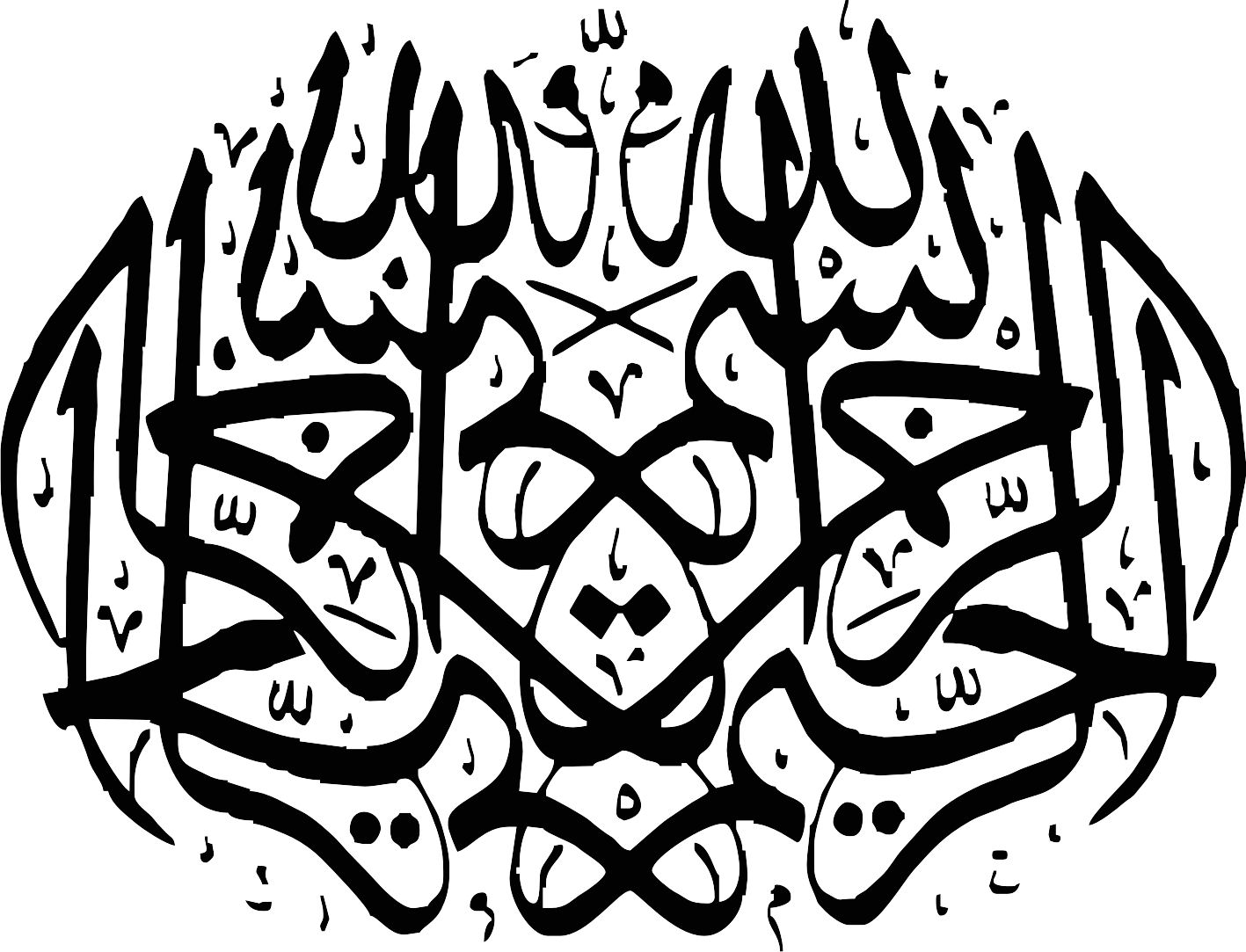


**گزارش تکلیف سوم درس یادگیری ماشین کاربردی**

**استاد درس: دکتر ناظرفرد**

تهیه کننده: سید نیما محمودیان

شماره دانشجویی: 402125005



فهرست مطالب

[1-پاسخ سوال اول 1](#_Toc165282316)

[1-1-لود کردن دیتاست، نمایش ده ردیف به صورت تصادفی، و بررسی کلی داده‌ها 1](#_Toc165282317)

[2-1-مصور سازی داده‌ها 6](#_Toc165282318)

[1-2-1-رسم هیستوگرام 6](#_Toc165282319)

[2-2-1-رسم و تفسیر نمودارهای جعبه‌ای 7](#_Toc165282320)

[3-2-1-رسم و تفسیر ماتریس همبستگی 8](#_Toc165282321)

[3-1-تشخیص داده‌های پرت 9](#_Toc165282322)

[4-1-پر کردن مقادیر خالی 10](#_Toc165282323)

[1-4-1-تعریف شی پیش‌پردازش برای پر کردن مقادیر خالی با Iterative Imputer 10](#_Toc165282324)

[2-4-1-تعریف شی پیش‌پردازش برای پر کردن مقادیر خالی با KNN 13](#_Toc165282325)

[5-1-انتخاب ویژگی 13](#_Toc165282326)

[6-1- آموزش مدل درخت تصمیم 14](#_Toc165282327)

[1-6-1-آموزش مدل با پارامترهای پیشفرض 14](#_Toc165282328)

[2-6-1-پیدا کردن هایپرپارامترهای بهینه با استفاده از GridSearch 15](#_Toc165282329)

[3-6-1-هرس کردن درخت و نمایش مجدد آن (بخش امتیازی) 17](#_Toc165282330)

[2-پاسخ سوال دوم 19](#_Toc165282331)

[1-2-آماده سازی داده‌ها 19](#_Toc165282332)

[2-2-حذف داده‌های پرت 21](#_Toc165282333)

[3-2-انتخاب ویژگی 22](#_Toc165282334)

[4-2-آموزش و تست رگرسیون لجستیک 23](#_Toc165282335)

[3-پاسخ سوال سوم 25](#_Toc165282336)

[1-3-پیش پردازش داده‌ها 27](#_Toc165282337)

[1-1-3-حذف تگ‌های <br /><br /> 27](#_Toc165282338)

[2-1-3-ساخت تابع پیش‌پردازش 27](#_Toc165282339)

[2-3-استفاده از CountVectorizer 28](#_Toc165282340)

[3-3- استفاده از TF-IDF 31](#_Toc165282341)

[4-3-ساخت تابعی که به نظرات جدید برچسب می‌دهد 31](#_Toc165282342)

[4-پاسخ سوال چهارم 33](#_Toc165282343)

[1-4-وارد کردن و پیش‌پردازش داده‌ها 33](#_Toc165282344)

[2-4-آموزش رگرسیون لجستیک چند کلاسه 35](#_Toc165282345)

فهرست شکل‌ها

[شکل 1: وارد کردن کتابخانه‌های مورد نیاز 1](#_Toc165282346)

[شکل 2: خواندن دیتاست 1](#_Toc165282347)

[شکل 3: حذف ستون load\_id 2](#_Toc165282348)

[شکل 4: کد مربوط به نمایش ده سطر تصادفی 2](#_Toc165282349)

[شکل 5: ده سطر تصادفی نمایش داده شده 2](#_Toc165282350)

[شکل 6: کد مربوط به نشان دادن تعداد مقادیر گمشده 2](#_Toc165282351)

[شکل 7: تعداد مقادیر خالی در هر سطر 3](#_Toc165282352)

[شکل 8: کد بررسی تعداد سطرهای تکراری 3](#_Toc165282353)

[شکل 9: کد بررسی تعداد صفرها در هر ستون 3](#_Toc165282354)

[شکل 10: تعداد مقادیر صفر در هر سطر 4](#_Toc165282355)

[شکل 11: بررسی تعداد و جمعیت موجود در سطرهای گسسته 4](#_Toc165282356)

[شکل 12: خروجی کد شکل 11 4](#_Toc165282357)

[شکل 13: خروجی اتریبیوت df.columns 5](#_Toc165282358)

[شکل 14: کد حذف فاصله از ستون‌ها 5](#_Toc165282359)

[شکل 15: جایگذاری صفر با np.nan 5](#_Toc165282360)

[شکل 16: جداسازی داده‌های عددی و دسته‌ای در دو لیست جدا 6](#_Toc165282361)

[شکل 17: کد مربوط به رسم هیستوگرام 6](#_Toc165282362)

[شکل 18: هیستوگرام‌های رسم شده 7](#_Toc165282363)

[شکل 19: کد رسم نمودار جعبه‌ای 7](#_Toc165282364)

[شکل 20: نمودارهای جعبه‌ای 8](#_Toc165282365)

[شکل 21: کد رسم ماتریس همبستگی 8](#_Toc165282366)

[شکل 22: ماتریس همبستگی رسم شده 9](#_Toc165282367)

[شکل 23: حذف داده‌های پرت با روش IQR 10](#_Toc165282368)

[شکل 24: ایجاد شی از کلاس پایپ‌لاین 10](#_Toc165282369)

[شکل 25: ایجاد شی از کلاس ColumnTransformer 11](#_Toc165282370)

[شکل 26: پیش پردازش داده‌های آموزش و تست 11](#_Toc165282371)

[شکل 27: Oversample کردن داده‌های آموزشی 12](#_Toc165282372)

[شکل 28: آموزش و تست مدل رندوم فارست 12](#_Toc165282373)

[شکل 29: محاسبه دقت 12](#_Toc165282374)

[شکل 30: پایپ‌لاین به روز شده 13](#_Toc165282375)

[شکل 31: پر کردن مقادیر خالی با KNN 13](#_Toc165282376)

[شکل 32: انتخاب ویژگی‌ها با استفاده از SelectKBest() 14](#_Toc165282377)

[شکل 33: پایپ‌لاین آموزش مدل پیشفرض 14](#_Toc165282378)

[شکل 34: گزارش به دست آمده از آموزش مدل پیشفرض 15](#_Toc165282379)

[شکل 35: استانداردسازی داده‌ها 15](#_Toc165282380)

[شکل 36: دیکشنری هایپرپارامترها 15](#_Toc165282381)

[شکل 37: پیداکردن ترکیب بهینه هایپرپارامترها با گرید سرچ 16](#_Toc165282382)

[شکل 38: نمایش پارامترهای بهینه 16](#_Toc165282383)

[شکل 39: درخت تصمیم با هایپرپارامترهای بهینه 16](#_Toc165282384)

[شکل 40: گزارش دسته بندی درخت تصمیم با هایپرپارامترهای بهینه 17](#_Toc165282385)

[شکل 41: کد مربوط به نمایش شکل درخت 17](#_Toc165282386)

[شکل 42: درخت به دست آمده از هایپرپارامترهای بهینه‌شده 18](#_Toc165282387)

[شکل 43: گزارش عملکرد درخت هرس شده 18](#_Toc165282388)

[شکل 44: درخت هرس شده 18](#_Toc165282389)

[شکل 45: وارد کردن کتابخانه‌های مورد نیاز 19](#_Toc165282390)

[شکل 46: خروجی متد .describe() 19](#_Toc165282391)

[شکل 47: ده سطر رندوم دیتافریم 20](#_Toc165282392)

[شکل 48: خروجی تعداد هر دسته در هر ستون 20](#_Toc165282393)

[شکل 49: پیدا کردن ؟ در دیتاست 21](#_Toc165282394)

[شکل 50: انکود کردن ستون‌های دودویی 21](#_Toc165282395)

[شکل 51: one-hot-encode کردن داده‌ها 22](#_Toc165282396)

[شکل 52: پیداکردن داده‌های پرت با استفاده از خوشه‌بندی 22](#_Toc165282397)

[شکل 53: پایپ‌لاین ساخته شده برای تحلیل مولفه اصلی 23](#_Toc165282398)

[شکل 54: گزارش به دست آمده از آموزش و تست مدل با 15 مولفه اصلی 23](#_Toc165282399)

[شکل 55: انتخاب 15 مولفه اصلی به عنوان ویژگی‌ها 24](#_Toc165282400)

[شکل 56: ساخت، آموزش و تست مدل رگرسیون لجستیک 24](#_Toc165282401)

[شکل 57: گزارش عملکرد رگرسیون لجستیک 24](#_Toc165282402)

[شکل 58: آموزش مدل با Cross Validation 25](#_Toc165282403)

[شکل 59: نتیجه Cross Validation 25](#_Toc165282404)

[شکل 60: وارد کردن کتابخانه‌های مورد نیاز 25](#_Toc165282405)

[شکل 61: تعداد ایمیل‌های با لیبل مثبت و منفی 26](#_Toc165282406)

[شکل 62: نمایش ده سطر رندوم 26](#_Toc165282407)

[شکل 63: جایگزینی لیبل‌ها با صفر و یک 26](#_Toc165282408)

[شکل 64: حذف تگ‌های <br /><br /> 27](#_Toc165282409)

[شکل 65: ساخت شی ریشه‌یاب و مجموعه ایست‌واژه‌ها 27](#_Toc165282410)

[شکل 66: تابع پیش‌پردازش متن 28](#_Toc165282411)

[شکل 67: پیش‌پردازش متن‌ها با استفاده از تابع تعریف شده 28](#_Toc165282412)

[شکل 68: بردار کردن متون با استفاده از CountVectorizer 28](#_Toc165282413)

[شکل 69: انتخاب 5000 کلمه مهم با استفاده از آزمون مربع کای 29](#_Toc165282414)

[شکل 70: استاندارد سازی و تبدیل ماتریس به ماتریس چگال 29](#_Toc165282415)

[شکل 71: پیش پردازش داده‌های تست 29](#_Toc165282416)

[شکل 72: تبدیل ماتریس‌های چگال به آرایه‌های خلوت 30](#_Toc165282417)

[شکل 73: آموزش و تست مدل بیز 30](#_Toc165282418)

[شکل 74: گزارش عملکرد مدل نایو بیز 30](#_Toc165282419)

[شکل 75: گزارش عملکرد مدل به دست آمده از دیتاست TF-IDF 31](#_Toc165282420)

[شکل 76: ذخیره کردن مدل برای استفاده در آینده 31](#_Toc165282421)

[شکل 77: تابع سنجش عواطف نظرات جدید 32](#_Toc165282422)

[شکل 78: کاربرد تابع تحلیل عواطف 32](#_Toc165282423)

[شکل 79: خروجی تابع تحلیل عواطف 32](#_Toc165282424)

[شکل 80: وارد کردن کتابخانه‌های مورد نیاز 33](#_Toc165282425)

[شکل 81: فایل digits.png 33](#_Toc165282426)

[شکل 82: تابع تقسیم کننده عکس بزرگ به عکس‌های کوچکتر 34](#_Toc165282427)

[شکل 83: آماده سازی داده‌ها برای آموزش 34](#_Toc165282428)

[شکل 84: آموزش و تست رگرسیون لجستیک 35](#_Toc165282429)

[شکل 85: محاسبه دقت 35](#_Toc165282430)

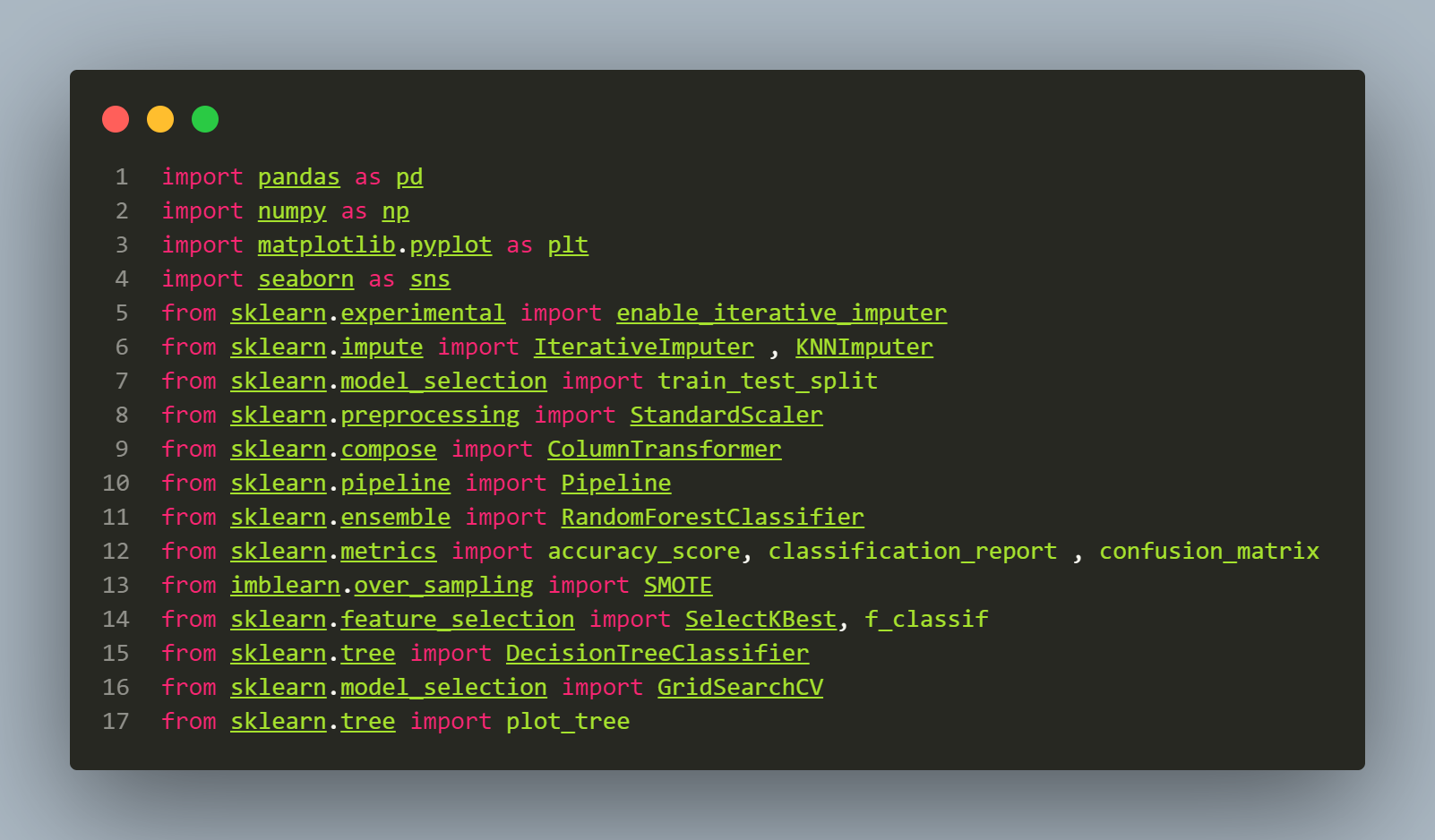
[شکل 86: نحوه محاسبه ماتریس درهم‌ریختگی 36](#_Toc165282431)

[شکل 87: ماتریس درهم‌ریختگی به دست آمده 36](#_Toc165282432)

# 1-پاسخ سوال اول

## 1-1-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 بازتولید نتایج مشابه تعریف می‌شود. شکل (2)، خواندن دیتاست و نمایش آن را نشان می‌دهد.

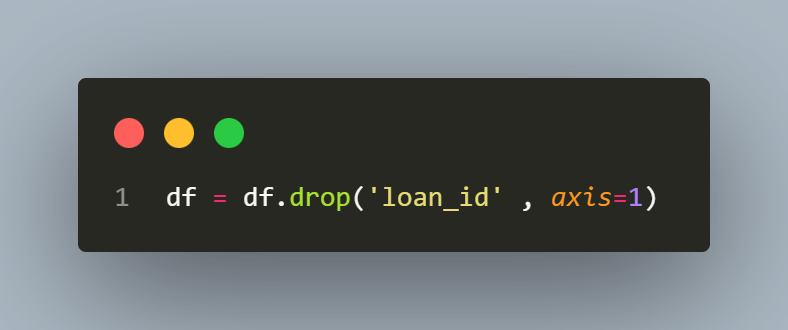


شکل 1: وارد کردن کتابخانه‌های مورد نیاز



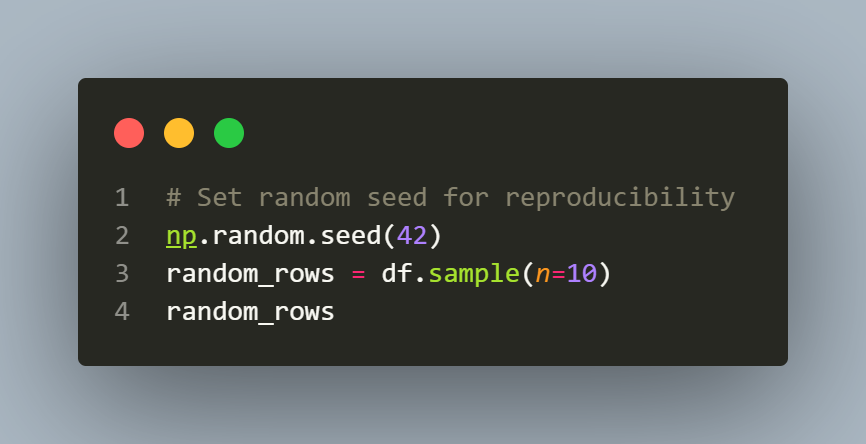
شکل 2: خواندن دیتاست

مشاهده می‌شود که ستون loan\_id کاملاً متناظر با اندیسی است که پنداز به طور خودکار به دیتافریم داده است. پس این ستون را حذف می‌کنیم. شکل (3) نحوه حذف این ستون با استفاده از متد .drop() را نمایش می‌دهد.



شکل 3: حذف ستون load\_id

شکل (4) نشان دادن ده ردیف رندوم را نمایش می‌دهد. این کار را با استفاده از متد .sample() انجام می‌شود.



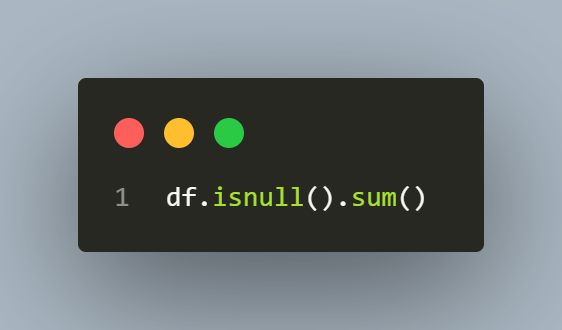
شکل 4: کد مربوط به نمایش ده سطر تصادفی

همچنین شکل (5) ده سطر تصادفی نمایش داده شده را نشان می‌دهد.



شکل 5: ده سطر تصادفی نمایش داده شده

در ادامه با استفاده از متد info() نگاهی کلی به ستون‌های باقی‌مانده می‌اندازیم. سپس با استفاده از method chaining نمایش داده شده در شکل (5) بررسی می‌کنیم که در هر ستون در مجموع چه تعداد مقادیر تعریف نشده وجود دارد.



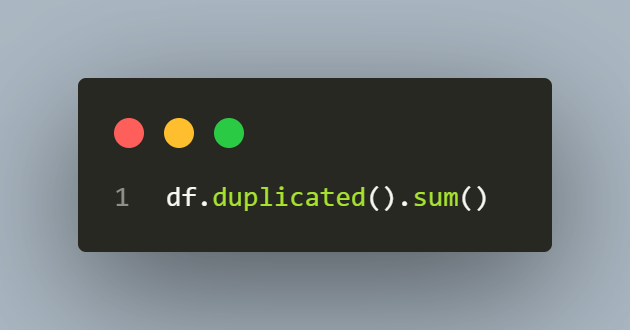
شکل 6: کد مربوط به نشان دادن تعداد مقادیر گمشده

با توجه به شکل (7) مشاهده می‌شود که هیچ ستونی، خانه‌ی بدون مقدار ندارد.

