Veri Seti İçeriği

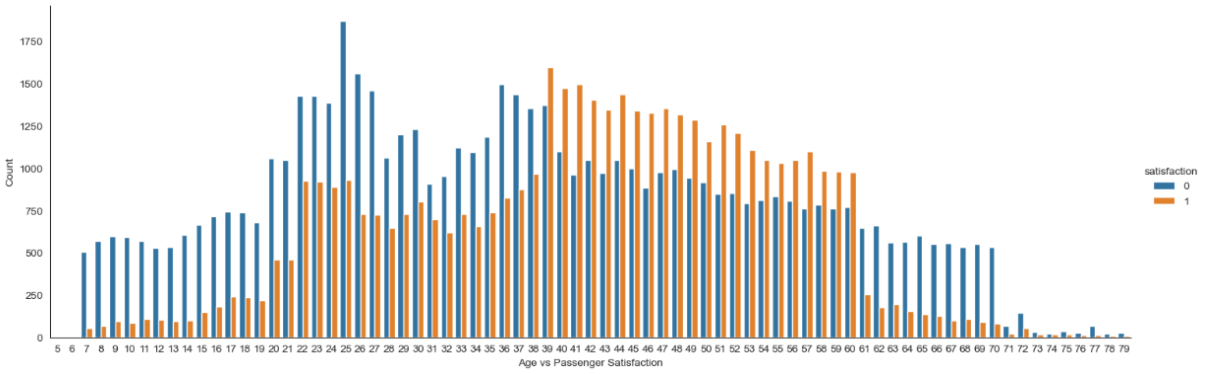
1. **Cinsiyet**: Yolcuların cinsiyeti (Kadın, Erkek)
2. **Müşteri Türü:** Müşteri türü (Sadık müşteri, sadık olmayan müşteri)
3. **Yaş:** Yolcuların gerçek yaşı
4. **Seyahat Türü:** Yolcuların uçuş amaçları (Kişisel Seyahat, İş Seyahati)
5. **Sınıf:** Yolcuların uçak sınıfı (İş, Ekonomi, Ekonomi Plus)
6. **Uçuş Mesafesi:** Bu yolculuğun uçuş mesafesi
7. **Uçak içi wifi hizmeti:** Uçak içi wifi hizmetinin memnuniyet düzeyi (0: Uygulanamaz; 1-5)
8. **Kalkış/Varış zamanı uygunluğu:** Kalkış/Varış zamanının uygunluk memnuniyet düzeyi
9. **Online rezervasyon kolaylığı:** Online rezervasyon memnuniyet düzeyi
10. **Kapı konumu:** Kapı konumu memnuniyet düzeyi
11. **Yiyecek ve içecek:** Yiyecek ve içecek memnuniyet düzeyi
12. **Online tahtadan biniş:** Online tahtadan biniş memnuniyet düzeyi
13. **Koltuk konforu:** Koltuk konforu memnuniyet düzeyi
14. **Uçak içi eğlence**: Uçak içi eğlence memnuniyet düzeyi
15. **Güzergah hizmeti**: Güzergah hizmeti memnuniyet düzeyi
16. **Ayak boşluğu hizmeti**: Ayak boşluğu hizmeti memnuniyet düzeyi
17. **Bagaj işleme**: Bagaj işleme memnuniyet düzeyi
18. **Check-in hizmeti**: Check-in hizmeti memnuniyet düzeyi
19. **Uçak içi hizmet**: Uçak içi hizmet memnuniyet düzeyi
20. **Temizlik**: Temizlik memnuniyet düzeyi
21. **Kalkışta Gecikme (Dakika cinsinden):** Kalkışta gecikme süresi
22. **Varışta Gecikme (Dakika cinsinden):** Varışta gecikme süresi
23. **Memnuniyet:** Hava yolu memnuniyet düzeyi (Memnuniyet, nötr veya memnuniyetsizlik)

Veri seti incelendiğinde göze çarpan ilk detay, kalkışta gecikme ile varışta gecikme süreleridir. Bu süreler birbirini etkilemektedir. Örneğin kalkışta yaşanan 30 dakikalık bir gecikme, muhtemelen varışta da o kadar gecikmeye sebebiyet verecektir. Ayrıca, yolcuların memnuniyetini etkileyebilecek diğer faktörlere de odaklanmak önemlidir.

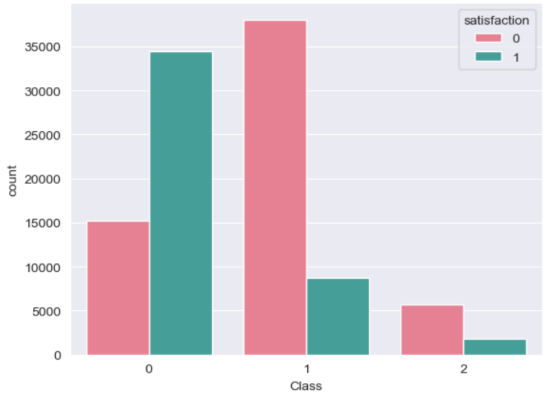
Örneğin, koltuk konforu, uçak içi eğlence ve yiyecek içecek hizmeti gibi faktörler, genel yolcu memnuniyetini belirlemede etkili olabilir. Aynı şekilde, online rezervasyon kolaylığı, check-in hizmeti ve uçak içi wifi gibi özellikler de müşteri memnuniyetini etkileyebilir.

Veri setindeki cinsiyet ve yaş gibi demografik faktörler de memnuniyet üzerinde etkili olabilir. Örneğin, belirli bir yaş grubundaki veya cinsiyetteki yolcuların belirli hizmetlere olan talepleri farklılık gösterebilir.

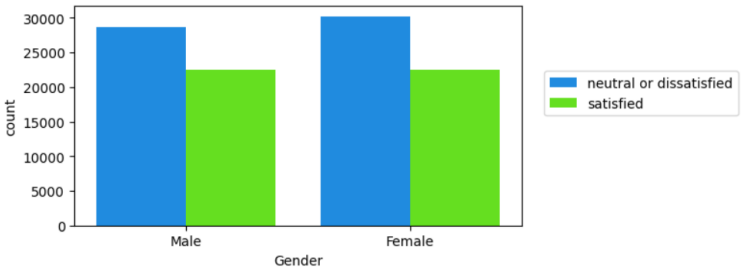
Veri Setiyle ilgili İstatistikler



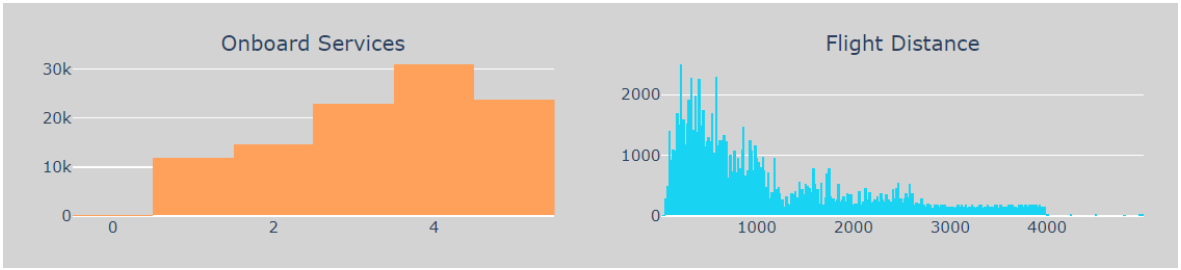
Erkek ve kadın yaşa göre memnuniyet dağılımı



