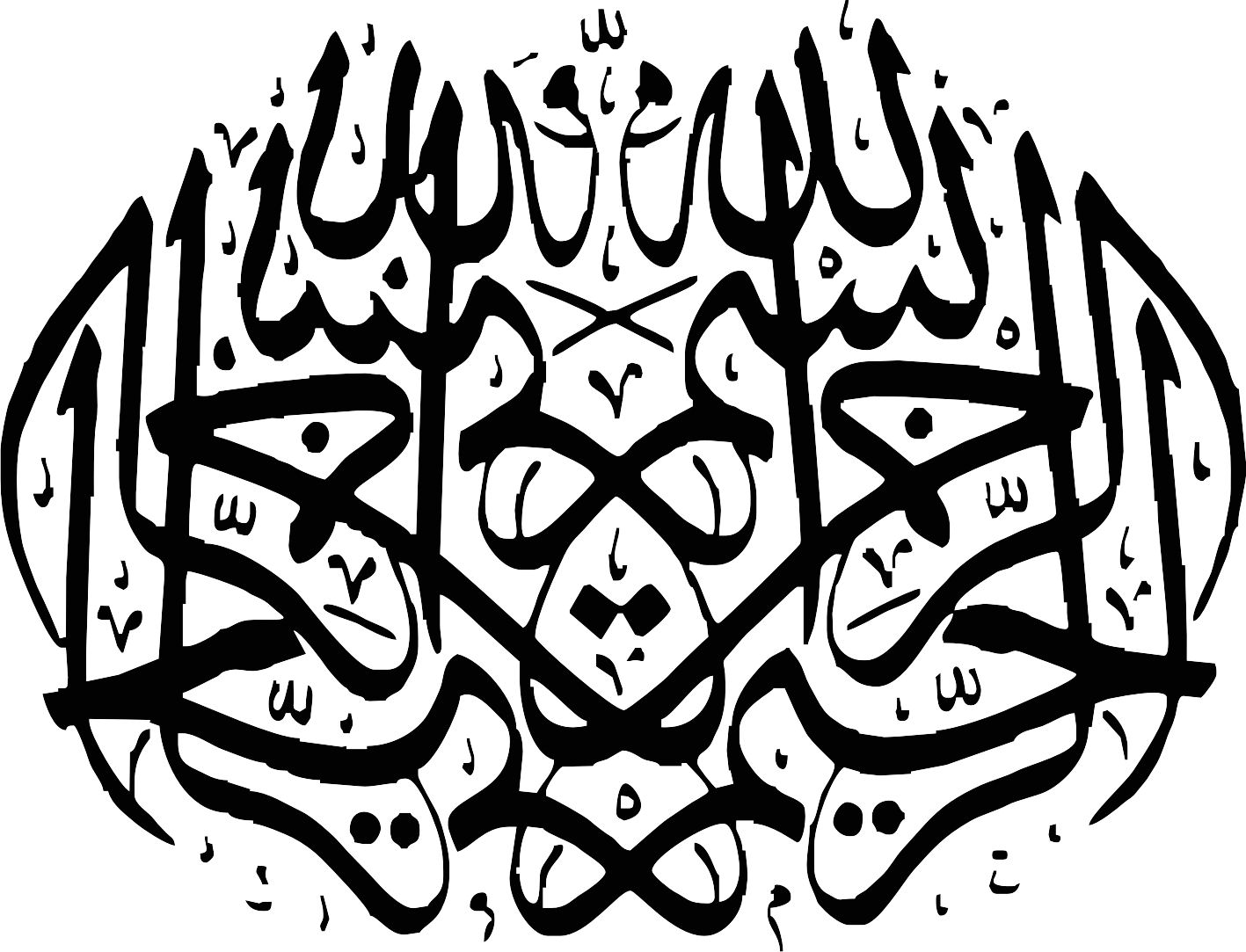


**گزارش تکلیف سوم درس یادگیری ماشین کاربردی**

**استاد درس: دکتر ناظرفرد**

تهیه کننده: سید نیما محمودیان

شماره دانشجویی: 402125005



فهرست مطالب

[1-پاسخ سوال اول 1](#_Toc164419980)

[1-1-لود کردن دیتاست، نمایش ده ردیف به صورت تصادفی، و بررسی کلی داده‌ها 1](#_Toc164419981)

[2-1-مصور سازی داده‌ها 4](#_Toc164419982)

[1-2-1-رسم و تفسیر هیستوگرام 4](#_Toc164419983)

[2-2-1- رسم نمودار دایره‌ای و تفسیر آن 6](#_Toc164419984)

[3-2-1-رسم و تفسیر نمودارهای جعبه‌ای 7](#_Toc164419985)

[4-2-1-رسم و تفسیر ماتریس همبستگی 9](#_Toc164419986)

[5-2-1-رسم و تفسیر نمودار همبستگی با ستون هدف 10](#_Toc164419987)

[3-1-تشخیص داده‌های پرت 12](#_Toc164419988)

[1-3-1-جایگزینی داده‌ها با حد پایین یا حد بالا 12](#_Toc164419989)

[2-3-1-حذف داده‌های پرت 15](#_Toc164419990)

[4-1-پر کردن مقادیر خالی 15](#_Toc164419991)

[2-4-1-تعریف شی پیش‌پردازش برای پر کردن مقادیر خالی با KNN 15](#_Toc164419992)

[4-4-1-تعریف شی پیش‌پردازش برای پر کردن مقادیر خالی با میانه 16](#_Toc164419993)

[5-1-انتخاب ویژگی 17](#_Toc164419994)

[1-5-1- انتخاب ویژگی با استفاده از روش PCA 17](#_Toc164419995)

[2-5-1- انتخاب ویژگی با روش Mutual Information 18](#_Toc164419996)

[2-پاسخ سوال دوم 23](#_Toc164419997)

[1-2-پر کردن مقادیر خالی 24](#_Toc164419998)

[2-2-استفاده از SelectKBest 27](#_Toc164419999)

[3-2-آموزش KNN 27](#_Toc164420000)

[4-2-آموزش و تست مدل با فاصله‌های متفاوت 29](#_Toc164420001)

[1-4-2-فاصله منهتن 29](#_Toc164420002)

[2-4-2-فاصله کسینوسی 30](#_Toc164420003)

[5-2-انجام پیش‌پردازش بیشتر برای رسیدن به نتایج بهتر 30](#_Toc164420004)

[3-پاسخ سوال سوم 32](#_Toc164420005)

[1-3-پیش پردازش داده‌ها 32](#_Toc164420006)

[1-1-3-حذف کاراکترهای \n 32](#_Toc164420007)

[2-1-3-حذف کاراکترهای انگلیسی 32](#_Toc164420008)

[3-1-3-حذف علائم نگارشی 32](#_Toc164420009)

[5-1-3- نرمال کردن متن‌ها 34](#_Toc164420010)

[6-1-3-توکن کردن متن 34](#_Toc164420011)

[7-1-3-حذف ایست‌واژه‌ها (کلمات پالایشی یا stop words) 34](#_Toc164420012)

[2-3-آموزش KNN و پیدا کردن بهترین K 35](#_Toc164420013)

[4-پاسخ سوال چهارم 38](#_Toc164420014)

[1-4-وارد کردن و پیش‌پردازش داده‌ها 38](#_Toc164420015)

[2-4-آموزش KNN 41](#_Toc164420016)

[3-4-استفاده از Cross Validation 41](#_Toc164420017)

فهرست شکل‌ها

[شکل 1: وارد کردن کتابخانه‌های مورد نیاز 1](#_Toc164420038)

[شکل 2: خواندن دیتاست 1](#_Toc164420039)

[شکل 3: کد مربوط به نمایش ده سطر تصادفی 2](#_Toc164420040)

[شکل 4: ده سطر تصادفی نمایش داده شده 2](#_Toc164420041)

[شکل 5: کد مربوط به نشان دادن تعداد مقادیر گمشده 2](#_Toc164420042)

[شکل 6: تعداد مقادیر خالی در هر سطر 2](#_Toc164420043)

[شکل 7: کد بررسی تعداد سطرهای تکراری 3](#_Toc164420044)

[شکل 8: کد بررسی تعداد صفرها در هر ستون 3](#_Toc164420045)

[شکل 9: بررسی تعداد و جمعیت موجود در سطرهای گسسته 3](#_Toc164420046)

[شکل 10: خروجی کد شکل 9 3](#_Toc164420047)

[شکل 11: انتقال سطر هدف به سطر آخر 4](#_Toc164420048)

[شکل 12: جداسازی داده‌های عددی و دسته‌ای در دو لیست جدا 4](#_Toc164420049)

[شکل 13: کد مربوط به رسم هیستوگرام 5](#_Toc164420050)

[شکل 14: هیستوگرام‌های رسم شده 5](#_Toc164420051)

[شکل 15: کد رسم نمودار دایره‌ای 6](#_Toc164420052)

[شکل 16: نمودار دایره‌ای 7](#_Toc164420053)

[شکل 17: کد رسم کننده نمودارهای جعبه‌ای 8](#_Toc164420054)

[شکل 18: نمودارهای جعبه‌ای 8](#_Toc164420055)

[شکل 19: آماده‌سازی دیتافریم برای محاسبه ماتریس همبستگی 9](#_Toc164420056)

[شکل 20: رسم ماتریس همبستگی 9](#_Toc164420057)

[شکل 21: ماتریس همبستگی رسم شده 10](#_Toc164420058)

[شکل 22: کد رسم کننده نمودار میزان همبستگی با سطر هدف 11](#_Toc164420059)

[شکل 23: نمودار میزان همبستگی با سطر هدف 11](#_Toc164420060)

[شکل 24: جایگزینی داده‌های پرت با حد بالا و حد پایین 13](#_Toc164420061)

[شکل 25: ایجاد شی از کلاس ColumnTransformer 13](#_Toc164420062)

[شکل 26: پیش‌پردازش داده‌های آموزشی 13](#_Toc164420063)

[شکل 27: پیش‌پردازش داده‌های تست 14](#_Toc164420064)

[شکل 28: تعریف مدل 14](#_Toc164420065)

[شکل 29: آموزش و تست مدل 14](#_Toc164420066)

[شکل 30: محاسبه و نمایش R2 15](#_Toc164420067)

[شکل 31: حذف داده‌های پرت 15](#_Toc164420068)

[شکل 32: ایجاد شی از کلاس پایپ‌لاین 16](#_Toc164420069)

[شکل 33: ایجاد شی از کلاس ColumnTransformer 16](#_Toc164420070)

[شکل 34: پایپ‌لاین به روز شده 17](#_Toc164420071)

[شکل 35: پایپ‌لاین ایجاد شده برای PCA 17](#_Toc164420072)

[شکل 36: اجرای پایپ‌لاین روی داده‌های آموزشی 18](#_Toc164420073)

[شکل 37: انتخاب نه ویژگی برتر با استفاده از Mutual Information 18](#_Toc164420074)

[شکل 38: رسم نمودار از میانگین MSE هر درجه 19](#_Toc164420075)

[شکل 39: ارزیابی درجه‌های رگرسیون با استفاده از CV 20](#_Toc164420076)

[شکل 40: نمودار MSE به ازای درجه 20](#_Toc164420077)

[شکل 41: پایپ‌لاین پیش‌پردازش و آموزش رگرسیون 21](#_Toc164420078)

[شکل 42: ایجاد ویژگی جدید با دو ویژگی قدیمی 21](#_Toc164420079)

[شکل 43: نمودار MSE بر درجه برای حالت جدید دیتاست 21](#_Toc164420080)

[شکل 44: نمودار MSE براساس درجه در حالت داشتن ویژگی Rooms\_per\_household 22](#_Toc164420081)

[شکل 45: خروجی متد .nunique() 23](#_Toc164420082)

[شکل 46: استفاده از method chaining 23](#_Toc164420083)

[شکل 47: تعداد مقادیر خالی در هر ستون 24](#_Toc164420084)

[شکل 48: ذخیره نام ستون‌ها در دو لیست مجزا 24](#_Toc164420085)

[شکل 49: ایجاد پایپ‌لاین برای داده‌های عددی 24](#_Toc164420086)

[شکل 50: پایپ‌لاین برای مقادیر دسته‌ای 25](#_Toc164420087)

[شکل 51: شی ساخته شده از ColumnTransformer 25](#_Toc164420088)

[شکل 52: پایپ‌لاین نهایی 25](#_Toc164420089)

[شکل 53: انجام Cross Validation و محاسبه R2 26](#_Toc164420090)

[شکل 54: پایپ‌لاین به روز شده برای مقادیر عددی 26](#_Toc164420091)

[شکل 55: انتخاب ویژگی با استفاده از SelectKBest 27](#_Toc164420092)

[شکل 56: Oversample کردن داده‌های آموزشی 28](#_Toc164420093)

[شکل 57: ماتریس درهم ریختگی مدل آموزش داده شده 28](#_Toc164420094)

[شکل 58: پایپ‌لاین آموزش/تست با فاصله متهتن 29](#_Toc164420095)

[شکل 59: ماتریس درهم ریختگی KNN با فاصله منهتن 29](#_Toc164420096)

[شکل 60: پایپ‌لاین تست و آموزش KNN با فاصله کسینوسی 30](#_Toc164420097)

[شکل 61: ماتریس درهم‌ریختگی مدل KNN با فاصله کسینوسی 30](#_Toc164420098)

[شکل 62: حذف داده‌های پرت 31](#_Toc164420099)

[شکل 63: تعداد ایمیل‌های با لیبل سالم و اسپم 32](#_Toc164420100)

[شکل 64: حذف کاراکتر \n 32](#_Toc164420101)

[شکل 65: حذف کاراکترهای انگلیسی 33](#_Toc164420102)

[شکل 66: حذف علائم نگارشی 33](#_Toc164420103)

[شکل 67: حذف اعداد با استفاده از رجکس 33](#_Toc164420104)

[شکل 68: نحوه انجام نرمال سازی متن 34](#_Toc164420105)

[شکل 69: توکن سازی متن 34](#_Toc164420106)

[شکل 70: پیدا کردن و حذف ایست‌واژه‌ها 35](#_Toc164420107)

[شکل 71: بازتولید متن‌ها از کلمه‌ها 35](#_Toc164420108)

[شکل 72: استفاده از روش TF-IDF 36](#_Toc164420109)

[شکل 73: تعریف مدل و دیکشنری پارامترها 36](#_Toc164420110)

[شکل 74: کد رسم نمودار دقت بر اساس K 37](#_Toc164420111)

[شکل 75: نمودار دقت بر اساس K 37](#_Toc164420112)

[شکل 76: وارد کردن کتابخانه‌های مورد نیاز 38](#_Toc164420113)

[شکل 77: تعریف الگوی دریافتی 38](#_Toc164420114)

[شکل 78: خواندن و انتخاب تصادفی فایل‌ها 39](#_Toc164420115)

[شکل 79: پیش‌پردازش یک به یک عکس‌ها 39](#_Toc164420116)

[شکل 80: تابع خواندن و پیش‌پردازش عکس‌ها 40](#_Toc164420117)

[شکل 81: فراخوانی و پیش‌پردازش عکس‌ها و ایجاد دیتاست نهایی 40](#_Toc164420118)

[شکل 82: کد نمایش گزارش عملکرد 41](#_Toc164420119)

[شکل 83: گزارش عملکرد مدل KNN 41](#_Toc164420120)

[شکل 84: دیکشنری محاسبه معیارهای ارزیابی 42](#_Toc164420121)

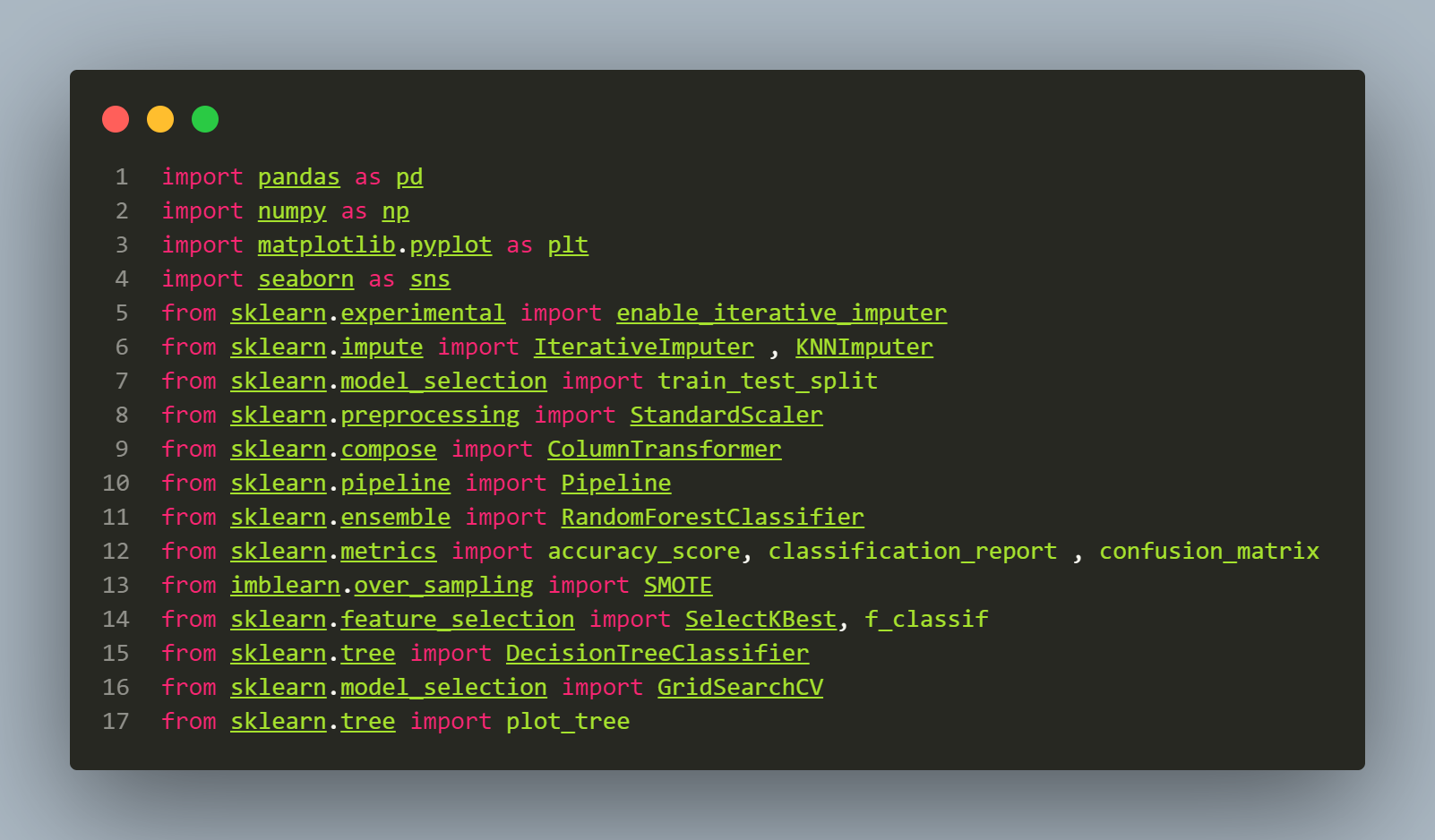
[شکل 85: انجام Cross Validation 42](#_Toc164420122)

[شکل 86: نتایج به دست آمده از Cross Validation 42](#_Toc164420123)

# 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.

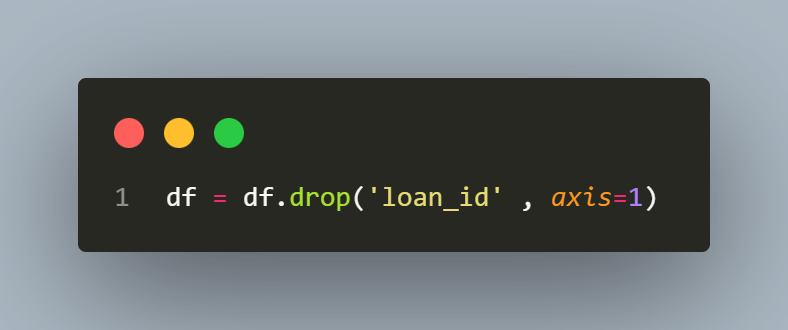


شکل 1: وارد کردن کتابخانه‌های مورد نیاز



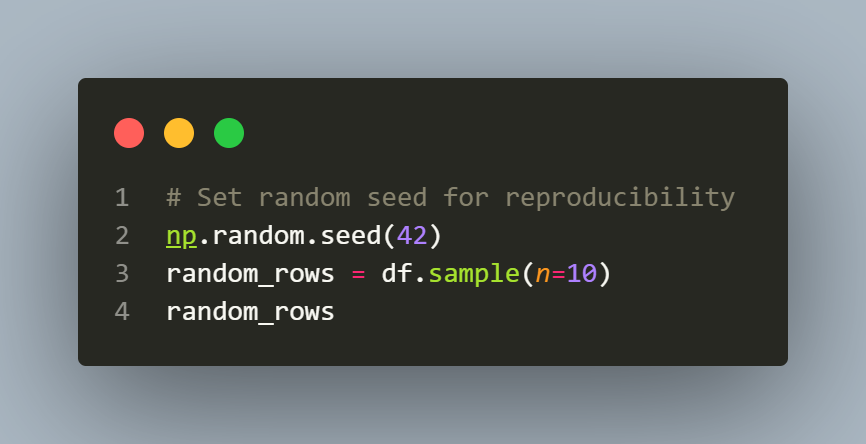
شکل 2: خواندن دیتاست

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.



