<u>MACHINE LEARNING PROJECT REPORT</u>

FOOD DELIVERY TIME PREDICTION

Bachelor of Science in Computer Science (Hons.)

Submitted by:

Lovleen Kaur Ahuja - 2020CSC1005
Saloni Kumari - 2020CSC1044

Date of Submission: 15-04-2023

Under the guidance of
Mrs. Shefali Gupta

Department Of Computer Science

SEMESTER VI - 2023

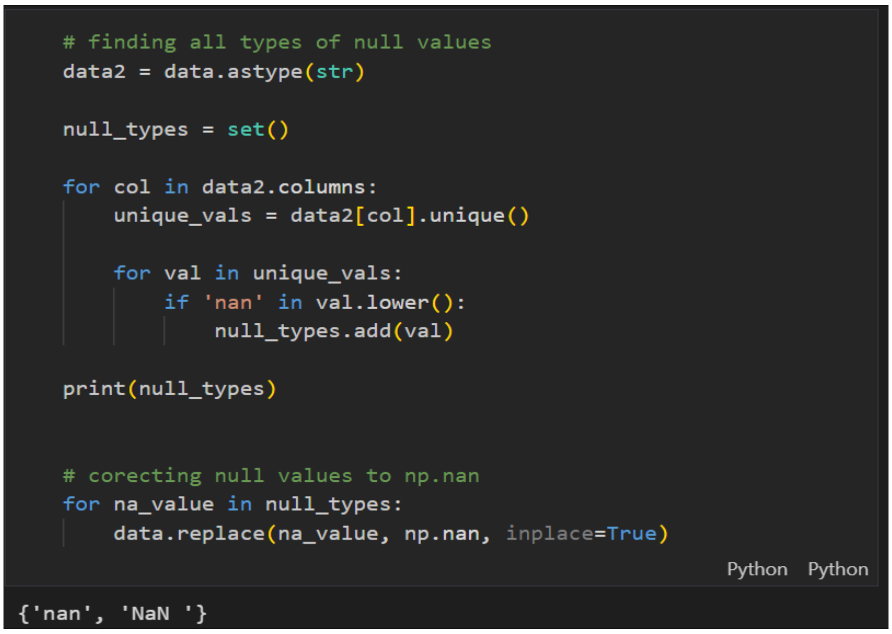
1) Project Statement:

The aim of this project is to develop a food delivery time prediction model using concepts of Recursive Neural Network (RNN) and Long-Short-Term Memory network (LSTM). The model will take into account various factors such as distance, traffic, weather conditions etc. While the initial focus is on predicting time delivery, the model can be extended to other industries that require time-sensitive delivery services.

2) Working of the project:

1. **Dataset:** The dataset consists of 56993 rows and 20 features of both categorical and non- categorical type of data. The dataset gives us some important information about the traffic, weather, vehicle, type of food and data about the delivery person which makes the predication more suitable for real life scenarios.
   The original dataset came in 3 files – sample.csv, test.csv, train.csv. All of it is merged in the following project.
   Source: https://www.kaggle.com/datasets/gauravmalik26/food-delivery-dataset

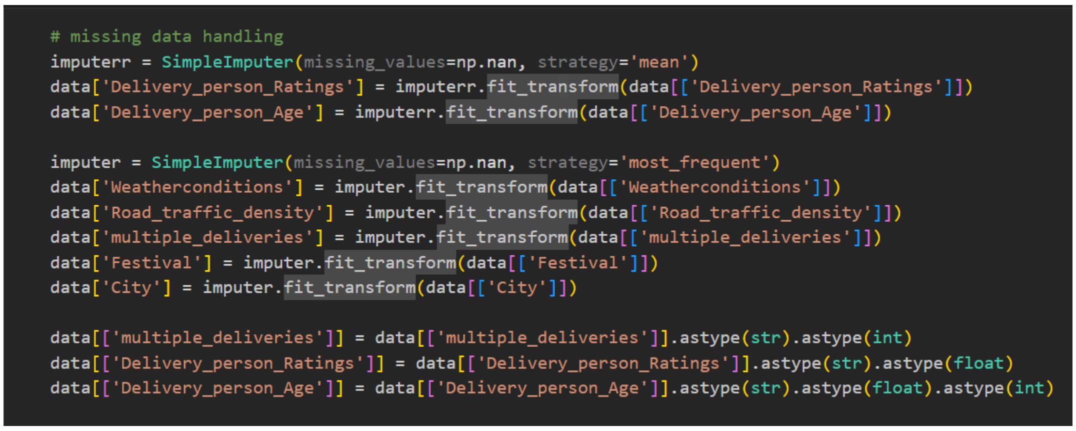
2. **Data Preprocessing:** The dataset is then preprocessed to find all types of null values and then replacing them all by a unique one to replace it in the following processing.



```python
# finding all types of null values
data2 = data.astype(str)

null_types = set()

for col in data2.columns:
    unique_vals = data2[col].unique()

    for val in unique_vals:
        if 'nan' in val.lower():
            null_types.add(val)

print(null_types)


# corecting null values to np.nan
for na_value in null_types:
    data.replace(na_value, np.nan, inplace=True)
```

