LOTUS AI

YAPAY ZEKA VE BİLİŞİM ÇÖZÜMLERİ A.Ş.

5 MAKİNE ÖĞRENME PROJESİ

HAZIRLAYAN : **Kübra Nur BABACAN**

Stajyer

NİSAN 2025

İÇİNDEKİLER

BÖLÜM 1

KARACİĞER HASTALIĞI OLAN VE OLMAYAN BİREYLERİN ANALİZİ

SINIFLANDIRMA ANALİZİ

1.1. Veri Setinin Durumu / Hikayesi

Hindistan'ın Andhra Pradesh eyaletinin Kuzey Doğu bölgesinden toplanan hasta kayıtları 416 tane karaciğer hastası kaydı ve 167 tane karaciğer hastası olmayan hasta kayıtlarının verisini içermektedir. Bu "Veri Seti" sütunu, karaciğer hastası (karaciğer hastalığı) veya karaciğer hastalığı olmayan (hastalık yok) olacak şekilde gruplara ayırmak için kullanılan bir sınıf etiketi halindedir. Veri seti 441 erkek hasta kaydı ve 142 kadın hasta kaydı içermektedir.

- Yaşı 89'u geçen her hasta "90" yaşında olarak listelendirilmiştir.

Araştırmanın yapılma sebebi Karaciğer hastalığı olan bireyler, aşırı alkol tüketimi, zararlı gazların solunması, kirlenmiş gıda, turşu ve ilaç alımı nedeniyle sürekli olarak artmaktadır. Bu veri seti ile doktorların yükünün azaltılması amacıyla tahmin algoritmaları geliştirilmiştir.

Veri de kullandığımız değişkenleri inceleyecek olursak,

- Hastanın yaş bilgisi
- Hastanın cinsiyet bilgisi
- Toplam Bilirubin bilgisi
- Direkt Bilirubin bilgisi
- Alkali Fosfotaz bilgisi
- Alamine Aminotransferaz bilgisi
- Aspartat Aminotransferaz bilgisi
- Toplam Proteinler bilgisi
- Albümin bilgisi
- Albümin ve Globulin Oranı bilgisi
- Veri kümesi bilgisi
    - Karaciğer hastalığı olan bireyler (1 ile belirtilmiş durumda) ve
    - Karaciğer hastalığı olmayan bireylerin (2 ile belirtilmiş durumda) bilgilerini içermektedir.

Bu veri seti ve bilgiler ile amacımız bireylerde karaciğer hastalığının olup olmadığını değerlendirmektir.

Veri seti kaynağı: https://www.kaggle.com/datasets/uciml/indian-liver-patient-records

1.2. Veri Analiz Platformları

Veri setinin analizi ilk olarak Anaconda uygulamasının içeriğinde bulunan JupyterLab ile Python analizleri gerçekleştirilecektir. Daha sonra aynı veri seti Knıme uygulamasında da analiz edilip tamamlanacaktır.

1.3. Verinin Analiz Durumu

Hindistan'ın Andhra Pradesh eyaletinin Kuzey Doğu bölgesinden toplanan bireylerin bilgileri, çeşitli değişken bilgileri ile bireylerin karaciğer hastalığının olup / olmamasına göre bireylerin hasta durumları kategorik olduğu için bu veri setinin sınıflandırma yöntemi ile en uygun modelinin bulunması hedeflenmektedir.

1.3.1. PYTHON İLE ANALİZ DURUMU VE SONUÇLARI

- **Verinin Yüklenip İlk Ve Son Satırlarının Kontrolü**

```
pip install kagglehub
```

```
...
```

```
import pandas as pd
import kagglehub

# Kaggle ile datasetin yolunu aldık
dataset_path = kagglehub.dataset_download("uciml/indian-liver-patient-records")

# veri setini okutuyoruz
data = pd.read_csv(dataset_path + '/indian_liver_patient.csv')

# verinin ilk 5 kısmını inceliyoruz
data.head()
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Proteins	Albumin	Albumin_and_Globulin_Ratio	Dataset
0	65	Female	0.7	0.1	187	16	18	6.8	3.3	0.90	1
1	62	Male	10.9	5.5	699	64	100	7.5	3.2	0.74	1
2	62	Male	7.3	4.1	490	60	68	7.0	3.3	0.89	1
3	58	Male	1.0	0.4	182	14	20	6.8	3.4	1.00	1
4	72	Male	3.9	2.0	195	27	59	7.3	2.4	0.40	1

```
data.tail(5)
```

	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Proteins	Albumin	Albumin_and_Globulin_Ratio	Dataset
578	60	Male	0.5	0.1	500	20	34	5.9	1.6	0.37	2
579	40	Male	0.6	0.1	98	35	31	6.0	3.2	1.10	1
580	52	Male	0.8	0.2	245	48	49	6.4	3.2	1.00	1
581	31	Male	1.3	0.5	184	29	32	6.8	3.4	1.00	1
582	38	Male	1.0	0.3	216	21	24	7.3	4.4	1.50	2

Veriyi Kaggle platformundan çektikten sonra değişkenlerin ilk ve son 5 verisi gözlemlenmektedir.

- **Verinin Değişken Türlerinin İncelenip Dönüştürülmesi**

```
# veride bazı değişkenler sayısal olup bazıları object geçtiği için verinin sayısal değerlere dönüştürülmesi gerekiyor
columns_to_convert = [
    "Total_Bilirubin", "Direct_Bilirubin",
    "Total_Protiens", "Albumin", "Albumin_and_Globulin_Ratio"]

for col in columns_to_convert:
    data[col] = pd.to_numeric(data[col], errors='coerce')

# veri tipi kontrölü
print(data.dtypes)
```

```
Age                           int64
Gender                       object
Total_Bilirubin             float64
Direct_Bilirubin            float64
Alkaline_Phosphotase          int64
Alamine_Aminotransferase      int64
Aspartate_Aminotransferase    int64
Total_Protiens              float64
Albumin                     float64
Albumin_and_Globulin_Ratio  float64
Dataset                       int64
dtype: object
```

Bazı değişkenlerin sayısal değerde olmasına rağmen analiz edildiğinde object şeklinde çıkmaktadır. İlerde analizimizde sıkıntı yaşamamak adına float ve int türlerine dönüştürülmüştür.

- **Verinin istatistiksel bilgileri / Eksik veri kontrolü**

```
# verinin istatistiksel bilgileri
data.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Age	583.0	44.746141	16.189833	4.0	33.0	45.00	58.0	90.0
Total_Bilirubin	583.0	3.298799	6.209522	0.4	0.8	1.00	2.6	75.0
Direct_Bilirubin	583.0	1.486106	2.808498	0.1	0.2	0.30	1.3	19.7
Alkaline_Phosphotase	583.0	290.576329	242.937989	63.0	175.5	208.00	298.0	2110.0
Alamine_Aminotransferase	583.0	80.713551	182.620356	10.0	23.0	35.00	60.5	2000.0
Aspartate_Aminotransferase	583.0	109.910806	288.918529	10.0	25.0	42.00	87.0	4929.0
Total_Protiens	583.0	6.483190	1.085451	2.7	5.8	6.60	7.2	9.6
Albumin	583.0	3.141852	0.795519	0.9	2.6	3.10	3.8	5.5
Albumin_and_Globulin_Ratio	579.0	0.947064	0.319592	0.3	0.7	0.93	1.1	2.8
Dataset	583.0	1.286449	0.452490	1.0	1.0	1.00	2.0	2.0

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Age                         583 non-null    int64
 1   Gender                      583 non-null    object
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphotase        583 non-null    int64
 5   Alamine_Aminotransferase    583 non-null    int64
 6   Aspartate_Aminotransferase  583 non-null    int64
 7   Total_Protiens              583 non-null    float64
 8   Albumin                     583 non-null    float64
 9   Albumin_and_Globulin_Ratio  579 non-null    float64
 10  Dataset                     583 non-null    int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.24 KB
```

```
data.shape
```

```
(583, 11)
```

Toplamda 583 tane verinin olduğu ve buna bağlı olarak 11 tane sütun yani 11 farklı değişkenle bireylerin karaciğer hastalıklarının durumu sınıflandırılmaya çalışılmaktadır. Verinin istatistiksel bilgileri incelendiğinde Albumin_and_Globulin_Ratio yani Albümin ve Globulin Oranı bilgisi 579 veri ile 4 tane verinin eksik olduğunu göstermektedir.

- **Eksik verinin tamamlanması / Sütunda isim değişikliği yapılması**

```
# Albumin_and_Globulin_Ratio  579 tane değeri var 583 olması gerekiyor. 4 tane verimiz eksik.
# eksik değeri değişkenin ortalaması ile tamamlıyoruz
data['Albumin_and_Globulin_Ratio'].fillna(data['Albumin_and_Globulin_Ratio'].mean(), inplace=True)
# yeniden eksik değer kontrolü yapılıyor
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 583 entries, 0 to 582
Data columns (total 11 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   Age                         583 non-null    int64
 1   Gender                      583 non-null    object
 2   Total_Bilirubin             583 non-null    float64
 3   Direct_Bilirubin            583 non-null    float64
 4   Alkaline_Phosphotase        583 non-null    int64
 5   Alamine_Aminotransferase    583 non-null    int64
 6   Aspartate_Aminotransferase  583 non-null    int64
 7   Total_Protiens              583 non-null    float64
 8   Albumin                     583 non-null    float64
 9   Albumin_and_Globulin_Ratio  583 non-null    float64
 10  Dataset                     583 non-null    int64
dtypes: float64(5), int64(5), object(1)
memory usage: 50.2+ KB
```

```
#  dataset isimli olan değişkeni bireyin hasta olup olmadığını anlamak için değiştirdik
data.rename(columns={'Dataset': 'Patient_Condition'}, inplace=True)
data.head()
```

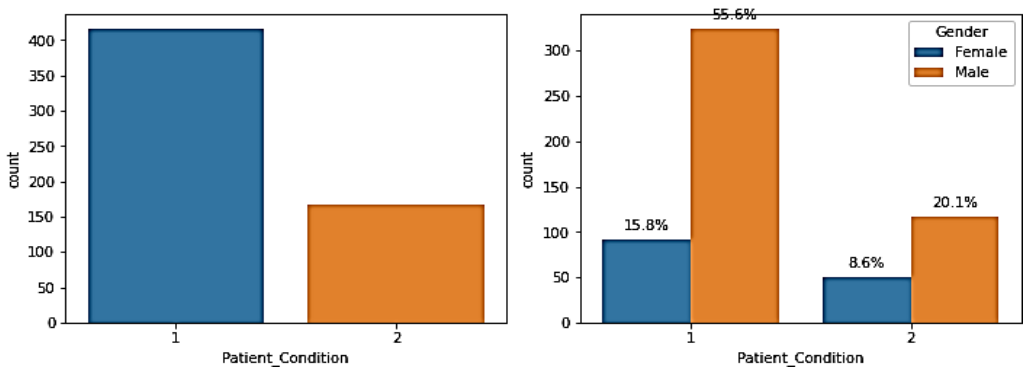
	Age	Gender	Total_Bilirubin	Direct_Bilirubin	Alkaline_Phosphotase	Alamine_Aminotransferase	Aspartate_Aminotransferase	Total_Protiens	Albumin	Albumin_and_Globulin_Ratio	Patient_Condition
0	65	Female	0.7	0.1	187	16	18	6.8	3.3	0.90	1
1	62	Male	10.9	5.5	699	64	100	7.5	3.2	0.74	1
2	62	Male	7.3	4.1	490	60	68	7.0	3.3	0.89	1
3	58	Male	1.0	0.4	182	14	20	6.8	3.4	1.00	1
4	72	Male	3.9	2.0	195	27	59	7.3	2.4	0.40	1

Albümin ve Globulin Oranı bilgisinin eksik verileri olduğu için bunu Albumin_and_Globulin_Ratio değişkenin ortalamasını alarak eksik veriler tamamlanmış olmaktadır. Dataset isimli olan değişkeni analizlerde daha rahat ayırt edebilmek adına hasta durumu olarak Patient_Condition olarak değiştiriyoruz.

- **Grafiklendirme**

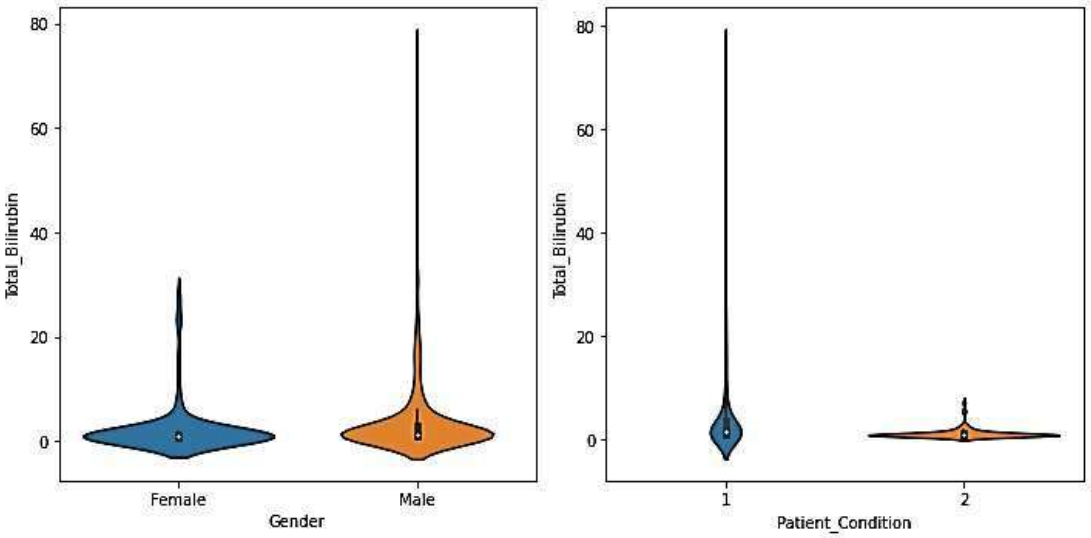
```
import matplotlib.pyplot as plt
import seaborn as sns

