

دانشگاه صنعتی امیرکبیر
(پلی تکنیک تهران)

گزارش تکلیف سوم درس یادگیری ماشین کاربردی

استاد درس: دکتر ناظرفرد

تهیه کننده: سید نیما محمودیان

شماره دانشجویی: ۴۰۲۱۲۵۰۰۵

اللهُ أَكْبَرُ
لِلّهِ الْحَمْدُ
لِلّهِ الْحَمْدُ
لِلّهِ الْحَمْدُ

أ

فهرست مطالب

۱	-پاسخ سوال اول.....
۱	۱-لود کردن دیتاست، نمایش ده ردیف به صورت تصادفی، و بررسی کلی دادهها
۶	۱-۲-تصویر سازی دادهها.....
۶	۱-۲-۱-رسم هیستوگرام.....
۷	۱-۲-۲-رسم و تفسیر نمودارهای جعبهای.....
۸	۱-۲-۳-رسم و تفسیر ماتریس همبستگی.....
۹	۱-۳-تشخیص دادههای پرت.....
۱۰	۱-۴-پر کردن مقادیر خالی.....
۱۰	۱-۴-۱-تعریف شی پیشپردازش برای پر کردن مقادیر خالی با Iterative Imputer
۱۳	۱-۴-۲-تعریف شی پیشپردازش برای پر کردن مقادیر خالی با KNN
۱۳	۱-۵-انتخاب ویژگی
۱۴	۱-۶-آموزش مدل درخت تصمیم.....
۱۴	۱-۶-۱-آموزش مدل با پارامترهای پیشفرض
۱۵	۱-۶-۲-پیدا کردن هایپرپارامترهای بهینه با استفاده از GridSearch
۱۷	۱-۶-۳-هرس کردن درخت و نمایش مجدد آن (بخش امتیازی).....
۱۹	۲-پاسخ سوال دوم
۱۹	۲-۱-آماده سازی دادهها
۲۱	۲-۲-حذف دادههای پرت
۲۲	۲-۳-انتخاب ویژگی
۲۳	۲-۴-آموزش و تست رگرسیون لجستیک.....
۲۵	۳-پاسخ سوال سوم
۲۷	۳-۱-پیش پردازش دادهها
۲۷	۳-۱-۱-حذف تگهای </ br />
۲۷	۳-۱-۲-ساختتابع پیشپردازش
۲۸	۳-۱-۳-استفاده از CountVectorizer
۳۱	۳-۲-استفاده از TF-IDF
۳۱	۳-۳-ساختتابعی که به نظرات جدید برچسب می دهد.....

۳۳.....	۴-پاسخ سوال چهارم.....
۳۳.....	۴-وارد کردن و پیش‌پردازش داده‌ها
۳۵.....	۴-۲-آموزش رگرسیون لجستیک چند کلاسه

فهرست شکل‌ها

۱	شکل ۱: وارد کردن کتابخانه‌های مورد نیاز.....
۱	شکل ۲: خواندن دیتابست
۲	شکل ۳: حذف ستون load_id
۲	شکل ۴: کد مربوط به نمایش ده سطر تصادفی.....
۲	شکل ۵: ده سطر تصادفی نمایش داده شده
۲	شکل ۶: کد مربوط به نشان دادن تعداد مقادیر گمشده
۳	شکل ۷: تعداد مقادیر خالی در هر سطر
۳	شکل ۸: کد بررسی تعداد سطرهای تکراری
۳	شکل ۹: کد بررسی تعداد صفرها در هر ستون
۴	شکل ۱۰: تعداد مقادیر صفر در هر سطر.....
۴	شکل ۱۱: بررسی تعداد و جمعیت موجود در سطرهای گسسته
۴	شکل ۱۲: خروجی کد شکل ۱۱
۵	شکل ۱۳: خروجی اتریبیوت df.columns
۵	شکل ۱۴: کد حذف فاصله از ستون‌ها
۵	شکل ۱۵: جایگذاری صفر با np.nan
۶	شکل ۱۶: جداسازی داده‌های عددی و دسته‌ای در دو لیست جدا
۶	شکل ۱۷: کد مربوط به رسم هیستوگرام
۷	شکل ۱۸: هیستوگرام‌های رسم شده
۷	شکل ۱۹: کد رسم نمودار جعبه‌ای
۸	شکل ۲۰: نمودارهای جعبه‌ای
۸	شکل ۲۱: کد رسم ماتریس همبستگی
۹	شکل ۲۲: ماتریس همبستگی رسم شده
۱۰	شکل ۲۳: حذف داده‌های پرت با روش IQR
۱۰	شکل ۲۴: ایجاد شی از کلاس پایپلاین
۱۱	شکل ۲۵: ایجاد شی از کلاس ColumnTransformer
۱۱	شکل ۲۶: پیش‌پردازش داده‌های آموزش و تست
۱۲	شکل ۲۷: Oversample کردن داده‌های آموزشی
۱۲	شکل ۲۸: آموزش و تست مدل رندوم فارست
۱۲	شکل ۲۹: محاسبه دقیقت
۱۳	شکل ۳۰: پایپلاین به روز شده

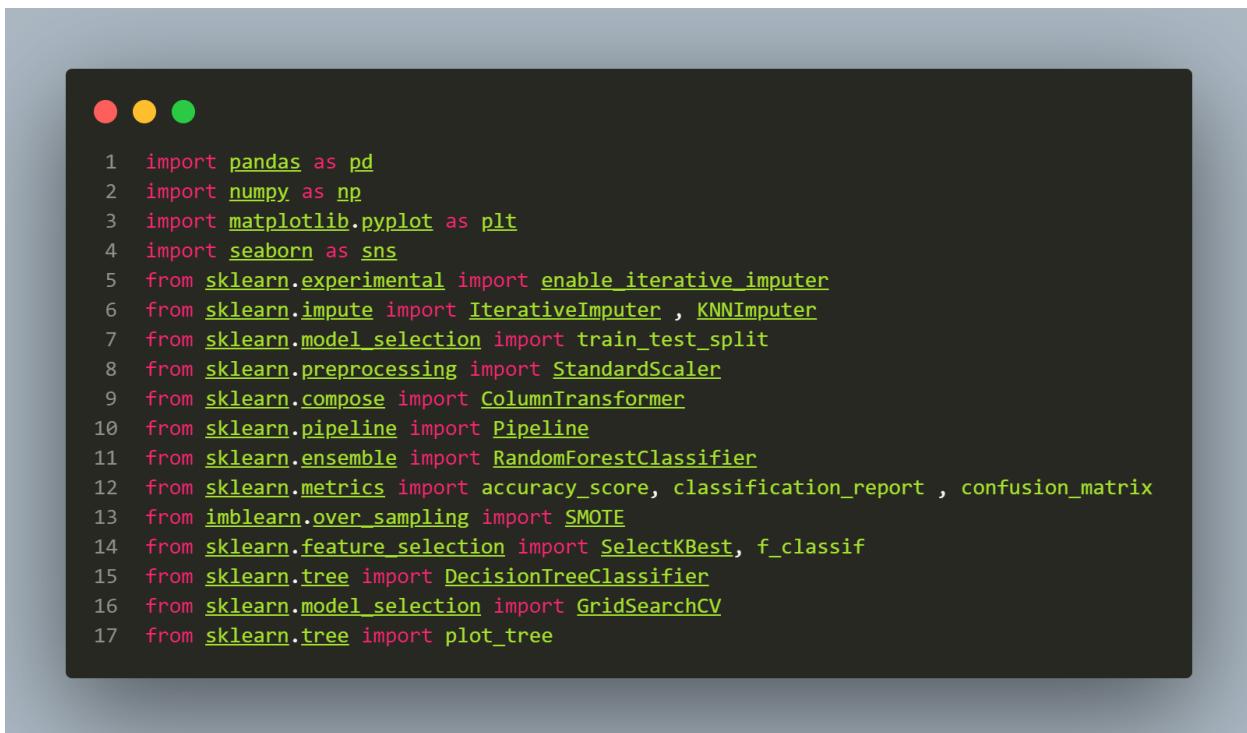
۱۳. شکل ۳۱: پر کردن مقادیر خالی با KNN
۱۴. شکل ۳۲: انتخاب ویژگی ها با استفاده از SelectKBest()
۱۴. شکل ۳۳: پایپلاین آموزش مدل پیشفرض
۱۵. شکل ۳۴: گزارش به دست آمده از آموزش مدل پیشفرض
۱۵. شکل ۳۵: استانداردسازی داده ها
۱۵. شکل ۳۶: دیکشنری هایپرپارامترها
۱۶. شکل ۳۷: پیدا کردن ترکیب بهینه هایپرپارامترها با گرید سرج
۱۶. شکل ۳۸: نمایش پارامترهای بهینه
۱۶. شکل ۳۹: درخت تصمیم با هایپرپارامترهای بهینه
۱۷. شکل ۴۰: گزارش دسته بندی درخت تصمیم با هایپرپارامترهای بهینه
۱۷. شکل ۴۱: کد مربوط به نمایش شکل درخت
۱۸. شکل ۴۲: درخت به دست آمده از هایپرپارامترهای بهینه شده
۱۸. شکل ۴۳: گزارش عملکرد درخت هرس شده
۱۸. شکل ۴۴: درخت هرس شده
۱۹. شکل ۴۵: وارد کردن کتابخانه های مورد نیاز
۱۹. شکل ۴۶: خروجی متدها describe()
۲۰. شکل ۴۷: ده سطر رندوم دیتافریم
۲۰. شکل ۴۸: خروجی تعداد هر دسته در هر ستون
۲۱. شکل ۴۹: پیدا کردن ؟ در دیتاباست
۲۱. شکل ۵۰: انکود کردن ستون های دو دویی
۲۲. شکل ۵۱: one-hot-encode کردن داده ها
۲۲. شکل ۵۲: پیدا کردن داده های پرت با استفاده از خوش بندی
۲۳. شکل ۵۳: پایپلاین ساخته شده برای تحلیل مولفه اصلی
۲۳. شکل ۵۴: گزارش به دست آمده از آموزش و تست مدل با ۱۵ مولفه اصلی
۲۴. شکل ۵۵: انتخاب ۱۵ مولفه اصلی به عنوان ویژگی ها
۲۴. شکل ۵۶: ساخت، آموزش و تست مدل رگرسیون لجستیک
۲۴. شکل ۵۷: گزارش عملکرد رگرسیون لجستیک
۲۵. شکل ۵۸: آموزش مدل با Cross Validation
۲۵. شکل ۵۹: نتیجه Cross Validation
۲۵. شکل ۶۰: وارد کردن کتابخانه های مورد نیاز
۲۶. شکل ۶۱: تعداد ایمیل های با لیبل مثبت و منفی
۲۶. شکل ۶۲: نمایش ده سطر رندوم
۲۶. شکل ۶۳: جایگزینی لیبل ها با صفر و یک
۲۷. شکل ۶۴: حذف تگ های </br>
۲۷. شکل ۶۵: ساخت شی ریشه یاب و مجموعه ایستواره ها
۲۸. شکل ۶۶:تابع پیش پردازش متن

۲۸.....	شکل ۶۷: پیش‌پردازش متن‌ها با استفاده ازتابع تعريف شده
۲۸.....	شکل ۶۸: بردار کردن متون با استفاده از CountVectorizer
۲۹.....	شکل ۶۹: انتخاب ۵۰۰۰ کلمه مهم با استفاده از آزمون مریع کای
۲۹.....	شکل ۷۰: استاندارد سازی و تبدیل ماتریس به ماتریس چگال
۲۹.....	شکل ۷۱: پیش‌پردازش داده‌های تست
۳۰.....	شکل ۷۲: تبدیل ماتریس‌های چگال به آرایه‌های خلوت
۳۰.....	شکل ۷۳: آموزش و تست مدل بیز
۳۰.....	شکل ۷۴: گزارش عملکرد مدل نایو بیز
۳۱.....	شکل ۷۵: گزارش عملکرد مدل به دست آمده از دیتاست TF-IDF
۳۱.....	شکل ۷۶: ذخیره کردن مدل برای استفاده در آینده
۳۲.....	شکل ۷۷: تابع سنجش عواطف نظرات جدید
۳۲.....	شکل ۷۸: کاربرد تابع تحلیل عواطف
۳۲.....	شکل ۷۹: خروجی تابع تحلیل عواطف
۳۳.....	شکل ۸۰: وارد کردن کتابخانه‌های مورد نیاز
۳۳.....	شکل ۸۱: فایل digits.png
۳۴.....	شکل ۸۲: تابع تقسیم کننده عکس بزرگ به عکس‌های کوچکتر
۳۴.....	شکل ۸۳: آماده سازی داده‌ها برای آموزش
۳۵.....	شکل ۸۴: آموزش و تست رگرسیون لجستیک
۳۵.....	شکل ۸۵: محاسبه دقت
۳۶.....	شکل ۸۶: نحوه محاسبه ماتریس درهم‌ریختگی
۳۶.....	شکل ۸۷: ماتریس درهم‌ریختگی به دست آمده

۱-پاسخ سوال اول

۱-۱-لود کردن دیتاست، نمایش ده ردیف به صورت تصادفی، و بررسی کلی دادهها

توجه شود که هرجا احتیاجی به ایپمورت کردن یک کتابخانه بود، برای حفظ نظم، ایمپورت در اولین بلوک کد انجام شده است شکل یک بلوک مربوط به وارد کردن کتابخانه‌ها را نمایش می‌دهد. برای لود کردن دیتاست، و نمایش ده ردیف رندوم از کتابخانه pandas استفاده می‌کنیم. با استفاده از کتابخانه numpy یک random seed ایجاد می‌کنیم. Random seed به مظور بازنولید نتایج مشابه تعریف می‌شود. شکل (۳)، خواندن دیتاست و نمایش آن را نشان می‌دهد.



```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn.experimental import enable_iterative_imputer
6 from sklearn.impute import IterativeImputer, KNNImputer
7 from sklearn.model_selection import train_test_split
8 from sklearn.preprocessing import StandardScaler
9 from sklearn.compose import ColumnTransformer
10 from sklearn.pipeline import Pipeline
11 from sklearn.ensemble import RandomForestClassifier
12 from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
13 from imblearn.over_sampling import SMOTE
14 from sklearn.feature_selection import SelectKBest, f_classif
15 from sklearn.tree import DecisionTreeClassifier
16 from sklearn.model_selection import GridSearchCV
17 from sklearn.tree import plot_tree
```

شکل ۱: وارد کردن کتابخانه‌های مورد نیاز



```
1 df = pd.read_csv("loan_approval_dataset.csv")
2 df
```

شکل ۲: خواندن دیتاست

مشاهده می‌شود که ستون loan_id کاملاً متناظر با اندیسی است که پندار به طور خودکار به دیتافریم داده است. پس این ستون را حذف می‌کنیم. شکل (۳) نحوه حذف این ستون با استفاده از متند drop(). را نمایش می‌دهد.

```

1 df = df.drop('loan_id', axis=1)

```

شکل ۳: حذف ستون load_id

شکل (۴) نشان دادن ده ردیف رندوم را نمایش می‌دهد. این کار را با استفاده از متده sample() انجام می‌شود.

```

1 # Set random seed for reproducibility
2 np.random.seed(42)
3 random_rows = df.sample(n=10)
4 random_rows

```

شکل ۴: کد مربوط به نمایش ده سطر تصادفی

همچنین شکل (۵) ده سطر تصادفی نمایش داده شده را نشان می‌دهد.