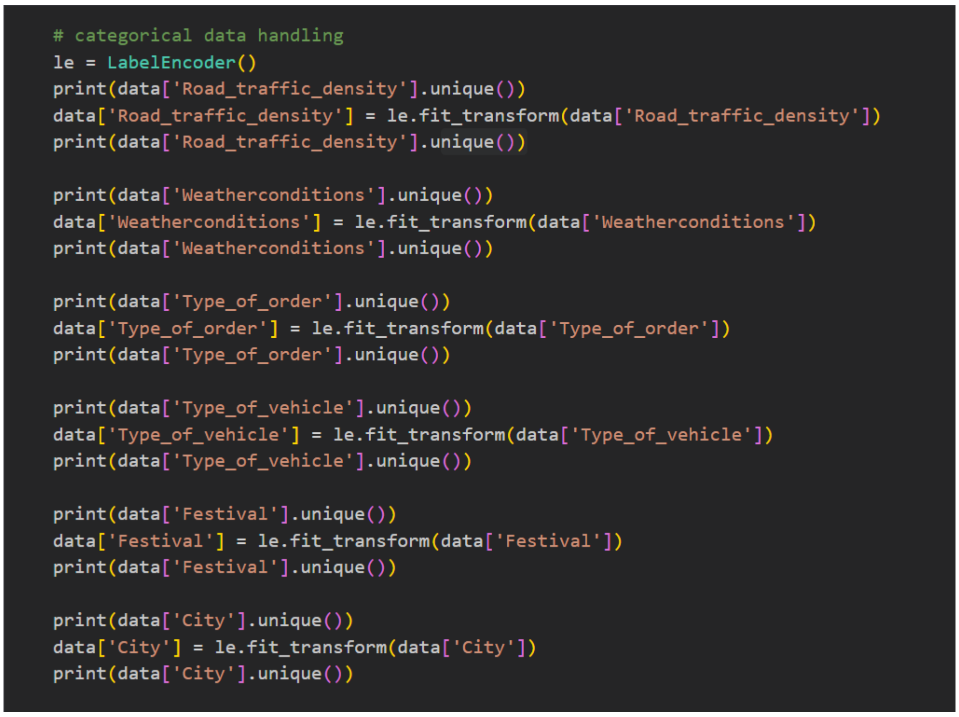
```
{'nan', 'NaN '}
```

   The missing values are then handled by putting mean in the numerical type feature and mode in the categorical type feature.



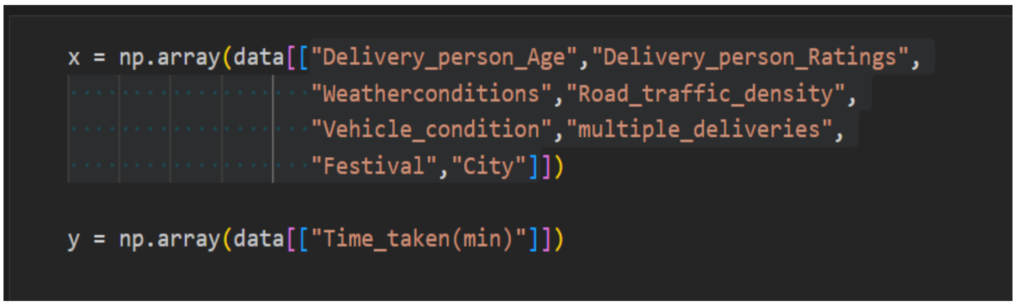
```
# missing data handling
imputerr = SimpleImputer(missing_values=np.nan, strategy='mean')
data['Delivery_person_Ratings'] = imputerr.fit_transform(data[['Delivery_person_Ratings']])
data['Delivery_person_Age'] = imputerr.fit_transform(data[['Delivery_person_Age']])

imputer = SimpleImputer(missing_values=np.nan, strategy='most_frequent')
data['Weatherconditions'] = imputer.fit_transform(data[['Weatherconditions']])
data['Road_traffic_density'] = imputer.fit_transform(data[['Road_traffic_density']])
data['multiple_deliveries'] = imputer.fit_transform(data[['multiple_deliveries']])
data['Festival'] = imputer.fit_transform(data[['Festival']])
data['City'] = imputer.fit_transform(data[['City']])

data[['multiple_deliveries']] = data[['multiple_deliveries']].astype(str).astype(int)
data[['Delivery_person_Ratings']] = data[['Delivery_person_Ratings']].astype(str).astype(float)
data[['Delivery_person_Age']] = data[['Delivery_person_Age']].astype(str).astype(float).astype(int)
```

Categorical data is then transformed into numerical format using Label Encoder.