fig, axes=plt.subplots(nrows=1, ncols=2, figsize=(10,4))
ax1 = sns.countplot(x="Patient_Condition", data=data, ax=axes[0])
ax2 = sns.countplot(x="Patient_Condition", hue="Gender", data=data, ax=axes[1])

total = len(data)
# 2. garfiği daha rahat yorumlanması için % hesabının yapılması
for p in ax2.patches:
    count = p.get_height()
    percentage = 100 * count / total
    ax2.annotate(f'{percentage:.1f}%', (p.get_x() + p.get_width() / 2., count),
                 ha='center', va='bottom', fontsize=10, color='black', xytext=(0, 5),
                 textcoords='offset points')

# karaciğer hastası olan/olmayan bireylerin sayıları
liver_patients, not_liver_patients = data['Patient_Condition'].value_counts()
print("Karaciğer hastası sayısı:", liver_patients)
print("Karaciğer hastası olmayanların sayısı:", not_liver_patients)

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

```
Karaciğer hastası sayısı: 416
Karaciğer hastası olmayanların sayısı: 167
```



Toplam da karaciğer hastası olan 416 birey varken karaciğer hastası olmayan 167 tane birey bulunmaktadır. İlk grafik bize bu durumu göstermektedir. 2. Grafik ise karaciğer hastası olan bireylerin %55.6 ile erkeklerin daha fazla olduğunu göstermektedir. Karaciğer hastası olmayan bireyler de ise %20.1 ile yine erkeklerin fazla olduğu gözlemlenmektedir.

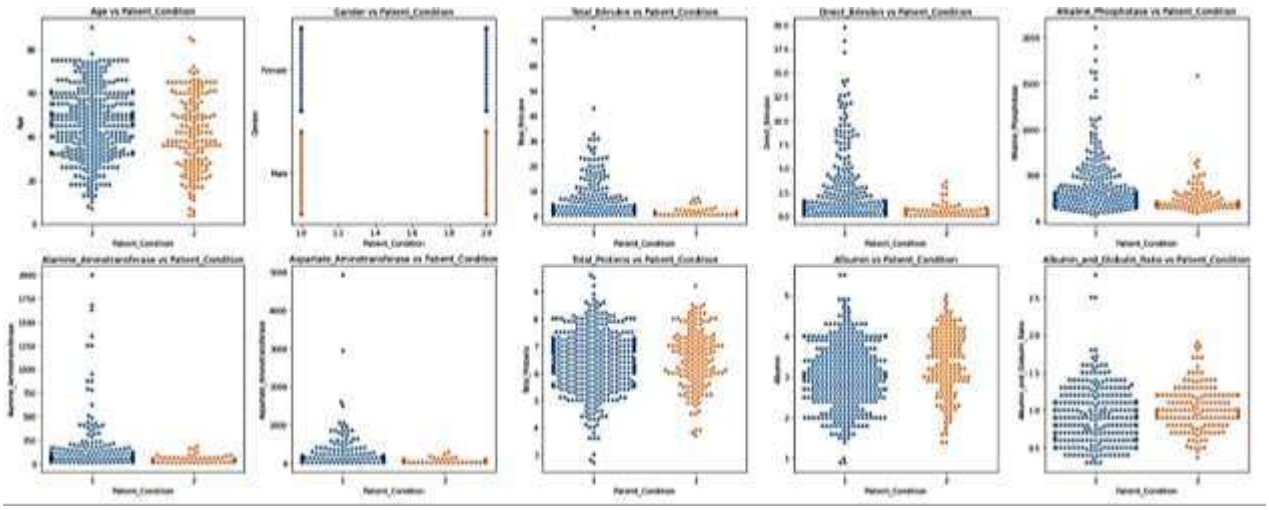
```python
# değişkenleri kategorik olarak sınıflandırıp görselleştirelim
import matplotlib.pyplot as plt
import seaborn as sns

categorical_features = categorical_features = ["Gender", "Patient_Condition"]

fig, axes = plt.subplots(1, len(categorical_features), figsize=(5*len(categorical_features), 5))
if len(categorical_features) == 1:
    axes = [axes]

for idx, feature in enumerate(categorical_features):
    sns.violinplot(x=feature, y="Total_Bilirubin", data=data, ax=axes[idx])

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



Toplam Bilirubin miktarının cinsiyet ve hastalık durumuyla karşılaştırılmasıdır. İlk grafik ile total bilirubin değerinin kadın ve erkekler içinde 0-10 arasında yoğunlaştığı hatta erkeklerde uç değerlerin biraz fazla olduğu gözlemlenmiştir. Genel olarak yorumlanacakta cinsiyetin total bilirubin üzerinde çok güçlü bir etkisi olmadığı yönündedir. 2 . grafik ile hasta durumlarının total bilirubinle durumu incelenmektedir. Hasta bireylerin total bilirubin 0-80ile geniş bir dağılım gösteriyorken , hasta olmayan bireylerin 0-20 arasında yoğunlaşarak çok sapma

göstermemektedir. Bu durum total bilirubin değerinin karaciğer hastası olan bireylerde değerinin yüksek olduğunu ve geniş dağıldığını göstermektedir.

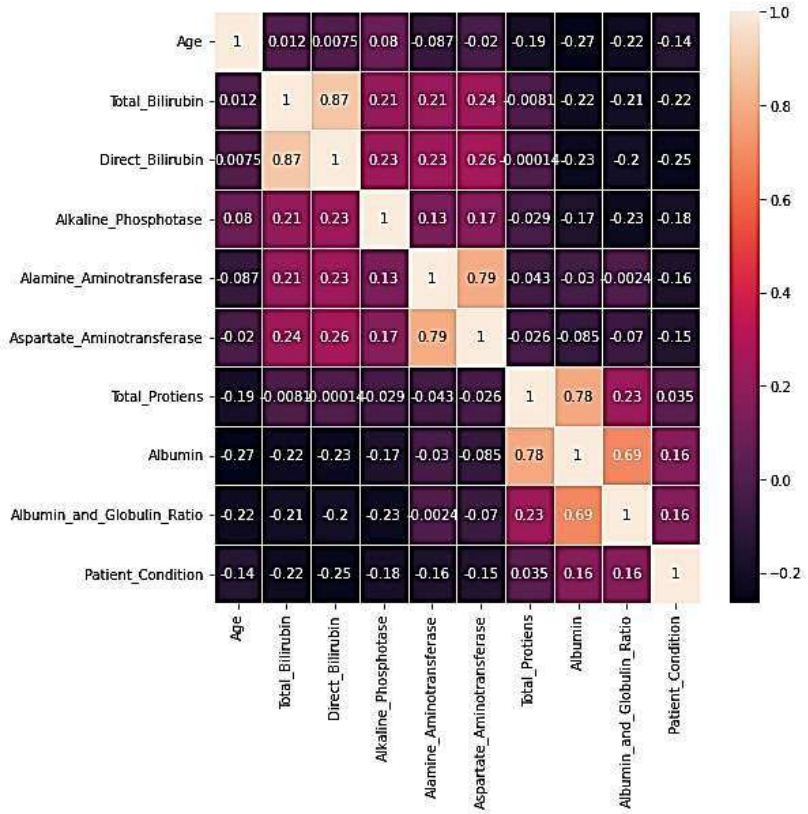
```
import matplotlib.pyplot as plt
import seaborn as sns

features = ["Age", "Gender", "Total_Bilirubin", "Direct_Bilirubin",
            "Alkaline_Phosphotase", "Alamine_Aminotransferase",
            "Aspartate_Aminotransferase", "Total_Protiens",
            "Albumin", "Albumin_and_Globulin_Ratio"]

# 5 sütun ile grafijleri çizdirme
n_cols = 5
n_rows = (len(features) + n_cols - 1) // n_cols

fig, axes = plt.subplots(nrows=n_rows, ncols=n_cols, figsize=(5*n_cols, 5*n_rows))
axes = axes.flatten()

# tüm değişkenlerin swarm grafiği
for idx, feature in enumerate(features):
    if feature != "Patient_Condition":
        sns.swarmplot(x="Patient_Condition", y=feature, data=data, ax=axes[idx])
        axes[idx].set_title(f'{feature} vs Patient_Condition')
    else:
        axes[idx].axis('off')

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



İki değişken arasındaki ilişkiyi anlamlandırabilmek adına serpilme grafiği kullanarak değişkenleri analiz etmiş bulunmaktayız. Hastalık durumuyla yaşın, toplam proteinin ve albümin miktarının normal dağıldığını incelemiş bulunmaktayız.

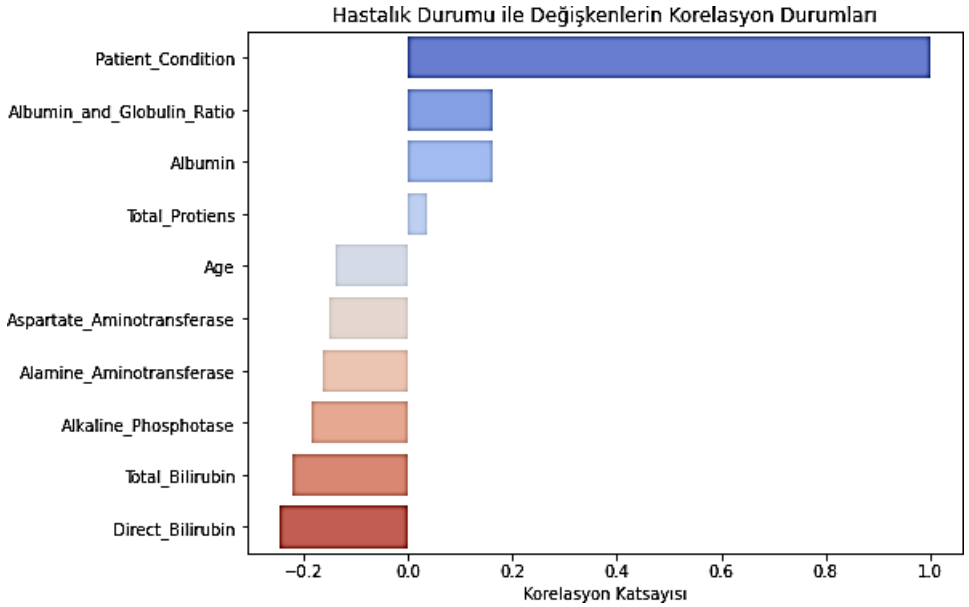
```
# değişkenlerin korelasyonlarını inceleyelim
corr_matrix=data.corr()
fig, ax = plt.subplots(figsize=(8,8))
sns.heatmap(corr_matrix,annot=True,linewidths=.5, ax=ax)
```

<AxesSubplot:>




```
# bireylerin hastalıkla en yüksek korelasyonlarının durumu
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

corr_matrix=data.corr()

# korelasyonun sıralanması
patient_corr = corr_matrix["Patient_Condition"].sort_values(ascending=False)

print(patient_corr)

# grafik hali
plt.figure(figsize=(8, 6))
sns.barplot(x=patient_corr.values, y=patient_corr.index, palette="coolwarm")
plt.title("Hastalık Durumu ile Değişkenlerin Korelasyon Durumları")
plt.xlabel("Korelasyon Katsayısı")
plt.grid(False)
plt.show()
```

```
Patient_Condition            1.000000
Albumin_and_Globulin_Ratio   0.162319
Albumin                      0.161388
Total_Protiens               0.035008
Age                         -0.137351
Aspartate_Aminotransferase  -0.151934
Alamine_Aminotransferase    -0.163416
Alkaline_Phosphotase        -0.184866
Total_Bilirubin             -0.220208
Direct_Bilirubin            -0.246046
Name: Patient_Condition, dtype: float64
```



En yüksek korelasyona sahip değerin 1 ile hastalık durumunu gösterip kuvvetli ve olumlu bir ilişkinin olduğunu gösterirken -0.24 değeri ile direkt bilirubin değerinin hastalık durumuyla arasında olumsuz bir ilişki olduğunu göstermektedir. Albumin ve Globulin oranı ise 0.16 korelasyon ile hastalık durumu ilişkisinin 0'a yaklaştığı gözlemlendiği için ilşkinin zayıflamaya başladığını göstermektedir.

