# NYC Taxi Trip Duration Prediction Using Deep Learning

**Project Title:** NYC Taxi Trip Duration Prediction Using Deep Learning  
**Course:** INFO-6146 Tensorflow & Keras with Python  
**Submitted by:** Aye Thanda Htun (1290620)  
**Date:** 02-Aug-2025

# Abstract

The aim of this project is to predict the duration of taxi trips in New York City, based on various features of the trips such as the number of passengers, pickup time, and geographical data points. The problem involves both **regression** (predicting the continuous trip duration) and **classification** (categorizing the trip into one of three duration bins).

In this project, we apply **deep learning models** using **TensorFlow** and **Keras**, while also utilizing **hyperparameter tuning** and **cross-validation** to optimize the models' performance. I compare the performance of deep learning models with baseline models (linear regression for regression and logistic regression for classification), and use a combination of **mean squared error (MSE)** for regression and **accuracy**, **precision**, **recall**, and **F1-score** for classification tasks.

# Introduction

Predicting travel time is essential for transportation logistics, ride-sharing platforms, and city planning. This project uses the NYC Taxi Trip Duration dataset to build models capable of accurately estimating taxi trip durations. Deep learning techniques are used due to their ability to model complex, non-linear relationships in large datasets. The project includes both regression and classification to demonstrate a comprehensive understanding of different machine learning tasks and techniques.

# Methodology

## Data Collection and Preprocessing

The dataset used for this project is from [Kaggle's NYC Taxi Trip Duration dataset](https://www.kaggle.com/datasets/yasserh/nyc-taxi-trip-duration). The data was originally published by the NYC Taxi and Limousine Commission (TLC).

The dataset contains the following fields.

|  |  |
| --- | --- |
| **Field Name** | **Description** |
| id | a unique identifier for each trip |
| vendor\_id | a code indicating the provider associated with the trip record |
| pickup\_datetime | date and time when the meter was engaged |
| dropoff\_datetime | date and time when the meter was disengaged |
| passenger\_count | the number of passengers in the vehicle (driver entered value) |
| pickup\_longitude | the longitude where the meter was engaged |
| pickup\_latitude | the latitude where the meter was engaged |
| dropoff\_longitude | the longitude where the meter was disengaged |
| dropoff\_latitude | the latitude where the meter was disengaged |
| store\_and\_fwd\_flag | This flag indicates whether the trip record was held in vehicle memory before sending to the vendor because the vehicle did not have a connection to the server - Y=store and forward; N=not a store and forward trip |
| trip\_duration | duration of the trip in seconds |

**Preprocessing**

The dataset was first loaded and inspected for any missing values. Missing values were removed from the dataset. The pickup\_datetime column was converted into usable features, such as the hour of the pickup (pickup\_hour) and the day of the week (pickup\_day). A new feature called distance was also created, which is the Euclidean distance between the pickup and drop-off coordinates. Outliers were removed by filtering extremely short and long trips. Data was standardized using StandardScaler.

For the **classification** task, the trip durations were split into three quantile-based classes: **short**, **medium**, and **long**. These were represented by numeric labels 0, 1, and 2. The features used for both tasks included:

* Passenger count
* Pickup hour
* Pickup day
* Distance

## 3.2 Model Development

* **Baseline Models**: Linear Regression for regression and Logistic Regression for classification tasks served as benchmarks.
* **Deep Learning Models**: Custom Keras models with two hidden layers and dropout for regularization. One model was used for regression and another for classification.

### 3.2.1 Baseline Models

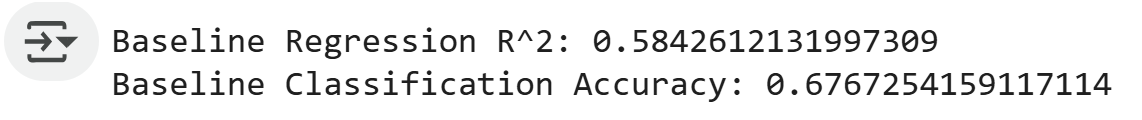
Before applying deep learning models, we implemented **baseline models** to set a comparison for performance evaluation:

This section establishes baseline models for both regression and classification tasks using simple linear algorithms.