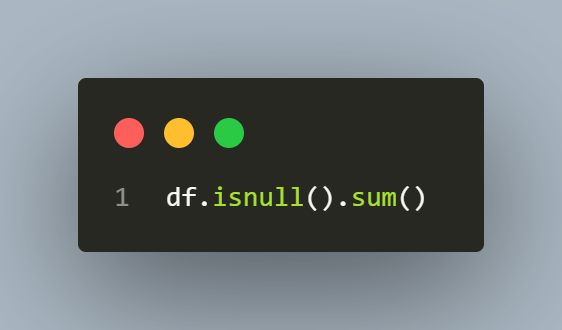
شکل 4: کد مربوط به نمایش ده سطر تصادفی

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.



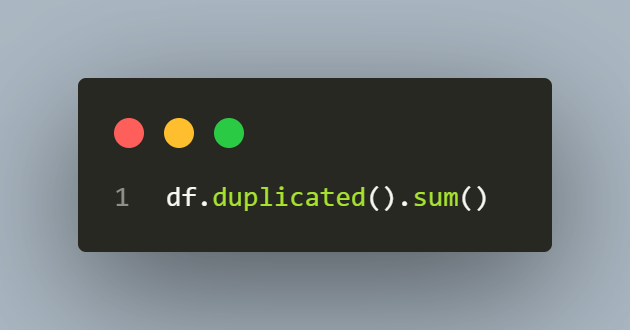
شکل 6: کد مربوط به نشان دادن تعداد مقادیر گمشده

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

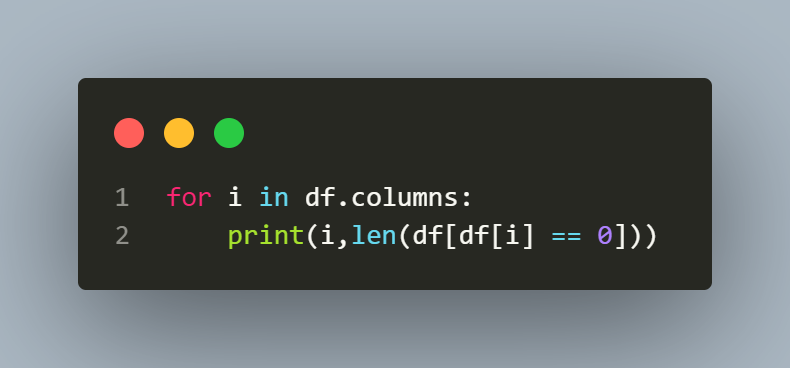


شکل 7: تعداد مقادیر خالی در هر سطر

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.



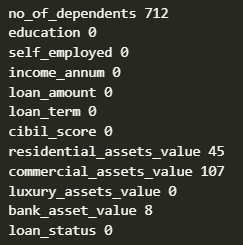
شکل 8: کد بررسی تعداد سطرهای تکراری



شکل 9: کد بررسی تعداد صفرها در هر ستون

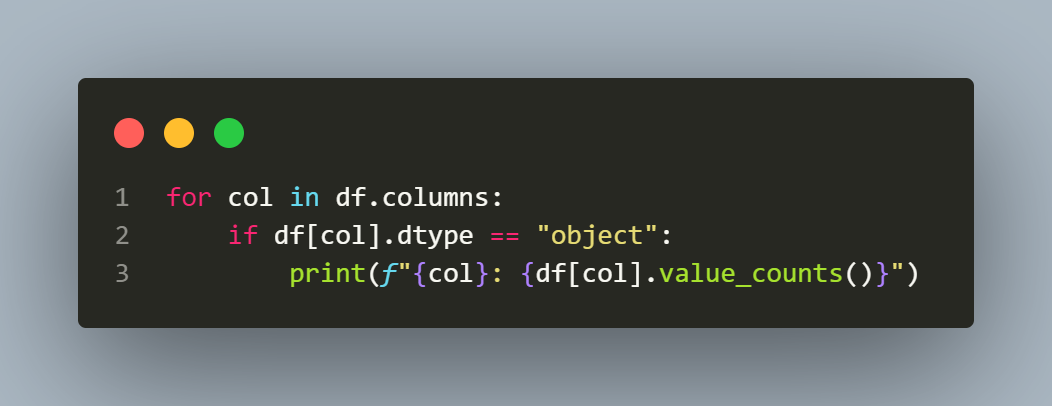
با بررسی خروجی این دو کد، متوجه می‌شویم که هیچ سطر تکراری در دیتافریم وجود ندارد. ولی مشاهده می‌شود که سه تا از ستون‌ها دارای مقادیر صفر هستند. شکل (10) خروجی کد شکل 9 را نمایش می‌دهد.

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.

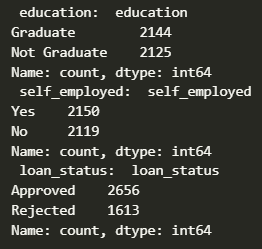


شکل 10: تعداد مقادیر صفر در هر سطر

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.

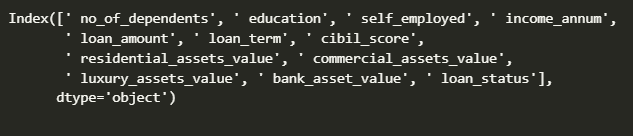


شکل 11: بررسی تعداد و جمعیت موجود در سطرهای گسسته



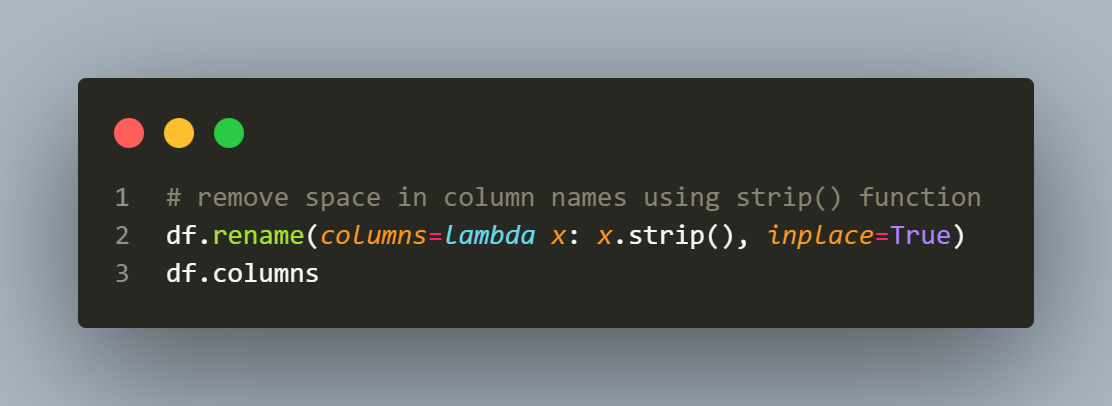
شکل 12: خروجی کد شکل 11

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



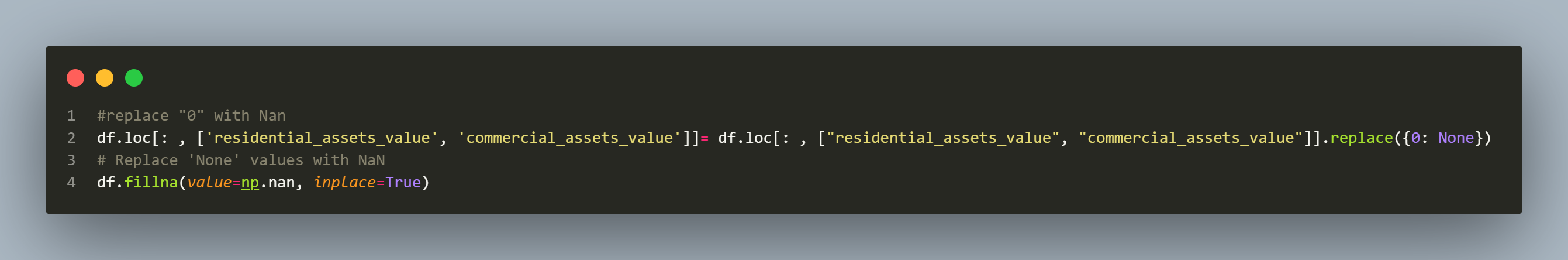
شکل 13: خروجی اتریبیوت df.columns

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.



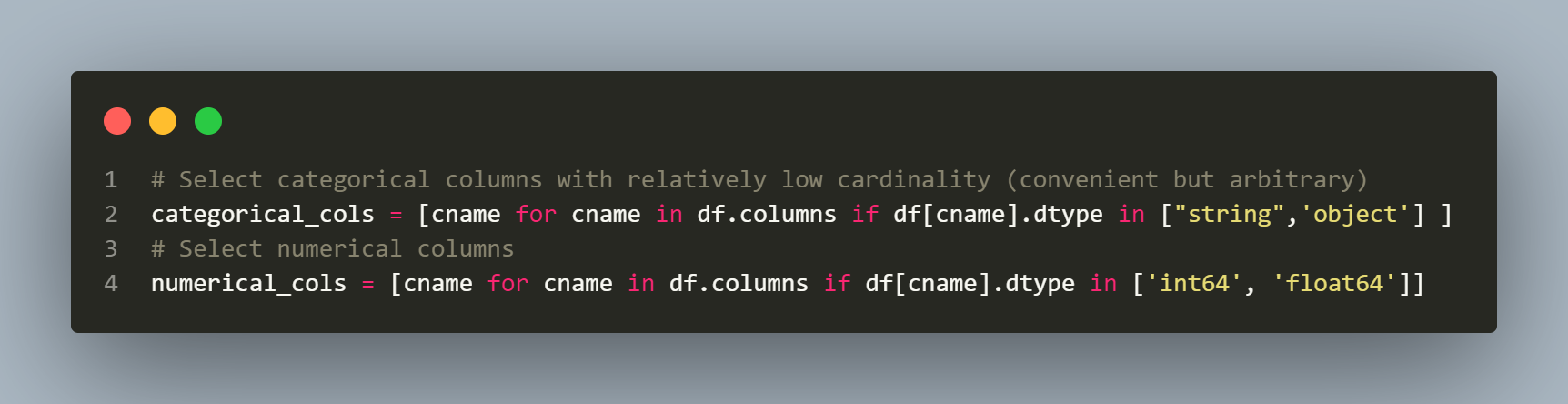
شکل 14: کد حذف فاصله از ستون‌ها

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.

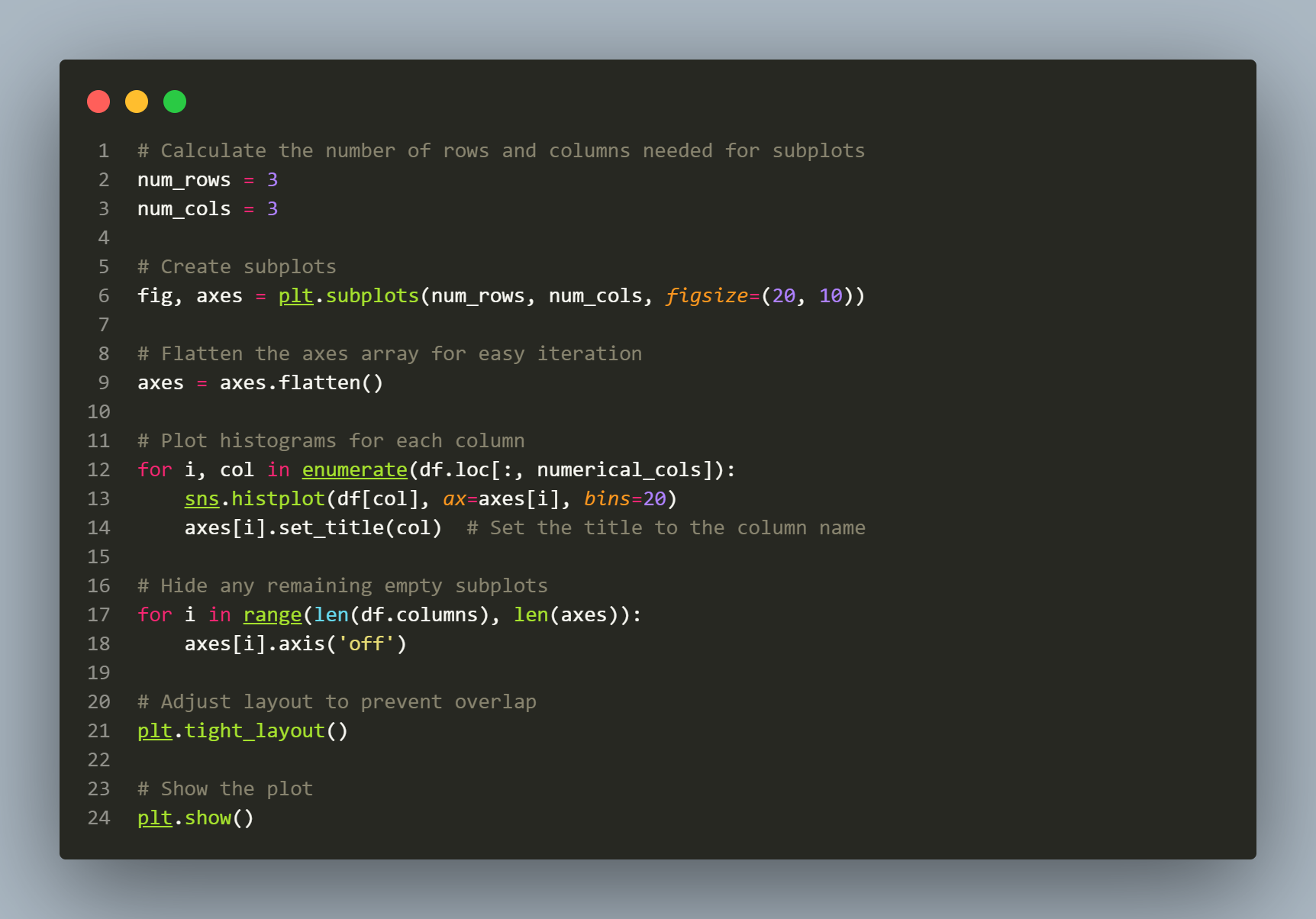
سپس حلقه روی هر ستون مشخص شده در لیست numerical\_cols از df تکرار می شود. برای هر ستون:

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.

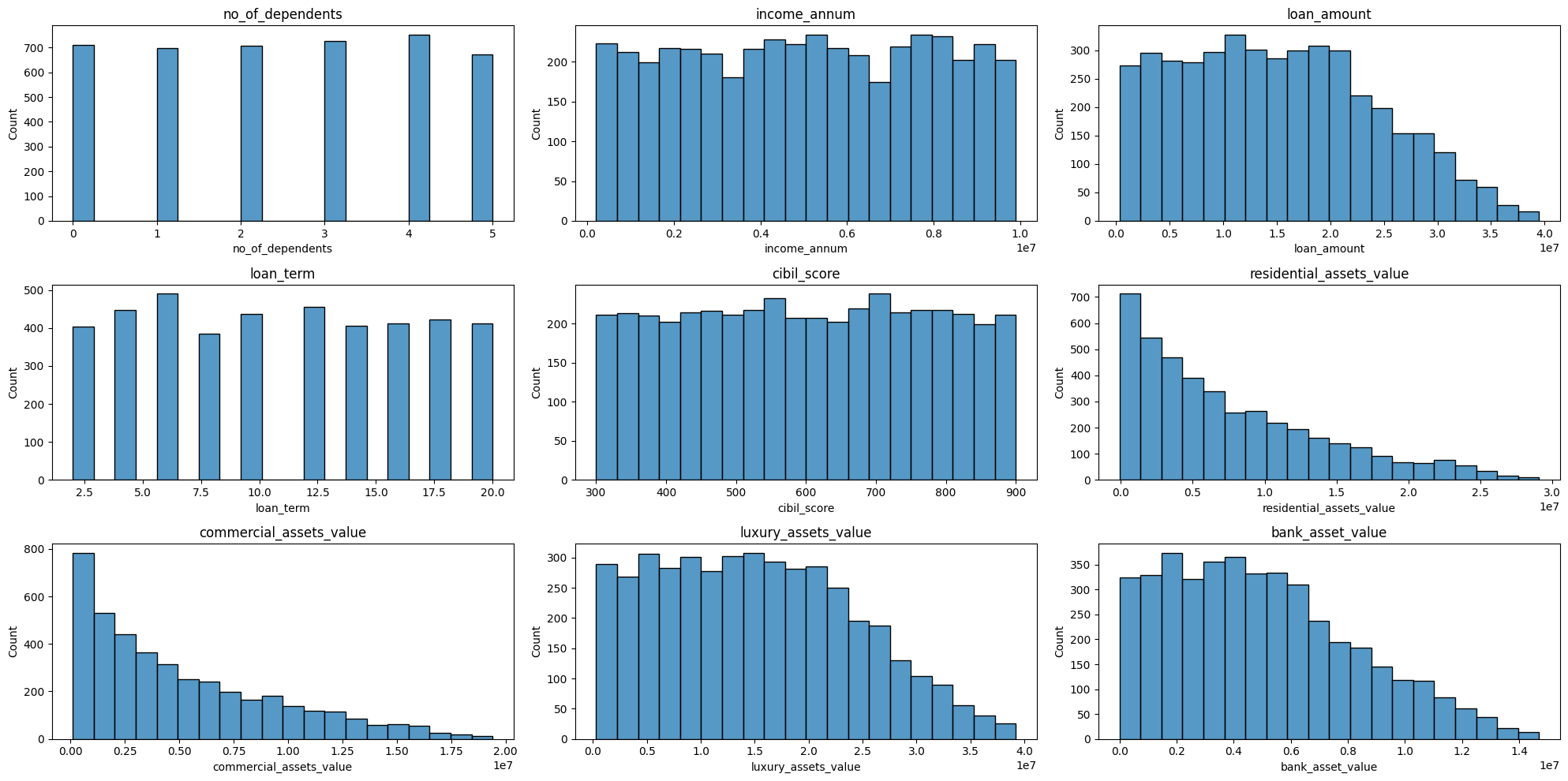
()plt.tight\_layout، نمودارهای فرعی و برچسب ها را طوری تنظیم می کند که به خوبی در ناحیه شکل قرار گیرند و روی هم قرار نگیرند.