- **Makine Öğrenmesi Analizinin Gerçekleştirilmesi**

```
Makine Öğrenmesi Modeli
```

```
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB

from xgboost import XGBClassifier

from sklearn.metrics import accuracy_score, classification_report
```

```
# değişkenleri ayırıyoruz
X = data.drop('Patient_Condition', axis=1)
y = data['Patient_Condition']   # bağımlı değişken
```

```
# XGBoost ile analiz yapılabilmesi için Patient_Condition olan değişkenin 1 ve 2 olan değerlerini
# uyumluluk için 0 ve 1'e dönüştürüyoruz.
y = y.map({1: 0, 2: 1})
```

```
# veriyi %70 eğitim %30 test verisi olarak bölüyoruz
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.30, random_state=42)
```

•••

```
# svm ve knn gibi sınıflandırmalar için veriyi ölçeklendiriyoruz
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

İlk olarak gerekli olan kütüphane ve modelleri ekledikten sonra değişkenleri ayırıyoruz. Hastalık durumu değişkeni 1 ve 2 olacak şekilde kategorik veri olduğu için bağımlı değişkendir. Araştırmanın amacı da hasta bireyleri ile hasta olmayan bireylerin sınıflandırılması için en uygun yöntemin seçilmesidir. Bu durumda değişkenleri ayırdıktan sonra sınıflandırma yöntemlerinde test edilmesi için hastalık durumu değişkenini 0 ve 1 olacak şekilde XGBoost analizine uygun haline getirerek dönüştürüyoruz. Veriyi %70 eğitim %30 test verisi olacak şekilde bölüyoruz. Yapılacak olan sınıflandırmalar için de svm ve knn gibi çeşitlerde de uygun olması açısından veriyi standartlaştırıp ölçeklendirmekteyiz.

```
# modeller
models_params = {
    "Logistic Regression": (LogisticRegression(max_iter=1000), {
        'C': [0.01, 0.1, 1, 10],
        'penalty': ['l1', 'l2'],
        'solver': ['liblinear']
    }),
    "Random Forest": (RandomForestClassifier(), {
        'n_estimators': [50, 100, 200],
        'max_depth': [5, 10, 20],
        'min_samples_split': [2, 5]
    }),
    "Decision Tree": (DecisionTreeClassifier(), {
        'max_depth': [3, 5, 10, None],
        'min_samples_split': [2, 5, 10]
    }),
    "SVC": (SVC(), {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf', 'poly'],
        'gamma': ['scale', 'auto']
    }),
    "KNN": (KNeighborsClassifier(), {
        'n_neighbors': [3, 5, 7, 9],
        'weights': ['uniform', 'distance']
    }),
    "XGBoost": (XGBClassifier(use_label_encoder=False, eval_metric='logloss'), {
        'n_estimators': [50, 100],
        'max_depth': [3, 5, 7],
        'learning_rate': [0.01, 0.1, 0.2]
    }),
    "Naive Bayes (GaussianNB)": (GaussianNB(), {})
}
```

```
# GridSearchCV ile her modelin en iyi parametresi bulunuyor
results = {}

for model_name, (model, params) in models_params.items():
    print(f"\n--- {model_name} için eğitim başlıyor ---")

    grid = GridSearchCV(estimator=model, param_grid=params, cv=5, n_jobs=-1, verbose=0)
    grid.fit(X_train, y_train)

    best_model = grid.best_estimator_
    y_pred = best_model.predict(X_test)
    acc = accuracy_score(y_test, y_pred)

    results[model_name] = {
        'Best Params': grid.best_params_,
        'Test Accuracy': acc
    }

    print(f"{model_name} | Test Set Accuracy: {acc:.4f}")
    print("En iyi parametreler:", grid.best_params_)
    print(classification_report(y_test, y_pred))
```

Karşılaştırılmak istenilen model çeşitleri belirtildikten sonra GridSearchCV ile her modelin en iyi parametresi bulunması amaçlanmıştır. Son durumda makine öğrenmesi analizi sonuçları şu şekilde sonuçlanmaktadır:

```
--- Logistic Regression için eğitim başlıyor ---
Logistic Regression | Test Set Accuracy: 0.7371
En iyi parametreler: {'C': 10, 'penalty': 'l1', 'solver': 'liblinear'}
              precision    recall  f1-score   support

           0       0.77      0.91      0.83       128
           1       0.52      0.28      0.36        47

    accuracy                           0.74       175
   macro avg       0.65      0.59      0.60       175
weighted avg       0.71      0.74      0.71       175
```

```
--- Random Forest için eğitim başlıyor ---
Random Forest | Test Set Accuracy: 0.7029
En iyi parametreler: {'max_depth': 5, 'min_samples_split': 2, 'n_estimators': 50}
              precision    recall  f1-score   support

           0       0.75      0.88      0.81       128
           1       0.40      0.21      0.28        47

    accuracy                           0.70       175
   macro avg       0.58      0.55      0.55       175
weighted avg       0.66      0.70      0.67       175
```

```
--- Decision Tree için eğitim başlıyor ---
Decision Tree | Test Set Accuracy: 0.7314
En iyi parametreler: {'max_depth': 3, 'min_samples_split': 5}
              precision    recall  f1-score   support

           0       0.76      0.92      0.83       128
           1       0.50      0.21      0.30        47

    accuracy                           0.73       175
   macro avg       0.63      0.57      0.57       175
weighted avg       0.69      0.73      0.69       175
```

```
--- SVC için eğitim başlıyor ---
SVC | Test Set Accuracy: 0.7314
En iyi parametreler: {'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
              precision    recall  f1-score   support

           0       0.73      1.00      0.84       128
           1       0.00      0.00      0.00        47

    accuracy                           0.73       175
   macro avg       0.37      0.50      0.42       175
weighted avg       0.53      0.73      0.62       175
```

```
--- KNN için eğitim başlıyor ---
KNN | Test Set Accuracy: 0.6971
En iyi parametreler: {'n_neighbors': 7, 'weights': 'distance'}
              precision    recall  f1-score   support

           0       0.80      0.79      0.79       128
           1       0.44      0.45      0.44        47

    accuracy                           0.70       175
   macro avg       0.62      0.62      0.62       175
weighted avg       0.70      0.70      0.70       175
```

```
XGBoost | Test Set Accuracy: 0.7314
En iyi parametreler: {'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 50}
              precision    recall  f1-score   support

           0       0.73      1.00      0.84       128
           1       0.00      0.00      0.00        47

    accuracy                           0.73       175
   macro avg       0.37      0.50      0.42       175
weighted avg       0.53      0.73      0.62       175
```

```
--- Naive Bayes (GaussianNB) için eğitim başlıyor ---
Naive Bayes (GaussianNB) | Test Set Accuracy: 0.5714
En iyi parametreler: {}
              precision    recall  f1-score   support

           0       0.90      0.42      0.59       128
           1       0.30      0.90      0.45        47

    accuracy                           0.57       175
   macro avg       0.60      0.70      0.57       175
weighted avg       0.82      0.57      0.56       175
```

```python
# sonuçların özeti
print("\n******** Tüm Modellerin Test Başarıları ********")
for model, res in results.items():
    print(f"{model}: Accuracy={res['Test Accuracy']:.4f}, Best Params={res['Best Params']}")
```