no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil_score	residential_assets_value	commercial_assets_value	luxury_assets_value	bank_asset_value	loan_status	
1703	5	Graduate	No	5400000	19700000	20	423	6500000	10000000	15700000	7300000	Rejected
1173	2	Graduate	No	5900000	14000000	8	599	4700000	9500000	17800000	6700000	Approved
308	3	Graduate	No	9600000	19900000	14	452	4200000	16200000	28500000	6600000	Rejected
1322	2	Graduate	No	6200000	23400000	8	605	10000000	10800000	21800000	9200000	Approved
3271	3	Not Graduate	Yes	5800000	14100000	12	738	11700000	4400000	15400000	8400000	Approved
3539	4	Graduate	Yes	4700000	12500000	8	678	13700000	200000	9800000	7000000	Approved
1522	4	Graduate	No	3400000	13500000	12	705	10000000	1100000	8300000	4900000	Approved
3399	5	Not Graduate	Yes	5100000	13700000	14	527	12100000	8900000	19400000	4700000	Rejected
1402	3	Graduate	Yes	3300000	8500000	12	586	7500000	5100000	7800000	3900000	Approved
1829	1	Not Graduate	Yes	3000000	6000000	16	518	3800000	1700000	11600000	3900000	Rejected

شکل ۵: ده سطر تصادفی نمایش داده شده

در ادامه با استفاده از متده info() نگاهی کلی به ستون‌های باقی‌مانده می‌اندازیم. سپس با استفاده از method chaining نمایش داده شده در

شکل (۶) بررسی می‌کنیم که در هر ستون در مجموع چه تعداد مقادیر تعریف نشده وجود دارد.

```

1 df.isnull().sum()

```

شکل ۶: کد مربوط به نشان دادن تعداد مقادیر گمشده

با توجه به شکل (۷) مشاهده می‌شود که هیچ ستونی، خانه‌ی بدون مقدار ندارد.

no_of_dependents	0
education	0
self_employed	0
income_annum	0
loan_amount	0
loan_term	0
cibil_score	0
residential_assets_value	0
commercial_assets_value	0
luxury_assets_value	0
bank_asset_value	0
loan_status	0
dtype:	int64

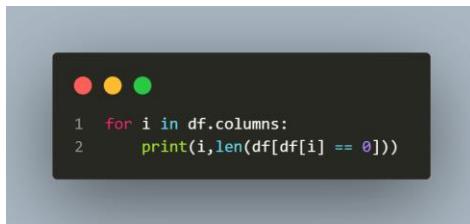
شکل ۷: تعداد مقادیر خالی در هر سطر

در ادامه سه مورد را بررسی می‌کنیم: اول اینکه چه تعداد سطر تکراری داریم. شکل (۸) نحوه انجام این کار را نمایش می‌دهد. سپس بررسی می‌کنیم که آیا در ستون‌هایی که مقدار صفر معنایی ندارد، رکوردی با مقدار صفر داریم یا خیر. شکل (۹) نحوه انجام این کار را نمایش می‌دهد.



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

شکل ۸: کد بررسی تعداد سطرهای تکراری



```
1 for i in df.columns:
2     print(i, len(df[df[i] == 0]))
```

شکل ۹: کد بررسی تعداد صفرها در هر ستون

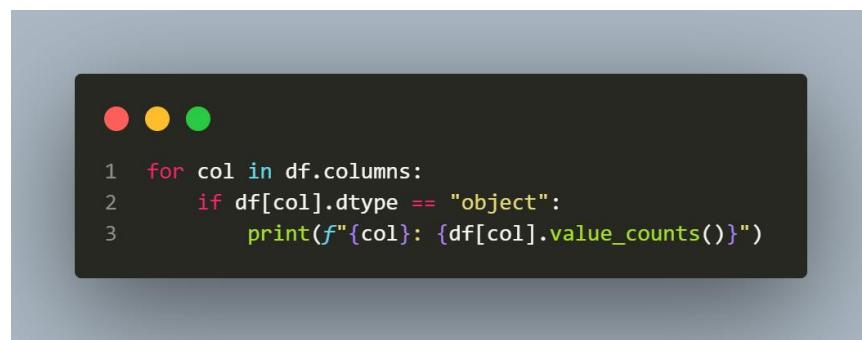
با بررسی خروجی این دو کد، متوجه می‌شویم که هیچ سطر تکراری در دیتافریم وجود ندارد. ولی مشاهده می‌شود که سه تا از ستون‌ها دارای مقادیر صفر هستند. شکل (۱۰) خروجی کد شکل ۹ را نمایش می‌دهد.

مشاهده می‌شود که ستون‌های bank_asset_value و residential_assets_value و no_of_dependents دارای مقادیر صفر هستند. به بررسی این سه ستون می‌پردازیم. به نظر می‌آید ستون no_of_dependents به تعداد افراد تحت پوشش فرد وام گیرنده اشاره دارد. با این اوصاف ممکن است که فرد درخواست کننده کسی را تحت پوشش نداشته باشد. بنابراین داشتن مقدار صفر در این ستون بی‌معنی نیست. دو ستون باقی‌بانده به ارزش دارایی‌های درخواست کننده اشاره دارد. به نظر می‌آید ارزش دارایی صفر چندان معنادار نباشد. بنابراین مقادیر صفر در این دو ستون را، گمشده در نظر می‌گیریم.

```
no_of_dependents 712
education 0
self_employed 0
income_annum 0
loan_amount 0
loan_term 0
cibil_score 0
residential_assets_value 45
commercial_assets_value 107
luxury_assets_value 0
bank_asset_value 8
loan_status 0
```

شکل ۱۰: تعداد مقادیر صفر در هر سطر

سپس بررسی می‌کنیم که در هر یک از ستون‌های دسته‌ای از هر دسته چه تعدادی وجود دارد. ابتدا یک حلقه روی لیست ستون‌های دیتافریم تعریف می‌کنیم که برای هر ستونی که دیتابایپ object دارد، با استفاده از متده استفاده از `value_counts()`. تعداد هر دسته در هر ستون را نمایش می‌دهیم. شکل (۱۱) کدی که این کار را انجام می‌دهد نمایش داده و خروجی در شکل (۱۲) قرار گرفته.



```
1 for col in df.columns:
2     if df[col].dtype == "object":
3         print(f'{col}: {df[col].value_counts()}'")
```

شکل ۱۱: بررسی تعداد و جمعیت موجود در سطرهای گستته

```
education: education
Graduate      2144
Not Graduate  2125
Name: count, dtype: int64
self_employed: self_employed
Yes        2150
No         2119
Name: count, dtype: int64
loan_status: loan_status
Approved    2656
Rejected    1613
Name: count, dtype: int64
```

شکل ۱۲: خروجی کد شکل ۱۱

حال با نمایش ستون‌های دیتا فریم، متوجه می‌شویم که در برخی از ستون‌ها، فاصله خالی وجود دارد. شکل (۱۳) خروجی اتریبیوت columns را نمایش می‌دهد.

```
Index(['no_of_dependents', 'education', 'self_employed', 'income_annum',
       'loan_amount', 'loan_term', 'cibil_score',
       'residential_assets_value', 'commercial_assets_value',
       'luxury_assets_value', 'bank_asset_value', 'loan_status'],
      dtype='object')
```

شکل ۱۳: خروجی اتریبیوت df.columns

سپس با استفاده از متدهrename() ستون‌ها را مجدداً نامگذاری می‌کنیم. در آرگومان columns یک تابع لامبدا پاس می‌دهیم که نام ستون‌ها را به عنوان آرگومان دریافت می‌کند و سپس با متدهstrip() فاصله‌های اضافی را حذف می‌کنیم. شکل (۱۴) این کد را نمایش می‌دهد.

```
1 # remove space in column names using strip() function
2 df.rename(columns=Lambda x: x.strip(), inplace=True)
3 df.columns
```

شکل ۱۴: کد حذف فاصله از ستون‌ها

سپس مقادیر صفر در ستون‌های bank_asset_value و residential_assets_value را با Nan جایگزین می‌کنیم. ابتدا با استفاده از متدهreplace() صفرها را با None جایگزین می‌کنیم و سپس با استفاده از متدهfillna() به عنوان آرگومان value مقادیر None را با np.nan جایگزین می‌کنیم. شکل (۱۵) نحوه انجام این کار را نمایش می‌دهد.

```
1 #replace "0" with Nan
2 df.loc[:, ['residential_assets_value', 'commercial_assets_value']] = df.loc[:, ['residential_assets_value', 'commercial_assets_value']].replace({0: None})
3 # Replace 'None' values with NaN
4 df.fillna(value=np.nan, inplace=True)
```

شکل ۱۵: جایگزینی صفر با np.nan

سپس با استفاده از List comprehension نام ستون‌هایی که داده‌های عددی دارند را در یک لیست و نام ستون‌هایی که دارای مقادیر دسته‌ای هستند را در یک لیست دیگر قرار می‌دهیم. شکل (۱۶) نحوه انجام این کار را نمایش می‌دهد.

```

1 # Select categorical columns with relatively low cardinality (convenient but arbitrary)
2 categorical_cols = [cname for cname in df.columns if df[cname].dtype in ["string", "object"] ]
3 # Select numerical columns
4 numerical_cols = [cname for cname in df.columns if df[cname].dtype in ['int64', 'float64']]

```

شکل ۱۶: جداسازی داده‌های عددی و دسته‌ای در دو لیست جدا

۱-۲-تصویر سازی داده‌ها

۱-۲-۱-رسم هیستوگرام

شکل ۱۷ نحوه رسم هیستوگرام را نمایش می‌دهد. در این مورد، یک شبکه سه در سه ایجاد می‌کند، بنابراین می‌توان تا ۹ نمودار فرعی را در خود جای داد.تابع plt.subplots() برای تولید یک شکل و مجموعه‌ای از نمودارهای فرعی (محور) استفاده می‌شود. (۰،۱۰) ۲۰) figsize کل شکل (۰،۱۰) اینچ عرض و ۱۰ اینچ ارتفاع) را مشخص می‌کند. سپس حلقه روی هر ستون مشخص شده در لیست numerical_cols از df تکرار می‌شود. برای هر ستون: sns.histplot() یک هیستوگرام از ستون ایجاد می‌کند و آن را بر روی نمودار فرعی مربوطه ترسیم می‌کند که هر هیستوگرام باید ۲۰ میله داشته باشد. سپس عنوان هر نمودار بالای آن مشخص می‌شود. سپس لوب بعدی هر فضای خالی که در آن نمودار وجود ندارد را حذف می‌کند. plt.tight_layout()، نمودارهای فرعی و برچسب‌ها را طوری تنظیم می‌کند که به خوبی در ناحیه شکل قرار گیرند و روی هم قرار نگیرند. plt.show() شکل کامل را با تمام نمودارهای فرعی نمایش می‌دهد.

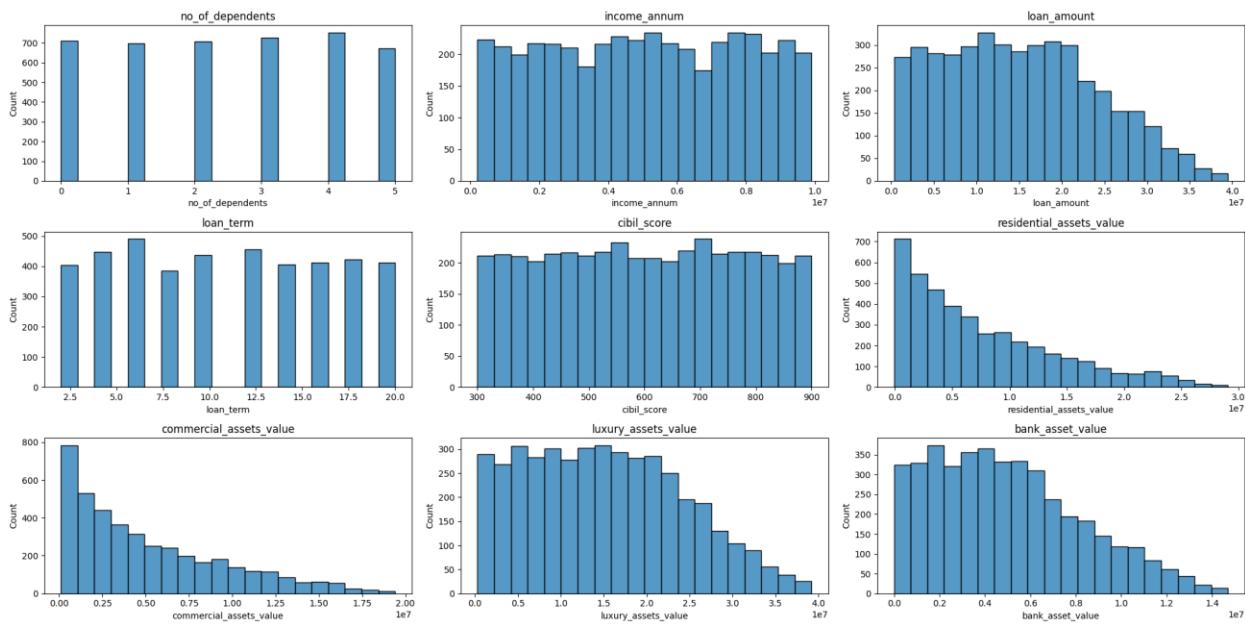
```

1 # Calculate the number of rows and columns needed for subplots
2 num_rows = 3
3 num_cols = 3
4
5 # Create subplots
6 fig, axes = plt.subplots(num_rows, num_cols, figsize=(20, 10))
7
8 # Flatten the axes array for easy iteration
9 axes = axes.flatten()
10
11 # Plot histograms for each column
12 for i, col in enumerate(df.loc[:, numerical_cols]):
13     sns.histplot(df[col], ax=axes[i], bins=20)
14     axes[i].set_title(col) # Set the title to the column name
15
16 # Hide any remaining empty subplots
17 for i in range(len(df.columns), len(axes)):
18     axes[i].axis('off')
19
20 # Adjust layout to prevent overlap
21 plt.tight_layout()
22
23 # Show the plot
24 plt.show()

```

شکل ۱۷: کد مربوط به رسم هیستوگرام

همچنین شکل (۱۸) هیستوگرام‌های رسم شده را نمایش می‌دهد.



