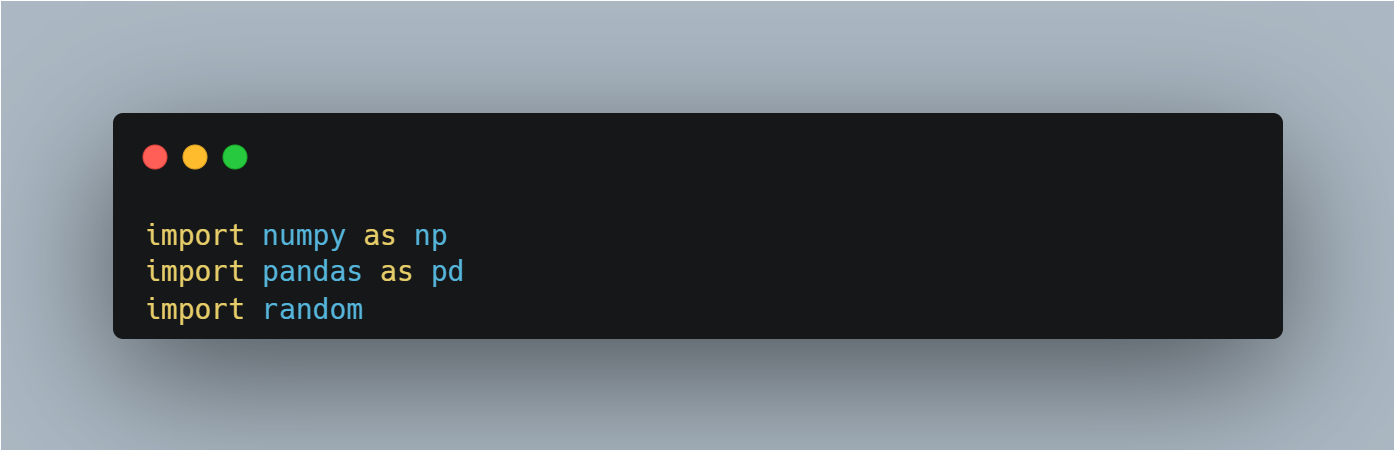
به نام خدا

پروژه دوم درس هوش مصنوعي – شبكه‌هاي عصبي بهاره كاوسي نژاد – 99431217

**تكنيك Cross-Validation:**

تكنيك Cross-Validation در يادگيري ماشين به منظور تخمين مهارت يك مدل يادگيري بر روي داده‌هاي unseen استفاده مي‌شود؛‌ يعني استفاده از يك نمونه محدود به منظور تخمين نحوه عملكرد كلي مدل، زماني كه براي پيش بيني داده‌هايي كه در طول آموزش مورد استفاده قرار نگرفته‌اند.

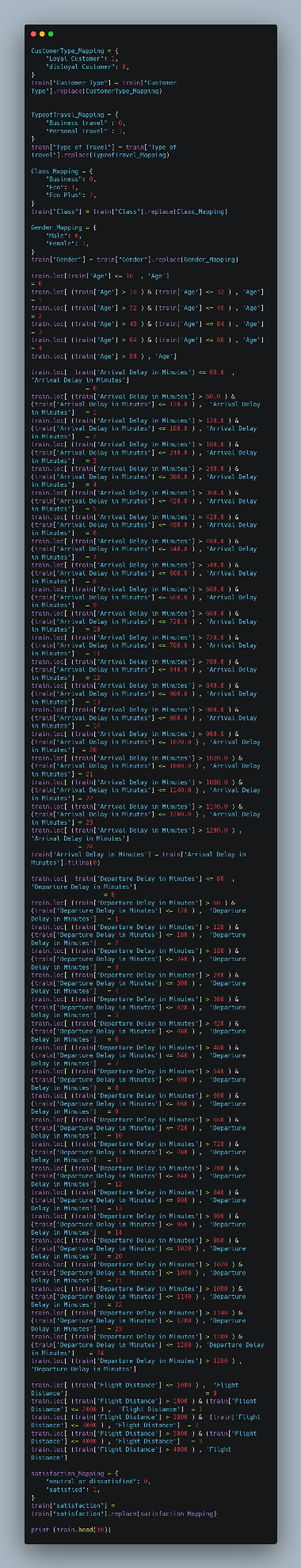
**بخش اول:**



ابتدا كتابخانه هاي لازم را import مي‌كنيم.

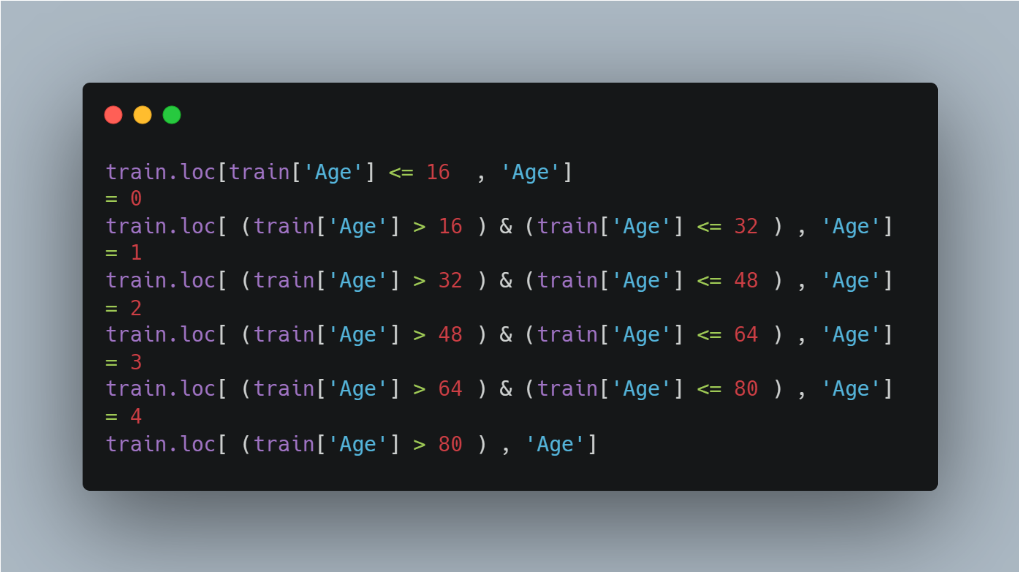


با دستور pd.read\_csv فايل csv را مي‌خوانيم و دو ستون اول آن را حذف مي‌كنيم زيرا شماره سطر و id هستند كه attribute نيستند و با دستور info اطلاعات attribute ها را چاپ مي‌كنيم.

در مرحله بعد بايد attribute ها را map كنيم و به آنها مقادير عددي نسبت دهيم. به عنوان مثال mapping ويژگي ‍Customer Type به صورت زير خواهد بود:



همچنين attributeهايي مانند Age و Arrival Delay in Minutes را به صورت بازه map مي‌كنيم:





از اين تابع براي انتخاب نمونه‌هاي رندوم پس از نمونه 2000ام (زيرا 2000 نمونه اول براي تست هستند) براي train كردن استفاده مي‌شود.

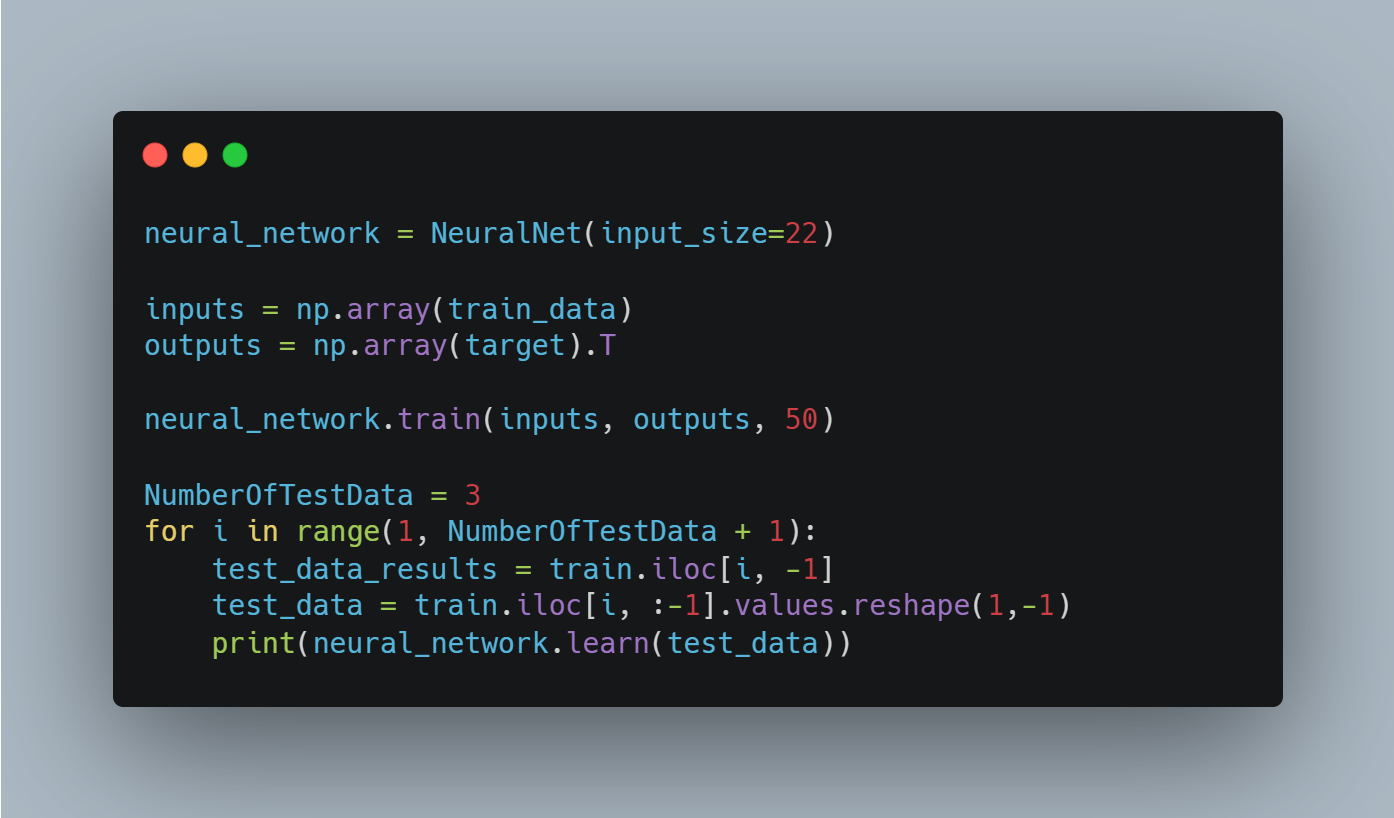


در كلاس NeuralNet شبكه عصبي خود را مي‌سازيم.

* متد \_\_init\_\_: اين متد constructor كلاس است و مقادير تصادفي براي synaptic connectionهاي بين نورون‌ها را توليد مي‌كند.
* متد \_\_sigmoid: اين متد كمكي براي محاسبه sigmoid activation function به ازاي ورودي x است. تابع sigmoid هر مقدار حقيقي را به بازه 0 تا 1 map مي‌كند.
* متد \_\_sigmoid\_derivative: اين متد كمكي براي محاسبه مشتق تابع sigmoid استفاده مي‌شود. اين مقدار، sigmoid curve gradient را تعيين مي‌كند كه در backpropagation براي تنظيم وزن‌ها مورد استفاده قرار مي‌گيرد.
* متد train: اين متد به تعداد training\_iteration بار اجرا مي‌شود و وزن‌ها را بر اساس error محاسبه شده تنظيم مي‌كند. Error به صورت اختلاف بين خروجي مورد انتظار و خروجي تابع learn تعيين مي‌شود.
* داخل حلقه train، وزن‌ها بر اساس error محاسبه شده و مشتق تابع sigmoid تنظيم مي‌شوند. Adjusment factor بر اساس ضرب داخلي transpose ماتريس ورودي و element-wise product ارور و مشتق خروجي محاسبه مي‌شود. سپس اين factor به وزن‌هاي synaptic اضافه مي‌شود.
* متد learn، به عنوان خروجي شبكه عصبي را برمي‌گرداند. اين متد ضرب داخلي ورودي ها و وزن‌هاي synaptic را محاسبه كرده و نتيجه را به activation function مي‌دهد تا خروجي به دست آيد.



داده‌هاي train و attribute‌ها را مانند پروژه قبل مشخص مي‌كنيم.



شبكه عصبي را مي‌سازيم و به عنوان input\_size تعداد attributeها را مي‌دهيم. سپس آن را train مي‌كنيم.