```
******** Tüm Modellerin Test Başarıları ********
Logistic Regression: Accuracy=0.7371, Best Params={'C': 10, 'penalty': 'l1', 'solver': 'liblinear'}
Random Forest: Accuracy=0.7029, Best Params={'max_depth': 5, 'min_samples_split': 2, 'n_estimators': 50}
Decision Tree: Accuracy=0.7314, Best Params={'max_depth': 3, 'min_samples_split': 5}
SVC: Accuracy=0.7314, Best Params={'C': 0.1, 'gamma': 'scale', 'kernel': 'linear'}
KNN: Accuracy=0.6971, Best Params={'n_neighbors': 7, 'weights': 'distance'}
XGBoost: Accuracy=0.7314, Best Params={'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 50}
Naive Bayes (GaussianNB): Accuracy=0.5714, Best Params={}
```

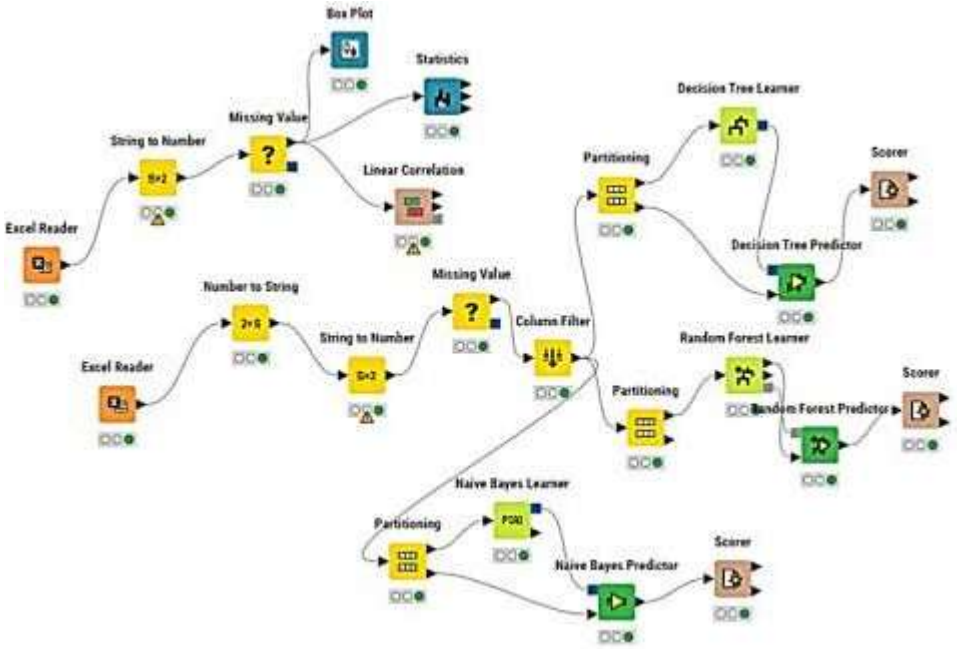
En sonda tüm modellerin doğruluk değerlerini özet şekilde gösterilmektedir. Tüm bu analiz sonucunu incelediğimizde en uygun testin lojistik regresyon test analizi olduğu doğruluk değeri 0.737 ile ortaya çıkmaktadır.

Tablo 1. : Lojistik Regresyon Test Sonucu

	Precision	*Recal*	*F1*	*Support*
K. Hastası	0.77	0.91	0.83	128
K.Hastası Değil	0.52	0.28	0.36	47

Tablo 1 incelediğimizde gerçek hastalardan %91 oranı doğru tespit edilmiştir. F_1 skoru 0.83 ile modelin başarılı bir sonuç ortaya çıkardığını göstermiş bulunmaktadır. Bu model karaciğer hastalığı olan bireyleri tespit etmede başarılı bir sonuç ortaya çıkarmıştır.

1.3.2. KNIME İLE ANALİZ DURUMU VE SONUÇLARI

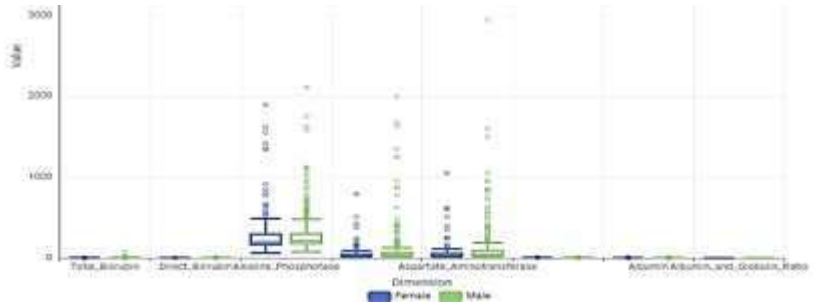


Excel Rearder : Karaciğer hastası olan ve olmayan bireylerin çeşitli değişkenlerle verisi excel rearder ile Knıme sistemine yüklenmiştir.

Strıng To Number : Yüklenen veriler incelendiğinde bazı verilerin string türünde olduğu gözlemlenip değiştirilmesi gereken değişkenler string türünden integer türüne dönüştürülmüştür.

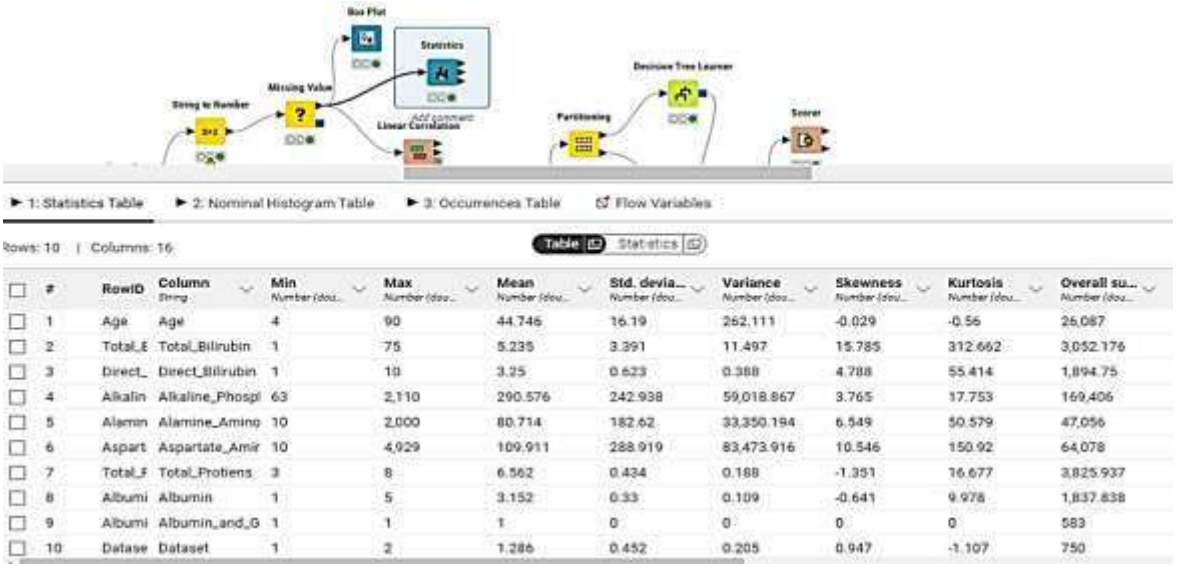
Mıssıng Value : Veride bulunan eksik değerler ilerde yapılacak olan makine öğrenmesi modelinde hatalar yaratacağı için eksik değerler tamamlanmıştır. Eksik değerler integer türünde ki verilerde eksik çıktığı için ortalama ile değerleri tamamlamış bulunmaktayız.

Box Plot :



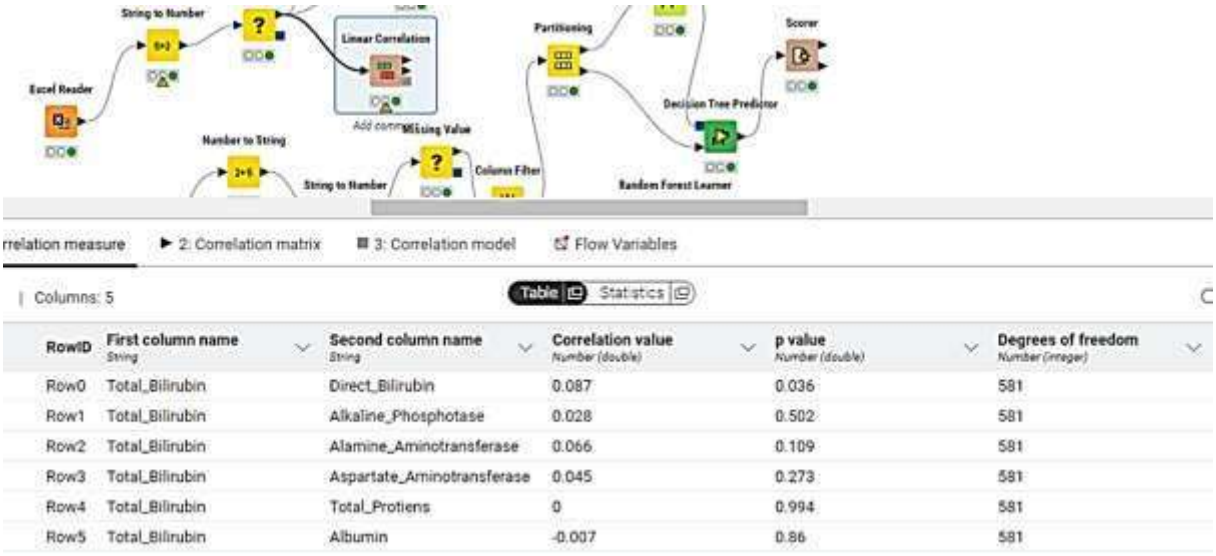
Kutu grafiği ile seçilmiş değişkenlerin durumları hakkında bilgiler edindik.

Statistics : Tüm verilerin istatistiksel bilgilerini özet şekilde incelemiş bulunmaktayız.



#	RowID	Column	Min	Max	Mean	Std. devia...	Variance	Skewness	Kurtosis	Overall su...
1	Age	Age	4	90	44.746	16.19	262.111	-0.029	-0.56	26,087
2	Total_E	Total_Bilirubin	1	75	5.235	3.391	11.497	15.785	312.662	3,052.176
3	Direct_	Direct_Bilirubin	1	19	3.25	0.623	0.388	4.788	55.414	1,894.75
4	Alkalin	Alkaline_Phospl	63	2,110	290.576	242.938	59,018.867	3.765	17.753	169,406
5	Alamin	Alamine_Amino	10	2,000	80.714	182.62	33,350.194	6.549	50.579	47,056
6	Aspart	Aspartate_Amir	10	4,929	109.911	288.919	83,473.916	10.546	150.92	64,078
7	Total_F	Total_Protiens	3	8	6.562	0.434	0.188	-1.351	16.677	3,825.937
8	Albumi	Albumin	1	5	3.152	0.33	0.109	-0.641	9.978	1,837.838
9	Albumi	Albumin_and_G	1	1	1	0	0	0	0	583
10	Datase	Dataset	1	2	1.286	0.452	0.205	0.947	-1.107	750

Linear Correlation : Verimizde bulunan sayısal değişkenler arasında ki doğrusal ilişkiyi , korelasyonu analiz etmiş bulunmaktayız.



RowID	First column name	Second column name	Correlation value	p value	Degrees of freedom
Row0	Total_Bilirubin	Direct_Bilirubin	0.087	0.036	581
Row1	Total_Bilirubin	Alkaline_Phosphotase	0.028	0.502	581
Row2	Total_Bilirubin	Alamine_Aminotransferase	0.066	0.109	581
Row3	Total_Bilirubin	Aspartate_Aminotransferase	0.045	0.273	581
Row4	Total_Bilirubin	Total_Protiens	0	0.994	581
Row5	Total_Bilirubin	Albumin	-0.007	0.86	581

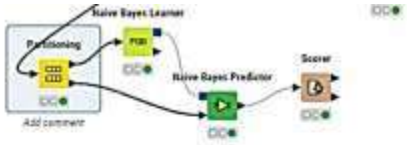
Sonucu incelediğimizde toplam bilirubin değeri ile direkt bilirubin değeri arasında 0.087 değeri ile çok düşük bir ilişki neredeyse aralarında ilişki yok denecek kadar az olduğu ortaya çıkmıştır.

İlk olarak veriyi anlayıp , verinin hikayesine biraz daha hakim olundu. Daha sonra bunlara uygun düzenlemeler gerçekleştirilerek veriyi makine öğrenmesine modeline hazırlamış bulunmaktayız.

Model eğitimi için ayrıyeten yeniden ***Excel Rearder*** node kullanmış bulunmaktayız. Daha sonra ***Number To String*** ve ***String To Number*** dönüşümler ile veri hazır hale gelmiş bulunmaktadır. ***Mıssıng Value*** ile eksik değerler tamamlanıp , ***Column Filter*** ile satır filtrelemesi kullanmak istemediğimiz değişkeni ayırt etmiş bulunmaktayız.

Artık makine öğrenmesi için verimiz hazır bulunmaktadır.

Partitioning : Veriyi makine öğrenmesinde test edebilmek için %30 test verisi %70 eğitim verisi olacak şekilde bölünmüştür.



First partition | 2: Second partition | Flow Variables

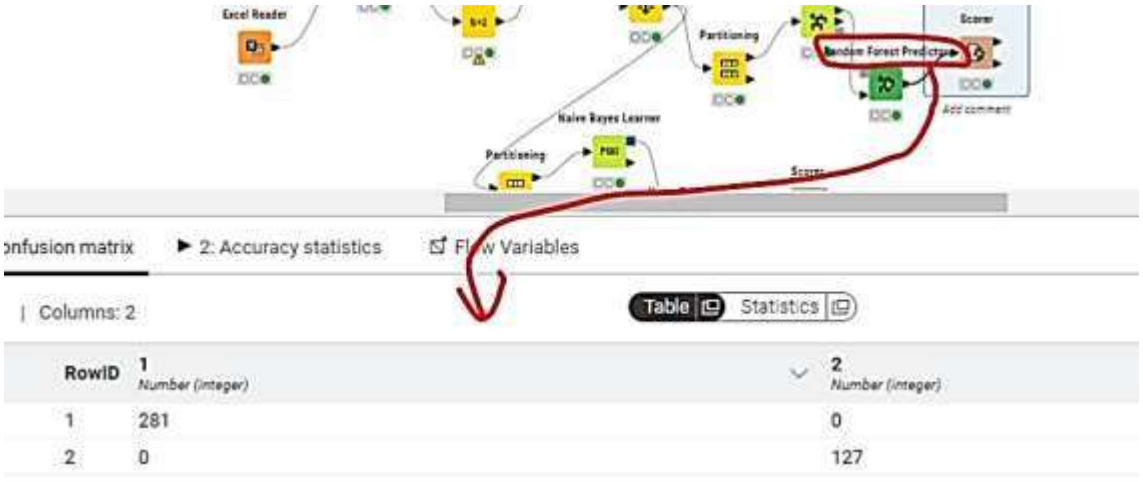
408 | Columns: 10 — Table | Statistics

#	RowID	Age	Gender	Total_Bilir...	Direct_Bil...	Alkaline_...	Alamine_...	Aspartate...	Total_Pro...	Albumin
		Number (dou...	String	Number (dou...	Number (dou...	Number (dou...	Number (dou...	Number (dou...	Number (dou...	Number (dou...
1	Row0	65	Female	5.235	3.25	187	16	18	6.562	3.152
2	Row1	62	Male	5.235	3.25	699	64	100	6.562	3.152
3	Row4	72	Male	5.235	2	195	27	59	6.562	3.152
4	Row6	26	Female	5.235	3.25	154	16	12	7	3.152
5	Row7	29	Female	5.235	3.25	202	14	11	6.562	3.152
6	Row9	55	Male	5.235	3.25	290	53	58	6.562	3.152
7	Row11	72	Male	5.235	3.25	260	31	56	6.562	3
8	Row12	64	Male	5.235	3.25	310	61	58	7	3.152

408 tane veri eğitim , 175 tane veri ise test verisi olarak ayrılmış bulunmaktadır.

Decision Tree Learner , Random Forest Learner , Leave Bayes Learner gibi çeşitli sınıflandırma türleri ile verinin öğrenilmesi sağlanılmıştır. Daha sonra bu testlere ait olan ***Decision Tree Predictor , Random Forest Predictor , Leave Bayes Predictor*** ile öğrenilen verinin tahminlenmesi sağlanılmıştır.

Scorer : Node ile yapılan tüm makine öğrenmesi analizinin sonucu değerlendirilmiştir. Sınıflandırma durumlarını özet şekilde inceleme fırsatı bulmuş bulunmaktayız.



Tüm sınıflandırma analizlerini incelediğimde en doğru sınıflandırmayı Random Forest algoritmasının yaptığı ortaya çıkmıştır. Yeni veriler verildiğinde aynı değişken durumları ile bir bireyin karaciğer hastası olup olmayacağını teste etmek için Random Forest modeli kullanılarak değerlendirilebilir.

NOT: Knıme ve Python da yapılan analiz ve sonuçlar değerlendirildiğinde ;

Python da yapılan analizde Lojistik regresyon analizi daha uygun bulunmuşken Knıme ile yapılan analizde Random Forest modeli daha uygun bulunmuştur. Sebebini Python da eksik veriyi kontrol ettiğimizde bir değişkenimizde 4 tane veri eksik olup tamamlanarak yapılmışken Knıme da veriyi indirip düzenlediğim halde daha fazla veri eksik çıkmış bulunmaktadır. Verinin tür ve durumuna göre eksik veri tamamlanarak analize devam edilmiştir. Bu durum da iki uygulama arasında yapılan analiz ve sonuçta ufak farklılık ortaya çıkmaktadır.

BÖLÜM 2

YETİŞKİN NÜFUS SAYIMI GELİRİ ANALİZİ

ANAMOLİ TESPİTİ / REGRESYON ANALİZİ / SINIFLANDIRMA / İLŞKİLENDİRME / KÜMELEME ANALİZLERİ

2.1. Veri Setinin Durumu / Hikayesi

Ronny Kohavi ve Barry Becker (Veri Madenciliği ve Görselleştirme, Silikon Grafikler) tarafından 1994 yılında Nüfus Sayımı bürosu veritabanından çıkarılan verilerdir. Yetişkin nüfus sayımı geliri analizin de amaç bir kişinin yılda 50.000 $ 'ın üzerinde kazanıp kazanmadığını , hangi durumların bu geliri etkilediği vb. durumların analizi gerçekleştirilecektir.

Veri de bulunan değişkenler şu şekildedir :

- **Age :** Yaş
- **Workclass :** İş sınıfı
- **Fnlwgt :**
- **Education :** Eğitim
- **Education.num :** Eğitim sayısı
- **Marital.status :** Medeni durum
- **Occupation :** Meslek
- **Relationship :** İlişki
- **Race :** Irk
- **Sex :** Cinsiyet
- **Capital.gain :**Sermaye kazancı
- **Capital.loss :** Sermaye kaybı
- **Hours.per.week :** Saat/hafta
- **Native.country :** Ülke
- **İncome :** Gelir durumlarını içermektedir.

Veri seti kaynağı: https://www.kaggle.com/datasets/uciml/adult-census-income/data

2.2. Veri Analiz Platformları

Veri setinin analizi Anaconda uygulamasının içeriğinde bulunan JupyterLab ile Python analizleri gerçekleştirilecektir.

NOT: Verinin miktarından dolayı Python'da yapılan analizlerde bile GPU çok zorlandığı için Knıme uygulamasında veri yüklenip test edilememiştir.

2.3. Verinin Analiz Durumu

Yetişkin nüfus sayımı gelir analizi ile veriler test edilerek uygun olan modellerin eğitilip test edilmesi sonucu analiz gerçekleştirilecektir.

2.3.1. PYTHON İLE ANALİZ DURUMU VE SONUÇLARI

- **Verinin yüklenip ilk ve son satırlarının kontrolü**