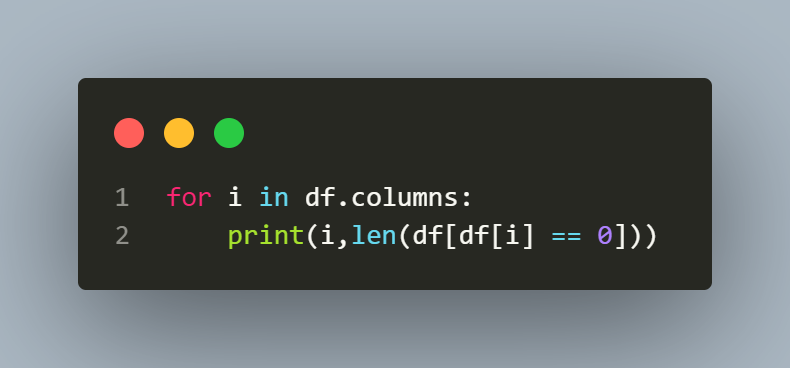


شکل 7: تعداد مقادیر خالی در هر سطر

در ادامه سه مورد را بررسی می‌کنیم: اول اینکه چه تعداد سطر تکراری داریم. شکل (8) 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 (9) نحوه انجام این کار را نمایش می‌دهد.



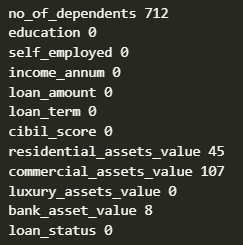
شکل 8: کد بررسی تعداد سطرهای تکراری



شکل 9: کد بررسی تعداد صفرها در هر ستون

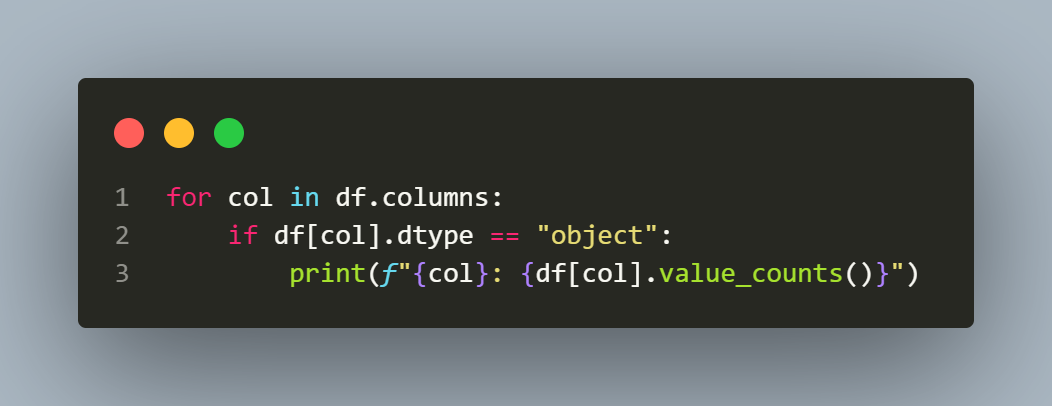
با بررسی خروجی این دو کد، متوجه می‌شویم که هیچ سطر تکراری در دیتافریم وجود ندارد. ولی مشاهده می‌شود که سه تا از ستون‌ها دارای مقادیر صفر هستند. شکل (10) خروجی کد شکل 9 را نمایش می‌دهد.

مشاهده می‌شود که ستون‌های no\_of\_dependents و residental\_assets\_value و bank\_asset\_value 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.

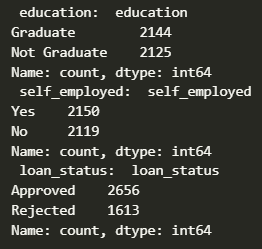


شکل 10: تعداد مقادیر صفر در هر سطر

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 دیتاتایپ object دارند، با استفاده از متد .value\_counts() تعداد هر دسته در هر ستون را نمایش می‌دهیم. شکل (11) کدی که این کار را انجام می‌دهد نمایش داده و خروجی در شکل (12) قرار گرفته.

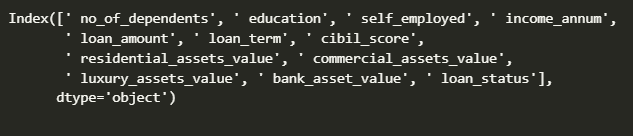


شکل 11: بررسی تعداد و جمعیت موجود در سطرهای گسسته



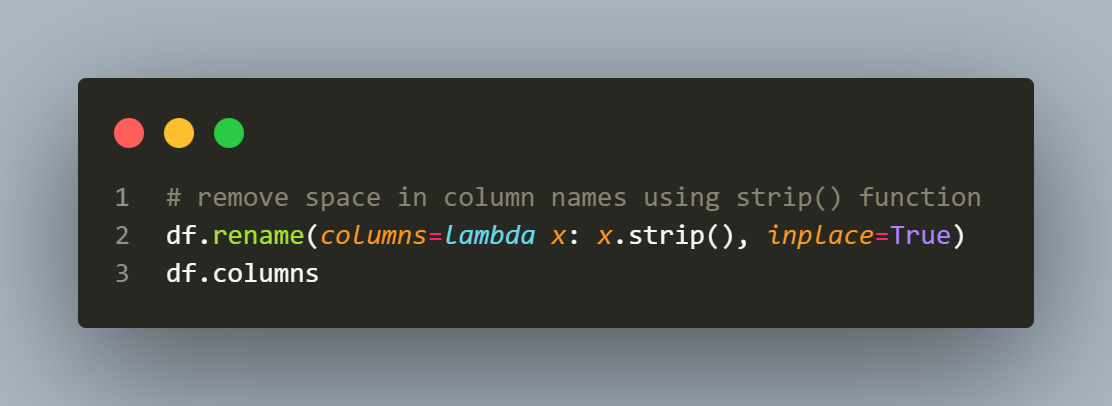
شکل 12: خروجی کد شکل 11

حال با نمایش ستون‌های دیتافریم، متوجه می‌شویم که در برخی از ستون‌ها، فاصله خالی وجود دارد. شکل (13) خروجی اتریبیوت .columns را نمایش می‌دهد.



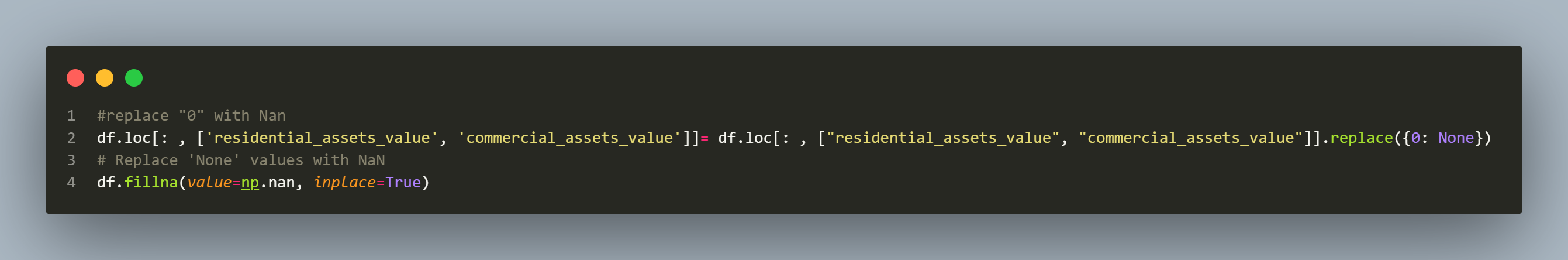
شکل 13: خروجی اتریبیوت df.columns

سپس با استفاده از متد .rename() 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 متد .strip() فاصله‌های اضافی را حذف می‌کنیم. شکل (14) این کد را نمایش می‌دهد.



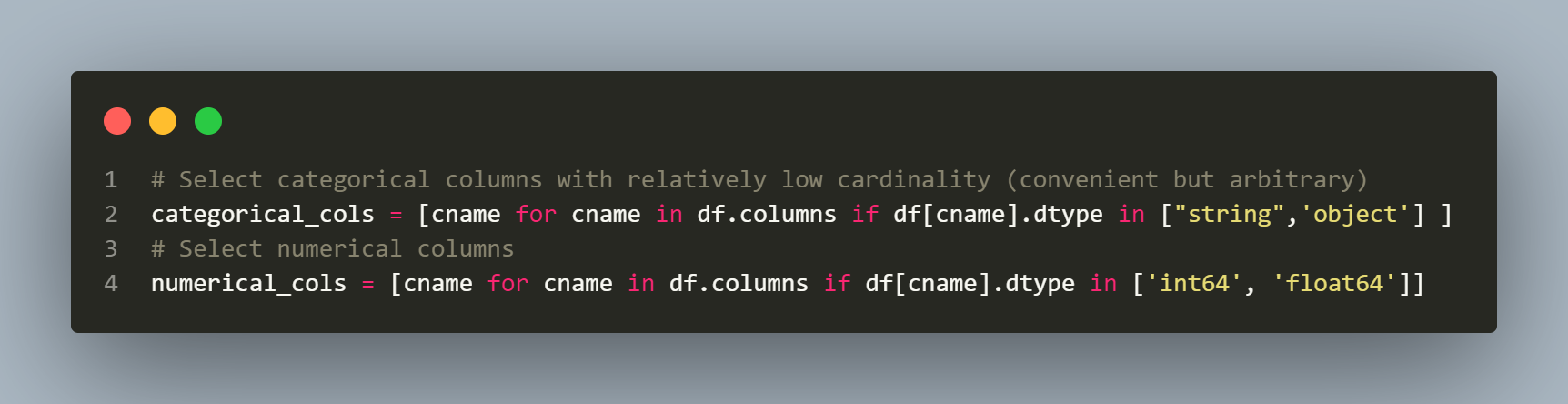
شکل 14: کد حذف فاصله از ستون‌ها

سپس مقادیر صفر در ستون‌های residental\_assets\_value و bank\_asset\_value را با Nan جایگزین می‌کنیم. ابتدا با استفاده از متد .replace() صفر‌ها را با None جایگزین می‌کنیم و سپس با استفاده از متد .fillna() و مقدار np.nan به عنوان آرگومان value مقادیر None را با np.nan جایگزین می‌کنیم. شکل (15) نحوه انجام این کار را نمایش می‌دهد.



شکل 15: جایگذاری 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 (16) نحوه انجام این کار را نمایش می‌دهد.



شکل 16: جداسازی داده‌های عددی و دسته‌ای در دو لیست جدا

## 2-1-مصور سازی داده‌ها

### 1-2-1-رسم هیستوگرام

شکل 17 نحوه رسم هیستوگرام را نمایش می‌دهد. num\_rows و num\_cols ابعاد شبکه فرعی را مشخص می کنند. در این مورد، یک شبکه سه در سه ایجاد می کند، بنابراین می توان تا 9 نمودار فرعی را در خود جای داد. تابع plt.subplots برای تولید یک شکل و مجموعه ای از نمودارهای فرعی (محور) استفاده می شود. (20,10) figsize اندازه کل شکل (20 اینچ عرض و 10 اینچ ارتفاع) را مشخص می کند.

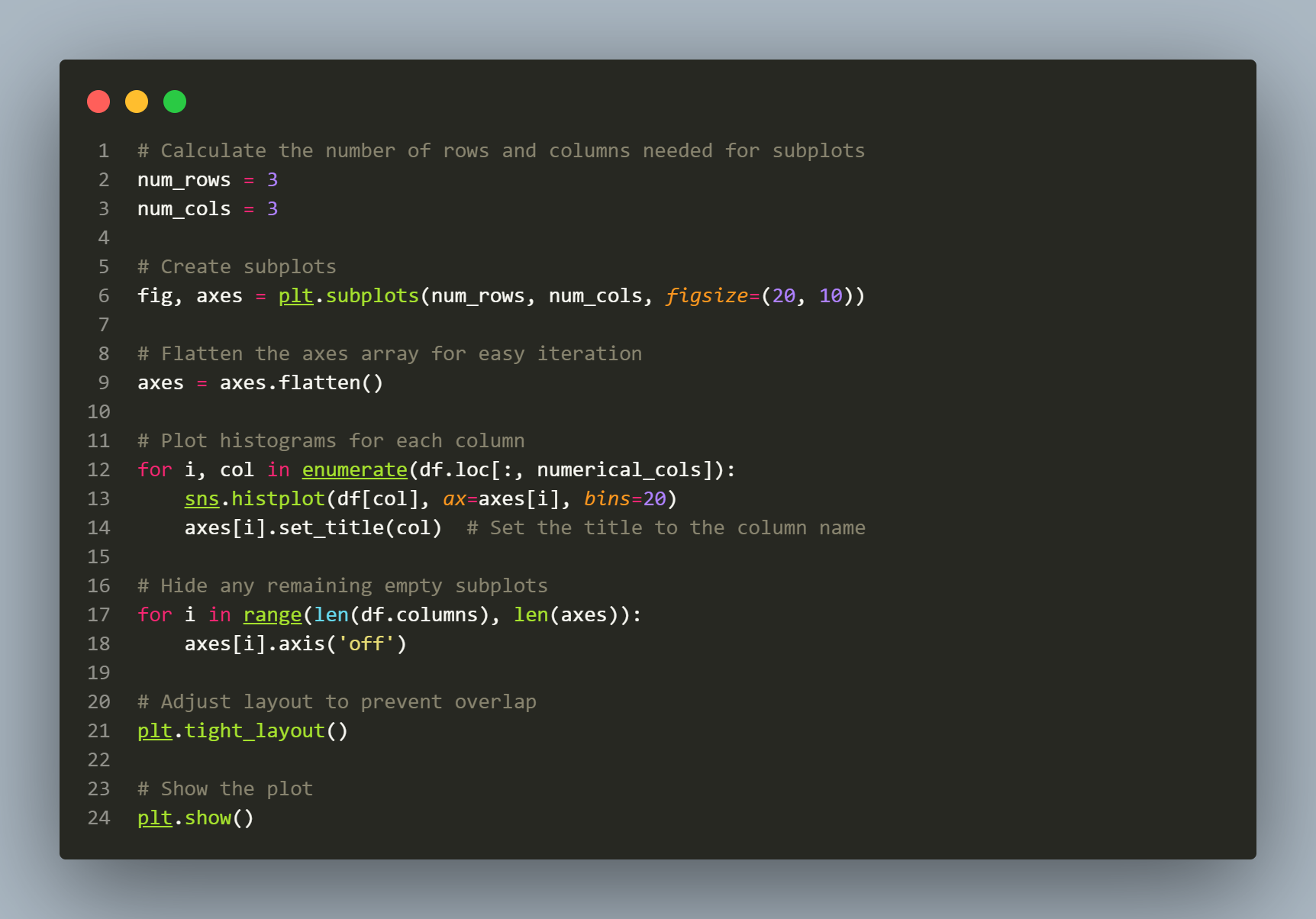
سپس حلقه روی هر ستون مشخص شده در لیست numerical\_cols از df تکرار می شود. برای هر ستون:

sns.histplot یک هیستوگرام از ستون ایجاد می کند و آن را بر روی نمودار فرعی مربوطه ترسیم می کند bins=20 مشخص می کند که هر هیستوگرام باید 20 میله داشته باشد. سپس عنوان هر نمودار بالای آن مشخص می‌شود.

سپس لوپ بعدی هر فضای خالی که در آن نمودار وجود ندارد را حذف می‌کند.

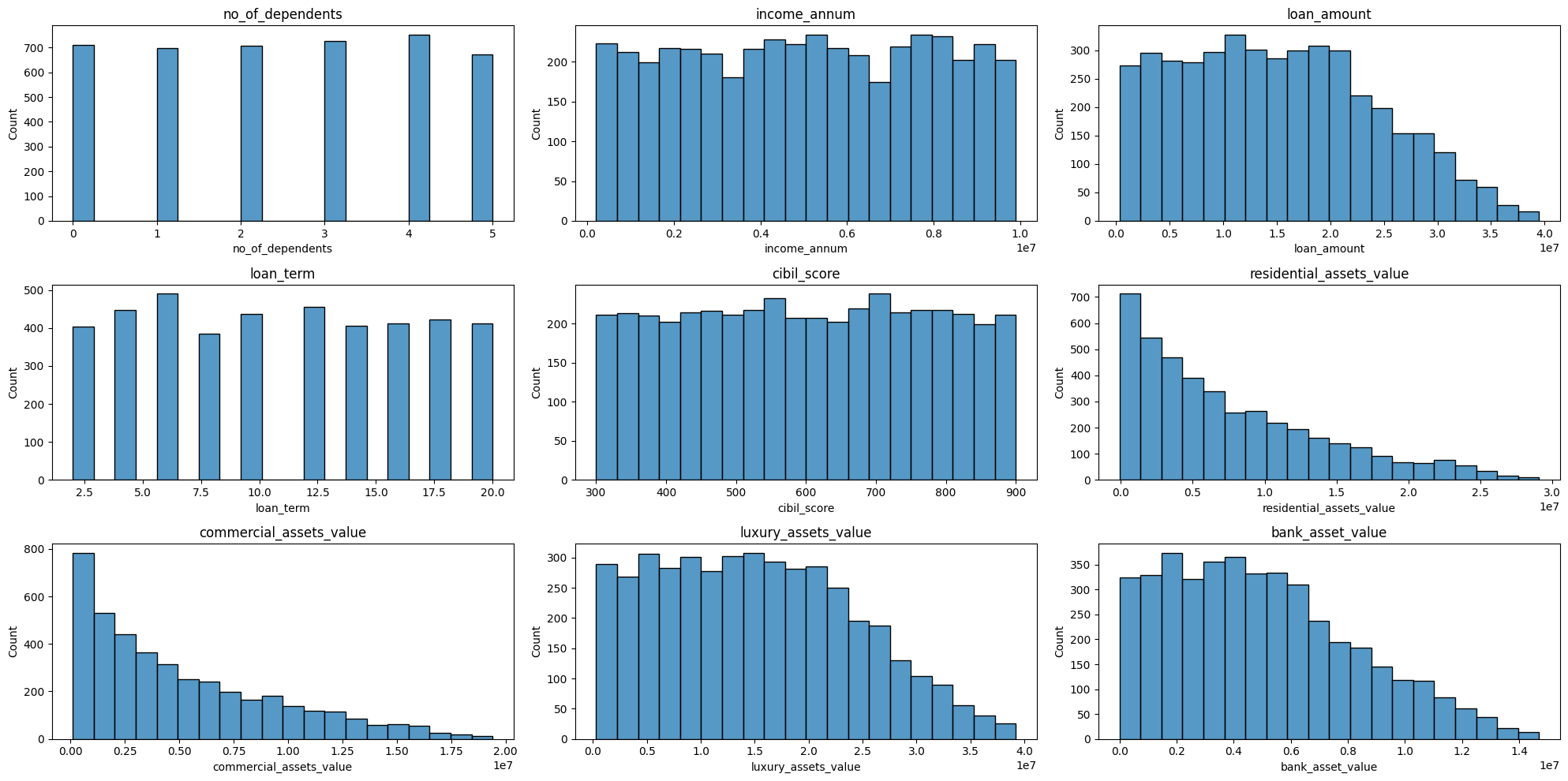
()plt.tight\_layout، نمودارهای فرعی و برچسب ها را طوری تنظیم می کند که به خوبی در ناحیه شکل قرار گیرند و روی هم قرار نگیرند.

plt.show () شکل کامل را با تمام نمودارهای فرعی نمایش می دهد.



شکل 17: کد مربوط به رسم هیستوگرام

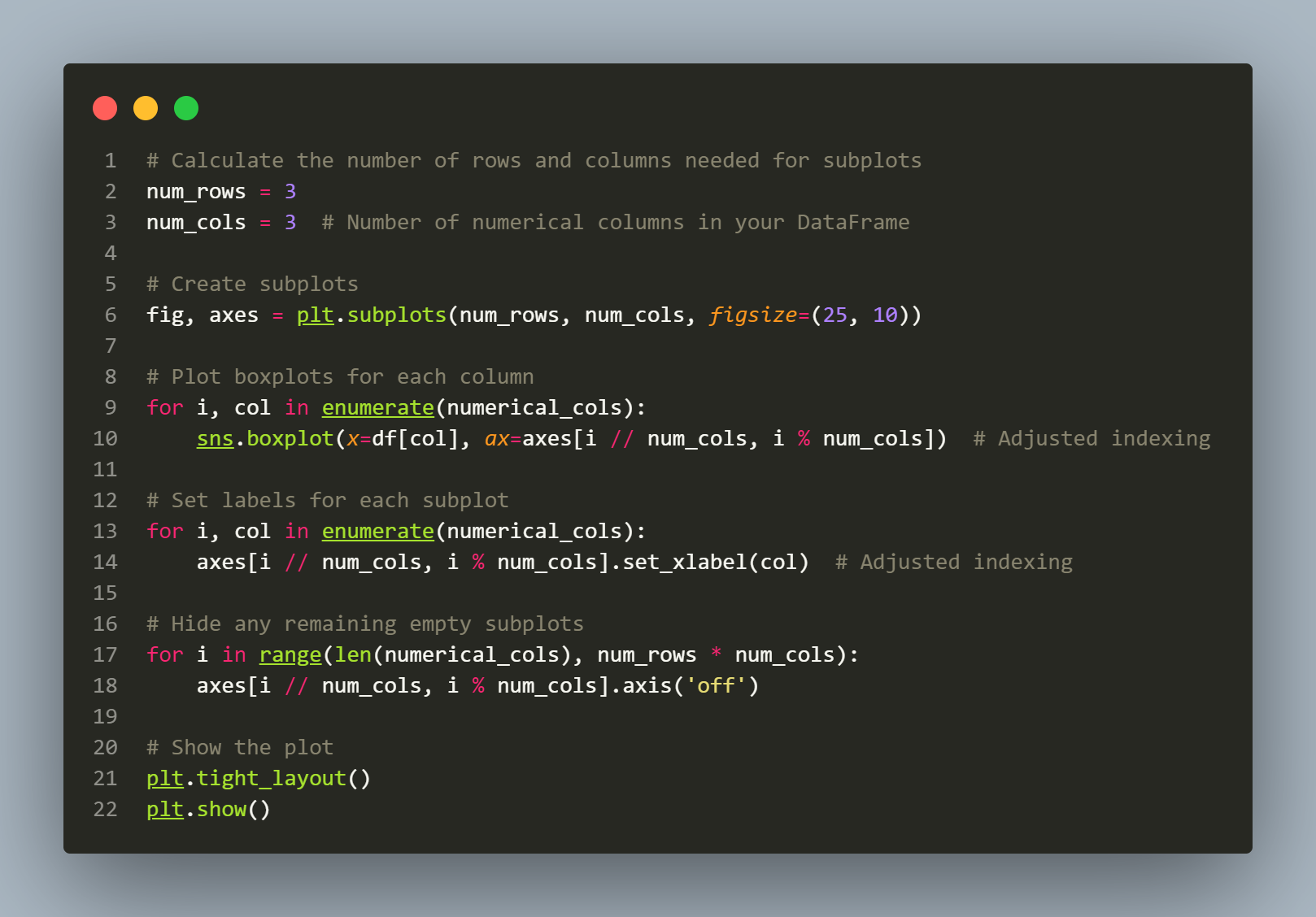
همچنین شکل (18) هیستوگرام‌های رسم شده را نمایش می‌دهد.