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



شکل 17: کد مربوط به رسم هیستوگرام

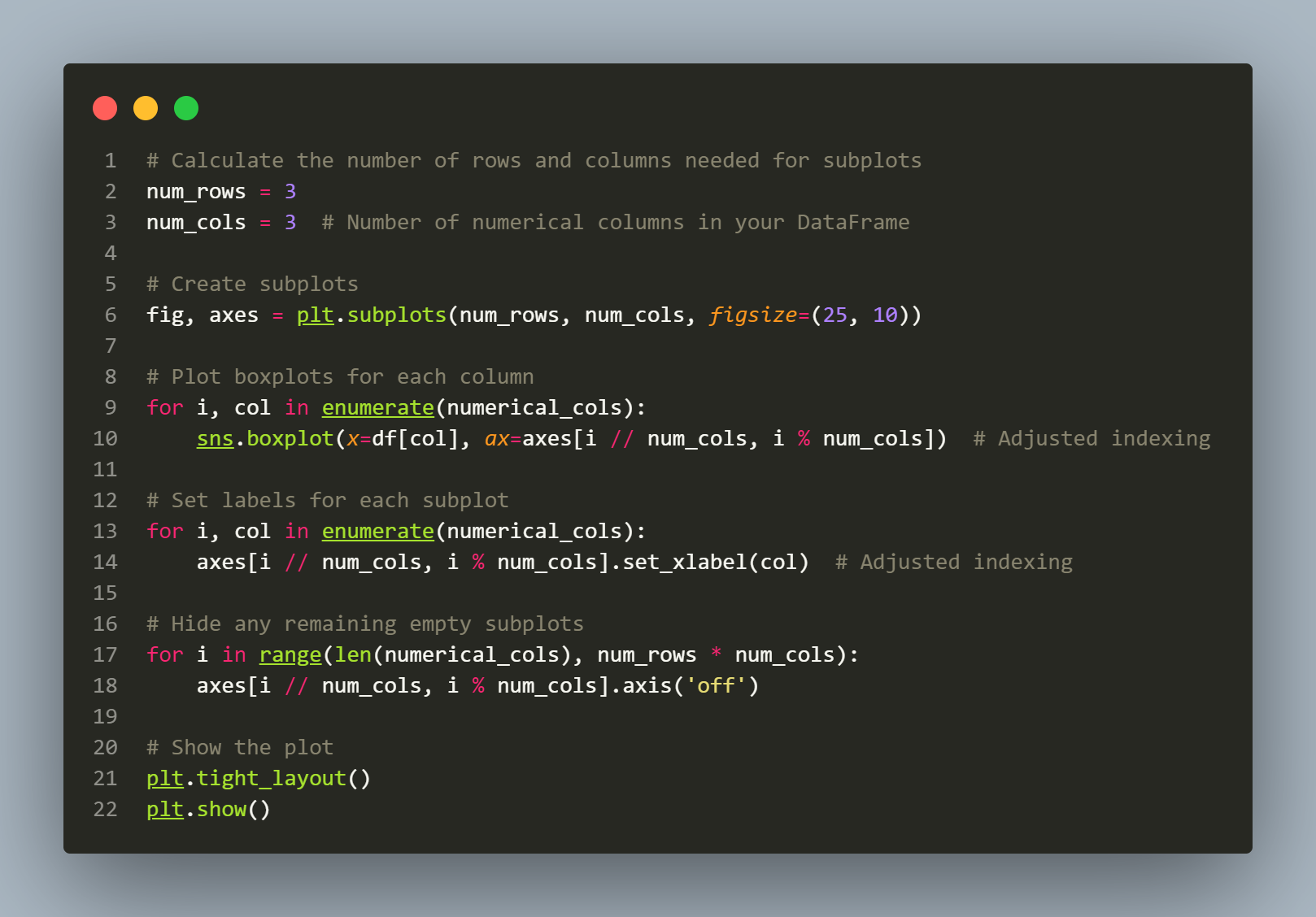
همچنین شکل (18) هیستوگرام‌های رسم شده را نمایش می‌دهد.



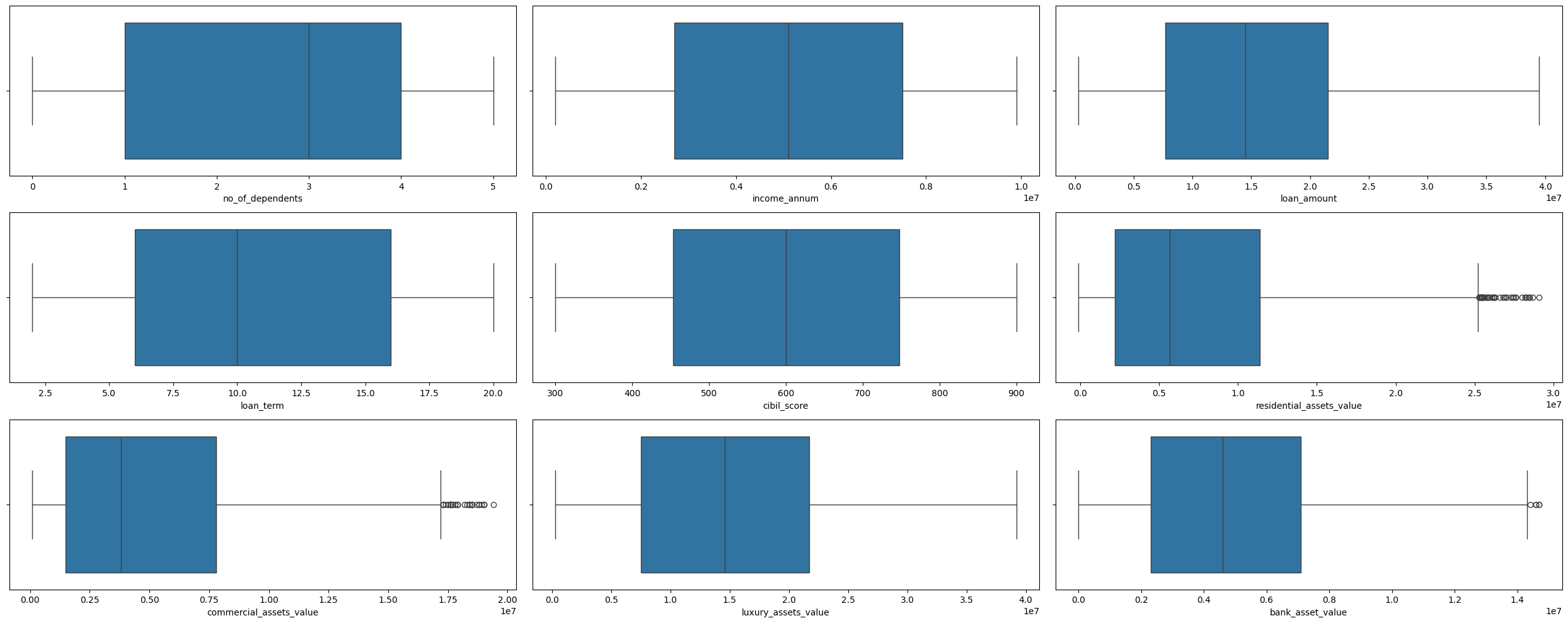
شکل 18: هیستوگرام‌های رسم شده

### 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

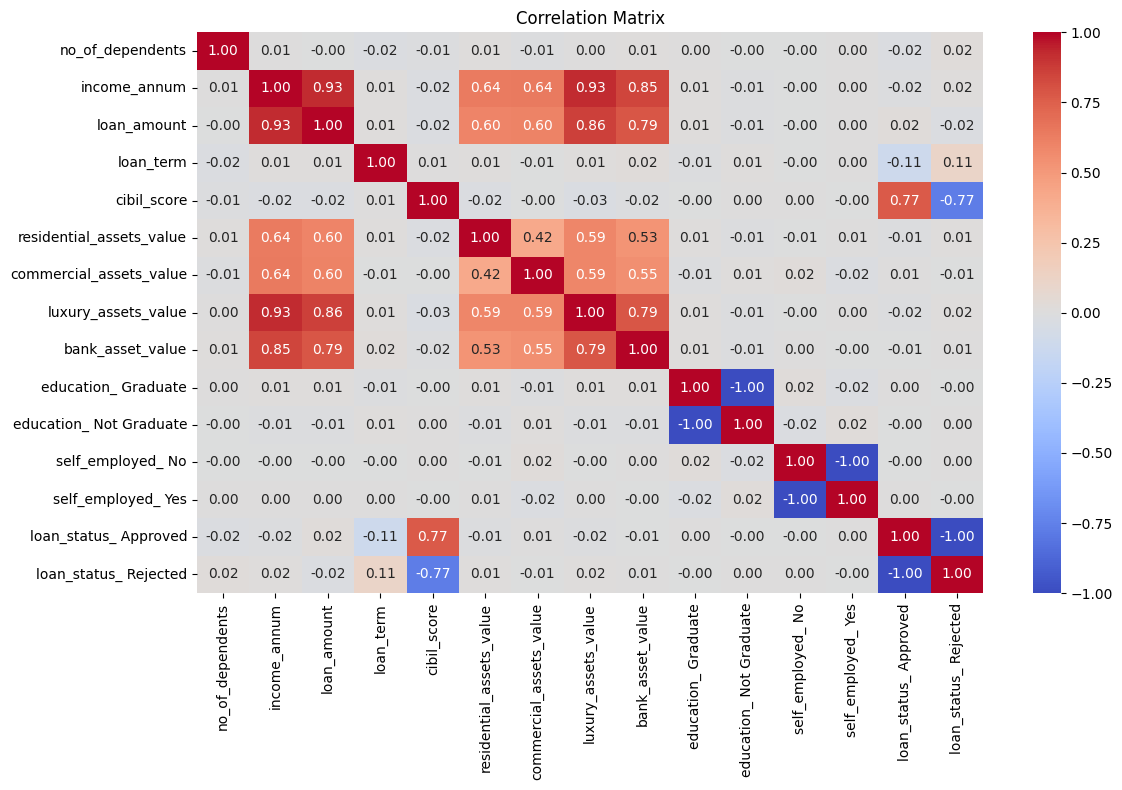
### 3-2-1-رسم و تفسیر ماتریس همبستگی

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.

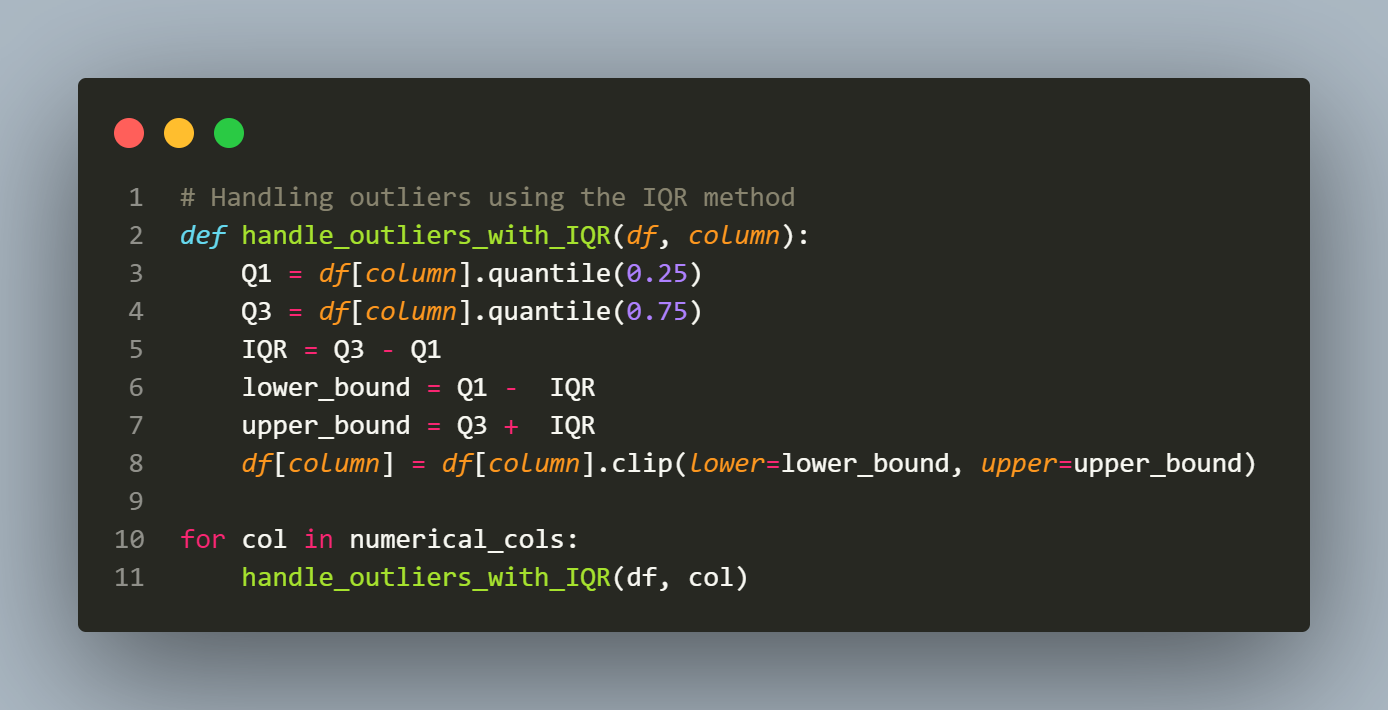


شکل 22: ماتریس همبستگی رسم شده

## 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.



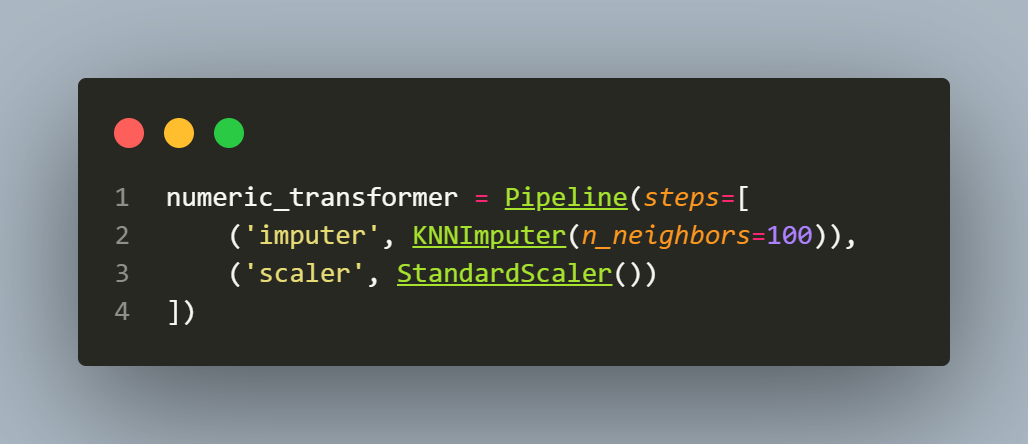
شکل 23: حذف داده‌های پرت با روش IQR

## 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

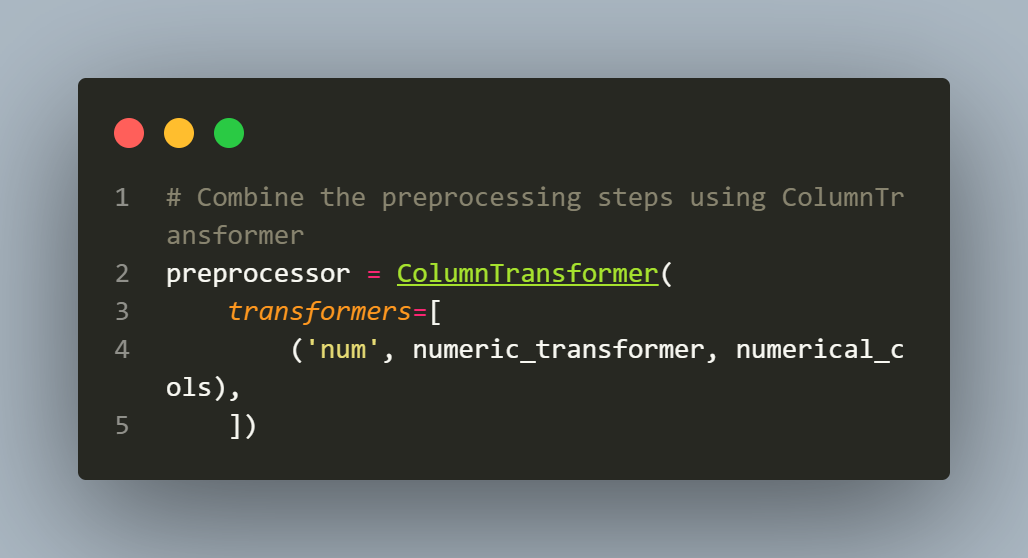
### 1-4-1-تعریف شی پیش‌پردازش برای پر کردن مقادیر خالی با Iterative Imputer

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

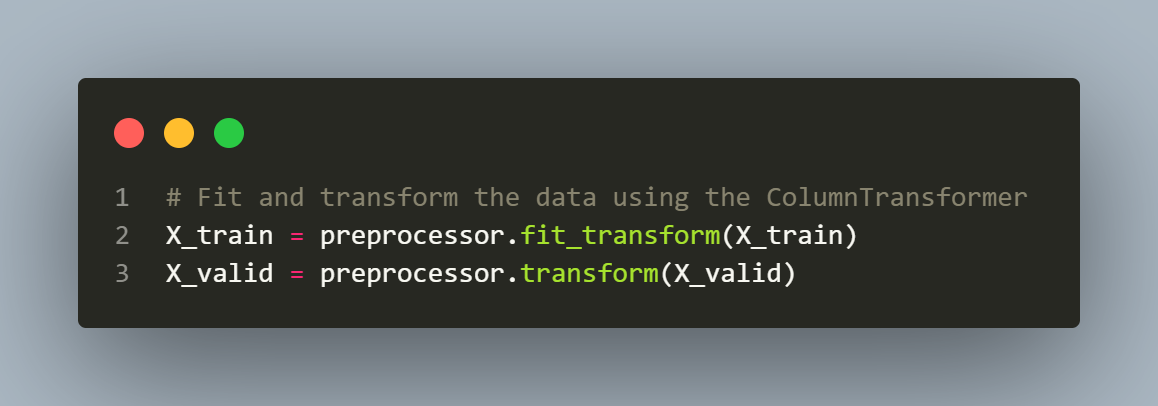
سپس یک شی از کلاس ColumnTransformer() ایجاد می‌کنیم در یک تاپل به ترتیب نام ترنسفورمر، شی ساخته شده از پایپ‌لاین، و ستون‌هایی که می‌خواهیم تغییرات روی آنها اعمال شوند را قرار می‌دهیم. شکل (25) ترنسفورمر ایجاد شده را نمایش می‌دهد.



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.