```
import pandas as pd
import kagglehub

# kaggledan veri setini çekiyoruz
path = kagglehub.dataset_download("uciml/adult-census-income")

# veriyi yükl0yoruz
data = pd.read_csv(path + '/adult.csv')

# verinin ilk 5 satırı
data.head()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-family	White	Female	0	4356	40	United-States	<=50K
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	0	4356	18	United-States	<=50K
2	66	?	186061	Some-college	10	Widowed	?	Unmarried	Black	Female	0	4356	40	United-States	<=50K
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	0	3900	40	United-States	<=50K
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	0	3900	40	United-States	<=50K

```
# verinin son 5 satırı
data.tail(5)
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
32556	22	Private	310152	Some-college	10	Never-married	Protective-serv	Not-in-family	White	Male	0	0	40	United-States	<=50K
32557	27	Private	257302	Assoc-acdm	12	Married-civ-spouse	Tech-support	Wife	White	Female	0	0	38	United-States	<=50K
32558	40	Private	154374	HS-grad	9	Married-civ-spouse	Machine-op-inspct	Husband	White	Male	0	0	40	United-States	>50K
32559	58	Private	151910	HS-grad	9	Widowed	Adm-clerical	Unmarried	White	Female	0	0	40	United-States	<=50K
32560	22	Private	201490	HS-grad	9	Never-married	Adm-clerical	Own-child	White	Male	0	0	20	United-States	<=50K

- **Veriler ve değişkenler hakkında bilgi edinme**

```
# verinin boyutu
data.shape
```

```
(32561, 15)
```

```
# veriler hakkında bilgi edinme
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype 
---  ------          --------------  ----- 
 0   age             32561 non-null  int64 
 1   workclass       32561 non-null  object
 2   fnlwgt          32561 non-null  int64 
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64 
 5   marital.status  32561 non-null  object
 6   occupation      32561 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital.gain    32561 non-null  int64 
 11  capital.loss    32561 non-null  int64 
 12  hours.per.week  32561 non-null  int64 
 13  native.country  32561 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

```
# tüm değişkenlerin eksik değerlerini inceliyoruz
data.isnull().sum()
```

```
age               0
workclass         0
fnlwgt            0
education         0
education.num     0
marital.status    0
occupation        0
relationship      0
race              0
sex               0
capital.gain      0
capital.loss      0
hours.per.week    0
native.country    0
income            0
dtype: int64
```

```
# eksik değer kontrolünde hiç eksik değer bulunmuyor fakat verinin
# ilk 5 verisini incelediğimizde "?" veriler bulunmaktadır.
# bunlar eksik ver olarak "NaN" dönüştürülüyor
import numpy as np
data[data == '?'] = np.nan
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   age             32561 non-null  int64
 1   workclass       30725 non-null  object
 2   fnlwgt          32561 non-null  int64
 3   education       32561 non-null  object
 4   education.num   32561 non-null  int64
 5   marital.status  32561 non-null  object
 6   occupation      30718 non-null  object
 7   relationship    32561 non-null  object
 8   race            32561 non-null  object
 9   sex             32561 non-null  object
 10  capital.gain    32561 non-null  int64
 11  capital.loss    32561 non-null  int64
 12  hours.per.week  32561 non-null  int64
 13  native.country  31978 non-null  object
 14  income          32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

```
# güncellenmiş tüm değişkenlerin eksik değerlerini inceliyoruz
data.isnull().sum()
```

```
age                0
workclass       1836
fnlwgt             0
education          0
education.num      0
marital.status     0
occupation      1843
relationship       0
race               0
sex                0
capital.gain       0
capital.loss       0
hours.per.week     0
native.country   583
income             0
dtype: int64
```

```
# eksik değerler en çok tekrar eden mod ile tamamlanıyor
for col in ['workclass', 'occupation', 'native.country']:
    data[col].fillna(data[col].mode()[0], inplace=True)
```

```
data.isnull().sum()
# kontrol edip eksik veri sorunun ortadan kaldırıyoruz
```

```
age               0
workclass         0
fnlwgt            0
education         0
education.num     0
marital.status    0
occupation        0
relationship      0
race              0
sex               0
capital.gain      0
capital.loss      0
hours.per.week    0
native.country    0
income            0
dtype: int64
```

```
data.drop(["fnlwgt","capital.gain","capital.loss"], axis=1, inplace=True)
```

```
data.head()
```

	age	workclass	education	education.num	marital.status	occupation	relationship	race	sex	hours.per.week	native.country	income
0	90	Private	HS-grad	9	Widowed	Prof-specialty	Not-in-family	White	Female	40	United-States	<=50K
1	82	Private	HS-grad	9	Widowed	Exec-managerial	Not-in-family	White	Female	18	United-States	<=50K
2	66	Private	Some-college	10	Widowed	Prof-specialty	Unmarried	Black	Female	40	United-States	<=50K
3	54	Private	7th-8th	4	Divorced	Machine-op-inspct	Unmarried	White	Female	40	United-States	<=50K
4	41	Private	Some-college	10	Separated	Prof-specialty	Own-child	White	Female	40	United-States	<=50K

Veri setimizin 32561 tane veriden ve 15 tane değişkenden olduğu incelenmiştir. Eksik değer kontrolü yapıldığında hiç eksik değer olmadığı fakat en başta verinin baş ve sonunu incelediğimizde bazı değerlerin " **?** " olarak girildiği ortaya çıkmıştı. Bu durum ileride yapılacak olan model eğitiminde hata yaratacağı için " **?** " olan değerler ilk olarak NoN olarak değiştirildi. Daha sonrasında değişkenlerimiz kategorik değişken oldukları için en çok tekrar eden değişkenlerin eksik değer ile doldurulması hedeflenerek mod değerleri alınarak eklenmiş bulunmaktadır. Eksik değer kontrolü yeniden yapılarak verinin mod değerlerini aldığını incelemek amacıyla yeniden verinin ilk 5 satırı incelenmiştir.

- **Makine Öğrenmesi Analizinin Gerçekleştirilmesi**

  - **Model eğitimi için hazırlık oluşturuluyor.**

```
# model eğitimine hazırlık
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# kategorik verileri sayısallaştırma
categorical_cols = data.select_dtypes(include=['object']).columns

le = LabelEncoder()
for col in categorical_cols:
    data[col] = le.fit_transform(data[col])

X = data.drop('income', axis=1)
y = data['income'] #bağımlı değişken

# veriyi %20 test %80'i eğitim olacak eşkilde ayırma
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- **Anomali tespiti yapılıyor.**

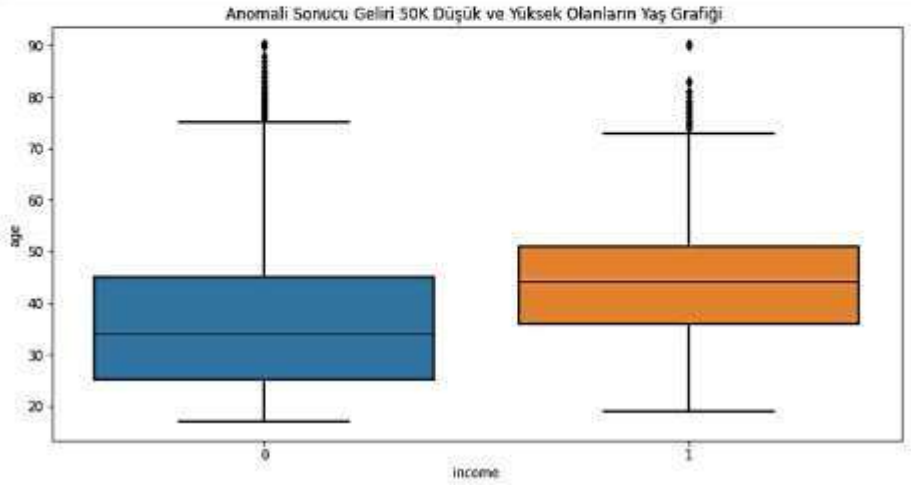
```
Anamoli Tespiti
```

```
from sklearn.ensemble import IsolationForest
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score

# veriyi hazırlama
columns_to_drop = []
if 'fnlwgt' in data.columns:
    columns_to_drop.append('fnlwgt')
if 'income' in data.columns:
    columns_to_drop.append('income')

X = data.drop(columns=columns_to_drop)
y = data['income']

# anomali tespiti
iso_forest = IsolationForest(contamination=0.05, random_state=42)
outliers = iso_forest.fit_predict(X_train)

# anomali olanları çıkarma temizleme
X_train_clean = X_train[outliers == 1]
y_train_clean = y_train[outliers == 1]

# model denemesi
model = LinearRegression()
model.fit(X_train_clean, y_train_clean)

y_pred = model.predict(X_test)

# değerlendirme
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Test Mean Squared Error (MSE): {mse:.4f}")
print(f"Test R-squared (R²): {r2:.4f}")
```

```
Test Mean Squared Error (MSE): 0.1410
Test R-squared (R²): 0.2182
```

Fnlwgt değişkenin değerinin çok yüksek olması model eğitiminde yanıltıcı hatalara yol verebilmektedir. Araştırmalarım sonucunda bu tarz değişkenin çoğu zaman veri de kullanılmadığını öğrenmiş olduğum için ben de model eğitirken çıkarıyorum. Anomali testi yaptıktan sonra bir regresyon analizinde deneyerek bu kısmı tamamlıyorum.