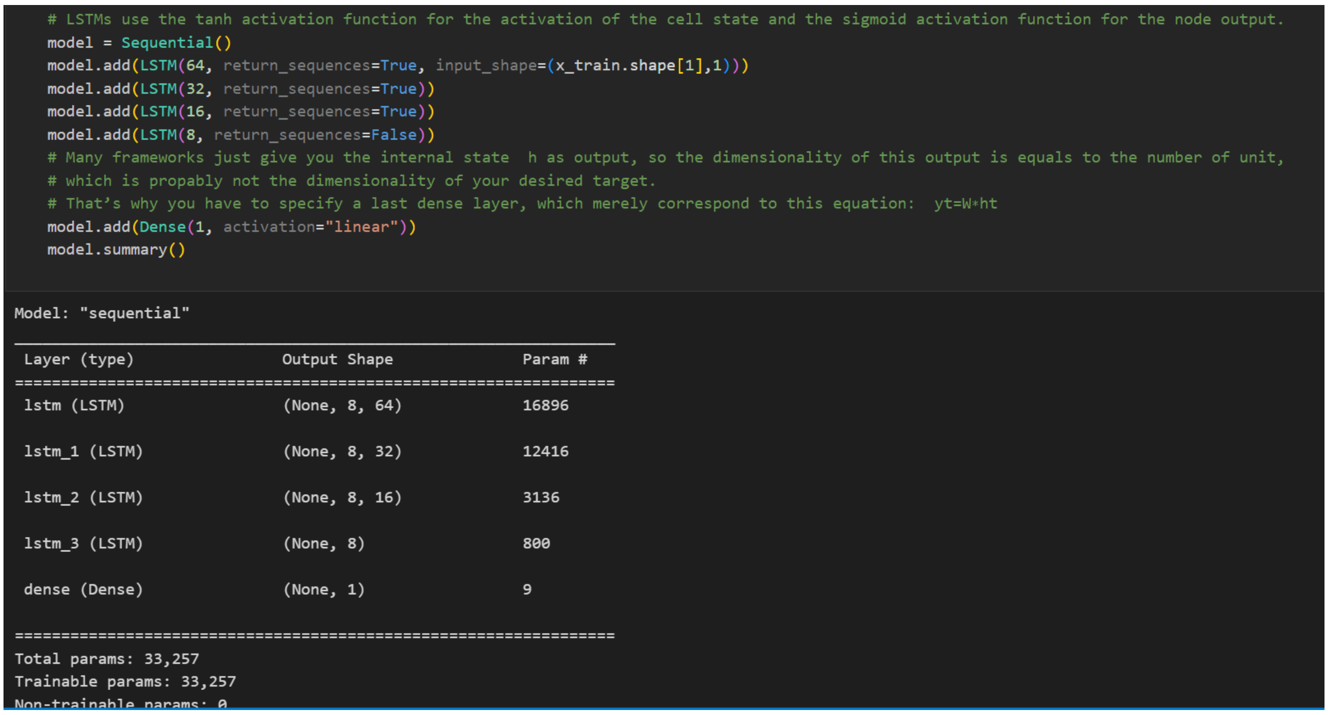
```
# categorical data handling
le = LabelEncoder()
print(data['Road_traffic_density'].unique())
data['Road_traffic_density'] = le.fit_transform(data['Road_traffic_density'])
print(data['Road_traffic_density'].unique())

print(data['Weatherconditions'].unique())
data['Weatherconditions'] = le.fit_transform(data['Weatherconditions'])
print(data['Weatherconditions'].unique())

print(data['Type_of_order'].unique())
data['Type_of_order'] = le.fit_transform(data['Type_of_order'])
print(data['Type_of_order'].unique())

print(data['Type_of_vehicle'].unique())
data['Type_of_vehicle'] = le.fit_transform(data['Type_of_vehicle'])
print(data['Type_of_vehicle'].unique())

print(data['Festival'].unique())
data['Festival'] = le.fit_transform(data['Festival'])
print(data['Festival'].unique())

print(data['City'].unique())
data['City'] = le.fit_transform(data['City'])
print(data['City'].unique())
```

Feature Selection done on the basis of heatmap drawn using seaborn library giving us the correlation between features.



```
x = np.array(data[["Delivery_person_Age","Delivery_person_Ratings",
                   "Weatherconditions","Road_traffic_density",
                   "Vehicle_condition","multiple_deliveries",
                   "Festival","City"]])

y = np.array(data[["Time_taken(min)"]])
```

3. **Model Architecture:** The recursive neural network model is then designed with an input layer,multiple hidden layers, and an output layer. The number of hidden layers and neurons in each layer is determined through experimentation and analysis.