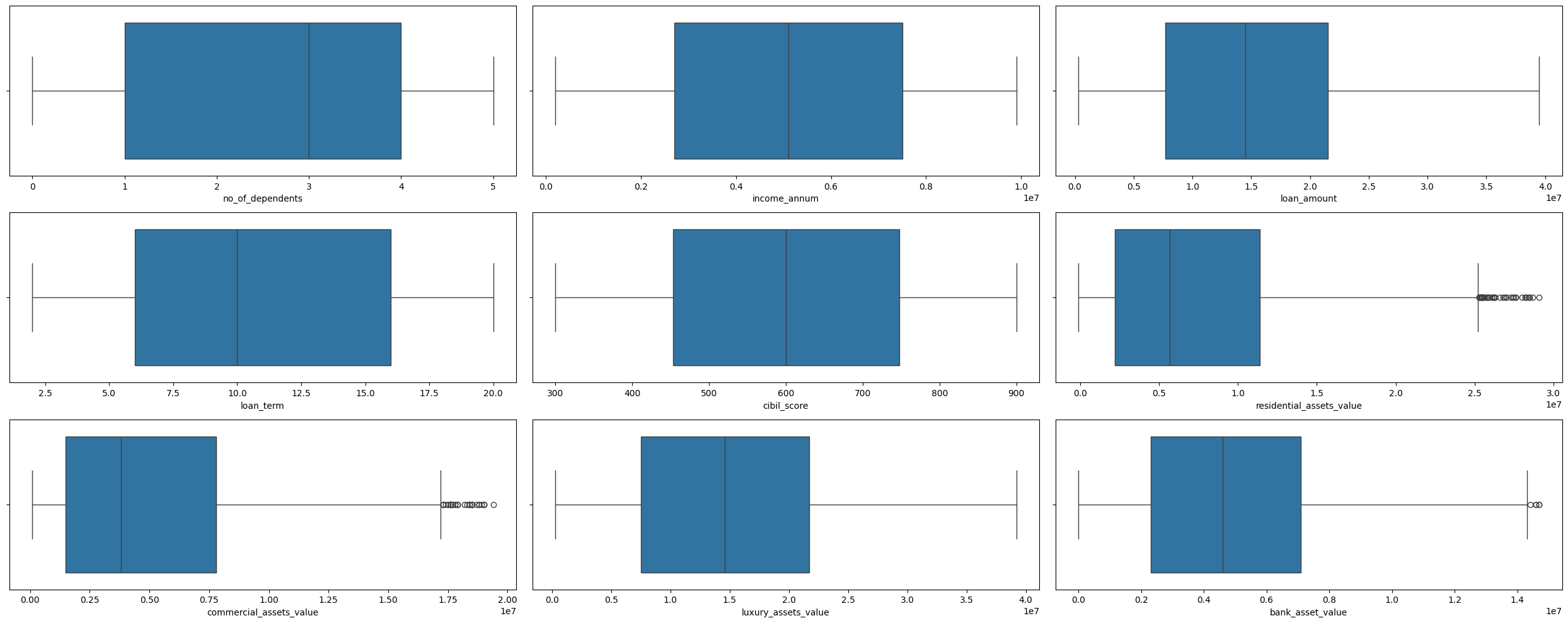
شکل 18: هیستوگرام‌های رسم شده

### 2-2-1-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 (19) کد رسم کننده این نمودارها را نمایش می‌دهد. همچنین شکل (20) خروجی این کد را نمایش می‌دهد.



شکل 19: کد رسم نمودار جعبه‌ای



شکل 20: نمودارهای جعبه‌ای

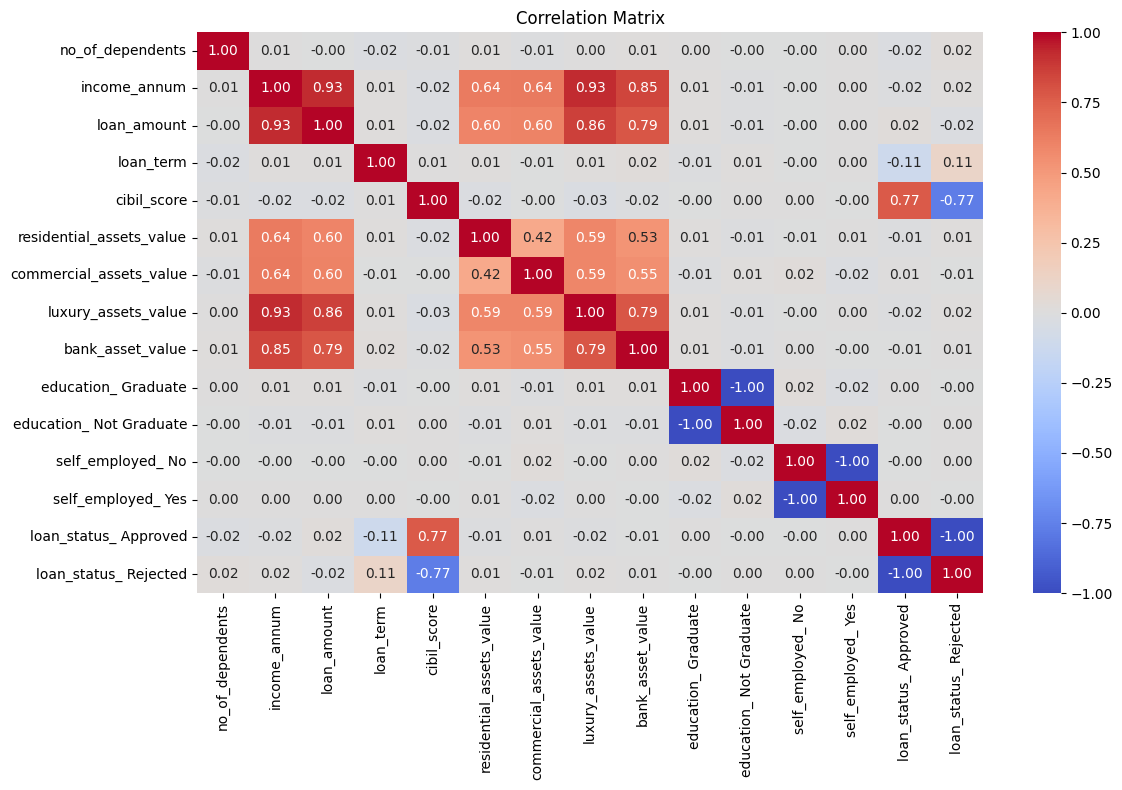
### 3-2-1-رسم و تفسیر ماتریس همبستگی

با استفاده از متد corr() ماتریس همبستگی را ایجاد می‌کنیم. سپس با استفاده از کتابخانه seaborn و تابع heatmap() ماتریس ایجاد شده را رسم می‌کنیم. شکل (21) نحوه رسم این ماتریس را نمایش می‌دهد.



شکل 21: کد رسم ماتریس همبستگی

همچنین شکل (22) ماتریس رسم شده را نمایش می‌دهد.

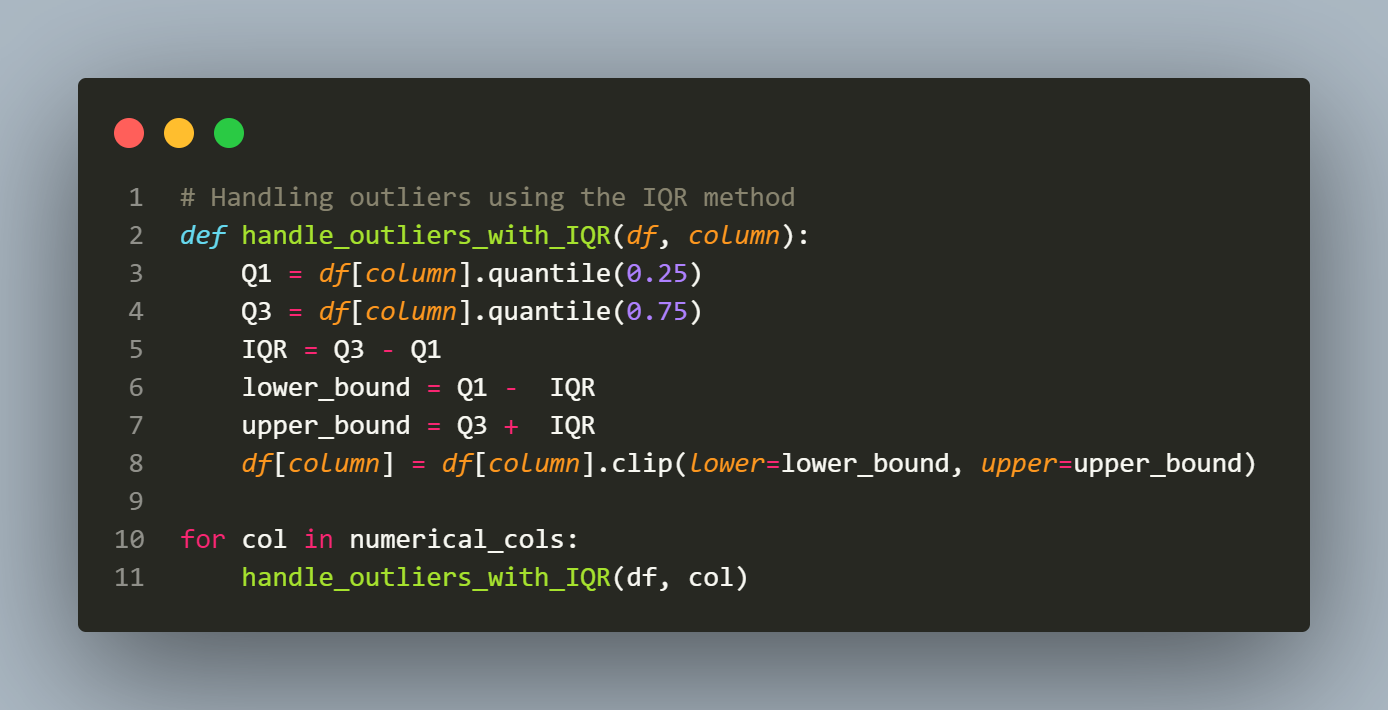


شکل 22: ماتریس همبستگی رسم شده

## 3-1-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 1.5 برابر دامنه به علاوه چارک سوم و داده‌های کوچک‌تر از 1.5 برابر دامنه منهای چارک اول هستند را حذف می‌کنیم. برای این منظور از تابع .clip() و تعریف دو مقدار توضیح داده شده به عنوان حد بالا و حد پایین، تنها داده‌هایی را نگه می‌داریم که بین دو حد تعریف شده هستند. شکل (23) نحوه انجام این کار را نمایش می‌دهد.

موردی که پیش می‌آید این است که مشاهده می‌شود که هیچ‌یک از داده‌ها در هر ستون، خارج از حدود تعیین شده قرار نمی‌گیرند. سپس با تنگ‌تر کردن محدود‌ه‌ها به یک برابر دامنه میان چارکی دوباره امتحان می‌کنیم. مشاهده می‌شود که مجدداً داده‌ای خارج از این حدود نیز قرار نمی‌گیرد. در نتیجه می‌توان متوجه شد که داده‌ پرتی در این دیتاست نداریم.



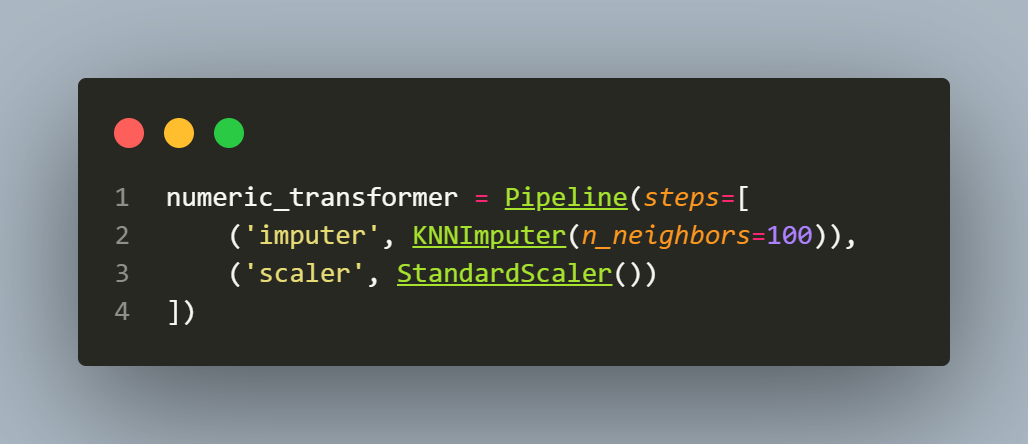
شکل 23: حذف داده‌های پرت با روش IQR

## 4-1-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

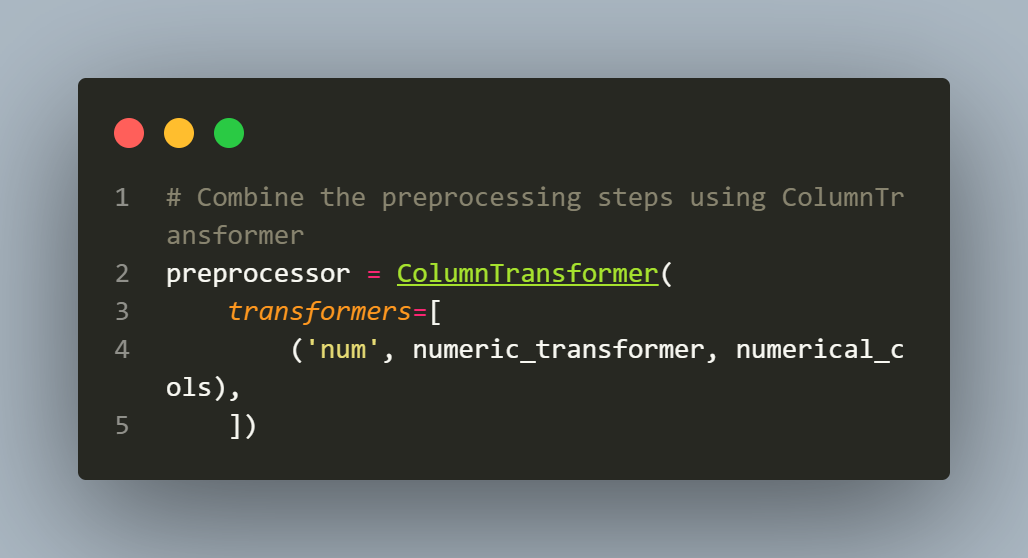
### 1-4-1-تعریف شی پیش‌پردازش برای پر کردن مقادیر خالی با Iterative Imputer

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 از تابع IterativeImputer() پر می‌کند و سپس داده‌ها را استاندارد سازی می‌کند. شکل (24) کد مربوط به این شی را نمایش می‌دهد.



شکل 24: ایجاد شی از کلاس پایپ‌لاین

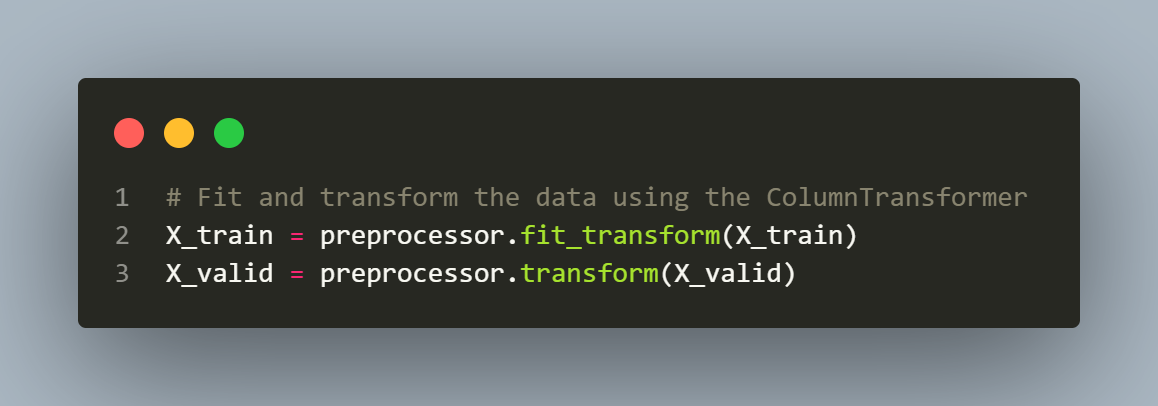
سپس یک شی از کلاس ColumnTransformer() ایجاد می‌کنیم در یک تاپل به ترتیب نام ترنسفورمر، شی ساخته شده از پایپ‌لاین، و ستون‌هایی که می‌خواهیم تغییرات روی آنها اعمال شوند را قرار می‌دهیم. شکل (25) ترنسفورمر ایجاد شده را نمایش می‌دهد.



شکل 25: ایجاد شی از کلاس ColumnTransformer

در ادامه از شی preprocessor متد .fit\_transform() را فراخوانی می‌کنیم و دیتاست آموزشی را به آن پاس می‌دهیم. این متد پارامتر دیتاستی 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.

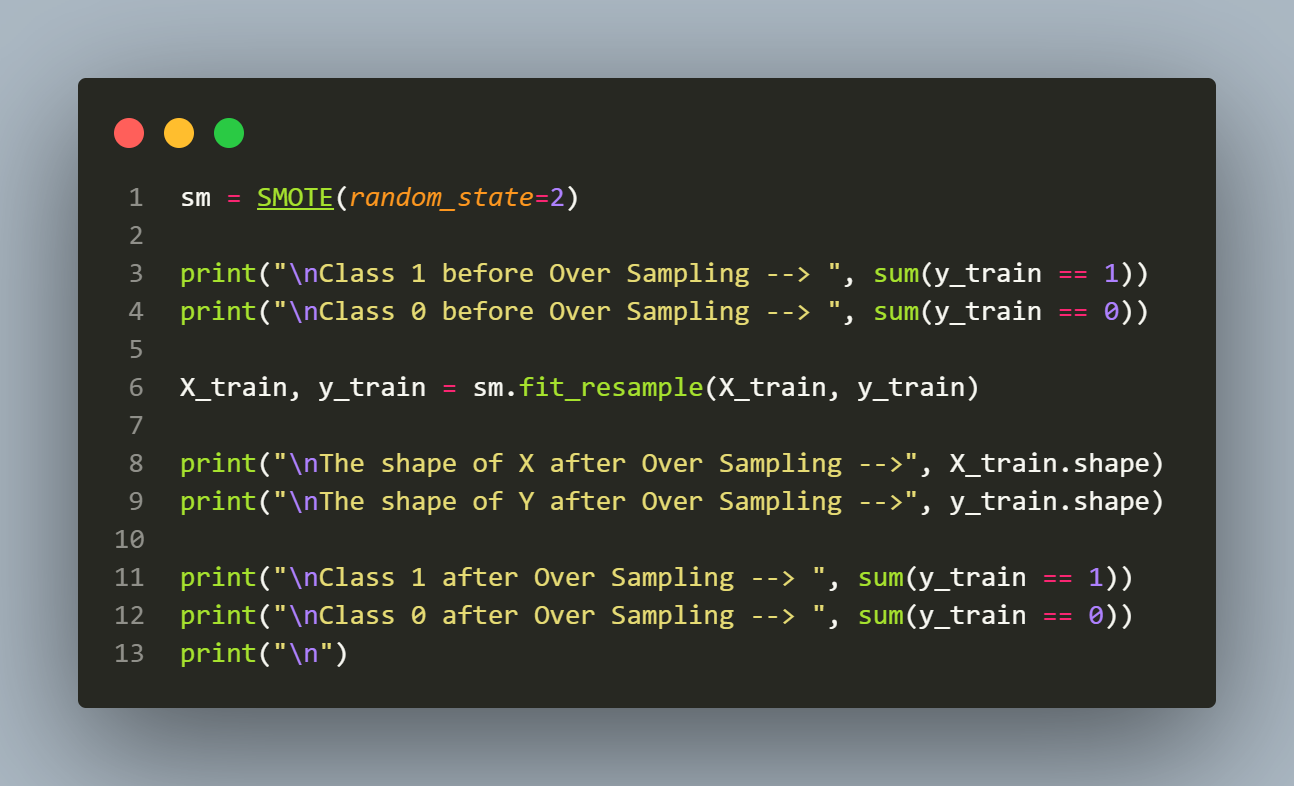
سپس برای پیش‌پردازش داده‌های تست، از متد transform() شی استفاده می‌کنیم. دلیل استفاده از این متد این است که این متد با استفاده از پارامترهای یادگرفته شده از داده‌های آموزشی، عملیات را روی داده‌های تستی انجام می‌دهد. توجه شود که اینجا از متد قبلی استفاده نمی‌شود زیرا فرض بر این است که پارامترهای داده‌های تست را نداریم و باید با استفاده از برآوردی که در داده‌های آموزشی انجام دادیم، عملیات‌ها را روی داده‌های تست انجام دهیم. عدم رعایت این مورد، موجب ایجاد نشت داده می‌شود. شکل (26) نحوه عملکرد این دو متد را نمایش می‌دهد.



شکل 26: پیش پردازش داده‌های آموزش و تست

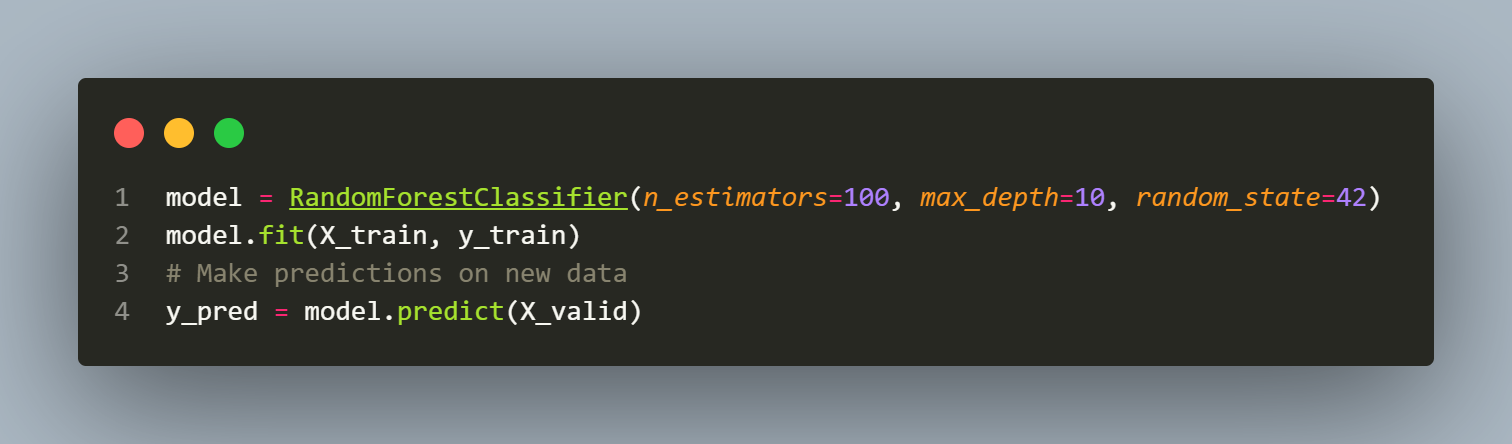
از آنجایی که دیتاست imbalance است، در اولین اقدام باید تعداد لیبل‌های صفر و تعداد لیبل‌های یک را در دیتاست آموزشی برابر کرد. برای این منظور از روش SMOTE استفاده می‌کنیم. این روش یکی از روش‌های oversampling است که از لیبلی که تعداد کمتری دارد داده‌های مصنوعی تولید می‌کند تا تعداد هر دو لیبل با هم برابر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

برای این منظور از کلاس SMOTE یک شی به نام sm می‌سازیم. سپس با استفاده از متد fit\_resample() داده‌های آموزشی را بالانس می‌کنیم. شکل (27) نحوه انجام این کار را نمایش می‌دهد.