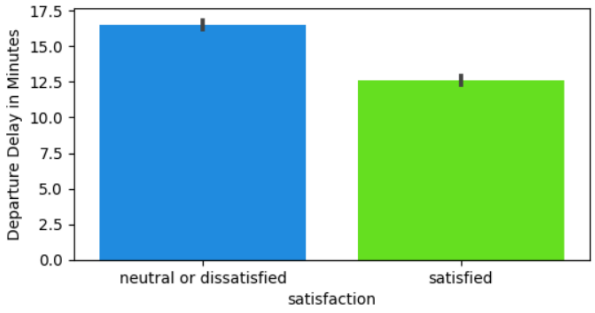
Erkek (yeşil) ve kadın(pembe) yolculuk sınıfları



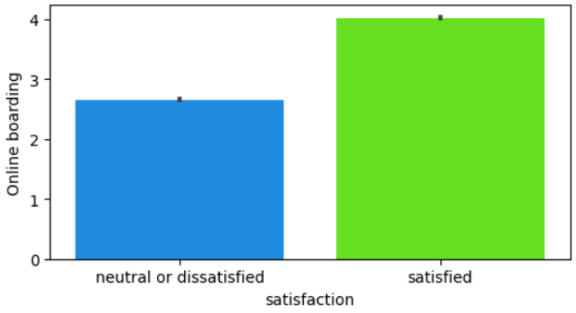
Erkek ve Kadın memnuniyet durumları



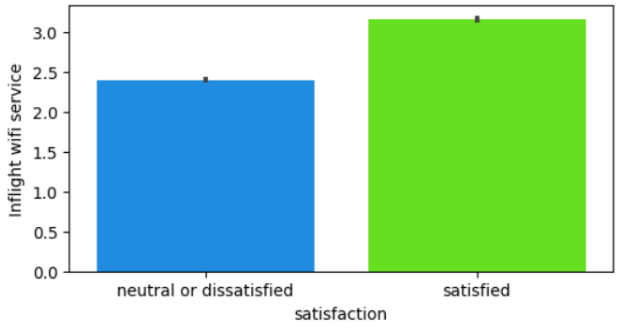
Onboard hizmeti ve uçuş mesafesi dağılımı