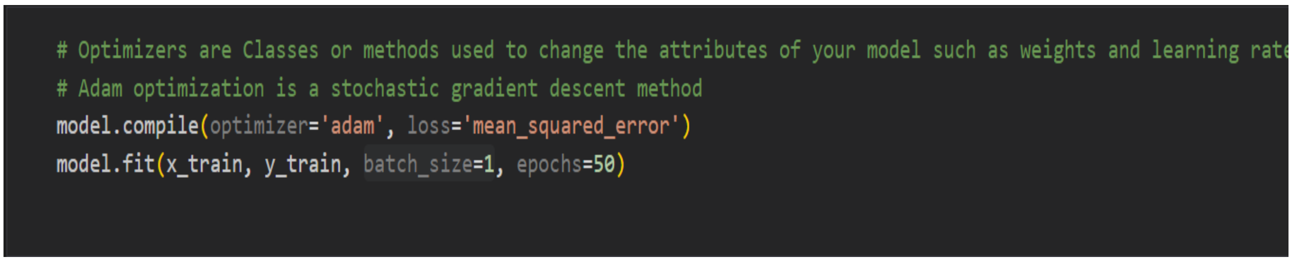


```
# LSTMs use the tanh activation function for the activation of the cell state and the sigmoid activation function for the node output.
model = Sequential()
model.add(LSTM(64, return_sequences=True, input_shape=(x_train.shape[1],1)))
model.add(LSTM(32, return_sequences=True))
model.add(LSTM(16, return_sequences=True))
model.add(LSTM(8, return_sequences=False))
# Many frameworks just give you the internal state  h as output, so the dimensionality of this output is equals to the number of unit,
# which is propably not the dimensionality of your desired target.
# That’s why you have to specify a last dense layer, which merely correspond to this equation:  yt=W*ht
model.add(Dense(1, activation="linear"))
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 8, 64)             16896

 lstm_1 (LSTM)               (None, 8, 32)             12416

 lstm_2 (LSTM)               (None, 8, 16)             3136

 lstm_3 (LSTM)               (None, 8)                 800

 dense (Dense)               (None, 1)                 9

=================================================================
Total params: 33,257
Trainable params: 33,257
Non-trainable params: 0
```

4. **Training:** The backpropagation algorithm with the Adam optimization algorithm is utilized to train the neural network on the training set. The objective of training is to minimize the loss function by adjusting the weights and biases of the neurons.

   Once the model is constructed, it needs to be compiled to enable training with the available dataset. The model is compiled using optimizers. The compilation of the model involves three essential parameters: optimizer, loss, and metrics.

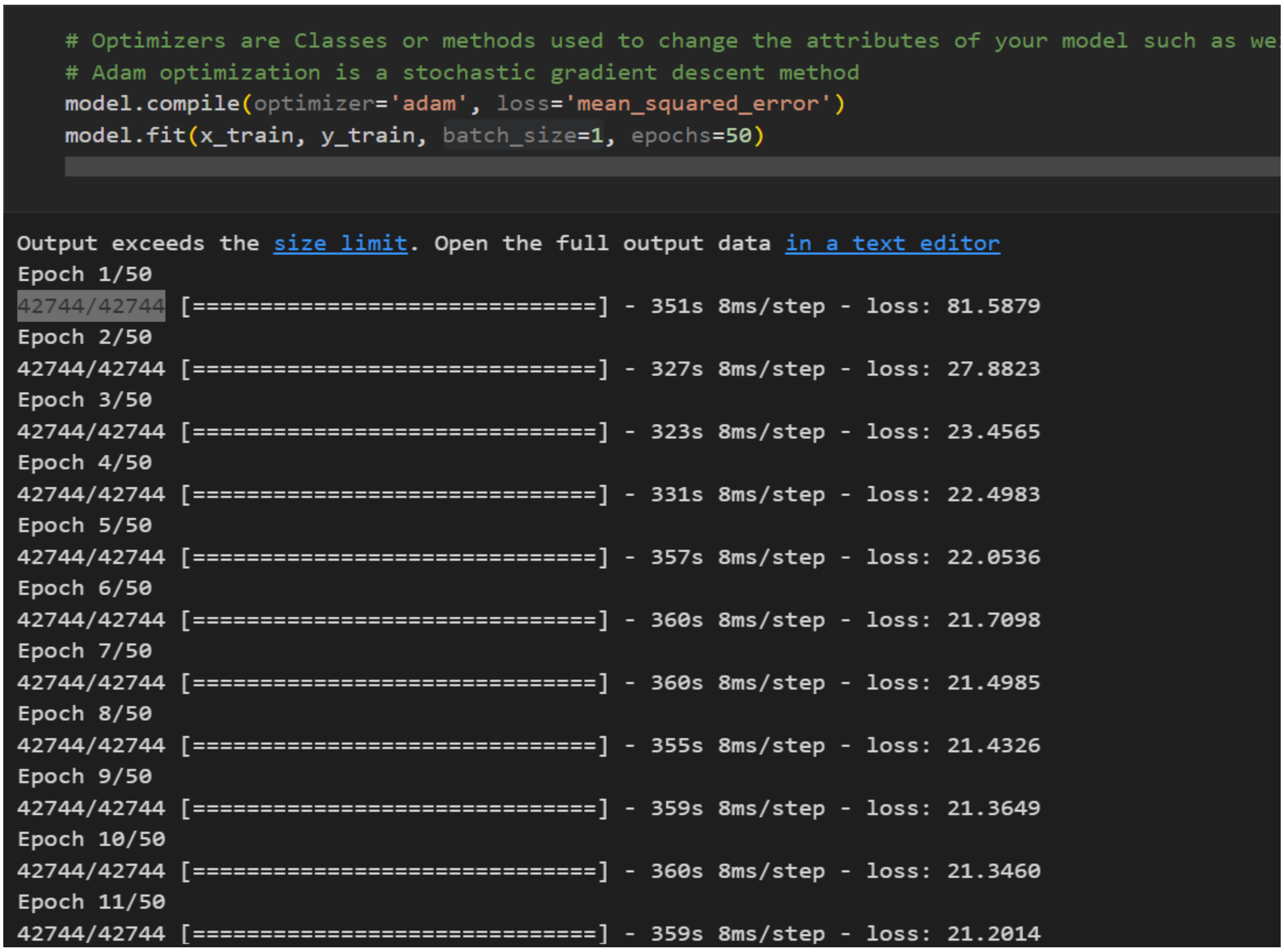


```
# Optimizers are Classes or methods used to change the attributes of your model such as weights and learning rate
# Adam optimization is a stochastic gradient descent method
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, batch_size=1, epochs=50)
```