شکل ۱۸: هیستوگرام‌های رسم شده

۱-۲-۲-رسم و تفسیر نمودارهای جعبه‌ای

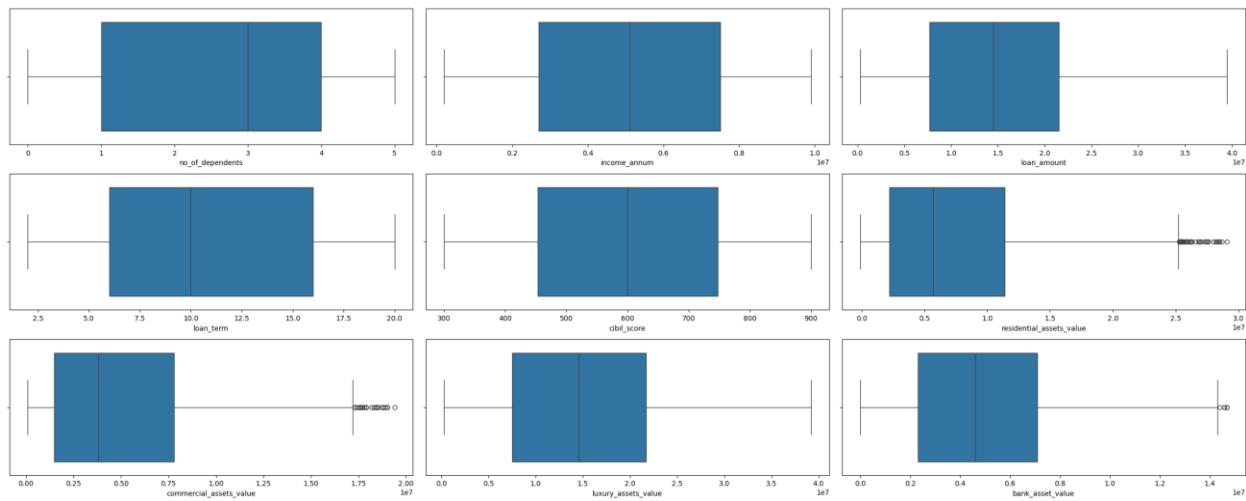
رسم نمودارهای جعبه‌ای برای ستونهای عددی همانند رسم هیستوگرام‌ها است با این تفاوت که از تابع `boxplot()` برای رسم نمودار استفاده می‌شود. شکل (۱۹) کد رسم کننده این نمودارها را نمایش می‌دهد. همچنین شکل (۲۰) خروجی این کد را نمایش می‌دهد.

```

● ● ●
1 # Calculate the number of rows and columns needed for subplots
2 num_rows = 3
3 num_cols = 3 # Number of numerical columns in your DataFrame
4
5 # Create subplots
6 fig, axes = plt.subplots(num_rows, num_cols, figsize=(25, 10))
7
8 # Plot boxplots for each column
9 for i, col in enumerate(numerical_cols):
10     sns.boxplot(x=df[col], ax=axes[i // num_cols, i % num_cols]) # Adjusted indexing
11
12 # Set labels for each subplot
13 for i, col in enumerate(numerical_cols):
14     axes[i // num_cols, i % num_cols].set_xlabel(col) # Adjusted indexing
15
16 # Hide any remaining empty subplots
17 for i in range(len(numerical_cols), num_rows * num_cols):
18     axes[i // num_cols, i % num_cols].axis('off')
19
20 # Show the plot
21 plt.tight_layout()
22 plt.show()

```

شکل ۱۹: کد رسم نمودار جعبه‌ای



شکل ۲۰: نمودارهای جعبه‌ای

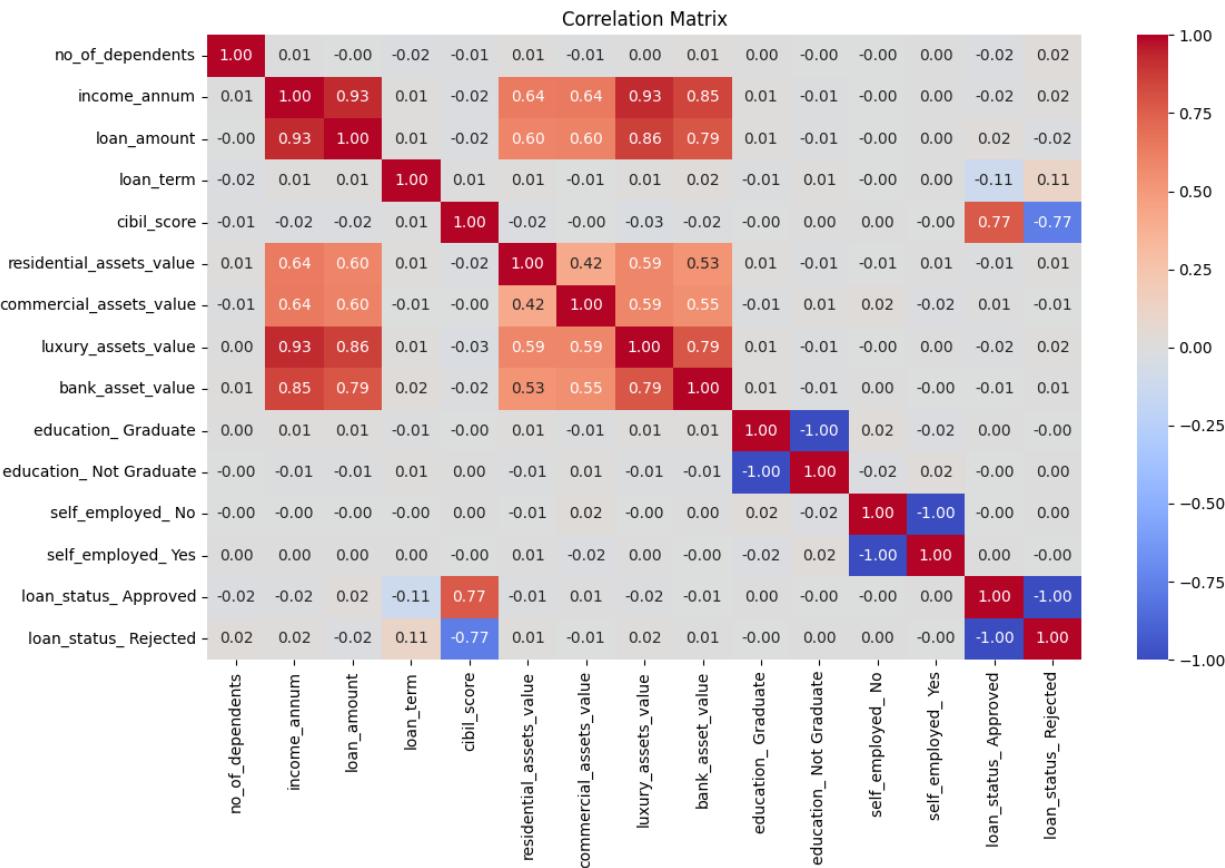
۳-۲-۱ رسم و تفسیر ماتریس همبستگی

با استفاده از متدهای `corr()` ماتریس همبستگی را ایجاد می‌کنیم. سپس با استفاده از کتابخانه `seaborn` و تابع `heatmap()` ماتریس را نمایش دهد.



شکل ۲۱: کد رسم ماتریس همبستگی

همچنین شکل (۲۲) ماتریس رسم شده را نمایش می‌دهد.

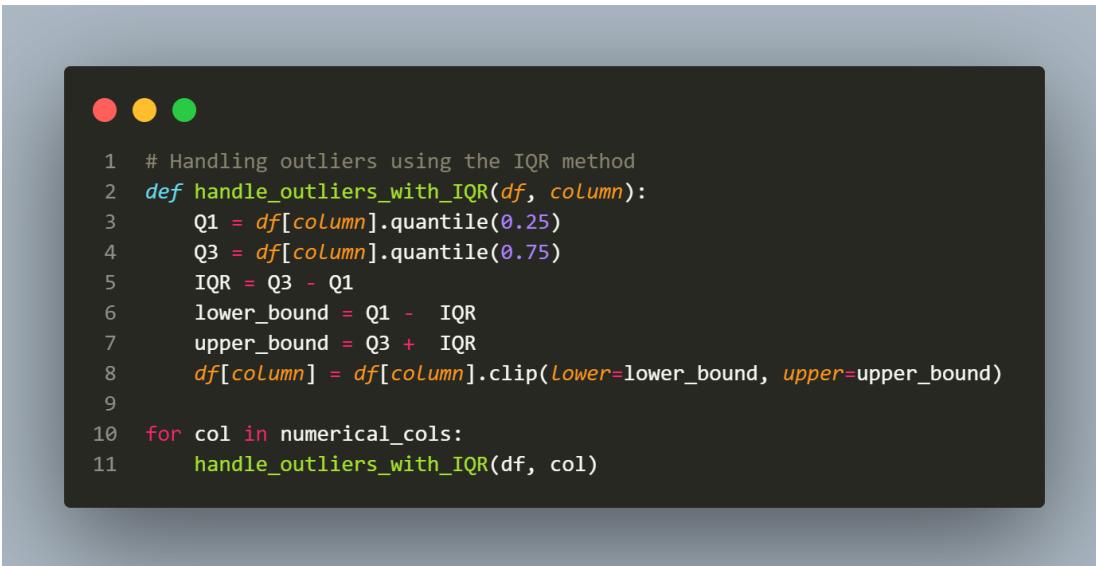


شکل ۲۲: ماتریس همبستگی رسم شده

۱-۳- تشخیص داده‌های پرت

برای تشخیص داده‌های پرت از روش دامنه میان چارکی استفاده می‌کنیم. ابتدا برای هر ستون، مقادیر چارک اول و سوم و دامنه میان چارکی را پیدا می‌کنیم، سپس داده‌های بزرگتر از ۱.۵ برابر دامنه به علاوه چارک سوم و داده‌های کوچکتر از ۱.۵ برابر دامنه منهای چارک اول هستند را حذف می‌کنیم. برای این منظور از تابع clip(). و تعريف دو مقدار توضیح داده شده به عنوان حد بالا و حد پایین، تنها داده‌هایی را نگه می‌داریم که بین دو حد تعريف شده هستند. شکل (۲۳) نحوه انجام این کار را نمایش می‌دهد.

موردي که پيش مي آيد اين است که مشاهده مي شود که هيچ يك از دادهها در هر ستون، خارج از حدود تعبيين شده قرار نمي گيرند. سپس با تنگ‌تر کردن محدوده‌ها به يك برابر دامنه میان چارکی دوباره امتحان می‌کنیم. مشاهده مي شود که مجدداً داده‌اي خارج از اين حدود نيز قرار نمي گيرد. در نتيجه مي توان متوجه شد که داده پرتی در اين دیتاست نداريم.



```

1 # Handling outliers using the IQR method
2 def handle_outliers_with_IQR(df, column):
3     Q1 = df[column].quantile(0.25)
4     Q3 = df[column].quantile(0.75)
5     IQR = Q3 - Q1
6     lower_bound = Q1 - IQR
7     upper_bound = Q3 + IQR
8     df[column] = df[column].clip(lower=lower_bound, upper=upper_bound)
9
10 for col in numerical_cols:
11     handle_outliers_with_IQR(df, col)

```

شکل ۲۳: حذف داده‌های پرت با روش IQR

۱-۴-پر کردن مقادیر خالی

برای پر کردن مقادیر خالی دو روش را با هم مقایسه می‌کنیم. اولین روش پر کردن مقادیر با استفاده از IterativeImputer و دومین روش استفاده از میانه است

۱-۴-۱-تعریف شی پیش‌پردازش برای پر کردن مقادیر خالی با Iterative Imputer

ابتدا داده‌ها را به دو دسته آموزش و تست تقسیم می‌کنیم. در ادامه برای انجام راحت پیش‌پردازش یک شی از کلاس پایپ‌لاین تعریف می‌کنیم. این شی دسته‌ای از عملیات‌ها را به ترتیب روی هدف انجام می‌دهد. شی تعریف شده در این قسمت، ابتدا مقادیر خالی را با استفاده ازتابع IterativeImputer پر می‌کند و سپس داده‌ها را استاندارد سازی می‌کند. شکل (۲۴) کد مربوط به این شی را نمایش می‌دهد.



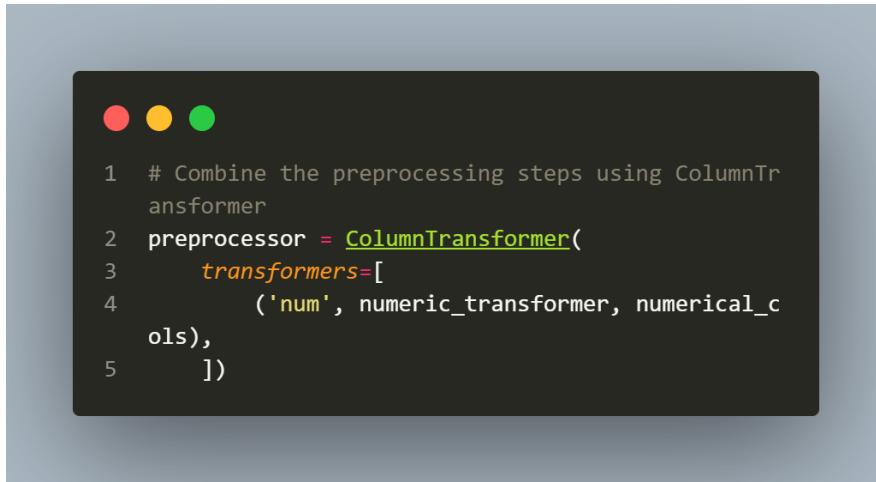
```

1 numeric_transformer = Pipeline(steps=[
2     ('imputer', KNNImputer(n_neighbors=100)),
3     ('scaler', StandardScaler())
4 ])

```

شکل ۲۴: ایجاد شی از کلاس پایپ‌لاین

سپس یک شی از کلاس ColumnTransformer() ایجاد می‌کنیم در یک تاپل به ترتیب نام ترانسفورمر، شی ساخته شده از پایپ‌لاین، و ستون‌هایی که می‌خواهیم تعییرات روی آنها اعمال شوند را قرار می‌دهیم. شکل (۲۵) ترانسفورمر ایجاد شده را نمایش می‌دهد.



```

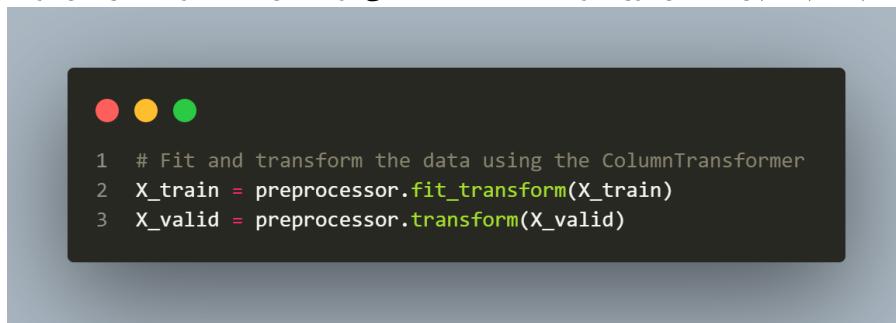
1 # Combine the preprocessing steps using ColumnTransformer
2 preprocessor = ColumnTransformer(
3     transformers=[
4         ('num', numeric_transformer, numerical_columns),
5     ])

```

شکل ۲۵: ایجاد شی از کلاس ColumnTransformer

در ادامه از شی `preprocessor` متدهای `fit_transform()` و `transform()` را فراخوانی می‌کنیم و دیتاست آموزشی را به آن پاس می‌دهیم. این متدهای پارامتر دیتابستی که به عنوان آرگومان دریافت کرده را یاد می‌گیرد، سپس با توجه به پارامترهایی که محاسبه کرده، عملیات‌های تعریف شده را انجام می‌دهد. و سپس نتیجه را در همان متغیر قبلی ذخیره می‌کنیم.

سپس برای پیش‌پردازش داده‌های تست، از متدهای `transform()` استفاده می‌کنیم. دلیل استفاده از این متدهای `transform()` این است که این متدهای استفاده از پارامترهای یادگرفته شده از داده‌های آموزشی، عملیات را روی داده‌های تستی انجام می‌دهد. توجه شود که اینجا از متدهای قبلي استفاده نمی‌شود زیرا فرض بر این است که پارامترهای داده‌های تست را نداریم و باید با استفاده از برآوردهای که در داده‌های آموزشی انجام دادیم، عملیات‌ها را روی داده‌های تست انجام دهیم. عدم رعایت این مورد، موجب ایجاد نشت داده می‌شود. شکل (۲۶) نحوه عملکرد این دو متدهای `transform()` را نمایش می‌دهد.



```

1 # Fit and transform the data using the ColumnTransformer
2 X_train = preprocessor.fit_transform(X_train)
3 X_valid = preprocessor.transform(X_valid)

```

شکل ۲۶: پیش‌پردازش داده‌های آموزش و تست

از آنجایی که دیتاست `imbalance` است، در اولین اقدام باید تعداد لیبل‌های صفر و تعداد لیبل‌های یک را در دیتاست آموزشی برابر کرد. برای این منظور از روش SMOTE استفاده می‌کنیم. این روش یکی از روش‌های oversampling است که از لیبلی که تعداد کمتری دارد داده‌های مصنوعی تولید می‌کند تا تعداد هر دو لیبل با هم برابر شود.

