1. Introduction

The purpose of this technical report is to provide a comprehensive overview of the code developed for building a car price prediction model. The code leverages Python, TensorFlow, and scikit-learn libraries to create and evaluate a neural network model that predicts the selling price of cars based on various features.

2. Data Preparation and Preprocessing

The code begins by loading a dataset from a CSV file (CAR DETAILS FROM CAR DEKHO.csv) using the pandas library. The following data preparation and preprocessing steps are applied:

2.1 Data Cleaning

Rows with missing values in specific columns ('name', 'year', 'selling_price', 'km_driven', 'fuel', 'seller_type', 'transmission') are dropped from the dataset to ensure data quality.

2.2 Data Type Conversion

The 'year' and 'km_driven' columns are converted from strings to integers since they represent numeric values.

2.3 Encoding Categorical Variables

Categorical variables ('fuel', 'seller_type', 'transmission') are one-hot encoded using the pd.get_dummies function to convert them into a format suitable for modeling.

3. Data Splitting

The dataset is split into training and testing sets using the train_test_split function from scikit-learn. This step allows for the evaluation of the model's performance on unseen data.

4. Feature Scaling

Standardization of features is performed using the StandardScaler from scikit-learn. 'year' and 'km_driven' columns are scaled to have a mean of 0 and a standard deviation of 1 to improve model convergence.

5. Model Development

5.1 Neural Network Architecture

A sequential neural network model is defined using TensorFlow's Keras API. The model consists of three layers: two hidden layers with ReLU activation functions and an output layer with a linear activation function. The architecture is as follows:

Input layer: 10 input features

Hidden layer 1: 128 units, ReLU activation

Hidden layer 2: 64 units, ReLU activation

Output layer: 1 unit (selling price prediction)

5.2 Model Compilation

The model is compiled using the Adam optimizer and mean squared error (MSE) as the loss function. This configuration is suitable for a regression problem where the goal is to predict continuous numerical values.

5.3 Model Training

The model is trained on the training dataset with 100 epochs using the model.fit method. During training, the neural network learns the relationships between the input features and the target variable (selling price).

6. Model Evaluation

The trained neural network model is evaluated on the testing dataset to assess its performance. The MSE (Mean Squared Error) is calculated to measure the accuracy of predictions.

7. Prediction

To demonstrate the model's predictive capabilities, an example car with specific features (year, km driven, fuel type, seller type, transmission) is provided. The features are scaled using the previously fitted StandardScaler, and the model predicts the selling price of the car.

8. Visualization

A scatter plot is generated to visualize the relationship between actual selling prices and predicted selling prices for the entire test dataset. Additionally, individual scatter plots are created for each row in the test set to provide a detailed view of model predictions for specific instances.

9. Conclusion

In conclusion, the code successfully builds and evaluates a neural network model for car price prediction. It demonstrates data preprocessing, model development, evaluation, and visualization steps, making it a valuable tool for predicting car prices based on various features.