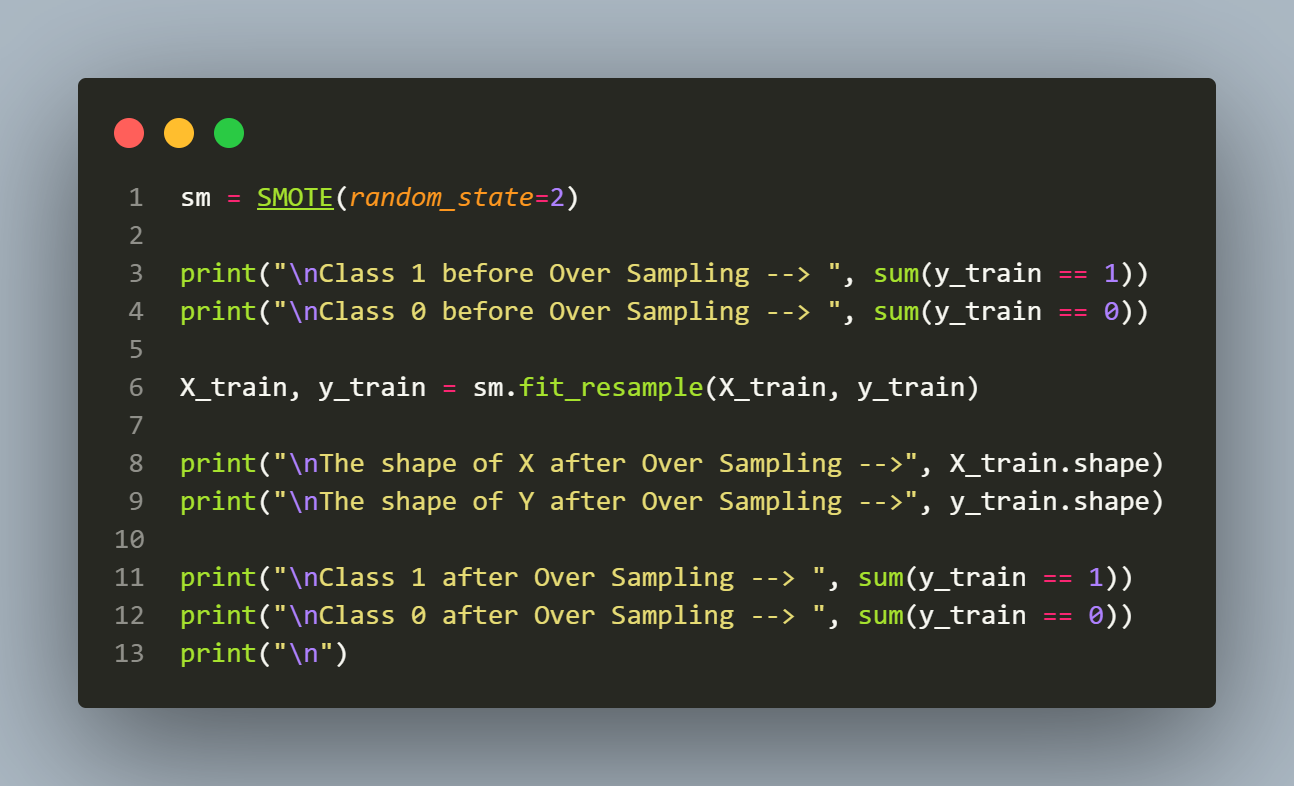
سپس برای پیش‌پردازش داده‌های تست، از متد transform() شی استفاده می‌کنیم. دلیل استفاده از این متد این است که این متد با استفاده از پارامترهای یادگرفته شده از داده‌های آموزشی، عملیات را روی داده‌های تستی انجام می‌دهد. توجه شود که اینجا از متد قبلی استفاده نمی‌شود زیرا فرض بر این است که پارامترهای داده‌های 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

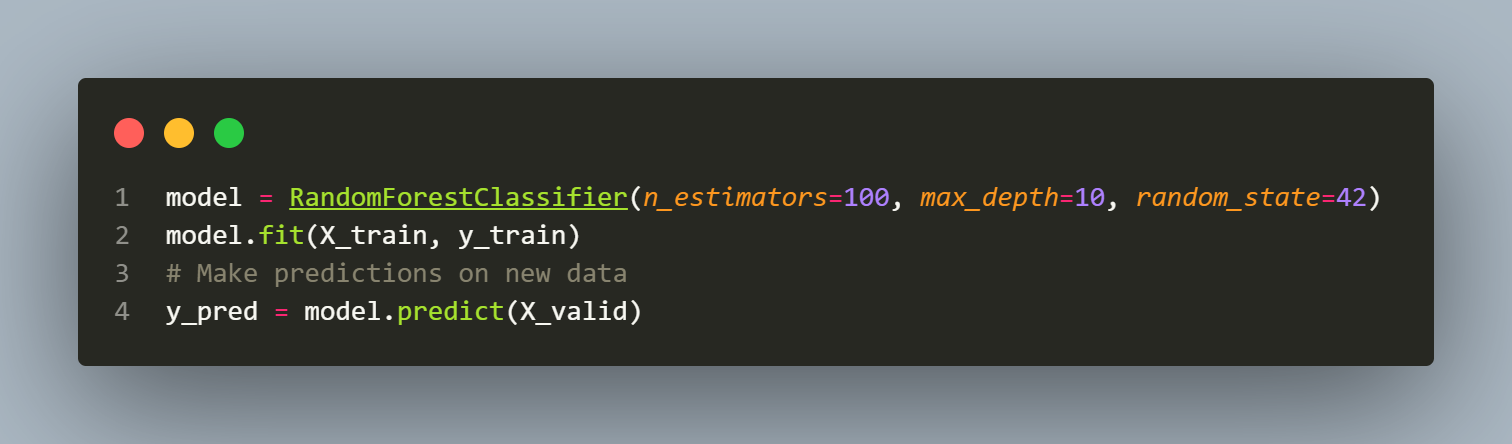
از آنجایی که دیتاست imbalance است، در اولین اقدام باید تعداد لیبل‌های صفر و تعداد لیبل‌های یک را در دیتاست آموزشی برابر کرد. برای این منظور از روش SMOTE استفاده می‌کنیم. این روش یکی از روش‌های oversampling است که از لیبلی که تعداد کمتری دارد داده‌های مصنوعی تولید می‌کند تا تعداد هر دو لیبل با هم برابر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



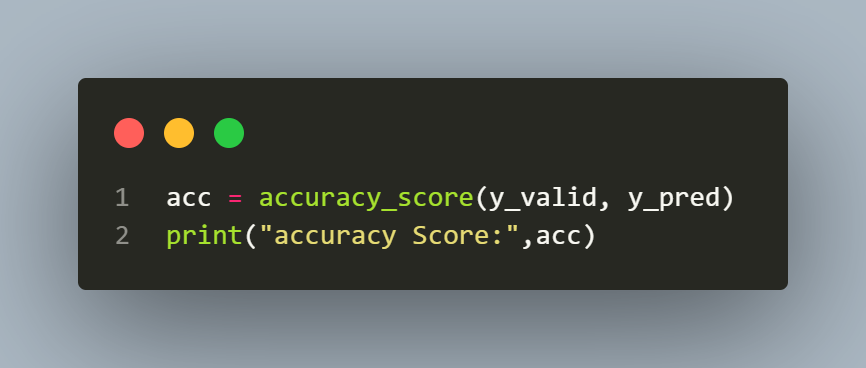
شکل 27: Oversample کردن داده‌های آموزشی

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.

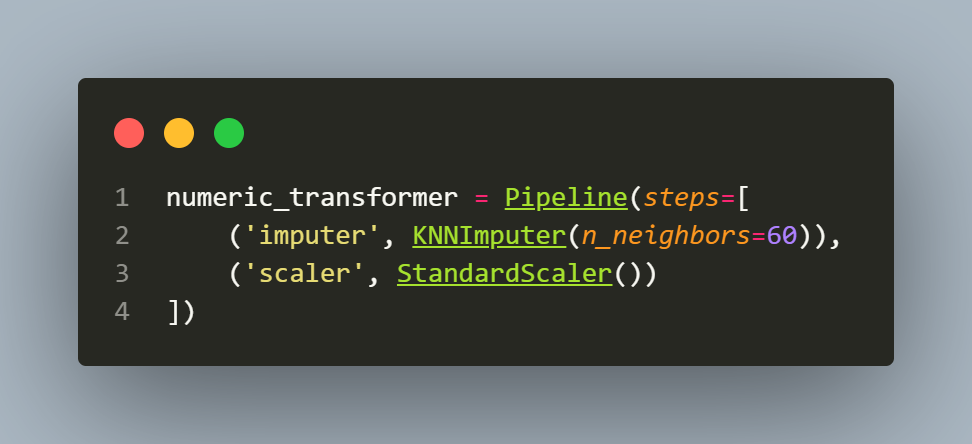


شکل 29: محاسبه دقت

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

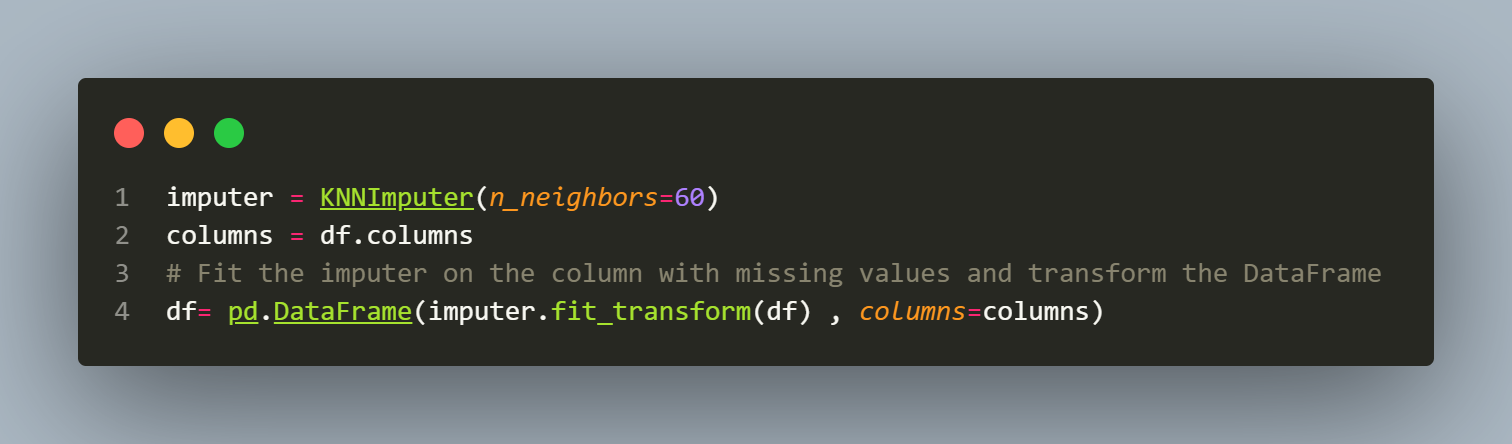
حال شی تعریف شده برای پیش‌پردازش را به نحوی تغییر می‌دهیم که مقادیر خالی را با KNN پر کند. برای این منظور، در شی پایپ‌لاین، از کلاس KNNImputer() به جای کلاس IterativeImputer() با آرگومان n\_neighbors=60 را پاس می‌دهیم. شکل (30) پایپ‌لاین به روز شده را نمایش می‌دهد.



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.

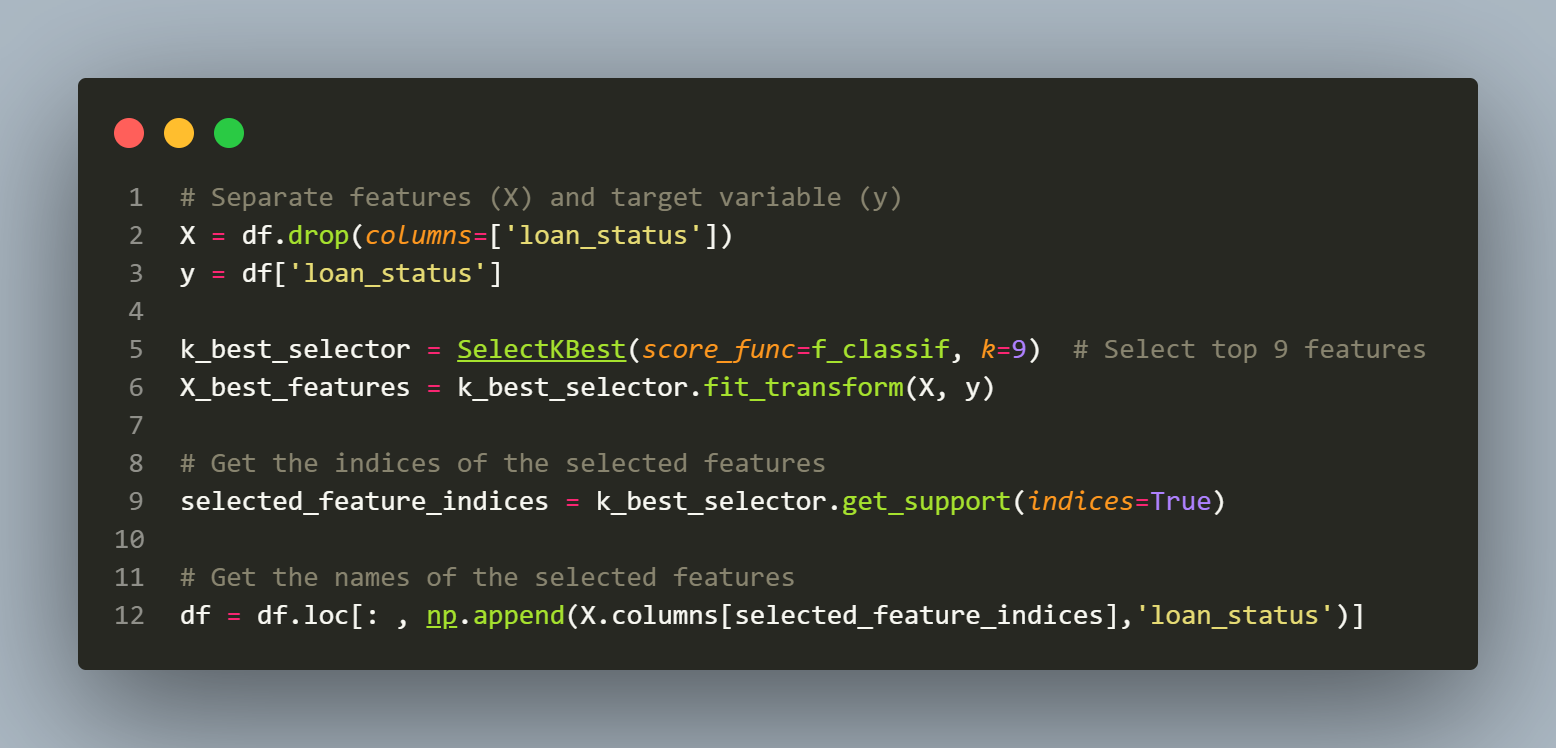


شکل 31: پر کردن مقادیر خالی با KNN

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.



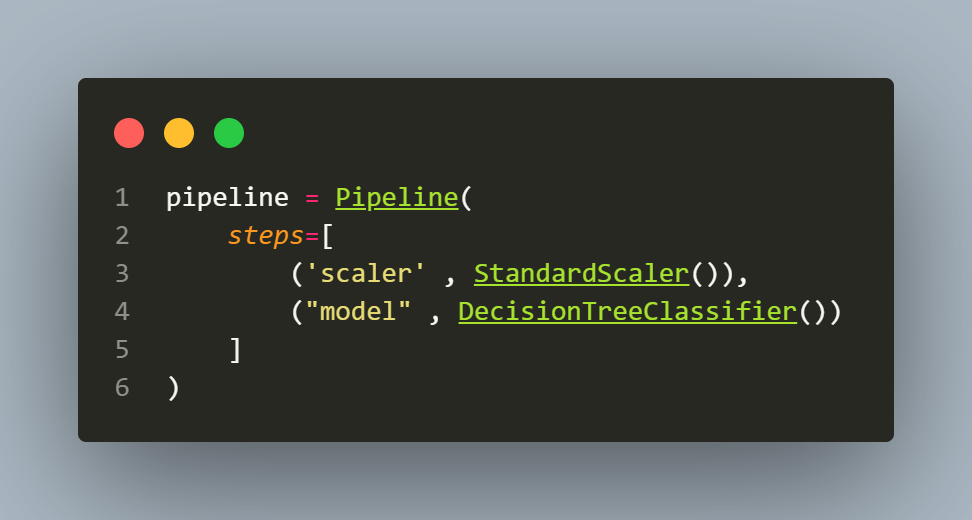
شکل 32: انتخاب ویژگی‌ها با استفاده از SelectKBest()

## 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.

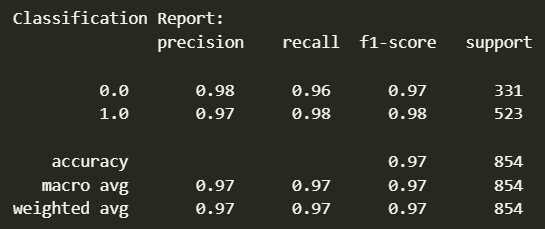
### 1-6-1-آموزش مدل با پارامترهای پیشفرض

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

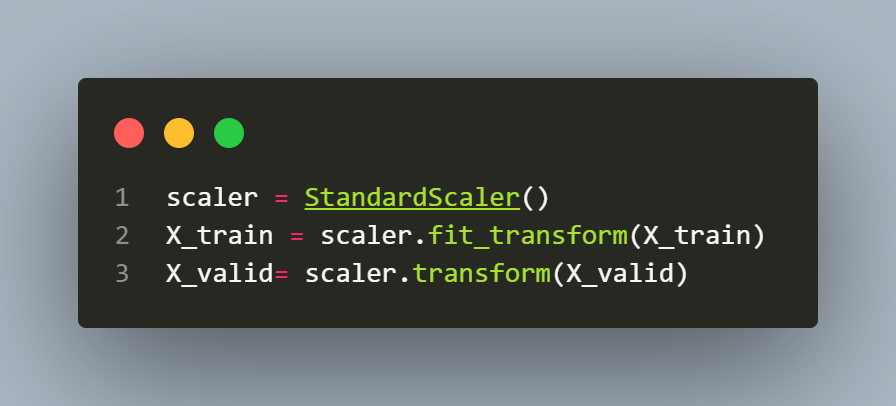
سپس با استفاده از متد .fit() از شی پایپ‌لاین و پاس دادن دیتاست آموزشی، استاندارد سازی و آموزش مدل را انجام می‌دهیم. همچنین با پاس دادن دیتاست تست به متد .predict() داده‌های تست را استانداردسازی می‌کنیم و سپس اجازه می‌دهیم مدل پیش‌بینی را انجام دهد. حال با استفاده از تابع classification\_report() معیارهای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

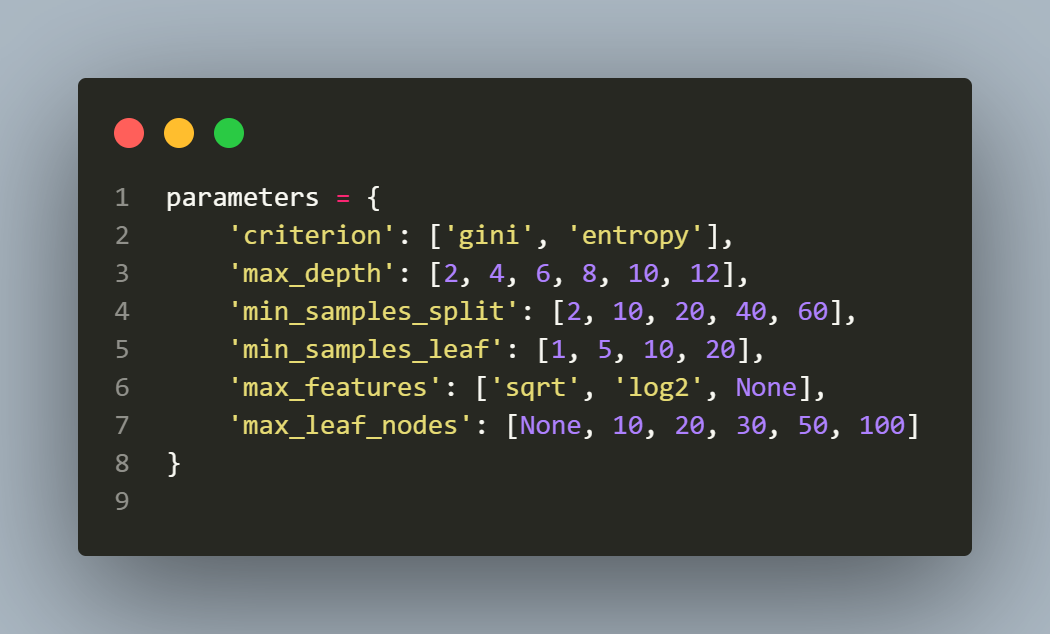
### 2-6-1-پیدا کردن هایپرپارامترهای بهینه با استفاده از GridSearch

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.



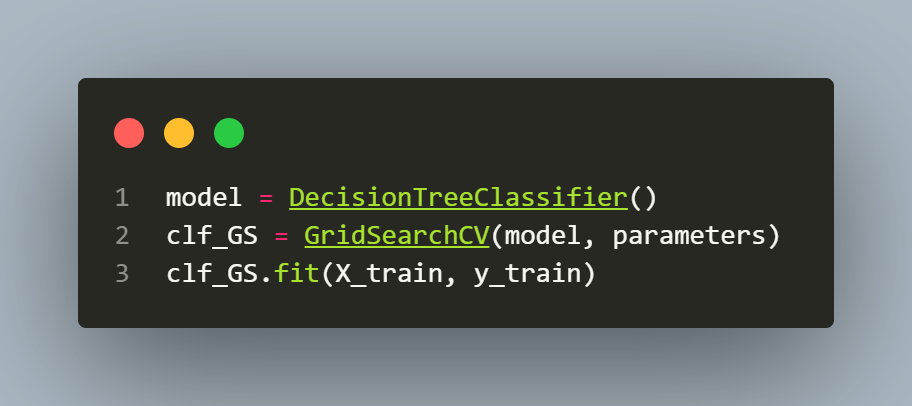
شکل 35: استانداردسازی داده‌ها

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.



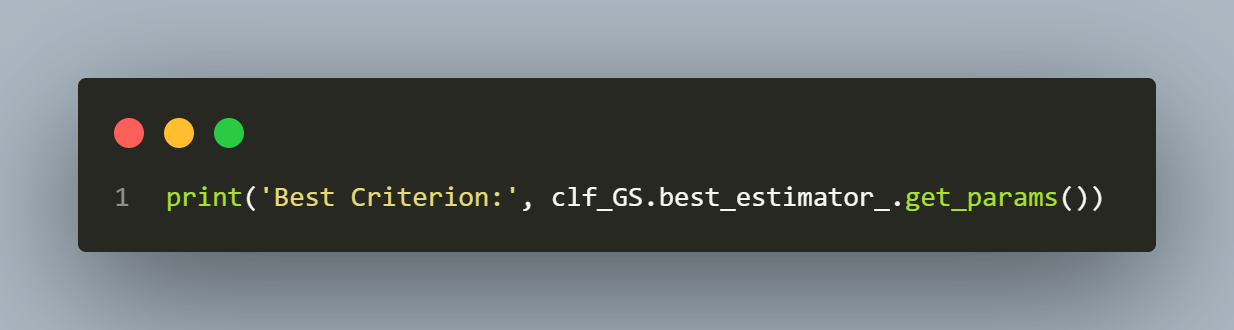
شکل 36: دیکشنری هایپرپارامترها

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.