- Optimizers are techniques or algorithms that are utilized to modify the attributes of the neural network, such as the weights and learning rate, in order to minimize losses. Their primary objective is to solve optimization problems by reducing the overall function. The optimizer has control over the learning rate, which determines how quickly the optimal weights for the model are computed. In this project, we have chosen to use the 'Adam' optimizer, which adjusts the learning rate throughout the training process. It is worth noting that a smaller learning rate may lead to more accurate weights, but it can also significantly increase the time it takes to calculate the weights. In order to achieve better learning rates and improved validation, it is recommended to increase the train and test data, respectively.

- The model is then trained over 50epochs(iterations), as the number of iterations increases, overfitting of the model was observed

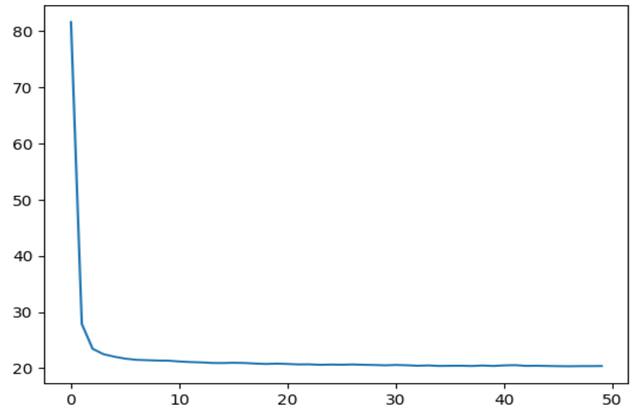
- Google Colab GPU took 5min/epoch and TPU took 12min/epoch