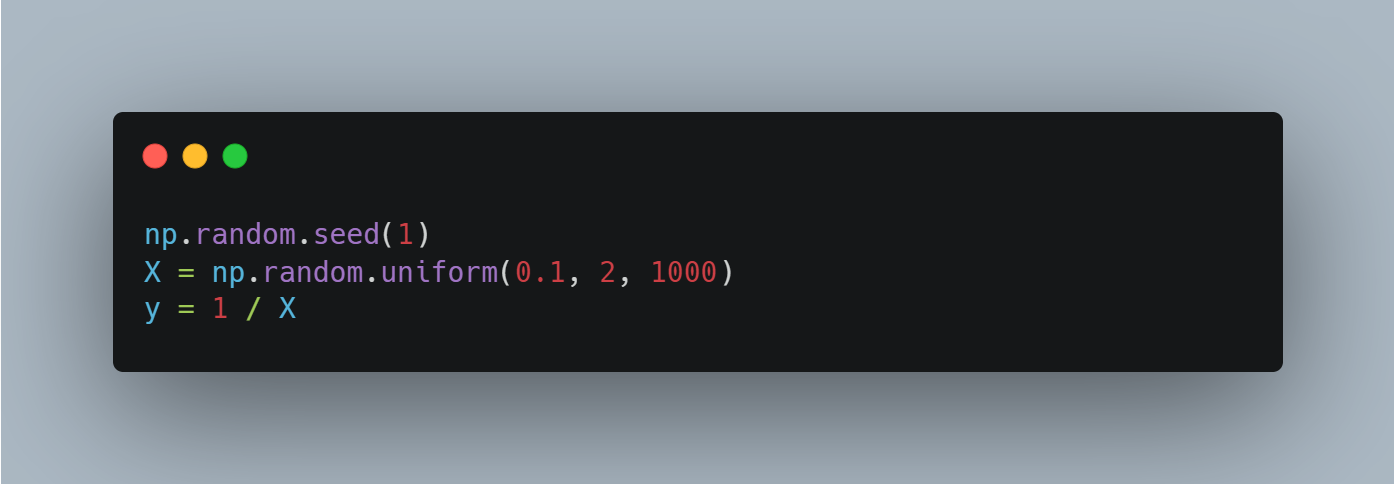
**دقت درخت تصميم و شبكه عصبي:**

دقت درخت تصميم و شبكه عصبي بسيار نزديك به هم است اما دقت شبكه عصبي اندكي بيشتر است.

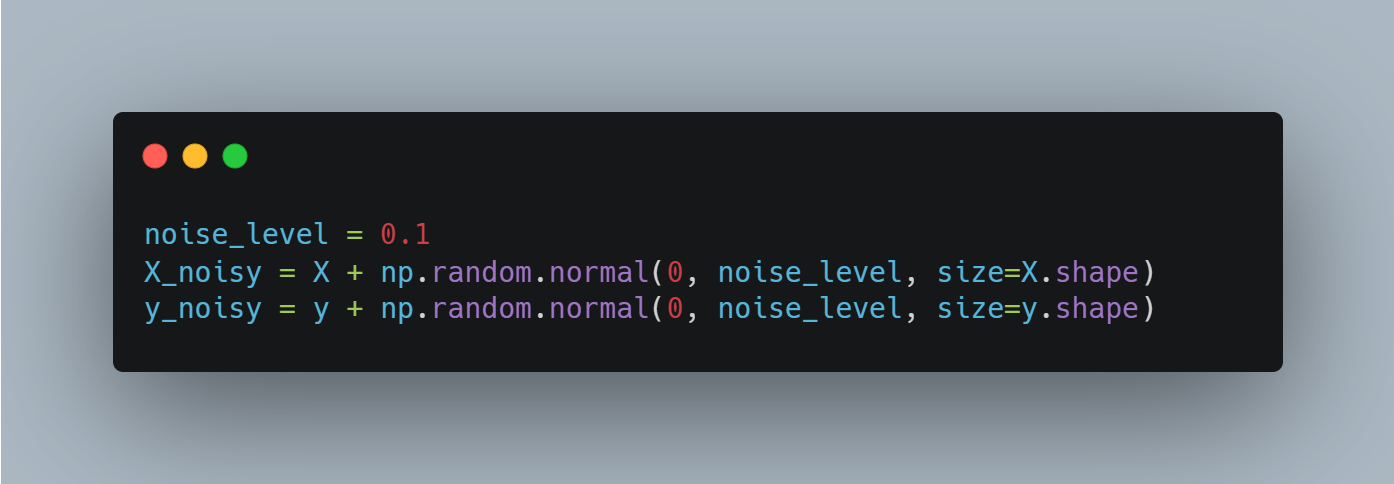
**بخش دوم و سوم:**



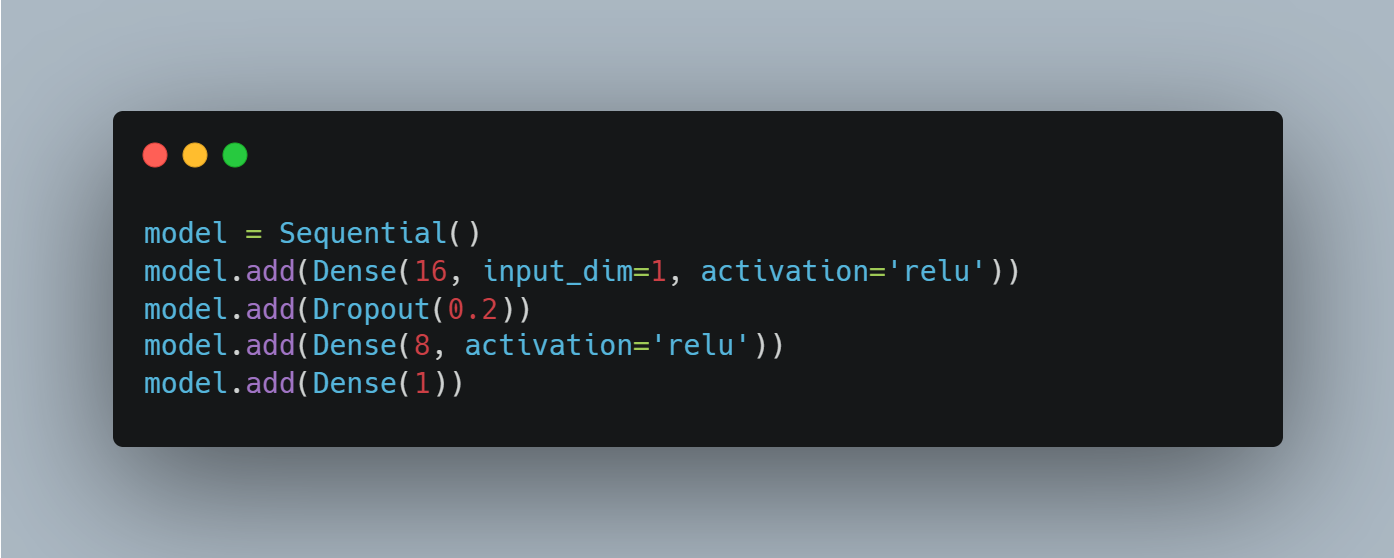
ابتدا كتابخانه‌هاي لازم را import مي‌كنيم.



مقادير X را به صورت تصادفي و مقادير y را بر اساس تابع 1/X تعيين مي‌كنيم.



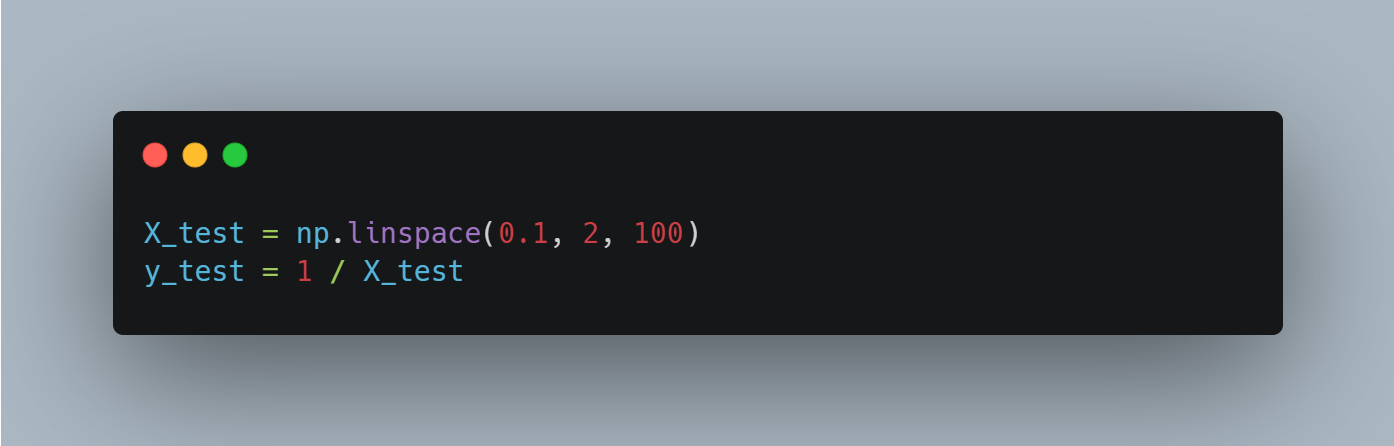
براي بخش سوم به مقادير x و y نويز اضافه مي‌كنيم.



* ابتدا يك instance از كلاس Sequential مي‌سازيم كه يك stack خطي از لايه‌هاست و به ما اجازه مي‌دهد كه شبكه‌هاي عصبي را با اضافه كردن لايه‌ها به ترتيب بسازيم.
* يك لايه fully connected به مدل اضافه مي‌كنيم. اين Dense layer يك لايه است كه در آن هر نورون به همه نورون‌هاي لايه قبل خود متصل شده است. در اينجا داراي 16 نورون است. پارامتر input\_dim تعداد ابعاد داده ورودي را مشخص مي‌كند. پارامتر activation=’relu’، تابع فعال سازي rectified linear unit (ReLU) را براي اين لايه مشخص مي‌كند. ReLU در لايه‌هاي پنهان براي non-linearity استفاده مي‌شود.
* در خط بعدي يك لايه Dropout به مدل اضافه مي‌كنيم. Dropout يك تكنيك regularization است كه به صورت تصادفي بخشي از ورودي‌ها را در حين training، صفر قرار مي‌دهد تا از overfitting و وابستگي بين نورون‌ها جلوگيري كند.
* سپس يك لايه fully connected ديگر با 8 نورون اضافه مي‌كنيم.
* در انتها يك لايه خروجي به مدل اضافه مي‌كنيم. اين لايه نيز fully connected است و يك نورون دارد و خروجي پيش بيني شده را نشان مي‌دهد. Activation Function به صورت پيش فرض خطي تعيين مي‌شود.



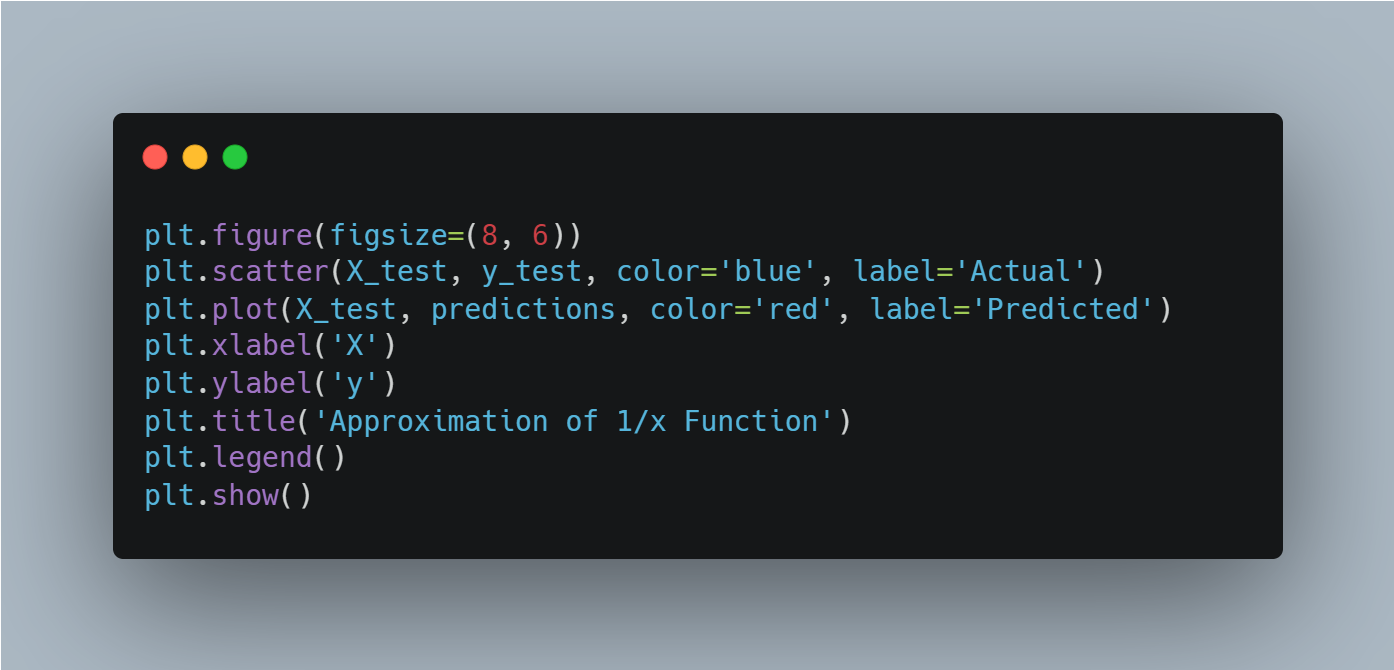
مدل را كامپايل و train مي‌كنيم.