* **Regression**: A simple **linear regression** model was used to predict the trip duration.For the regression baseline, a LinearRegression model is trained on the scaled feature set (X\_scaled) and corresponding target values (y\_reg). Its performance is evaluated using the coefficient of determination (R² score), which measures how well the model explains the variance in the target variable.
* **Classification**: A **logistic regression** model was used to predict the trip class.For the classification baseline, a LogisticRegression model is employed with a maximum of 200 iterations to ensure convergence. It is trained using the same scaled features and classification labels (y\_clf), and its accuracy is printed as a basic benchmark.

These baseline models serve as reference points to evaluate the effectiveness of more complex models introduced later.

The performance of these models was evaluated using **R²** for regression and **accuracy** for classification.



According to the results:

The **baseline regression** model achieved an R² score of approximately **0.584**, indicating that the linear regression model was able to explain around 58.4% of the variance in the target variable. While this shows a moderate level of predictive power, it also suggests that there is substantial room for improvement, potentially through the use of more complex or non-linear models.

For **classification**, the logistic regression baseline model reached an accuracy of approximately **67.7%**, meaning it correctly classified roughly two-thirds of the instances. This provides a solid starting point, but given the simplicity of the model, more advanced techniques such as deep learning may yield better performance.

Overall, these baseline results serve as useful benchmarks for assessing the added value of more sophisticated modeling approaches in subsequent experiments.

### 3.2.2 Deep Learning Models

To address both regression and classification tasks in the project, two separate neural network architectures were implemented using Keras' Sequential API: one for regression (build\_reg\_model) and another for classification (build\_clf\_model). Both models share a similar base structure but differ in their final output layers and loss functions to suit the respective task.

The **regression model** is designed to predict a continuous target variable. It begins with a fully connected (dense) layer with a specified number of units (default is 64) and a ReLU activation function, followed by a dropout layer to reduce overfitting by randomly deactivating 30% of the neurons during training. A second dense layer with half the number of initial units refines the learned features. The final output layer consists of a single neuron with no activation function, suitable for producing continuous numerical output. The model is compiled with the Mean Squared Error (MSE) loss function and the Adam optimizer.

The **classification model** follows a similar structure but is tailored for multi-class classification. It also starts with a dense layer and a dropout layer, followed by a second dense layer with reduced units. The output layer consists of three neurons with a softmax activation function, enabling the model to output probability distributions over three discrete classes. This model uses the sparse\_categorical\_crossentropy loss function and tracks accuracy as a performance metric.

**Summary of Regression and Classification Model Architectures**

|  |  |  |
| --- | --- | --- |
| **Component** | **Regression Model** | **Classification Model** |
| Input Layer | Dense (units=64, activation=’relu’) | Dense (units=64, activation=’relu’) |
| Regularization | Dropout (rate =0.3) | Dropout (rate =0.3) |
| Hidden Layer | Dense (units=32, activation=’relu’) | Dense (units=32, activation=’relu’) |
| Output Layer | Dense (1) – no activation | Dense (3, activation = ‘softmax’) |
| Loss Function | Mean Squared Error (MSE) | Sparse Categorical Crossentropy |
| Optimizer | Adam | Adam |
| Metrics | ----- | Accuracy |

