```
import pandas as pd

# Load the CSV file
df = pd.read_csv('/content/car data.csv')

# Display the first few rows of the DataFrame
df.head()
```

	Car_Name	Year	Selling_Price	Present_Price	Kms_Driven	Fuel_Type	Seller_Type	Transmission	Owner
0	ritz	2014	3.35	5.59	27000	Petrol	Dealer	Manual	0
1	sx4	2013	4.75	9.54	43000	Diesel	Dealer	Manual	0
2	ciaz	2017	7.25	9.85	6900	Petrol	Dealer	Manual	0
3	wagon r	2011	2.85	4.15	5200	Petrol	Dealer	Manual	0
4	swift	2014	4.60	6.87	42450	Diesel	Dealer	Manual	0

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 301 entries, 0 to 300
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Car_Name       301 non-null    object
 1   Year           301 non-null    int64
 2   Selling_Price  301 non-null    float64
 3   Present_Price  301 non-null    float64
 4   Kms_Driven     301 non-null    int64
 5   Fuel_Type      301 non-null    object
 6   Seller_Type    301 non-null    object
 7   Transmission   301 non-null    object
 8   Owner          301 non-null    int64
dtypes: float64(2), int64(3), object(4)
memory usage: 21.3+ KB
```

```
final_dataset = df[['Year', 'Selling_Price', 'Present_Price', 'Kms_Driven',
       'Fuel_Type', 'Seller_Type', 'Transmission', 'Owner']]
```

```
final_dataset['Current_Year'] = 2021
```

```
final_dataset['Age'] = final_dataset['Current_Year']-final_dataset['Year']
```

```
final_dataset.drop(['Year'],axis=1,inplace=True)
#drops the column labelled as "year" and doesnt return a copy as inplace = true. and axis = 1 represents columns
```

```
final_dataset.drop(['Current_Year'],axis=1,inplace=True)
#drops the column labelled as "Current_year" and doesnt return a copy as inplace = true. and axis = 1 represents columns
```

```
final_dataset=pd.get_dummies(final_dataset,drop_first=True)
#removes multiple columns of the dataset as some column contain the same
#information because the original column could assume a binary value.
```

```
final_dataset.corr(method ='pearson')
#to find the pairwise correlation of all columns in the dataframe
```

	Selling_Price	Present_Price	Kms_Driven	Owner	
Selling_Price	1.000000	0.878983	0.029187	-0.088344	-0.:
Present_Price	0.878983	1.000000	0.203647	0.008057	0.(
Kms_Driven	0.029187	0.203647	1.000000	0.089216	0..
Owner	-0.088344	0.008057	0.089216	1.000000	0.
Age	-0.236141	0.047584	0.524342	0.182104	1.(
Fuel_Type_Diesel	0.552339	0.473306	0.172515	-0.053469	-0.(
Fuel_Type_Petrol	-0.540571	-0.465244	-0.172874	0.055687	0.(
Seller_Type_Individual	-0.550724	-0.512030	-0.101419	0.124269	0.(
Transmission_Manual	-0.367128	-0.348715	-0.162510	-0.050316	-0.(

```
import matplotlib.pyplot as plt
%matplotlib inline
#to display the plot directly below the code cell.
```

```
corrmat = final_dataset.corr(method='pearson')
```

```
X= final_dataset.iloc[:,1:]
#slicing the dataset and reomoving the selling price for training the model
Y = final_dataset.iloc[:,0]
#storing the selling price for checking..as this is the value to be predicted
```

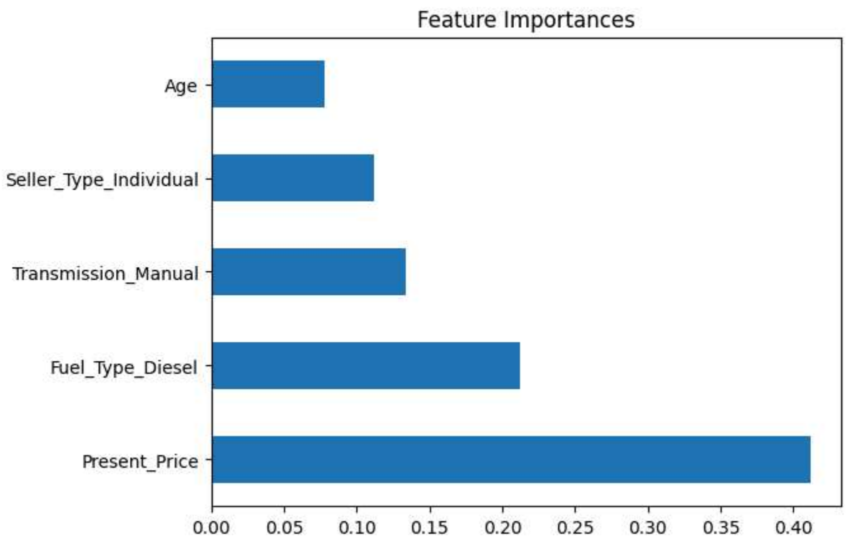
```
from sklearn.ensemble import ExtraTreesRegressor
#in ensemble predictions of several base estimators are built in with a given learning algorithm.
#we used ExtraTreesRegressor
model = ExtraTreesRegressor()
#This class implements a meta estimator that fits a number of randomized decision trees
#on various sub-samples of the dataset and uses averaging to improve the
#predictive accuracy and control over-fitting.
model.fit(X,Y)
```