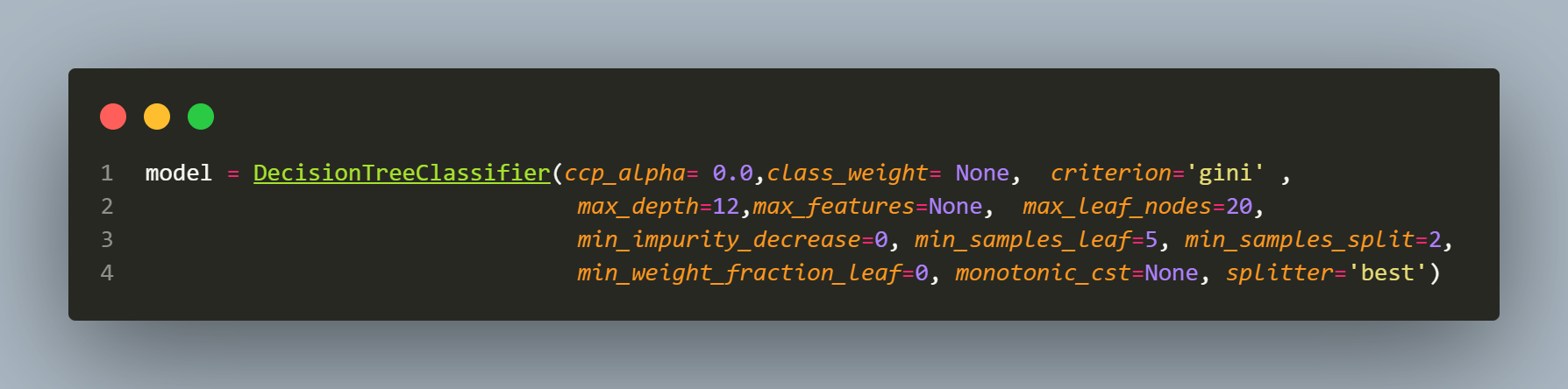


شکل 37: پیداکردن ترکیب بهینه هایپرپارامترها با گرید سرچ

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

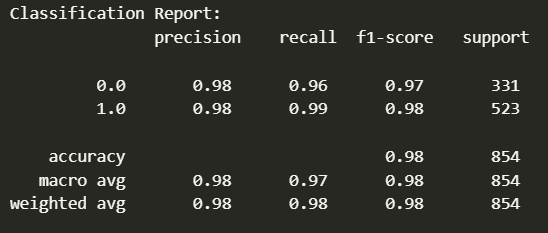


شکل 38: نمایش پارامترهای بهینه

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

شکل 39: درخت تصمیم با هایپرپارامترهای بهینه

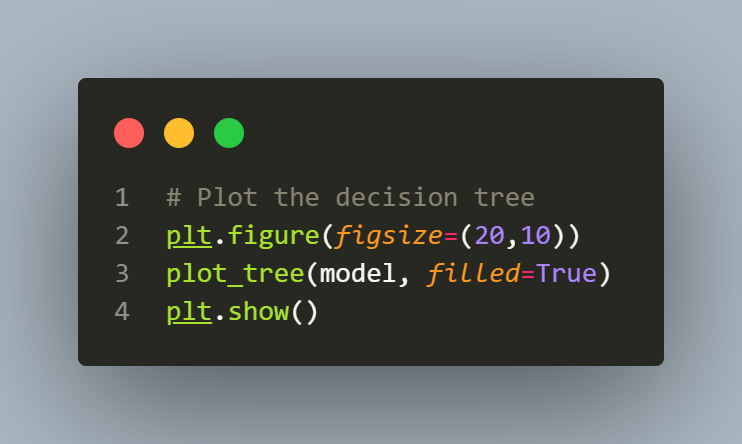
حال به طور مشابه با قبل مدل را آموزش و تست می‌کنیم. شکل (40) گزارش به دست آمده را نشان می‌دهد.



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.



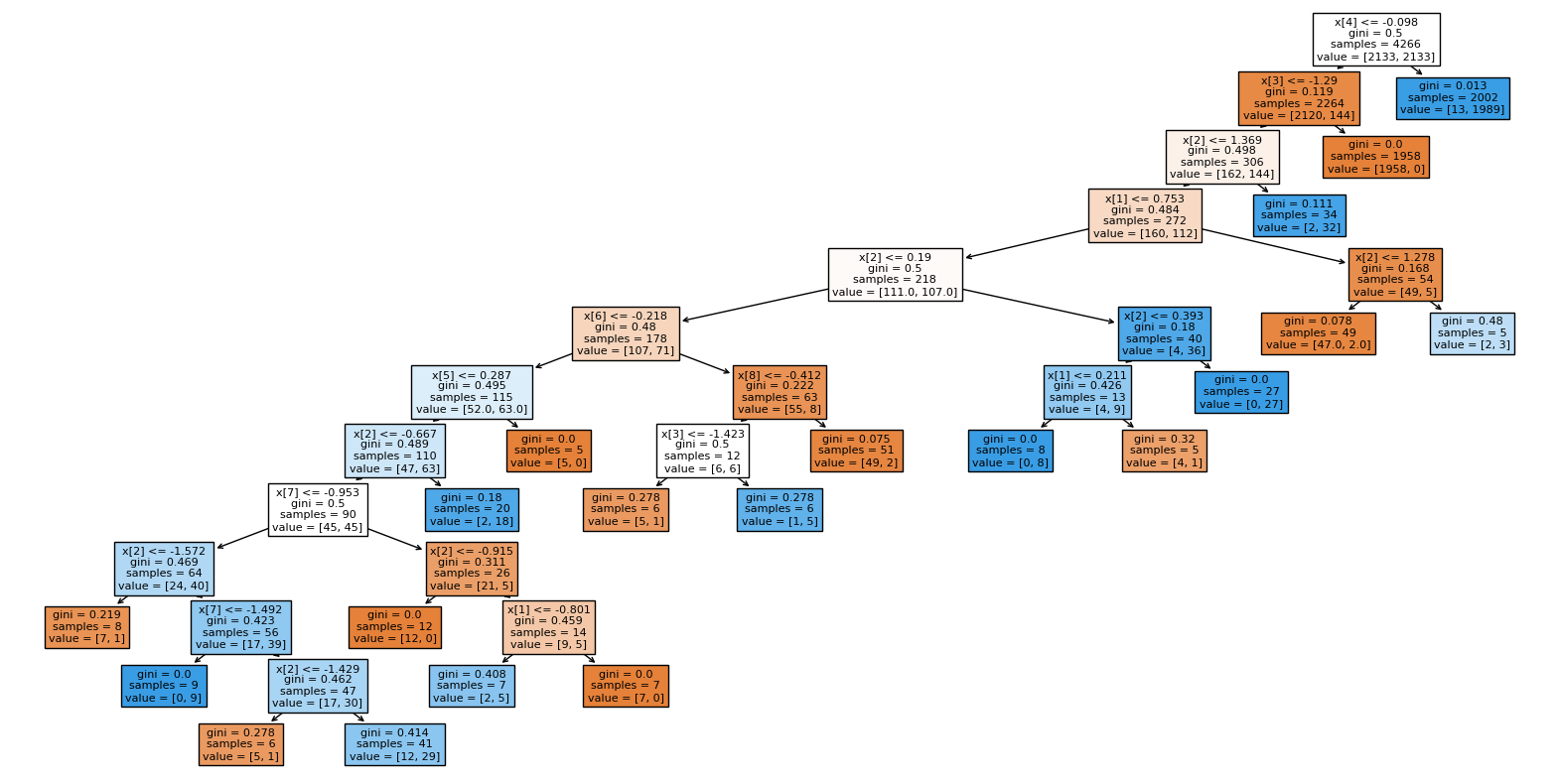
شکل 41: کد مربوط به نمایش شکل درخت

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

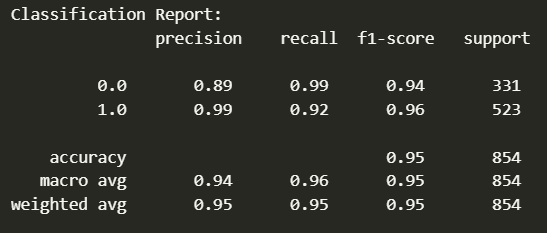
پارامتر ccp\_alpha پارامتر کلیدی برای هرس کردن درخت است. اگر مقدار آن برابر صفر باشد، هیچگونه هرس کردنی صورت نمی‌گیرد. همچنین با پارامترهای max\_depth و max\_leaf\_nodes و min\_samples\_leaf و min\_samples\_split نیز می‌توان درخت را هرس کرد. این پارامترها به ترتیب، حداکثر عمق درخت، حداکثر شاخه‌ها، مینیم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

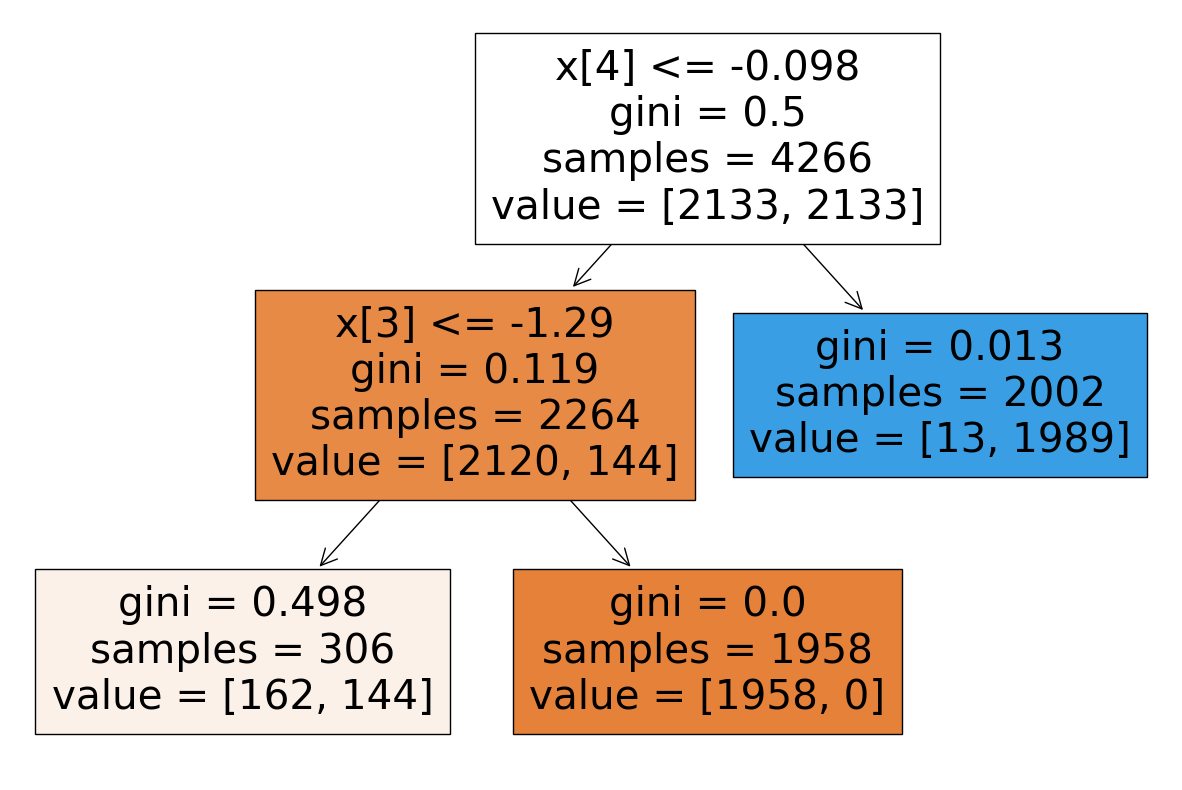
مشاهده می‌شود که درخت هرس شده بسیار کوچکتر از درخت اصلی است ولی گزارش عملکرد آن تنها چند درصد ضعیف‌تر از درخت اصلی است. مزایای این درخت، توضیح پذیری ساده آن و مقاومت آن در برابر بیش برازش است. در زمینه‌هایی که این دو مورد اهمیت دارند، استفاده از درخت هرس شده توصیه می‌شود.



شکل 42: درخت به دست آمده از هایپرپارامترهای بهینه‌شده



شکل 43: گزارش عملکرد درخت هرس شده

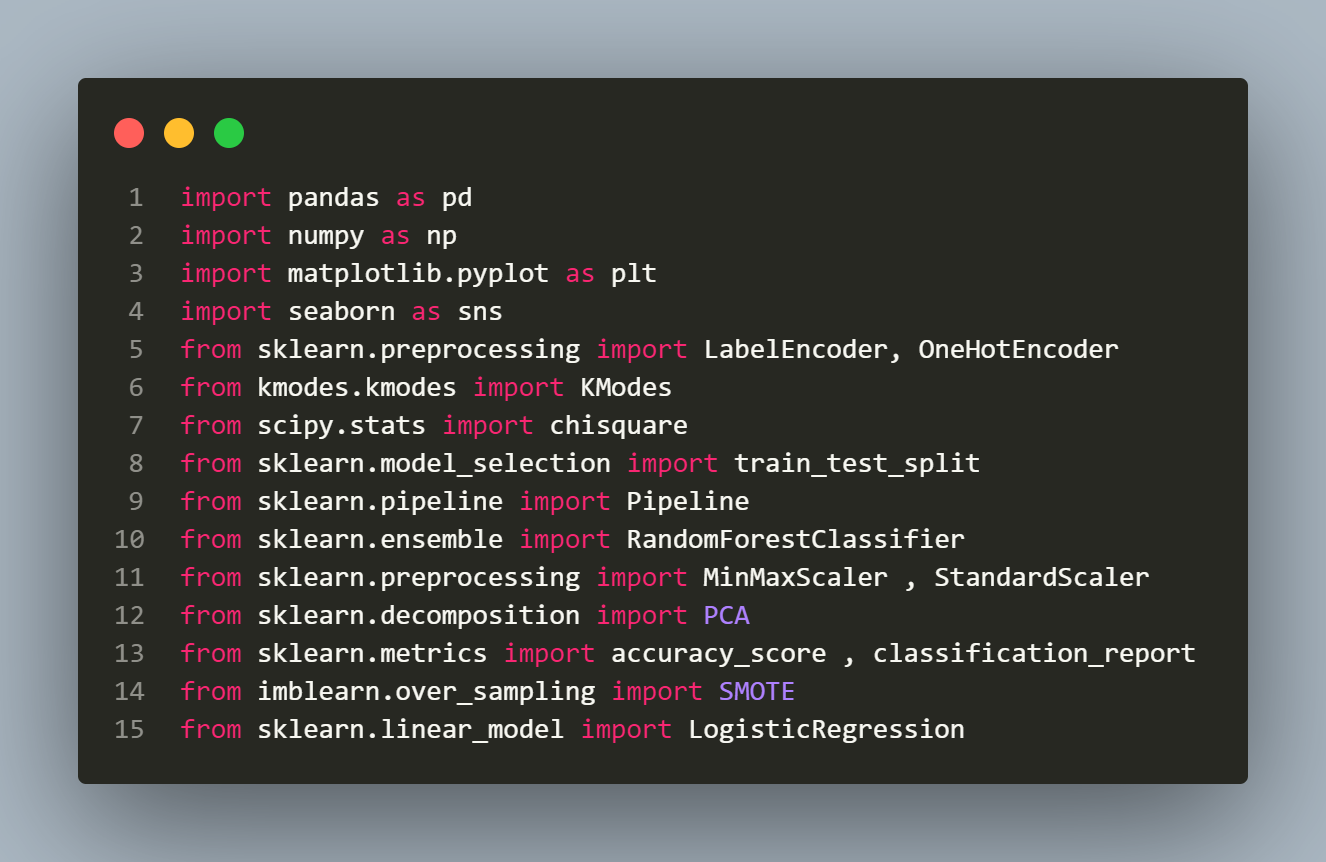


شکل 44: درخت هرس شده

# 2-پاسخ سوال دوم

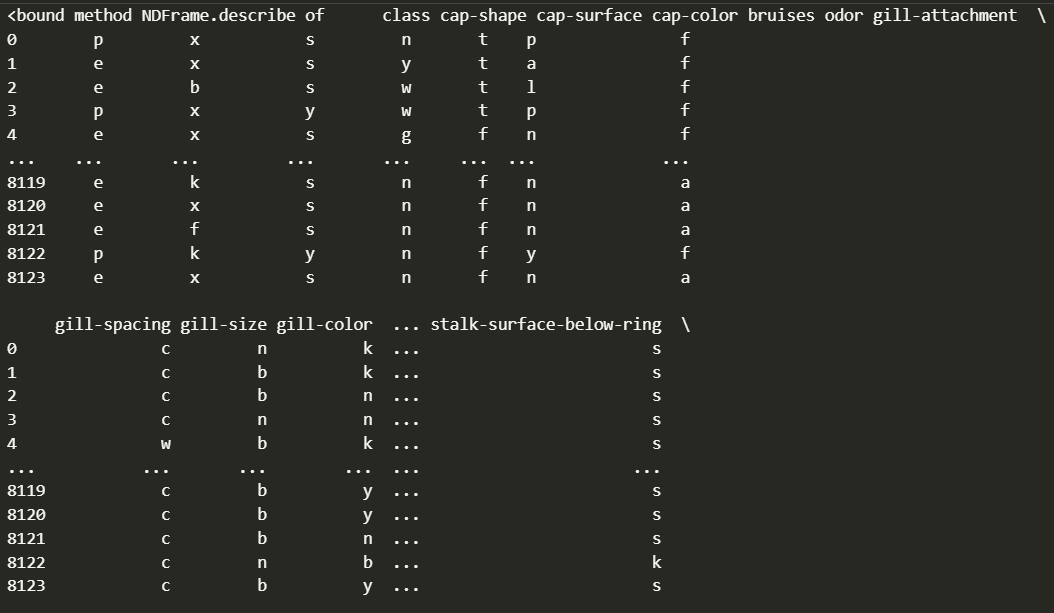
## 1-2-آماده سازی داده‌ها

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



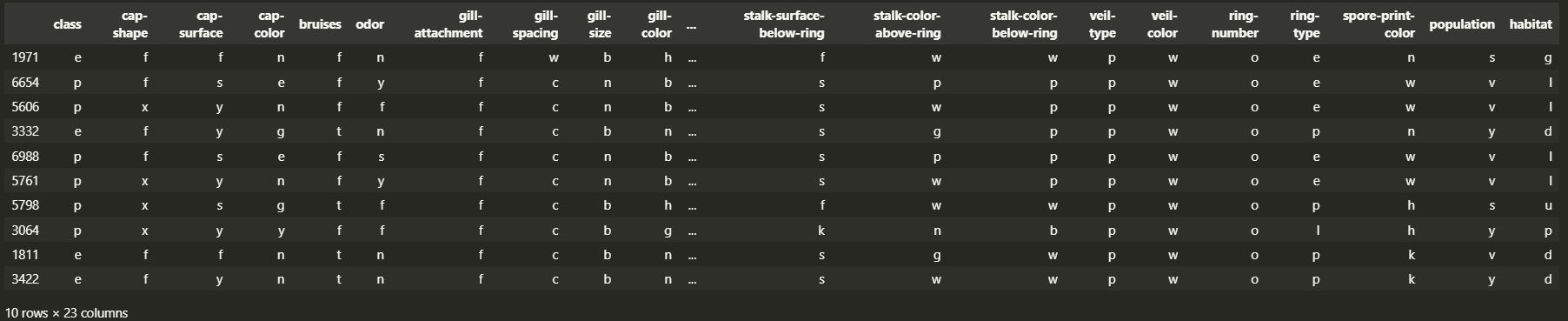
شکل 45: وارد کردن کتابخانه‌های مورد نیاز

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



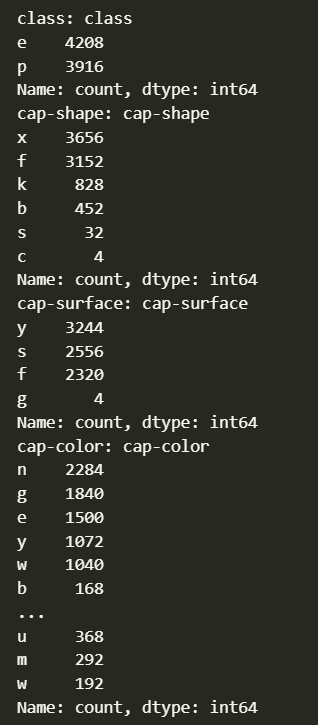
شکل 46: خروجی متد .describe()

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.