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

# anomali sonucu bir durumu göstermek adına yapılan görselleştirme
plt.figure(figsize=(12, 6))
sns.boxplot(x=y_train_clean, y=X_train_clean['age'])
plt.title('Anomali Sonucu Geliri 50K Düşük ve Yüksek Olanların Yaş Grafiği ')
plt.show()
```



Kutu grafiğinde görünen aykırı değerler aslında yaşı yüksek olan bireylerin maaşı 50K ‘dan az veya fazla olması durumunu göstermektedir. Veri de eksik değer veya anormal bir durum söz konusu değildir

- **Regresyon analizi yapılıyor.**

Regresyon Analizi

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge, Lasso, ElasticNet
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error, r2_score

# modeller
models = {
    'Linear Regression': LinearRegression(),
    'Ridge': Ridge(),
    'Lasso': Lasso(),
    'ElasticNet': ElasticNet(),
    'Decision Tree': DecisionTreeRegressor(),
    'Random Forest': RandomForestRegressor(),
    'Gradient Boosting': GradientBoostingRegressor()
}

param_grid = {
    'Linear Regression': {'fit_intercept': [True, False]},
    'Ridge': {'alpha': [0.1, 1, 10, 100]},
    'Lasso': {'alpha': [0.1, 1, 10, 100]},
    'ElasticNet': {'alpha': [0.1, 1, 10, 100], 'l1_ratio': [0.1, 0.5, 0.9]},
    'Decision Tree': {'max_depth': [None, 10, 20], 'min_samples_split': [2, 10]},
    'Random Forest': {'n_estimators': [100, 200], 'max_depth': [None, 10, 20]},
    'Gradient Boosting': {'n_estimators': [100, 200], 'learning_rate': [0.01, 0.1]}
}

# tüm sonuçlar için sözlük oluşturuyoruz
best_models = {}

for name, model in models.items():
    print(f"\nModel: {name}")
    grid_search = GridSearchCV(estimator=model, param_grid=param_grid[name], cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
    grid_search.fit(X_train, y_train)

    best_models[name] = grid_search.best_estimator_

    y_pred = grid_search.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"Best Parameters: {grid_search.best_params_}")
    print(f"Test MSE: {mse:.4f}")
    print(f"Test R²: {r2:.4f}")
```

```
Model: Linear Regression
Best Parameters: {'fit_intercept': True}
Test MSE: 0.1408
Test R²: 0.2192

Model: Ridge
Best Parameters: {'alpha': 10}
Test MSE: 0.1408
Test R²: 0.2192

Model: Lasso
Best Parameters: {'alpha': 0.1}
Test MSE: 0.1485
Test R²: 0.1765

Model: ElasticNet
Best Parameters: {'alpha': 0.1, 'l1_ratio': 0.1}
Test MSE: 0.1418
Test R²: 0.2134

Model: Decision Tree
Best Parameters: {'max_depth': 10, 'min_samples_split': 10}
Test MSE: 0.1224
Test R²: 0.3209

Model: Random Forest
Best Parameters: {'max_depth': 10, 'n_estimators': 200}
Test MSE: 0.1125
Test R²: 0.3760

Model: Gradient Boosting
Best Parameters: {'learning_rate': 0.1, 'n_estimators': 200}
Test MSE: 0.1119
Test R²: 0.3796
```

Tablo 2. : Regresyon Analizi Sonuçları

	Lineer	***Ridge***	***Lasso***	***Elastik Ağ***	***Decision Tree***	***Random Forest***	***Gradient Boosting***
MSE	0.140	0.140	0.148	0.141	0.122	0.112	0.111
R^2	0.219	0.219	0.176	0.213	0.320	0.376	0.379

Tablo 2 ile regresyon analizi sonuçları karşımıza çıkmaktadır. En düşük hata değeri (MSE) 0.111 ve 0.112 ile açıklama gücü en yüksek olan 0.379 ve 0.376 değerleri ile Gradient Boosting ve Random Forest modelleri diğerlerine göre daha başarılı görünmektedir. Fakat genel olarak incelediğimizde başarı oranları 1'den çok düşük olduğu gözlemlenmektedir. Bu durum başarı oranını düşürmektedir. MSE değerlerimizin düşük olması tahminlerin gerçek değere yakın olduğunu göstermektedir. Regresyon analizi için bu verilerin çok uygun olduğu düşünülmemektedir. Sebebi ise gelir değişkeni ile araştırılan diğer değişkenler arasında güçlü bir ilişkinin olmamasıdır. Gelir durumunu iyi açıklayamamaktadır. Gelir veri setine sınıflandırma modelinin uygulanması daha uygun olacaktır. Bu sayede daha doğru sonuçlara ulaşılmış olunacaktır.

- **Sınıflandırma analizi yapılıyor.**

```
import pandas as pd

# veriden rastgele 1000 örneklem seçiliyor
new_data = data.sample(n=1000, random_state=42)

print(new_data.head())
```

```
       age  workclass  fnlwgt  education  education.num  marital.status  \
14160   29          3  280618         15             10               2
27048   19          3  439779         15             10               4
28868   28          3  204734         15             10               2
5667    35          3  107991          1              7               4
7827    20          3   54152         15             10               4

       occupation  relationship  race  sex  capital.gain  capital.loss  \
14160           5             0     4    1             0             0
27048          11             3     4    1             0             0
28868          12             5     4    0             0             0
5667           11             1     4    1             0             0
7827            0             3     4    0             0             0

       hours.per.week  native.country  income
14160              40              38       0
27048              15              38       0
28868              40              38       0
5667               45              38       0
7827               30              38       0
```

Verinin boyutundan, makine öğrenmesi analizinde yaşanılan yavaşlama ve sitemi durdurmasından kaynaklı analizlere rastgele 1000 adet veri çekerek devam edilmektedir. Eksik değer kontrolleri kontrol edildikten sonra sınıflandırma analizine başlanılıyor. Buraya kadar kullanmış olduğum Anaconda / JupyterLab alanında sistemsel yaşanılan zorlanmalardan dolayı kalan analizleri Kaggle platformunun Notebook alanından devam edilmiştir.

```
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score
```

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# kategorik değişkenleri sayısallaştırma
categorical_cols = new_data.select_dtypes(include=['object']).columns

le = LabelEncoder()
for col in categorical_cols:
    new_data[col] = le.fit_transform(new_data[col])  # düzeltildi


X = new_data.drop('income', axis=1)
y = new_data['income'] # bağımlı değişken

# veriyi %20 test ve %80 eğitim olacak şekilde bölme
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
models = {
    'Random Forest': RandomForestClassifier(),
    'Logistic Regression': LogisticRegression(max_iter=1000),
    'SVM': SVC(),
    'KNN': KNeighborsClassifier(),
    'Naive Bayes': GaussianNB(),
    'Decision Tree': DecisionTreeClassifier()
}

param_grid = {
    'Random Forest': {
        'n_estimators': [50, 100, 200],
        'max_depth': [10, 20, None],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    },
    'Logistic Regression': {
        'C': [0.1, 1, 10],
        'penalty': ['l2'],
        'solver': ['liblinear']
    },
    'SVM': {
        'C': [0.1, 1, 10],
        'kernel': ['linear', 'rbf'],
        'gamma': ['scale', 'auto']
    },
    'KNN': {
        'n_neighbors': [3, 5, 7],
        'weights': ['uniform', 'distance'],
        'metric': ['euclidean', 'manhattan']
    },
    'Naive Bayes': {

    },
    'Decision Tree': {
        'max_depth': [10, 20, None],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 2, 4]
    }
}
```

```
# GridSearchCV kullanıyorum
best_models = {}

for model_name in models:
    print(f"Training {model_name}...")

    model = models[model_name]
    grid = GridSearchCV(estimator=model, param_grid=param_grid[model_name], cv=5, n_jobs=-1, verbose=2)
    grid.fit(X_train, y_train)

    best_models[model_name] = grid.best_estimator_

    # en iyi parametre ve sonuç
    print(f"{model_name} - En iyi parametreler: {grid.best_params_}")
    print(f"{model_name} - En iyi skor: {grid.best_score_}")

    y_pred = grid.predict(X_test)

    # başarı değeri
    acc = accuracy_score(y_test, y_pred)
    print(f"{model_name} - Test Accuracy: {acc:.4f}\n")

# tüm sonuçlar
for model_name, model in best_models.items():
    print(f"En iyi model {model_name}: {model}")
```

Öncelikle gerekli kütüphaneler ve kurulumlar gerçekleştirildikten sonra test edilmek istenilen modeller şekildeki gibi belirtilmektedir. Modeli eğitirken GridSearchCV kullanarak en iyi hiperparametre seçilerek modeli bu şekilde eğitip iyi sonuçlar almayı hedeflemekteyim. Tamamı çalıştırıldıktan sonra çıkarılan analiz ise şekildedir :

```
Random Forest - En iyi skor: 0.8375
Random Forest - Test Accuracy: 0.8400

Training Logistic Regression...
Fitting 5 folds for each of 3 candidates, totalling 15 fits
Logistic Regression - En iyi parametreler: {'C': 0.1, 'penalty': 'l2', 'solver': 'liblinear'}
Logistic Regression - En iyi skor: 0.8025
Logistic Regression - Test Accuracy: 0.7700
```

Şeklinde devam etmektedir. En iyi skoru gözlemlediğimiz için devamının konulmasına ihtiyaç duyulmamıştır. En başarılı doğru sınıflandırmayı %83 oranla Lojistik regresyon modelinin yaptığı tespit edilmiştir.

- **AutoML ile sonuçlar şu şekildedir** :

```
import h2o
from h2o.automl import H2OAutoML
import pandas as pd

h2o.init()

# eğitim ve test verileri
train_data = X_train.copy()
train_data['income'] = y_train

# pandasla dönüşüm
train_h2o = h2o.H2OFrame(train_data)
test_h2o = h2o.H2OFrame(X_test)

target = 'income'
features = X_train.columns.tolist()

# kategorik dönüşüm
train_h2o[target] = train_h2o[target].asfactor()

# AutoML modelini başlat
aml = H2OAutoML(max_models=10, seed=42)
aml.train(x=features, y=target, training_frame=train_h2o)

# En iyi model
lb = aml.leaderboard
print(lb.head(rows=5))

# Tahmin yap
preds = aml.leader.predict(test_h2o)
preds_df = preds.as_data_frame()
print(preds_df.head())
```

```
stackedensemble prediction progress: |████████████████████████████████| (done) 100%
  predict       p0       p1
0       0  0.991616  0.008384
1       0  0.942138  0.057862
2       0  0.884728  0.115272
3       0  0.924767  0.075233
4       0  0.895883  0.104117
```

Bu sonuç incelendiğinde seçilen ilk 5 veri için sırasıyla doğru sınıflandırma oranlarını göstermektedir. İlkini yorumlayacak olursak maaşı 50K'dan az olan bir bireyin %99 oranla maaşı 50K'dan az olan gruba (p0) sınıflandırılması , %0.08 oranla maaşı 50K'dan fazla olan gruba (p1) göre daha yüksek sonuçtadır. Genel olarak ilk 5 birey incelendiğinde hepsinde sınıflandırmanın yüksek oranda başarılı gerçekleştiği görülmektedir.

- **İlişki Kuralı (Assoaction) analizi yapılıyor.**

```
import pandas as pd
from sklearn.preprocessing import LabelEncoder

# kategirikler sayısala dönüşüyor
label_encoders = {}
categorical_columns = ['workclass', 'education', 'marital.status', 'occupation', 'relationship', 'race', 'sex', 'native.country', 'income']

for col in categorical_columns:
    le = LabelEncoder()
    new_data[col] = le.fit_transform(new_data[col])
    label_encoders[col] = le

new_data.head()
```

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income
14160	29	2	280618	14	10	2	4	0	4	1	0	0	40	26	0
27048	19	2	439779	14	10	4	10	3	4	1	0	0	15	26	0
28868	28	2	204734	14	10	2	11	5	4	0	0	0	40	26	0
5667	35	2	107991	1	7	4	10	1	4	1	0	0	45	26	0
7827	20	2	54152	14	10	4	0	3	4	0	0	0	30	26	0

+ Code + Markdown