داده‌هاي test را generate مي‌كنيم.

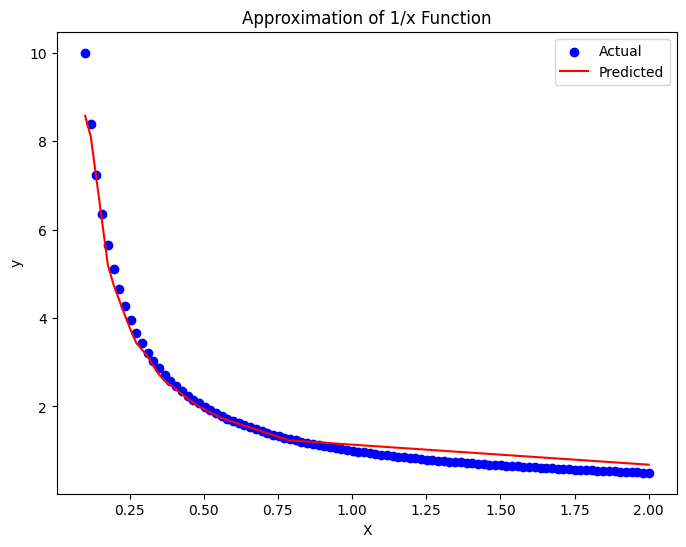


داده‌هاي پيش بيني شده را توليد مي‌كنيم.

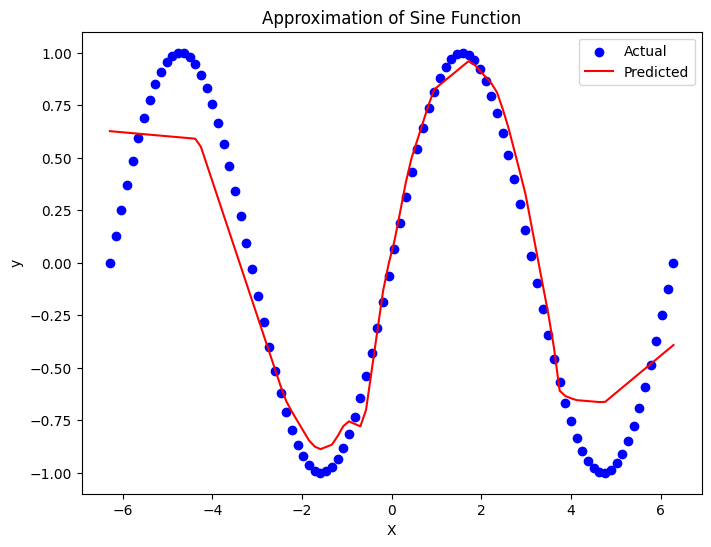


نتايج را رسم مي‌كنيم.

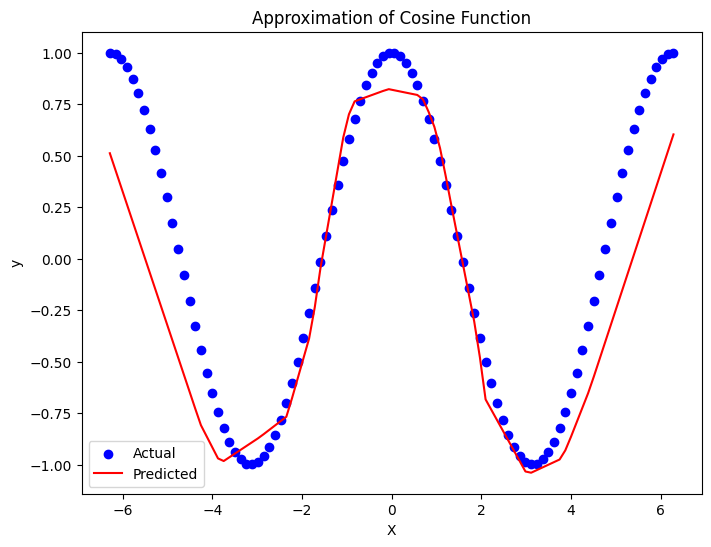
تابع 1/x:



تابع sin x:

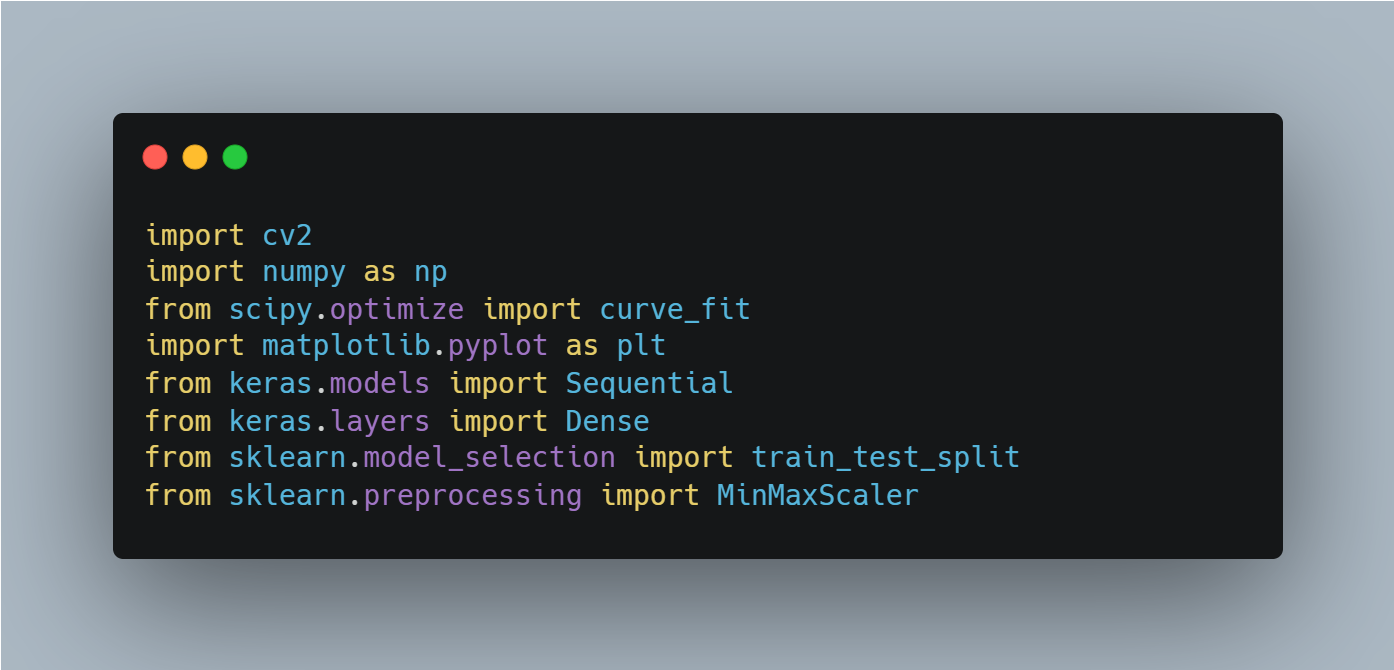


تابع cos x:

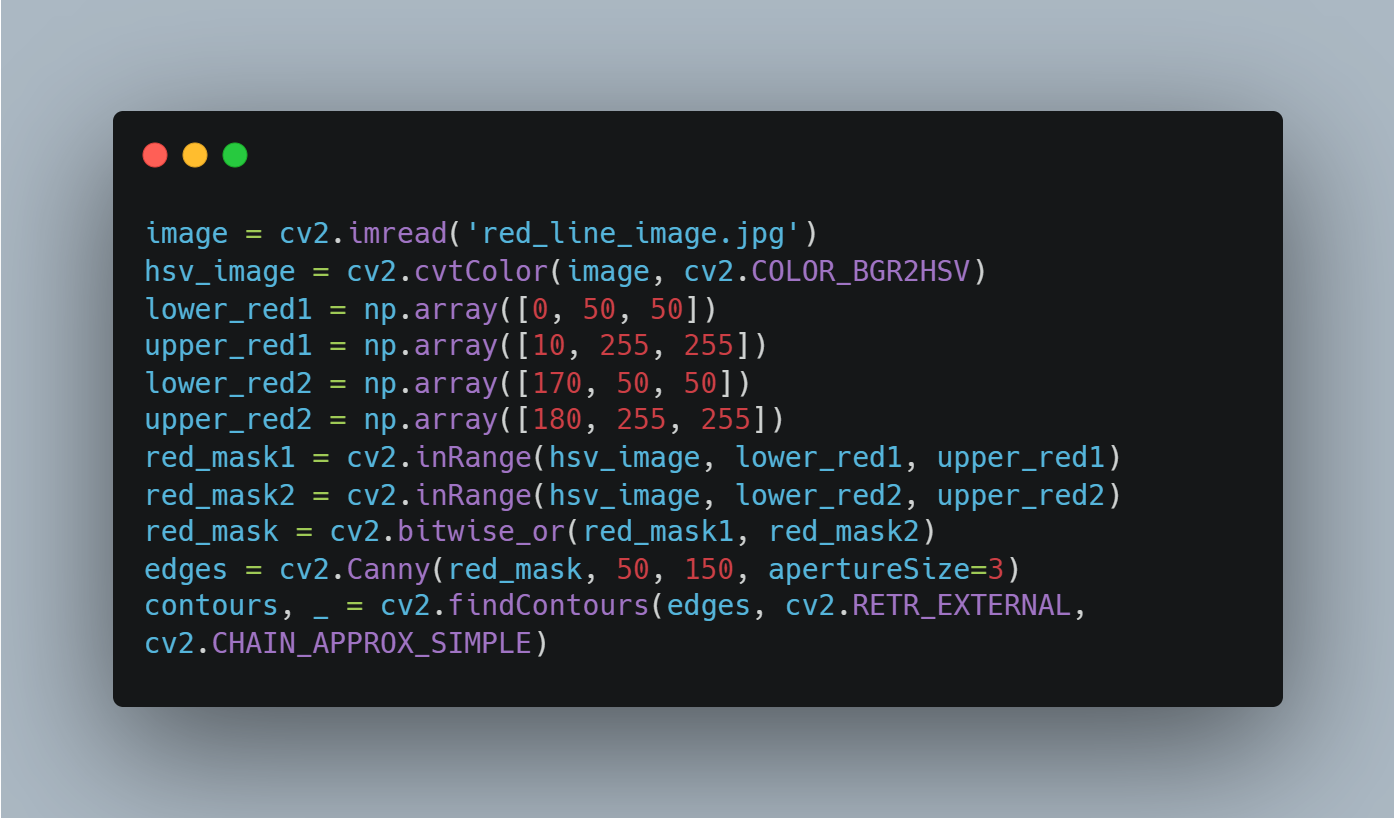


هرچه تعداد نقاط ورودي و تعداد لايه‌هاي شبكه و نورون‌هاي هرلايه بيشتر باشد، دقت افزايش مي‌يابد و هرچه تابع پيچيده‌تر باشد، دقت كاهش مي‌يابد.

**بخش چهارم:**



ابتدا كتابخانه‌هاي لازم را import مي‌كنيم.

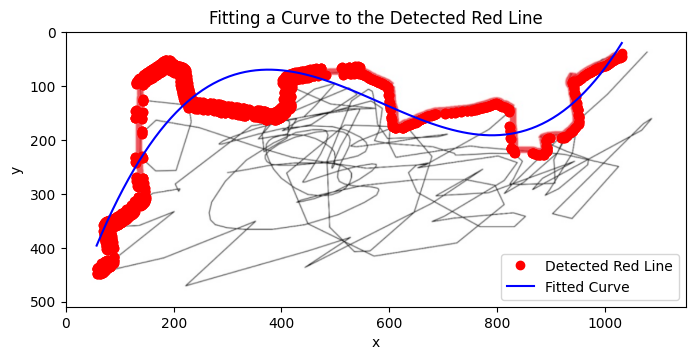


فرايند image processing را با كتابخانه OpenCV انجام مي‌دهيم.