This technical report provides an overview of the code's functionality and the steps involved in building and evaluating a car price prediction model. It serves as documentation for the code and helps readers understand its purpose and implementation details.

```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

# Load the dataset (assuming it's in a CSV file)
data = pd.read_csv('/content/CAR DETAILS FROM CAR DEKHO.csv')

# Data Cleaning
# Drop rows with missing values in any of the selected columns
columns_toclean = ['name', 'year', 'selling_price', 'km_driven',
'fuel', 'seller_type', 'transmission']
data_cleaned = data.dropna(subset=columns_to_clean)

# Convert 'year' and 'km_driven' to numeric data types (assuming they
are stored as strings)
data_cleaned['year'] = data_cleaned['year'].astype(int)
# Assuming 'km_driven' is already in numeric format
data_cleaned['km_driven'] = data_cleaned['km_driven'].astype(int)

# Encode categorical variables
data_cleaned = pd.get_dummies(data_cleaned, columns=['fuel',
'seller_type', 'transmission'])

# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(data_cleaned[['year', 'km_driven',
                                                                          'fuel_C
NG', 'fuel_Diesel', 'fuel_Petrol',
                                                                          'seller
_type_Dealer', 'seller_type_Individual', 'seller_type_Trustmark
Dealer',
                                                                          'transm
ission_Automatic', 'transmission_Manual']],
                                                             data_cleaned['selli
ng_price'], test_size=0.25)
```

```
# Feature Scaling
scaler = StandardScaler()
X_train[['year', 'km_driven']] = scaler.fit_transform(X_train[['year',
'km_driven']])
X_test[['year', 'km_driven']] = scaler.transform(X_test[['year',
'km_driven']])

# Define the neural network architecture
model = tf.keras.models.Sequential([
  tf.keras.layers.Dense(128, activation='relu',
input_shape=(X_train.shape[1],)),
  tf.keras.layers.Dense(64, activation='relu'),
  tf.keras.layers.Dense(1)
])

# Compile the neural network model
model.compile(optimizer='adam', loss='mse')

# Train the neural network model
model.fit(X_train, y_train, epochs=100)

# Evaluate the trained neural network model on the testing set
loss = model.evaluate(X_test, y_test)
print('Mean Squared Error (MSE):', loss)

# Predict the selling price of a car
# Scale the 'year' and 'km_driven' features for the car_features input
car_features = [2018, 10000, 0, 1, 0, 1, 0, 0, 1, 0]  # Example for a
car with features
# Predict the selling price of a car using the scaled features
# Predict the selling price of a car using the scaled features
scaled_car_features = np.array(car_features).reshape(1, -1)  # Reshape
to match the input shape of the model
predicted_selling_price = model.predict(scaled_car_features)

# Print the predicted selling price
print('Predicted selling price:', predicted_selling_price[0][0])
```

```
Epoch 1/100
102/102 [==============================] - 1s 2ms/step - loss:
574117773312.0000
Epoch 2/100
102/102 [==============================] - 0s 2ms/step - loss:
573698932736.0000
Epoch 3/100
102/102 [==============================] - 0s 2ms/step - loss:
571842297856.0000
Epoch 4/100
102/102 [==============================] - 0s 2ms/step - loss:
567428120576.0000
Epoch 5/100
102/102 [==============================] - 0s 2ms/step - loss:
559502393344.0000
Epoch 6/100
102/102 [==============================] - 0s 2ms/step - loss:
547355754496.0000
Epoch 7/100
102/102 [==============================] - 0s 2ms/step - loss:
530839764992.0000
Epoch 8/100
102/102 [==============================] - 0s 2ms/step - loss:
509981065216.0000
Epoch 9/100
102/102 [==============================] - 0s 2ms/step - loss:
485254561792.0000
Epoch 10/100
102/102 [==============================] - 0s 2ms/step - loss:
457716334592.0000
Epoch 11/100
102/102 [==============================] - 0s 2ms/step - loss:
428562612224.0000
Epoch 12/100
102/102 [==============================] - 0s 2ms/step - loss:
399180398592.0000
Epoch 13/100
102/102 [==============================] - 0s 2ms/step - loss:
371169689600.0000