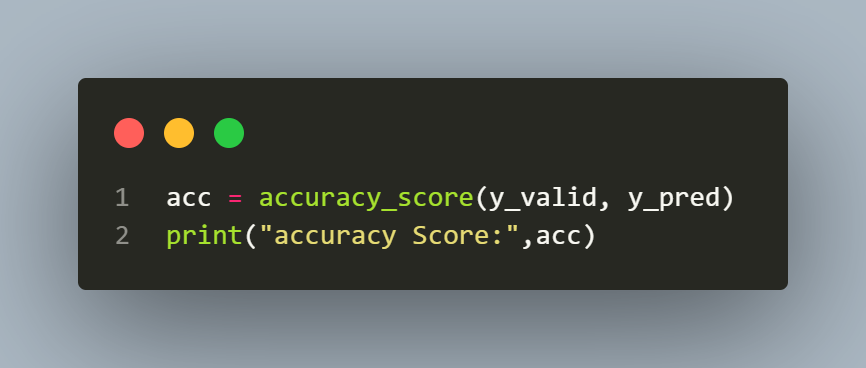
شکل 27: Oversample کردن داده‌های آموزشی

سپس از کلاس RandomForestClassifier یک شی به نام مدل می‌سازیم. سپس به متد .fit() از این مدل، دیتاست آموزشی را پاس می‌دهیم. و با استفاده از متد .predict() و پاس دادن دیتاست تست، پیش بینی‌های مدل را در متغیر y\_pred ذخیره می‌کنیم. شکل (28) نحوه انجام این کار را نمایش می‌دهد.



شکل 28: 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 (29) نحوه محاسبه دقت را نمایش می‌دهد.

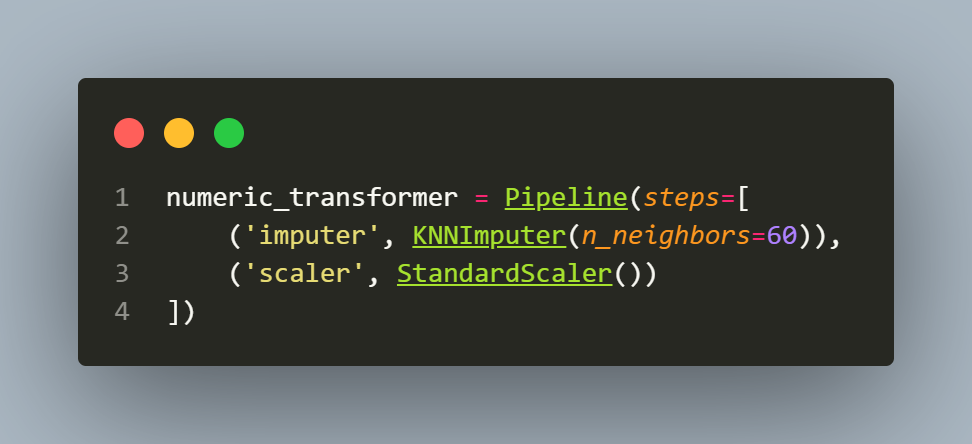


شکل 29: محاسبه دقت

با استفاده از IterativeImputer میزان دقت برابر با 0.9748 به دست می‌آید.

### 2-4-1-تعریف شی پیش‌پردازش برای پر کردن مقادیر خالی با KNN

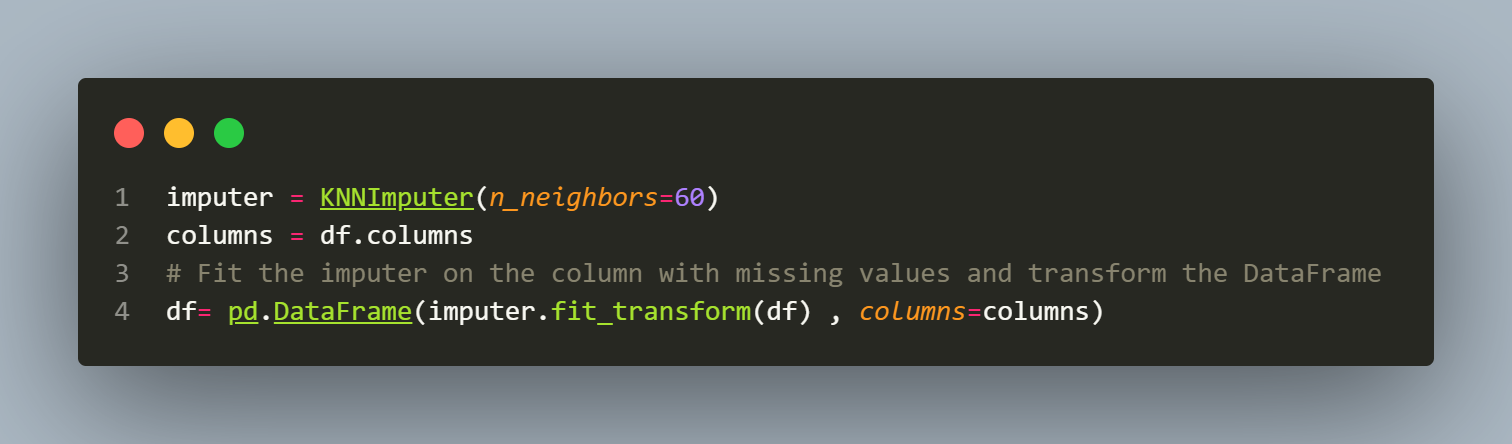
حال شی تعریف شده برای پیش‌پردازش را به نحوی تغییر می‌دهیم که مقادیر خالی را با KNN پر کند. برای این منظور، در شی پایپ‌لاین، از کلاس KNNImputer() به جای کلاس IterativeImputer() با آرگومان n\_neighbors=60 را پاس می‌دهیم. شکل (30) پایپ‌لاین به روز شده را نمایش می‌دهد.



شکل 30: 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 0.9754 خواهد شداز 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 متد .fit\_transform() را به همراه نام ستون‌های دیتافریم را به تابع dataframe() می‌دهیم و دیتافریم خروجی را در متغیر df ذخیره می‌کنیم. شکل (31) نحوه انجام این کار را نمایش می‌دهد.

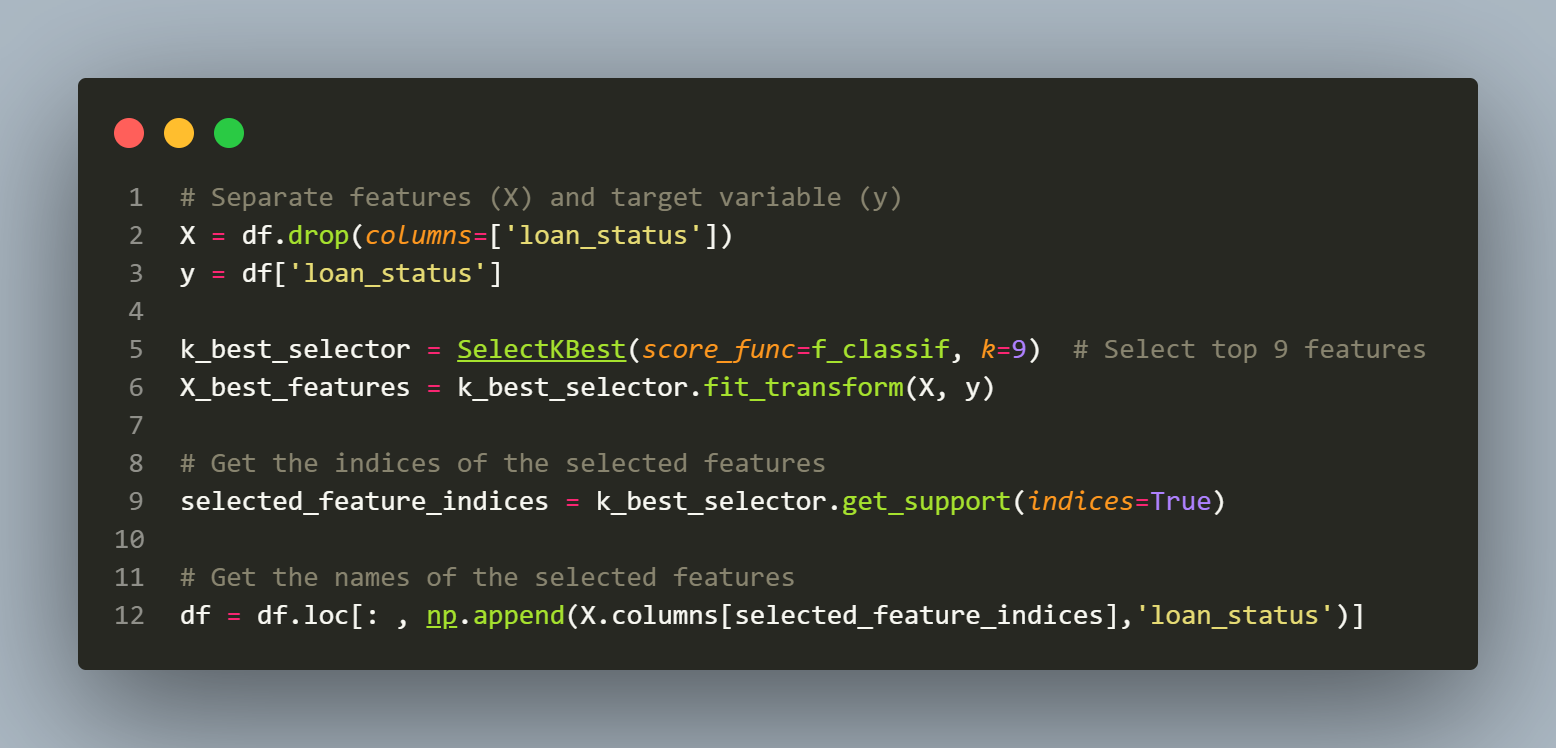


شکل 31: پر کردن مقادیر خالی با KNN

در ادامه به انتخاب ویژگی می‌پردازیم.

## 5-1-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 9 قرار می‌دهیم همچنین تست آماری را f\_classif قرار می‌دهیم. سپس در متد .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 متد .get\_support() نام ستون‌های انتخاب شده را در متغیری ذخیره می‌کنیم و با استفاده از آن، ستون‌های مربوطه از دیتافریم را انتخاب می‌کنیم. شکل (32) نحوه انجام این کار را نمایش می‌دهد.



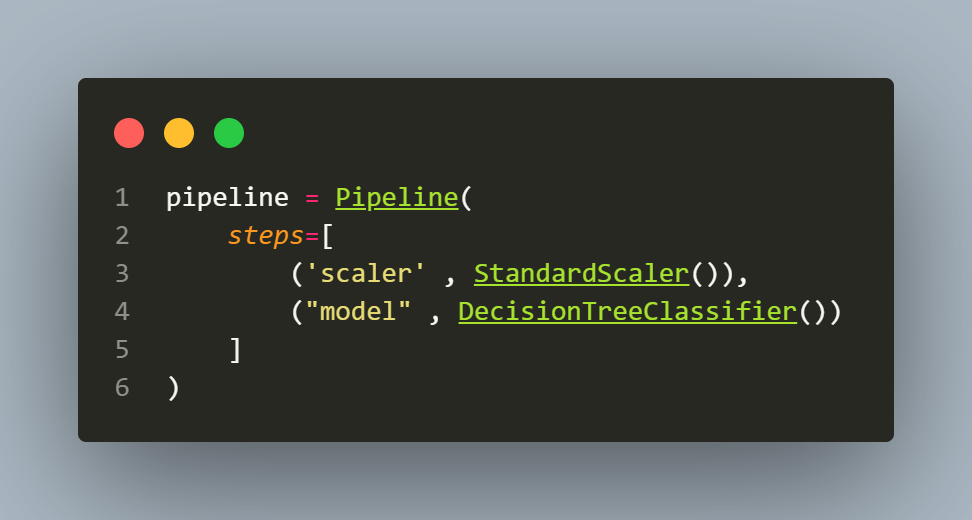
شکل 32: انتخاب ویژگی‌ها با استفاده از SelectKBest()

## 6-1- آموزش مدل درخت تصمیم

در این بخش به آموزش مدل درخت تصمیم می‌پردازیم. ابتدا دیتاست را به نسبت 80 به 20 به دیتای آموزش و تست تقسیم بندی می‌کنیم و سپس دیتای آموزشی را با استفاده از oversampling بالانس می‌کنیم. حال دیتای آموزش برای آموزش مدل آماده است.

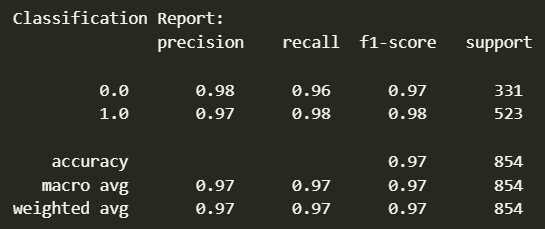
### 1-6-1-آموزش مدل با پارامترهای پیشفرض

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 (33) این پایپ‌لاین را نمایش می‌دهد.



شکل 33: پایپ‌لاین آموزش مدل پیشفرض

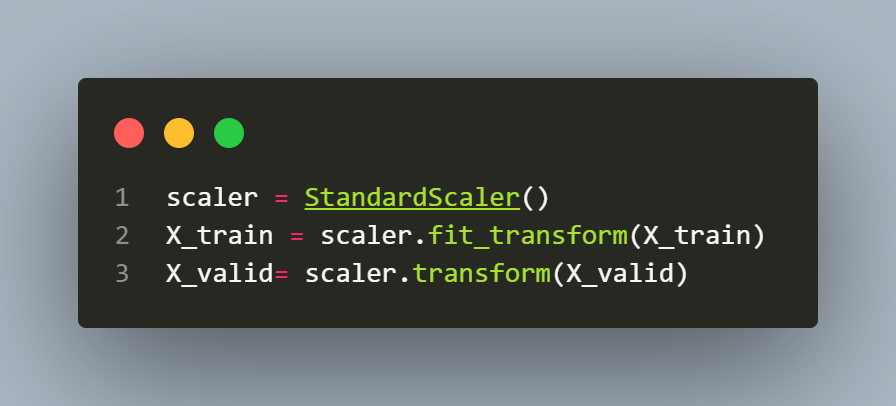
سپس با استفاده از متد .fit() از شی پایپ‌لاین و پاس دادن دیتاست آموزشی، استاندارد سازی و آموزش مدل را انجام می‌دهیم. همچنین با پاس دادن دیتاست تست به متد .predict() داده‌های تست را استانداردسازی می‌کنیم و سپس اجازه می‌دهیم مدل پیش‌بینی را انجام دهد. حال با استفاده از تابع classification\_report() معیارهای دقت، یادآوری و f1 را نمایش می‌دهیم. شکل (34) گزارش به دست آمده را نمایش می‌دهد.



شکل 34: گزارش به دست آمده از آموزش مدل پیشفرض

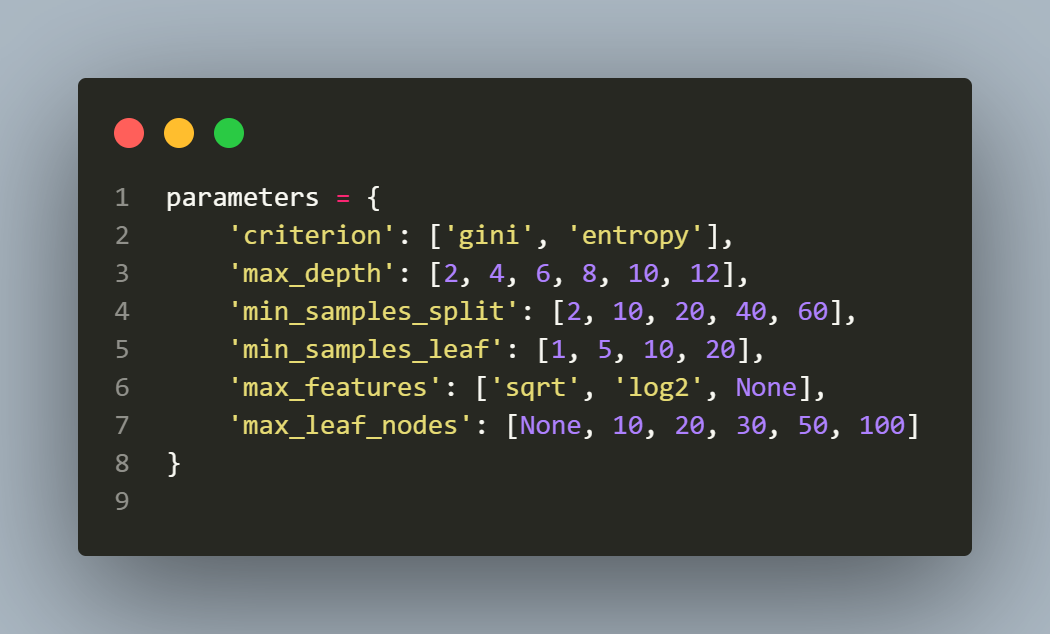
### 2-6-1-پیدا کردن هایپرپارامترهای بهینه با استفاده از GridSearch

حال به بهینه سازی هایپرپارامترهای مدل می‌پردازیم. دیتاست را به دو بخش تقسیم می‌کنیم، و سپس دیتاست آموزش را بالانس می‌کنیم. سپس یک شی از کلاس StandardScaler() تعریف می‌کنیم و با متدهای .fit\_transform() و .transform() به ترتیب دیتاست‌های آموزشی و تست را استاندارد سازی می‌کنیم. شکل (35) نحوه انجام این کار را نمایش می‌دهد.



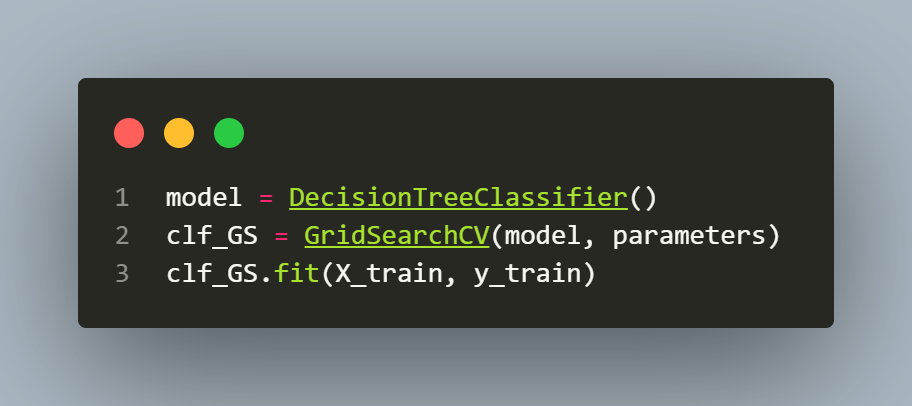
شکل 35: استانداردسازی داده‌ها

سپس یک دیکشنری تعریف می‌کنیم که کلیدهای آن، هایپرپارامترهای مدل هستند که می‌تواند 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 (36) دیکشنری هایپرپارامترها را نمایش می‌دهد.



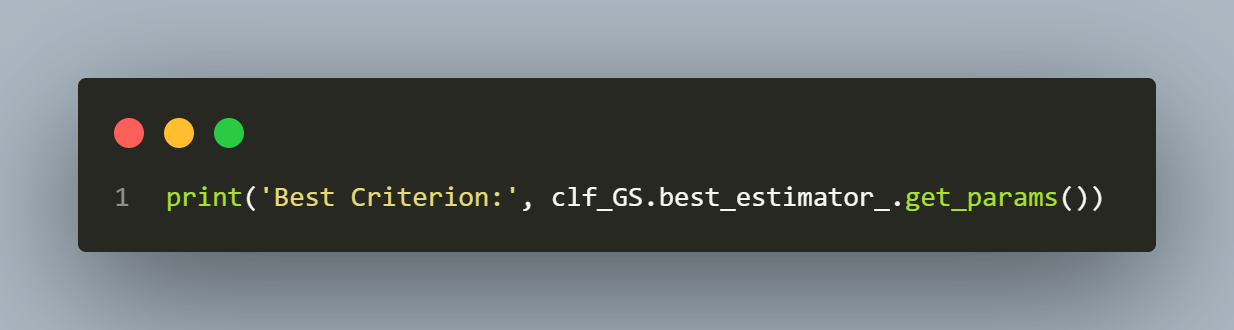
شکل 36: دیکشنری هایپرپارامترها

سپس یک شی از کلاس DecisionTreeClassifier() می‌سازیم. همچنین یک شی از کلاس GridSearchCV() می‌سازیم و مدل ساخته شده و دیکشنری پارامترها را به آن پاس می‌دهیم. سپس با استفاده از متد .fit() و پاس دادن دیتاست آموزشی، Grid Search را آغاز می‌کنیم. این متد، همه ترکیب‌های ممکن از پارامترها را به مدل می‌دهد و مدل را با آن آموزش می‌دهد و ارزیابی می‌کند. شکل (37) این کد را نمایش می‌دهد.

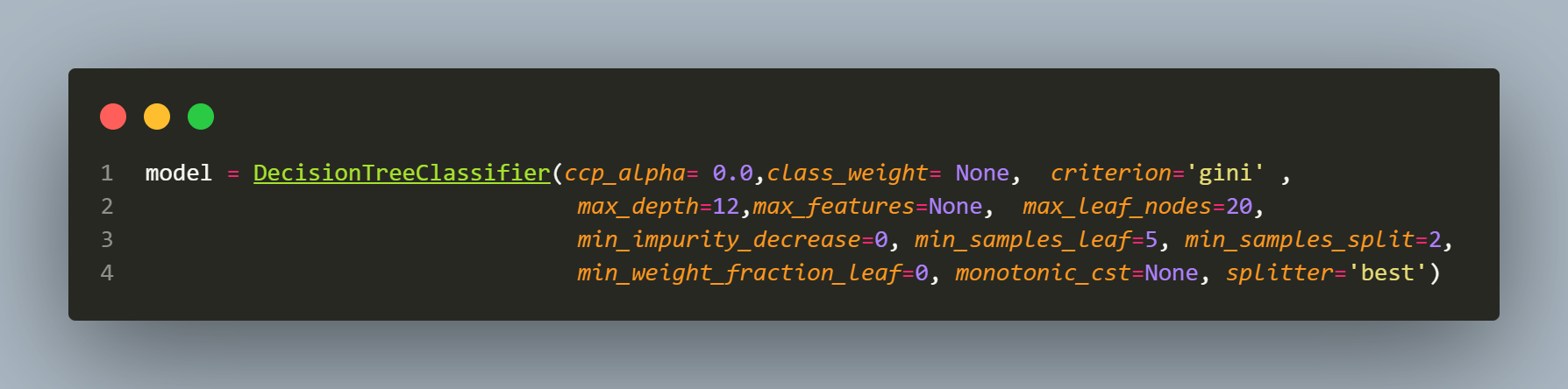


شکل 37: پیداکردن ترکیب بهینه هایپرپارامترها با گرید سرچ

سپس با استفاده از کد شکل (38) پارامترهای بهینه را نمایش می‌دهیم.