برای این منظور از کلاس SMOTE یک شی به نام `sm` می‌سازیم. سپس با استفاده از متدهای `fit_resample()` داده‌های آموزشی را بالанс می‌کنیم. شکل (۲۷) نحوه انجام این کار را نمایش می‌دهد.



```

1 sm = SMOTE(random_state=2)
2
3 print("\nClass 1 before Over Sampling --> ", sum(y_train == 1))
4 print("\nClass 0 before Over Sampling --> ", sum(y_train == 0))
5
6 X_train, y_train = sm.fit_resample(X_train, y_train)
7
8 print("\nThe shape of X after Over Sampling -->", X_train.shape)
9 print("\nThe shape of Y after Over Sampling -->", y_train.shape)
10
11 print("\nClass 1 after Over Sampling --> ", sum(y_train == 1))
12 print("\nClass 0 after Over Sampling --> ", sum(y_train == 0))
13 print("\n")

```

شکل ۲۷: Oversample کردن داده‌های آموزشی

سپس از کلاس RandomForestClassifier یک شی به نام مدل می‌سازیم. سپس به متده است fit(). از این مدل، دیتاست آموزشی را پاس می‌دهیم و با استفاده از متده predict(). و پاس دادن دیتاست تست، پیش‌بینی‌های مدل را در متغیر y_pred ذخیره می‌کنیم. شکل (۲۸) نحوه انجام این کار را نمایش می‌دهد.



```

1 model = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
2 model.fit(X_train, y_train)
3 # Make predictions on new data
4 y_pred = model.predict(X_valid)

```

شکل ۲۸: آموزش و تست مدل رندوم فارست

در ادامه میزان دقت را با مقایسه پیش‌بینی‌ها و مقدار واقعی لیبل‌ها محاسبه می‌کنیم. برای این منظور از تابع accuracy_score() استفاده می‌کنیم. شکل (۲۹) نحوه محاسبه دقت را نمایش می‌دهد.



```

1 acc = accuracy_score(y_valid, y_pred)
2 print("accuracy Score:", acc)

```

شکل ۲۹: محاسبه دقت

با استفاده از IterativeImputer میزان دقت برابر با ۰.۹۷۴۸ به دست می‌آید.

۱-۴-۲-تعریف شی پیشبردازش برای پر کردن مقادیر خالی با KNN

حال شی تعریف شده برای پیشبردازش را به نحوی تغییر می‌دهیم که مقادیر خالی را با KNN پر کنند. برای این منظور، در شی پایپلاین، از کلاس () KNNImputer به جای کلاس IterativeImputer() با آرگومان $n_neighbors=60$ را پاس می‌دهیم. شکل (۳۰) پایپلاین به روز شده را نمایش می‌دهد.

```
1 numeric_transformer = Pipeline(steps=[  
2     ('imputer', KNNImputer(n_neighbors=60)),  
3     ('scaler', StandardScaler())  
4 ])
```

شکل ۳۰: پایپلاین به روز شده

سایر مراحل را مشابه بخش قبلی انجام می‌دهیم. و مدل را آموزش و تمرین می‌دهیم و مقدار دقت را محاسبه می‌کنیم. در این حالت مقدار دقت برابر ۹۷۵۴٪ خواهد شد از آنجایی که مقدار دقت در این روش بیشتر است، از KNN برای پر کردن مقادیر خالی استفاده می‌کنیم.

برای این منظور یک شی از کلاس KNNImputer درست می‌کنیم و خروجی متدها fit_transform() را به همراه نام ستون‌های دیتافریم را به تابع() dataframe می‌دهیم و دیتافریم خروجی را در متغیر df ذخیره می‌کنیم. شکل (۳۱) نحوه انجام این کار را نمایش می‌دهد.

```
1 imputer = KNNImputer(n_neighbors=60)  
2 columns = df.columns  
3 # Fit the imputer on the column with missing values and transform the DataFrame  
4 df = pd.DataFrame(imputer.fit_transform(df), columns=columns)
```

شکل ۳۱: پر کردن مقادیر خالی با KNN

در ادامه به انتخاب ویژگی می‌پردازیم.

۱-۵-انتخاب ویژگی

برای انتخاب ویژگی‌های مناسب، از روش KBest استفاده می‌کنیم. این روش یک روش انتخاب ویژگی تک متغیره است که با استفاده از یک تست آماری، بهترین ویژگی‌ها را انتخاب می‌کند. برای این منظور از کلاس SelectKBest یک شی می‌سازیم تعداد ویژگی‌هایی که قرار است انتخاب شوند را برابر ۹ قرار می‌دهیم همچنین تست آماری را f_classif() قرار می‌دهیم. سپس در متدها fit_transform() آن، متغیرها و سطر هدف را پاس می‌دهیم و نتایج به دست آمده را در متغیر X_best_features ذخیره می‌کنیم. سپس با استفاده از متدها get_support() و نام ستون‌های انتخاب شده را در متغیری ذخیره می‌کنیم و با استفاده از آن، ستون‌های مربوطه از دیتافریم را انتخاب می‌کنیم. شکل (۳۲) نحوه انجام این کار را نمایش می‌دهد.

```

1 # Separate features (X) and target variable (y)
2 X = df.drop(columns=['loan_status'])
3 y = df['loan_status']
4
5 k_best_selector = SelectKBest(score_func=f_classif, k=9) # Select top 9 features
6 X_best_features = k_best_selector.fit_transform(X, y)
7
8 # Get the indices of the selected features
9 selected_feature_indices = k_best_selector.get_support(indices=True)
10
11 # Get the names of the selected features
12 df = df.loc[:, np.append(X.columns[selected_feature_indices], 'loan_status')]

```

شکل ۳۲: انتخاب ویژگی‌ها با استفاده از SelectKBest()

۶-۱-آموزش مدل درخت تصمیم

در این بخش به آموزش مدل درخت تصمیم می‌پردازیم. ابتدا دیتاست را به نسبت ۸۰ به ۲۰ به دیتای آموزش و تست تقسیم بندی می‌کنیم و سپس دیتای آموزشی را با استفاده از oversampling بالанс می‌کنیم. حال دیتای آموزش برای آموزش مدل آماده است.

۶-۱-۱-آموزش مدل با پارامترهای پیشفرض

برای آموزش مدل با پارامترهای پیشفرض، یک شی Pipeline می‌سازیم که در آن ابتدا داده‌ها را استانداردسازی می‌کنیم و سپس مدل را آموزش می‌دهیم. شکل (۳۳) این پایپلاین را نمایش می‌دهد.

```

1 pipeline = Pipeline(
2     steps=[
3         ('scaler', StandardScaler()),
4         ("model", DecisionTreeClassifier())
5     ]
6 )

```

شکل ۳۳: پایپلاین آموزش مدل پیشفرض

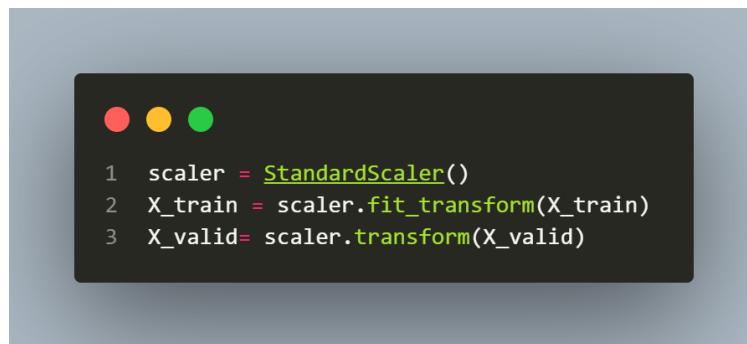
سپس با استفاده از متد fit() از شی پایپلاین و پاس دادن دیتاست آموزشی، استاندارد سازی و آموزش مدل را انجام می‌دهیم. همچنین با پاس دادن دیتاست تست به متد predict() داده‌های تست را استانداردسازی می‌کنیم و سپس اجازه می‌دهیم مدل پیش‌بینی را انجام دهد. حال با استفاده از تابع classification_report() معیارهای دقت، یادآوری و f1 را نمایش می‌دهیم. شکل (۳۴) گزارش به دست آمده را نمایش می‌دهد.

Classification Report:				
	precision	recall	f1-score	support
0.0	0.98	0.96	0.97	331
1.0	0.97	0.98	0.98	523
accuracy			0.97	854
macro avg		0.97	0.97	854
weighted avg		0.97	0.97	854

شکل ۳۴: گزارش به دست آمده از آموزش مدل پیشفرض

۱-۶-۲- پیدا کردن هایپرپارامترهای بهینه با استفاده از GridSearch

حال به بهینه سازی هایپرپارامترهای مدل می بردازیم. دیتاست را به دو بخش تقسیم می کنیم، و سپس دیتاست آموزش را بالانس می کنیم. سپس یک شی از کلاس StandardScaler() تعریف می کنیم و با متدهای .fit_transform() و .transform(). به ترتیب دیتاستهای آموزشی و تست را استاندارد سازی می کنیم. شکل (۳۵) نحوه انجام این کار را نمایش می دهد.



```

1  scaler = StandardScaler()
2  X_train = scaler.fit_transform(X_train)
3  X_valid= scaler.transform(X_valid)

```

شکل ۳۵: استانداردسازی دادهها

سپس یک دیکشنری تعریف می کنیم که کلیدهای آن، هایپرپارامترهای مدل هستند که می تواند آنها را به عنوان آرگومان دریافت کند و ولیوهای هر کلید این دیکشنری، یک لیست شامل مقادیری است که میخواهیم مدل به ترتیب آنها را دریافت کند. شکل (۳۶) دیکشنری هایپرپارامترها را نمایش می دهد.



```

1  parameters = {
2      'criterion': ['gini', 'entropy'],
3      'max_depth': [2, 4, 6, 8, 10, 12],
4      'min_samples_split': [2, 10, 20, 40, 60],
5      'min_samples_leaf': [1, 5, 10, 20],
6      'max_features': ['sqrt', 'log2', None],
7      'max_leaf_nodes': [None, 10, 20, 30, 50, 100]
8  }
9

```

شکل ۳۶: دیکشنری هایپرپارامترها

سپس یک شی از کلاس DecisionTreeClassifier() می‌سازیم، همچنین یک شی از کلاس GridSearchCV() می‌سازیم. سپس با استفاده از متدهای fit() و پاس دادن دیتاست آموزشی، Grid Search را آغاز می‌کنیم. این متدها همه ترکیب‌های ممکن از پارامترها را به مدل می‌دهد و مدل را با آن آموزش می‌دهد و ارزیابی می‌کند. شکل (۳۷) این کد را نمایش می‌دهد.

```
1 model = DecisionTreeClassifier()
2 clf_GS = GridSearchCV(model, parameters)
3 clf_GS.fit(X_train, y_train)
```

شکل ۳۷: پیداکردن ترکیب بهینه هایپرپارامترها با گرید سرج

سپس با استفاده از کد شکل (۳۸) پارامترهای بهینه را نمایش می‌دهیم.

```
1 print('Best Criterion:', clf_GS.best_estimator_.get_params())
```

شکل ۳۸: نمایش پارامترهای بهینه

شکل (۳۹) کلاس درخت تصمیم با آرگومان‌ها بهینه را نمایش می‌دهد.

```
1 model = DecisionTreeClassifier(ccp_alpha= 0.0, class_weight= None, criterion='gini',
2                                max_depth=12,max_features=None, max_Leaf_nodes=20,
3                                min_impurity_decrease=0, min_samples_Leaf=5, min_samples_split=2,
4                                min_weight_fraction_Leaf=0, monotonic_cst=None, splitter='best')
```

شکل ۳۹: درخت تصمیم با هایپرپارامترهای بهینه

حال به طور مشابه با قبل مدل را آموزش و تست می‌کنیم. شکل (۴۰) گزارش به دست آمده را نشان می‌دهد.

Classification Report:					
	precision	recall	f1-score	support	
0.0	0.98	0.96	0.97	331	
1.0	0.98	0.99	0.98	523	
accuracy			0.98	854	
macro avg	0.98	0.97	0.98	854	
weighted avg	0.98	0.98	0.98	854	

شکل ۴۰: گزارش دسته بندی درخت تصمیم با هایپرپارامترهای بهینه

مشاهده می شود که معیارهای ارزیابی مدل، یک تا دو درصد بهبود داشته اند.

حال با تابع `plot_tree()` و پاس دادن مدل، درخت ایجاد شده را رسم می کنیم. شکل (۴۱) کدی که وظیفه انجام این کار را دارد نمایش می دهد. همچنین شکل (۴۲) درخت به دست آمده را نمایش می دهد.



```

1 # Plot the decision tree
2 plt.figure(figsize=(20,10))
3 plot_tree(model, filled=True)
4 plt.show()

```

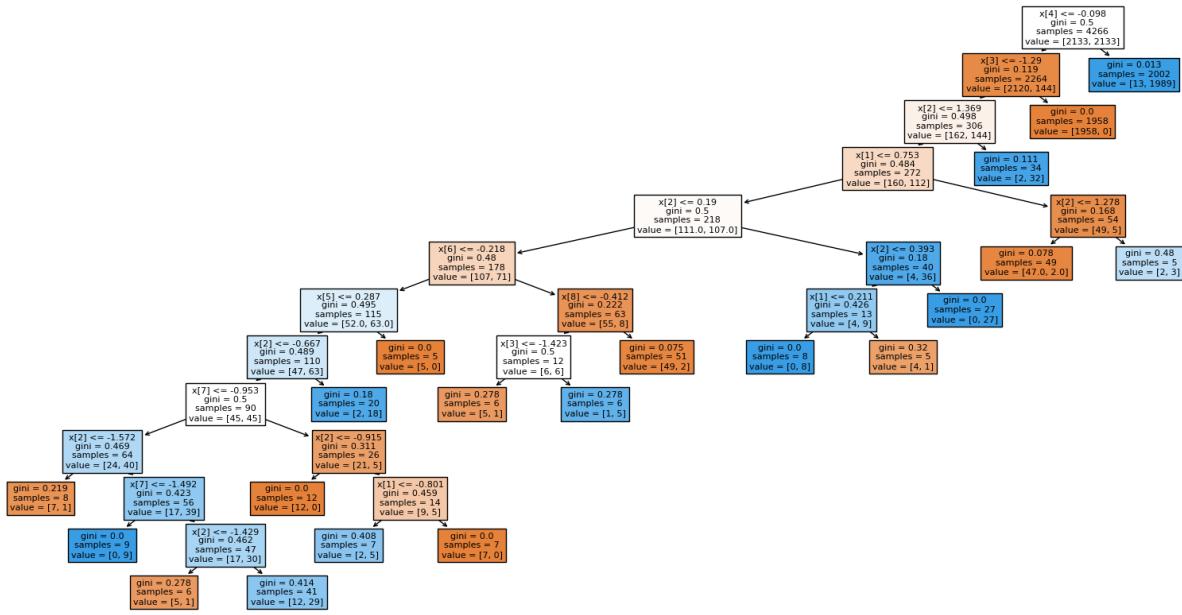
شکل ۴۱: کد مربوط به نمایش شکل درخت

۱-۶-۳- هرس کردن درخت و نمایش مجدد آن (بخش امتیازی)

پارامتر `ccp_alpha` پارامتر کلیدی برای هرس کردن درخت است. اگر مقدار آن برابر صفر باشد، هیچگونه هرس کردنی صورت نمی گیرد. همچنین با پارامترهای `min_samples_split` و `min_samples_leaf` و `max_leaf_nodes` و `max_depth` نیز می توان درخت را هرس کرد. این پارامترها به ترتیب، حداقل عمق درخت، حداقل شاخه ها، مینیمم تعداد شاخه های نمونه و مینیمم تعداد تقسیم نمونه ها را تعیین می کنند.

