette geçirilen süre, iş kazası durumu, son beş yılda terfi durumu, departman ve maaş düzeyi gibi özellikler verilmiştir. Bu sözlük, pandas kütüphanesi kullanılarak pd.DataFrame fonksiyonu ile bir DataFrame'e dönüştürülmüştür. Sonuç olarak, obs adında bir DataFrame oluşturulur ve bu DataFrame, iki gözlemi içeren sütunlar ve değerleriyle veri analizleri ve model tahminleri için kullanılabilir.

```
# Yeni gözlemler üzerinde tahmin yapın
pred_rf = rf_final.predict(obs)
pred_rf

array([0, 1])
```

Yeni gözlemler üzerinde yapılan tahminler sonucunda, rf_final modelinin iki gözlem için tahmin ettiği sonuçlar sırasıyla 0 ve 1'dir. Bu tahminler, modelin her gözlemin left (işten ayrılma) etiketine dair tahmin ettiği değerlerdir. İlk gözlem için 0 tahmini, işten ayrılma olasılığının düşük olduğunu; ikinci gözlem için 1 tahmini ise işten ayrılma olasılığının yüksek olduğunu belirtir. Bu tahminler, modelin eğitim verileri ve özelliklere dayanarak nasıl sonuçlar üreteceğini gösterir.

```
import pickle

pickle.dump(rf_final, open('rf_model_app', 'wb'))
```

Kodda yer alan pickle.dump(rf_final, open('rf_model_app', 'wb')) ifadesi, rf_final isimli RandomForest modelini bir dosyaya kaydetmek için kullanılır. İşlem şu şekilde gerçekleşir:

pickle.dump: Bu fonksiyon, Python nesnelerini (bu durumda rf_final modelini) bir dosyaya serileştirir ve kaydeder.

open('rf_model_app', 'wb'): open fonksiyonu, 'rf_model_app' isimli bir dosya açar. 'wb' modunda açıldığı için, bu dosya yazma (write) ve ikili (binary) moda olacak, yani model dosyası binary formatında kaydedilecektir.

Sonuç olarak, bu kod parçası rf_final modelini rf_model_app isimli bir dosyaya kaydeder ve bu dosya ileride modelin yeniden yüklenmesi ve kullanılması için saklanabilir. Bu işlem, modelin tekrar eğitilmesine gerek kalmadan doğrudan kullanılabilmesini sağlar.

```
rf_final_new = pickle.load(open('rf_model_app', 'rb'))
```

Bu kod satırı, Python'da bir model dosyasını yeniden yüklemek için kullanılır. pickle modülü, Python objelerini bir dosyaya kaydetmek ve bu dosyadan geri yüklemek için kullanılır. Burada, pickle.load fonksiyonu, 'rf_model_app' isimli dosyadan (bu dosya 'rb' (read binary) modunda açılmış) bir objeyi okur ve bu objeyi rf_final_new değişkenine atar. Genellikle bu tür dosyalar, bir makine öğrenme modelini veya başka bir büyük veri yapısını içerebilir. Bu sayede, model veya veri yeniden eğitilmeden veya yeniden oluşturulmadan doğrudan kullanılabilir.

```
rf_final_new.predict(obs)

array([0, 1])
```

Bu kod parçasığı, rf_final_new isimli makine öğrenme modelinin obs adlı veri gözlemi üzerinde tahminde bulunmasını sağlar. predict metodу, verilen gözlem verileri için modelin tahminlerini döndürür. Sonuç olarak, array([0, 1]) döndürülür, bu da modelin iki farklı tahminde bulunduğu gösterir. Bu iki tahminden biri 0, diğeri ise 1 olarak belirlenmiştir.

Örneğin, eğer model bir sınıflandırma modeliyse, bu durumda 0 ve 1 farklı sınıfları temsil edebilir. obs verisinin her bir gözlemi için model, tahmin edilen sınıfları döndürüyor. Bu tahminler, modelin gözlem verilerini analiz ederek hangi sınıfa ait olduğunu belirlemiştir.