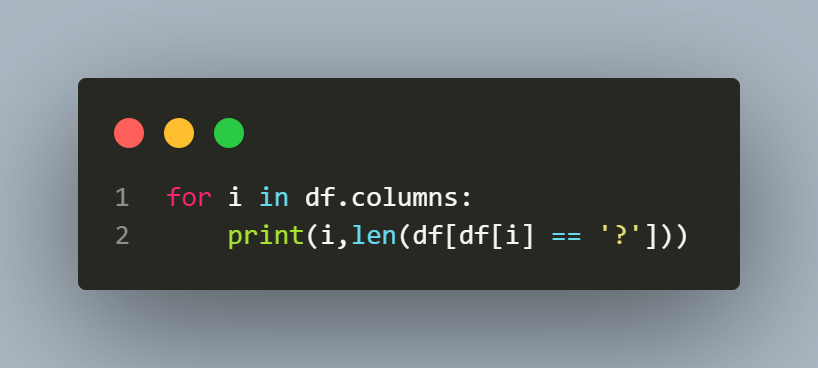


شکل 48: خروجی تعداد هر دسته در هر ستون

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.

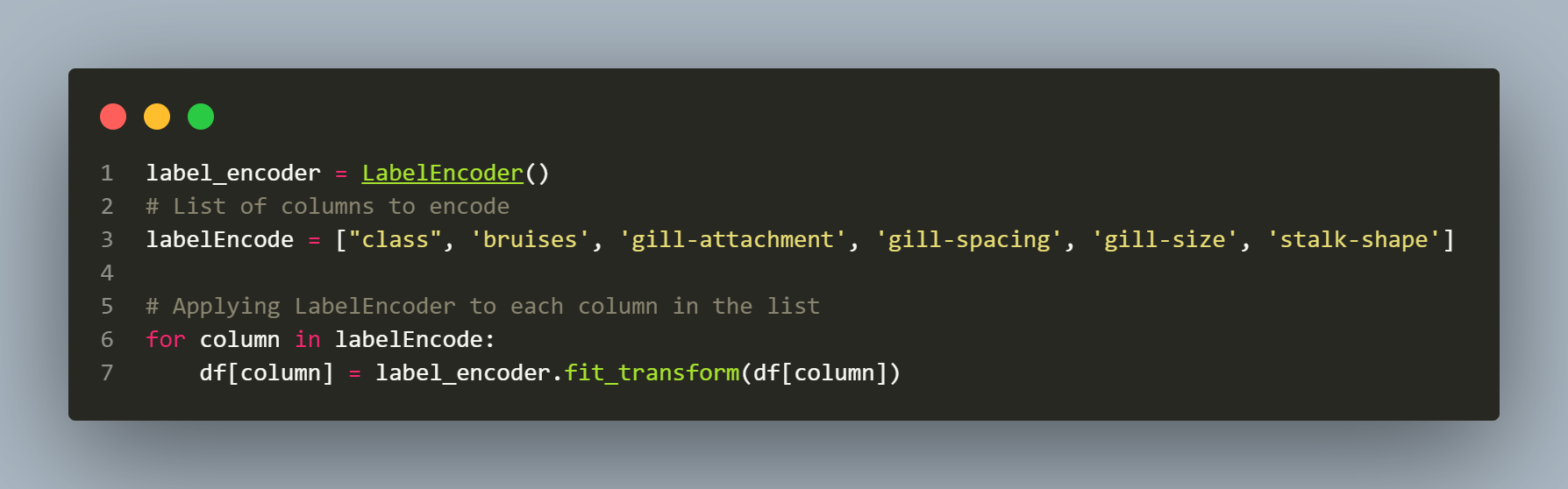
حال بررسی می‌کنیم که آیا در ستون‌ها، مقدار ؟ داریم یا خیر. برای این منظور از کد شکل (49) استفاده می‌کنیم.



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.



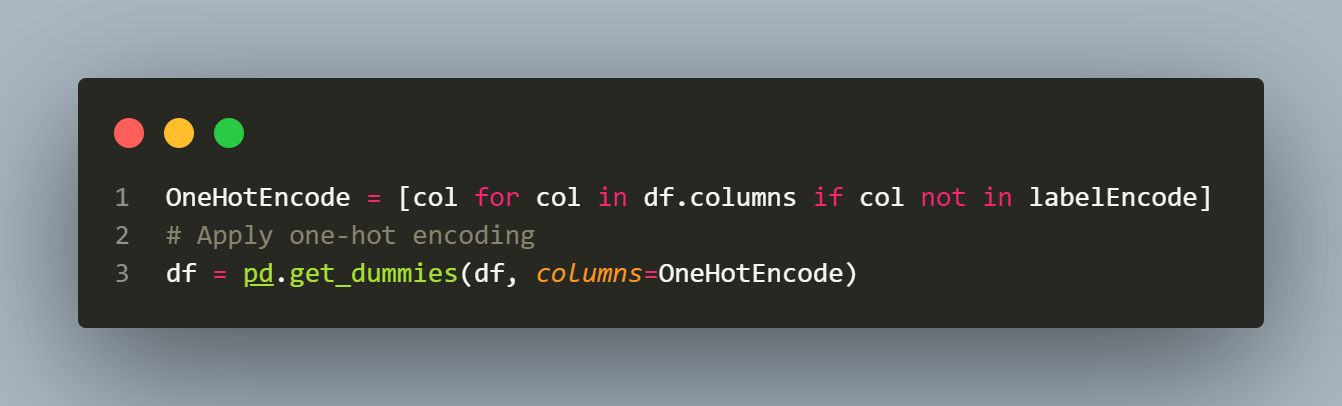
شکل 50: انکود کردن ستون‌های دودویی

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.

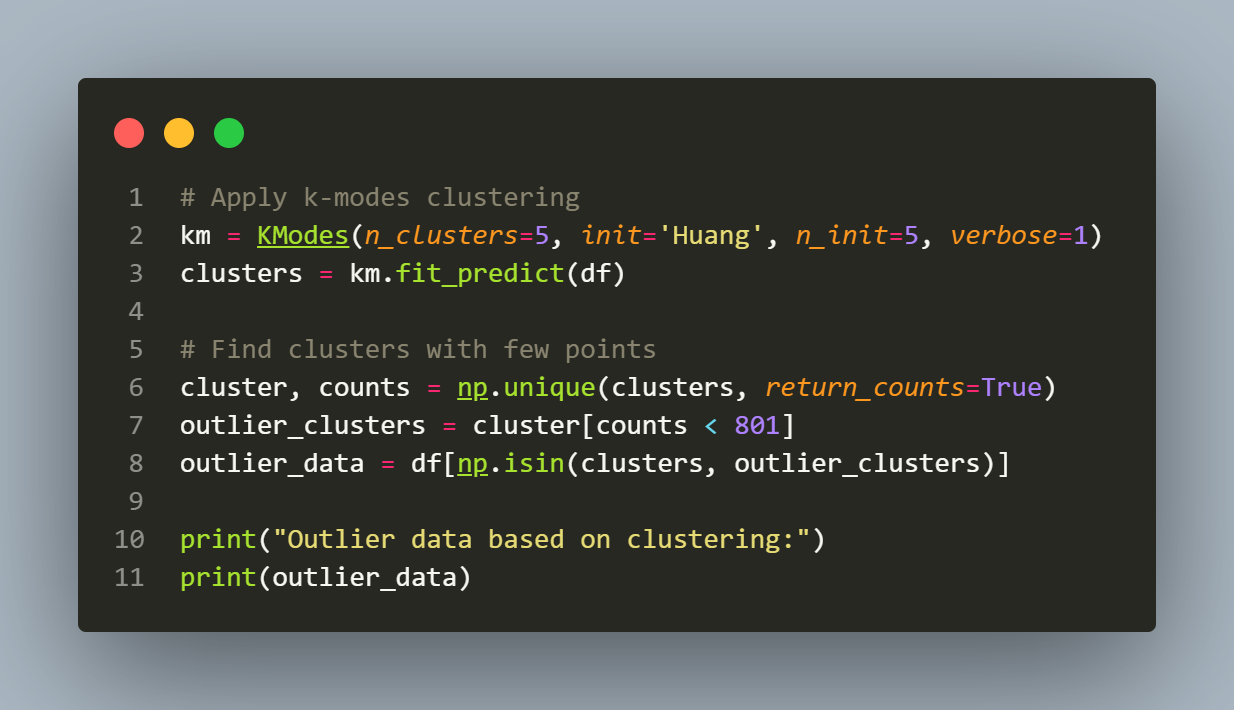
## 2-2-حذف داده‌های پرت

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

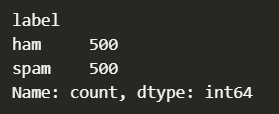
برای این منظور از کلاس KModes یک شی می‌سازیم. تعداد خوشه‌ها را روی 5 تنظیم می‌کنیم. با استفاده از دستور fit\_predict() و پاس دادن دیتافریم، خوشه‌بندی را انجام می‌دهیم. سپس خوشه‌هایی را که کمتر از یک مقدار به خصوص دارند را در لیست outlier\_cluster ذخیره می‌کنیم و با استفاده از تابع isin() آنها را حذف 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: پیداکردن داده‌های پرت با استفاده از خوشه‌بندی

# 3-پاسخ سوال سوم

پس از خواندن دیتاست، ابتدا تعداد رکورد موجود از هر لیبل را نمایش می‌دهیم. شکل (63) خروجی این کار را نمایش می‌دهد.



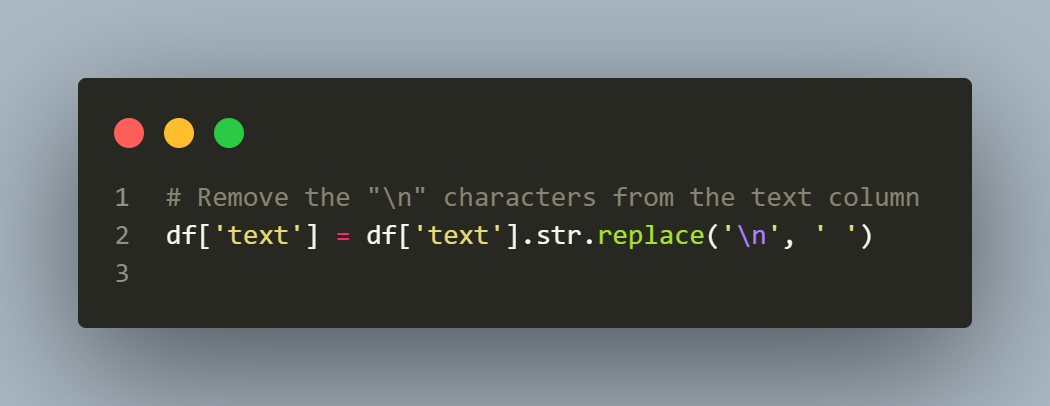
شکل 53: تعداد ایمیل‌های با لیبل سالم و اسپم

مشاهده می‌شود که دیتاست بالانس است. در ادامه لیبل‌های ham را به صفر و spam را به یک تبدیل می‌کنیم.

## 1-3-پیش پردازش داده‌ها

### 1-1-3-حذف کاراکترهای \n

اولین قدم، حذف کاراکتر \n از رشته‌ی هر ایمیل است. برای این منظور از متد .replace() استفاده می‌کنیم. کد شکل (64) نحوه انجام این کار را نمایش می‌دهد.



شکل 54: حذف کاراکتر \n

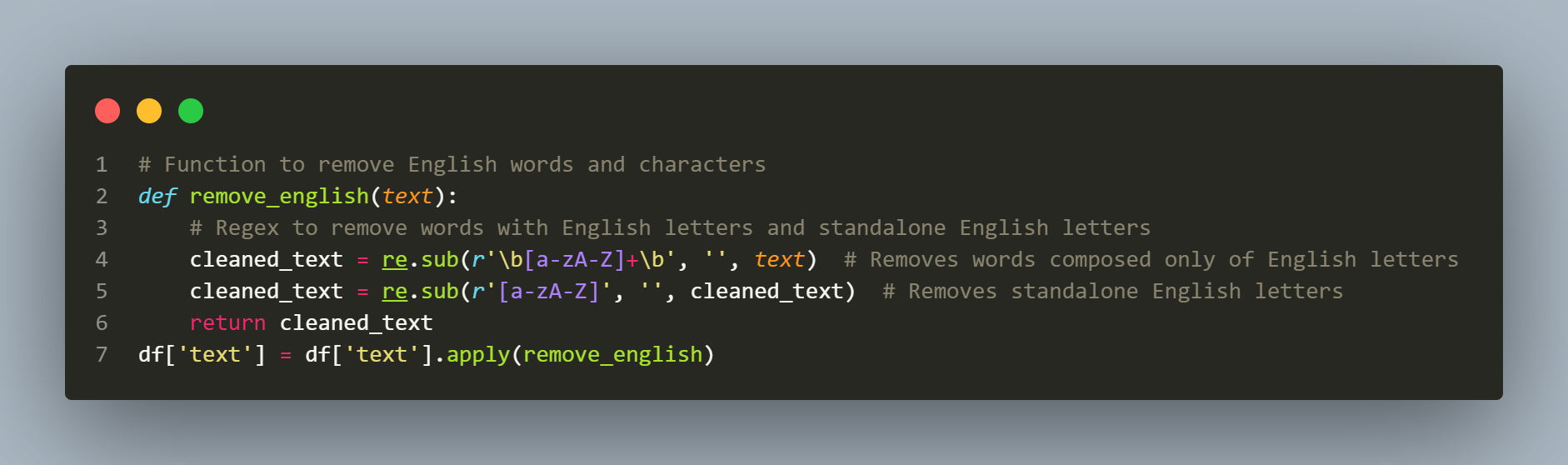
### 2-1-3-حذف کاراکترهای انگلیسی

قدم بعدی حذف کلمات و کاراکترهای انگلیسی است. دلیل انجام این کار این است که زبان ایمیل‌ها فارسی است و وجود کاراکترهای انگلیسی باعث می‌شود که مدل روی کلماتی آموزش ببیند که لزوماً اطلاعاتی برای تصمیم‌گیری مدل ارائه نمی‌دهد. همچنین پس از حذف علائم نگارشی، از مواردی مانند ایمیل یا لینک‌ها، یک رشته‌ی بلند و نه چندان معنادار از کاراکترهای فارسی می‌ماند که مدل را گمراه می‌کند.

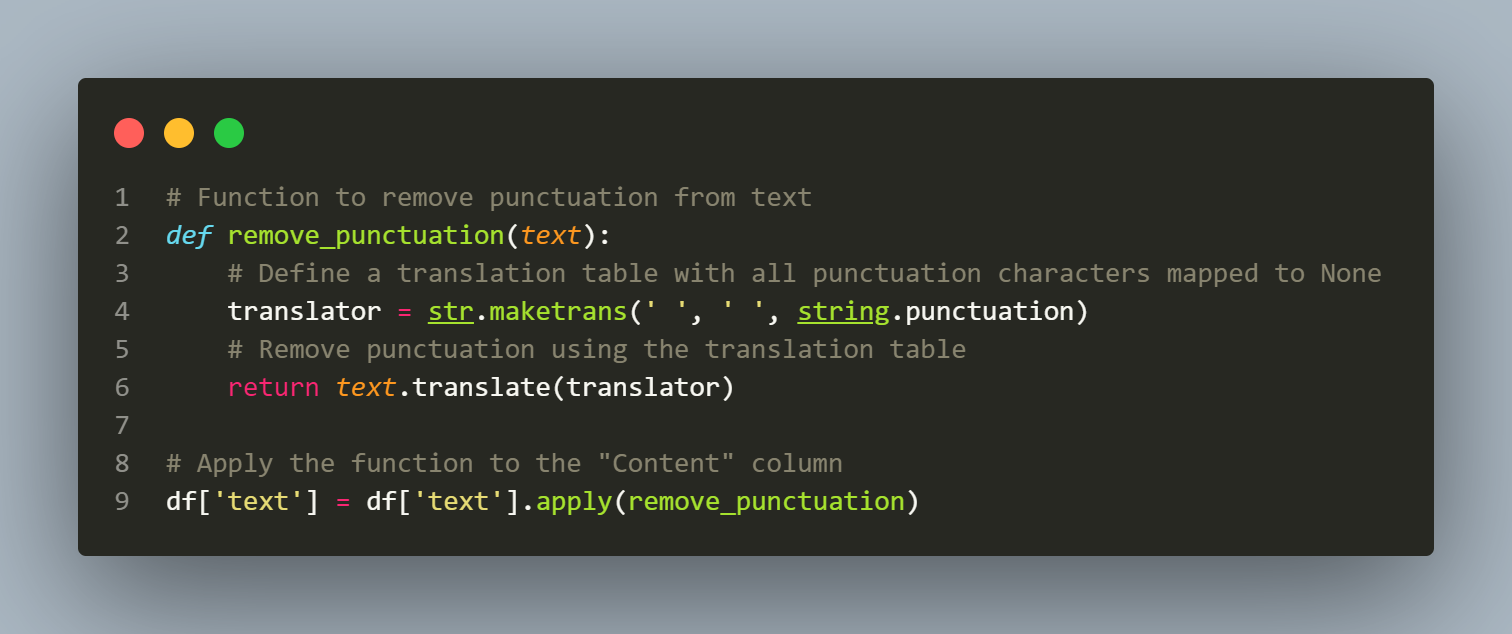
برای این منظور از کتابخانه رجکس استفاده می‌کنیم. ابتدا یک تابع را تعریف می‌کنیم که ردیف به ردیف سطرها را دریافت می‌کند و سپس با استفاده از متد sub() کاراکترهای انگلیسی را حذف می‌کند و متن تمیز شده را برمی‌گرداند. شکل (65) نحوه انجام این کار را نمایش می‌دهد.

### 3-1-3-حذف علائم نگارشی

در مرحله بعدی، علائم نگارشی را حذف می‌کنیم. دلیل انجام این کار این است که تکرار یکیسری از علائم مانند نقطه یا ات‌ساین باعث گمراه شدن مدل می‌شود در حالی که حضور این موارد تعیین کننده اسپم بودن ایمیل‌ها نیست. شکل (66) نحوه انجام این کار با متد maketrans()



شکل 55: حذف کاراکترهای انگلیسی

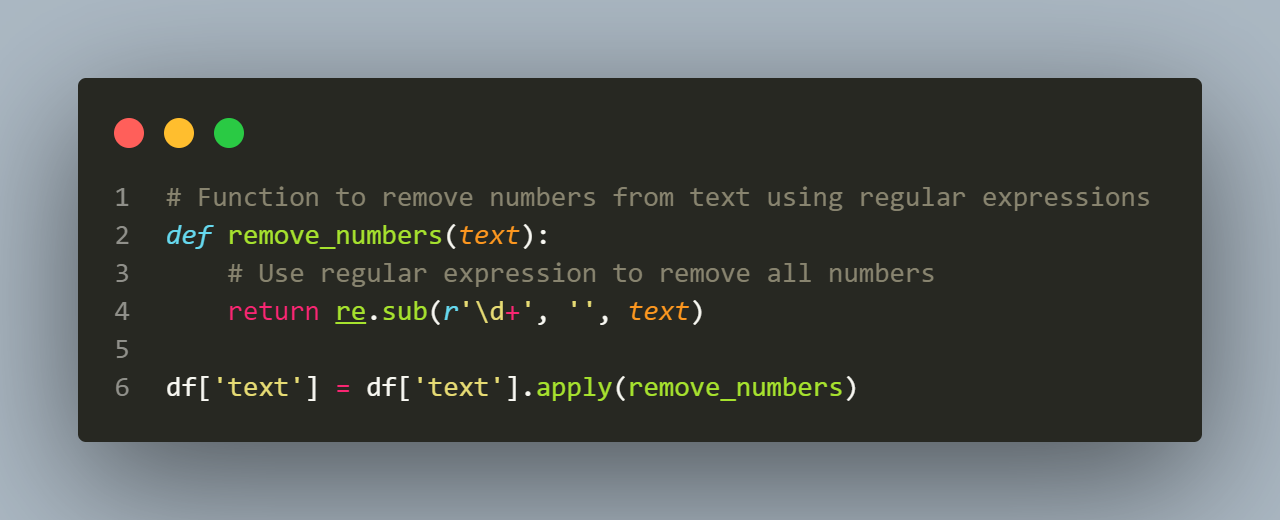


شکل 56: حذف علائم نگارشی

را نمایش می‌دهد. همچنین علائم نگارشی از اتریبیوت string.punctuation فراخوانی خواهند شد.