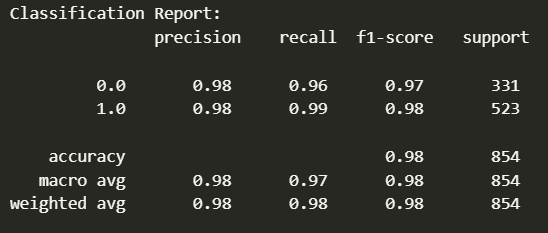


شکل 38: نمایش پارامترهای بهینه

شکل (39) کلاس درخت تصمیم با آرگومان‌ها بهینه را نمایش می‌دهد. 

شکل 39: درخت تصمیم با هایپرپارامترهای بهینه

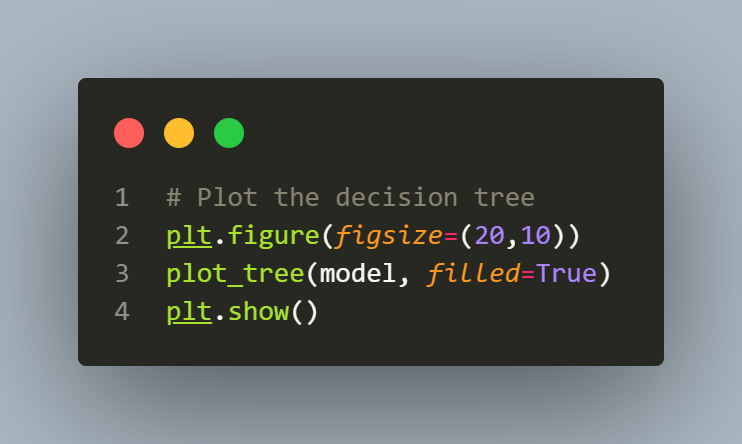
حال به طور مشابه با قبل مدل را آموزش و تست می‌کنیم. شکل (40) گزارش به دست آمده را نشان می‌دهد.



شکل 40: گزارش دسته بندی درخت تصمیم با هایپرپارامترهای بهینه

مشاهده می‌شود که معیارهای ارزیابی مدل، یک تا دو درصد بهبود داشته‌اند.

حال با تابع plot\_tree() و پاس دادن مدل، درخت ایجاد شده را رسم می‌کنیم. شکل (41) کدی که وظیفه انجام این کار را دارد نمایش می‌دهد. همچنین شکل (42) درخت به دست آمده را نمایش می‌دهد.



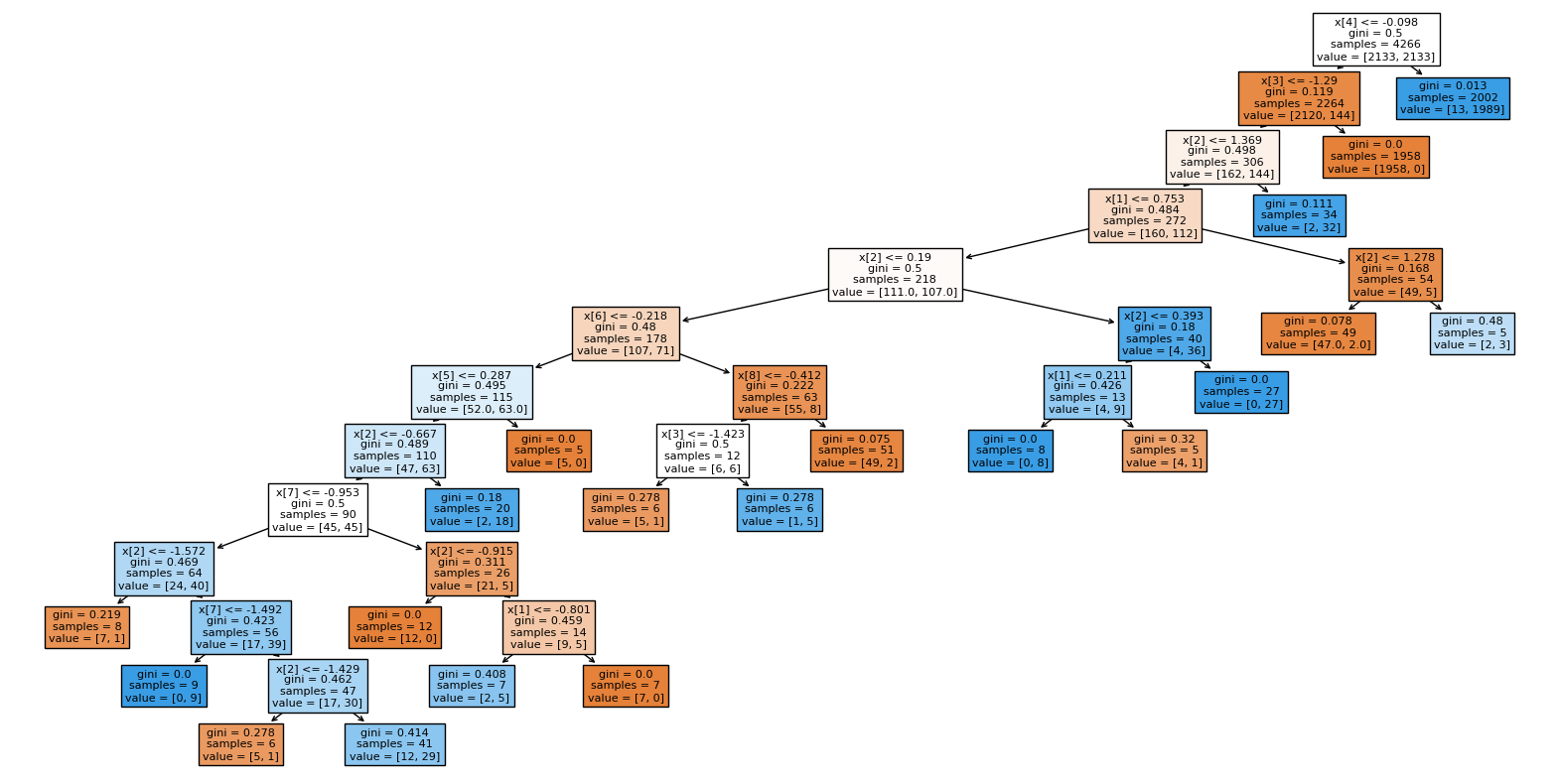
شکل 41: کد مربوط به نمایش شکل درخت

### 3-6-1-هرس کردن درخت و نمایش مجدد آن (بخش امتیازی)

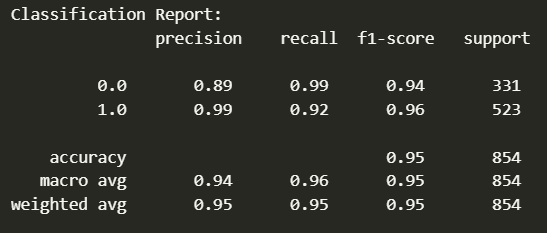
پارامتر ccp\_alpha پارامتر کلیدی برای هرس کردن درخت است. اگر مقدار آن برابر صفر باشد، هیچگونه هرس کردنی صورت نمی‌گیرد. همچنین با پارامترهای max\_depth و max\_leaf\_nodes و min\_samples\_leaf و min\_samples\_split نیز می‌توان درخت را هرس کرد. این پارامترها به ترتیب، حداکثر عمق درخت، حداکثر شاخه‌ها، مینیم تعداد شاخه‌های نمونه و مینیمم تعداد تقسیم نمونه‌ها را تعیین می‌کنند.

برای هرس کردن درخت، تمرکز خود را روی پارامتر ccp\_alpha قرار می‌دهیم. مدل را با پارامترهای قبلی آموزش می‌دهیم، ولی مقدار این پارامتر را برابر 0.01 قرار می‌دهیم. سپس مدل را آموزش می‌دهیم و تست می‌کنیم. شکل (43) گزارش به دست آمده از این مدل هرس شده را نمایش می‌دهد. همچنین شکل (44) درخت هرس شده را نمایش می‌دهد.

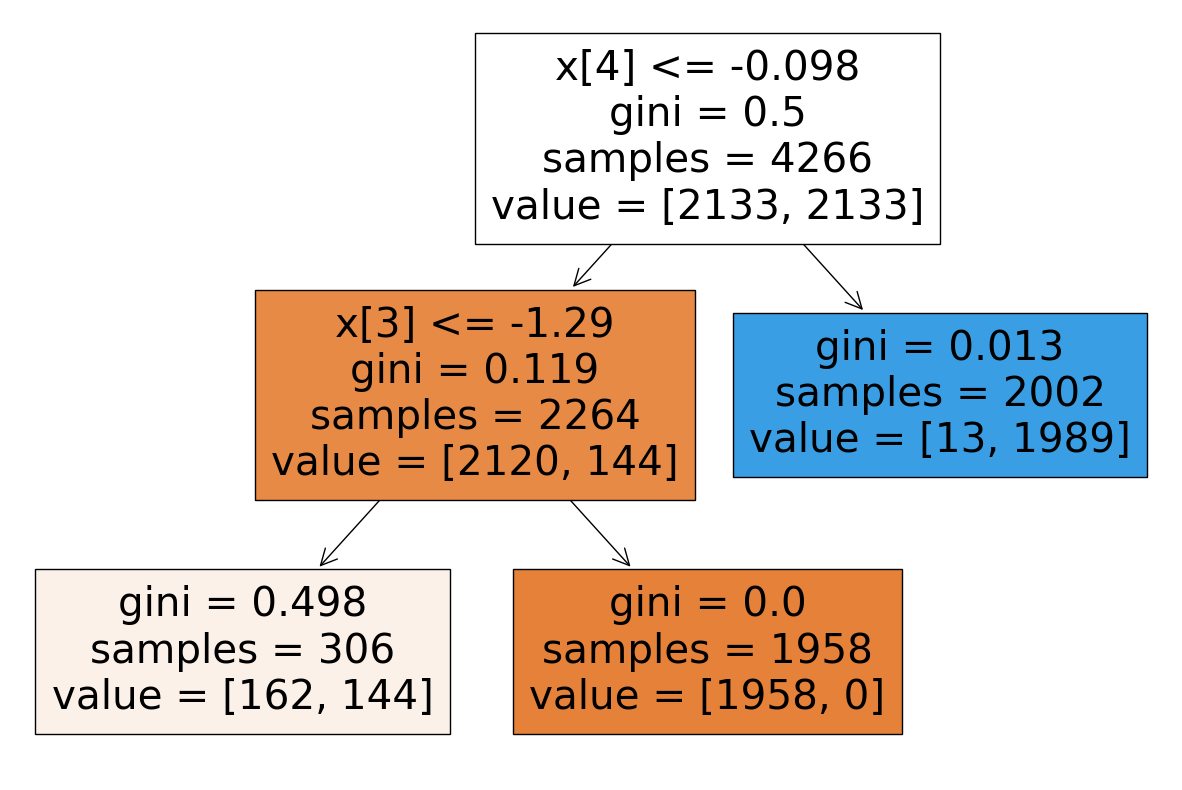
مشاهده می‌شود که درخت هرس شده بسیار کوچکتر از درخت اصلی است ولی گزارش عملکرد آن تنها چند درصد ضعیف‌تر از درخت اصلی است. مزایای این درخت، توضیح پذیری ساده آن و مقاومت آن در برابر بیش برازش است. در زمینه‌هایی که این دو مورد اهمیت دارند، استفاده از درخت هرس شده توصیه می‌شود.



شکل 42: درخت به دست آمده از هایپرپارامترهای بهینه‌شده



شکل 43: گزارش عملکرد درخت هرس شده

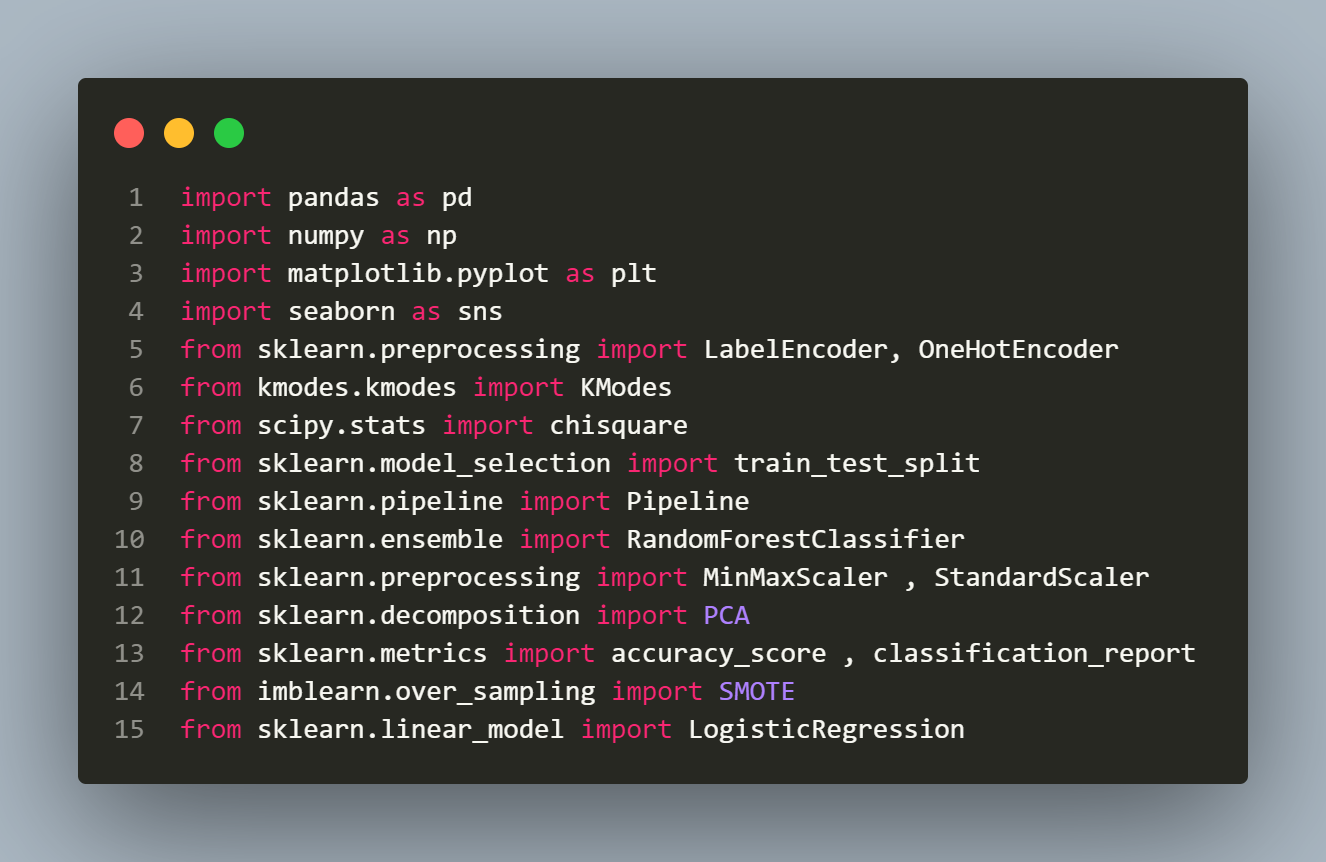


شکل 44: درخت هرس شده

# 2-پاسخ سوال دوم

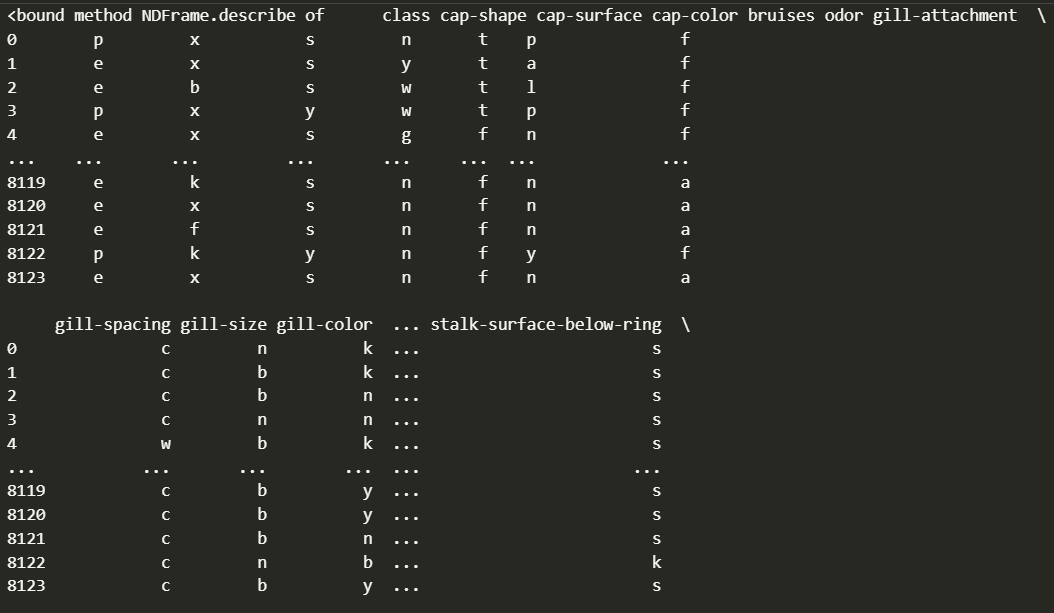
## 1-2-آماده سازی داده‌ها

ابتدا کتابخانه‌های مورد نیاز را وارد می‌کنیم. شکل (45) این کتابخانه‌ها را نمایش می‌دهد.



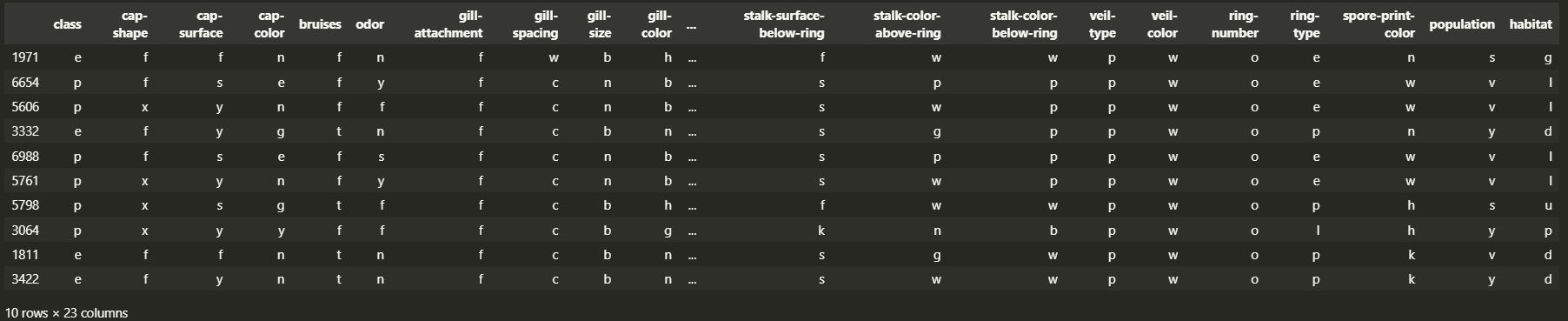
شکل 45: وارد کردن کتابخانه‌های مورد نیاز

سپس متد .describe() را استفاده می‌کنیم. شکل (46) خروجی این متد را نمایش می‌دهد.



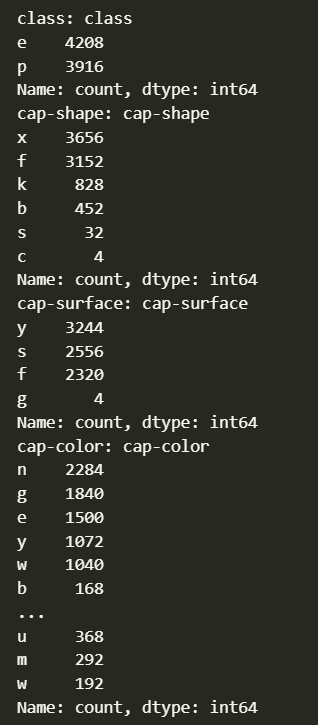
شکل 46: خروجی متد .describe()

سپس ده سطر رندوم را نمایش می‌دهیم. برای انجام این کار، مشابه سوال یک عمل می‌کنیم. شکل (47) این خروجی را نمایش می‌دهد.



شکل 47: ده سطر رندوم دیتافریم

از خروجی این بخش و خروجی متد .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 دوتایی هستند. برای این منظور از متد .value\_counts() استفاده می‌کنیم. برای هر ستون این متد را اجرا می‌کنیم و نتیجه را نمایش می‌دهیم. کد این بخش دقیقاً همانند کد شکل (11) است. بخشی از خروجی این کد در شکل (48) نمایش داده شده است.

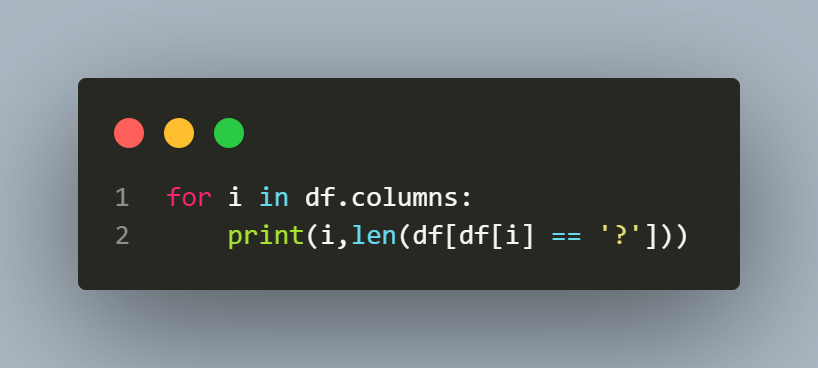


شکل 48: خروجی تعداد هر دسته در هر ستون

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

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

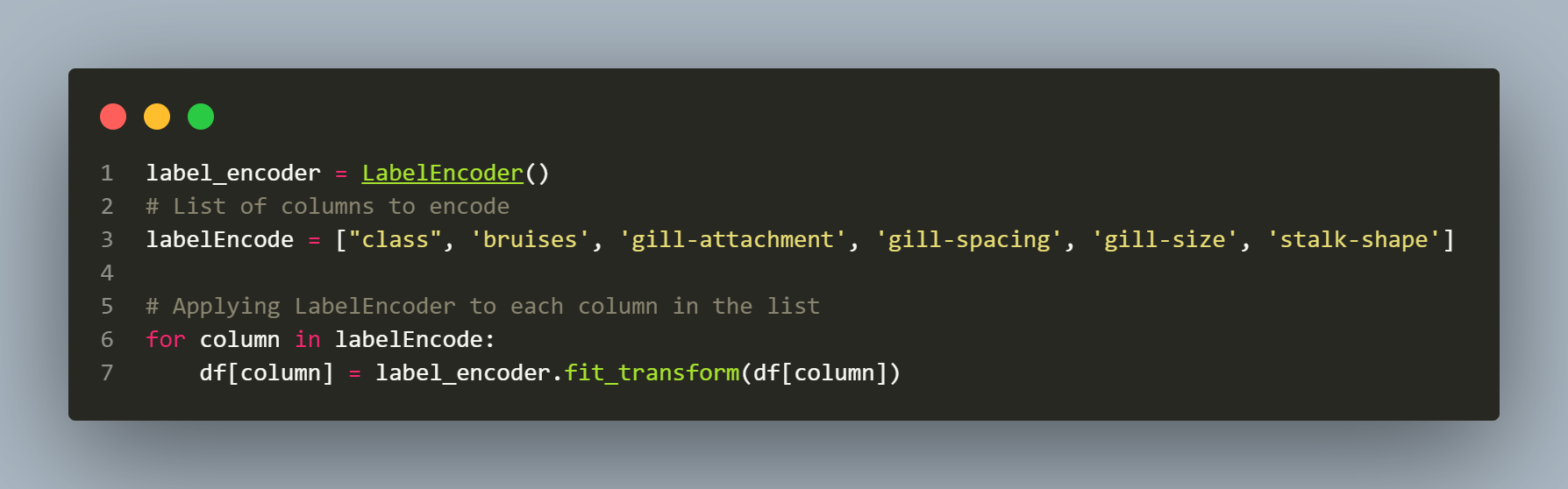
حال بررسی می‌کنیم که آیا در ستون‌ها، مقدار ؟ داریم یا خیر. برای این منظور از کد شکل (49) استفاده می‌کنیم.