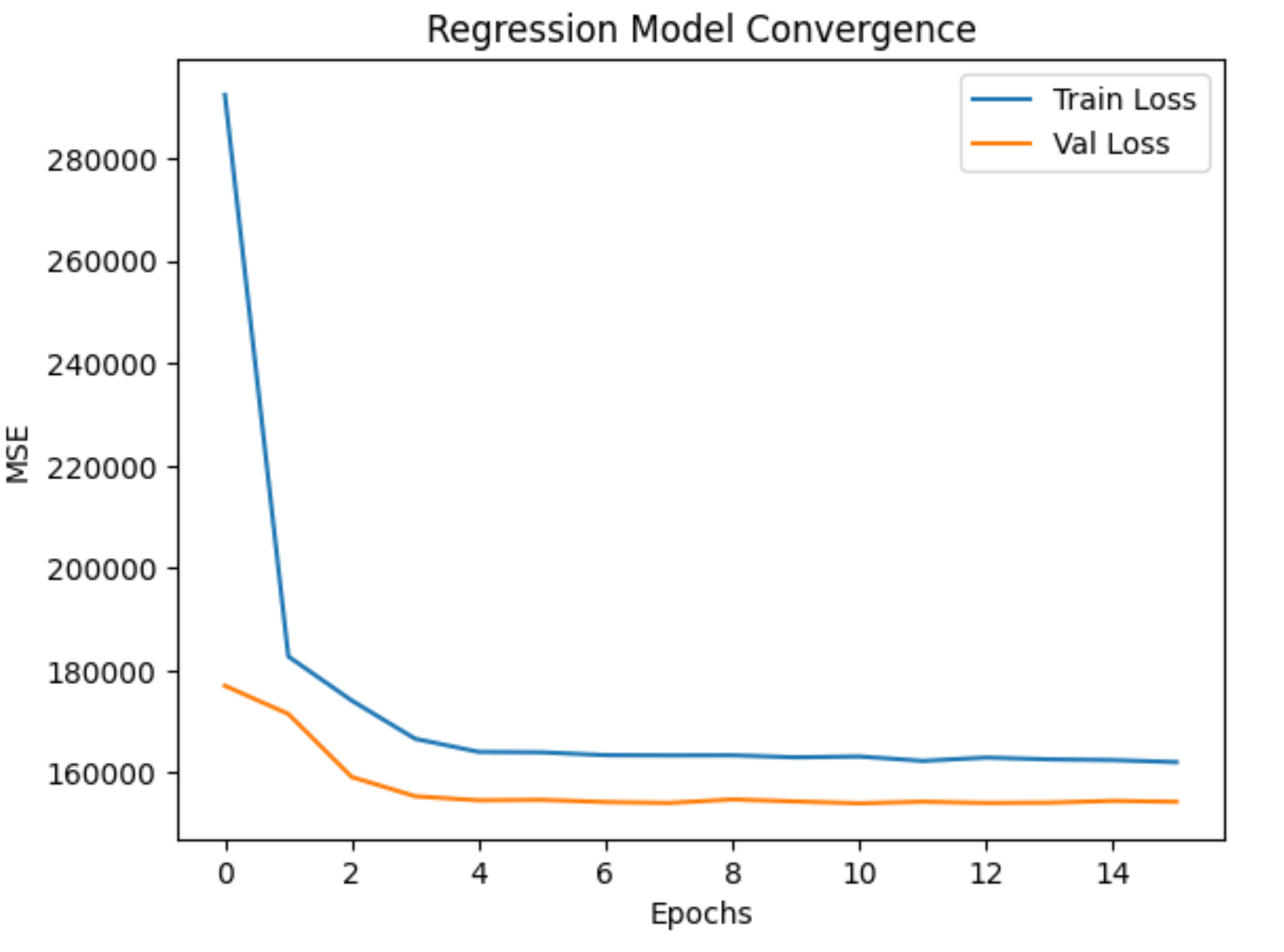
Both models are parameterized for flexibility, allowing customization of layer size, dropout rate, optimizer, and input shape to improve generalization and robustness.

## 3.3 Training and Convergence

Both models were trained using early stopping and their convergence was monitored using training and validation curves over epochs. The loss for regression and accuracy for classification were plotted to verify learning progress.