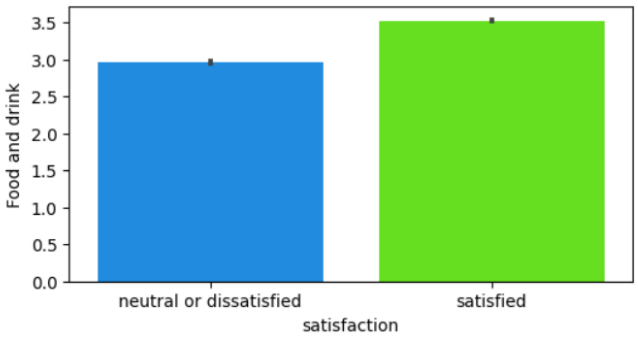
Gecikmenin memnuniyete etkisi



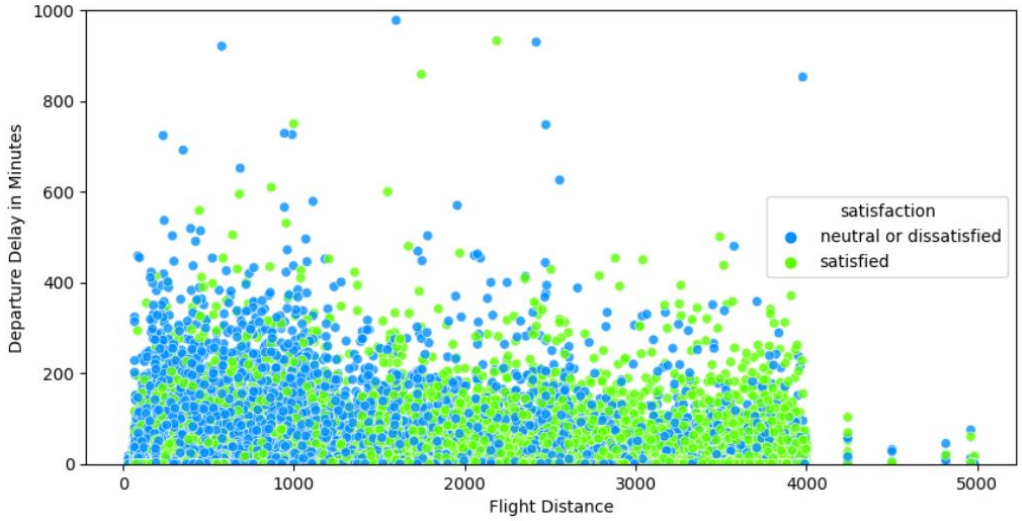
Çevrimiçi binişin memnuniyete etkisi



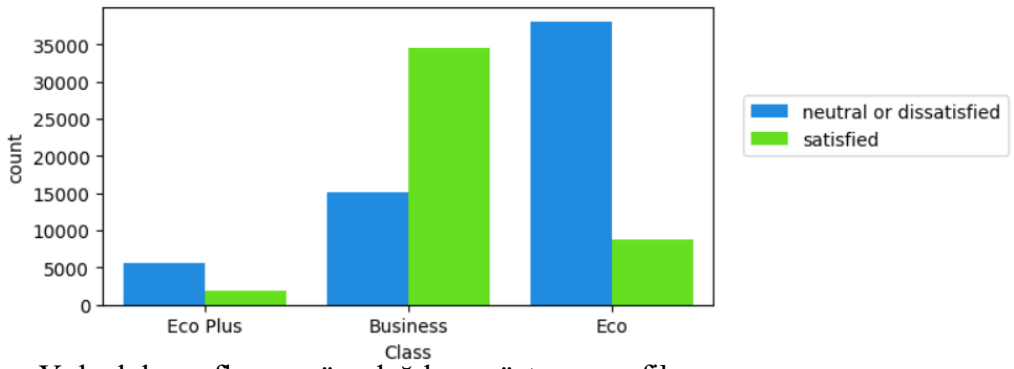
Wifi servisinin etkisi



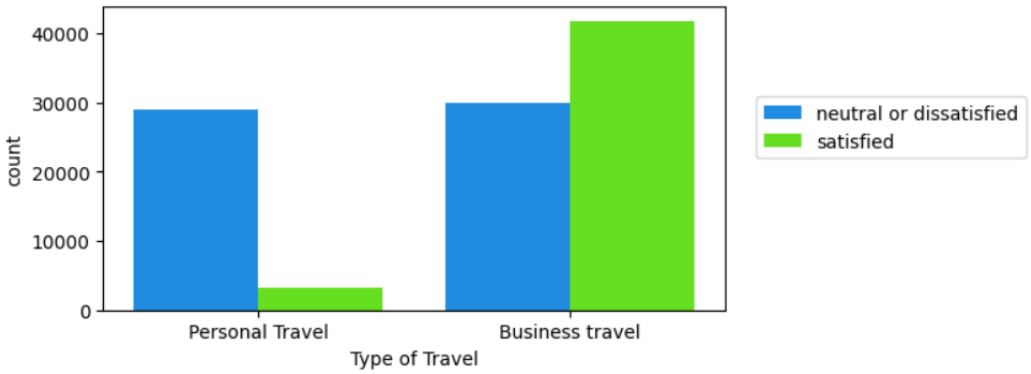
Yemek ve içeceğin etkisi



Bu dağılım bize kalkışta yaşanan gecikmenin memnuniyet üzerindeki etkisini göstermektedir. Grafiği incelediğimizde, kısa mesafe yolculuklarda yaşanan gecikmelerin olduğu insanlarda genellikle memnuniyetsiz görüş, uzun mesafeli yolculuklarda yaşanan gecikmelerde ise insanlar memnuniyet ile memnuniyetsizlik yakın olarak dağılmıştır. Bu grafik bize kısa mesafelerde yaşanan gecikmelerin memnuniyetsizlik üzerinde daha fazla etkisi olduğunu göstermektedir.

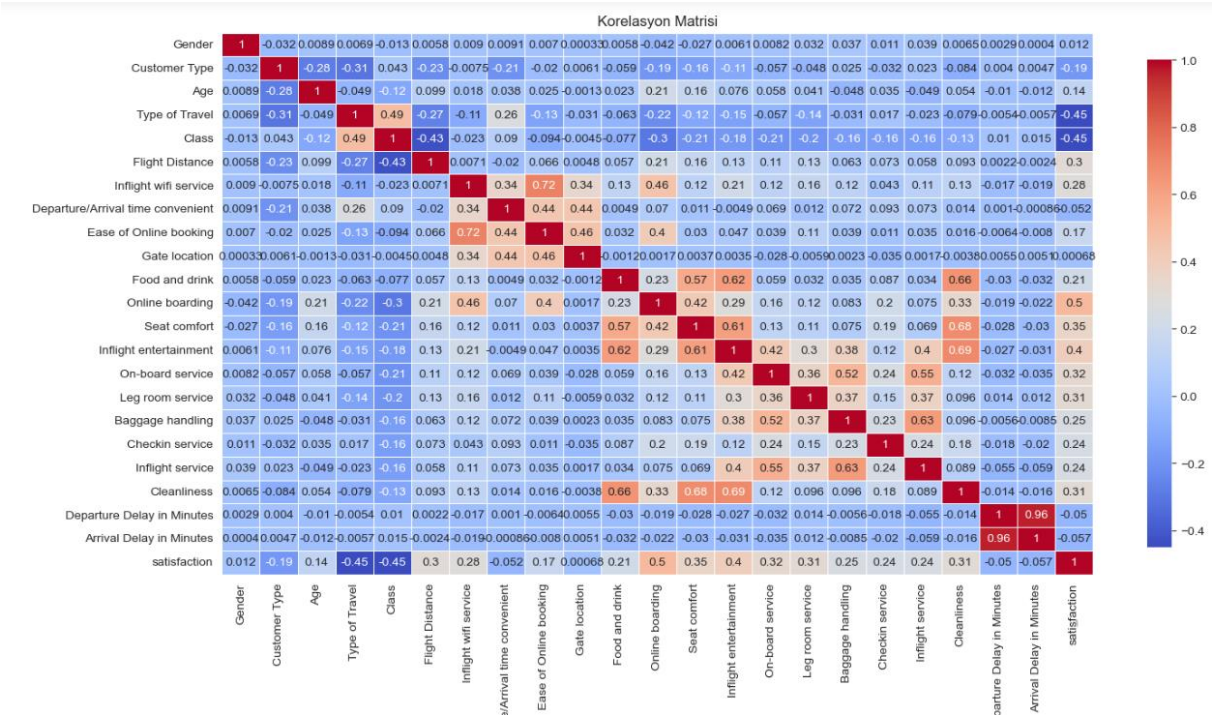


Yolculuk sınıflarına göre dağılımı gösteren grafik



Yolculuk türünün dağılımını gösteren grafik

Korelasyon Matrisi



Korelasyon matrisi incelendiğinde, kalkıştaki gecikme ile varıştaki gecikmenin birbirini çok etkilediği görülmektedir. Bu nedenle ilk olarak ikisinden biri kaldırılacaktır. Temizlik özelliği de koltuk konforu, yemek, içecek ile yakından ilgili olduğundan dolayı veri setinden kaldırılmasına karar verilmiştir. Ayrıca veri setindeki 'Unnamed: 0' ve 'id' sınıflarında memnuniyete bir etkisi olmadığından dolayı kaldırılmıştır.

Veri Setleri Tipleri

```
-----Sutün--------------Veri Tipi--------
Unnamed: 0                                int64
id                                        int64
Gender                                   object
Customer Type                            object
Age                                       int64
Type of Travel                           object
Class                                    object
Flight Distance                           int64
Inflight wifi service                     int64
Departure/Arrival time convenient         int64
Ease of Online booking                    int64
Gate location                             int64
Food and drink                            int64
Online boarding                           int64
Seat comfort                              int64
Inflight entertainment                    int64
On-board service                          int64
Leg room service                          int64
Baggage handling                          int64
Checkin service                           int64
Inflight service                          int64
Cleanliness                               int64
Departure Delay in Minutes                int64
Arrival Delay in Minutes                float64
satisfaction                             object
```

Kütüphane

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from plotly.subplots import make_subplots
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix,precision_score, recall_score, f1_score
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
import plotly.graph_objects as go
from xgboost import XGBClassifier
import time
```

numpy: Bilimsel hesaplama için kullanılan Python kütüphanesidir. Çok boyutlu diziler ve matrisler üzerinde yüksek performanslı çalışma imkanı sunar.

pandas: Veri manipülasyonu ve veri analizi için kullanılan Python kütüphanesidir. Veri yapılarını ve analiz araçlarını sağlar.

matplotlib.pyplot: Veri görselleştirmesi için kullanılan bir Python kütüphanesidir. Grafik ve çizimler oluşturmak için kullanılır.

seaborn: Matplotlib'in üzerine inşa edilmiş bir veri görselleştirme kütüphanesidir. Veri görselleştirmesini daha basit ve çekici hale getirir.

plotly.subplots: Plotly kütüphanesinin bir alt modülüdür. İnteraktif grafikler oluşturmak için kullanılır.

sklearn: Scikit-learn, Python'da kullanılabilen popüler bir makine öğrenimi kütüphanesidir. Sınıflandırma, regresyon, kümeleme ve boyut azaltma gibi birçok makine öğrenimi algoritmasını içerir.

XGBClassifier: Extreme Gradient Boosting (XGBoost) algoritmasını uygulayan bir sınıflandırıcıdır.

PCA: Principal Component Analysis (Temel Bileşen Analizi) için kullanılan bir sınıf içerir. Veri setinin boyutunu azaltmak için kullanılır.

PassengerSatisfactionAnalysis Class Yapısı