```
from scipy.stats import chi2_contingency

# ki-kare testi
contingency_table = pd.crosstab(new_data['education'], new_data['income'])
chi2, p, dof, expected = chi2_contingency(contingency_table)

print(f"Chi2 Değeri: {chi2}")
print(f"P Değeri: {p}")
```

```
Chi2 Değeri: 121.48210309376425
P Değeri: 3.223972850217177e-19
```

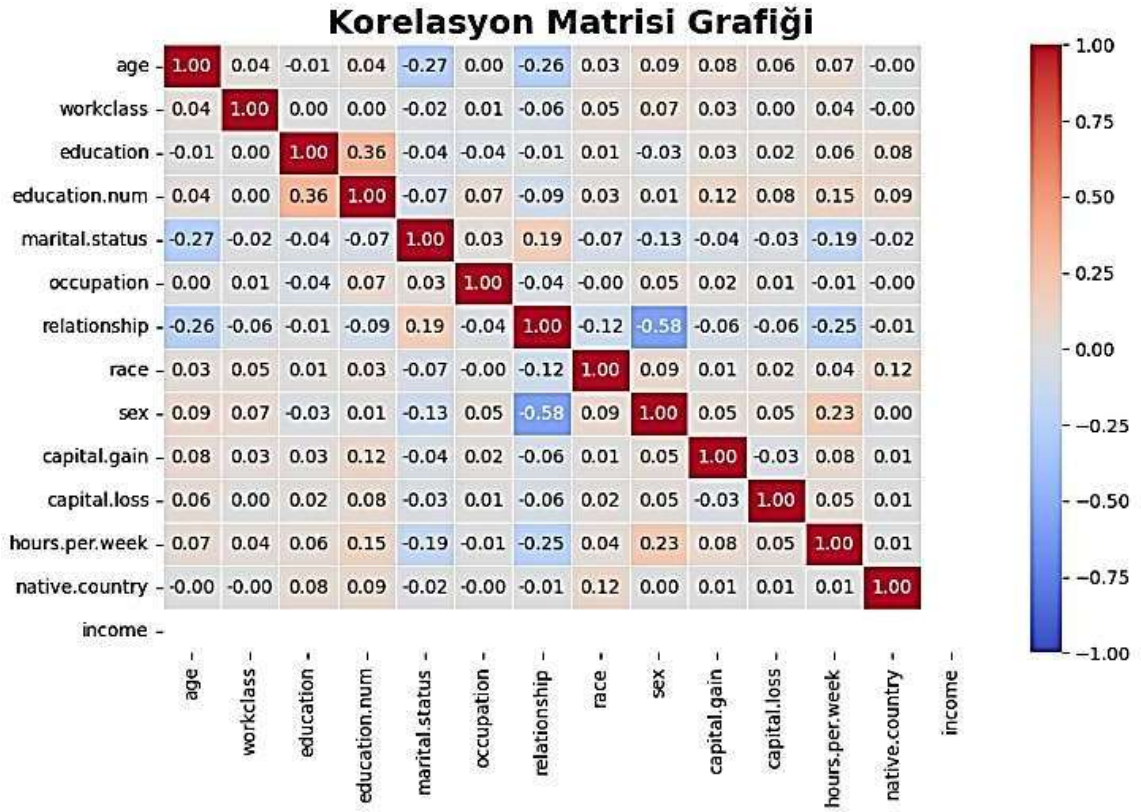
İlişki kuralı analizi yaparken ilk olarak belirli düzenlenmeler yapıldıktan sonra basit bir örnekle eğitim durumu ve gelir arsındaki ilişkinin nasıl olduğu merak edilmiştir. Ki-Kare değerinin 121 çıkması eğitim durumu ile gelir seviyesinin güçlü bir şekilde anlamlı olduğunu gösterirken p değerinin 3.22 > 0.05 olması ise %95 oranla gelir eğitim durumu ve gelir seviyesinin istatistiksel olarak anlamlı bir ilişkisi olmadığını belirtmektedir. Ki-Kare değerinin büyük olmasına rağmen p değerinin 0.05'ten baya bir yüksek olması net bir şekilde eğitim durumu ve gelir seviyesinin birbirlerinden bağımsız olduğunu göstermektedir.

```python
import numpy as np

# Cramér's V hesabı
def cramers_v(confusion_matrix):
    chi2_stat, p_val, dof, ex = chi2_contingency(confusion_matrix)
    n = confusion_matrix.sum().sum()
    phi2 = chi2_stat / n
    r, k = confusion_matrix.shape
    return np.sqrt(phi2 / min(k - 1, r - 1))

contingency_table = pd.crosstab(new_data['education'], new_data['income'])
cramers_v_value = cramers_v(contingency_table)

print(f"Cramér's V Değeri: {cramers_v_value:.4f}")
```

```
Cramér's V Değeri: 0.3485
```

Ki-Kare testinde ki sonuçtan sonra eğitim seviyesi ve gelir durumu arasında ki ilşkinin gücünü değerlendirmek için Cramer's V değeri analizi yapılmıştır. Bu analiz sonucunda 0.348 değeri ile eğitim seviyesi ile gelir durumu arasında ki ilişkinin orta düzeyde olduğu görülmektedir. Belirli bir düzeye kadar çok güçlü olmasa da belirli bir ilişkilerinin olduğu aynı zamanda çok da bağımsız olmadıklarını göstermektedir.

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

# income değişkeni <=50K ya da >50K değerlerini 0 ve 1 olacak şekilde düzenliyorum
data['income'] = data['income'].map({'<=50K': 0, '>50K': 1})

# fnlwgt'i çıkarıyoruz ve income'yu bağımlı değişken olarak alıyoruz
numerical_data = data.select_dtypes(include=['int64', 'float64']).drop('fnlwgt', axis=1)

# korelasyon hesaplama
correlation_matrix = numerical_data.corr()

# görselleştirme
plt.figure(figsize=(10,6))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5, vmin=-1, vmax=1)

# Grafik başlığı
plt.title('Korelasyon Matrisi Grafiği', fontsize=18, fontweight='bold')

plt.show()
```



Tüm değişkenlerin birbirleriyle ilişkisini daha iyi anlamlandırılması adına korelasyon matrisi grafiği yapılmıştır. Eğitim ile eğitim sayısı arasında ki ilişkinin 0.36 ile pozitif iyi bir ilişkiye sahip olduğunu gösterirken , ilişki ile cinsiyet arasında ilişkinin de -0.58 ile orta düzeyde ilerleyerek bir değişken olumlu etkilenirken bir değişkenin olumsuz etkilenerek ilerlediğini göstermektedir.

- **<u>AutoMl analizi ile sonuçlar şu şekildedir :</u>**

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder

categorical_cols = ['workclass', 'education', 'marital.status', 'occupation',
                    'relationship', 'race', 'sex', 'native.country']

data_list = new_data[categorical_cols].values.tolist()

#  TransactionEncoder ile dönüşüm
te = TransactionEncoder()
te_ary = te.fit(data_list).transform(data_list)

df = pd.DataFrame(te_ary, columns=te.columns_)
```

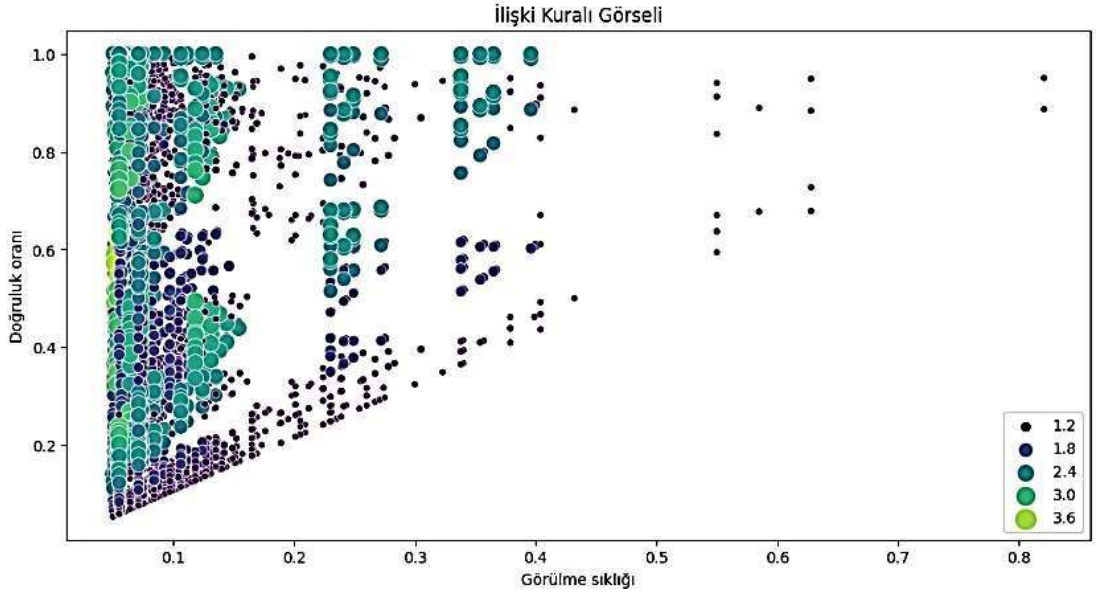
+ Code + Markdown

```
from mlxtend.frequent_patterns import apriori, association_rules

# Apriori algoritması en sık görülen item set'leri bulmamıza yardımcı oluyor
frequent_itemsets = apriori(df, min_support=0.05, use_colnames=True)

#ilişki kuralı
rules = association_rules(frequent_itemsets, metric="lift", min_threshold=1.0)

print(rules.head())
```

```
       antecedents       consequents  antecedent support  consequent support  \
0   (Adm-clerical)          (Female)               0.107               0.342
1          (Female)   (Adm-clerical)               0.342               0.107
2   (Adm-clerical)  (Never-married)               0.107               0.351
3  (Never-married)   (Adm-clerical)               0.351               0.107
4   (Adm-clerical)  (United-States)               0.107               0.926

    support  confidence      lift  representativity  leverage  conviction  \
0     0.070    0.654206  1.912882               1.0  0.033406    1.902865
1     0.070    0.204678  1.912882               1.0  0.033406    1.122816
2     0.050    0.467290  1.331310               1.0  0.012443    1.218298
3     0.050    0.142450  1.331310               1.0  0.012443    1.041339
4     0.102    0.953271  1.029450               1.0  0.002918    1.583600
```

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

# ilişki kuralının görseli
plt.figure(figsize=(12, 6))
sns.scatterplot(x='support', y='confidence', data=rules, hue='lift', size='lift', sizes=(20, 200), palette='viridis')
plt.title('İlişki Kuralı Görseli')
plt.xlabel('Görülme sıklığı')
plt.ylabel('Doğruluk oranı')
plt.legend()
plt.show()
```

Burada ilişki kurallarını daha ayrıntılı incelemek için yapılan çalışma mevcuttur. Confidence ile kuralın doğru olma olasılığını incelerken Lift ile kuralın veri setinde ne kadar güçlü olduğunu göstermektedir. Buradan yorumlamanın daha zor olacağı düşünüldüğü için grafiklendirilerek bu durum daha iyi ortaya çıkmaktadır.

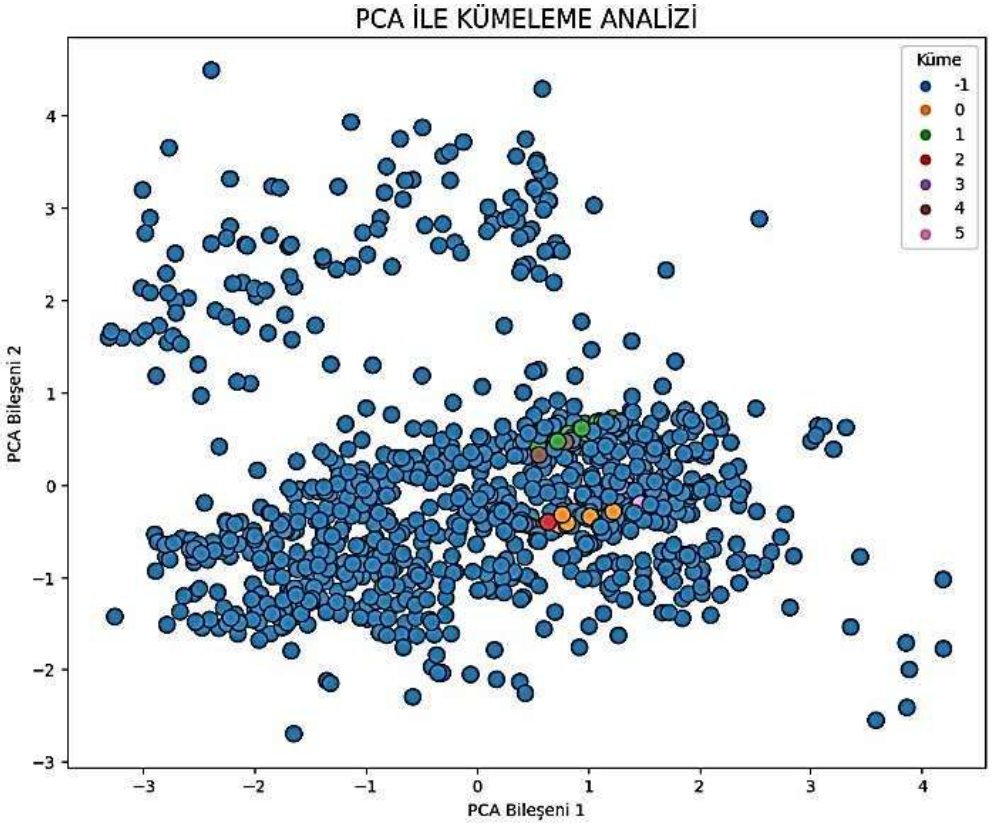


Grafik incelendiğinde doğruluk oranımız 1 veya 1'e doğru yanaştıkça 3.0 olan beneğimizin büyük olasılıkla görülme sıklıkları da fazladır. Tam tersi doğruluk oranımız 0.2'den aşağı düştükçe bol bol 1.2 büyüklükte beneğin fazla olduğu görülmektedir. Grafik bize daha çok büyük beneklerle daha canlı ve net renkleri ortaya koyduğu için değişkenlerin diğer değişkenlerle ilişkisinin daha güçlü olduğunu göstermektedir.