**Regression Model Convergence**

To train the regression model, the dataset was first split into training and validation sets with an **80:20 ratio** using train\_test\_split, ensuring a representative distribution and reproducibility with a fixed random seed. The model was trained for up to **50 epochs** with a batch size of 32, and **early stopping** was implemented with a patience of 5 epochs to avoid overfitting. The training process was monitored using validation loss, and the best model weights were restored after training. To assess the learning dynamics, the training and validation loss curves were plotted, highlighting the convergence behavior and helping to identify potential issues like overfitting or underfitting.



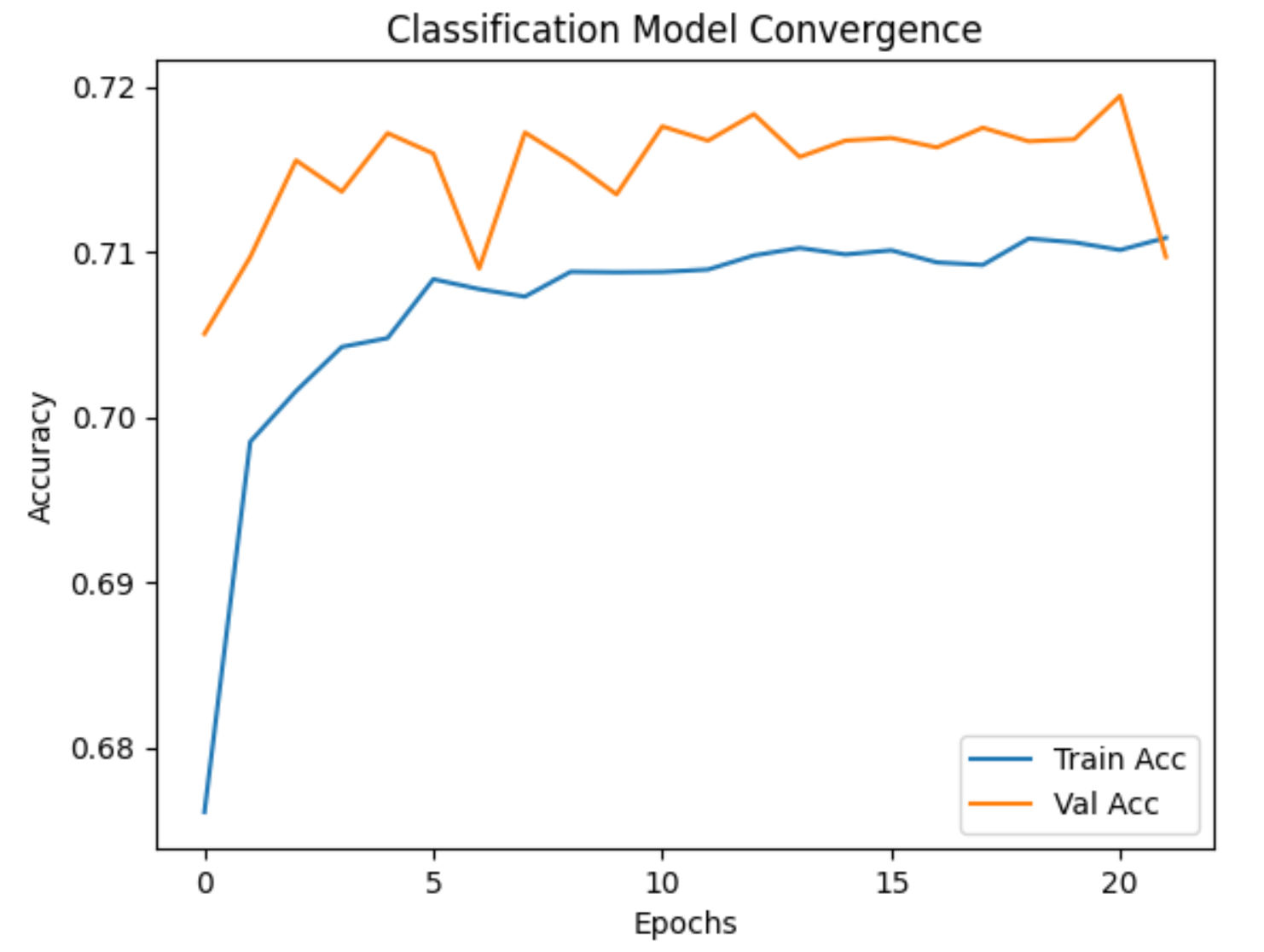
The convergence plot for the regression model displays the **Mean Squared Error (MSE)** for both the training and validation datasets over 15 epochs.