```python
class PassengerSatisfactionAnalysis:
    def __init__(self, train_path, test_path):
        self.dataTrain = pd.read_csv(train_path)
        self.dataTest = pd.read_csv(test_path)

    def fill_na_mean(self):
        mean_arrival_delay = self.dataTrain['Arrival Delay in Minutes'].mean()
        self.dataTrain['Arrival Delay in Minutes'].fillna(mean_arrival_delay, inplace=True)
        mean_arrival_delay2 = self.dataTest['Arrival Delay in Minutes'].mean()
        self.dataTest['Arrival Delay in Minutes'].fillna(mean_arrival_delay2, inplace=True)

    def drop_columns(self,colums):
        self.dataTrain = self.dataTrain.drop(columns=colums, axis=1)
        self.dataTest = self.dataTest.drop(columns=colums, axis=1)

    def data_info(self):
        print('-----------Boyutu---------')
        print(self.dataTrain.shape)
        print('-----Sutün--------------Veri Tipi-------- ')
        print(self.dataTrain.dtypes)
        print('--------Benzersiz Değer--------')
        print(self.dataTrain.nunique())
        print('--------Boş Değer----------')
        print(self.dataTrain.isnull().sum())
        print('--------Veri Özeti-------')
        print(self.dataTrain.describe(include='all'))

    def label_encode_categorical_columns(self):
        object_columns = self.dataTrain.select_dtypes(include=['object']).columns
        label_encoder = LabelEncoder()
        for column in object_columns:
            self.dataTrain[column] = label_encoder.fit_transform(self.dataTrain[column])
            self.dataTest[column] = label_encoder.transform(self.dataTest[column])

    def plot_correlation_matrix(self):
        corr_matrix = self.dataTrain.corr()
        plt.figure(figsize=(16, 8))
        sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=.5, cbar_kws={"shrink": 0.9})
        plt.title("Korelasyon Matrisi")
        plt.show()
```

init(): Bu, sınıfın başlatıcı yöntemidir. İlk olarak, eğitim ve test veri setlerinin dosya yollarını alır ve bu dosyalardan veri setlerini yükler.

fill_na_mean(): Eksik değerleri ortalama ile doldurmak için kullanılır. 'Arrival Delay in Minutes' sütunundaki eksik değerler, ilgili veri setinin ortalama varış gecikmesi ile doldurulur.

drop_columns(): Belirtilen sütunları (colums) düşürmek için kullanılır. Bu, gereksiz veya analizde kullanılmayacak sütunların veri setinden çıkarılmasını sağlar.

data_info(): Veri setinin genel bilgilerini yazdırmak için kullanılır. Boyut, sütun türleri, benzersiz değerler, eksik değerler ve veri özeti gibi bilgileri görüntüler.

label_encode_categorical_columns(): Kategorik sütunları etiket kodlayıcı ile kodlamak için kullanılır. Bu, kategorik verileri sayısal değerlere dönüştürmek için kullanılır, çünkü makine öğrenimi algoritmaları genellikle sayısal verilerle çalışır.

plot_correlation_matrix(): Veri setindeki özellikler arasındaki korelasyonu görselleştirmek için kullanılır. Bu, özellikler arasındaki ilişkiyi anlamak ve gerektiğinde özellik seçimini yapmak için kullanışlı olabilir.

Eğitim Veri Setindeki Tekrar, Violin Plot Ve Subplots Histogramı Çizimi

```python
def find_duplicate_rows(self):
    duplicated_rows = self.dataTrain[self.dataTrain.duplicated()]
    if not duplicated_rows.empty:
        print("Tekrar eden satırlar:")
        print(duplicated_rows)
    else:
        print("Tekrar eden satırlar bulunmamaktadır.")

def plot_violin(self, x_column, y_column, category_column, title='', x_title='', y_title=''):
    categories = self.dataTrain[category_column].unique()
    fig = go.Figure()
    for category in categories:
        category_data = self.dataTrain[self.dataTrain[category_column] == category]
        fig.add_trace(go.Violin(x=category_data[x_column],
                                line_color='lightseagreen' if category == 'satisfied' else 'red',
                                y0=0, name=f'{category} passengers {y_column}'))
    fig.update_traces(orientation='h', side='positive', meanline_visible=True)
    fig.update_layout(title=f'<b>{title}<b>',
                      titlefont={'size': 20},
                      xaxis_zeroline=False,
                      paper_bgcolor='lightgrey',
                      plot_bgcolor='lightgrey')
    fig.update_xaxes(showgrid=False, title=f'<b>{x_title}<b>')
    fig.update_yaxes(title=f'<b>{y_title}<b>')
    fig.update_traces(opacity=0.7)
    fig.show()

def plot_histograms_subplots(self, features, nbins=20, title='', title_size=30):
    total_plots = len(features)
    cols = 2
    rows = (total_plots + 1) // cols
    fig = make_subplots(rows=rows, cols=cols, subplot_titles=features)
    for i, feature in enumerate(features, start=1):
        fig.add_trace(go.Histogram(x=self.dataTrain[feature], nbinsx=nbins, name=feature),
                      row=i // cols + 1, col=i % cols + 1)
    fig.update_layout(title_text=f'<b>{title}<b>',
                      title_font=dict(size=title_size),
                      paper_bgcolor='lightgrey',
                      plot_bgcolor='lightgrey',
                      showlegend=False,
                      height=rows * 300)
    fig.show()
```

Find_duplicate_rows(self):

- Bu yöntem, eğitim veri setindeki tekrar eden satırları bulur.
- `self.dataTrain.duplicated()` ile her bir satırın, daha önce gözlemlenen bir satır olup olmadığını belirleyen bir mask oluşturulur.
- `self.dataTrain[self.dataTrain.duplicated()` ile bu maskeden yararlanarak tekrar eden satırlar elde edilir.
- Eğer tekrar eden satırlar varsa, bu satırlar ekrana yazdırılır. Aksi takdirde, "Tekrar eden satırlar bulunmamaktadır." mesajı görüntülenir.

Plot_violin(self, x_column, y_column, category_column, title='', x_title='', y_title=''):

- Bu yöntem, keman grafiği (violin plot) çizer.
- `go.Violin` ile her bir kategori için keman grafiği oluşturulur. Kategoriler, veri setindeki bir kategorik sütuna (category_column) dayanır.
- Grafiğin x ve y ekseni sütunları, veri setindeki sayısal sütunlardan (x_column ve y_column) alınır.
- Keman grafiği, 'satisfied' kategorisi için yeşil, diğer kategoriler için kırmızı renkte çizilir.
- Grafik, Plotly kütüphanesi kullanılarak oluşturulur ve gösterilir.

Plot_histograms_subplots(self, features, nbins=20, title='', title_size=30):

- Bu yöntem, alt grafikler (subplots) içinde histogramlar çizer.
- `make_subplots` ile belirtilen özellikler (features) için alt grafikler oluşturulur.
- Her bir özellik için bir histogram çizilir ve alt grafiklerde uygun konumlara yerleştirilir.
- Histogramlar, veri setindeki belirtilen özelliklerin dağılımını gösterir.
- Grafik, Plotly kütüphanesi kullanılarak oluşturulur ve gösterilir.

Veri setindeki belirli özelliklere göre Histogramlar Çizmek ve Analiz Kısmı

