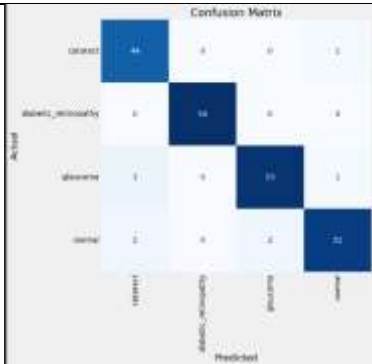


## Project Development Phase Model Performance Test

Date	19 November 2023
Team ID	Team-591850
Project Name	Eye Disease prediction
Maximum Marks	10 Marks

### Model Performance Testing:

S.No	Parameter	Values	Screenshot																									
1.	Metrics	<p><b>Regression Model:</b> MAE - , MSE - , RMSE - , R2 score -</p> <p><b>Classification Model:</b> Confusion Matrix - , Accuray Score- &amp; Classification Report -</p>	<div><table><thead><tr><th></th><th>Actual cataract</th><th>Actual diabetic_retinopathy</th><th>Actual glaucoma</th><th>Actual normal</th></tr></thead><tbody><tr><th>Predicted cataract</th><td>45</td><td>0</td><td>0</td><td>0</td></tr><tr><th>Predicted diabetic_retinopathy</th><td>0</td><td>54</td><td>0</td><td>0</td></tr><tr><th>Predicted glaucoma</th><td>0</td><td>0</td><td>57</td><td>0</td></tr><tr><th>Predicted normal</th><td>0</td><td>0</td><td>0</td><td>55</td></tr></tbody></table></div> <div>accuracy 0.957345971563981</div> <div><pre>Classification Report: -----                precision    recall  f1-score   support   cataract         0.90      0.98      0.94         45 diabetic_retinopathy  1.00      1.00      1.00         54  glaucoma         0.96      0.93      0.95         57  normal           0.96      0.93      0.94         55   accuracy         0.96  macro avg        0.96      0.96      0.96        211  weighted avg     0.96      0.96      0.96        211</pre></div>		Actual cataract	Actual diabetic_retinopathy	Actual glaucoma	Actual normal	Predicted cataract	45	0	0	0	Predicted diabetic_retinopathy	0	54	0	0	Predicted glaucoma	0	0	57	0	Predicted normal	0	0	0	55
	Actual cataract	Actual diabetic_retinopathy	Actual glaucoma	Actual normal																								
Predicted cataract	45	0	0	0																								
Predicted diabetic_retinopathy	0	54	0	0																								
Predicted glaucoma	0	0	57	0																								
Predicted normal	0	0	0	55																								



			<div><pre>def fit(self, X_train, y_train, X_test, y_test):     """Fit the model to the training data and evaluate on the test data.      Parameters     ----------     X_train : array-like of shape (n_samples, n_features)         Training data.     y_train : array-like of shape (n_samples,)         Training labels.     X_test : array-like of shape (n_samples, n_features)         Test data.     y_test : array-like of shape (n_samples,)         Test labels.      Returns     -------     dict         Dictionary with keys 'train_loss', 'train_acc', 'test_loss', 'test_acc'.     """     # Fit the model to the training data     self.fit(X_train, y_train)      # Evaluate on the training data     train_loss, train_acc = self.evaluate(X_train, y_train)      # Evaluate on the test data     test_loss, test_acc = self.evaluate(X_test, y_test)      return {'train_loss': train_loss, 'train_acc': train_acc, 'test_loss': test_loss, 'test_acc': test_acc}</pre></div> <div><pre>def fit(self, X_train, y_train, X_test, y_test):     """Fit the model to the training data and evaluate on the test data.      Parameters     ----------     X_train : array-like of shape (n_samples, n_features)         Training data.     y_train : array-like of shape (n_samples,)         Training labels.     X_test : array-like of shape (n_samples, n_features)         Test data.     y_test : array-like of shape (n_samples,)         Test labels.      Returns     -------     dict         Dictionary with keys 'train_loss', 'train_acc', 'test_loss', 'test_acc'.     """     # Fit the model to the training data     self.fit(X_train, y_train)      # Evaluate on the training data     train_loss, train_acc = self.evaluate(X_train, y_train)      # Evaluate on the test data     test_loss, test_acc = self.evaluate(X_test, y_test)      return {'train_loss': train_loss, 'train_acc': train_acc, 'test_loss': test_loss, 'test_acc': test_acc}</pre></div> <div><h3>Hold-out validation</h3><pre>def hold_out_validation(X, y, n_splits=5, random_state=None):     """Perform hold-out validation.      Parameters     ----------     X : array-like of shape (n_samples, n_features)         Training data.     y : array-like of shape (n_samples,)         Training labels.     n_splits : int         Number of splits.     random_state : int or None         Random state.      Returns     -------     dict         Dictionary with keys 'train_loss', 'train_acc', 'test_loss', 'test_acc'.     """     # Split the data into n_splits splits     splits = KFold(n_splits=n_splits, random_state=random_state)      # Evaluate on each split     train_loss, train_acc, test_loss, test_acc = [], [], [], []      for train_idx, test_idx in splits:         X_train, X_test = X[train_idx], X[test_idx]         y_train, y_test = y[train_idx], y[test_idx]          # Fit the model to the training data         model = LogisticRegression()         model.fit(X_train, y_train)          # Evaluate on the training data         train_loss, train_acc = model.evaluate(X_train, y_train)          # Evaluate on the test data         test_loss, test_acc = model.evaluate(X_test, y_test)          train_loss.append(train_loss)         train_acc.append(train_acc)         test_loss.append(test_loss)         test_acc.append(test_acc)      # Return the average values     return {'train_loss': np.mean(train_loss), 'train_acc': np.mean(train_acc), 'test_loss': np.mean(test_loss), 'test_acc': np.mean(test_acc)}</pre></div> <div><pre>def hold_out_validation(X, y, n_splits=5, random_state=None):     """Perform hold-out validation.      Parameters     ----------     X : array-like of shape (n_samples, n_features)         Training data.     y : array-like of shape (n_samples,)         Training labels.     n_splits : int         Number of splits.     random_state : int or None         Random state.      Returns     -------     dict         Dictionary with keys 'train_loss', 'train_acc', 'test_loss', 'test_acc'.     """     # Split the data into n_splits splits     splits = KFold(n_splits=n_splits, random_state=random_state)      # Evaluate on each split     train_loss, train_acc, test_loss, test_acc = [], [], [], []      for train_idx, test_idx in splits:         X_train, X_test = X[train_idx], X[test_idx]         y_train, y_test = y[train_idx], y[test_idx]          # Fit the model to the training data         model = LogisticRegression()         model.fit(X_train, y_train)          # Evaluate on the training data         train_loss, train_acc = model.evaluate(X_train, y_train)          # Evaluate on the test data         test_loss, test_acc = model.evaluate(X_test, y_test)          train_loss.append(train_loss)         train_acc.append(train_acc)         test_loss.append(test_loss)         test_acc.append(test_acc)      # Return the average values     return {'train_loss': np.mean(train_loss), 'train_acc': np.mean(train_acc), 'test_loss': np.mean(test_loss), 'test_acc': np.mean(test_acc)}</pre></div>
--	--	--	---