Prediction of XGBoost

```
# Define your feature columns
numeric_features = ['satisfaction_level', 'last_evaluation',
'number_project',
'average_monthly_hours', 'time_spend_company',
'work_accident',
'promotion_last_5years'] # numeric columns
cat_onehot = ['departments'] # categorical column for OneHotEncoder
cat_ordinal = ['salary'] # categorical column for OrdinalEncoder

# Define your preprocessing steps
numeric_transformer = StandardScaler() # for numeric columns
onehot_transformer = OneHotEncoder() # for departments
(OneHotEncoding)
ordinal_transformer = OrdinalEncoder(categories=[['low', 'medium',
'high']]) # for salary (OrdinalEncoding)

# Combine preprocessing for both numeric, one-hot categorical, and
# ordinal categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat_onehot', onehot_transformer, cat_onehot),
        ('cat_ordinal', ordinal_transformer, cat_ordinal)])

xgb_model = XGBClassifier(class_weight="balanced", random_state=101)

operations_xgb = [
    ("preprocessor", preprocessor),
    ("xgb_model", XGBClassifier(class_weight="balanced",
random_state=101)),
]

# Create the pipeline
xgb_final = Pipeline(steps=[
    ('preprocessor', preprocessor),
    ('xgb_model', xgb_model)
])

# Fit the pipeline
xgb_final.fit(X, y)

Pipeline(steps=[('preprocessor',
    ColumnTransformer(transformers=[('num',
StandardScaler(),
```

```

['satisfaction_level',
                         'last_evaluation',
                         'number_project',
'monthly_hours',
'time_spend_company',
                         'work_accident',
'promotion_last_5years']),
                         ('cat_onehot',
OneHotEncoder(),
                         ['departments']),
('cat_ordinal',
OrdinalEncoder(categories=[['low',
'medium',
'high']])),
                         [ 's...
feature_types=None, gamma=None,
grow_policy=None,
                         importance_type=None,
interaction_constraints=None,
learning_rate=None,
                         max_bin=None, max_cat_threshold=None,
max_cat_to_onehot=None,
max_delta_step=None,
                         max_depth=None, max_leaves=None,
min_child_weight=None, missing=nan,
monotone_constraints=None,
multi_strategy=None,
                         n_estimators=None, n_jobs=None,
num_parallel_tree=None, ...))])

```

Bu kod parçası, bir makine öğrenme iş akışını oluşturmak ve eğitmek için kullanılır. İlk olarak, veri özniteliklerini işlemeye yönelik çeşitli adımlar tanımlanır: sayısal öznitelikler için StandardScaler kullanılır, kategorik öznitelikler için OneHotEncoder ve sıralı kategorik öznitelikler için OrdinalEncoder kullanılır. Bu adımlar, ColumnTransformer aracılığıyla bir araya getirilir ve bu işlem veri setindeki sayısal ve kategorik özniteliklere uygun şekilde uygulanır. Daha sonra, bu ön işleme adımlarını ve XGBoost sınıflandırma modelini içeren bir Pipeline oluşturulur. Pipeline, veriyi ön işler ve ardından XGBoost modelini eğitir. Son olarak, xgb_final isimli bu pipeline, fit metodу ile X (özellikler) ve y (hedef değişken) ile eğitilir. Bu yapı, modelin eğitim ve ön işleme adımlarını bir arada ve düzenli bir şekilde gerçekleştirmeyi sağlar.

```

pred_xgb= xgb_final.predict(obs)
pred_xgb

```

```
array([0, 1])
```

Bu kod parçası, xgb_final isimli makine öğrenme pipeline'ının obs adlı veri gözlemi üzerinde tahminde bulunmasını sağlar. predict metodu, pipeline'da tanımlı olan veri ön işleme adımlarını uygular ve ardından XGBoost modelini kullanarak tahminlerde bulunur. Sonuç olarak array([0, 1]) döndürülür; bu, modelin obs gözlemi için iki farklı tahmin ürettiğini gösterir. Bu durumda, model her bir gözlem için 0 veya 1 sınıflarından birini tahmin etmiştir. Tahminler, modelin veriyi analiz ederek hangi sınıfı ait olduğunu belirlediğini gösterir.

```
import pickle  
pickle.dump(xgb_final, open('xgb_model_app', 'wb'))
```