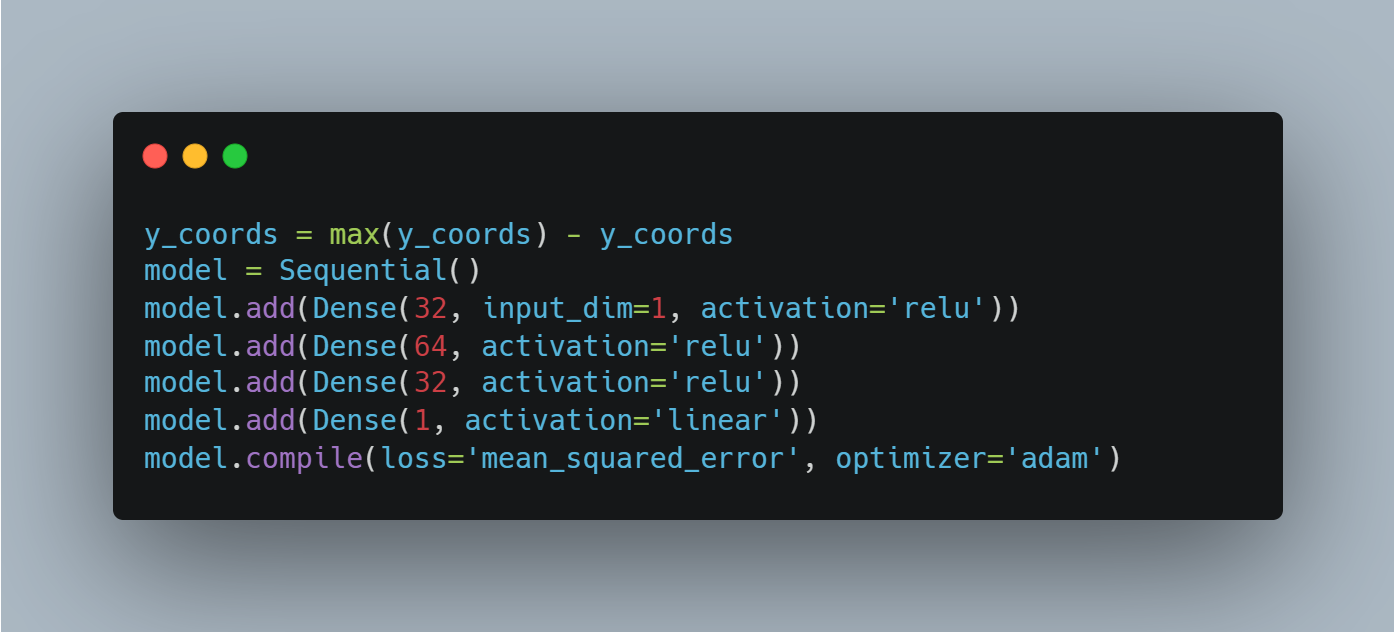
* تصوير مورد نظر را load مي‌كنيم.
* تصوير load شده را از BGR به HSV تبديل مي‌كنيم. HSV معمولا براي image processingهايي كه بر اساس رنگ هستند استفاده مي‌شود.
* چند بازه براي رنگ قرمز تعريف مي‌كنيم و سپس آنها را به صورت bitwise، or مي‌كنيم.
* الگوريتم Canny edge را روي تصوير اعمال مي‌كنيم. اين الگوريتم بر اساس gradientها و thresholding عمل مي‌كند و لبه‌ها را پيدا مي‌كند.
* در مرحله بعد حدفاصل‌ها (contours) را پيدا مي‌كنيم.



* اگر حدفاصل‌ها پيدا شدند، ماكسيمم آنها را خط قرمز در نظر مي‌گيريم و مختصات خط را پيدا مي‌كنيم.
* سپس يك تابع polynomial براي خط پيدا شده fit مي‌كنيم (درجه آن را 3 در نظر مي‌گيريم).
* تابع پيدا شده را رسم مي‌كنيم.



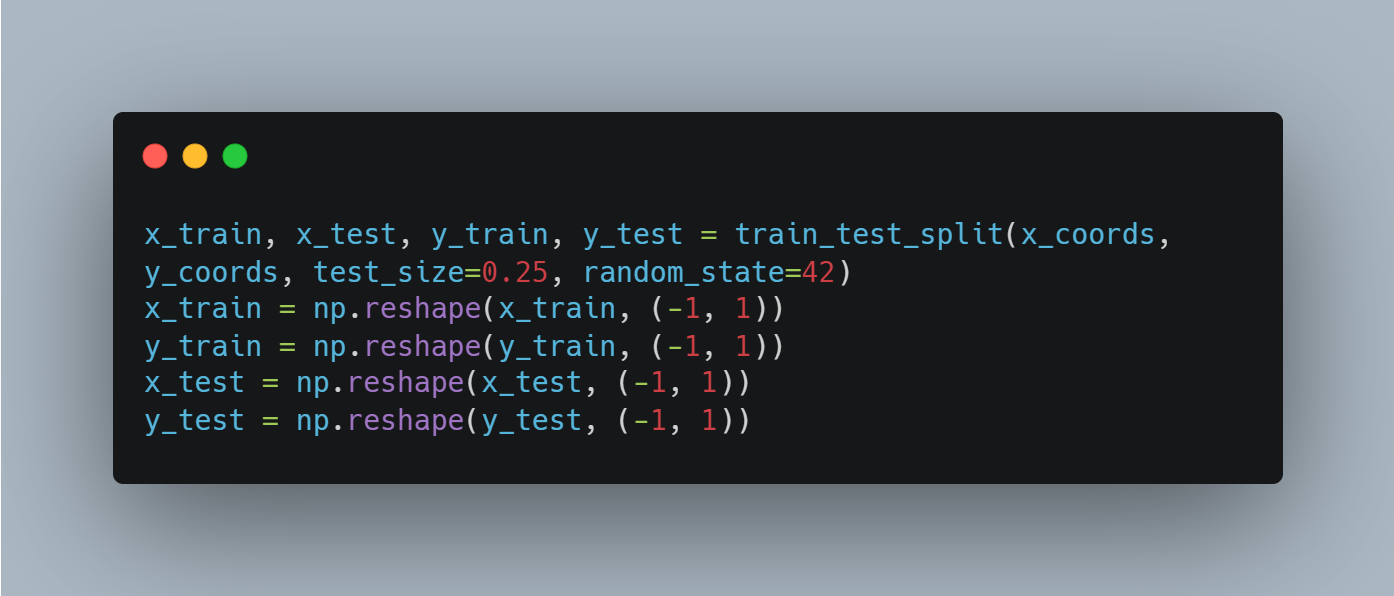
در اين تصوير تشخيص خط قرمز و fit كردن يك تابع درجه 3 نشان داده شده است.



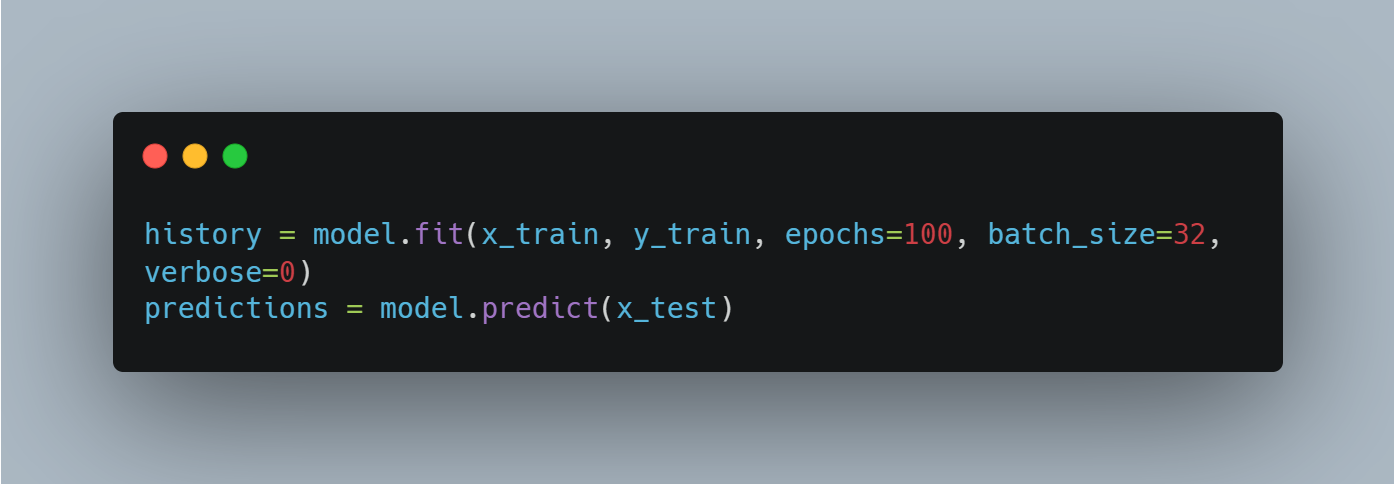
به ساخت شبكه عصبي مي‌پردازيم:

* ابتدا يك instance از كلاس Sequential مي‌سازيم كه يك stack خطي از لايه‌هاست و به ما اجازه مي‌دهد كه شبكه‌هاي عصبي را با اضافه كردن لايه‌ها به ترتيب بسازيم.
* يك لايه fully connected به مدل اضافه مي‌كنيم. اين Dense layer يك لايه است كه در آن هر نورون به همه نورون‌هاي لايه قبل خود متصل شده است. در اينجا داراي 32 نورون است. پارامتر input\_dim تعداد ابعاد داده ورودي را مشخص مي‌كند. پارامتر activation=’relu’، تابع فعال سازي rectified linear unit (ReLU) را براي اين لايه مشخص مي‌كند. ReLU در لايه‌هاي پنهان براي non-linearity استفاده مي‌شود.
* يك لايه fully connected ديگر با 64 نورون اضافه مي‌كنيم.
* يك لايه fully connected ديگر با 32 نورون اضافه مي‌كنيم.
* در انتها يك لايه خروجي به مدل اضافه مي‌كنيم. اين لايه نيز fully connected است و يك نورون دارد و خروجي پيش بيني شده را نشان مي‌دهد. Activation Function خطي تعيين مي‌شود.

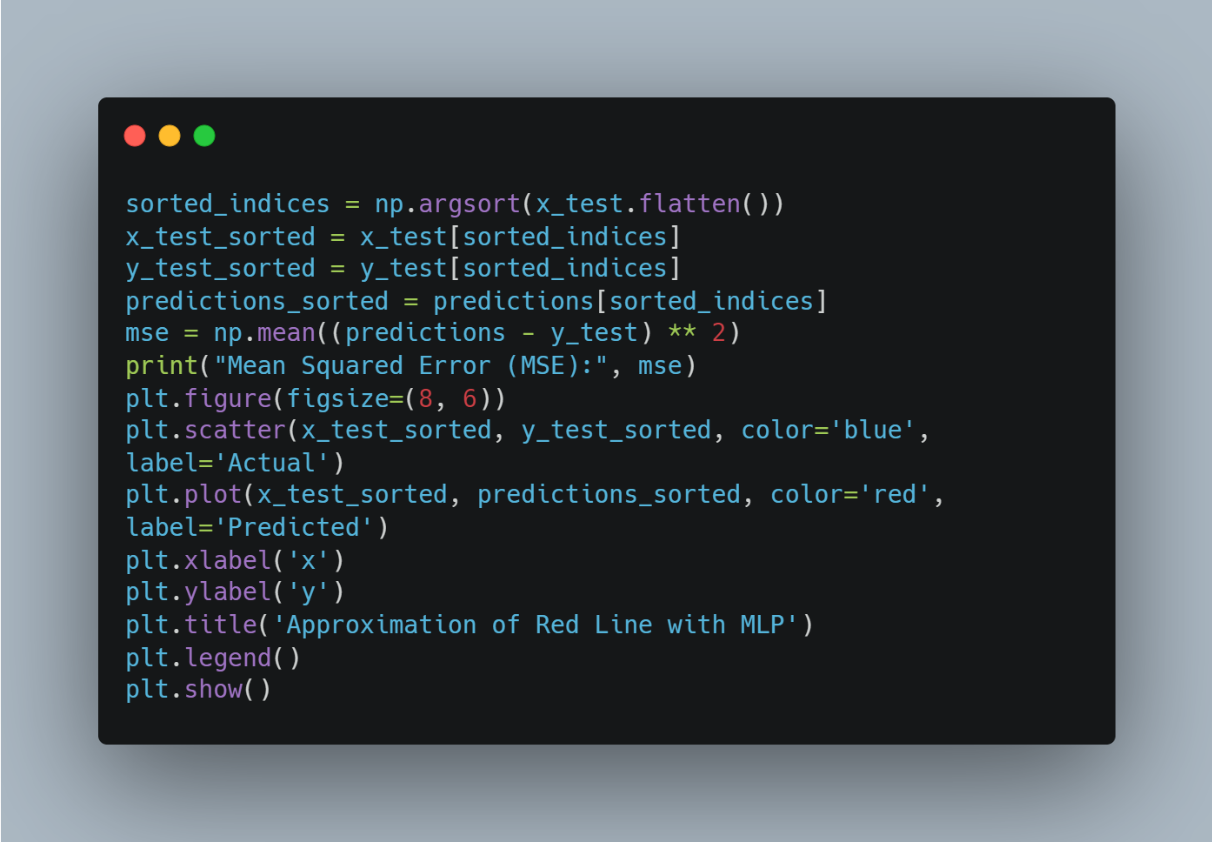
مدل را كامپايل مي‌كنيم.