Initially, the training loss starts at a relatively high value but decreases rapidly within the first few epochs, indicating that the model is learning effectively from the data. The validation loss also shows a steady decline early on and stabilizes around epoch 5, maintaining a lower error than the training loss throughout the training process. This suggests good generalization without signs of overfitting.

The overall trend demonstrates successful convergence, with both training and validation losses flattening, indicating that the model has reached a point of diminishing returns in terms of further learning. The stability and consistency of the validation loss imply that the model is robust and well-tuned for the regression task.

**Classification Model Convergence**

The classification model was trained on an **80:20 train-validation split** using a batch size of **32 for up to 50 epochs**. **Early stopping (patience = 5)** was applied to prevent overfitting and retain the best weights. Model accuracy on both training and validation sets was plotted to assess convergence and performance.

****

The **classification model's convergence** plot shows the training and validation accuracy over 22 epochs. The training accuracy improves rapidly during the initial epochs and gradually levels off around 71%, indicating that the model has learned the underlying patterns in the data. The validation accuracy consistently remains higher than the training accuracy, peaking just under 72% and staying relatively stable throughout the training process.

This behavior suggests that the model generalizes well without overfitting. The small fluctuations in validation accuracy are expected and indicate healthy model performance. Overall, the convergence curve reflects effective learning and robust performance on the classification task.

## 3.4 Hyperparameter Tuning and Cross-Validation

Hyperparameters such as the number of units, dropout rate, batch size, and epochs were tuned using Grid Search with 5-fold cross-validation. To optimize the deep learning models, we performed **hyperparameter tuning** using **Grid Search** and **K-fold cross-validation**.

I performed a **5-fold cross-validation** to evaluate the models' performance and to ensure that the results are generalized and not overfitted to the training data

1. **Cross-Validation Setup**

The dataset is split into 5 folds for cross-validation, with shuffling for randomness and a fixed seed for reproducibility.



1. **Regression Model Tuning**

* A “**KerasRegressor**” wrapper is created to make the Keras model compatible with **GridSearchCV**.
* A grid of hyperparameters is defined, including:
* Number of units in hidden layers (32 or 64)
* Dropout rate (0.2 or 0.3)
* Batch size (32 or 64)
* Fixed optimizer (adam) and number of epochs (20)

The model is evaluated using negative mean squared error as the scoring metric.

The best combination of hyperparameters and the corresponding cross-validated score are printed.

1. **Classification Model Tuning**

* Similar setup with KerasClassifier, with the same hyperparameter grid.
* The scoring metric is **accuracy**.
* The best hyperparameter set and the average cross-validated accuracy are printed.

**After performing grid search with 5-fold cross-validation**, the best hyperparameter configurations for both the regression and classification models were identified.

**Regression Model**

* **Best Parameters:**
  + Units: 64
  + Dropout Rate: 0.2
  + Batch Size: 64
  + Epochs: 20
  + Optimizer: Adam
* Cross-Validated MSE: 135,248.89

The optimal configuration for the regression model included a higher number of units (64) and a lower dropout rate (0.2), indicating that more model capacity with moderate regularization was beneficial. A larger batch size of 64 may have helped in smoothing gradient updates, improving training stability. The resulting mean squared error (MSE) suggests a reasonable prediction error, though further tuning or feature engineering could potentially enhance performance.