برای هرس کردن درخت، تمرکز خود را روی پارامتر `ccp_alpha` قرار می دهیم. مدل را با پارامترهای قبلی آموزش می دهیم، ولی مقدار این پارامتر را برابر ۰.۰۱ قرار می دهیم. سپس مدل را آموزش می دهیم و تست می کنیم. شکل (۴۳) گزارش به دست آمده از این مدل هرس شده را نمایش می دهد. همچنین شکل (۴۴) درخت هرس شده را نمایش می دهد.

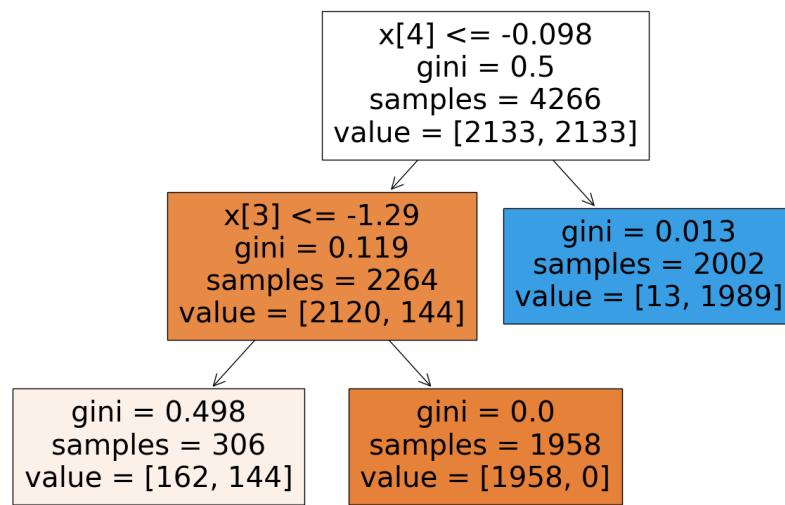
مشاهده می شود که درخت هرس شده بسیار کوچکتر از درخت اصلی است ولی گزارش عملکرد آن تنها چند درصد ضعیفتر از درخت اصلی است. مزایای این درخت، توضیح پذیری ساده آن و مقاومت آن در برابر بیش برازش است. در زمینه هایی که این دو مورد اهمیت دارند، استفاده از درخت هرس شده توصیه می شود.



شکل ۴۲: درخت به دست آمده از هایپرپارامترهای بهینه شده

Classification Report:					
	precision	recall	f1-score	support	
0.0	0.89	0.99	0.94	331	
1.0	0.99	0.92	0.96	523	
accuracy			0.95	854	
macro avg	0.94	0.96	0.95	854	
weighted avg	0.95	0.95	0.95	854	

شکل ۴۳: گزارش عملکرد درخت هرس شده

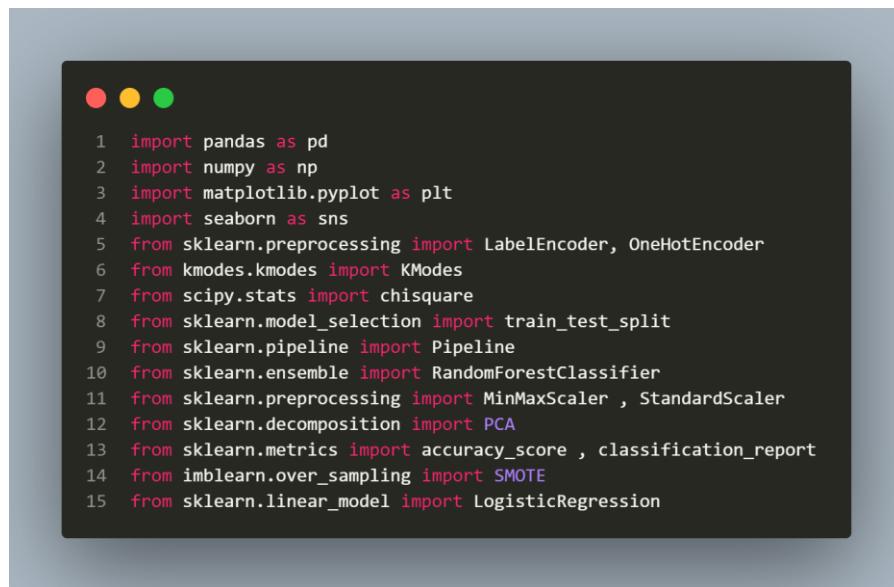


شکل ۴۴: درخت هرس شده

۲-پاسخ سوال دوم

۱-آماده سازی داده‌ها

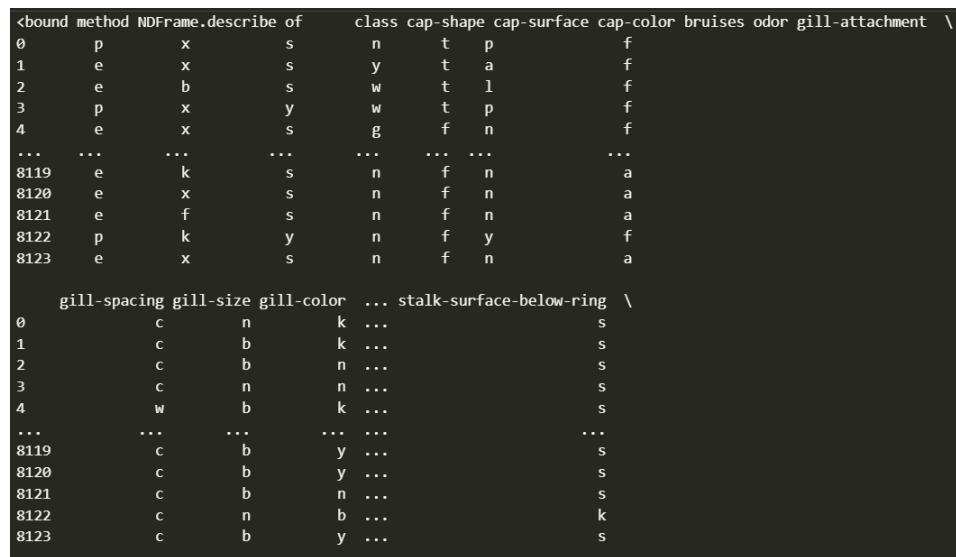
ابتدا کتابخانه‌های مورد نیاز را وارد می‌کنیم. شکل (۴۵) این کتابخانه‌ها را نمایش می‌دهد.



```
● ● ●
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 import seaborn as sns
5 from sklearn.preprocessing import LabelEncoder, OneHotEncoder
6 from kmodes.kmodes import KModes
7 from scipy.stats import chisquare
8 from sklearn.model_selection import train_test_split
9 from sklearn.pipeline import Pipeline
10 from sklearn.ensemble import RandomForestClassifier
11 from sklearn.preprocessing import MinMaxScaler, StandardScaler
12 from sklearn.decomposition import PCA
13 from sklearn.metrics import accuracy_score, classification_report
14 from imblearn.over_sampling import SMOTE
15 from sklearn.linear_model import LogisticRegression
```

شکل ۴۵: وارد کردن کتابخانه‌های مورد نیاز

سپس متدهای describe() را استفاده می‌کنیم. شکل (۴۶) خروجی این متدهای را نمایش می‌دهد.



```
<bound method NDFrame.describe of      class cap-shape cap-surface cap-color bruises odor gill-attachment \
0   p       x     s     n     t     p         f
1   e       x     s     y     t     a         f
2   e       b     s     w     t     l         f
3   p       x     y     w     t     p         f
4   e       x     s     g     f     n         f
...
...   ...   ...
8119  e       k     s     n     f     n         a
8120  e       x     s     n     f     n         a
8121  e       f     s     n     f     n         a
8122  p       k     y     n     f     y         f
8123  e       x     s     n     f     n         a

gill-spacing gill-size gill-color ... stalk-surface-below-ring \
0           c     n     k   ...           s
1           c     b     k   ...           s
2           c     b     n   ...           s
3           c     n     n   ...           s
4           w     b     k   ...           s
...
...   ...
8119       c     b     y   ...           s
8120       c     b     y   ...           s
8121       c     b     n   ...           s
8122       c     n     b   ...           k
8123       c     b     y   ...           s
```

شکل ۴۶: خروجی متدهای describe()

سپس ده سطر رندوم را نمایش می‌دهیم. برای انجام این کار، مشابه سوال یک عمل می‌کنیم. شکل (۴۷) این خروجی را نمایش می‌دهد.

class	cap-shape	cap-surface	cap-color	bruises	odor	gill-attachment	gill-spacing	gill-size	gill-color	---	stalk-surface-below-ring	stalk-color-above-ring	stalk-color-below-ring	veil-type	veil-color	ring-number	ring-type	spore-print-color	population	habitat	
1971	e	f	f	n	f	n	f	w	b	h	-	f	w	w	p	w	o	e	n	s	g
6654	p	f	s	e	f	y	f	c	n	b	-	s	p	p	p	w	o	e	w	v	l
5606	p	x	y	n	f	f	f	c	n	b	-	s	w	p	p	w	o	e	w	v	l
3332	e	f	y	g	t	n	f	c	b	n	-	s	g	p	p	w	o	p	n	y	d
6988	p	f	s	e	f	s	f	c	n	b	-	s	p	p	p	w	o	e	w	v	l
5761	p	x	y	n	f	y	f	c	n	b	-	s	w	p	p	w	o	e	w	v	l
5798	p	x	s	g	t	f	f	c	b	h	-	f	w	w	p	w	o	p	h	s	u
3064	p	x	y	y	f	f	f	c	b	g	-	k	n	b	p	w	o	l	h	y	p
1811	e	f	f	n	t	n	f	c	b	n	-	s	g	w	p	w	o	p	k	v	d
3422	e	f	y	n	t	n	f	c	b	n	-	s	w	w	p	w	o	p	k	y	d

10 rows × 23 columns

شکل ۴۷: ده سطر رندوم دیتافریم

از خروجی این بخش و خروجی متدهای info() مشاهده می‌شود که تمامی ستون‌های این دیتافریم به صورت داده‌های دسته‌ای هستند. حال در قدم بعدی بررسی می‌کنیم که کدام یک از دسته‌ها به صورت دوستایی هستند. برای این منظور از متدهای value_counts() استفاده می‌کنیم. برای هر ستون این متدها را اجرا می‌کنیم و نتیجه را نمایش می‌دهیم. کد این بخش دقیقاً همانند کد شکل (۱۱) است. بخشی از خروجی این کد در شکل (۴۸) نمایش داده شده است.

```

class: class
e    4208
p    3916
Name: count, dtype: int64
cap-shape: cap-shape
x    3656
f    3152
k    828
b    452
s     32
c      4
Name: count, dtype: int64
cap-surface: cap-surface
y    3244
s    2556
f    2320
g      4
Name: count, dtype: int64
cap-color: cap-color
n    2284
g    1840
e    1500
y    1072
w    1040
b     168
...
u     368
m     292
w     192
Name: count, dtype: int64

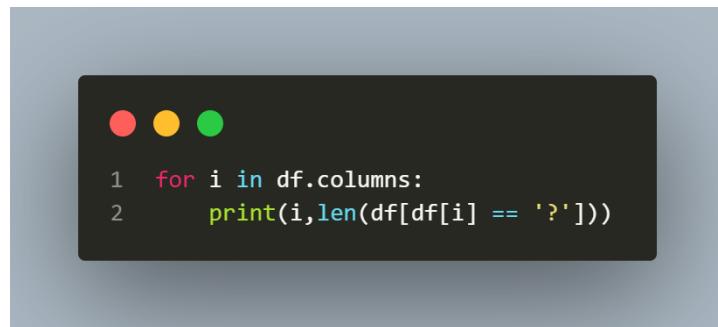
```

شکل ۴۸: خروجی تعداد هر دسته در هر ستون

از بررسی این خروجی متوجه دو مورد می‌شویم. مورد اول اینکه، ستون‌های bruises gill-attachment gill-spacing gill-size stalk-shape شامل دسته‌های دو دوستی هستند. خود ستون هدف نیز به صورت دو دوستی است.

موردن بعدی این است که ستون veil-type تنها شامل یک مقدار است. در نتیجه آن را با استفاده از متدهای drop() حذف می‌کنیم.

حال بررسی می‌کنیم که آیا در ستون‌ها، مقدار؟ داریم یا خیر. برای این منظور از کد شکل (۴۹) استفاده می‌کنیم.



```
● ● ●  
1 for i in df.columns:  
2     print(i, len(df[df[i] == '?']))
```

شکل ۴۹: پیدا کردن ؟ در دیتاست

مشاهده می‌شود که ستون stalk-root مقدار ۲۴۸۰ دارد. این مقدار حدود یک چهارم مقادیر دیتاست است. از آنجایی که دیتاست را به زودی one-hot-encode می‌کنیم و تعداد فیچرها بسیار افزایش می‌یابد و همچنین تعداد گمشده‌ها بسیار زیاد است، حذف این ستون تصمیمی منطقی است. پس این ستون را نیز حذف می‌کنیم.

حال ستون‌های دودویی را با استفاده از LabelEncoder به ستون‌های عددی تبدیل می‌کنیم. ابتدا از کلاس () یک شی می‌سازیم و همچنین ستون‌هایی که نیاز به این نوع انکود دارند را در یک لیست قرار می‌دهیم. سپس برای هر ستون در این لیست، با استفاده از متدها fit() و transform() یکی ستون‌ها را انکود می‌کنیم. شکل (۵۰) نحوه انجام این کار را نمایش می‌دهد.



```
● ● ●  
1 label_encoder = LabelEncoder()  
2 # List of columns to encode  
3 labelEncode = ["class", "bruises", 'gill-attachment', 'gill-spacing', 'gill-size', 'stalk-shape']  
4  
5 # Applying LabelEncoder to each column in the list  
6 for column in labelEncode:  
7     df[column] = label_encoder.fit_transform(df[column])
```

شکل ۵۰: انکود کردن ستون‌های دودویی

حال سایر ستون‌ها که نیاز به one-hot-encode دارند را در یک لیست دیگر قرار می‌دهیم و با استفاده از تابع get_dummies() ستون‌ها را one-hot-encode می‌کنیم. از آنجایی که خروجی این تابع به شکل True و False است، با استفاده از متد astype() و پاس دادن "int" داده‌ها را به صفر و یک تبدیل می‌کنیم.

در قدم بعدی، به پیدا کردن و حذف داده‌های پرت می‌پردازیم.

۲-۲-حذف داده‌های پرت

حذف داده‌های پرت در حالتی که همه داده‌ها حالت گسسته دارند، با روش‌های متداول مانند IQR یا شش سیگما امکان‌پذیر نیست. برای این منظور از خوشبندی K-mode استفاده می‌کنیم و خوشبندی که اندازه‌ای کوچکتر از یک مقدار به خصوص دارند را حذف می‌کنیم.



```

1 OneHotEncode = [col for col in df.columns if col not in labelEncode]
2 # Apply one-hot encoding
3 df = pd.get_dummies(df, columns=OneHotEncode)

```

شکل ۵۱: one-hot-encode کردن داده‌ها

برای این منظور از کلاس KModes یک شی می‌سازیم. تعداد خوشه‌ها را روی ۵ تنظیم می‌کنیم. با استفاده از دستور fit_predict() و پاس دادن دیتافریم، خوشه‌بندی را انجام می‌دهیم. سپس خوشه‌هایی را که کمتر از یک مقدار به خصوص دارند را در لیست outlier_cluster می‌کنیم و با استفاده از تابع isin() آنها را حذف می‌کنیم. در نهایت داده‌های پرت را نمایش می‌دهیم. مشاهده می‌شود با پنج خوشه و با افزایش جمعیت خوشه‌ها حتی تا هشت‌تصد مورد، داده‌ی پرتی یافت نمی‌شود. شکل (۵۲) کد پیدا کردن داده‌های پرت را با استفاده از خوشه‌بندی نمایش می‌دهد.



```

1 # Apply k-modes clustering
2 km = KModes(n_clusters=5, init='Huang', n_init=5, verbose=1)
3 clusters = km.fit_predict(df)
4
5 # Find clusters with few points
6 cluster, counts = np.unique(clusters, return_counts=True)
7 outlier_clusters = cluster[counts < 801]
8 outlier_data = df[np.isin(clusters, outlier_clusters)]
9
10 print("Outlier data based on clustering:")
11 print(outlier_data)

```

شکل ۵۲: پیدا کردن داده‌های پرت با استفاده از خوشه‌بندی

۳-۲-انتخاب ویژگی

از آنجایی که تعداد ویژگی‌ها بسیار زیاد است، یکی از بهترین راه‌هایی که می‌توان با استفاده از آن انتخاب ویژگی کرد، PCA است. مزیت PCA این است که تعداد زیاد ابعاد را به یک تعداد کم از مولفه‌های‌ها که تاثیر زیادی روی ستون هدف دارند تبدیل می‌کند.