داده‌هاي train و test را جدا مي كنيم و براي تطابق با مدل آرايه‌هاي يك بعدي را به دو بعدي تبديل مي‌كنيم.



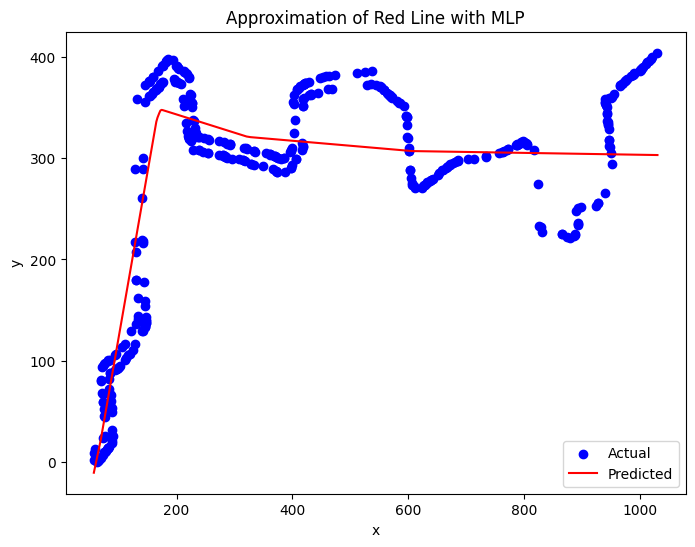
مدل را train كرده و پيش بيني ها را انجام مي‌دهيم.



داده‌هاي تست را sort مي‌كنيم و مدل را ارزيابي و رسم مي‌كنيم.

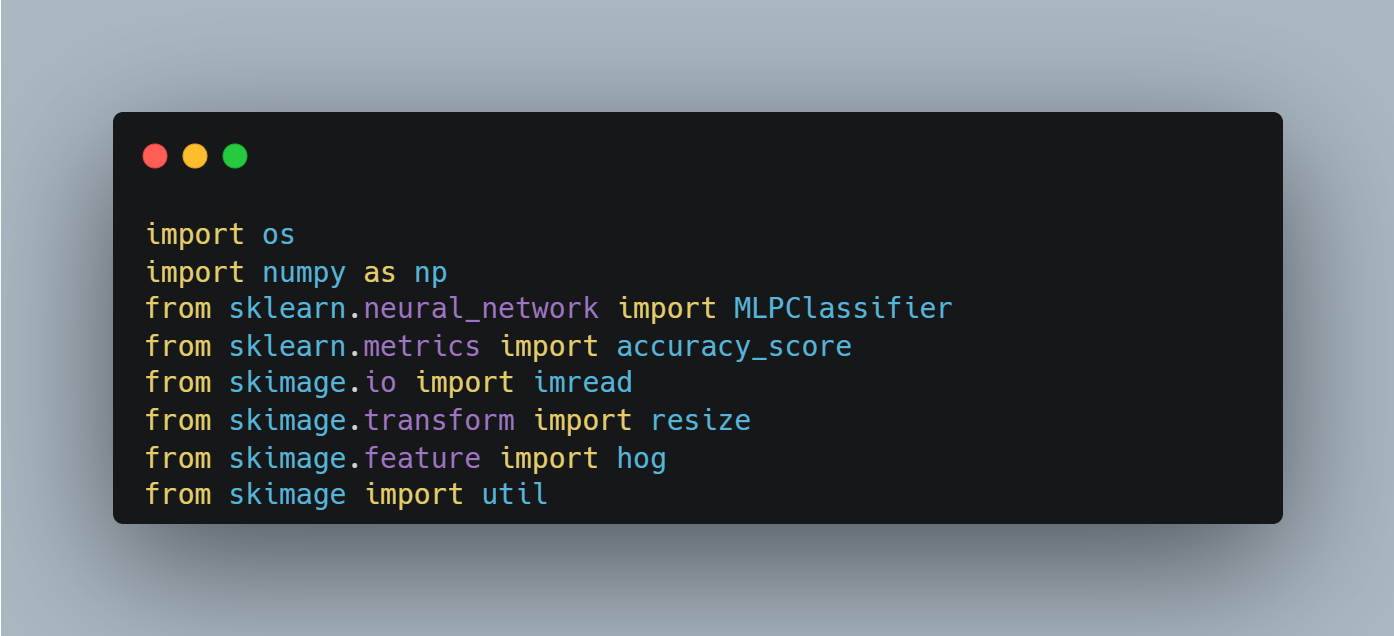
مقدار MSE:

Mean Squared Error (MSE): 2458.833037601837

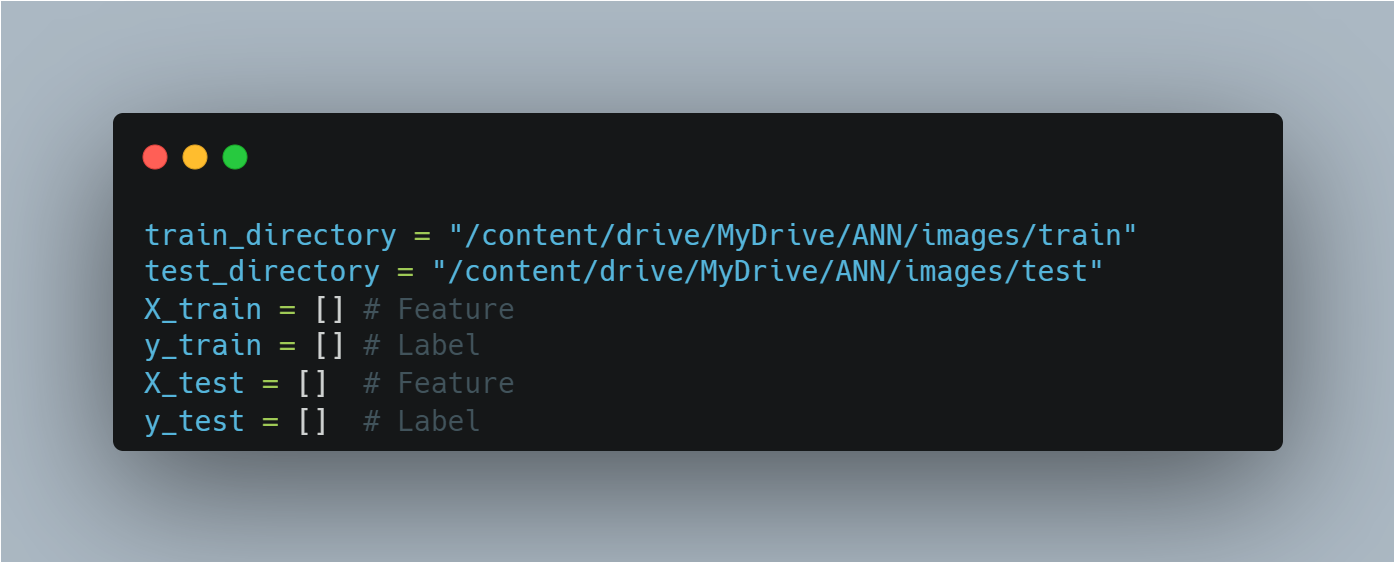


**بخش پنجم:**

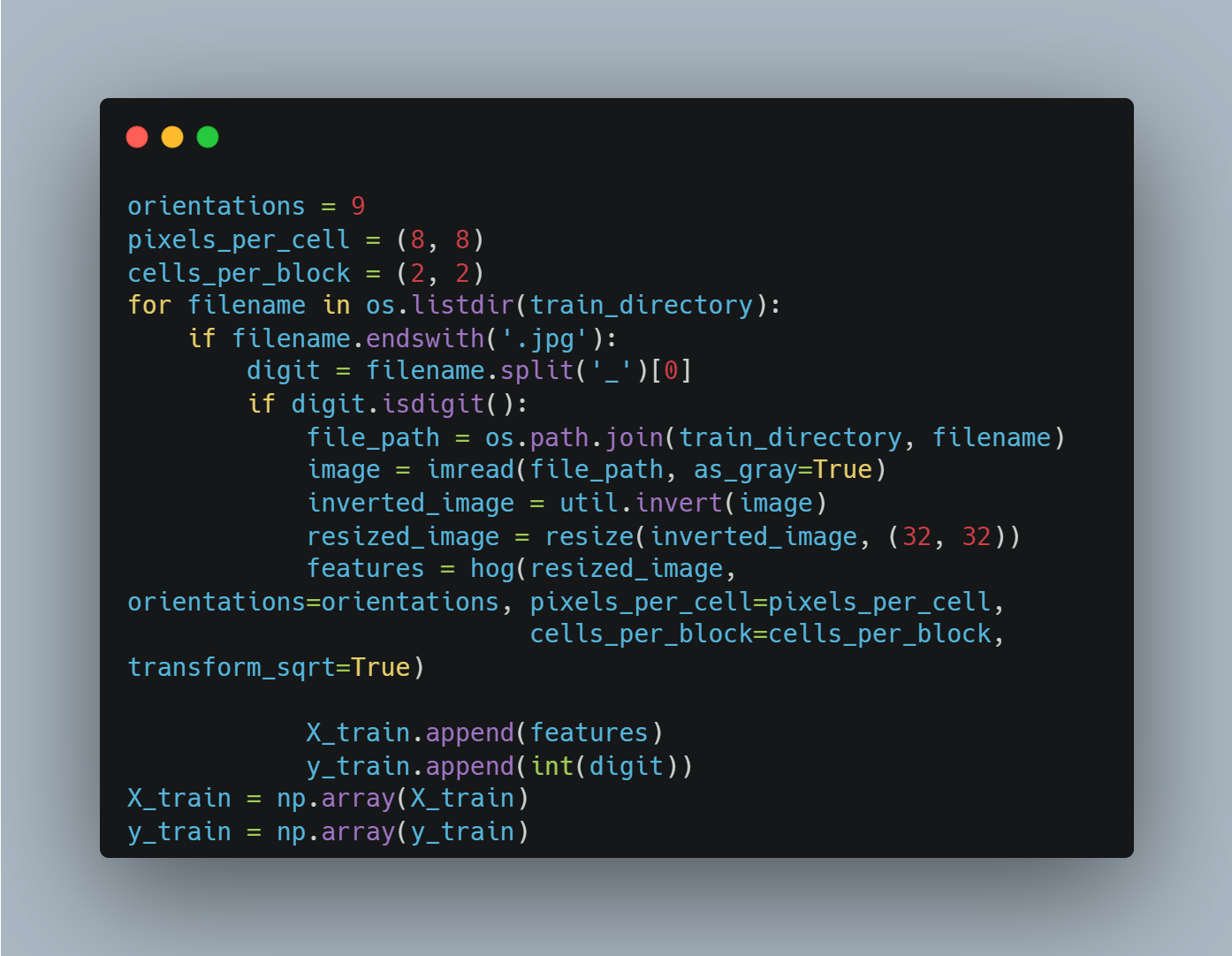
در اين بخش از dataset پيشنهادي USPS استفاده شده است.