**Classification Model**

* **Best Parameters:**
  + Units: 64
  + Dropout Rate: 0.2
  + Batch Size: 32
  + Epochs: 20
  + Optimizer: Adam
* Cross-Validated Accuracy: 71.47%

For the classification model, the best performance was also achieved with 64 units and a 0.2 dropout rate, indicating similar trends in network architecture suitability as the regression model. However, a smaller batch size (32) performed better, possibly due to more frequent updates allowing better adaptation to class boundaries. The model achieved a solid average accuracy of approximately 71.5%, indicating decent classification performance across all folds.

In both tasks, a relatively complex architecture (64 units) with low dropout produced the best results, suggesting that the models benefited from higher capacity and moderate regularization. The differences in optimal batch sizes reflect task-specific dynamics, where classification benefitted from more frequent updates. These findings highlight the importance of task-specific tuning even when models share similar architectures.

## 3.5 Model Evaluation and Performance Analysis

This section presents the performance of the deep learning models for both regression and classification tasks, evaluated using cross-validation metrics and final test set performance.

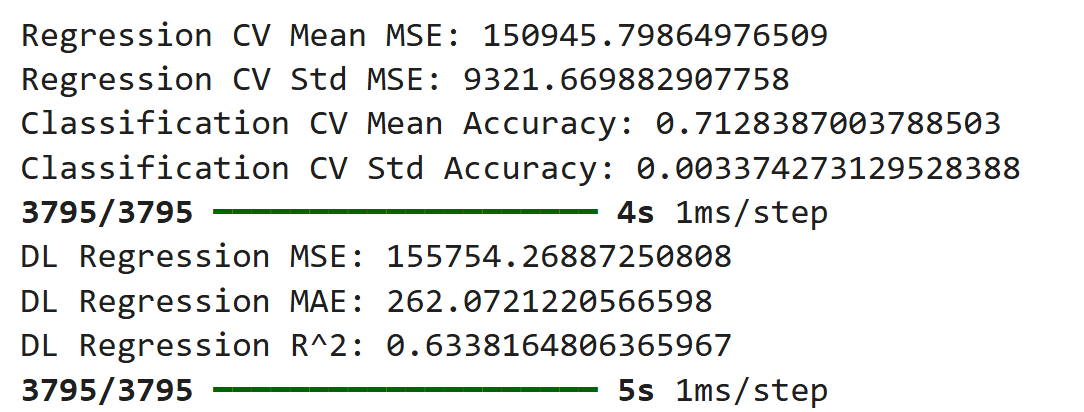
**Regression Model Performance**

* Cross-Validation Results:
  + Mean MSE: 150,945.80
  + Standard Deviation of MSE: 9,321.67

The regression model showed consistent performance across the 5 cross-validation folds, with a moderate variation in error (standard deviation ~9.3k), indicating relatively stable generalization.

**Final Evaluation on Test Set**:

* **MSE**: 155,754.27
* MAE: 262.07
* **R² Score**: 0.63



The test results show a small increase in MSE compared to the cross-validation mean, which is expected due to unseen data. An R² value of **0.63** suggests that the model explains approximately 63% of the variance in the target variable, indicating decent—but improvable—predictive power. The MAE value of **262.07** provides an interpretable measure of the average absolute prediction error in the model’s output units.