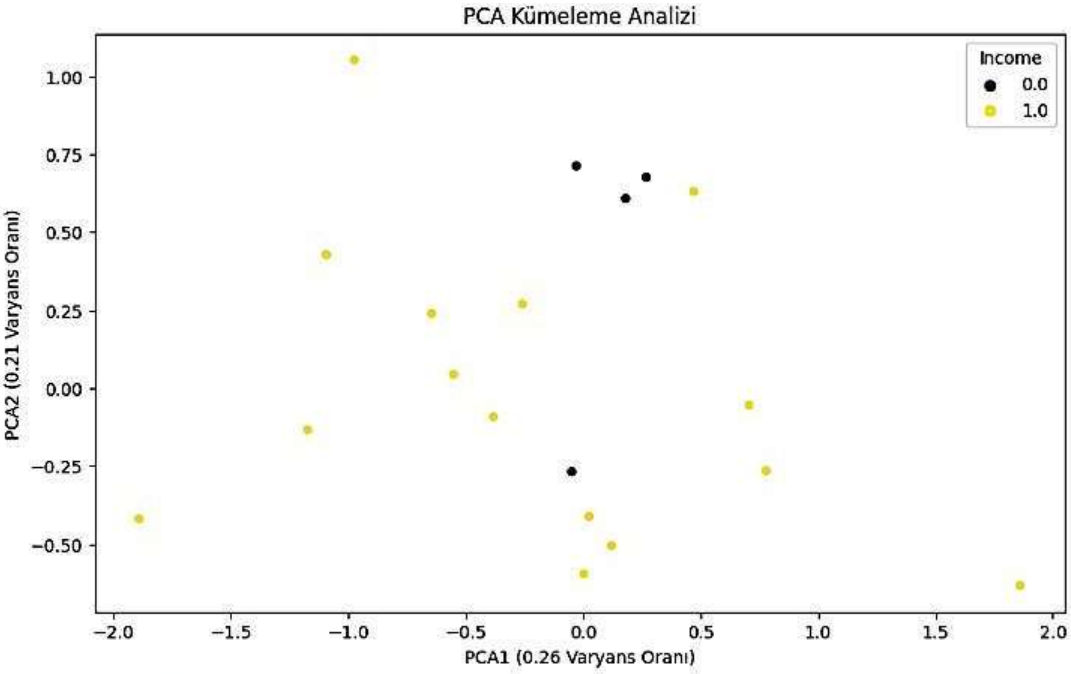
- **Kümeleme analizi yapılıyor.**

Gerekli kütüphaneler , dönüşümler ve düzenlemeler gerçekleştikten sonra temel bileşenler analizi ile kümele analizi gerçekleştiriliyor. Analiz sonucunda oluşan PCA1 ve PCA2 bize analizde en yüksek varyans ile modeli açıklayan değerler olduğunu göstermektedir. Aşağıda yapılan analiz sonucu oluşturulan grafik ile varyansı en yüksek olan PCA bileşini 1'e doğru yoğunluğun fazla olduğu görülmektedir.

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.cluster import DBSCAN
from sklearn.decomposition import PCA

# kategorik dönüşüm
categorical_cols = ['workclass', 'education', 'marital.status', 'occupation',
                    'relationship', 'race', 'sex', 'native.country']
numerical_cols = ['age', 'education.num', 'capital.gain', 'capital.loss', 'hours.per.week']

le = LabelEncoder()
for col in categorical_cols:
    new_data[col] = le.fit_transform(new_data[col])

data_for_clustering = new_data[categorical_cols + numerical_cols]

# standartlaştırma
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data_for_clustering)

# DBSCAN ile kümeleme oluşturma
dbscan = DBSCAN(eps=0.5, min_samples=5)
new_data['cluster'] = dbscan.fit_predict(data_scaled)

# PCA ile görselleştirme
pca = PCA(n_components=2)
pca_components = pca.fit_transform(data_scaled)

new_data['PCA1'] = pca_components[:, 0]
new_data['PCA2'] = pca_components[:, 1]

plt.figure(figsize=(10, 8))
sns.scatterplot(x='PCA1', y='PCA2', hue='cluster', data=new_data, palette='tab10', s=100, marker='o', edgecolor='k')

plt.title('PCA ILE KÜMELEME ANALİZİ', fontsize=15)
plt.xlabel('PCA Bileşeni 1')
plt.ylabel('PCA Bileşeni 2')

# Küme merkezlerini çizme
plt.legend(title='Küme', loc='upper right')
plt.show()
```



```python
import pandas as pd
from sklearn.decomposition import PCA
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

# sayısalları seçme
numerical_cols = ['age', 'education.num', 'capital.gain', 'capital.loss', 'hours.per.week']

# normalizasyon
scaler = StandardScaler()
new_data_scaled = scaler.fit_transform(new_data[numerical_cols])

pca = PCA(n_components=2)
pca_components = pca.fit_transform(new_data_scaled)

# PCA bileşenleri ile dataframe birleşimi
pca_df = pd.DataFrame(data=pca_components, columns=['PCA1', 'PCA2'])

# varyans oranlarını görme
explained_variance = pca.explained_variance_ratio_

plt.figure(figsize=(10, 6))
sns.scatterplot(x=pca_df['PCA1'], y=pca_df['PCA2'], hue=new_data['income'], palette='viridis')
plt.title('PCA Kümeleme Analizi')
plt.xlabel(f'PCA1 ({explained_variance[0]:.2f} Varyans Oranı)')
plt.ylabel(f'PCA2 ({explained_variance[1]:.2f} Varyans Oranı)')
plt.legend(title='Income')
plt.show()

print(f'PCA1 Varyans Oranı: {explained_variance[0]:.2f}')
print(f'PCA2 Varyans Oranı: {explained_variance[1]:.2f}')
```




```
PCA1 Varyans Oranı: 0.26
PCA2 Varyans Oranı: 0.21
```

Yaş , eğitim sayısı , saatlik haftada çalışması , sermaye kazancı ve kaybı ile maaşı 50 K'dan fazla veya az olması durumlarına göre yapılan temel bileşenler analizi sonucu ile çıkarılan 0.26 varyans oranı bize maaşı 50K'dan yükse olan bireylerin bu değişkenler ile maaşı 50K'dan az olanlara göre temel bileşen analizi 1 tarafında yoğun ve çok olduğunu göstermektedir.

- **<u>AutoML ile analiz edildiğinde çıkarılan sonuç şu şekildedir :</u>**

```
import h2o
from h2o.estimators import H2OKMeansEstimator
import pandas as pd

h2o.init()

# Kmeans modelini oluşturma
data_h2o = h2o.H2OFrame(new_data)
kmeans = H2OKMeansEstimator(k=3, seed=42)
kmeans.train(x=numerical_cols, training_frame=data_h2o)

# küme sonuçları
clusters = kmeans.predict(data_h2o)

data_h2o['cluster'] = clusters['predict']
data_h2o.head()
```

```
Parse progress: |████████████████████████████████████████████████| (done) 100%
kmeans Model Build progress: |████████████████████████████████████| (done) 100%
kmeans prediction progress: |████████████████████████████████████| (done) 100%
```

age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationship	race	sex	capital.gain	capital.loss	hours.per.week	native.country	income	cluster	PCA1	PCA2
29	2	280618	14	10	2	4	0	4	1	0	0	40	26	0	0	0.790021	-0.425018
19	2	439779	14	10	4	10	3	4	1	0	0	15	26	0	0	-1.52207	-0.936182
28	2	204734	14	10	2	11	5	4	0	0	0	40	26	0	0	-1.67081	-1.79965
35	2	107991	1	7	4	10	1	4	1	0	0	45	26	0	0	-0.343235	2.59277
20	2	54152	14	10	4	0	3	4	0	0	0	30	26	0	0	-2.17015	-1.40432
36	2	240823	14	10	6	0	4	2	0	0	0	40	26	0	0	-2.54205	-1.17941
46	2	279196	9	13	4	1	1	4	0	0	0	40	26	0	0	-0.49654	-0.625684
35	2	48123	2	8	2	1	5	4	0	0	0	50	26	0	0	-2.03949	1.09888
28	2	166481	5	4	2	4	0	3	1	0	2179	40	23	0	2	0.456418	2.39246
23	2	129042	11	9	4	5	4	2	0	0	0	40	26	0	0	-2.49958	-0.673123

[10 rows x 18 columns]

Tablo sonucunda gösterilen verileri yorumlayacak olursak :

Yaklaşık olarak 33 bin bireyin olduğu ve elde edilen değişkenler sonucunda yapılan analizler ile rastgele 1000 örneklemin seçilerek , kümelendirilerek analiz yapılması sonucunda çıkarılan sonuçlar da gelir durumu ve sınıflandırılmasına bakıldığında çoğu bireyin doğru şekilde kümelendiğini göstermektedir. PCA 1 ve PCA2 değerlerini incelediğimizde daha çok negatif olarak değerler alarak kümelendiğini yani bireylerin değişkenler açısından genel olarak elde edilen toplam varyans değerlerine göre eğitim , gelir durumu , yaşı vb durumlarla dezavantajlı bir grubu temsil ettiği görülmektedir.

```
from h2o.automl import H2OAutoML

data_h2o['target'] = data_h2o['cluster']

# AutoML eğitimi
aml = H2OAutoML(max_models=10, seed=42, max_runtime_secs=300)
aml.train(x=numerical_cols, y='target', training_frame=data_h2o)

aml.leader
```

Cross-Validation Metrics Summary:

	mean	sd	cv_1_valid	cv_2_valid	cv_3_valid	cv_4_valid	cv_5_valid
aic	86.49582	58.353813	90.39273	-6.0454764	142.01857	129.61124	76.502014
loglikelihood	0.0	0.0	0.0	0.0	0.0	0.0	0.0
mae	0.1343602	0.0189441	0.1365154	0.1029192	0.1531463	0.1437509	0.1354693
mean_residual_deviance	0.0874231	0.0227483	0.0863113	0.0534249	0.1118028	0.1042096	0.0813670
mse	0.0874231	0.0227483	0.0863113	0.0534249	0.1118028	0.1042096	0.0813670
null_deviance	116.70346	14.754754	135.52673	120.28758	110.62398	121.45055	95.62848
r2	0.8467166	0.0450700	0.8695612	0.9124423	0.7963138	0.8249249	0.8303411
residual_deviance	17.52773	4.7412205	17.43489	10.524704	22.472366	21.25875	15.947937
rmse	0.2934719	0.0402704	0.2937879	0.2311383	0.3343693	0.3228151	0.2852491
rmsle	0.1717030	0.0217281	0.1669916	0.1379571	0.1864776	0.1942387	0.1728498

Burada 5 katlı çapraz doğrulama analizi yapılarak model hakkında emin olunmak istenilmiştir. R^2 değerinin 0.846 ile modelin değişkenlerle gözlemlenen toplam varyansı %84 oranla açıkladığını göstermektedir. Ortalama mutlak hatanın 0.13 olması ise bize ortalama olarak gerçek değerden 0.13 birim saptığını belirtmektedir. Artık sapmanın ortalaması ise 0.087 ile tahminlemenin veriyle oldukça doğru olduğunu göstermektedir. Genel olarak incelediğimizde R^2 değerinin de yüksek çıkması ile modelin doğru tahminler yaptığını göstermektedir.


LOTUS AI
YAPAY ZEKA VE BİLİŞİM ÇÖZÜMLERİ A.Ş.