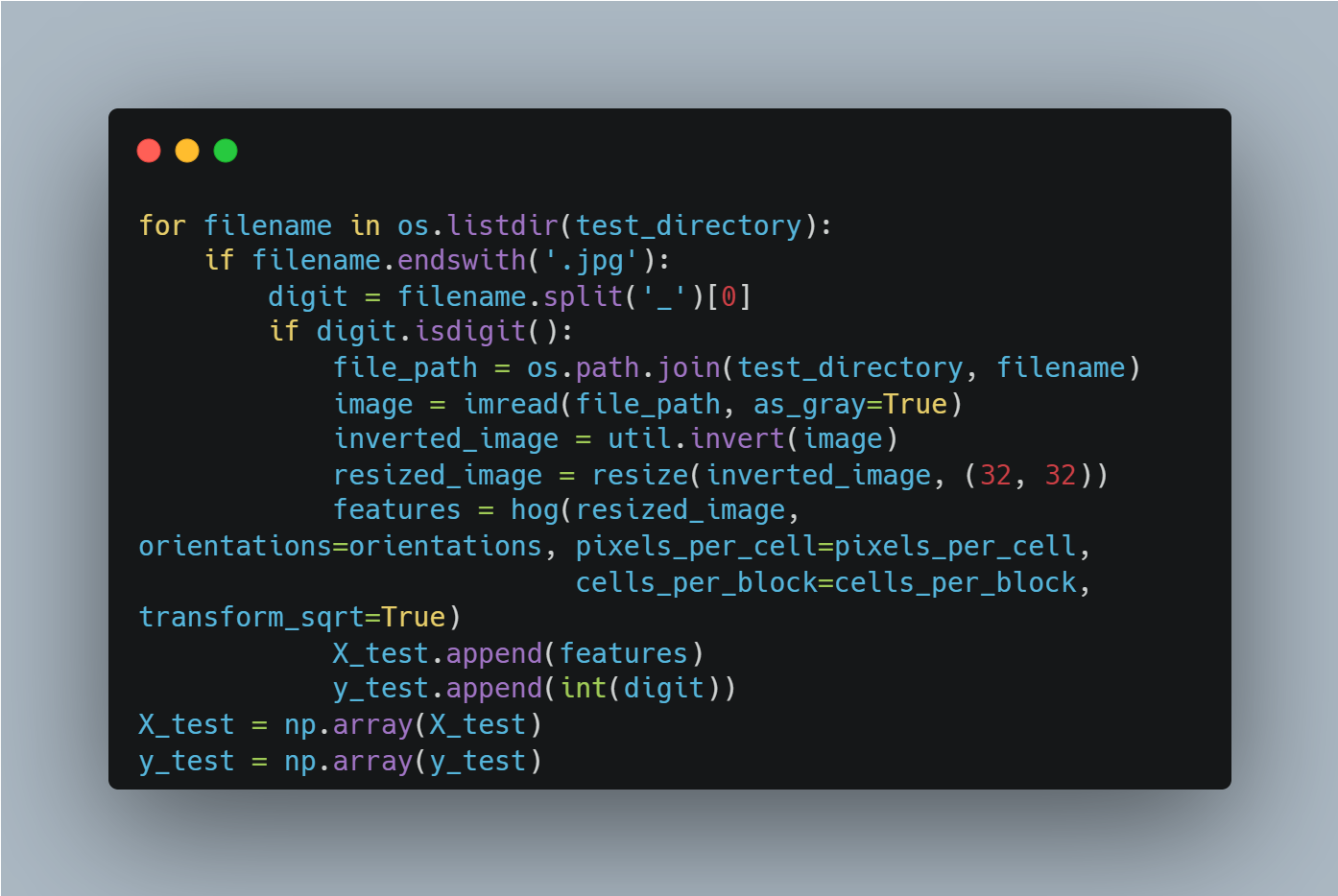
ابتدا كتابخانه‌هاي لازم را import مي‌كنيم.



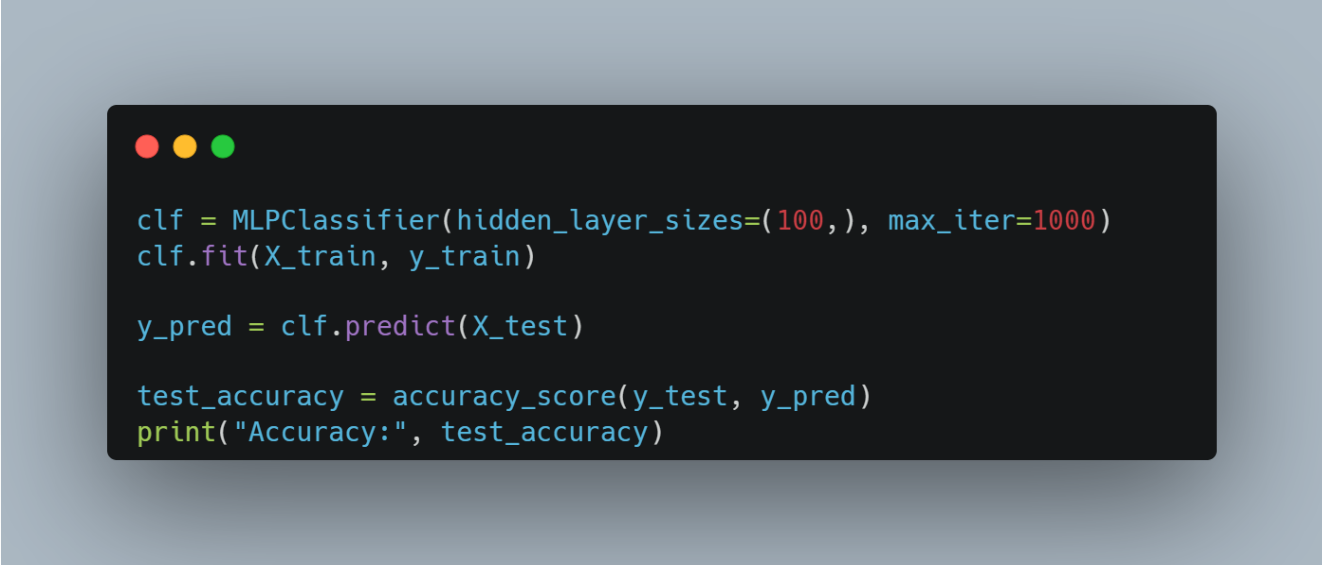
پوشه‌اي كه داده‌هاي train و test در آن قرار دارند را مشخص كرده و آرايه‌هايي براي داده‌هاي test و train، initialize مي‌كنيم.



* به ازاي تمامي تصاويري كه در پوشه train قرار دارند، برچسب آن تصوير را جدا مي‌كنيم و مطمئن مي‌شويم كه رقم است.
* آدرس تصوير را به دست آورده و آن را به grayscale تبديل مي‌كنيم.
* تصوير را invert مي‌كنيم يعني زمينه سياه آن را به سفيد تبديل كرده و رقم سفيد را به سياه تبديل مي‌كنيم تا مطمئن شويم كه رقم در برابر زمينه مشخص است.
* همه تصاوير را به يك اندازه مشخص (32 x 32 pixels)، resize مي‌كنيم.
* مقدار Histogram of Oriented Gradient (HOG) تصوير را به دست مي‌آوريم. HOG يك ويژگي است كه اطلاعات local gradient تصوير را ذخيره مي‌كند. اين اطلاعات براي تشخيص يك object در تصوير مورد استفاده قرار مي‌گيرند.
* مقادير HOG محاسبه شده را در X\_train و برچسب رقم هر تصوير را در y\_train ذخيره كرده و در انتها اين دو ليست را به Numpy array تبديل مي‌كنيم.



مراحل بالا را براي داده‌هاي تست نيز انجام مي‌دهيم.



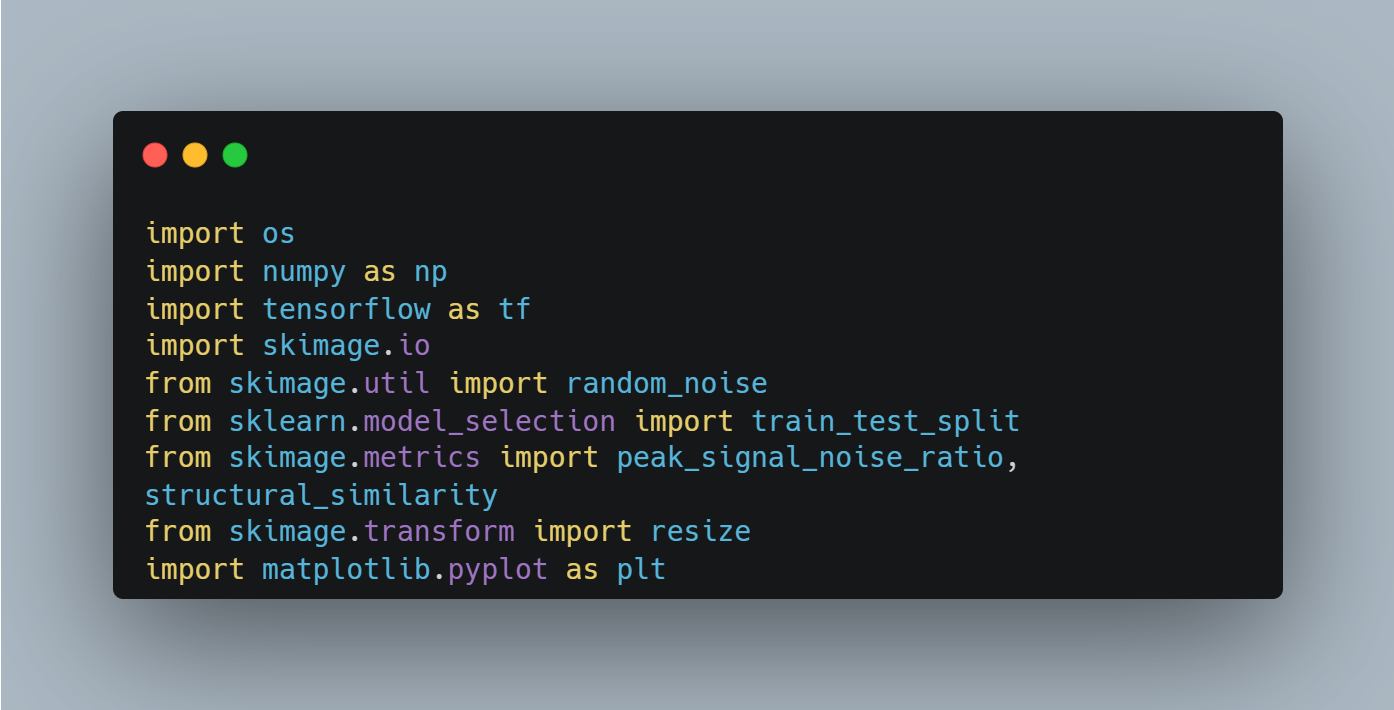
يك MLP (Multi-Layer Perceptron) classifier مي‌سازيم.

* تعداد نورون‌ها در هر لايه پنهان را 100 نورون تعيين مي‌كنيم.
* حداكثر تعداد iterationها براي train كردن MLP را 1000 تعيين مي‌كنيم.