شکل 49: پیدا کردن ؟ در دیتاست

مشاهده می‌شود که ستون stalk-root دارای 2480 مقدار گمشده است. 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 LabelEncoder() 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 متد .fit\_transform() یکی یکی ستون‌ها را انکود می‌کنیم. شکل (50) نحوه انجام این کار را نمایش می‌دهد.



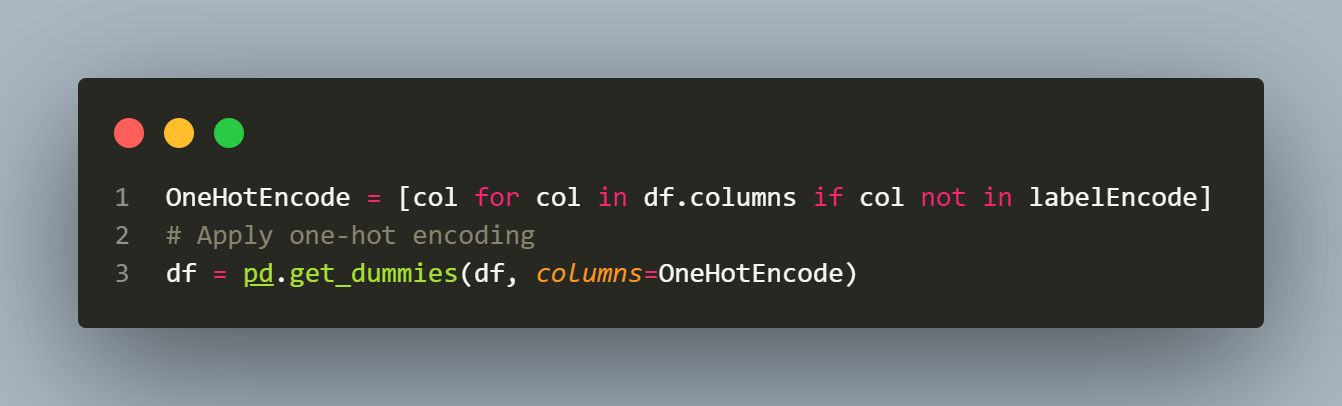
شکل 50: انکود کردن ستون‌های دودویی

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.

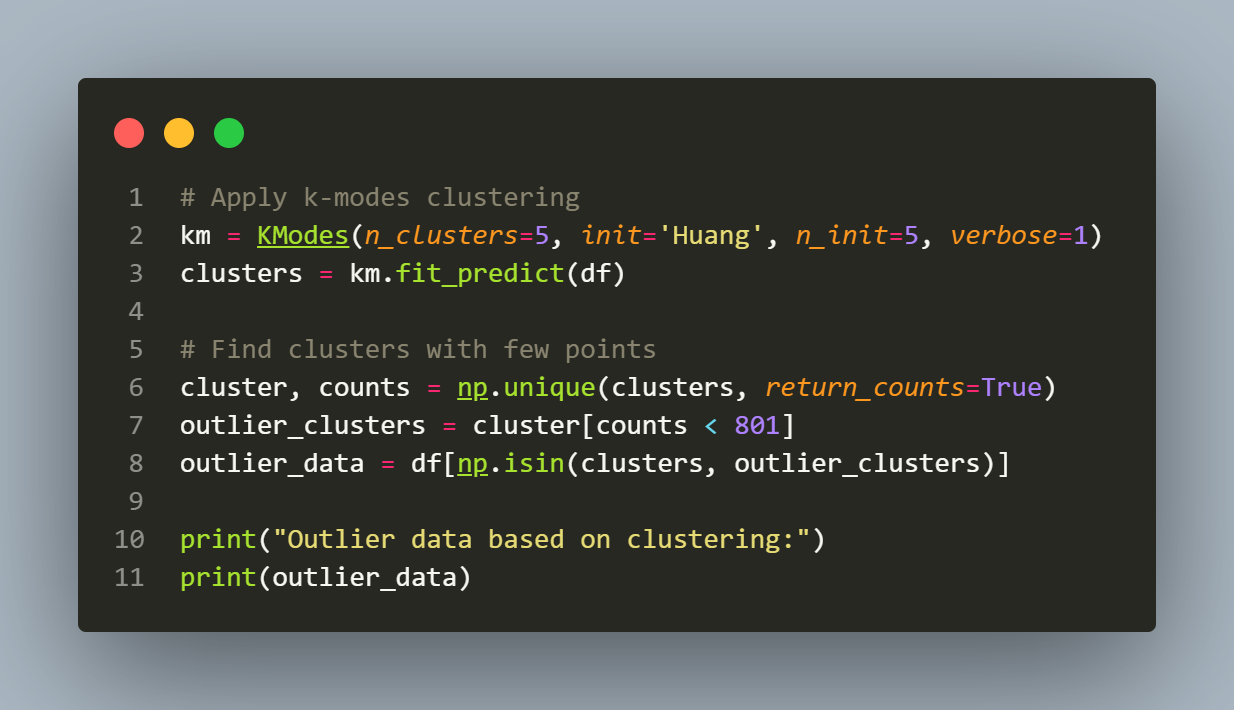
## 2-2-حذف داده‌های پرت

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



شکل 51: one-hot-encode کردن داده‌ها

برای این منظور از کلاس KModes یک شی می‌سازیم. تعداد خوشه‌ها را روی 5 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 ذخیره 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 هشتصد مورد، داده‌ی پرتی یافت نمی‌شود. شکل (52) کد پیدا کردن داده‌های پرت را با استفاده از خوشه‌بندی نمایش می‌دهد.



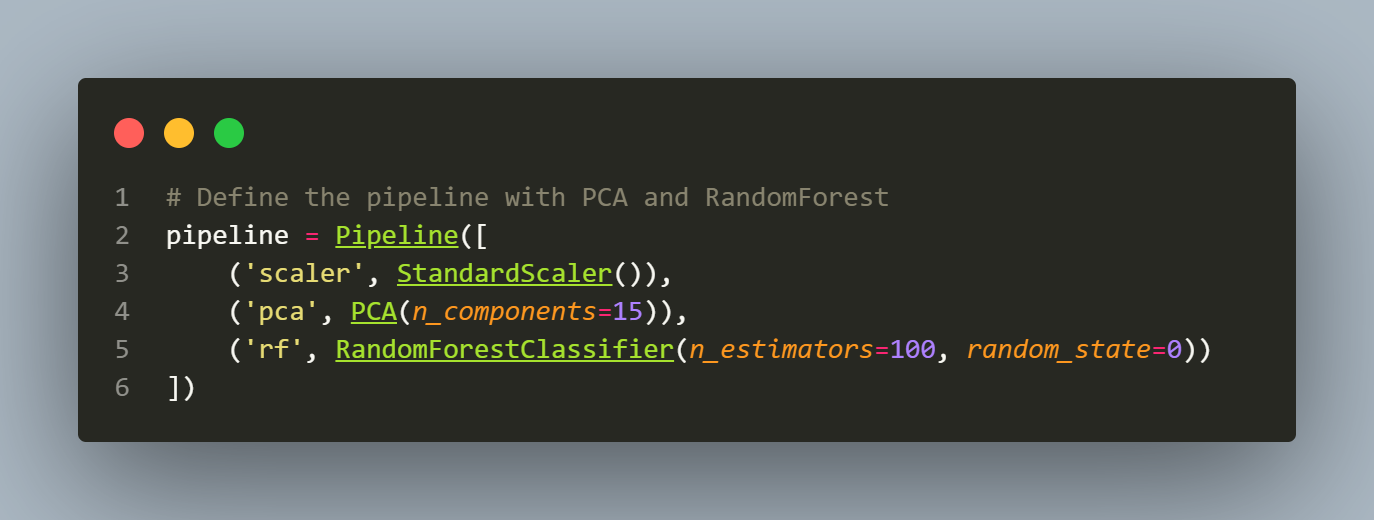
شکل 52: پیداکردن داده‌های پرت با استفاده از خوشه‌بندی

## 3-2-انتخاب ویژگی

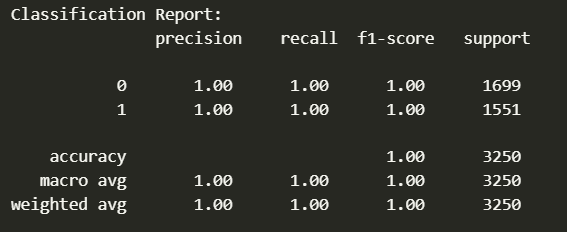
از آنجایی که تعداد ویژگی‌ها بسیار زیاد است، یکی از بهترین راه‌هایی که می‌توان با استفاده از آن انتخاب ویژگی کرد، PCA است. مزیت PCA این است که تعداد زیاد ابعاد را به یک تعداد کم از مولفه‌های‌ها که تاثیر زیادی روی ستون هدف دارند تبدیل می‌کند.

برای این منظور پس از تقسیم‌بندی به داده‌های تست و آموزش، یک شی از کلاس پایپ‌لاین درست می‌کنیم و سپس به ترتیب کلاس‌های StandardScaler، PCA، 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 80 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 100 درصد را می‌دهد. سپس تعداد کمتر را امتحان می‌کنیم. متوجه می‌شویم که برای دستیابی به دقت صد درصد تنها به 15 مولفه اصلی احتیاج داریم.

شکل (53) شی پایپ‌لاین را نمایش می‌دهد و شکل (54) گزارش عملکرد به دست آمده از آموزش و تست 15 مولفه را نمایش می‌دهد.



شکل 53: پایپ‌لاین ساخته شده برای تحلیل مولفه اصلی



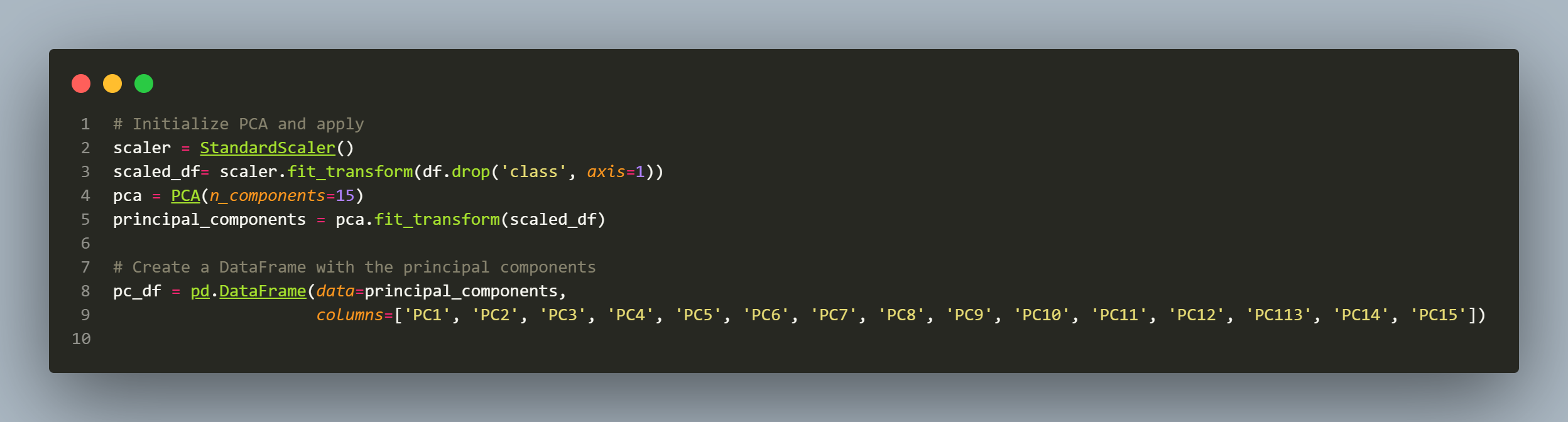
شکل 54: گزارش به دست آمده از آموزش و تست مدل با 15 مولفه اصلی

در نتیجه برای انتخاب ویژگی، از 15 مولفه اصلی استفاده می‌کنیم.

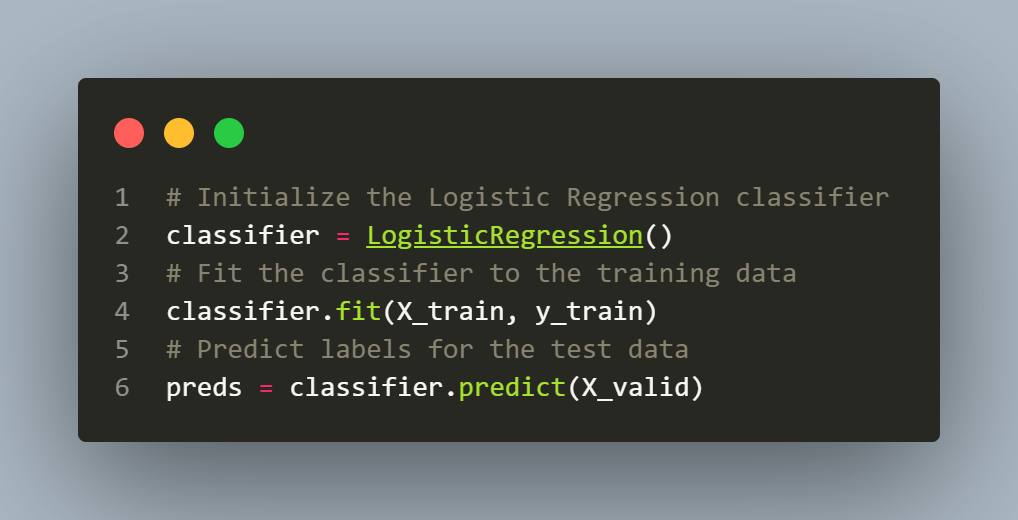
برای این منظور ابتدا دیتاست را با استفاده از کلاس StandardScaler() و متد fit\_transform() استاندارد سازی می‌کنیم. سپس PCA را انجام می‌دهیم و 15 مولفه اصلی اول را انتخاب می‌کنیم. شکل (55) نحوه انجام این کار را نمایش می‌دهد.

## 4-2-آموزش و تست رگرسیون لجستیک

همانند بخش‌های قبلی، ابتدا دیتاست 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 متد .fit() مدل را آموزش می‌دهیم و با استفاده از متد .predict() مدل را تست می‌کنیم. شکل (55) نحوه انجام این کار را نمایش می‌دهد.

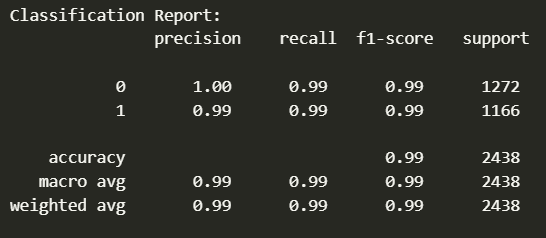


شکل 55: انتخاب 15 مولفه اصلی به عنوان ویژگی‌ها



شکل 56: ساخت، آموزش و تست مدل رگرسیون لجستیک

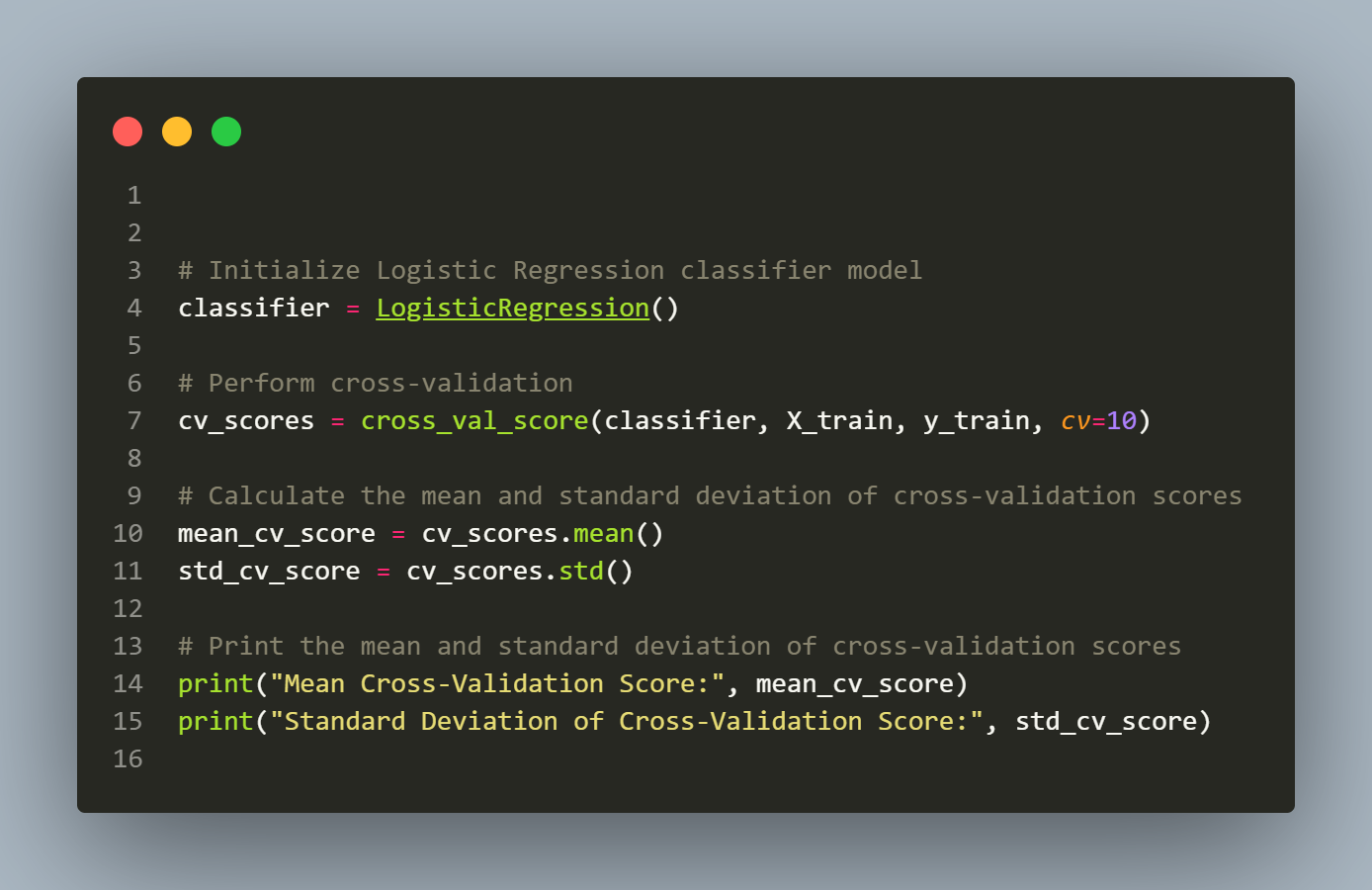
سپس همانند قسمت‌های قبل گزارش عملکرد را نمایش می‌دهیم. شکل (57) گزارش عملکرد رگرسیون لجستیک را نمایش می‌دهد.



شکل 57: 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 (58) نحوه انجام این کار و شکل (59) خروجی میانگین و انحراف استاندارد را نمایش می‌دهیم.



شکل 58: آموزش مدل با Cross Validation

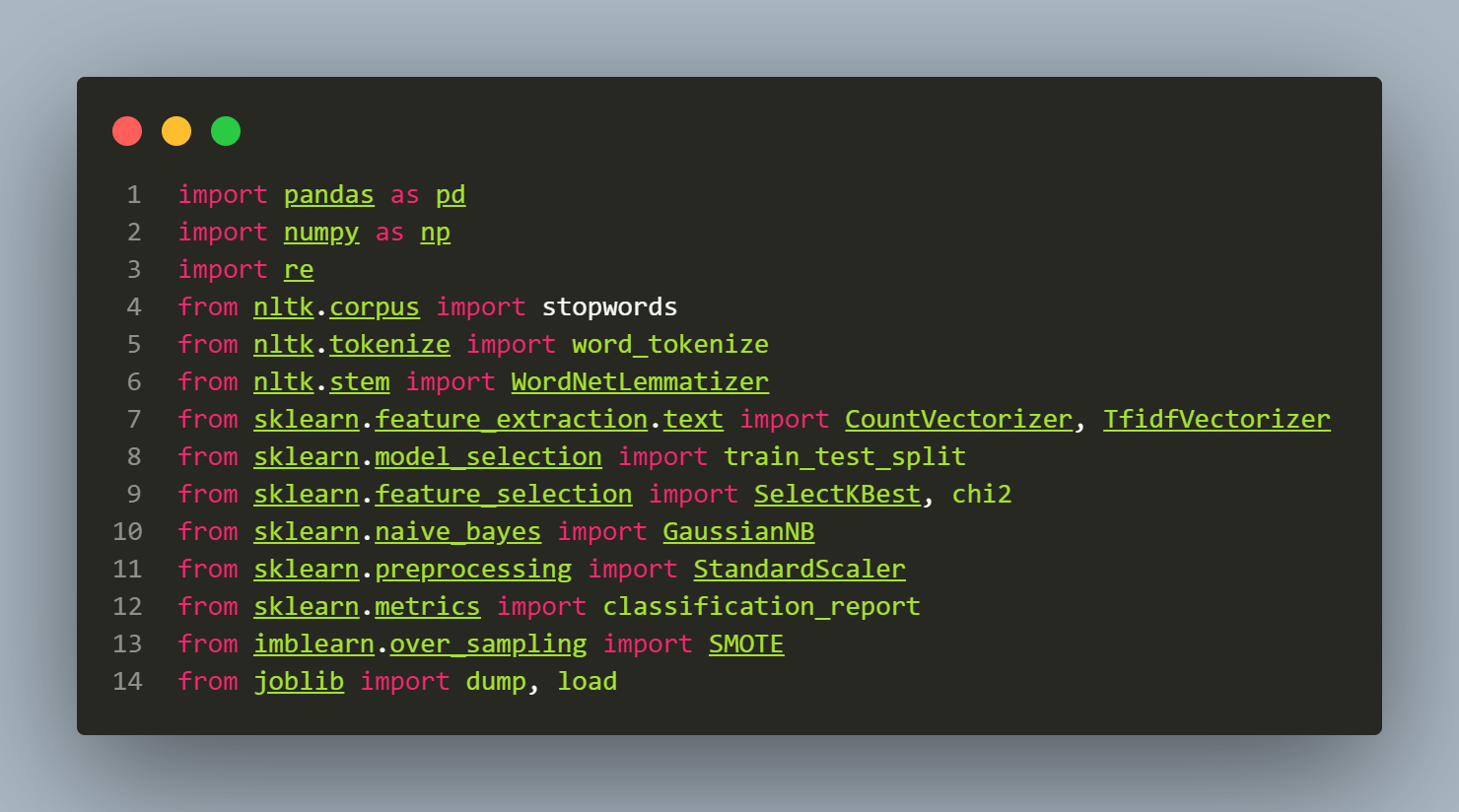


شکل 59: نتیجه Cross Validation

مشاهده می‌شود که هر دو روش دقت یکسانی نتیجه می‌دهند که نشان می‌دهد مدل دچار بیش‌برازش نشده.