برای این منظور پس از تقسیم‌بندی به داده‌های تست و آموزش، یک شی از کلاس پایپ‌لاین درست می‌کنیم و سپس به ترتیب کلاس‌های PCA، StandardScaler و RandomForestClassifier را به آن پاس می‌دهیم. به کلاس PCA هم آرگومان n_components را پاس می‌دهیم. هدف این است که با کمترین تعداد PC ممکن، بهترین نتیجه ممکن را برای آموزش و تست جنگل تصادفی پیدا کنیم. ابتدا ۸۰ مولفه را امتحان می‌کنیم و همانند قبل با استفاده از متدهای fit() و predict(). مدل را آموزش می‌دهیم و تست می‌کنیم. این تعداد به ما دقت ۱۰۰ درصد را می‌دهد. سپس تعداد کمتر را امتحان می‌کنیم. متوجه می‌شویم که برای دستیابی به دقت صد درصد تنها به ۱۵ مولفه اصلی احتیاج داریم.

شکل (۵۳) شی پایپ‌لاین را نمایش می‌دهد و شکل (۵۴) گزارش عملکرد به دست آمده از آموزش و تست ۱۵ مولفه را نمایش می‌دهد.



```

1 # Define the pipeline with PCA and RandomForest
2 pipeline = Pipeline([
3     ('scaler', StandardScaler()),
4     ('pca', PCA(n_components=15)),
5     ('rf', RandomForestClassifier(n_estimators=100, random_state=0))
6 ])

```

شکل ۵۳: پایپ‌لاین ساخته شده برای تحلیل مولفه اصلی

Classification Report:					
	precision	recall	f1-score	support	
0	1.00	1.00	1.00	1699	
1	1.00	1.00	1.00	1551	
accuracy			1.00	3250	
macro avg		1.00	1.00	1.00	3250
weighted avg		1.00	1.00	1.00	3250

شکل ۵۴: گزارش به دست آمده از آموزش و تست مدل با ۱۵ مولفه اصلی

در نتیجه برای انتخاب ویژگی، از ۱۵ مولفه اصلی استفاده می‌کنیم.

برای این منظور ابتدا دیتابست را با استفاده از کلاس StandardScaler() و متدهای fit_transform() استاندارد سازی می‌کنیم. سپس PCA را اجرا می‌کنیم و ۱۵ مولفه اصلی اول را انتخاب می‌کنیم. شکل (۵۵) نحوه انجام این کار را نمایش می‌دهد.

۲-۴-آموزش و تست رگرسیون لجستیک

همانند بخش‌های قبلی، ابتدا دیتابست جدید را به قسمت‌های آموزش و تست تقسیم می‌کنیم. سپس همانند بخش قبلی داده‌ها را استاندارد می‌کنیم. سپس یک شی از کلاس LogisticRegression() ایجاد می‌کنیم. سپس با استفاده از متدهای fit() و predict() مدل را آموزش می‌دهیم و با استفاده از متدهای score() و accuracy() مدل را تست می‌کنیم. شکل (۵۵) نحوه انجام این کار را نمایش می‌دهد.



```

1 # Initialize PCA and apply
2 scaler = StandardScaler()
3 scaled_df= scaler.fit_transform(df.drop('class', axis=1))
4 pca = PCA(n_components=15)
5 principal_components = pca.fit_transform(scaled_df)
6
7 # Create a DataFrame with the principal components
8 pc_df = pd.DataFrame(data=principal_components,
9                      columns=['PC1', 'PC2', 'PC3', 'PC4', 'PC5', 'PC6', 'PC7', 'PC8', 'PC9', 'PC10', 'PC11', 'PC12', 'PC13', 'PC14', 'PC15'])
10

```

شکل ۵۵: انتخاب ۱۵ مولفه اصلی به عنوان ویژگی‌ها



```

1 # Initialize the Logistic Regression classifier
2 classifier = LogisticRegression()
3 # Fit the classifier to the training data
4 classifier.fit(X_train, y_train)
5 # Predict labels for the test data
6 preds = classifier.predict(X_valid)

```

شکل ۵۶: ساخت، آموزش و تست مدل رگرسیون لجستیک

سپس همانند قسمت‌های قبل گزارش عملکرد را نمایش می‌دهیم. شکل (۵۷) گزارش عملکرد رگرسیون لجستیک را نمایش می‌دهد.

Classification Report:					
	precision	recall	f1-score	support	
0	1.00	0.99	0.99	1272	
1	0.99	0.99	0.99	1166	
accuracy			0.99	2438	
macro avg	0.99	0.99	0.99	2438	
weighted avg	0.99	0.99	0.99	2438	

شکل ۵۷: گزارش عملکرد رگرسیون لجستیک

با توجه به عملکرد مدل رگرسیون لجستیک، مشاهده می‌شود که تکنیک‌های استفاده شده برای انتخاب ویژگی و تشخیص داده‌های نویز تکنیک‌های مناسبی بوده‌اند.

حال برای جلوگیری از بیش برازش، یک مدل را با استفاده از Cross Validation آموزش می‌دهیم. همانند قبل یک شی از کلاس رگرسیون لجستیک می‌سازیم، سپس از کلاس cross_val_score یک شی می‌سازیم. آرگومان‌های کلاس را به ترتیب شی مدل، داده‌های آموزش و تعداد بخش‌ها قرار می‌دهیم. پس از انجام آموزش میانگین و انحراف استاندارد دقت را نمایش می‌دهیم. شکل (۵۸) نحوه انجام این کار و شکل (۵۹) خروجی میانگین و انحراف استاندارد را نمایش می‌دهیم.

```

1
2
3 # Initialize Logistic Regression classifier model
4 classifier = LogisticRegression()
5
6 # Perform cross-validation
7 cv_scores = cross_val_score(classifier, X_train, y_train, cv=10)
8
9 # Calculate the mean and standard deviation of cross-validation scores
10 mean_cv_score = cv_scores.mean()
11 std_cv_score = cv_scores.std()
12
13 # Print the mean and standard deviation of cross-validation scores
14 print("Mean Cross-Validation Score:", mean_cv_score)
15 print("Standard Deviation of Cross-Validation Score:", std_cv_score)
16

```

شکل ۵۸: آموزش مدل با Cross Validation

```

Mean Cross-Validation Score: 0.99063510644462
Standard Deviation of Cross-Validation Score: 0.003163559810982977

```

شکل ۵۹: نتیجه Cross Validation

مشاهده می شود که هر دو روش دقت یکسانی نتیجه می دهند که نشان می دهد مدل دچار بیش برازش نشده.

۳-پاسخ سوال سوم

ابتدا کتابخانه های مورد نیاز را وارد می کنیم. شکل (۶۰) این کتابخانه ها را نمایش می دهد.

```

1 import pandas as pd
2 import numpy as np
3 import re
4 from nltk.corpus import stopwords
5 from nltk.tokenize import word_tokenize
6 from nltk.stem import WordNetLemmatizer
7 from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
8 from sklearn.model_selection import train_test_split
9 from sklearn.feature_selection import SelectKBest, chi2
10 from sklearn.naive_bayes import GaussianNB
11 from sklearn.preprocessing import StandardScaler
12 from sklearn.metrics import classification_report
13 from imblearn.over_sampling import SMOTE
14 from joblib import dump, load

```

شکل ۶۰: وارد کردن کتابخانه های مورد نیاز

پس از خواندن دیتابست، ابتدا با استفاده از متدهای info(). تعداد رکورد موجود از هر لیبل را نمایش می‌دهیم. شکل (۶۱) خروجی این کار را نمایش می‌دهد.

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50000 entries, 0 to 49999
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype  
---  --          -----          object 
 0   review      50000 non-null   object 
 1   sentiment   50000 non-null   object 
dtypes: object(2)
memory usage: 781.4+ KB
```

شکل ۶۱: تعداد ایمیل‌های با لیبل مثبت و منفی

سپس ده سطر رندوم را نمایش می‌دهیم. برای انجام این کار، مشابه سوال یک عمل می‌کنیم. شکل (۶۲) این خروجی را نمایش می‌دهد.

		review	sentiment
33553	I really liked this Summerslam due to the look...		positive
9427	Not many television shows appeal to quite as m...		positive
199	The film quickly gets to a major chase scene w...		negative
12447	Jane Austen would definitely approve of this o...		positive
39489	Expectations were somewhat high for me when I ...		negative
42724	I've watched this movie on a fairly regular ba...		positive
10822	For once a story of hope highlighted over the ...		positive
49498	Okay, I didn't get the Purgatory thing the fir...		positive
4144	I was very disappointed with this series. It h...		negative
36958	The first 30 minutes of Tinseltown had my fing...		negative

شکل ۶۲: نمایش ده سطر رندوم

حال برای تبدیل لیبل‌ها به صفر و یک، روی سطر هدف متدهای apply() را اجرا می‌کنیم. آرگومان این متدها قرار می‌دهیم که یکی لیبل‌ها رو می‌گیرد و اگر مثبت بود آن را با یک و اگر منفی بود آن را با صفر جایگزین می‌کنیم. شکل (۶۳) نحوه انجام این کار را نمایش می‌دهد.

```
● ● ●
1 df['sentiment']=df['sentiment'].apply(Lambda x:1 if x=='positive' else 0)
```

شکل ۶۳: جایگزینی لیبل‌ها با صفر و یک

۱-۳-پیش پردازش داده‌ها

۱-۱-۳-حذف تگ‌های

اولین قدم، حذف تگ‌های `

` از رشته‌ی هرنظر است. برای این منظور از متده استفاده می‌کنیم، کد شکل (۶۴) نحوه انجام این کار را نمایش می‌دهد.

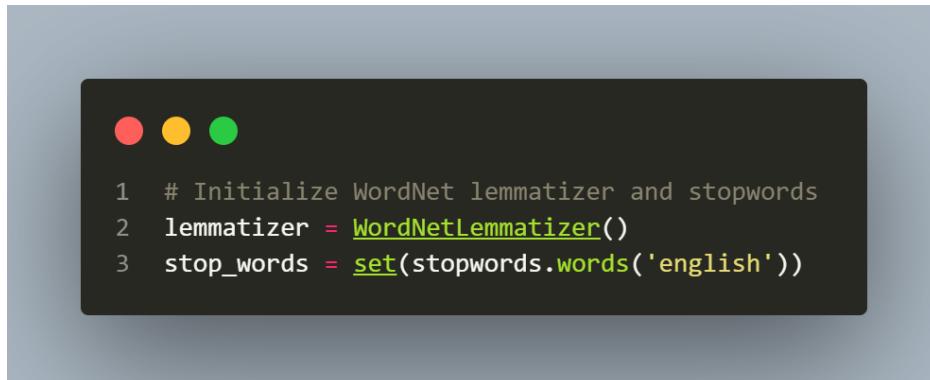


```
1 # Remove the "<br /><br />" characters from the text column
2 df['review'] = df['review'].str.replace('<br /><br />', ' ')
```

شکل ۶۴: حذف تگ‌های

۲-۱-۳-ساخت تابع پیش‌پردازش

برای انجام پیش‌پردازش یک تابع تعریف می‌کنیم که به ترتیب مراحل پیش‌پردازش را روی متن پاس داده شده به عنوان آرگومان انجام می‌دهد. ابتدا احتیاج داریم که یک شی از کلاس WordNetLemmatizer() بسازیم و همچنین ایستواژه‌های زبان انگلیسی را در یک مجموعه ذخیره کنیم. شکل (۶۵) نحوه انجام این کار را نمایش می‌دهد.



```
1 # Initialize WordNet lemmatizer and stopwords
2 lemmatizer = WordNetLemmatizer()
3 stop_words = set(stopwords.words('english'))
```

شکل ۶۵: ساخت شی ریشه‌یاب و مجموعه ایستواژه‌ها

حال تابع را به صورت نشان داده شده در شکل (۶۶) تعریف می‌کنیم. این تابع، یکی متن‌ها را دریافت می‌کند. سپس با استفاده از تابع word_tokenize() هر کاراکتری غیر از حروف کوچک و بزرگ انگلیسی و اسپیس را با اسپیس جایگذاری می‌کنیم. سپس با استفاده از تابع Lemmatization() و پاس دادن متن به عنوان آرگومان، متن را توکن می‌کنیم. سپس هر توکن را با استفاده از List Comprehension برسی می‌کنیم. حال توکن‌هایی را نگه می‌داریم که در مجموعه ایستواژه‌ها قرار ندارند و این توکن‌ها را به کاراکترهای انگلیسی با حروف کوچک تبدیل می‌کنیم. سپس هر توکن را با استفاده از متده lemmatize() به ریشه بر می‌گردانیم. در نهایت توکن‌ها را دوباره در کنار هم قرار می‌دهیم و جملات را تشکیل می‌دهیم.



```
1 def preprocess_text(text):
2     # Remove special characters and numbers
3     text = re.sub(r'[^a-zA-Z\s]', '', text)
4     # Tokenization
5     tokens = word_tokenize(text)
6     # Lowercasing and removing stopwords
7     tokens = [token.lower() for token in tokens if token.lower() not in stop_words]
8     # Lemmatization
9     tokens = [lemmatizer.lemmatize(token) for token in tokens]
10    # Join tokens back into a string
11    preprocessed_text = ' '.join(tokens)
12    return preprocessed_text
```

شکل ۶۶: تابع پیش‌پردازش متن

سپس با استفاده از متدها apply() و استفاده از یک تابع لامبدا، یکی متن‌ها را به این تابع پاس میدهیم و متن‌های پیش‌پردازش شده را دریافت می‌کنیم. شکل (۶۷) نحوه انجام این کار را نمایش می‌دهد.



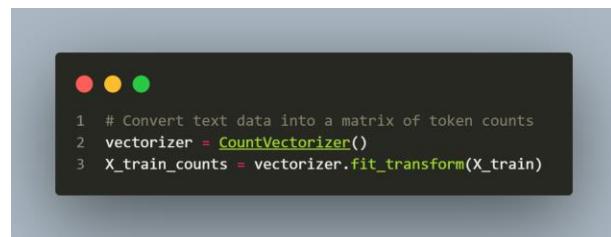
```
1 df['review'] = df['review'].apply(lambda text : preprocess_text(text))
```

شکل ۶۷: پیش‌پردازش متن‌ها با استفاده از تابع تعريف شده

۲-۳-استفاده از CountVectorizer

مجموعه‌ای از اسناد متنی را به یک ماتریس تبدیل می‌کند که در آن ردیف‌ها نشان دهنده اسناد و ستون‌ها نشان دهنده کلمه‌ها هستند. این تابع تعداد تکرار هر توکن را در هر سند شمارش می‌کند، و یک ماتریس Document-term با مقادیر صحیح نشان دهنده فراوانی هر کلمه ایجاد می‌کند.

برای استفاده از این متدها، ابتدا دیتاست را به دو دسته آموزشی و آزمایشی تقسیم می‌کنیم. سپس از کلاس CountVectorizer یک شی می‌سازیم. سپس با پاس دادن دیتاست آموزشی به متدها fit_transform() ماتریس Document-term را از داده‌های آموزشی ایجاد می‌کنیم. شکل (۶۸) نحوه انجام این کار را نمایش می‌دهد.