Epoch 14/100
102/102 [==============================] - 0s 2ms/step - loss:
345906675712.0000
Epoch 15/100
102/102 [==============================] - 0s 2ms/step - loss:
324625432576.0000
Epoch 16/100
102/102 [==============================] - 0s 2ms/step - loss:
307113885696.0000
Epoch 17/100
102/102 [==============================] - 0s 2ms/step - loss:
293516410880.0000
Epoch 18/100
102/102 [==============================] - 0s 2ms/step - loss:
283087831040.0000
Epoch 19/100
```

```
102/102 [==============================] - 0s 2ms/step - loss:
274991185920.0000
Epoch 20/100
102/102 [==============================] - 0s 2ms/step - loss:
268398116864.0000
Epoch 21/100
102/102 [==============================] - 0s 2ms/step - loss:
262867681280.0000
Epoch 22/100
102/102 [==============================] - 0s 2ms/step - loss:
257966014464.0000
Epoch 23/100
102/102 [==============================] - 0s 2ms/step - loss:
253565321216.0000
Epoch 24/100
102/102 [==============================] - 0s 2ms/step - loss:
249379782656.0000
Epoch 25/100
102/102 [==============================] - 0s 2ms/step - loss:
245468725248.0000
Epoch 26/100
102/102 [==============================] - 0s 2ms/step - loss:
241754914816.0000
Epoch 27/100
102/102 [==============================] - 0s 2ms/step - loss:
238173241344.0000
Epoch 28/100
102/102 [==============================] - 0s 2ms/step - loss:
234760290304.0000
Epoch 29/100
102/102 [==============================] - 0s 2ms/step - loss:
231505723392.0000
Epoch 30/100
102/102 [==============================] - 0s 2ms/step - loss:
228388159488.0000
Epoch 31/100
102/102 [==============================] - 0s 2ms/step - loss:
225409105920.0000
Epoch 32/100
102/102 [==============================] - 0s 2ms/step - loss:
222495555584.0000
Epoch 33/100
102/102 [==============================] - 0s 2ms/step - loss:
219727396864.0000
Epoch 34/100
102/102 [==============================] - 0s 2ms/step - loss:
217066291200.0000
Epoch 35/100
102/102 [==============================] - 0s 3ms/step - loss:
214515302400.0000
Epoch 36/100
102/102 [==============================] - 0s 3ms/step - loss:
212101349376.0000
Epoch 37/100
102/102 [==============================] - 0s 3ms/step - loss:
209693343744.0000
Epoch 38/100
```

```
102/102 [==============================] - 0s 3ms/step - loss:
207450603520.0000
Epoch 39/100
102/102 [==============================] - 0s 3ms/step - loss:
205303103488.0000
Epoch 40/100
102/102 [==============================] - 0s 3ms/step - loss:
203198676992.0000
Epoch 41/100
102/102 [==============================] - 0s 3ms/step - loss:
201227501568.0000
Epoch 42/100
102/102 [==============================] - 0s 3ms/step - loss:
199244087296.0000
Epoch 43/100
102/102 [==============================] - 0s 3ms/step - loss:
197410684928.0000
Epoch 44/100
102/102 [==============================] - 0s 3ms/step - loss:
195581313024.0000
Epoch 45/100
102/102 [==============================] - 0s 2ms/step - loss:
193871659008.0000
Epoch 46/100
102/102 [==============================] - 0s 2ms/step - loss:
192227164160.0000
Epoch 47/100
102/102 [==============================] - 0s 2ms/step - loss:
190664130560.0000
Epoch 48/100
102/102 [==============================] - 0s 2ms/step - loss:
189138173952.0000
Epoch 49/100
102/102 [==============================] - 0s 2ms/step - loss:
187676704768.0000
Epoch 50/100
102/102 [==============================] - 0s 2ms/step - loss:
186280460288.0000
Epoch 51/100
102/102 [==============================] - 0s 2ms/step - loss:
184923111424.0000
Epoch 52/100
102/102 [==============================] - 0s 2ms/step - loss:
183657709568.0000
Epoch 53/100
102/102 [==============================] - 0s 2ms/step - loss:
182374137856.0000
Epoch 54/100
102/102 [==============================] - 0s 2ms/step - loss:
181235400704.0000
Epoch 55/100
102/102 [==============================] - 0s 2ms/step - loss:
180027785216.0000
Epoch 56/100
102/102 [==============================] - 0s 2ms/step - loss:
178936889344.0000
Epoch 57/100
```

```
102/102 [==============================] - 0s 2ms/step - loss:
177905352704.0000
Epoch 58/100
102/102 [==============================] - 0s 2ms/step - loss:
176883154944.0000
Epoch 59/100
102/102 [==============================] - 0s 2ms/step - loss:
175903506432.0000
Epoch 60/100
102/102 [==============================] - 0s 2ms/step - loss:
175028846592.0000
Epoch 61/100
102/102 [==============================] - 0s 2ms/step - loss:
174076411904.0000
Epoch 62/100
102/102 [==============================] - 0s 2ms/step - loss:
173224673280.0000
Epoch 63/100
102/102 [==============================] - 0s 2ms/step - loss:
172379668480.0000
Epoch 64/100
102/102 [==============================] - 0s 2ms/step - loss:
171613454336.0000
Epoch 65/100
102/102 [==============================] - 0s 2ms/step - loss:
170883481600.0000
Epoch 66/100
102/102 [==============================] - 0s 2ms/step - loss:
170116612096.0000
Epoch 67/100
102/102 [==============================] - 0s 2ms/step - loss:
169401958400.0000
Epoch 68/100
102/102 [==============================] - 0s 2ms/step - loss:
168721252352.0000
Epoch 69/100
102/102 [==============================] - 0s 2ms/step - loss:
168091779072.0000
Epoch 70/100
102/102 [==============================] - 0s 2ms/step - loss:
167489044480.0000
Epoch 71/100
102/102 [==============================] - 0s 2ms/step - loss:
166856572928.0000
Epoch 72/100
102/102 [==============================] - 0s 2ms/step - loss:
166250414080.0000
Epoch 73/100
102/102 [==============================] - 0s 2ms/step - loss:
165701042176.0000
Epoch 74/100
102/102 [==============================] - 0s 2ms/step - loss:
165125406720.0000
Epoch 75/100
102/102 [==============================] - 0s 2ms/step - loss:
164562960384.0000
Epoch 76/100
```

```
102/102 [==============================] - 0s 2ms/step - loss:
164039704576.0000
Epoch 77/100
102/102 [==============================] - 0s 2ms/step - loss:
163568025600.0000
Epoch 78/100
102/102 [==============================] - 0s 2ms/step - loss:
162979119104.0000
Epoch 79/100
102/102 [==============================] - 0s 2ms/step - loss:
162494005248.0000
Epoch 80/100
102/102 [==============================] - 0s 2ms/step - loss:
162038431744.0000
Epoch 81/100
102/102 [==============================] - 0s 2ms/step - loss:
161488715776.0000
Epoch 82/100
102/102 [==============================] - 0s 2ms/step - loss:
161075363840.0000
Epoch 83/100
102/102 [==============================] - 0s 2ms/step - loss:
160539590656.0000
Epoch 84/100
102/102 [==============================] - 0s 2ms/step - loss:
160112115712.0000
Epoch 85/100
102/102 [==============================] - 0s 2ms/step - loss:
159638732800.0000
Epoch 86/100
102/102 [==============================] - 0s 2ms/step - loss:
159211945984.0000
Epoch 87/100
102/102 [==============================] - 0s 2ms/step - loss:
158745116672.0000
Epoch 88/100
102/102 [==============================] - 0s 2ms/step - loss:
158348492800.0000
Epoch 89/100
102/102 [==============================] - 0s 2ms/step - loss:
157904338944.0000
Epoch 90/100
102/102 [==============================] - 0s 2ms/step - loss:
157510549504.0000
Epoch 91/100
102/102 [==============================] - 0s 2ms/step - loss:
157078126592.0000
Epoch 92/100
102/102 [==============================] - 0s 2ms/step - loss:
156681273344.0000
Epoch 93/100
102/102 [==============================] - 0s 3ms/step - loss:
156325249024.0000
Epoch 94/100
102/102 [==============================] - 0s 3ms/step - loss:
155921285120.0000
Epoch 95/100
```

```
102/102 [==============================] - 0s 3ms/step - loss:
155591180288.0000
Epoch 96/100
102/102 [==============================] - 0s 3ms/step - loss:
155220131840.0000
Epoch 97/100
102/102 [==============================] - 0s 3ms/step - loss:
154842365952.0000
Epoch 98/100
102/102 [==============================] - 0s 3ms/step - loss:
154520305664.0000
Epoch 99/100
102/102 [==============================] - 0s 3ms/step - loss:
154166640640.0000
Epoch 100/100
102/102 [==============================] - 0s 3ms/step - loss:
153861193728.0000
34/34 [==============================] - 0s 2ms/step - loss:
196904108032.0000
Mean Squared Error (MSE): 196904108032.0
1/1 [==============================] - 0s 62ms/step
Predicted selling price: 367819940.0
```