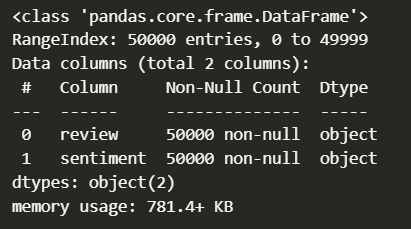
# 3-پاسخ سوال سوم

ابتدا کتابخانه‌های مورد نیاز را وارد می‌کنیم. شکل (60) این کتابخانه‌ها را نمایش می‌دهد.



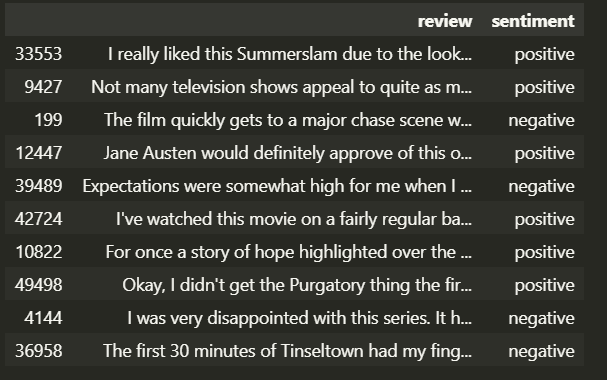
شکل 60: وارد کردن کتابخانه‌های مورد نیاز

پس از خواندن دیتاست، ابتدا با استفاده از متد .info() تعداد رکورد موجود از هر لیبل را نمایش می‌دهیم. شکل (61) خروجی این کار را نمایش می‌دهد.



شکل 61: تعداد ایمیل‌های با لیبل مثبت و منفی

سپس ده سطر رندوم را نمایش می‌دهیم. برای انجام این کار، مشابه سوال یک عمل می‌کنیم. شکل (62) این خروجی را نمایش می‌دهد.



شکل 62: نمایش ده سطر رندوم

حال برای تبدیل لیبل‌ها به صفر و یک، روی سطر هدف متد .apply() را اجرا می‌کنیم. آرگومان این متد را یک تابع لامبدا قرار می‌دهیم که یکی یکی لیبل‌ها رو می‌گیرد و اگر مثبت بود آن را با یک و اگر منفی بود آن را با صفر جایگزین می‌کنیم. شکل (63) نحوه انجام این کار را نمایش می‌دهد.

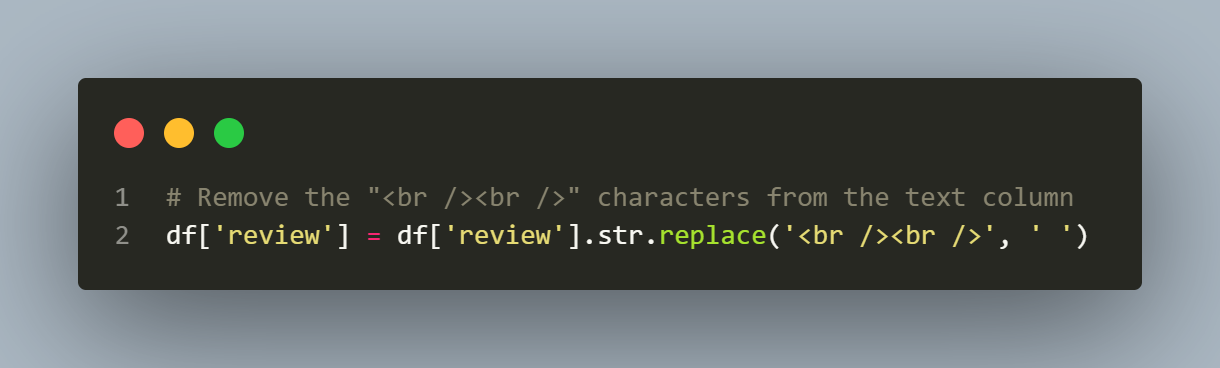


شکل 63: جایگزینی لیبل‌ها با صفر و یک

## 1-3-پیش پردازش داده‌ها

### 1-1-3-حذف تگ‌های <br /><br />

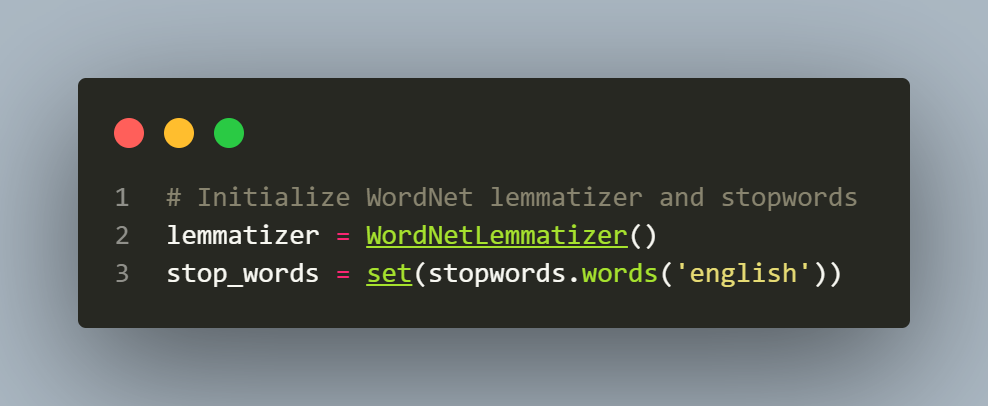
اولین قدم، حذف تگ‌های <br /><br /> از رشته‌ی هرنظر است. برای این منظور از متد .replace() استفاده می‌کنیم. کد شکل (64) نحوه انجام این کار را نمایش می‌دهد.



شکل 64: حذف تگ‌های <br /><br />

### 2-1-3-ساخت تابع پیش‌پردازش

برای انجام پیش پردازش 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 (65) نحوه انجام این کار را نمایش می‌دهد.



شکل 65: ساخت شی ریشه‌یاب و مجموعه ایست‌واژه‌ها

حال تابع را به صورت نشان داده شده در شکل (66) تعریف می‌کنیم. این تابع، یکی یکی متن‌ها را دریافت می‌کند. سپس با استفاده از تابع sub() هر کاراکتری غیر از حروف کوچک و بزرگ انگلیسی و اسپیس را با اسپیس جایگذاری می‌کنیم. سپس با استفاده از تابع word\_tokenize() و پاس دادن متن به عنوان آرگومان، متن را توکن می‌کنیم. سپس هر توکن را با استفاده از List Comprehension بررسی 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 متد .lemmatize() به ریشه بر می‌گردانیم. در نهایت توکن‌‌ها را دوباره در کنار هم قرار می‌دهیم و جملات را تشکیل می‌دهیم.



شکل 66: تابع پیش‌پردازش متن

سپس با استفاده از متد .apply() و استفاده از یک تابع لامبدا، یکی یکی متن‌ها را به این تابع پاس می‌دهیم و متن‌های پیش پردازش شده را دریافت می‌کنیم. شکل (67) نحوه انجام این کار را نمایش می‌دهد.

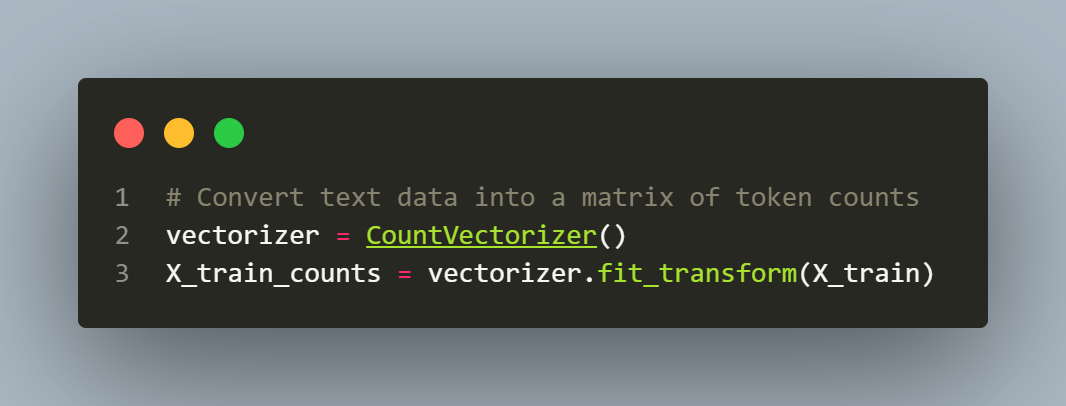


شکل 67: پیش‌پردازش متن‌ها با استفاده از تابع تعریف شده

## 2-3-استفاده از CountVectorizer

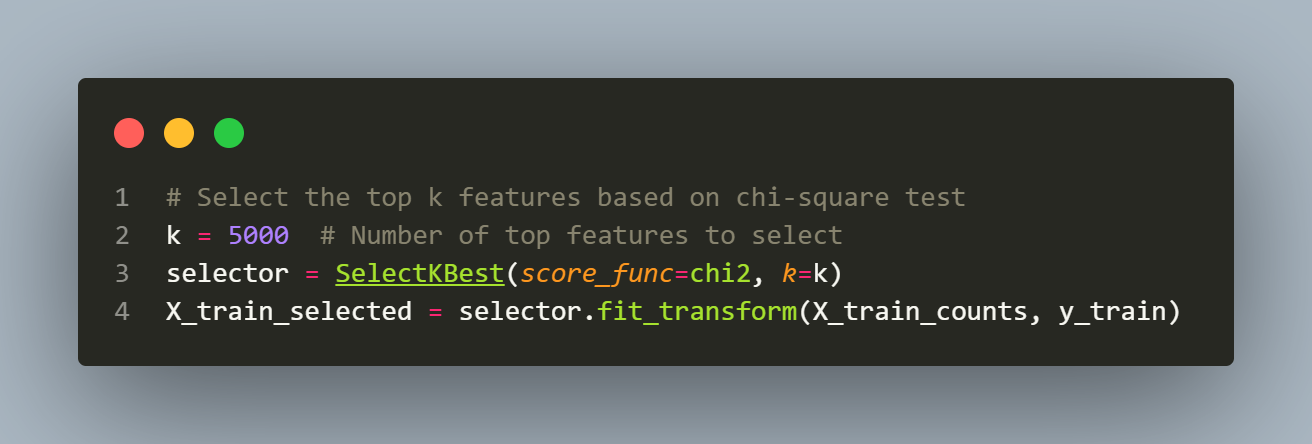
CountVetorizer مجموعه ای از اسناد متنی را به یک ماتریس تبدیل می کند که در آن ردیف ها نشان دهنده اسناد و ستون ها نشان دهنده کلمه‌ها هستند. این تابع تعداد تکرار هر توکن را در هر سند شمارش می کند، و یک ماتریس Document-term با مقادیر صحیح نشان دهنده فراوانی هر کلمه ایجاد می کند.

برای استفاده از این متد، 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 متد .fit\_transform() ماتریس Document-term را از داده‌های آموزشی ایجاد می‌کنیم. شکل (68) نحوه انجام این کار را نمایش می‌دهد.



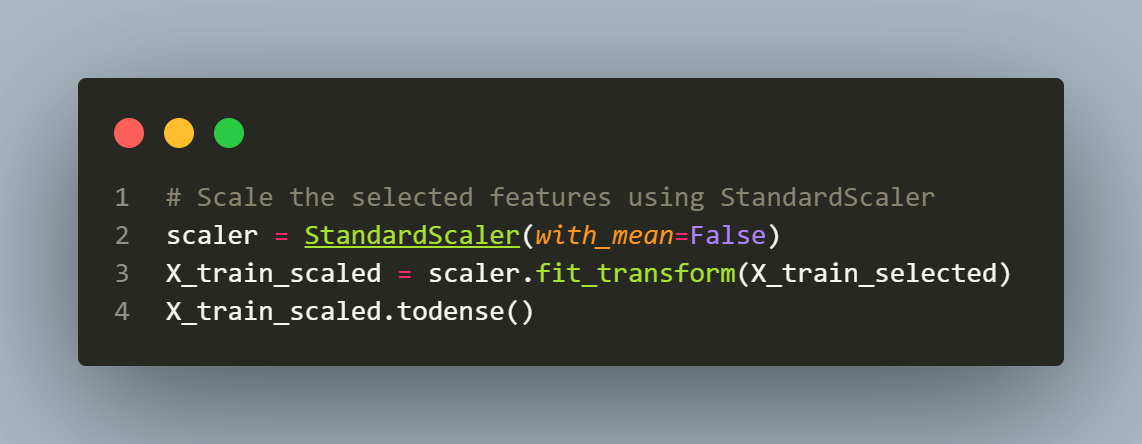
شکل 68: بردار کردن متون با استفاده از CountVectorizer

سپس با استفاده از تابع SelectKBest و آزمون مربع کای، 5000 کلمه را به عنوان ویژگی‌های مهم انتخاب می‌کنیم. شکل (69) نحوه انجام این کار را نمایش می‌دهد.



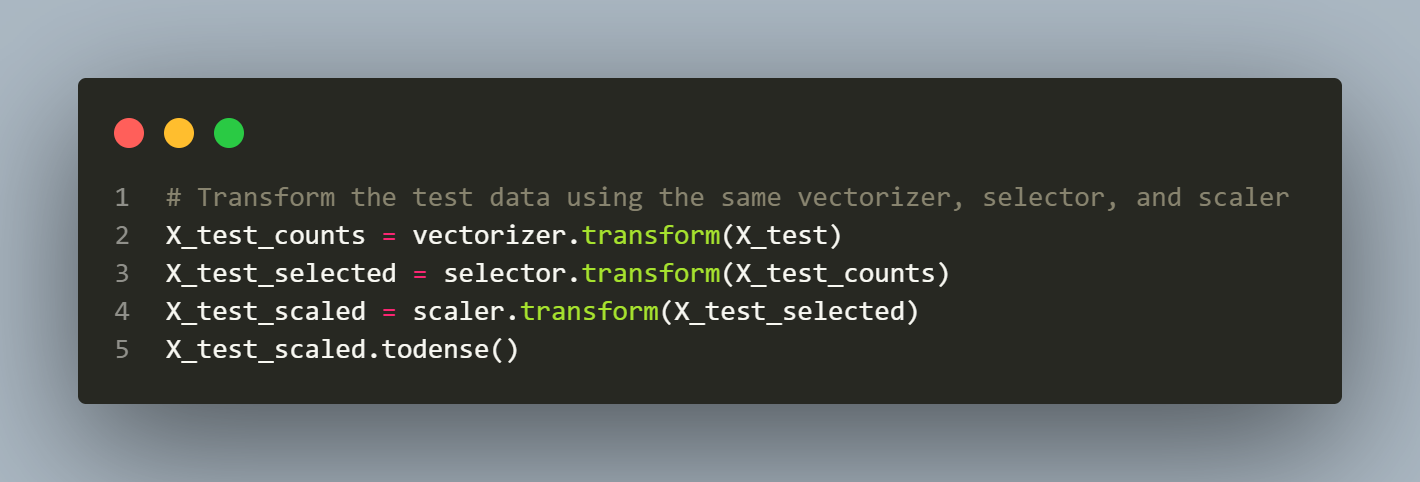
شکل 69: انتخاب 5000 کلمه مهم با استفاده از آزمون مربع کای

در قدم بعدی ماتریس به دست آمده را استاندارد سازی می‌کنیم و به شکل یک ماتریس چگال در می‌آوریم. شکل 70 نحوه انجام این کار را نمایش می‌دهد.



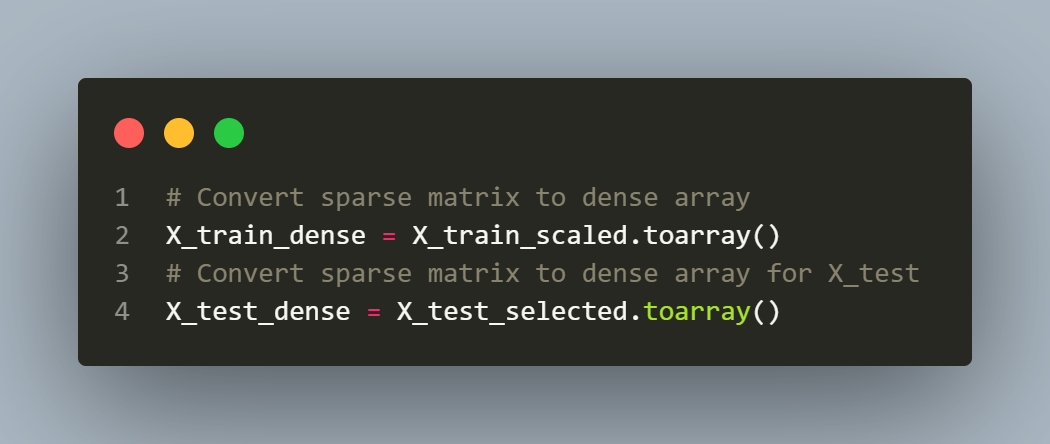
شکل 70: استاندارد سازی و تبدیل ماتریس به ماتریس چگال

حال با استفاده از متد .transform() از هر یک از اشیا ساخته شده، دیتاست تست را هم به ترتیب برداری می‌کنیم، ویژگی‌ها را انتخاب می‌کنیم و ماتریس به دست آمده را استاندارد می‌کنیم. همچنین در نهایت ماتریس به دست آمده را چگال می‌کنیم. شکل (71) نحوه انجام این کار را نمایش می‌دهد.



شکل 71: پیش پردازش داده‌های تست

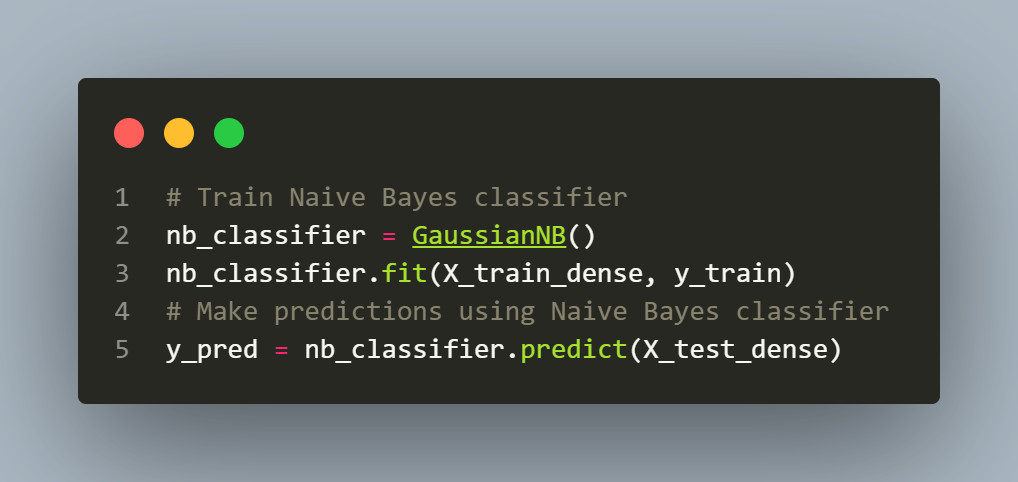
در قدم بعدی ماتریس‌های ایجاد شده را به آرایه‌های خلوت تبدیل می‌کنیم. دلیل انجم این کار این است که متدهای یادگیری ماشین کتابخانه‌ سایکیت‌لرن توانایی کار با ماتریس‌های چگال را ندارند. شکل (72) نحوه انجام این کار را نمایش می‌دهد.



شکل 72: تبدیل ماتریس‌های چگال به آرایه‌های خلوت

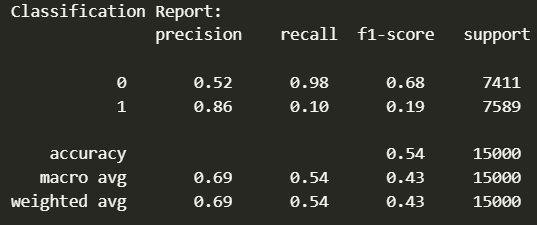
سپس در قدم بعدی با استفاده از oversampling دیتاست آموزشی را بالانس می‌کنیم. نحوه انجام این کار دقیقاً به شکل کار انجام شده در قسمت 1-4-1 و شکل (27) است. حال داده‌ها برای آموزش و تست مدل آماده هستند.

برای آموزش مدل، از کلاس GaussianNB() یک شی می‌سازیم. با پاس دادن دیتاست آموزشی به متد .fit() از این شی، آموزش مدل را انجام می‌دهیم. همچنین با پاس دادن دیتاست تست به متد .predict() پیش‌بینی را انجام می‌دهیم. شکل (73) نحوه انجام این کار را نمایش می‌دهد.



شکل 73: آموزش و تست مدل بیز

شکل (74) نیز گزارش عملکرد این مدل روی دیتاهای تست را نمایش می‌دهد. نحوه ایجاد این گزارش دقیقاً مشابه توضیحات ارائه شده در بخش 1-6-1 است.

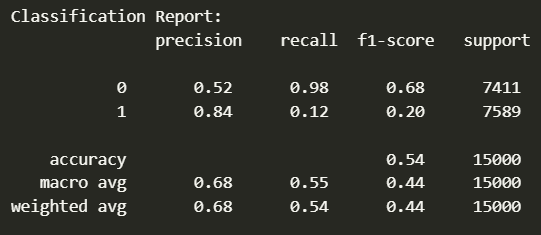


شکل 74: گزارش عملکرد مدل نایو بیز

## 3-3- 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 آموزنده‌تر می کند.

برای استفاده از TF-IDF پس از تقسیم دیتاست به آموزش و تست، از کلاس TfidfVectorizer() یک شی می‌سازیم. سپس با استفاده از متد fit\_transform() و پاس دادن ستون متن بردارها را ایجاد می‌کنیم و سپس نتیجه را در یک متغیر دیگر ذخیره می‌کنیم. سپس تمامی مراحل بالانس کردن دیتاست آموزشی، پیدا کردن 5000 ویژگی برتر، استاندارد کردن، ایجاد آرایه‌های خلوت، و آموزش و تست مدل را همانند بخش قبلی انجام می‌دهیم. شکل (75) گزارش عملکرد مدل آموزش دیده روی ویژگی‌های به دست آمده از TF-IDF را نمایش می‌دهد.



شکل 75: گزارش عملکرد مدل به دست آمده از دیتاست TF-IDF

مشاهده می‌شود که هر دو مدل عملکرد چندان مناسب ندارند. ولی عملکرد مدل آخر کمی بهتر است. پس این مدل را برای استفاده در آینده ذخیره می‌کنیم. برای این کار از کتابخانه joblib و تابع dump() استفاده می‌کنیم. آرگومان‌های این تابع را به ترتیب شی مدل و نامی که می‌خواهیم با آن ذخیره شود قرار می‌دهیم. شکل (76) نحوه استفاده از این مدل را نمایش می‌دهد.