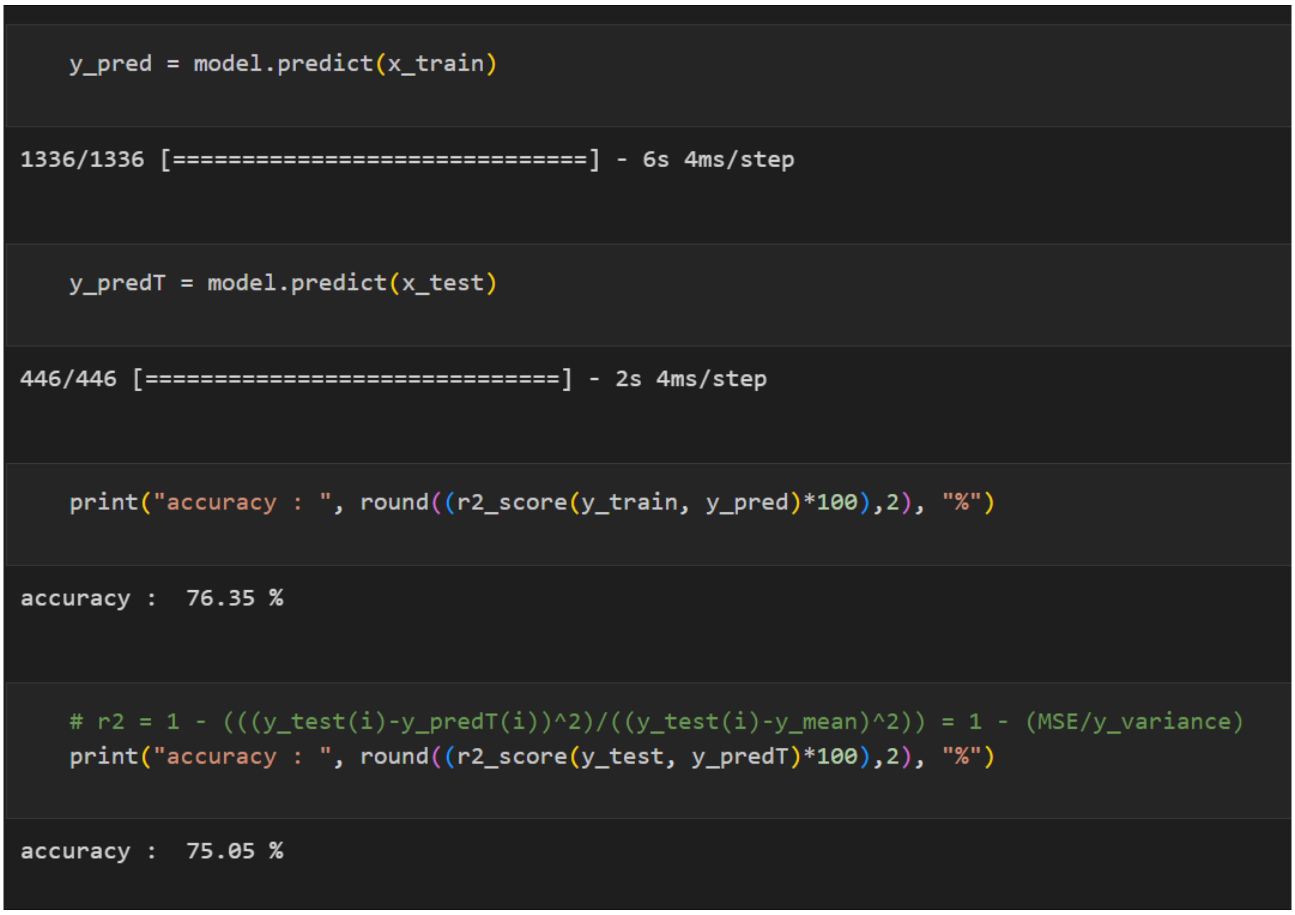
```
# Optimizers are Classes or methods used to change the attributes of your model such as we
# Adam optimization is a stochastic gradient descent method
model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(x_train, y_train, batch_size=1, epochs=50)
```

```
Output exceeds the size limit. Open the full output data in a text editor
Epoch 1/50
42744/42744 [==============================] - 351s 8ms/step - loss: 81.5879
Epoch 2/50
42744/42744 [==============================] - 327s 8ms/step - loss: 27.8823
Epoch 3/50
42744/42744 [==============================] - 323s 8ms/step - loss: 23.4565
Epoch 4/50
42744/42744 [==============================] - 331s 8ms/step - loss: 22.4983
Epoch 5/50
42744/42744 [==============================] - 357s 8ms/step - loss: 22.0536
Epoch 6/50
42744/42744 [==============================] - 360s 8ms/step - loss: 21.7098
Epoch 7/50
42744/42744 [==============================] - 360s 8ms/step - loss: 21.4985
Epoch 8/50
42744/42744 [==============================] - 355s 8ms/step - loss: 21.4326
Epoch 9/50
42744/42744 [==============================] - 359s 8ms/step - loss: 21.3649
Epoch 10/50
42744/42744 [==============================] - 360s 8ms/step - loss: 21.3460
Epoch 11/50
42744/42744 [==============================] - 359s 8ms/step - loss: 21.2014
```

Loss Per Epoch



5. **Testing**: After the model is trained, it is evaluated on the testing set to determine its accuracy and performance. The performance of the model is by accuracy calculated by r2_score.



```
    y_pred = model.predict(x_train)
```

```
1336/1336 [==============================] - 6s 4ms/step
```

```
    y_predT = model.predict(x_test)
```

```
446/446 [==============================] - 2s 4ms/step
```

```
    print("accuracy : ", round((r2_score(y_train, y_pred)*100),2), "%")
```

```
accuracy :  76.35 %
```

```
    # r2 = 1 - (((y_test(i)-y_predT(i))^2)/((y_test(i)-y_mean)^2)) = 1 - (MSE/y_variance)
    print("accuracy : ", round((r2_score(y_test, y_predT)*100),2), "%")
```

```
accuracy :  75.05 %
```

3) Algorithms:

Backpropagation Algorithm:

The algorithm is used to effectively train a neural network through a method called chain rule. In simple terms, after each forward pass through a network, backpropagation performs a backward pass while adjusting the model's parameters (weights and biases). Inputs X, arrive through the preconnected path

1. Input is modelled using real weights W. The weights are usually randomlyselected.
2. Calculate the output for every neuron from the input layer, to the hidden layers, to the output layer.
3. Calculate the error in the outputs.
4. Travel back from the output layer to the hidden layer to adjust the weights suchthat the error is decreased.