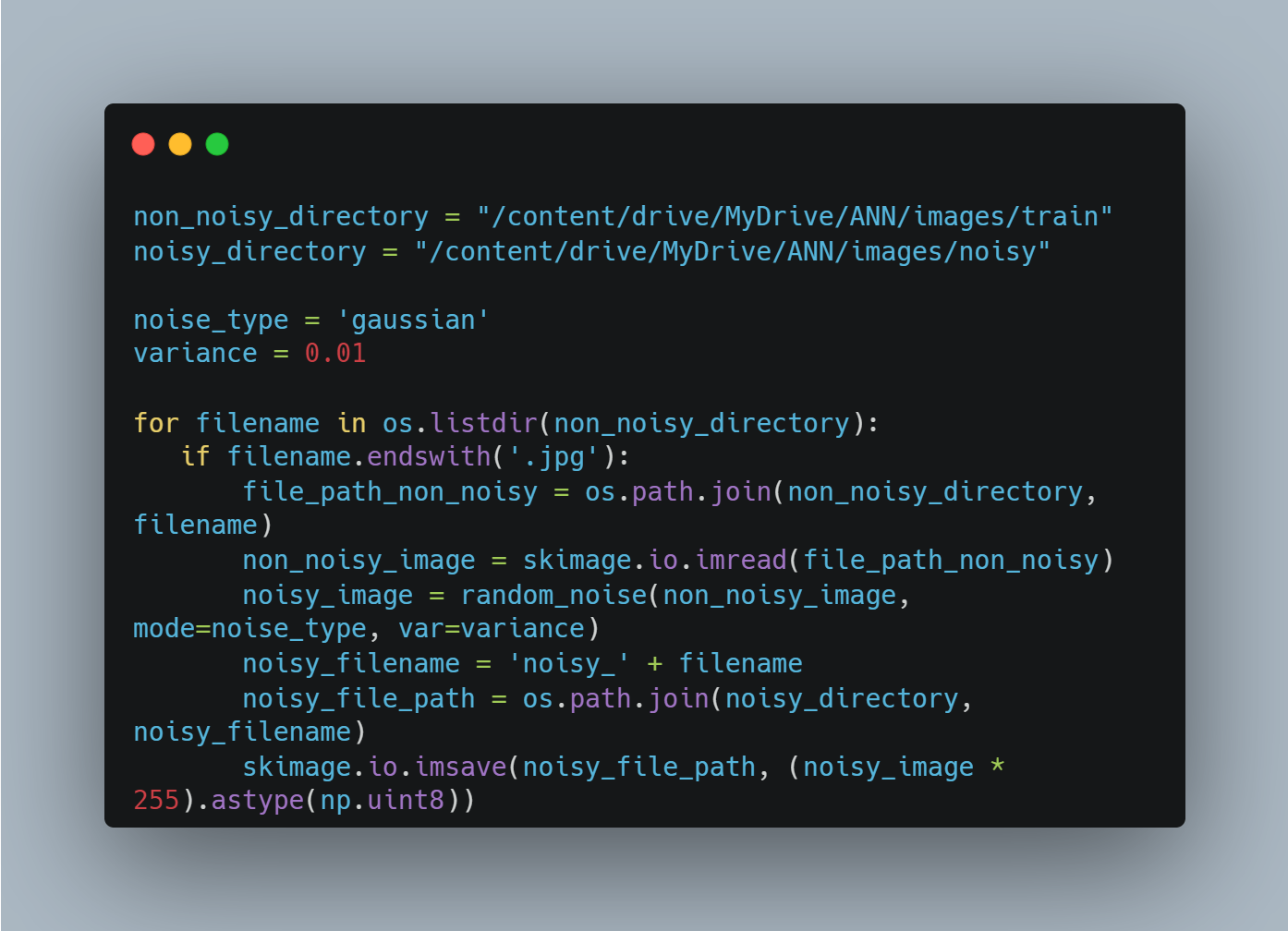
دقت به دست آمده:

Accuracy: 0.9586447433981067

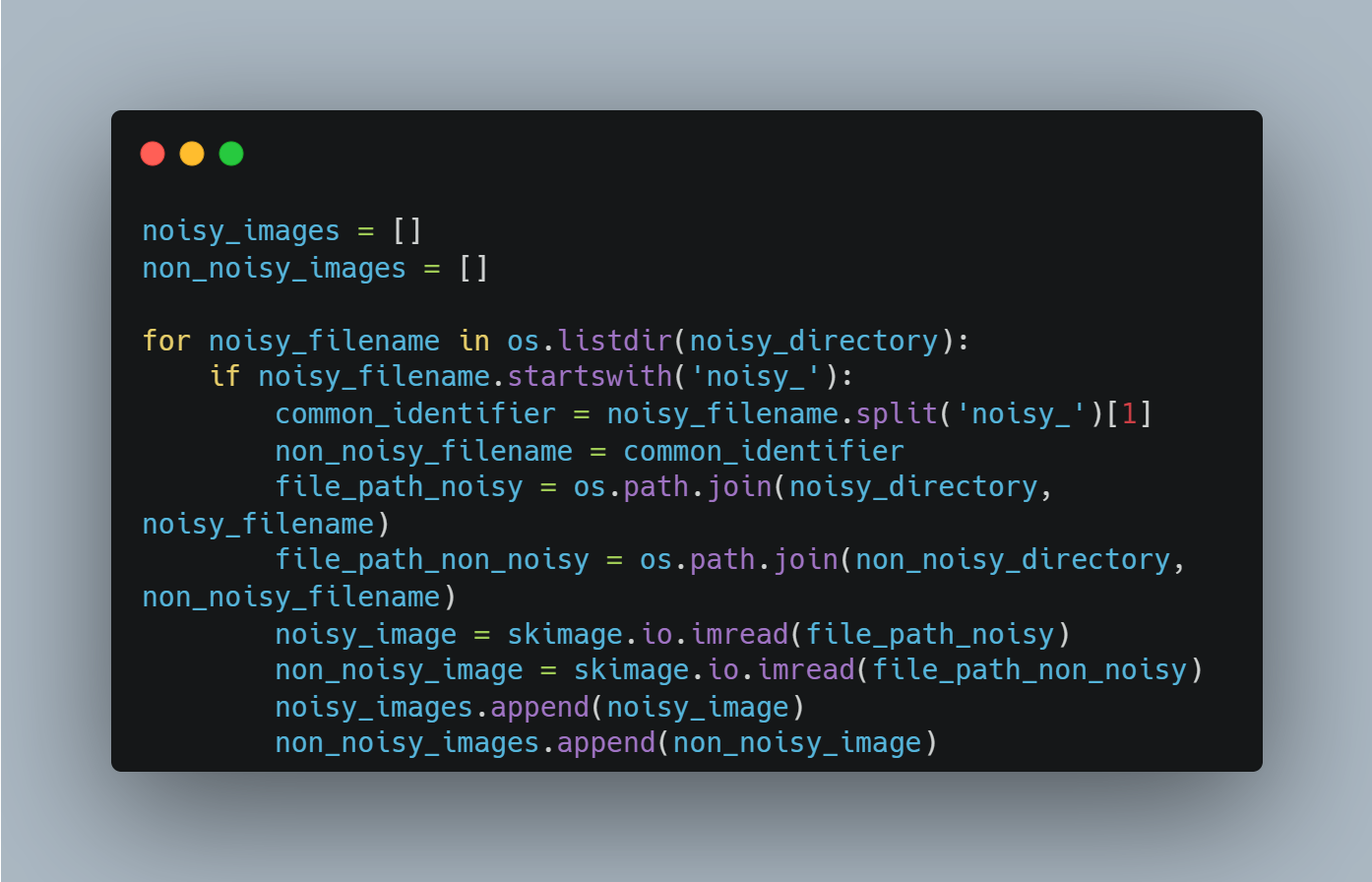
**بخش ششم:**

****

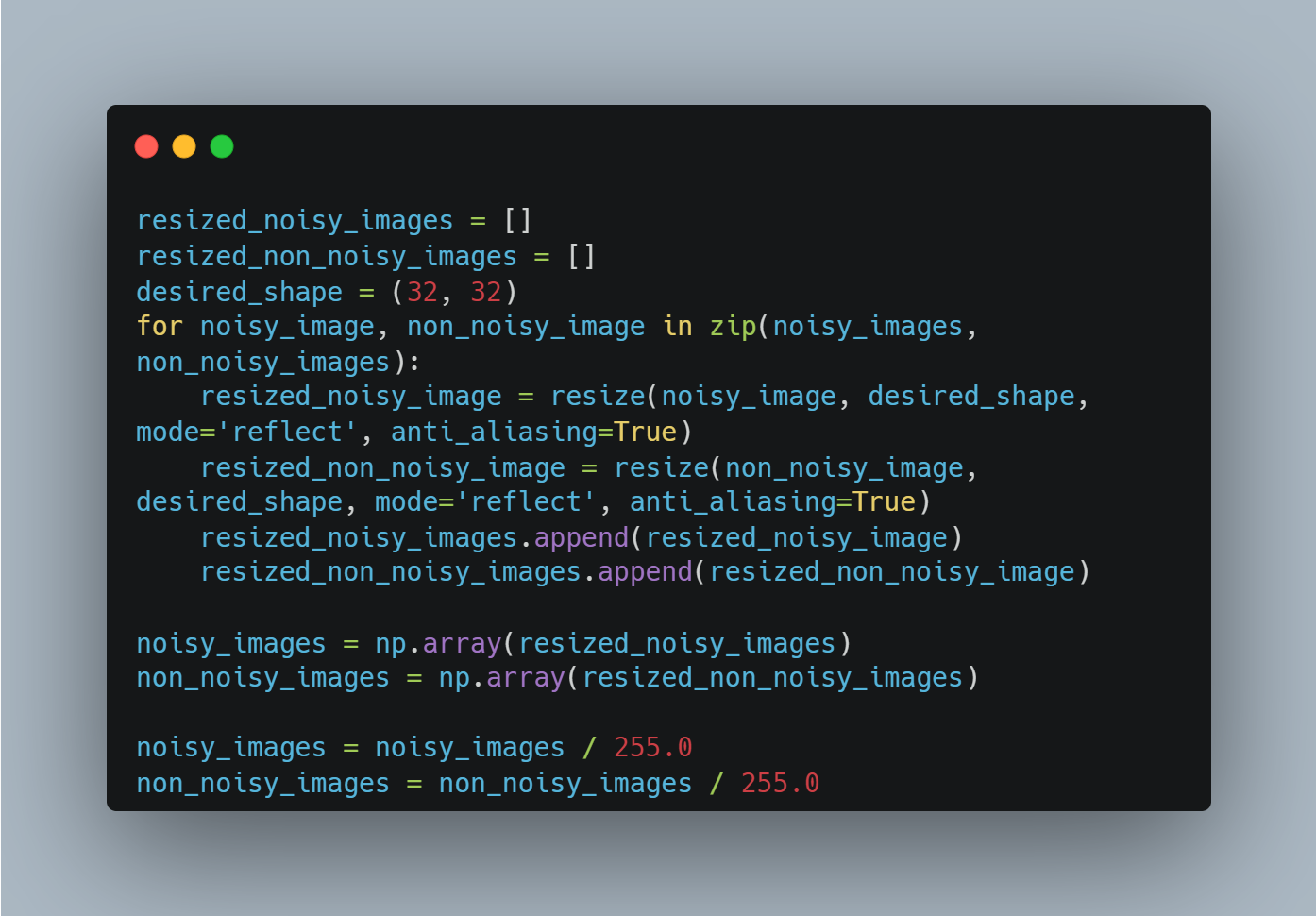
ابتدا كتابخانه‌هاي لازم را import مي‌كنيم.

****

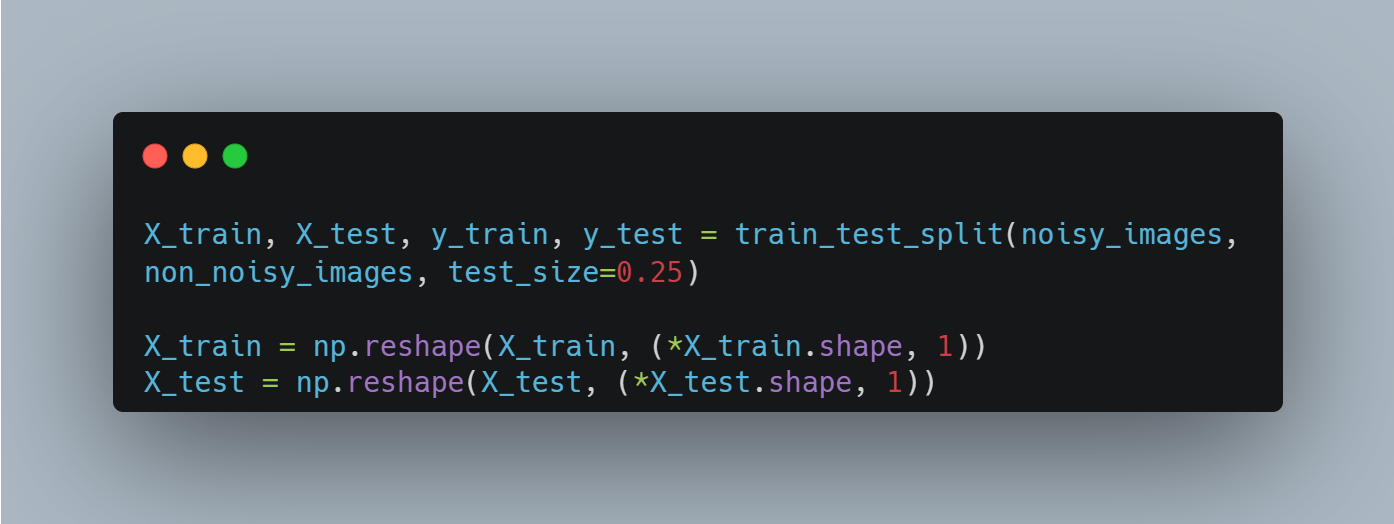
* پوشه‌اي كه در آن فايل‌هاي اصلي قرار دارند و پوشه‌اي كه مي‌خواهيم تصاوير noisy را در آن ذخيره كنيم را مشخص مي‌كنيم.
* نوع نويز را Gaussian و وارياس آن را 0.01 در نظر مي‌گيريم. با تغيير اين مقدار مي‌توان مقدار نويز را تغيير داد.
* نويز را روي تصاوير اعمال مي‌كنيم و تصاوير noisy را ذخيره مي‌كنيم.