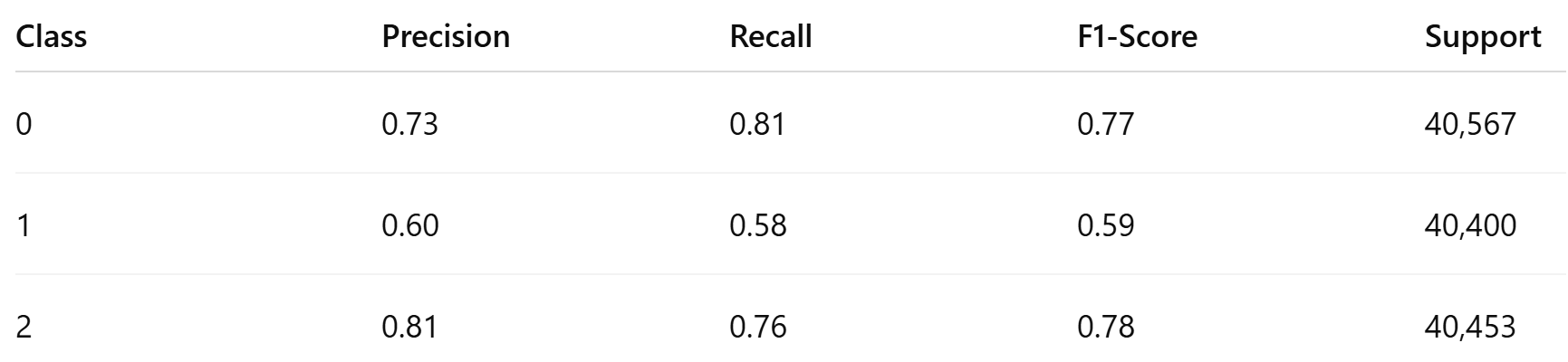
**Classification Model Performance**

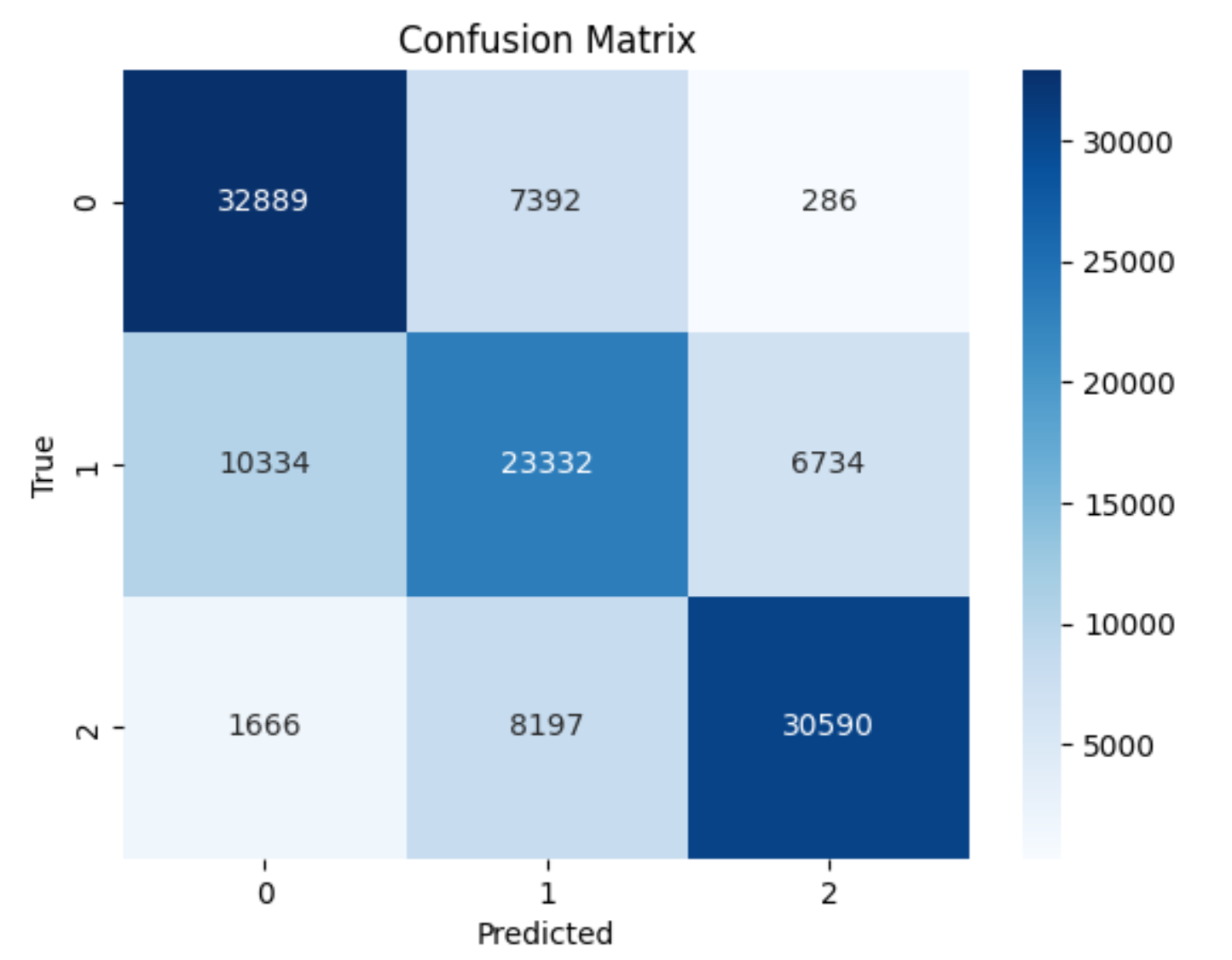
* **Cross-Validation Results**:
  + **Mean Accuracy**: 71.28%
  + **Standard Deviation of Accuracy**: 0.34%