4-1-3-حذف اعداد

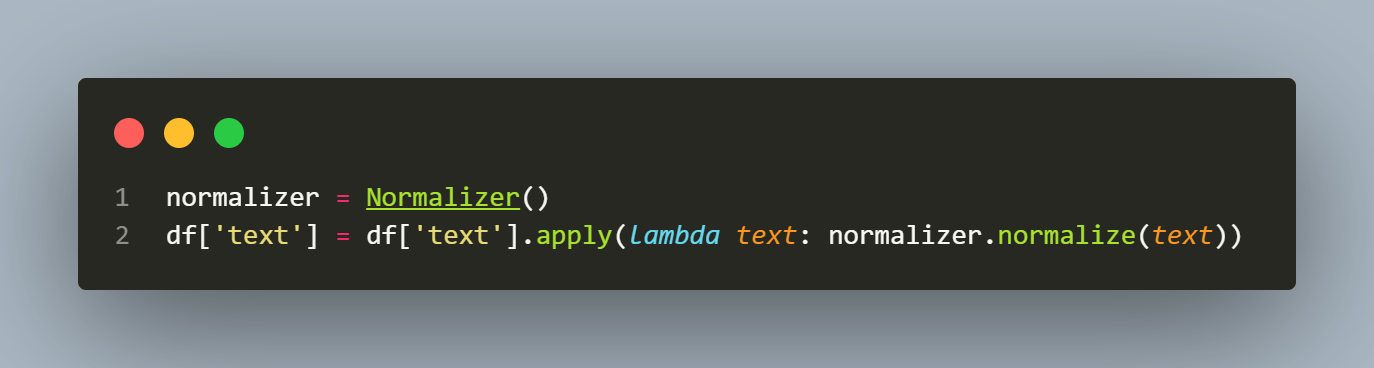
پس از انجام این مراحل، اعداد را از ایمیل‌ها حذف می‌کنیم. وجود اعداد باعث ایجاد نویز در داده‌ها می‌شود. برای حذف اعداد مجدداً همانند قبل از رجکس و تابعی استفاده می‌کنیم که با متد apply() سطر به سطر ایمیل‌ها را دریافت می‌کند و اعداد را از آنها حذف می‌کند.



شکل 57: حذف اعداد با استفاده از رجکس

### 5-1-3- نرمال کردن متن‌ها

نرمال کردن متن ریشه‌بندی، لغت‌سازی و حذف کلید واژه، ابعاد داده‌های متن را با جمع کردن کلمات مشابه در یک نمایش مشترک کاهش می‌دهد. این کار فضای ویژگی ها را ساده می‌کند و کارایی الگوریتم‌ها را بهبود می‌بخشد. برای نرمال کردن متن از کتابخانه Hazm یک شی از کلاس Normalizer() می‌سازیم. سپس روی ستون text متد apply() را اجرا می‌کنیم و یک تابع لامبدا به این متد پاس می‌دهیم که آرگومان آن متن است. سپس متد normalize() را روی متن پاس داده شده اجرا می‌کنیم و نتیجه را بر می‌گردانیم. شکل (68) نحوه انجام نرمال‌سازی را نمایش می‌دهد.

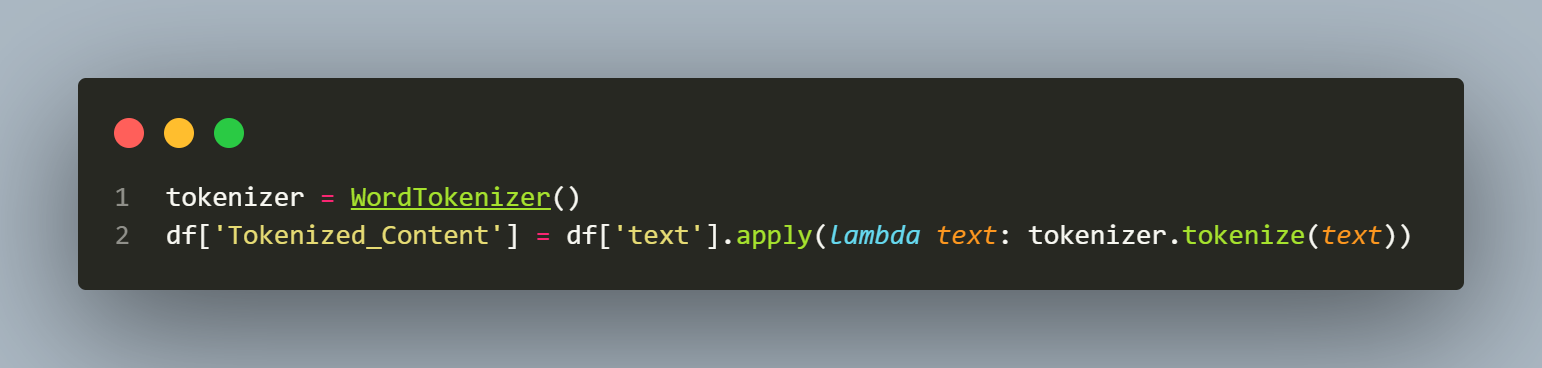


شکل 58: نحوه انجام نرمال سازی متن

### 6-1-3-توکن کردن متن

توکن‌سازی متن را به واحدهای کوچک‌تری که می توانند کلمات، عبارات یا کاراکترها باشند، تجزیه می‌کند. این جزئیات تجزیه و تحلیل دقیق‌تری از متن را امکان پذیر می کند و به الگوریتم‌ها اجازه می دهد تا بر روی واحدهای جداگانه کار کنند تا متن. توکن‌ها ساده‌ترین جز سازنده برای ایجاد ویژگی‌ها از متن هستند.

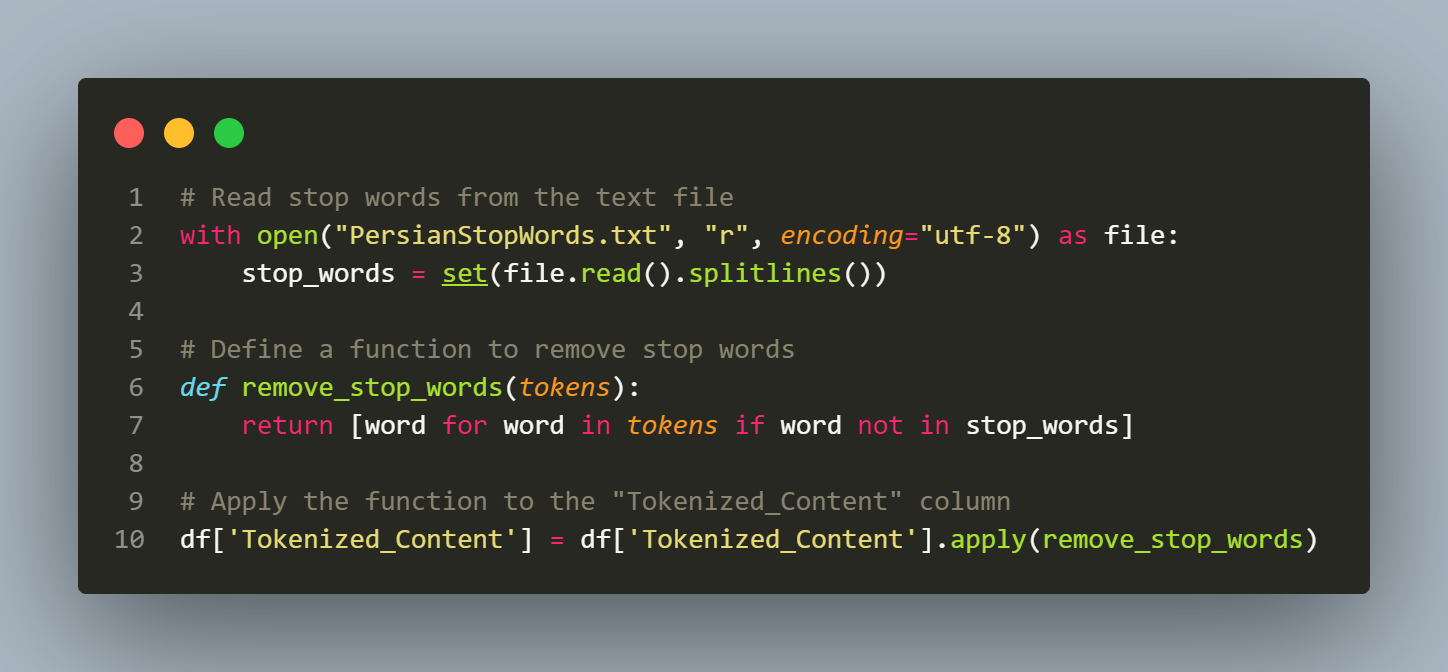
برای توکن کردن متن از کلاس WordTokenizer کتابخانه Hazm استفاده می‌کنیم. دقیقاً مانند قسمت قبلی در متد apply() یک تابع لامبدا ایجام می‌کنیم و با متد tokenize سطر به سطر متن‌ها را توکن می‌کنیم. لیست توکن‌های هر سطر را در یک ستون جداگانه به نام Tokenized\_Content ذخیره می‌کنیم. شکل (69) نحوه انجام این کار را نمایش می‌دهد.



شکل 59: توکن سازی متن

### 7-1-3-حذف ایست‌واژه‌ها (کلمات پالایشی یا stop words)

برای حذف ایست‌واژه‌ها، ابتدا این واژه‌ها را از فایل PersianStopWords.txt می‌خوانیم و در متغیر stop\_words ذخیره می‌کنیم. سپس لیست‌های موجود در ستون Tokenized\_Content را با متد apply یک به یک می‌خوانیم. سپس با استفاده از list comprehension توکن ها را یک به یک بررسی می‌کنیم که آیا در لیست ایست‌واژه‌ها موجود هستند یا نه. سپس توکن‌هایی که در ایست‌واژه‌ها موجود هستند را حذف می‌کنیم. شکل 70 نحوه انجام این کار را نمایش می‌دهد.



شکل 60: پیدا کردن و حذف ایست‌واژه‌ها

سپس توکن‌ها را در کنار هم قرار می‌دهیم و متن‌های بلند را مجددا تشکیل می‌دهیم. (به اصطلاح flatten کردن) شکل (71) نحوه انجام این کار را نمایش می‌دهد.



شکل 61: بازتولید متن‌ها از کلمه‌ها

8-1-3- 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 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 ماتریس به دست آمده را به دیتافریم تبدیل می‌کنیم. شکل (72) نحوه انجام این کار را نمایش می‌دهد.

حال دیتاست برای آموزش KNN آماده است. این دیتاست هزار سطر و 18886 ستون دارد.

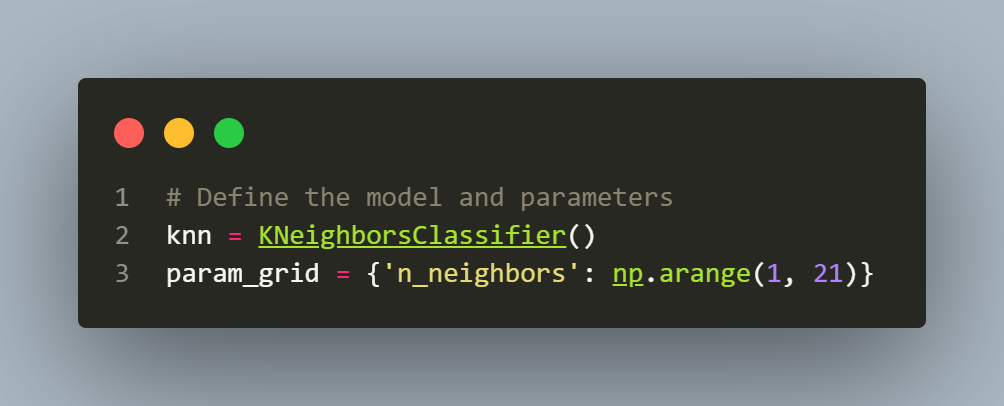
## 2-3-آموزش KNN و پیدا کردن بهترین K

ابتدا دیتاست را به دو دسته آموزشی و آزمایشی تقسیم می‌کنیم. ابتدا داده‌ها را استاندارد سازی می‌کنیم. در مرحله بعدی با استفاده از Oversampling تعداد سطرهای با لیبل‌ صفر و تعداد لیبل‌های یک را با هم برابر می‌کنیم. برای این منظور از روش SMOTE استفاده می‌کنیم. برای این منظور از کلاس SMOTE یک شی می‌سازیم و با پاس دادن داده‌های آموزشی به متد fit\_resample() کار oversampling را انجام می‌دهیم.



شکل 62: استفاده از روش TF-IDF

سپس از کلاس KNeighborsClassifier() یک شی می‌سازیم. همچنین یک دیکشنری ایجاد می‌کنیم و پارامترهایی که قرار است روی آنها Grid search انجام بدهیم را به عنوان کلید و مقادیر پارامترها را به عنوان value در دیکشنری قرار می‌دهیم. شکل (73) تعریف شی و این دیکشنری را نمایش می‌دهد.



شکل 63: تعریف مدل و دیکشنری پارامترها

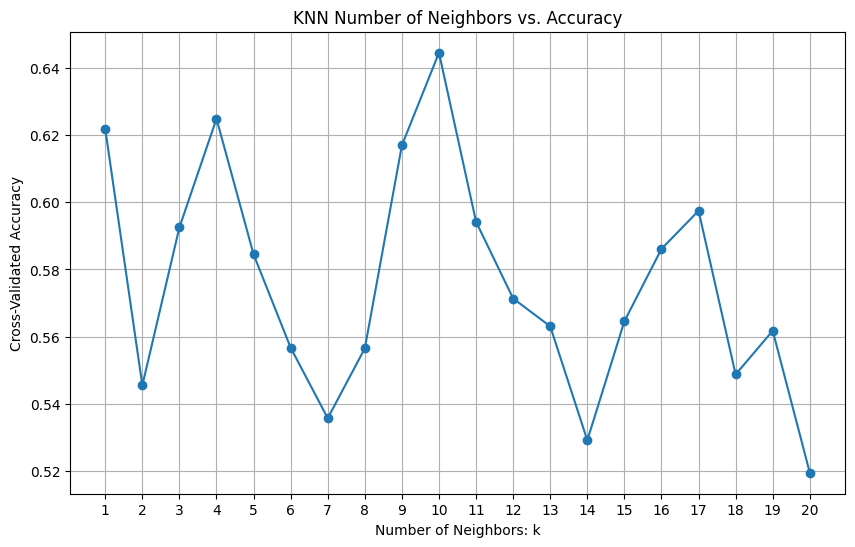
سپس از کلاس GridSearchCV() یک شی می‌سازیم. آرگومان‌های پاس داده شده به این کلاس، به ترتیب مدل تعریف شده، دیکشنری پارامترها، تعداد بخش‌ها برای CrossValidation و معیار انتخاب شده برای ارزیابی است. تعداد بخش‌ها را پنج قرار می‌دهیم و معیار ارزیابی را دقت قرار می‌دهیم. سپس با متد .fit() و پاس دادن داده‌های آموزشی، Grid search را آغاز می‌کنیم.

شی ساخته شده از GridSearchCV() هر بار پارامتر n\_neighbors را تنظیم می‌کند، CV را انجام می‌دهد و در نهایت یک مقدار برای دقت مدل باز می‌گرداند. حال می‌توان از این مقادیر، یک نمودار دقت بر اساس K رسم کرد.

شکل (74) کدی که این نمودار را رسم می‌کند نمایش می‌دهد. همچنین شکل (76) خروجی این نمودار را نمایش می‌دهد. با توجه مشاهده می‌شود که K بهینه برابر با ده است و در این حالت دقت برابر با 0.6444 خواهد شد.



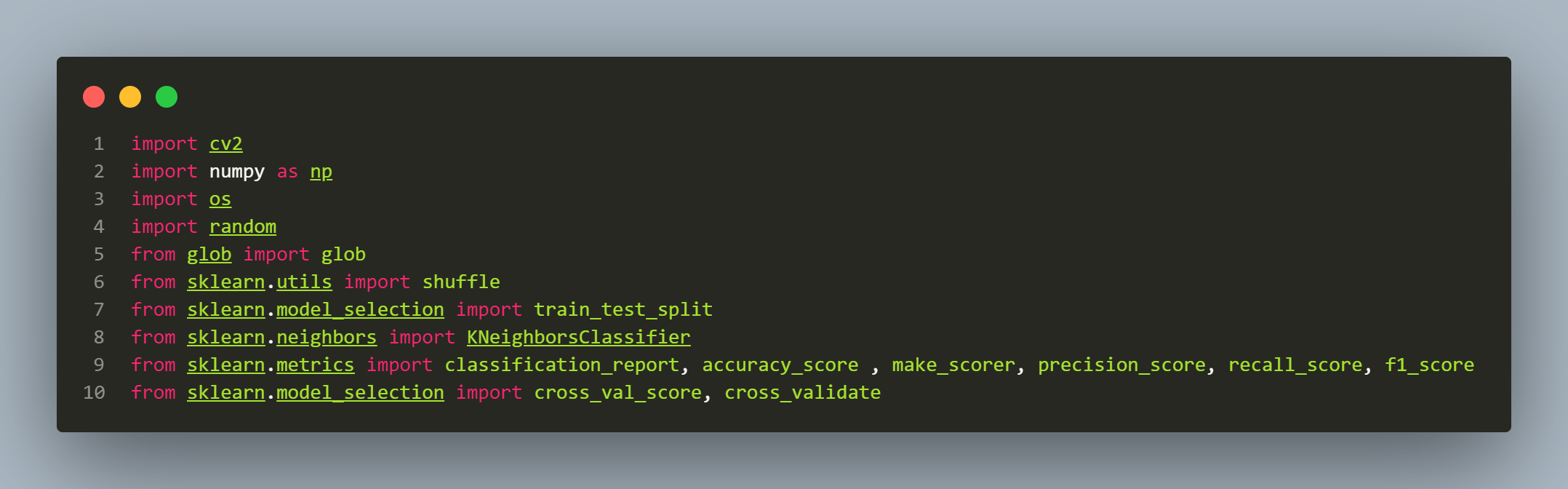
شکل 64: کد رسم نمودار دقت بر اساس K



شکل 65: نمودار دقت بر اساس K

# 4-پاسخ سوال چهارم

ابتدا کتابخانه‌هایی که در ادامه مورد نیاز هستند را ایمپورت می‌کنیم. شکل (76) این کتابخانه‌ها را نمایش می‌دهد.



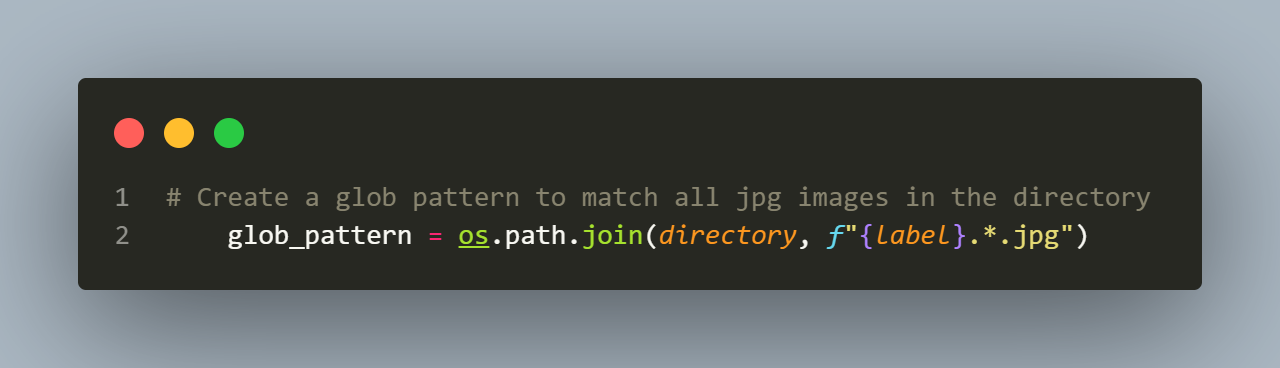
شکل 66: وارد کردن کتابخانه‌های مورد نیاز

## 1-4-وارد کردن و پیش‌پردازش داده‌ها

با استفاده از یک تابع، دو عملکرد وارد‌ کردن تصاویر و پیش‌پردازش توابع را انجام می‌دهیم. این تابع عکس‌ها را دریافت می‌کند و چند عملیات پیش‌پردازش که در ادامه به آنها پرداخته می‌شود را انجام می‌دهد و عکس آماده برای آموزش مدل را برمی‌گرداند. این رویکرد چندان همسو با کدنویسی تمیز و این قانون که هر تابع صرفاً یک عملیات را انجام دهد نیست ولی برای احرا در مقیاس کوچک موثر است.