```
1 # Convert text data into a matrix of token counts
2 vectorizer = CountVectorizer()
3 X_train_counts = vectorizer.fit_transform(X_train)
```

شکل ۶۸: بردار کردن متون با استفاده از CountVectorizer

سپس با استفاده از تابع SelectKBest و آزمون مربع کای، ۵۰۰۰ کلمه را به عنوان ویژگی های مهم انتخاب می کنیم. شکل (۶۹) نحوه انجام این کار را نمایش می دهد.



```
● ● ●  
1 # Select the top k features based on chi-square test  
2 k = 5000 # Number of top features to select  
3 selector = SelectKBest(score_func=chi2, k=k)  
4 X_train_selected = selector.fit_transform(X_train_counts, y_train)
```

شکل ۶۹: انتخاب ۵۰۰۰ کلمه مهم با استفاده از آزمون مربع کای

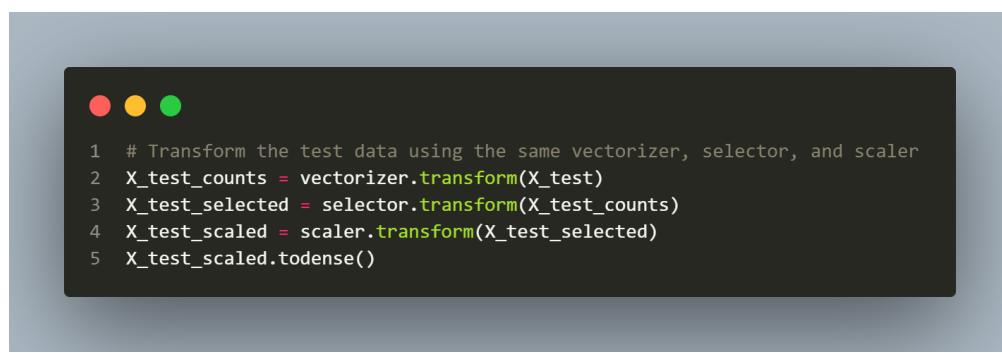
در قدم بعدی ماتریس به دست آمده را استاندارد سازی می کنیم و به شکل یک ماتریس چگال در می آوریم. شکل ۷۰ نحوه انجام این کار را نمایش می دهد.



```
● ● ●  
1 # Scale the selected features using StandardScaler  
2 scaler = StandardScaler(with_mean=False)  
3 X_train_scaled = scaler.fit_transform(X_train_selected)  
4 X_train_scaled.todense()
```

شکل ۷۰: استاندارد سازی و تبدیل ماتریس به ماتریس چگال

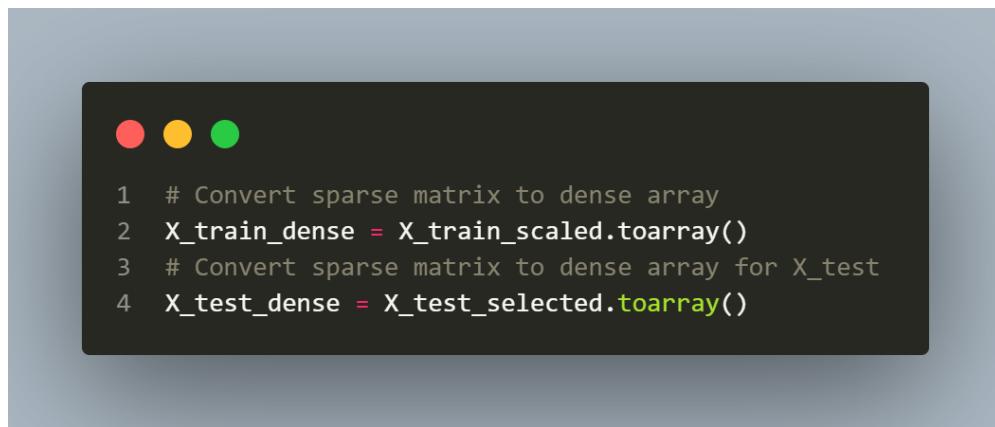
حال با استفاده از متدها transform() و fit() از هر یک اشیا ساخته شده، دیتاست تست را هم به ترتیب برداری می کنیم، ویژگی ها را انتخاب می کنیم و ماتریس به دست آمده را استاندارد می کنیم. همچنین در نهایت ماتریس به دست آمده را چگال می کنیم. شکل (۷۱) نحوه انجام این کار را نمایش می دهد.



```
● ● ●  
1 # Transform the test data using the same vectorizer, selector, and scaler  
2 X_test_counts = vectorizer.transform(X_test)  
3 X_test_selected = selector.transform(X_test_counts)  
4 X_test_scaled = scaler.transform(X_test_selected)  
5 X_test_scaled.todense()
```

شکل ۷۱: پیش پردازش داده های تست

در قدم بعدی ماتریس‌های ایجاد شده را به آرایه‌های خلوت تبدیل می‌کنیم. دلیل انجام این کار این است که متدهای یادگیری ماشین کتابخانه سایکیتلرن توانایی کار با ماتریس‌های چگال را ندارند. شکل (۷۲) نحوه انجام این کار را نمایش می‌دهد.



```
1 # Convert sparse matrix to dense array
2 X_train_dense = X_train_scaled.toarray()
3 # Convert sparse matrix to dense array for X_test
4 X_test_dense = X_test_selected.toarray()
```

شکل ۷۲: تبدیل ماتریس‌های چگال به آرایه‌های خلوت

سپس در قدم بعدی با استفاده از oversampling دیتاست آموزشی را بالанс می‌کنیم. نحوه انجام این کار دقیقاً به شکل کار انجام شده در قسمت ۱-۴ و شکل (۷۲) است. حال داده‌ها برای آموزش و تست مدل آماده هستند.

برای آموزش مدل، از کلاس GaussianNB() یک شی می‌سازیم، با پاس دادن دیتاست آموزشی به متده fit(). از این شی، آموزش مدل را انجام می‌دهیم. همچنین با پاس دادن دیتاست تست به متده predict(). پیش‌بینی را انجام می‌دهیم. شکل (۷۳) نحوه انجام این کار را نمایش می‌دهد.



```
1 # Train Naive Bayes classifier
2 nb_classifier = GaussianNB()
3 nb_classifier.fit(X_train_dense, y_train)
4 # Make predictions using Naive Bayes classifier
5 y_pred = nb_classifier.predict(X_test_dense)
```

شکل ۷۳: آموزش و تست مدل بیز

شکل (۷۴) نیز گزارش عملکرد این مدل روی دیتاهای تست را نمایش می‌دهد. نحوه ایجاد این گزارش دقیقاً مشابه توضیحات ارائه شده در بخش ۱-۶ است.

Classification Report:					
	precision	recall	f1-score	support	
0	0.52	0.98	0.68	7411	
1	0.86	0.10	0.19	7589	
accuracy			0.54	15000	
macro avg	0.69	0.54	0.43	15000	
weighted avg	0.69	0.54	0.43	15000	

شکل ۷۴: گزارش عملکرد مدل نابو بیز

۳-۳- استفاده از TF-IDF

TF-IDF به هر عبارت در یک سند بر اساس فراوانی آن در سند (TF) و نادر بودن آن در همه اسناد (IDF) وزن اختصاص می دهد. این طرح وزن دهی به اولویت بندی عباراتی که هم در یک سند متداول هستند و هم در کل مجموعه منحصر به فرد هستند، کمک می کند و آنها را برای کارهای پایین دستی مانند طبقه بندی یا خوشه بندی متمایزتر و آموزنده تر می کند.

برای استفاده از TF-IDF پس از تقسیم دیتابست به آموزش و تست، از کلاس TfidfVectorizer() یک شی می سازیم. سپس با استفاده از متدها fit_transform() و پاس دادن متن بردارها را ایجاد می کنیم و سپس نتیجه را در یک متغیر دیگر ذخیره می کنیم. سپس تمامی مراحل بالا نس کردن دیتابست آموزشی، پیدا کردن ۵۰۰۰ ویژگی برتر، استاندارد کردن، ایجاد آرایه های خلوت، و آموزش و تست مدل را همانند بخش قبلی انجام می دهیم. شکل (۷۵) گزارش عملکرد مدل آموزش دیده روی ویژگی های به دست آمده از TF-IDF را نمایش می دهد.

Classification Report:					
	precision	recall	f1-score	support	
0	0.52	0.98	0.68	7411	
1	0.84	0.12	0.20	7589	
accuracy			0.54	15000	
macro avg	0.68	0.55	0.44	15000	
weighted avg	0.68	0.54	0.44	15000	

شکل ۷۵: گزارش عملکرد مدل به دست آمده از دیتابست TF-IDF

مشاهده می شود که هر دو مدل عملکرد چندان مناسب ندارند. ولی عملکرد مدل آخر کمی بهتر است. پس این مدل را برای استفاده در آینده ذخیره می کنیم. برای این کار از کتابخانه joblib و تابع dump() استفاده می کنیم. آرگومان های این تابع را به ترتیب شی مدل و نامی که می خواهیم با آن ذخیره شود قرار می دهیم. شکل (۷۶) نحوه استفاده از این مدل را نمایش می دهد.

```
# Save the trained model to a file
dump(nb_classifier, 'naive_bayes_model.joblib')
```

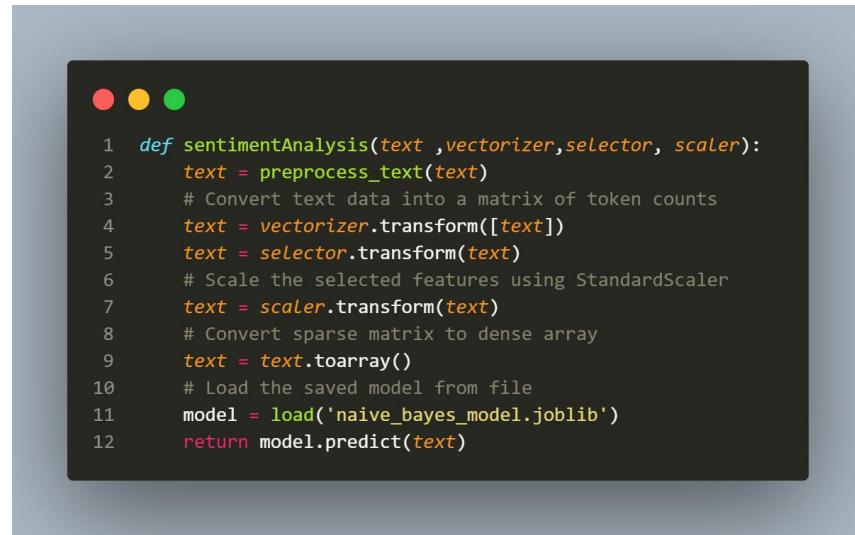
شکل ۷۶: ذخیره کردن مدل برای استفاده در آینده

۴-۳- ساخت تابعی که به نظرات جدید برچسب می دهد

برای اینکه عواطف نظرات جدید را بسنجیم، یک تابع همانند کدی که در شکل (۷۷) نشان داده شده ایجاد می کنیم.

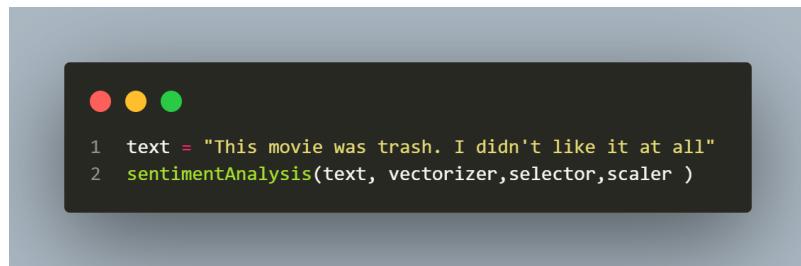
این تابع، متن، شی انتخاب کننده ویژگی، و شی استانداردساز را دریافت می کند و به ترتیب اعمال زیر را انجام می دهد:

ابتدا با استفاده از تابع تعریف شده در بخش ۲-۱-۳ متن را آمده می کند. سپس آن را به بردار تبدیل می کند. سپس انتخاب ویژگی را انجام می دهد و بردار ایجاد شده را استاندارد و در نهایت به یک آرایه خلوت تبدیل می کند. سپس با استفاده از تابع load() از کتابخانه joblib مدل را لود می کند و با استفاده از آن پیش بینی را انجام می دهد و نتیجه را برمی گرداند. شکل (۷۸) یک نمونه از کاربرد این تابع را نمایش می دهد.



```
1 def sentimentAnalysis(text ,vectorizer,selector, scaler):
2     text = preprocess_text(text)
3     # Convert text data into a matrix of token counts
4     text = vectorizer.transform([text])
5     text = selector.transform(text)
6     # Scale the selected features using StandardScaler
7     text = scaler.transform(text)
8     # Convert sparse matrix to dense array
9     text = text.toarray()
10    # Load the saved model from file
11    model = load('naive_bayes_model.joblib')
12    return model.predict(text)
```

شکل ۷۷: تابع سنجش عواطف نظرات جدید



```
1 text = "This movie was trash. I didn't like it at all"
2 sentimentAnalysis(text, vectorizer,selector,scaler )
```

شکل ۷۸: کاربرد تابع تحلیل عواطف

و شکل (۷۹) خروجی این تابع را نمایش می‌دهد.