Bu kod parçası, Python'da bir nesneyi dosyaya kaydetmek için pickle modülünü kullanır. pickle.dump fonksiyonu, xgb_final isimli pipeline nesnesini 'xgb_model_app' isimli dosyaya (ikili (binary) modda 'wb' ile açılmış) kaydeder. Bu işlem, xgb_final nesnesinin tüm yapı ve içeriğiyle birlikte dosyada saklanması sağlar. Böylece, bu model ve onun ön işleme adımları, gelecekte bu dosyadan yüklenerek tekrar kullanılabilir, bu da modelin yeniden eğitilmesini gerektirmeden tahminler yapabilmenizi sağlar.

```
xgb_final_new = pickle.load(open('xgb_model_app', 'rb'))
```

Bu kod parçası, daha önce pickle ile kaydedilmiş olan xgb_final isimli pipeline nesnesini dosyadan yükler. pickle.load fonksiyonu, 'xgb_model_app' isimli dosyayı ikili (binary) modda 'rb' ile açar ve dosyada saklanan nesneyi (xgb_final pipeline'ını) xgb_final_new isimli değişkene yükler. Bu işlem, daha önce kaydedilmiş olan modelin ve ön işleme adımlarının yeniden kullanılmasını sağlar, böylece modelin yeniden eğitilmesine gerek kalmadan tahminlerde bulunabilirsiniz.

```
xgb_final_new.predict(obs)  
array([0, 1])
```

Bu kod parçası, xgb_final_new isimli yüklenmiş pipeline nesnesinin obs adlı veri gözlemi üzerinde tahminde bulunmasını sağlar. predict metodu, daha önce kaydedilip yüklenen pipeline'daki ön işleme adımlarını uygular ve ardından XGBoost modelini kullanarak tahminlerde bulunur. Sonuç olarak array([0, 1]) döndürülür; bu, modelin obs gözlemi için iki farklı tahmin ürettiğini gösterir. Yani, modelin tahmin ettiği sınıflar 0 ve 1'dir. Bu, xgb_final_new modelinin veri gözlemi analiz ederek hangi sınıfı ait olabileceği dair tahminler yaptığı anlamına gelir.

6. Model Deployment

You cooked the food in the kitchen and moved on to the serving stage. The question is how do you showcase your work to others? Model Deployment helps you showcase your work to the world and make better decisions with it. But, deploying a model can get a little tricky at times. Before deploying the model, many things such as data storage, preprocessing, model building and monitoring need to be studied. Streamlit is a popular open source framework used by data scientists for model distribution.

Deployment of machine learning models, means making your models available to your other business systems. By deploying models, other systems can send data to them and get their predictions, which are in turn populated back into the company systems. Through machine learning model deployment, you can begin to take full advantage of the model you built.

Data science is concerned with how to build machine learning models, which algorithm is more predictive, how to design features, and what variables to use to make the models more accurate. However, how these models are actually used is often neglected. And yet this is the most important step in the machine learning pipeline. Only when a model is fully integrated with the business systems, real values can be extracted from its predictions.

After doing the following operations in this notebook, jump to new .py file and create your web app with Streamlit.

Model dağıtımları, makine öğrenme modellerinizi iş sistemlerine entegre ederek onlara veri gönderip tahminler almanızı sağlar. Bu, modelinizin iş dünyasında etkili bir şekilde kullanılmasını ve gerçek değerlerin elde edilmesini mümkün kılar. Model dağıtımları aşamasında veri saklama, ön işleme, model oluşturma ve izleme gibi faktörler dikkate alınmalıdır. Streamlit, veri bilimcilerinin modellerini kolayca paylaşabileceği popüler bir açık kaynaklı çerçevedir. Modelinizi dağıttıktan sonra, diğer sistemlerle etkileşime geçebilir ve tahminlerinizi iş süreçlerine dahil edebilirsiniz. Bu, makine öğrenme sürecinin önemli bir aşamasıdır ve modelinizin gerçek dünyada etkili bir şekilde kullanılmasını sağlar.

Save and Export the Best Model