```
    def plt_histogram(self):
        sns.countplot(x='Class', hue='satisfaction', palette="YlOrBr", data=self.dataTrain)
        plt.show()
        with sns.axes_style(style='ticks'):
            g = sns.catplot(x="satisfaction", col="Gender", col_wrap=2, data=self.dataTrain, kind="count", height=4.5,
                            aspect=1.0)
        g.set_axis_labels("Satisfaction", "Count")
        g.set_titles(col_template="{col_name}")
        plt.show()
        with sns.axes_style('white'):
            g = sns.catplot(x="Age", data=self.dataTrain, aspect=3.0, kind='count', hue='satisfaction', order=range(5, 80))
            g.set_ylabels('Count')
            g.set_xlabels('Age vs Passenger Satisfaction')
            plt.show()
        self.dataTrain[['Gender', 'satisfaction']].groupby(['Gender'], as_index=False).mean().sort_values(
            by='satisfaction', ascending=False)

    def prepare_data_for_models(self, apply_pca=False, n_components=None):
        xTrain = self.dataTrain.drop(columns='satisfaction', axis=1)
        xTest = self.dataTest.drop(columns='satisfaction', axis=1)
        yTrain = self.dataTrain['satisfaction']
        yTest = self.dataTest['satisfaction']

        scaler = StandardScaler()
        xTrain = scaler.fit_transform(xTrain)
        xTest = scaler.transform(xTest)

        if apply_pca:
            pca = PCA(n_components=n_components)
            xTrain = pca.fit_transform(xTrain)
            xTest = pca.transform(xTest)

        return xTrain, xTest, yTrain, yTest

    def plot_confusion_matrix(self, cm, class_names):
        plt.figure(figsize=(8, 6))
        sns.heatmap(cm, annot=True, fmt="d", cmap="Blues", xticklabels=class_names, yticklabels=class_names)
        plt.title('Confusion Matrix')
        plt.xlabel('Predicted Labels')
        plt.ylabel('True Labels')
        plt.show()
```

plt_histogram(self):

- Bu yöntem, üç farklı histogram grafiği çizer.
- İlk olarak, `sns.countplot()` kullanılarak "Class" sütununa göre "satisfaction" sütununun dağılımını gösteren bir grafik çizilir. Bu, her bir "Class" kategorisinin "satisfaction" sütununa göre dağılımını gösterir.
- Ardından, `sns.catplot()` kullanılarak "Gender" ve "satisfaction" sütunlarına göre ayrılmış tatmin sayısı grafiği çizilir. Bu, cinsiyetin tatmin durumu üzerindeki etkisini göstermek için çizilir.
- Son olarak, yaşa (Age) göre tatmin durumu (satisfaction) histogramı çizilir. Bu, yaşın tatmin durumu üzerindeki etkisini görselleştirmek için kullanılır.`plt.show()` çağrılarıyla her bir grafik ekrana çizilir.

prepare_data_for_models(self, apply_pca=False, n_components=None):

- Bu yöntem, makine öğrenimi modelleri için veri setini hazırlar.
- Eğitim ve test veri setlerinden hedef sütun olan "satisfaction" sütunu çıkarılır ve giriş (özellikler) ve çıktı (etiketler) verileri ayrılır.
- Veriler ölçeklenir (ölçeklendirme işlemi: StandardScaler kullanılarak gerçekleştirilir).
- Eğer `apply_pca` parametresi `True` olarak belirlenmişse, PCA (Principal Component Analysis) kullanılarak boyut indirgeme işlemi gerçekleştirilir.
- Hazırlanan eğitim ve test veri setleri, model eğitimi için kullanılmak üzere döndürülür.

plot_confusion_matrix(self, cm, class_names):

Bu yöntem, bir karışıklık matrisi (confusion matrix) çizer. Karışıklık matrisi, modelin gerçek ve tahmin edilen sınıflar arasındaki ilişkiyi gösterir. `sns.heatmap()` kullanılarak karışıklık matrisi görselleştirilir. Grafik, doğru ve tahmin edilen sınıflar arasındaki ilişkiyi gösterir.

Makine Öğrenimi Modellerini Eğitmek

```
    def evaluate_model(self, model, xTrain, yTrain, xTest, yTest, class_names):
        model.fit(xTrain, yTrain)
        y_pred = model.predict(xTest)
        cm = confusion_matrix(yTest, y_pred)
        self.plot_confusion_matrix(cm, class_names)

        # Accuracy
        acc = model.score(xTest, yTest)
        print(f"Model Accuracy: {acc}")

        # Precision, Recall, F1-Score
        report = classification_report(yTest, y_pred, target_names=class_names)
        print("Classification Report:\n", report)

        precision = precision_score(yTest, y_pred, average='weighted')
        recall = recall_score(yTest, y_pred, average='weighted')
        f1 = f1_score(yTest, y_pred, average='weighted')

        print(f'Weighted Precision: {precision}')
        print(f'Weighted Recall: {recall}')
        print(f'Weighted F1-Score: {f1}')

    def _apply_model_with_timing(self, model, xTrain, yTrain, xTest, yTest, class_names=None):
        start_time = time.time()  # Başlangıç zamanını kaydet
        self.evaluate_model(model, xTrain, yTrain, xTest, yTest, class_names=class_names)
        end_time = time.time()  # Bitiş zamanını kaydet
        elapsed_time = end_time - start_time  # Geçen süreyi hesapla
        print(f"Modelin çalışma süresi: {elapsed_time:.2f} saniye")

    def apply_random_forest(self, xTrain, yTrain, xTest, yTest):
        model = RandomForestClassifier()
        self._apply_model_with_timing(model, xTrain, yTrain, xTest, yTest,
                                      class_names=['neutral or dissatisfied', 'satisfied'])

