Below is a line-by-line explanation of the app.py code for diabetic retinopathy detection using Streamlit:

**Import Necessary Libraries**

*Code :*

*import streamlit as st*

*import tensorflow as tf*

*import numpy as np*

*from tensorflow.keras.models import load\_model*

* **import streamlit as st**: Streamlit is a Python framework for creating web apps. Here, it's used to build the app interface for uploading images and displaying predictions.
* **import tensorflow as tf**: TensorFlow is a deep learning framework used to load and run the pre-trained model for diabetic retinopathy detection.
* **import numpy as np**: NumPy is a library for numerical operations and is used to process the model predictions.
* **from tensorflow.keras.models import load\_model**: This function loads a pre-trained TensorFlow model saved earlier (e.g., during training).

**Set the Page Title and Description**

*Code :*

*st.title("Diabetic Retinopathy Detection")*

*st.write("Upload a retinal image to predict the stage of Diabetic Retinopathy.")*

* **st.title()**: Displays the main title at the top of the app, making it clear to users what the app is about.
* **st.write()**: Adds a description or additional information for the user, explaining the purpose of the app.

**Load the Pre-trained Model**

*model = load\_model('DR\_Detection\_Model.keras')*

*class\_names = ['Healthy', 'Mild DR', 'Moderate DR', 'Proliferative DR', 'Severe DR']*

* **load\_model('DR\_Detection\_Model.keras')**: Loads the saved deep learning model that was trained to classify retinal images into one of five categories of diabetic retinopathy (Healthy, Mild DR, etc.).
* **class\_names**: A list defining the five stages of diabetic retinopathy. The model's output is a prediction of one of these categories.

**Define Image Dimensions for Input**

*img\_height, img\_width = 180, 180 # Change these values if trained on a different size*

* The model expects input images to be of a specific size (180x180 pixels). These dimensions match the size of images used during training. Images uploaded to the app are resized to these dimensions to ensure compatibility.

**Create File Uploader**

*uploaded\_file = st.file\_uploader("Choose an image file", type=["jpg", "jpeg", "png"])*

* **st.file\_uploader()**: A Streamlit widget that allows the user to upload an image file.
* **type=["jpg", "jpeg", "png"]**: Ensures the uploaded file is an image in one of these formats.

**Check if a File Has Been Uploaded**

*if uploaded\_file is not None:*

* This condition checks whether the user has uploaded an image file. If no file is uploaded, the code inside this block is skipped.

**Process the Uploaded Image**

*img = tf.keras.utils.load\_img(uploaded\_file, target\_size=(img\_height, img\_width))*

*img\_array = tf.keras.utils.img\_to\_array(img)*

*img\_array = tf.expand\_dims(img\_array, 0) # Create batch*

* **load\_img()**: Loads the uploaded image from the file into a format TensorFlow can process, resizing it to 180x180 pixels (target\_size).
* **img\_to\_array()**: Converts the image into a numerical array (a grid of pixel values) that the model can process.
* **expand\_dims(img\_array, 0)**: Adds a "batch dimension" to the array. Models expect input as batches, even for a single image, so this step transforms the shape from (180, 180, 3) to (1, 180, 180, 3).

**Display the Uploaded Image**

*st.image(img, caption="Uploaded Image", width=300, use\_container\_width=False)*

* **st.image()**: Displays the uploaded image in the app so the user can see what they uploaded.
* **caption="Uploaded Image"**: Adds a label below the image.
* **width=300**: Resizes the displayed image to a smaller size for better readability in the app.
* **use\_container\_width=False**: Ensures the image doesn't stretch to fill the page width.

**Predict Using the Model**

*predictions = model.predict(img\_array)*

*score = tf.nn.softmax(predictions[0])*

* **model.predict(img\_array)**: Passes the processed image through the model. The output is a list of probabilities for each class (Healthy, Mild DR, etc.).
* **tf.nn.softmax()**: Converts the raw model output (logits) into probabilities. It ensures that the probabilities for all classes add up to 1, making it easier to interpret.

**Display Prediction Results**

*st.write(f"The image likely belongs to: \*\*{class\_names[np.argmax(score)]}\*\*")*

*st.write(f"Confidence: \*\*{100 \* np.max(score):.2f}%\*\*")*

* **np.argmax(score)**: Finds the index of the class with the highest probability. This index corresponds to the predicted class in class\_names.
* **np.max(score)**: Retrieves the highest probability (confidence) associated with the predicted class.
* **st.write()**: Displays the predicted class and its confidence percentage in the app.

**If No File is Uploaded**

*else:*

*st.write("Please upload an image.")*

* If the user hasn't uploaded an image, this block provides a message prompting them to upload one.

**Summary:**

1. The app allows users to upload an image of the retina.
2. The image is resized and prepared for input to the model.
3. The pre-trained model predicts the stage of diabetic retinopathy.
4. Results, including the predicted stage and confidence, are displayed in a user-friendly interface.