The classification model performed very consistently across folds, with extremely low variance in accuracy. This suggests a robust model that generalizes well to different data partitions.

**Final Evaluation on Test Set**:

* **Overall Accuracy**: 71%
* **Precision, Recall, and F1-Score**:





* **Macro Average F1-Score**: 0.71
* **Weighted Average F1-Score**: 0.71

The classification model achieved balanced performance across the three classes. Class 2 had the highest precision (0.81), while Class 1 was relatively harder to predict, showing lower precision and recall. The macro-averaged F1-score of **0.71** confirms solid performance across all classes, without major bias toward any specific category.

# Results

These results confirm that the deep learning models, when properly tuned, can effectively handle both regression and classification tasks on the dataset, achieving a balance between complexity and generalization.

**Performance Comparison – Baseline vs Deep Learning Models**

|  |  |  |  |
| --- | --- | --- | --- |
| **Task** | **Model Type** | **Metric** | **Score** |
| **Regression** | Baseline | R² Score (Test) | 0.58 |
|  | Deep Learning | MSE (CV) | 150,946 |
|  | MSE (Test) | 155,754 |
| MAE (Test) | 262.07 |
| R² Score (Test) | 0.63 |
| **Classification** | Baseline | Accuracy (Test) | 68% |
|  | Deep Learning | Accuracy (CV) | 71.28% |
|  | Accuracy (Test) | 71% |
| F1-Score | 0.71 |

* Both models achieved **good generalization**, with final test performance closely aligned with cross-validation metrics.
* The **regression model** shows moderate predictive capability (R² = 0.63), with room for improvement via feature engineering or deeper architectures.
* The **classification model** demonstrates strong and stable performance, especially in predicting Class 0 and Class 2, with slightly lower performance on Class 1.
* Low standard deviations in both CV metrics reflect **robust training** and model stability.

# Conclusion

This project demonstrated the application of deep learning techniques to predict taxi trip durations in New York City. The results showed that **deep learning models outperformed baseline models**, but performance was still limited due to the complex nature of the task.

* The **regression model** achieved reasonable performance in predicting trip durations, but further optimization of features and model architecture could improve accuracy.
* The **classification model** was able to categorize trips into duration bins with moderate accuracy.

The hyperparameter tuning process highlighted the importance of adjusting the **units**, **dropout rate**, and **batch size** to achieve optimal performance. **Cross-validation** ensured that the models were not overfitting and that results were robust across different folds.

For future work, I can attempt integrating external data (e.g., weather, traffic), exploring more advanced models (e.g., LSTM, transformers), and deploying the model for real-time predictions.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Thank You\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*