    def apply_logistic_regression(self, xTrain, yTrain, xTest, yTest):
        model = LogisticRegression()
        self._apply_model_with_timing(model, xTrain, yTrain, xTest, yTest,
                                      class_names=['neutral or dissatisfied', 'satisfied'])
```

evaluate_model(self, model, xTrain, yTrain, xTest, yTest, class_names):

- Bu yöntem, bir makine öğrenimi modelinin performansını değerlendirir. Model eğitimi için eğitim veri seti (`xTrain` ve `yTrain`) kullanılır.
- Modelin test veri seti üzerindeki performansı ölçülür (`xTest` ve `yTest`). Karışıklık matrisi (confusion matrix) çizilir ve ekrana basılır.
- Doğruluk (accuracy), hassasiyet (precision), geri çağırma (recall) ve F1 puanı (F1-score) gibi performans metrikleri hesaplanır ve ekrana basılır. `class_names` parametresi, sınıf etiketlerini içeren bir liste veya dizi olarak verilir.

_apply_model_with_timing(self, model, xTrain, yTrain, xTest, yTest, class_names=None):

Bu yöntem, bir makine öğrenimi modelini belirli veri setleri üzerinde uygular ve çalışma süresini ölçer. `evaluate_model` yöntemini çağırarak modelin performansını değerlendirir. Başlangıç ve bitiş zamanlarını kaydederek, modelin çalışma süresini hesaplar ve ekrana basar.

Apply_random_forest(self, xTrain, yTrain, xTest, yTest) ve apply_logistic_regression(self, xTrain, yTrain, xTest, yTest):

Bu yöntemler, belirli makine öğrenimi modellerini kullanarak veri seti üzerinde çalıştırır. RandomForestClassifier() veya `LogisticRegression()` gibi belirli bir model örneği oluşturulur.

`_apply_model_with_timing` yöntemi çağrılarak modelin performansı değerlendirilir ve çalışma süresi ölçülür. `class_names` parametresi, sınıf etiketlerini içeren bir liste veya dizi olarak verilir. Bu, `evaluate_model` yöntemine aktarılır.

Makine Öğrenimi Modellerini Eğitmek

```
def apply_support_vector_classifier(self, xTrain, yTrain, xTest, yTest, kernel='linear'):
    model = SVC(kernel=kernel)
    self._apply_model_with_timing(model, xTrain, yTrain, xTest, yTest,
                                  class_names=['neutral or dissatisfied', 'satisfied'])

def apply_decision_tree(self, xTrain, yTrain, xTest, yTest):
    model = DecisionTreeClassifier()
    self._apply_model_with_timing(model, xTrain, yTrain, xTest, yTest,
                                  class_names=['neutral or dissatisfied', 'satisfied'])

def apply_knn(self, xTrain, yTrain, xTest, yTest, n_neighbors=5):
    model = KNeighborsClassifier(n_neighbors=n_neighbors)
    self._apply_model_with_timing(model, xTrain, yTrain, xTest, yTest,
                                  class_names=['neutral or dissatisfied', 'satisfied'])

def apply_gaussianNB(self, xTrain, yTrain, xTest, yTest):
    model = GaussianNB()
    self._apply_model_with_timing(model, xTrain, yTrain, xTest, yTest,
                                  class_names=['neutral or dissatisfied', 'satisfied'])

def apply_gradiendtBoostingClassifier(self, xTrain, yTrain, xTest, yTest):
    model = GradientBoostingClassifier()
    self._apply_model_with_timing(model, xTrain, yTrain, xTest, yTest,
                                  class_names=['neutral or dissatisfied', 'satisfied'])

def apply_neural_network(self, xTrain, yTrain, xTest, yTest, hidden_layer_sizes=(50, 30), max_iter=500,
                         random_state=42):
    model = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes, max_iter=max_iter, random_state=random_state)
    self._apply_model_with_timing(model, xTrain, yTrain, xTest, yTest,
                                  class_names=['neutral or dissatisfied', 'satisfied'])

def apply_xgb(self, xTrain, yTrain, xTest, yTest):
    model = XGBClassifier()
    self._apply_model_with_timing(model, xTrain, yTrain, xTest, yTest,
                                  class_names=['neutral or dissatisfied', 'satisfied'])
```

1. apply_support_vector_classifier(self, xTrain, yTrain, xTest, yTest, kernel='linear'):

Bu yöntem, Destek Vektör Sınıflandırıcısı (Support Vector Classifier, SVC) kullanarak bir model eğitir. `kernel` parametresi, SVC'nin çekirdek tipini belirler. Varsayılan olarak, 'linear' olarak ayarlanmıştır.

2. apply_decision_tree(self, xTrain, yTrain, xTest, yTest):

Bu yöntem, Karar Ağaçları (Decision Trees) kullanarak bir model eğitir. Karar ağaçları, veri setini özelliklerine göre bölerek kararlar veren bir modeldir.

3. apply_knn(self, xTrain, yTrain, xTest, yTest, n_neighbors=5):

Bu yöntem, K En Yakın Komşular (K-Nearest Neighbors, KNN) kullanarak bir model eğitir. `n_neighbors` parametresi, komşu sayısını belirler.

4. apply_gaussianNB(self, xTrain, yTrain, xTest, yTest):

Bu yöntem, Gauss Naive Bayes (GaussianNB) kullanarak bir model eğitir. Naive Bayes yöntemleri, sınıflandırma problemleri için olasılık temelli bir yaklaşım kullanır.

5. apply_gradientBoostingClassifier(self, xTrain, yTrain, xTest, yTest):

Bu yöntem, Gradyan Arttırma Sınıflandırıcısı (Gradient Boosting Classifier) kullanarak bir model eğitir. Gradyan arttırma, zayıf öğrenicileri birleştirerek güçlü bir öğrenici oluşturan bir ansambıl yöntemidir.

6. apply_neural_network(self, xTrain, yTrain, xTest, yTest, hidden_layer_sizes=(50, 30), max_iter=500, random_state=42):

Bu yöntem, Yapay Sinir Ağları (Neural Networks) kullanarak bir model eğitir. `hidden_layer_sizes` parametresi, gizli katmanların boyutunu belirler. `max_iter` parametresi, eğitim döngülerinin maksimum sayısını belirler. `random_state` parametresi, modelin tekrarlanabilirliğini sağlar. Parametreler, gridsearch ile bulundu.

7. apply_xgb(self, xTrain, yTrain, xTest, yTest):

Bu yöntem, XGBoost (eXtreme Gradient Boosting) kullanarak bir model eğitir. XGBoost, diğer gradyan arttırma algoritmalarına kıyasla daha yüksek hız ve performans sunan bir gradyan arttırma kütüphanesidir.

Genel Kullanım Örneği

```
# Kullanım örneği
train_path = 'C:/Users/ABREBO/Desktop/csv/train.csv/train.csv'
test_path = 'C:/Users/ABREBO/Desktop/csv/test.csv/test.csv'

analysis = PassengerSatisfactionAnalysis(train_path, test_path)
analysis.fill_na_mean()
analysis.drop_columns(['Unnamed: 0', 'id', 'Departure Delay in Minutes', 'Cleanliness'])
#analysis.data_info()
analysis.label_encode_categorical_columns()
#analysis.plot_correlation_matrix()
#nalysis.find_duplicate_rows()
#analysis.plot_violin('Age', 'Seat comfort', 'satisfaction', 'Distribution of satisfaction over Age','Age', 'Seat comfort')
#analysis.plot_histograms_subplots(['Age', 'Class', 'Seat comfort'], title='Distribution of Features',title_size=25)
#analysis.plt_histogram()

xTrain, xTest, yTrain, yTest = analysis.prepare_data_for_models()
#analysis.apply_random_forest(xTrain, yTrain, xTest, yTest)
#analysis.apply_logistic_regression(xTrain, yTrain, xTest, yTest)