****

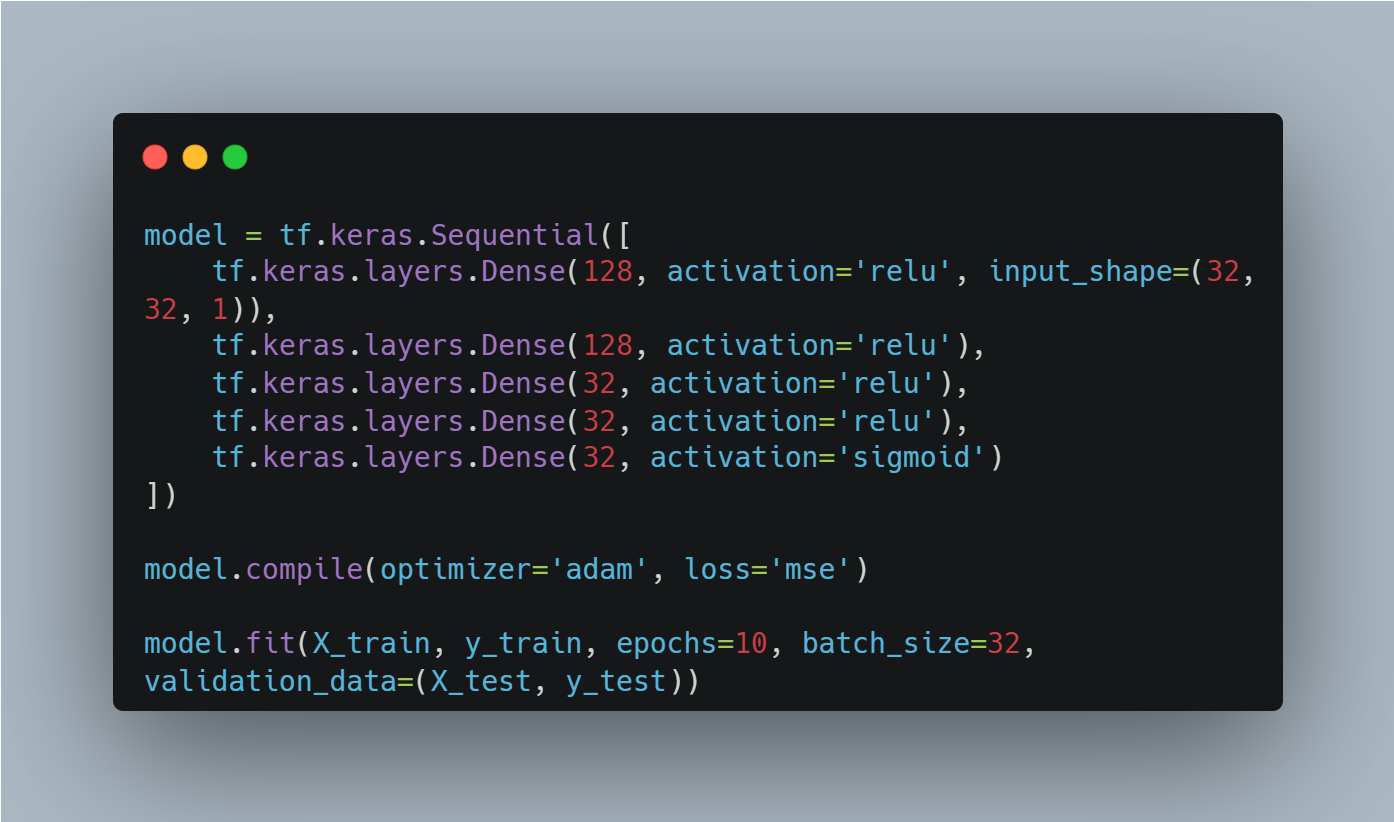
بر اساس نام تصاوير اصلي و noisy، ورودي (تصوير noisy) و خروجي مورد نظر (تصوير اصلي و بدون نويز) را مشخص مي‌كنيم.

****

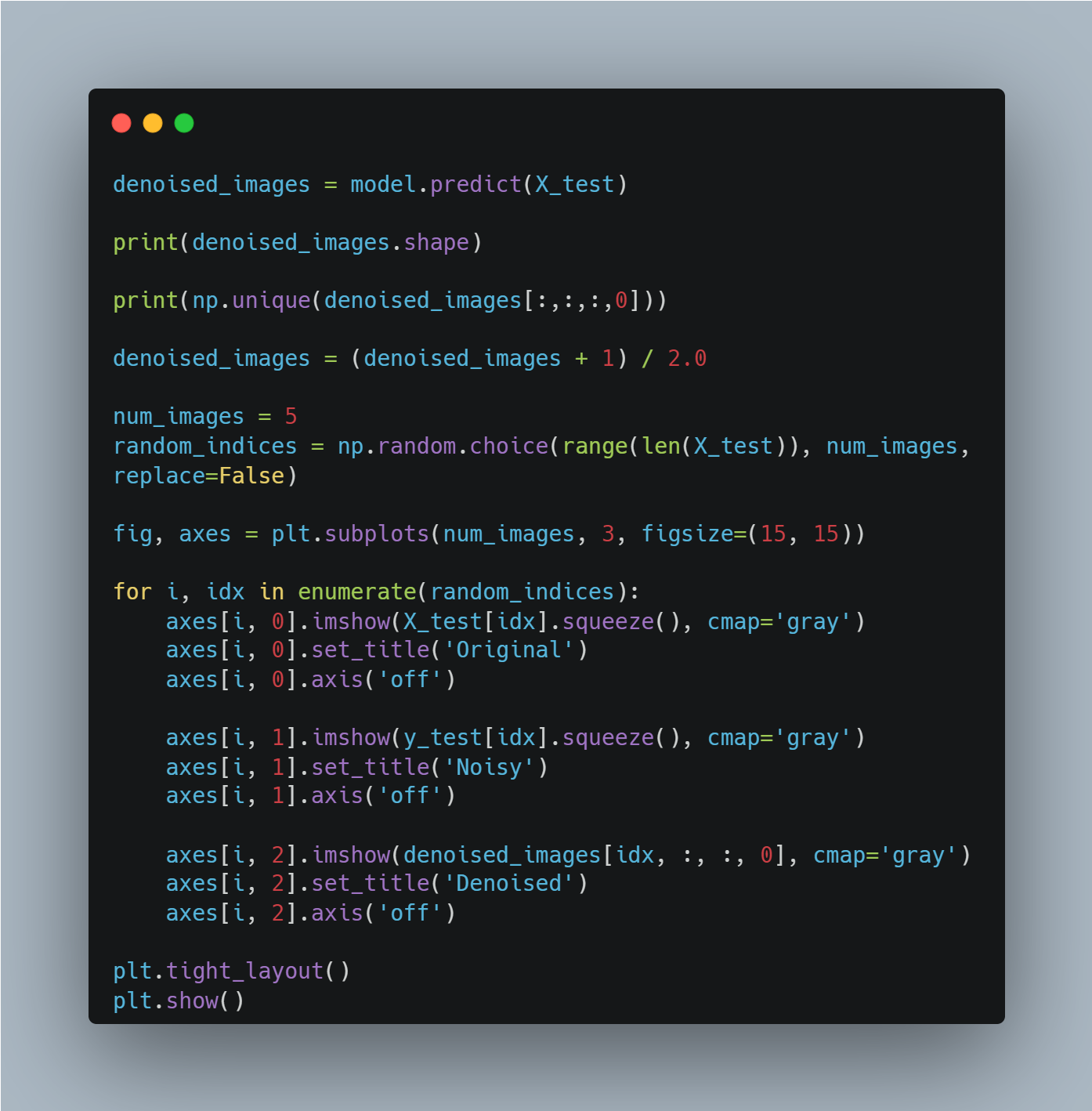
* تمامي تصاوير را به ابعاد مشخصي (تمامي تصاوير را به ابعاد مشخصي (32 x 32 pixels)، resize مي‌كنيم.
* ليست تصاوير اصلي و تصاوير noisy را به Numpy array تبديل مي‌كنيم.
* مقادير پيكسل‌ها را به بازه 0 تا 1، normalize مي‌كنيم.

****

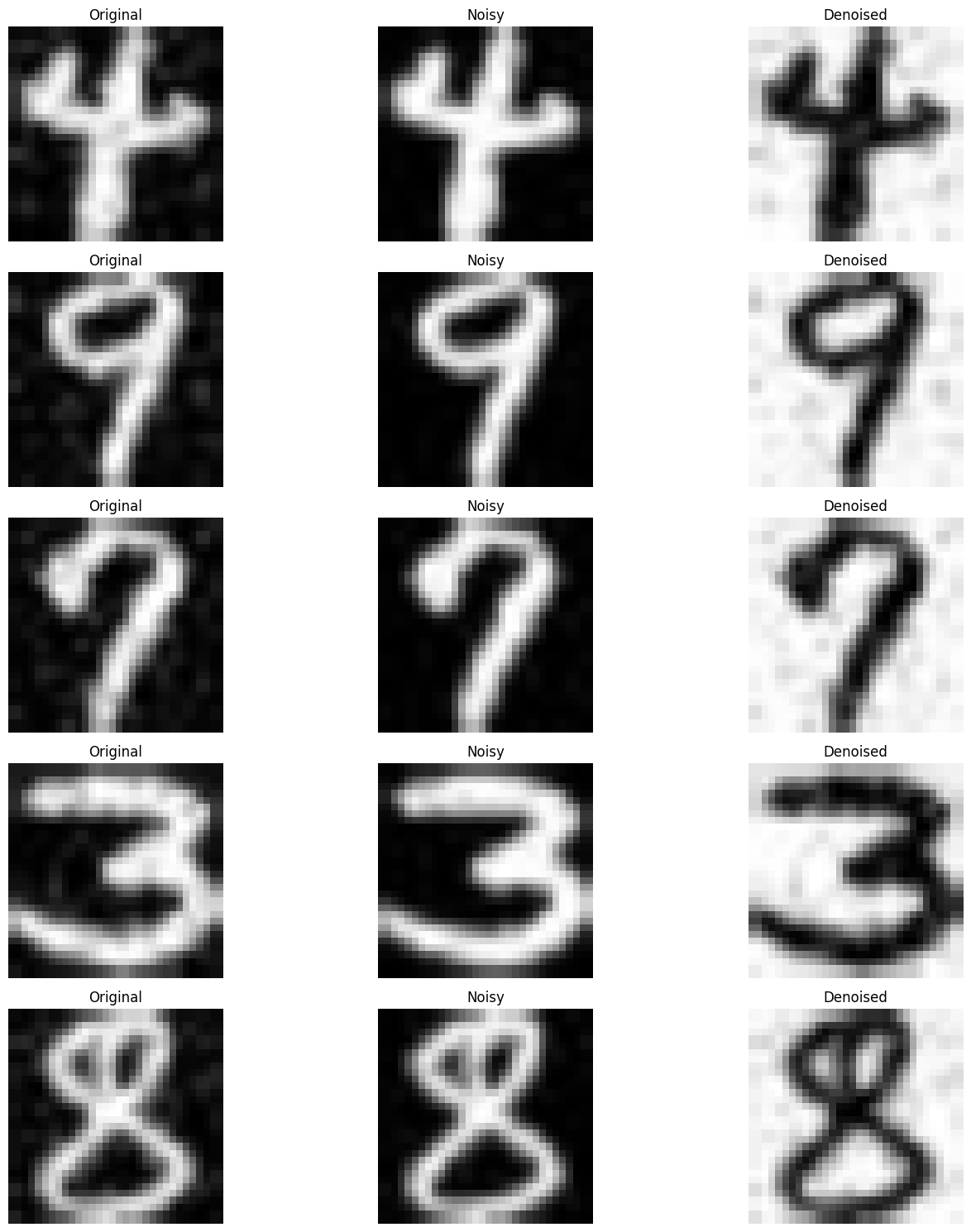
داده‌هاي train و test را جدا كرده و ابعاد داده‌هاي ورودي را مشخص مي‌كنيم.

****

* همانند بخش‌هاي قبلي پروژه، لايه‌هاي شبكه عصبي را مي‌سازيم.
* شبكه عصبي را كامپايل كرده و مدل را train مي‌كنيم.

****

* تصاویر denoise شده را با استفاده از تابع predict در denoised\_images ذخیره می کنیم.
* این تصاویر را از بازه -1 تا 1 به 0 تا 1، rescale می کنیم تا به بازه grayscale بازگردند.
* 5 مورد از داده های تست را به صورت تصادفی انتخاب می کنیم و آنها را نمایش می دهیم.

****