```
array([0], dtype=int64)
```

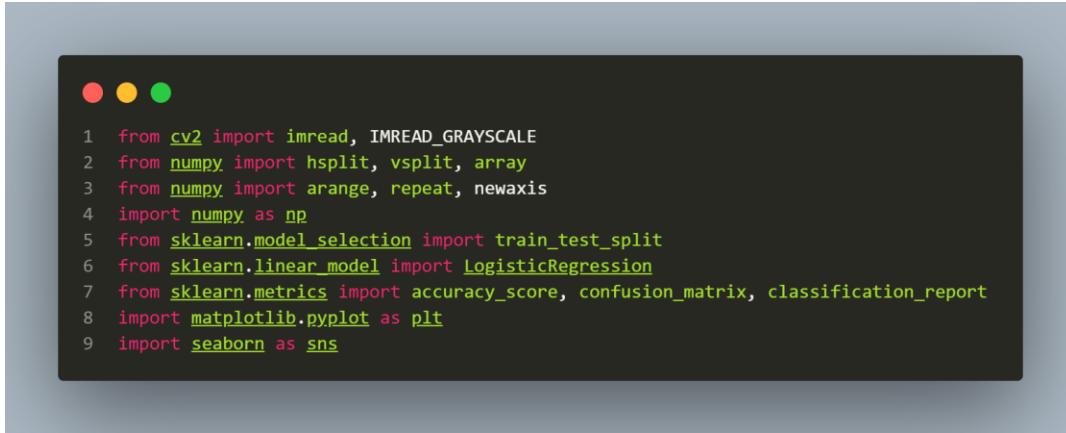
شکل ۷۹: خروجی تابع تحلیل عواطف

مشاهده می‌شود که خروجی تابع صفر است که به معنای بار منفی نظر پاس داده شده به تابع است.

هر دو مدل ایجاد شده، عملکرد خوبی در تشخیص جملات مثبت به نمایش نمی‌گذارند. درواقع باید نظر بسیار مثبت و پر از تعاریف و صفت‌های مثبت متعدد باشد تا مدل بتواند نظر را مشبت تشخیص دهد.

۴-پاسخ سوال چهارم

ابتدا کتابخانه‌هایی که در ادامه مورد نیاز هستند را ایمپورت می‌کنیم. شکل (۸۰) این کتابخانه‌ها را نمایش می‌دهد.

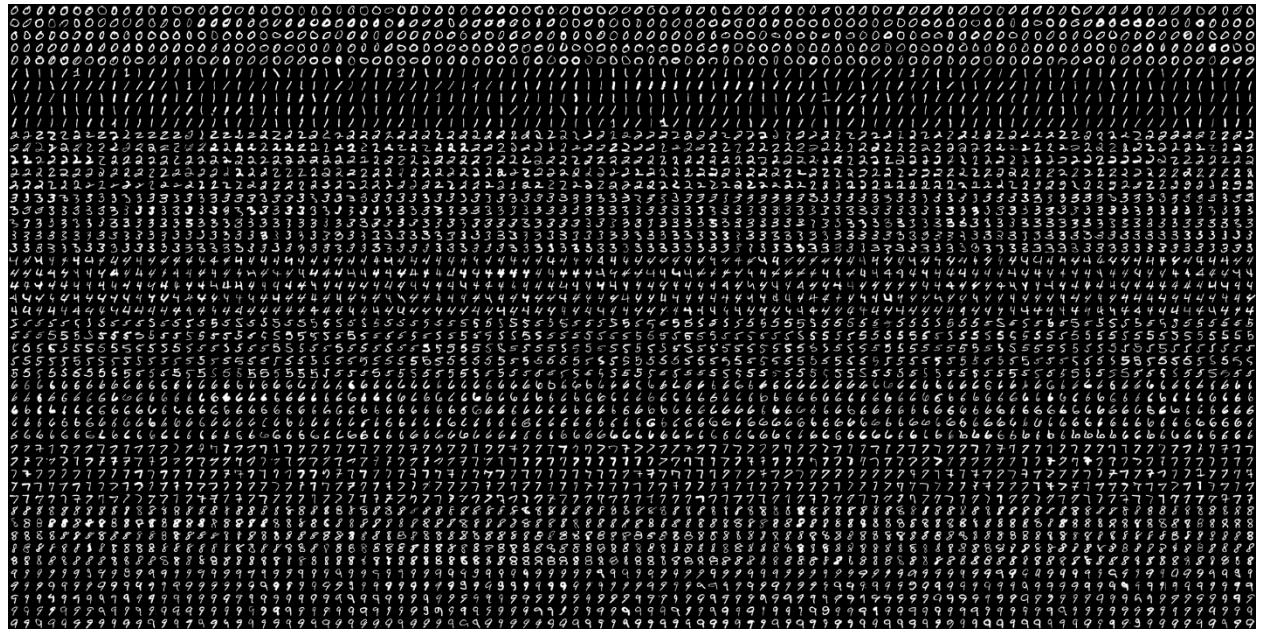


```
1 from cv2 import imread, IMREAD_GRAYSCALE
2 from numpy import hsplit, vsplit, array
3 from numpy import arange, repeat, newaxis
4 import numpy as np
5 from sklearn.model_selection import train_test_split
6 from sklearn.linear_model import LogisticRegression
7 from sklearn.metrics import accuracy_score, confusion_matrix, classification_report
8 import matplotlib.pyplot as plt
9 import seaborn as sns
```

شکل ۸۰: وارد کردن کتابخانه‌های مورد نیاز

۱-۴-وارد کردن و پیش‌پردازش داده‌ها

ابتدا فایل اصلی عکس‌ها را دانلود می‌کنیم. این فایل شامل همه عکس‌ها است که به ترتیب از صفر تا نه به صورت ردیفی قرار گرفته‌اند. شکل (۸۱) این فایل را نمایش می‌دهد.



شکل ۸۱: فایل digits.png

با کمی جست و جو متوجه می‌شویم که هر یک از اعداد، در یک مربع به ابعاد ۲۰ در ۲۰ پیکسل قرار گرفته‌اند. بنابراین با نوشتنتابعی می‌توانیم عکس را دریافت کنیم و آن را به عکس‌های کوچکتر تقسیم کنیم. شکل (۸۲) اینتابع را نمایش می‌دهد.

حال به توضیح اینتابع می‌پردازیم. اینتابع، مسیر عکس و ابعاد عکس را دریافت می‌کند. ابتدا با استفاده ازتابع imread() از کتابخانه OpenCV عکس را به صورت سیاه سفید می‌خوانیم و در آرگومان img ذخیره

```

1 def split_images(img_name, img_size):
2
3     img = imread(img_name, IMREAD_GRAYSCALE)
4
5     num_rows = img.shape[0] / img_size
6     num_cols = img.shape[1] / img_size
7
8     sub_imgs = [hsplit(row, num_cols) for row in vsplit(img, num_rows)]
9
10    return img, array(sub_imgs)

```

شکل ۸۲: تابع تقسیم کننده عکس بزرگ به عکس‌های کوچکتر

می‌کنیم. عکس به صورت یک آرایه دو بعدی در این متغیر ذخیره می‌شود. در مرحله بعدی با استفاده از `shape`. بعد از `shape` را فراخوانی می‌کنیم و این مقدار را تقسیم بر اندازه عکس‌های کوچک می‌کنیم تا تعداد سطرهای و تعداد ستون‌ها به دست بیاید.

`vsplit(img, num_rows)` تصویر را به صورت عمودی به قسمت‌هایی که توسط `num_rows` مشخص شده تقسیم می‌کند. هر برش هنوز عرض کامل تصویر اصلی را دارد.

`hsplit(row, num_cols)` سپس به صورت افقی هر یک از این برش‌های ردیف را به تصاویر کوچکتر که توسط `num_cols` مشخص شده است تقسیم می‌کند.

در `List Comprehension` هر برش عمودی (از `vsplit`) را پردازش می‌شود و برای هر برش، `hsplit` را برای ایجاد شبکه نهایی تصاویر فرعی اعمال می‌کند. نتیجه یک لیست تودرتو از تصاویر فرعی است که در آن هر تصویر فرعی یک آرایه دو بعدی از مقادیر پیکسل در مقیاس خاکستری است.

حال از یک تابع دیگر برای جداکردن و ایجاد لیبل‌ها استفاده می‌کنیم. این تابع داده‌های تصویر و برچسب‌های مربوطه را برای آموزش یادگیری ماشین آماده می‌کند. این تابع آرایه‌ای از تصاویر فرعی را به فرمت مسطح مناسب برای ورودی مدل تبدیل می‌کند و یک دنباله تکراری از برچسب‌ها برای این تصاویر ایجاد می‌کند. شکل (۸۳) این تابع را نمایش می‌دهد.

```

1 def split_data(img_size, sub_imgs):
2
3     train_imgs = sub_imgs.reshape(-1, img_size ** 2)
4     labels = orange(10)
5     train_labels = repeat(labels, train_imgs.shape[0] / labels.shape[0])[ :, newaxis]
6
7     return train_imgs, train_labels

```

شکل ۸۳: آماده سازی داده‌ها برای آموزش

خط اول کد آرایه `sub_imgs` را به یک آرایه دو بعدی تغییر شکل می‌دهد که در آن هر ردیف یک تصویر واحد را نشان می‌دهد. سپس تصاویر به بردارهایی با طول $2 \times \text{img_size}$ مسطح می‌شوند، به این معنی که هر تصویر به یک آرایه یک بعدی حاوی تمام مقادیر پیکسل تبدیل می‌شود.

سپس (10) یک آرایه از برچسب ها از ۰ تا ۹ ایجاد می کند. خط بعدی به اندازه کافی آرایه برچسب ها را تکرار می کند تا با تعداد تصاویر در train_imgs مطابقت داشته باشد. عبارت [0 : train_imgs.shape[0]] / labels.shape[0] محاسبه می کند که چند مجموعه کامل از ۰-۹ برچسب برای برچسب زدن هر تصویر یک بار مورد نیاز است. این فرض توزیع یکنواخت کلاس های تصویر را دارد.تابع تکرار هر برچسب را در برچسب ها به تعداد دفعات محاسبه شده تکرار می کند. آرایه را از یک بعدی به دو بعدی (بردار ستونی) تبدیل می کند.

۲-۴-آموزش رگرسیون لجستیک چند کلاسه

ابتدا دیتابست را به دو دسته آموزشی و آزمایشی تقسیم‌بندی می‌کنیم. هشتاد درصد داده‌ها را برای آموزش و بیست درصد را برای آزمایش کنار می‌گذاریم:

سپس از کلاس LogisticRegression یک شی می‌سازیم. آرگومان multi_class را برابر با 'multinomial' قرار می‌دهیم. این کار باعث می‌شود که رگرسیون به صورت چند کلاسه عمل کند. حالت دیگر One vs All است که در آن تلاش می‌کنیم یک لیبل به خصوص را از بین بقیه لیبل‌ها تشخیص دهیم که در این مسئله کاربرد ندارد. از حل کننده lbfgs و تعداد تکرار ۱۰۰۰ استفاده می‌کنیم.

سپس با استفاده از متدها fit() و predict() پاس داده‌های آموزشی مدل را آموزش می‌دهیم. سپس با استفاده از متدها accuracy_score() و predict() پاس دادن دیتابست تست، اجازه می‌دهیم مدل پیش‌بینی را انجام دهد. شکل (۸۴) نحوه انجام این کار را نمایش می‌دهد.

```

1 # Initialize the Logistic Regression model for multi-class classification
2 model = LogisticRegression(multi_class='multinomial', solver='lbfgs', max_iter=1000)
3 # Fit the model on the training data
4 model.fit(X_train, y_train)
5 # Predict the responses for the test dataset
6 y_pred = model.predict(X_test)

```

شکل ۸۴: آموزش و تست رگرسیون لجستیک

سپس با استفاده از تابع accuracy_score و پاس دادن مقادیر پیش‌بینی و مقادیر اصلی، دقت را محاسبه می‌کنیم. شکل (۸۵) نحوه انجام این کار را نمایش می‌دهد.

```

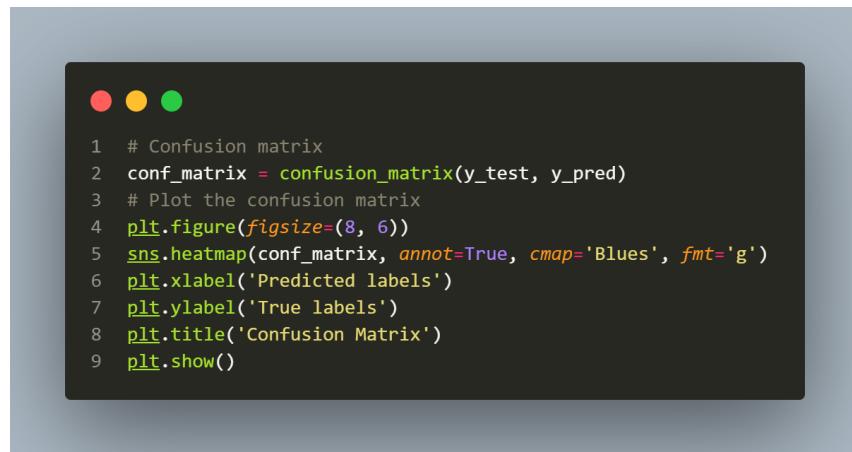
1 # Calculate accuracy
2 accuracy = accuracy_score(y_test, y_pred)
3 print(f"Accuracy: {accuracy:.2f}")

```

شکل ۸۵: محاسبه دقت

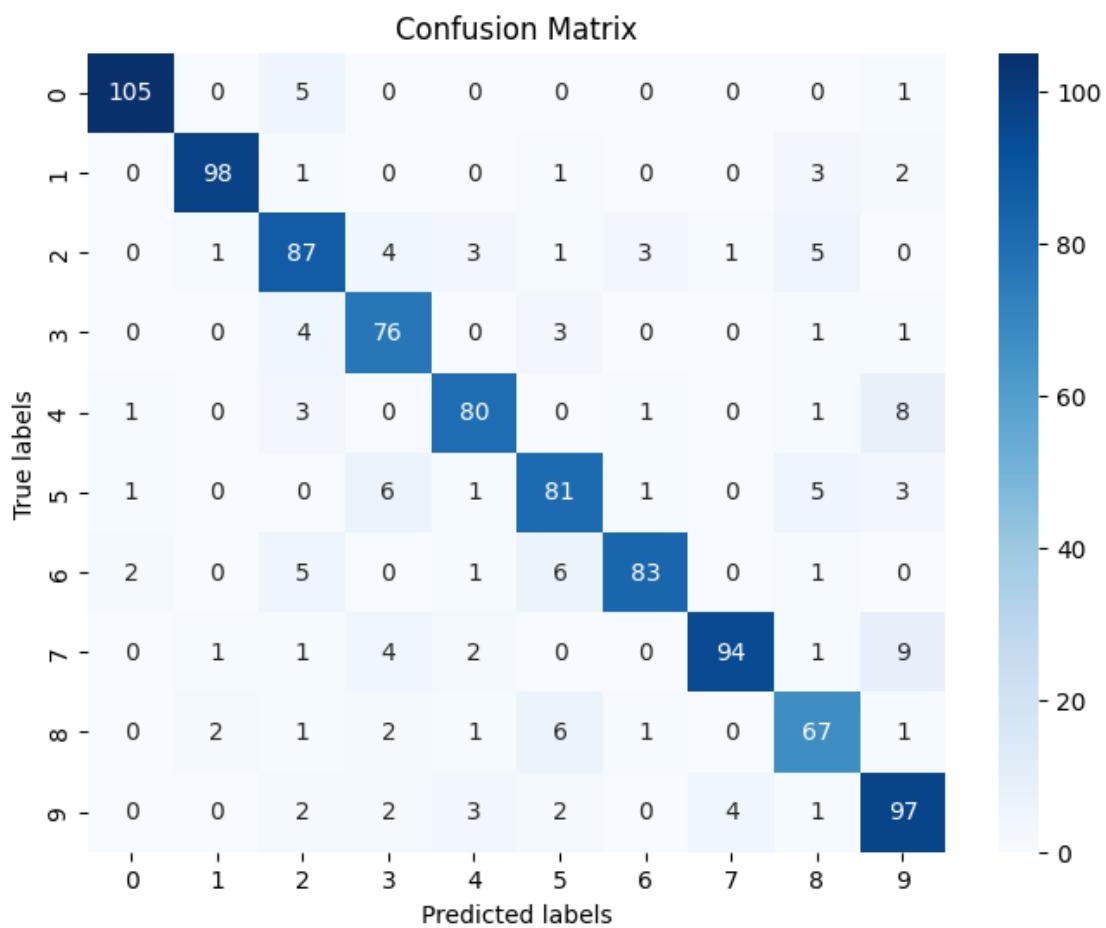
با استفاده از این روش دقت برایر با ۰.۷۸ به دست می‌آید که از دقت خواسته شده (۰.۷۸) بیشتر است.

سپس ماتریس درهمریختگی را محاسبه می‌کنیم. شکل (۸۶) نحوه انجام این کار را نشان می‌دهد.



شکل ۸۶: نحوه محاسبه ماتریس درهمریختگی

شکل (۸۷) ماتریس درهمریختگی به دست آمده را نشان می‌دهد.



شکل ۸۷: ماتریس درهمریختگی به دست آمده

با بررسی ماتریس درهم ریختگی مشاهده می شود که مدل نمی توان عدد هشت را به خوبی بقیه اعداد دسته بندی کند.