#analysis.apply_support_vector_classifier(xTrain, yTrain, xTest, yTest,kernel='linear')

#analysis.apply_decision_tree(xTrain, yTrain, xTest, yTest)
#analysis.apply_knn(xTrain, yTrain, xTest, yTest,n_neighbors=7)
#analysis.apply_gaussianNB(xTrain, yTrain, xTest, yTest)
#analysis.apply_gradiendtBoostingClassifier(xTrain, yTrain, xTest, yTest)
#analysis.apply_neural_network(xTrain, yTrain, xTest, yTest)
analysis.apply_xgb(xTrain, yTrain, xTest, yTest)
```

1. Veri Yüklemesi ve Hazırlığı:

- İlk olarak, eğitim ve test veri setlerinin dosya yolları `train_path` ve `test_path` olarak belirlenir.
- `PassengerSatisfactionAnalysis` sınıfı oluşturulur ve veri setleri bu sınıfa yüklenir.
- `fill_na_mean()` yöntemi ile eksik değerler ortalama değerlerle doldurulur.
- `drop_columns()` yöntemi ile belirli sütunlar (Unnamed: 0, id, Departure Delay in Minutes, Cleanliness) düşürülür.
- Kategorik sütunlar için etiket kodlama yapmak üzere `label_encode_categorical_columns()` yöntemi kullanılır.
- Son olarak, veri seti modelleme için hazırlanır: giriş ve çıkış (X ve Y) verileri ayrılır, özellikler ölçeklenir ve gerekiyorsa boyut indirgeme (PCA) yapılır.

2. Model Eğitimi ve Değerlendirmesi:

- `apply_support_vector_classifier()` yöntemi kullanılarak Destek Vektör Sınıflandırıcısı (SVC) modeli eğitilir ve değerlendirilir.
- SVC modeli, doğrusal çekirdek kullanılarak eğitilir.
- Sonuçlar, modelin başarı oranı (doğruluk) ve diğer performans metrikleri (hassasiyet, geri çağırma, F1 puanı) ile birlikte karışıklık matrisi olarak görüntülenir.
- Ayrıca, modelin çalışma süresi de ölçülür ve ekrana basılır.

3. Diğer Modelleme Yaklaşımları:

- Kodun yorum satırları arasında farklı modelleme yaklaşımları mevcuttur. Bunlar arasında Rastgele Orman, Lojistik Regresyon, Karar Ağaçları, K En Yakın Komşular, Gauss Naive Bayes, Gradyan Arttırma Sınıflandırıcısı, Yapay Sinir Ağları ve XGBoost yer almaktadır.
- Her biri, aynı modelleme ve değerlendirme prosedürlerini izleyerek farklı makine öğrenimi algoritmalarını kullanarak model eğitir ve değerlendirir.

SONUÇLAR

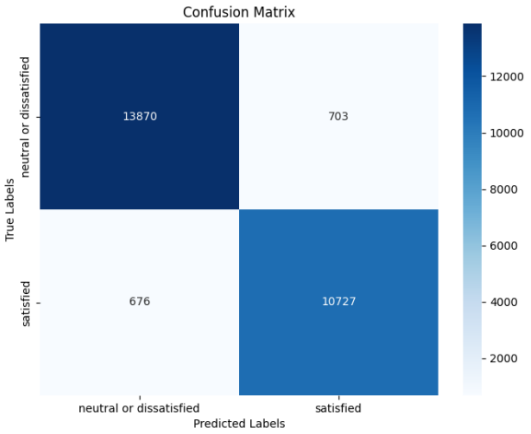
Decision Trees

```
Model Accuracy: 0.9469125346473668
Classification Report:
                         precision    recall  f1-score   support

neutral or dissatisfied       0.95      0.95      0.95     14573
              satisfied       0.94      0.94      0.94     11403

               accuracy                           0.95     25976
              macro avg       0.95      0.95      0.95     25976
           weighted avg       0.95      0.95      0.95     25976

Weighted Precision: 0.9469281587609883
Weighted Recall: 0.9469125346473668
Weighted F1-Score: 0.9469193104493036
Modelin çalışma süresi: 1.23 saniye
```



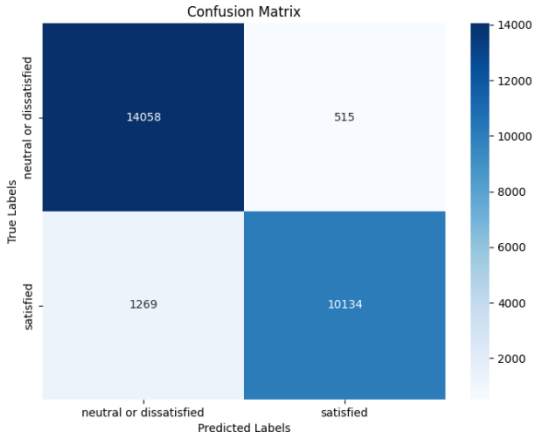
KNN

```
Model Accuracy: 0.9313212195873114
Classification Report:
                         precision    recall  f1-score   support

neutral or dissatisfied       0.92      0.96      0.94     14573
              satisfied       0.95      0.89      0.92     11403

               accuracy                           0.93     25976
              macro avg       0.93      0.93      0.93     25976
           weighted avg       0.93      0.93      0.93     25976

Weighted Precision: 0.9323207200554177
Weighted Recall: 0.9313212195873114
Weighted F1-Score: 0.9310130400868483
Modelin çalışma süresi: 27.19 saniye
```



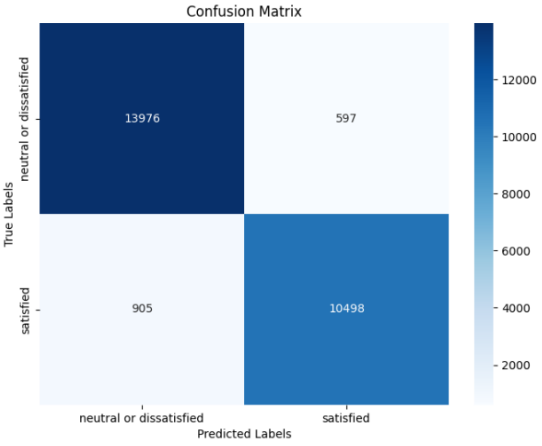
Gradient Boosting

```
Model Accuracy: 0.9421773945180166
Classification Report:
                         precision    recall  f1-score   support

neutral or dissatisfied       0.94      0.96      0.95     14573
              satisfied       0.95      0.92      0.93     11403

               accuracy                           0.94     25976
              macro avg       0.94      0.94      0.94     25976
           weighted avg       0.94      0.94      0.94     25976