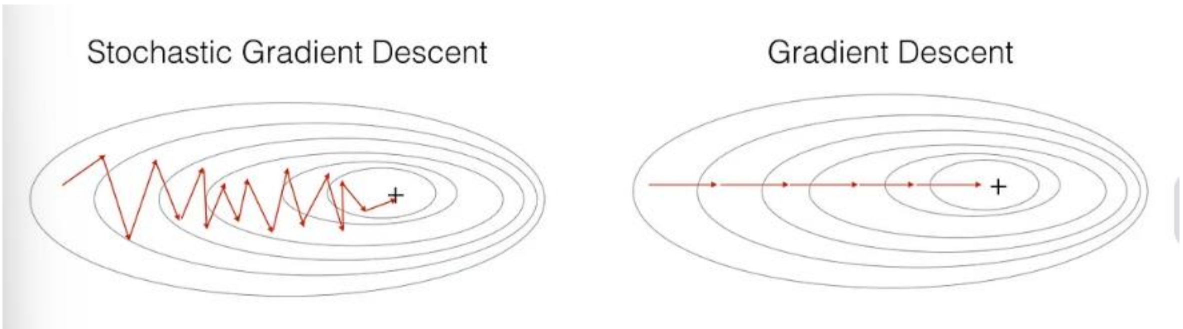
Keep repeating the process until the desired output is achieved

Stochastic Gradient Descent:

Stochastic Gradient Descent picks a few samples selected randomly instead of the whole data set for each descent iteration.

Using the whole dataset is a problem when the data is large, typical gradient descent will have to use all of the data for completing one iteration and it has to be done for every iteration until minima is reached. Therefore, it becomes computationally very expensive to perform the traditional complete batch descent.

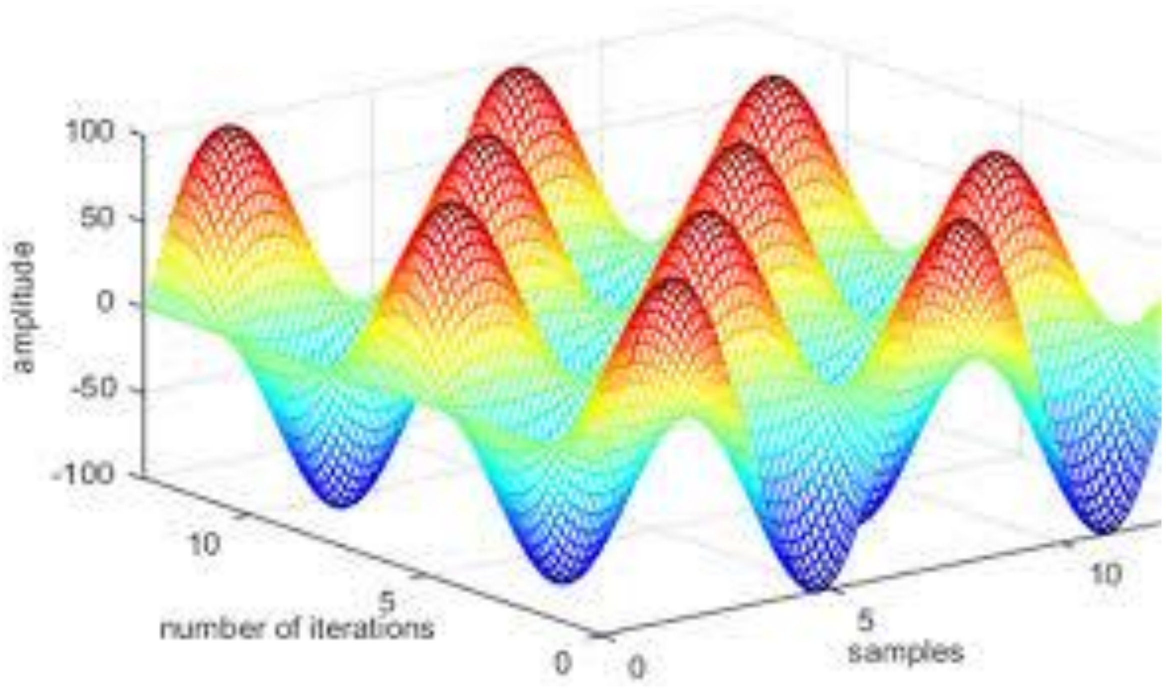
Due to the noisy updates, SGD may not converge to the exact global minimum and can result in a suboptimal solution. This can be mitigated by using techniques such as learning rate scheduling and momentum-based updates. So, in SGD, we find out the gradient of the cost function of a single example at each iteration instead of the sum of the gradient of the cost function of all the examples.



SGD leads to many oscillations in comparison to GD. However, SGD is much faster since it doesn't go through the whole batch (Source: deeplearning.ai)

Adam optimization algorithm:
Adam is defined as "a method for efficient stochastic optimization that only requires first-order gradients with little memory requirement". Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data.