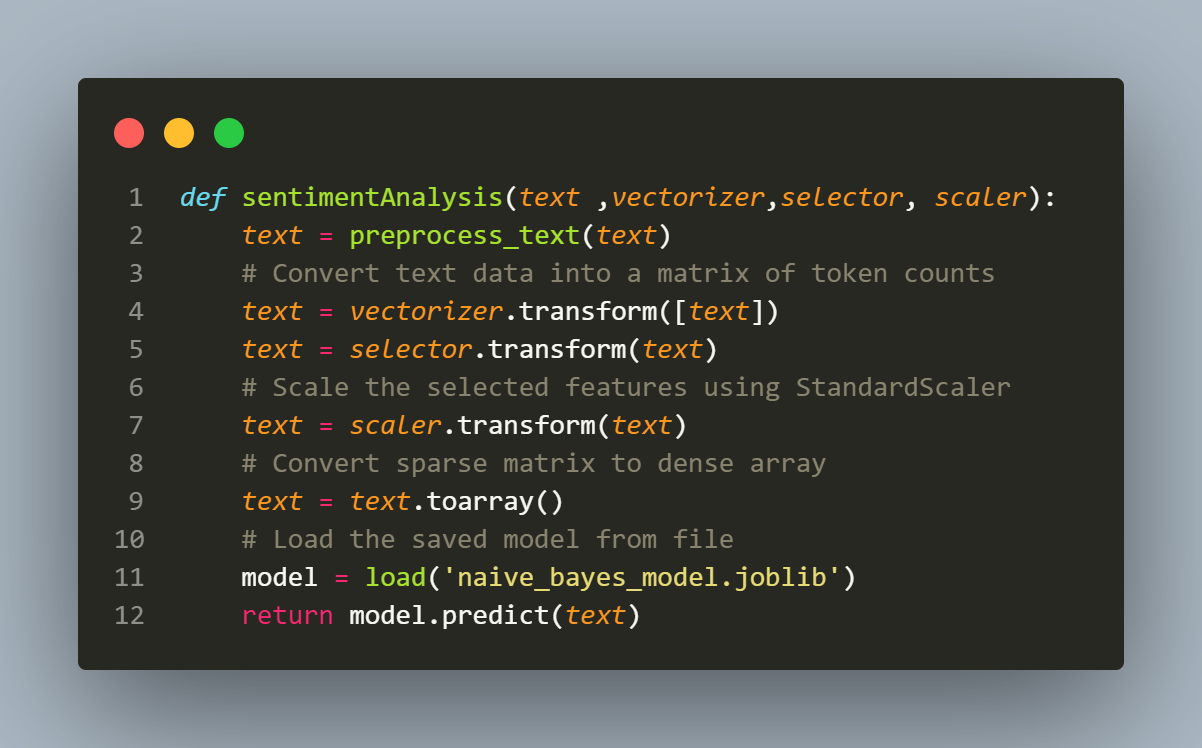
شکل 76: ذخیره کردن مدل برای استفاده در آینده

## 4-3-ساخت تابعی که به نظرات جدید برچسب می‌دهد

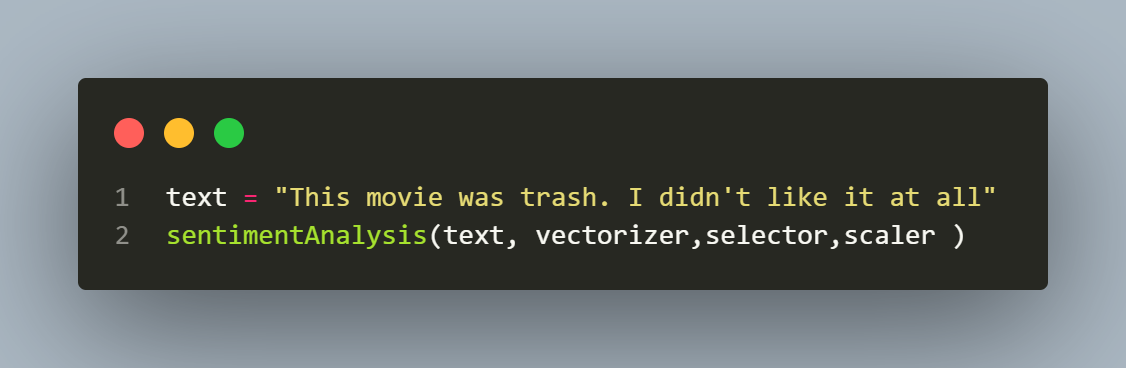
برای اینکه عواطف نظرات جدید را بسنجیم، یک تابع همانند کدی که در شکل (77) نشان داده شده ایجاد می‌کنیم.

این تابع، متن، شی TF-IDF ، شی انتخاب کننده ویژگی، و شی استانداردساز را دریافت می‌کند و به ترتیب اعمال زیر را انجام می‌دهد:

ابتدا با استفاده از تابع تعریف شده در بخش 3-1-2 متن را آماده می‌کند. سپس آن را به بردار تبدیل می‌کند. سپس انتخاب ویژگی را انجام می‌دهد و بردار ایجاد شده را استاندارد و در نهایت به یک آرایه خلوت تبدیل می‌کند. سپس با استفاده از تابع load() از کتابخانه joblib، مدل را لود می‌کند و با استفاده از آن پیش‌بینی را انجام می‌دهد و نتیجه را برمی‌گرداند. شکل (78) یک نمونه از کاربرد این تابع را نمایش می‌دهد.



شکل 77: تابع سنجش عواطف نظرات جدید



شکل 78: کاربرد تابع تحلیل عواطف

و شکل (79) خروجی این تابع را نمایش می‌دهد.



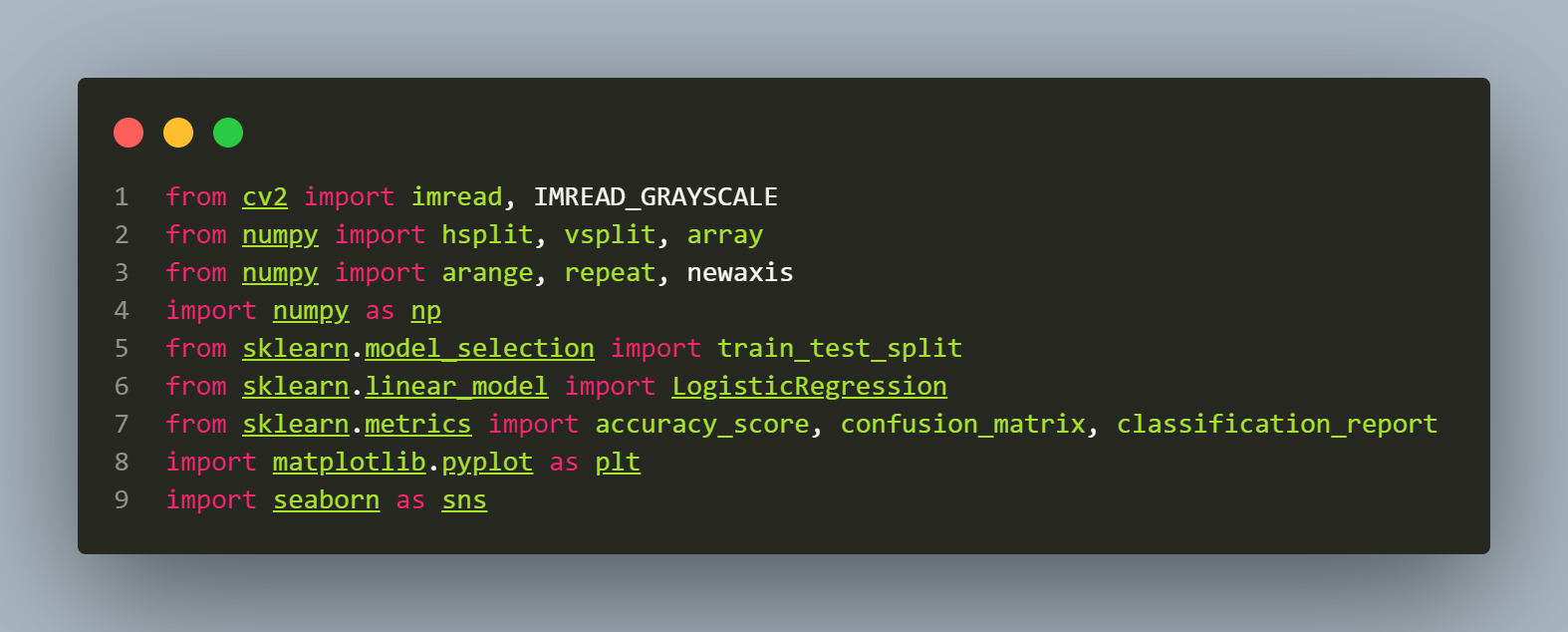
شکل 79: خروجی تابع تحلیل عواطف

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 مثبت تشخیص دهد.

# 4-پاسخ سوال چهارم

ابتدا کتابخانه‌هایی که در ادامه مورد نیاز هستند را ایمپورت می‌کنیم. شکل (80) این کتابخانه‌ها را نمایش می‌دهد.



شکل 80: وارد کردن کتابخانه‌های مورد نیاز

## 1-4-وارد کردن و پیش‌پردازش داده‌ها

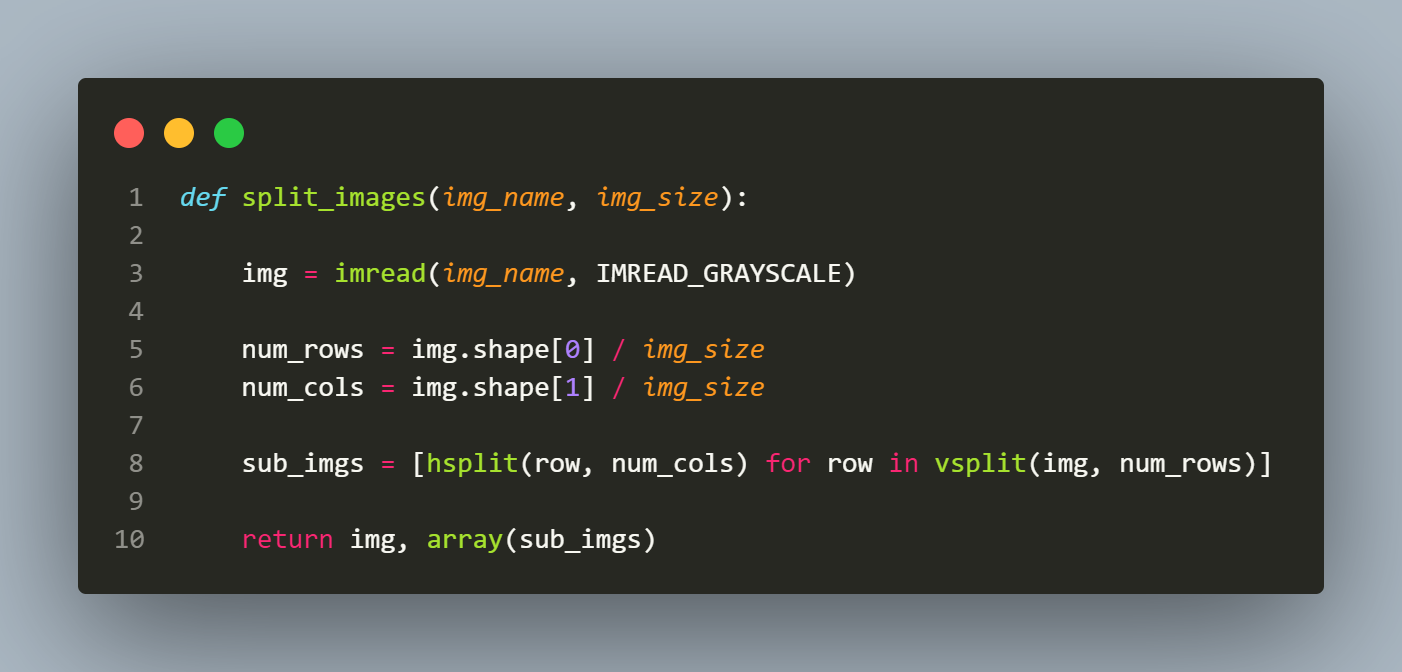
ابتدا فایل اصلی عکس‌ها را دانلود می‌کنیم. این فایل شامل همه عکس‌ها است که به ترتیب از صفر تا نه به صورت ردیفی قرار گرفته‌اند. شکل (81) این فایل را نمایش می‌دهد.



شکل 81: فایل digits.png

با کمی جست و جو متوجه می‌شویم که هر یک از اعداد، در یک مربع به ابعاد 20 در 20 پیکسل قرار گرفته‌اند. بنابراین با نوشتن تابعی می‌توانیم عکس را دریافت کنیم و آن را به عکس‌های کوچکتر تقسیم کنیم. شکل (82) این تابع را نمایش می‌دهد.

حال به توضیح این تابع می‌پردازیم. این تابع، مسیر عکس و ابعاد عکس را دریافت می‌کند. ابتدا با استفاده از تابع imread() از کتابخانه OpenCV و پاس دادن مسیر عکس و آرگومان IMREAD\_GRAYSCALE عکس را به صورت سیاه سفید می‌خوانیم و در آرگومان img ذخیره



شکل 82: تابع تقسیم کننده عکس بزرگ به عکس‌های کوچکتر

می‌کنیم. عکس به صورت یک آرایه دو بعدی در این متغیر ذخیره می‌شود.

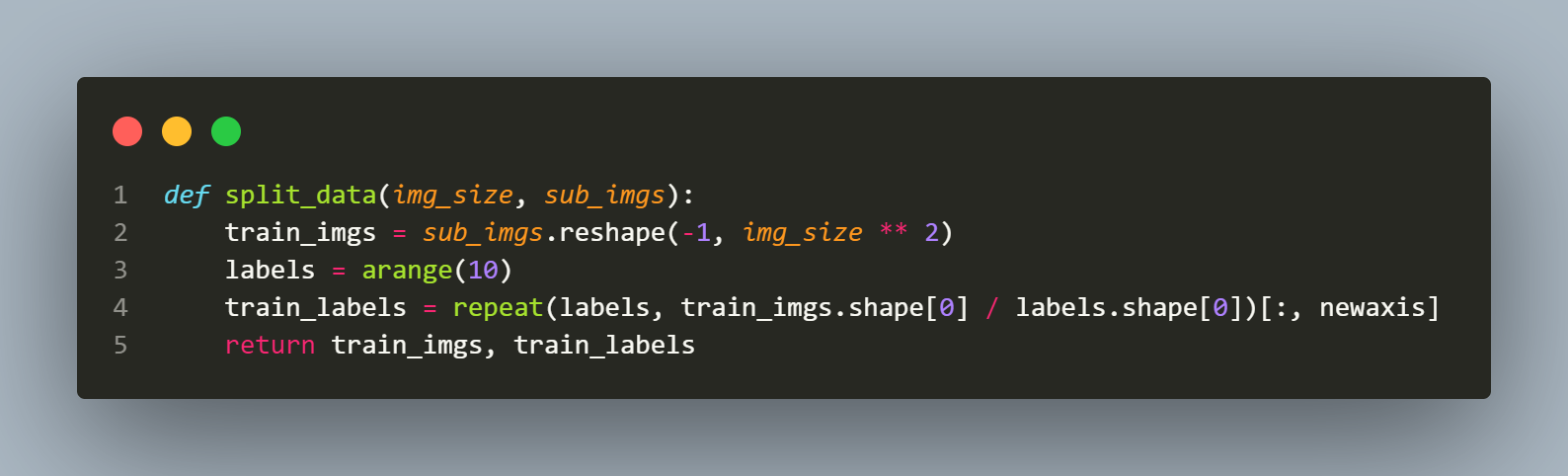
در مرحله بعدی با استفاده از .shape تعداد اعضای هر بعد را فراخوانی می‌کنیم و این مقدار را تقسیم بر اندازه عکس‌های کوچک می‌کنیم تا تعداد سطرها و تعداد ستون‌ها به دست بیاید.

vsplit(img, num\_rows)، تصویر را به صورت عمودی به قسمت هایی که توسط num\_rows مشخص شده تقسیم می کند. هر برش هنوز عرض کامل تصویر اصلی را دارد.

hsplit(row, num\_cols) سپس به صورت افقی هر یک از این برش های ردیف را به تصاویر کوچکتر که توسط num\_cols مشخص شده است تقسیم می کند.

در List Comprehension هر برش عمودی (از vsplit) را پردازش می‌شود و برای هر برش، hsplit را برای ایجاد شبکه نهایی تصاویر فرعی اعمال می کند. 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 داده های تصویر و برچسب های مربوطه را برای آموزش یادگیری ماشین آماده می کند. این تابع آرایه ای از تصاویر فرعی را به فرمت مسطح مناسب برای ورودی مدل تبدیل می کند و یک دنباله تکراری از برچسب ها برای این تصاویر ایجاد می کند. شکل (83) این تابع را نمایش می‌دهد.



شکل 83: آماده سازی داده‌ها برای آموزش

خط اول کد آرایه sub\_imgs را به یک آرایه دو بعدی تغییر شکل می دهد که در آن هر ردیف یک تصویر واحد را نشان می دهد. سپس تصاویر به بردارهایی با طول img\_size \*\* 2 مسطح می شوند، به این معنی که هر تصویر به یک آرایه یک بعدی حاوی تمام مقادیر پیکسل تبدیل می شود.

سپس arrange(10) یک آرایه از برچسب ها از 0 تا 9 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 train\_imgs.shape[0] / labels.shape[0] محاسبه می کند که چند مجموعه کامل از 0-9 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. [:, newaxis] آرایه را از یک بعدی به دو بعدی (بردار ستونی) تبدیل می کند.

## 2-4-آموزش رگرسیون لجستیک چند کلاسه

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

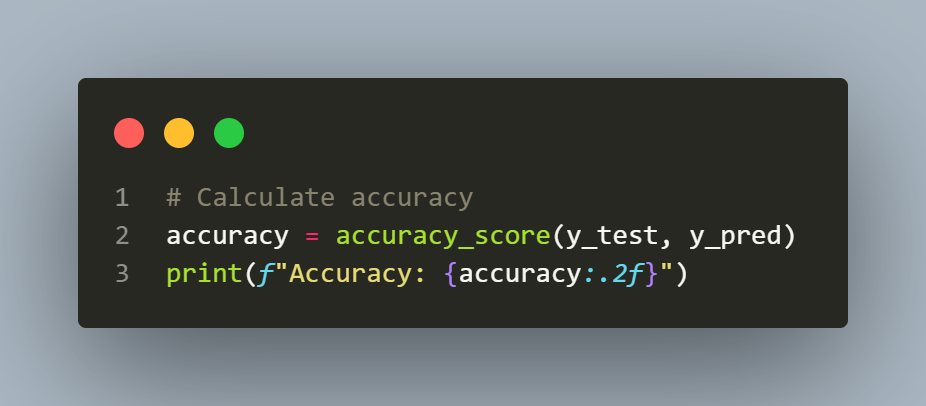
سپس از کلاس LogisticRegression یک شی می‌سازیم. آرگومان multi\_class را برابر با 'multinomial' قرار می‌دهیم. این کار باعث می‌شود که رگرسیون به صورت چند کلاسه عمل کند. حالت دیگر One vs All است که در آن تلاش می‌کنیم یک لیبل به خصوص را از بین بقیه لیبل‌ها تشخیص دهیم که در این مسئله کاربرد ندارد. از حل کننده lbfgs و تعداد تکرار 1000 استفاده می‌کنیم.

سپس با استفاده از متد .fit() و پاس دادن داده‌های آموزشی مدل را آموزش می‌دهیم. سپس با استفاده از متد .predict() و پاس دادن دیتاست تست، اجازه می‌دهیم مدل پیش‌بینی را انجام دهد. شکل(84) نحوه انجام این کار را نمایش می‌دهد.



شکل 84: آموزش و تست رگرسیون لجستیک

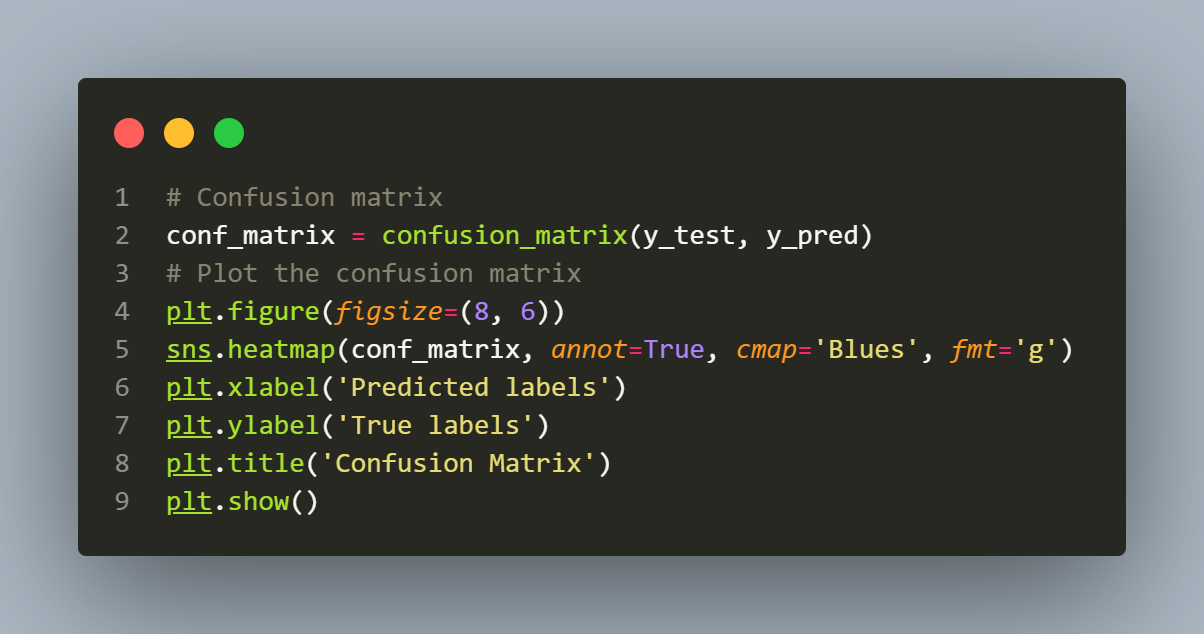
سپس با استفاده از تابع accuracy\_score و پاس دادن مقادیر پیش‌بینی و مقادیر اصلی، دقت را محاسبه می‌کنیم. شکل (85) نحوه انجام این کار را نمایش می‌دهد.



شکل 85: محاسبه دقت

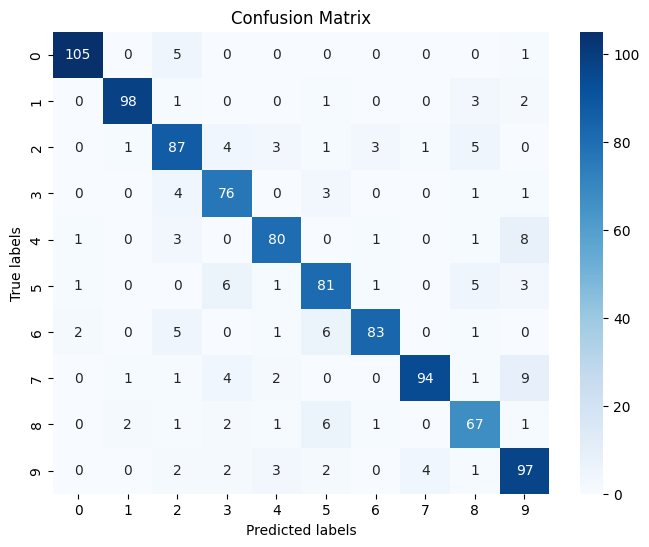
با استفاده از این روش دقت برابر با 0.87 به دست می‌آید که از دقت خواسته شده (0.78) بیشتر است.

سپس ماتریس درهم‌ریختگی را محاسبه می‌کنیم. شکل (86) نحوه انجام این کار را نشان می‌دهد.



شکل 86: نحوه محاسبه ماتریس درهم‌ریختگی

شکل (87) ماتریس درهم‌ریختگی به دست آمده را نشان می‌دهد.



شکل 87: ماتریس درهم‌ریختگی به دست آمده

با بررسی ماتریس درهم‌ریختگی مشاهده می‌شود که مدل نمی‌توان عدد هشت را به خوبی بقیه اعداد دسته‌بندی کند.