Weighted Precision: 0.9422604865028289
Weighted Recall: 0.9421773945180166
Weighted F1-Score: 0.9420839207486169
Modelin çalışma süresi: 23.40 saniye
```



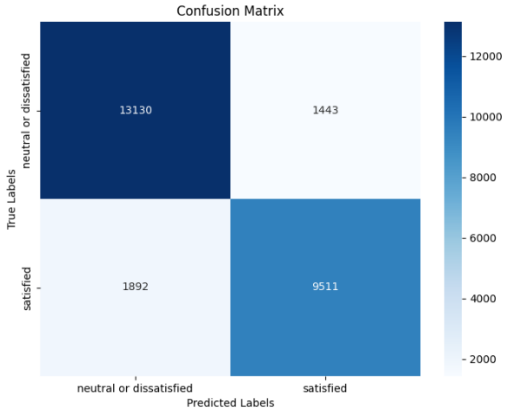
LogisticRegression

```
Model Accuracy: 0.8716122574684324
Classification Report:
                          precision    recall  f1-score   support

neutral or dissatisfied       0.87      0.90      0.89     14573
              satisfied       0.87      0.83      0.85     11403

               accuracy                           0.87     25976
              macro avg       0.87      0.87      0.87     25976
           weighted avg       0.87      0.87      0.87     25976

Weighted Precision: 0.8715122783660837
Weighted Recall: 0.8716122574684324
Weighted F1-Score: 0.8712969555482334
Modelin çalışma süresi: 0.83 saniye
```



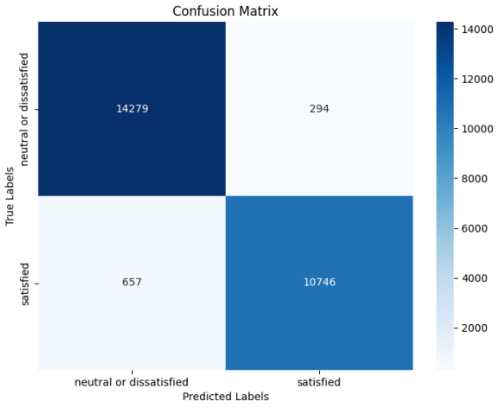
Random Forest

```
Model Accuracy: 0.9633892824145365
Classification Report:
                          precision    recall  f1-score   support

neutral or dissatisfied       0.96      0.98      0.97     14573
              satisfied       0.97      0.94      0.96     11403

               accuracy                           0.96     25976
              macro avg       0.96      0.96      0.96     25976
           weighted avg       0.96      0.96      0.96     25976

Weighted Precision: 0.9636318401714618
Weighted Recall: 0.9633892824145365
Weighted F1-Score: 0.9633183861910146
Modelin çalışma süresi: 22.90 saniye
```



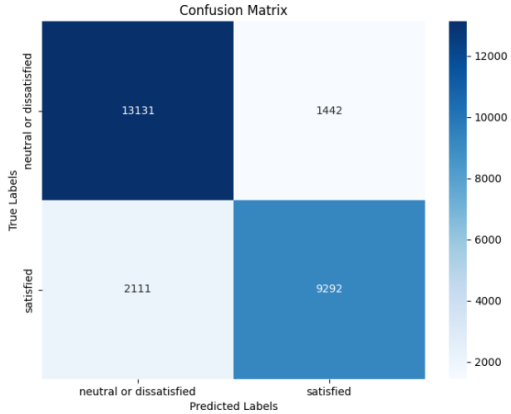
Gaussian Naive Bayes

```
Model Accuracy: 0.8632198952879581
Classification Report:
                          precision    recall  f1-score   support

neutral or dissatisfied       0.86      0.90      0.88     14573
              satisfied       0.87      0.81      0.84     11403

               accuracy                           0.86     25976
              macro avg       0.86      0.86      0.86     25976
           weighted avg       0.86      0.86      0.86     25976

Weighted Precision: 0.8633270188006955
Weighted Recall: 0.8632198952879581
Weighted F1-Score: 0.8626876478457994
Modelin çalışma süresi: 0.70 saniye
```



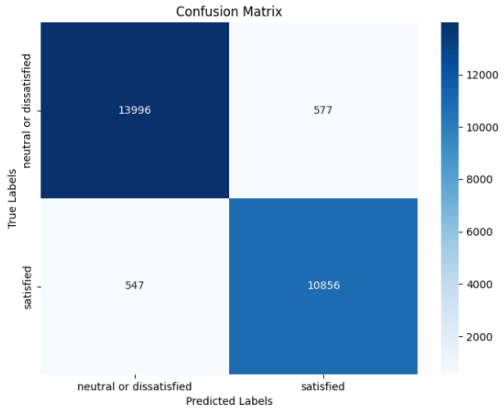
Neural Network

```
Model Accuracy: 0.9567292885740684
Classification Report:
                          precision    recall  f1-score   support

neutral or dissatisfied       0.96      0.96      0.96     14573
              satisfied       0.95      0.95      0.95     11403

               accuracy                           0.96     25976
              macro avg       0.96      0.96      0.96     25976
           weighted avg       0.96      0.96      0.96     25976

Weighted Precision: 0.9567441353709726
Weighted Recall: 0.9567292885740684
Weighted F1-Score: 0.9567354190359072
Modelin çalışma süresi: 112.65 saniye
```



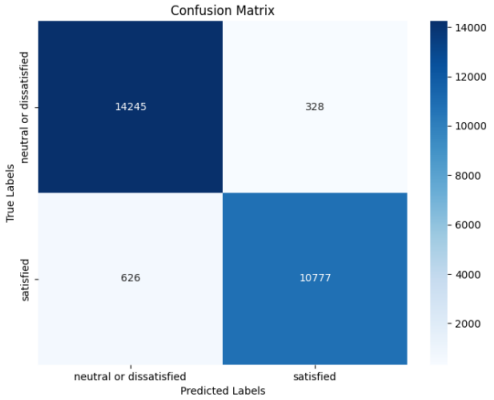
XGB

```
Model Accuracy: 0.9632737911918694
Classification Report:
                          precision    recall  f1-score   support

neutral or dissatisfied       0.96      0.98      0.97     14573
              satisfied       0.97      0.95      0.96     11403

               accuracy                           0.96     25976
              macro avg       0.96      0.96      0.96     25976
           weighted avg       0.96      0.96      0.96     25976

Weighted Precision: 0.963417870892917
Weighted Recall: 0.9632737911918694
Weighted F1-Score: 0.9632165198001973
Modelin çalışma süresi: 2.38 saniye
```



MODEL	F1 SKOR	HIZ
Random Forest	0,96	22,90
XGB	0,96	2,38
Neural Network	0,96	112,65
Decision Trees	0,95	1,23
Gradient Boosting	0,94	23,40
Knn	0,93	27,19
LogisticRegression	0,87	0,83
Gaussian Naive Bayes	0,86	0,70
Ortalama	**0,93**	**23,91**

Skor olarak f1 skor seçilmiştir. Çünkü genel olarak, modelinizin kesinliğinin ve sağlamlığının bir ölçüsüdür. Bir modelin hassasiyet (precision) ve duyarlılık (recall) ölçümlerini birleştirerek, dengeli bir performans değeri sağlar.

Bu sonuçlara göre hem hız hemde skor açısından bakıldığında, Decision Trees algoritması en performanslısı seçilmiştir.