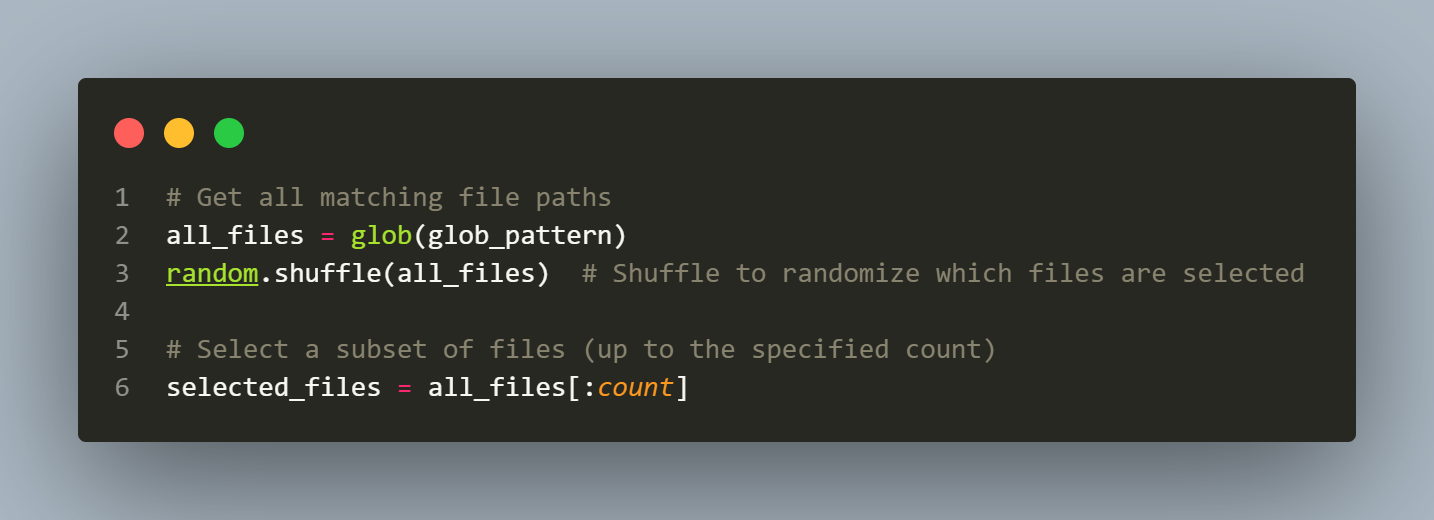
حال به بررسی بخش به بخش این تابع می‌پردازیم.

* تابع os.path.join(directory, f"{label}.\*.jpg") آرگومان directory را دریافت می‌کند و در مسیر تخصیص داده شده به این آرگومان، دنبال یک الگوی خاص می‌گردد. بخش اول الگو با استفاده از f sring از آرگومان label دریافت می‌شود. بخش دوم \* است که یک wild card است و هر کاراکتری را قبول می‌کند و بخش سوم .jpg است که فرمت دریافتی را مشخص می‌کند. آرگومان directory مسیر فایل را دریافت می‌کند و آرگومان lable به سطر هدف عکس اشاره دارد. این الگو در glob\_pattern ذخیره می‌شود. شکل (77) این بخش از تابع را نمایش می‌دهد.



شکل 67: تعریف الگوی دریافتی

* حال با استفاده از تابع glob() و پاس دادن الگوی ساخته شد، تمامی مسیر‌هایی که در پوشه مشخص شده هستند را دریافت می‌کنیم. سپس با استفاده از تابع suffle() این مسیر‌ها را بر می‌زنیم تا انتخاب فایل‌ها به صورت تصادفی باشد. در ادامه به تعداد مشخص شده در آرگومان count مسیر عکس‌ها را انتخاب می‌کنیم. شکل (78) نحوه انجام این کار را نمایش می‌دهد.
* سپس دو لیست خالی برای نگهداری جداگانه متغیرها و سطر هدف آنها ایجاد می‌کنیم.



شکل 68: خواندن و انتخاب تصادفی فایل‌ها

* در قدم بعدی، در یک حلقه، به ازای هر مسیر دریافت شده، عملیات‌های زیر را انجام می‌دهیم:

1-ابتدا با استفاده از تابع imread از این مسیر، عکس را به صورت رنگی می‌خوانیم و در متغیر img ذخیره می‌کنیم. آرگومان file\_path مسیر را مشخص می‌کند و آرگومان cv2.IMREAD\_COLOR نحوه خواندن عکس را. در صورتی که عکس به درستی خوانده نشود، پیام خطایی نمایش داده می‌شود

2-سپس عکس خوانده شده را با استفاده از تابع cvtColor و پاس دادن img به تابع و آرگومان cv2.COLOR\_BGR2GRAY به صورت gray scale تبدیل می‌کنیم و نتیجه را در متغیر img\_gray ذخیره می‌کنیم.

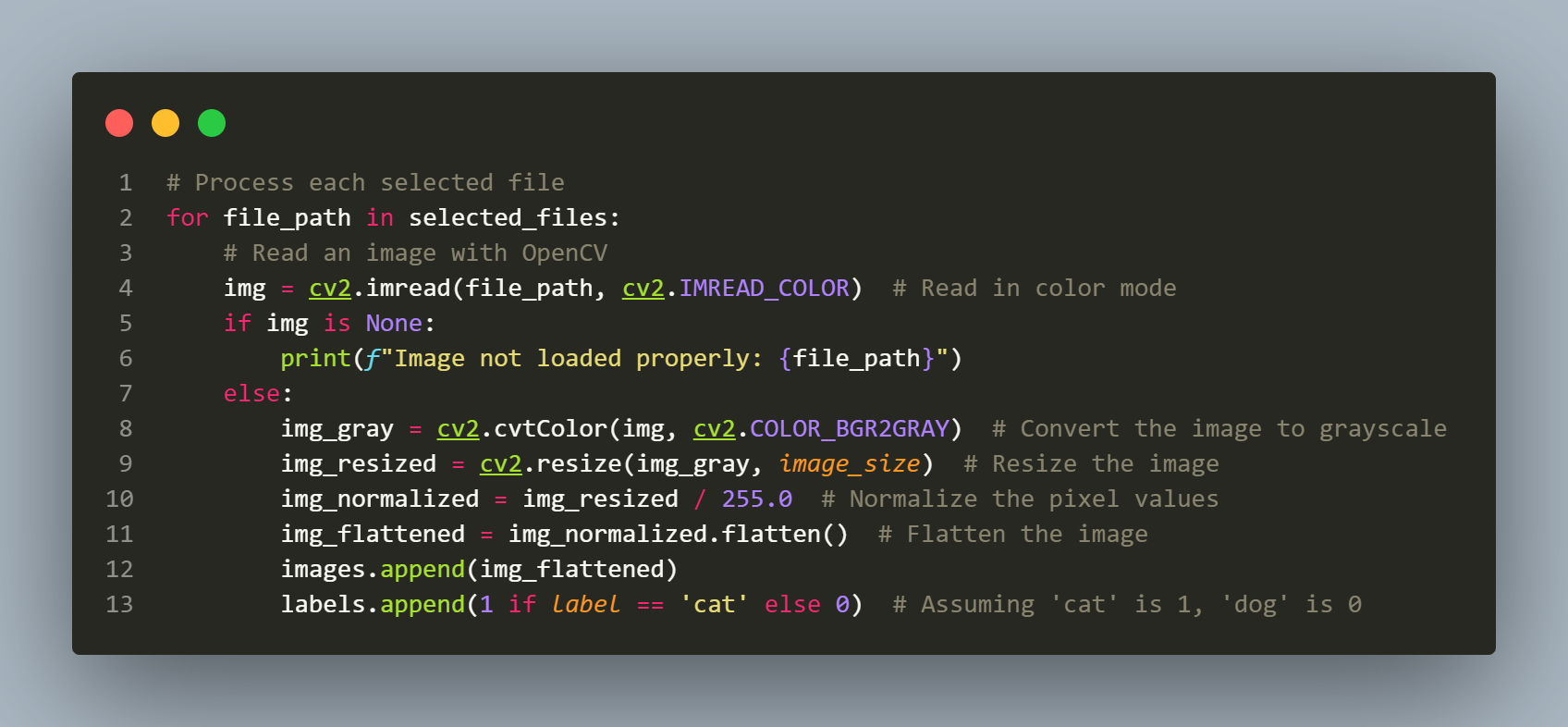
3-سپس با استفاده از تابع resize عکس را مجدداً سایز بندی می‌کنیم. متغیر image\_size که به این تابع پاس داده شده است، آرگومان دریافت شده توسط تابع اصلی است.

4-عکس با ابعاد جدید را با تقسیم بر 255 نرمال می‌کنیم.

5-سپس با استفاده از متد .flatten() ماتریس عکس را به یک بردار تک ستونی تبدیل می‌کنیم.

6-در نهایت این بردار را در لیست ایجاد شده برای متغیرها ذخیره می‌کنیم. برای ذخیره لیبل، اگر لیبل مربوط به گربه بود، آن را با یک و اگر مربوط به سگ بود آن را با صفر در لیست مربوط به لیبل ذخیره می‌کنیم

شکل (79) این حلقه را نمایش می‌دهد.



شکل 69: پیش‌پردازش یک به یک عکس‌ها

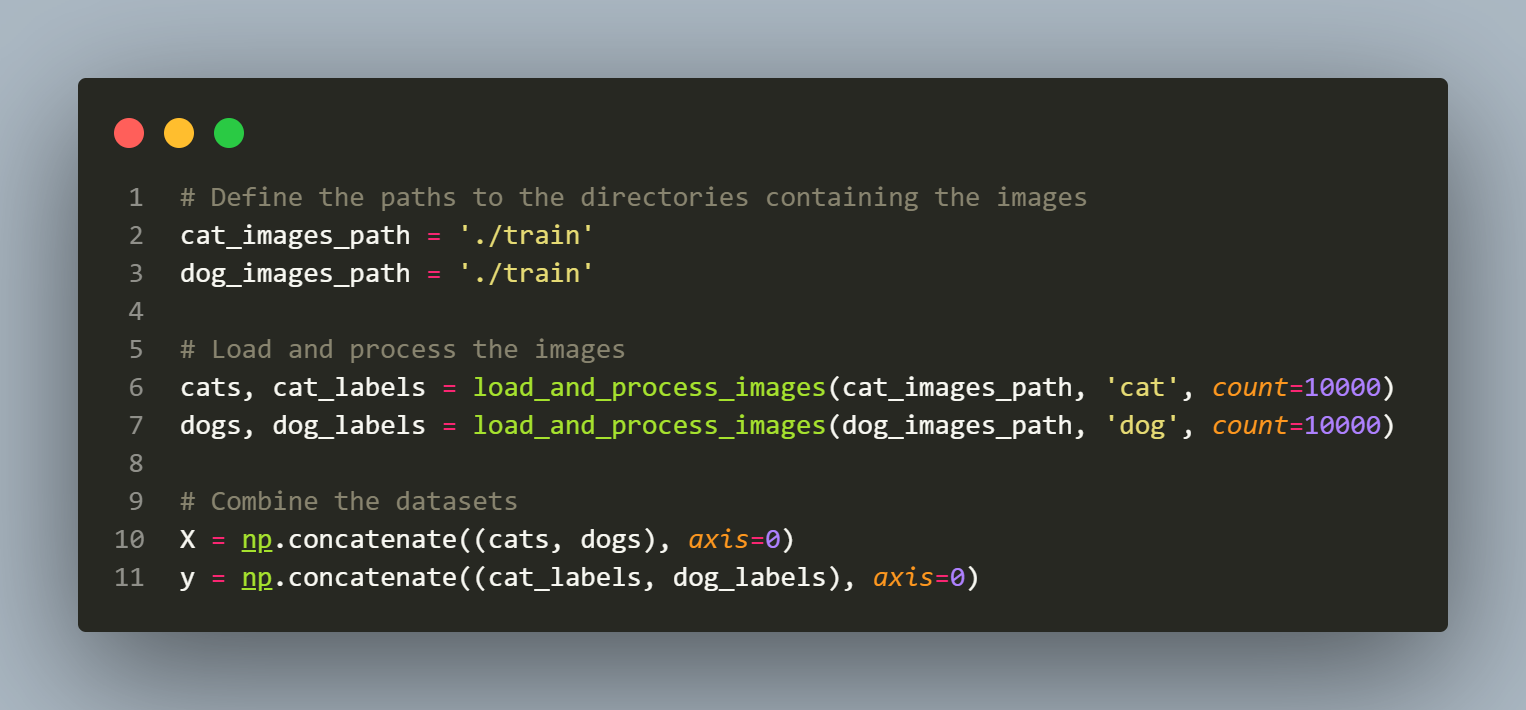
* در نهایت تابع لیست‌های ایجاد شده را به شکل آرایه‌های نامپای بر می‌گرداند

شکل (80) نمای کلی این تابع را نمایش می‌دهد.



شکل 70: تابع خواندن و پیش‌پردازش عکس‌ها

در قدم بعد، ابتدا آدرس هر یک از دیتاست‌های موجود را در یک متغیر ذخیره می‌کنیم. فایل نوت‌بوک به عنوان root در نظر گرفته می‌شود، در نتیجه اگر فولد دیتاست در کنار فایل نوت‌بوک باشد، آدرس دهی به شکل ‘./train’ انجام می‌شود. سپس با استفاده از تابع ایجاد شده، ده هزار عکس گربه، و ده هزار عکس سگ را می‌خوانیم و در لیست‌های متناظر با متغیرها و سطر هدف هر یک ذخیره می‌کنیم. سپس لیست‌ها را دوتا دوتا با هم ادغام می‌کنیم تا دیتاست نهایی ایجاد شود. شکل (81) نحوه انجام این کار را نمایش می‌دهد.



شکل 71: فراخوانی و پیش‌پردازش عکس‌ها و ایجاد دیتاست نهایی

حال آماده آموزش مدل بر روی عکس‌ها هستیم.

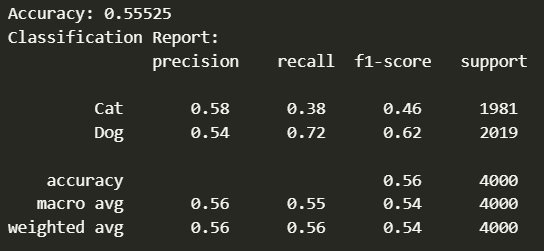
## 2-4-آموزش KNN

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 مدل را با پارامتر پنج همسایه آموزش می‌دهیم و روی داده‌های آزمایشی تست می‌کنیم. در ادامه با استفاده از تابع classification\_report() و پاس دادن پیش‌بینی‌ها و لیبل‌های سطر هدف و نام کلاس‌ها، یک گزارش از معیارهای عملکرد مدل درست می‌کنیم. شکل (82) کد مربوط به این بخش را نمایش می‌دهد.



شکل 72: کد نمایش گزارش عملکرد

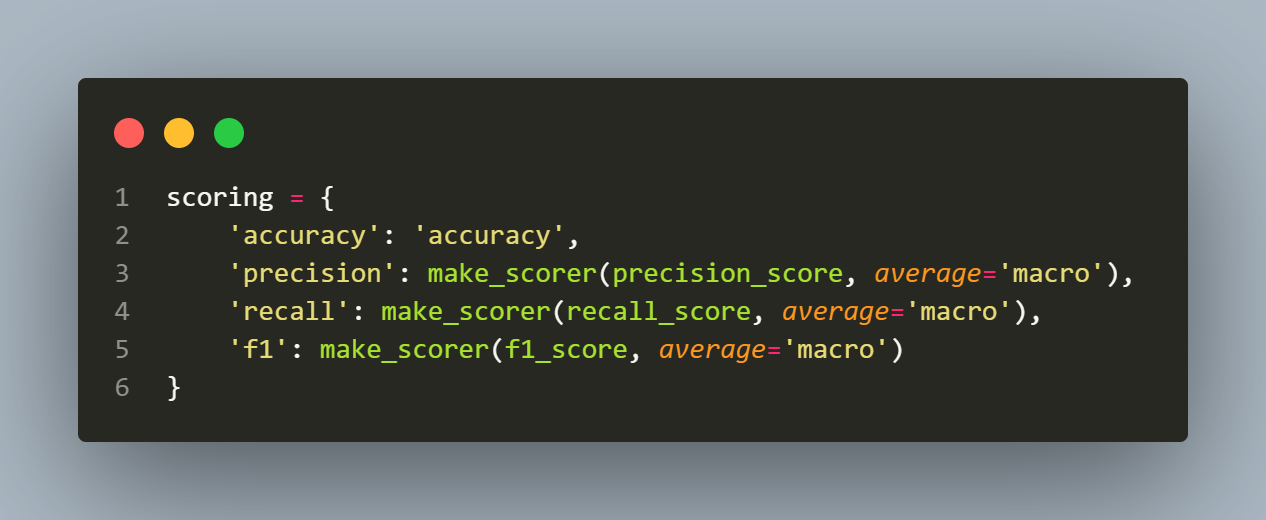
همچنین شکل (83) گزارش درست شده را نمایش می‌دهد.



شکل 73: گزارش عملکرد مدل KNN

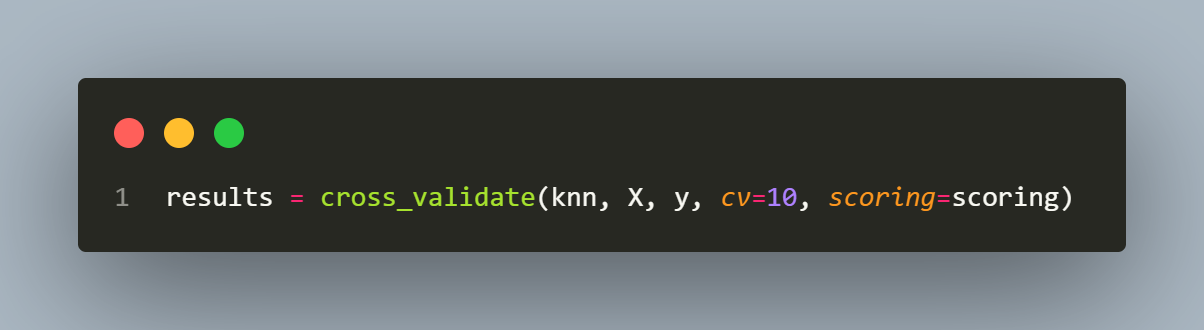
## 3-4-استفاده از Cross Validation

مجدداً مدل را با همان پارامتر قبلی ایجاد می‌کنیم. برای محاسبه معیار‌های گفته شده به همراه انجراف استانداردشان، ابتدا یک دیکشنری از معیارها ایجاد می‌کنیم که کلیدها نام هر معیار و valueها یک تابع make\_scorer() هستند که آرگومان هر کدام متناظر با معیار تعریف شده در کلید دیکشنری است. شکل (84) این دیکشنری را نمایش می‌دهد.



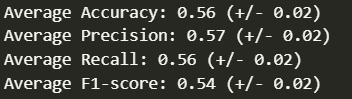
شکل 74: دیکشنری محاسبه معیارهای ارزیابی

سپس با استفاده از تابع cross\_validate و پاس دادن مدل، دیتاست، تعداد بخش‌ها که برابر ده است و دیکشنری معیارها، Cross Validation را انجام می‌دهیم.



شکل 75: انجام Cross Validation

سپس نتایج به دست آمده را نمایش می‌دهیم. شکل (86) این نتایج را نشان می‌دهد.



شکل 76: نتایج به دست آمده از Cross Validation

مشاهده می‌شود که نتایج به دست آمده بسیار ضعیف هستند. درواقع عملکرد KNN در دسته بندی عکس‌ها تنها اندکی بهتر از دسته بندی عکس‌ها با استفاده از پرتاب یک سکه است.