1. Actual step size taken by the Adam in each iteration is approximately bounded by the step size hyper-parameter. This property adds intuitive understanding to previous unintuitive learning rate hyper-parameter.
2. Step size of Adam update rule is invariant to the magnitude of the gradient, which helps a lot when going through areas with tiny gradients (such as saddle points or ravines). In theseareas SGD struggles to quickly navigate through them.



4) Advantages:

- The food delivery prediction project has a scalable architecture that can be extended to incorporate additional features such as weather conditions, restaurant popularity, and customer preferences. This scalability allows for more accurate predictions, making it suitable for a wide range of food delivery applications.

- The project's ability to automatically predict the delivery time also ensures a better customer experience by providing accurate delivery estimates and minimizing delays.

- One of the main advantages of using Long Short-Term Memory (LSTM) network in food delivery time prediction is its ability to handle sequential data. LSTMs can effectively capture the temporal dependencies in the food delivery data and use it to make accurate predictions. This is particularly useful in food delivery systems where the time taken for order preparation, traffic conditions, and distance between the restaurant and customer's location are critical factors affecting delivery time.

5) System Description:

Google Colab:

Google Colab was developed by Google to provide free access to GPU’s and TPU’s to anyone who needs them to build a machine learning or deep learning model.

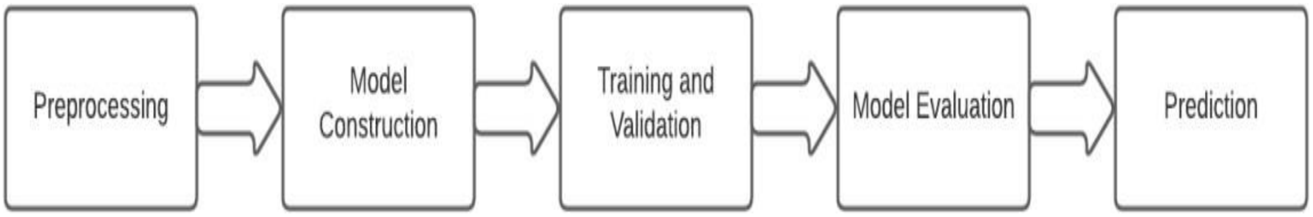
- n1-highmem-2 instance
- 2vCPU @ 2.2GHz
- 13GB RAM
- 100GB Free Space
- idle cut-off 90 minutes
- maximum 12 hours

6) Project Cycle:

- **Planning and requirement analysis**
  Planning: This stage involves defining the project objectives, identifying thestakeholders, and determining the scope of the project.

- **Designing the Software**
  Model Design: The neural network model is designed with an input layer, multiple hidden layers, and an output layer. The number of layers and neuronsin each layer are determined through experimentation and analysis.

- **Developing the project**



- **Testing**
  Model Testing: After training, the model is tested on the testing set to evaluate its performance using metrics such as accuracy, precision, recall,and R2 score.
  Model Tuning: The model is fine-tuned by adjusting the hyperparameters such as learning rate, batch size, and number of hidden layers to improve its performance.

- **Deployment**
  Deployment: Once the model is trained and fine-tuned, it can be deployed to recognize digits in new images.

- **Maintenance**
  Maintenance: The model needs to be maintained by periodically retraining andfine-tuning it with new data to improve its accuracy and performance.

7) System Requirements:

Processor: 2.3Ghz 8-Core Intel core i5

Memory: 8 GB

8) Disadvantages/Limitations:

- **Accuracy problem**
  It isn't working well for outside data, as the data used for testing was taken only from Kaggle data source.

- **Time complexity**
  Takes a lot of time and iterations to train the data.

9) Applications:

- Improved customer experience: By accurately predicting delivery times, customers can plan their meals and schedules more effectively, leading to a better overall experience.
- Optimal staffing and resource allocation: By accurately predicting food delivery times, restaurants can optimize their staffing and resource allocation, ensuring that the right number of employees and resources are available to meet customer demand.
- Real-time tracking and updates: A food delivery time predic-

tion model can be integrated with real-time tracking systems to provide customers with up-to-date information on the status of their order, including expected delivery time and any delays.

- Inventory management: By accurately predicting food delivery times, restaurants can better manage their inventory, ensuring that they have the right amount of food on hand to meet customer demand.
- Demand forecasting: By analyzing past delivery times and customer order history, the model can predict future demand for food delivery services, allowing restaurants to adjust their staffing and resource allocation accordingly.

10) References:

- https://www.kaggle.com/code/kmkarakaya/lstm-output-types-return-sequences-state
- https://www.simplilearn.com/tutorials/deep-learning-tutorial/rnn
- https://youtu.be/dxnXNiyKsGE
- https://medium.com/analytics-vidhya/lstms-explained-a-complete-technically-accurate-conceptual-guide-with-keras-2a650327e8f2
- https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/
- https://medium.com/@priyadarshi.cse/calculating-number-of-parameters-in-a-lstm-unit-layer-7e491978e1e4