```
▾ ExtraTreesRegressor
ExtraTreesRegressor()
```

```
print(model.feature_importances_)
#shows the feature importance that contribute to the selling price feature
```

```
[4.11963286e-01 4.25127321e-02 3.48229421e-04 7.78686520e-02
 2.12120300e-01 9.23020409e-03 1.11877918e-01 1.34078678e-01]
```

```
#plot graph of feature importances for better visualization
feat_importances = pd.Series(model.feature_importances_, index=X.columns)
feat_importances.nlargest(5).plot(kind='barh')
plt.title('Feature Importances')
plt.show()
```




```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Convert categorical variables into numerical representations
encoder = LabelEncoder()
df_encoded = df.copy()
for column in df_encoded.columns:
    df_encoded[column] = encoder.fit_transform(df_encoded[column])

import numpy as np
import pandas as pd

# Define a function to remove outliers using the IQR method
def remove_outliers(df):
    # Calculate Q1 (25th percentile) and Q3 (75th percentile) for each numerical feature
    Q1 = df.quantile(0.25)
    Q3 = df.quantile(0.75)
    # Calculate the interquartile range (IQR)
    IQR = Q3 - Q1
    # Define lower and upper bounds for outliers detection
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    # Identify outliers for each feature
    outliers = (df < lower_bound) | (df > upper_bound)
    # Remove rows containing outliers
    df_no_outliers = df[~outliers.any(axis=1)]
    return df_no_outliers

# Apply the remove_outliers function to your dataset
df_clean = remove_outliers(df_encoded)


X= df_clean.iloc[:,1:]
#slicing the dataset and reomoving the selling price for training the model
y = df_clean.iloc[:,0]
#storing the selling price for checking..as this is the value to be predicted

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Reshape the input data for RNN
X_train_rnn = np.expand_dims(X_train.values, axis=2)
X_test_rnn = np.expand_dims(X_test.values, axis=2)
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Define and compile the RNN regression model
rnn_model = Sequential([
    LSTM(64, activation='relu', return_sequences=True, input_shape=(X_train_rnn.shape[1], X_train_rnn.shape[2])),
    LSTM(32, activation='relu', return_sequences=True),
    LSTM(16, activation='relu'),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1)  # Output layer for regression
], name='RNN_Model')

# Compile the model
rnn_model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Print model summary
rnn_model.summary()
```

```
Model: "RNN_Model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_46 (LSTM)              (None, 8, 64)             16896

 lstm_47 (LSTM)              (None, 8, 32)             12416

 lstm_48 (LSTM)              (None, 16)                3136

 dense_47 (Dense)            (None, 64)                1088

 dropout_17 (Dropout)        (None, 64)                0

 dense_48 (Dense)            (None, 1)                 65

=================================================================
Total params: 33601 (131.25 KB)
Trainable params: 33601 (131.25 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Train the RNN model
history = rnn_model.fit(X_train_rnn, y_train, epochs=200, batch_size=32, validation_data=(X_test_rnn, y_test), verbose=1)
```

```
Epoch 1/200
5/5 [==============================] - 80s 200ms/step - loss: 3901.6331 - mae: 57.8545 - val_loss: 2958.2617 - val_ma
Epoch 2/200
5/5 [==============================] - 0s 29ms/step - loss: 3306.0911 - mae: 52.1605 - val_loss: 2032.1295 - val_mae
Epoch 3/200
5/5 [==============================] - 0s 31ms/step - loss: 2167.1062 - mae: 40.3247 - val_loss: 862.2363 - val_mae:
Epoch 4/200
5/5 [==============================] - 0s 37ms/step - loss: 1893.1112 - mae: 35.1895 - val_loss: 430.7460 - val_mae:
Epoch 5/200
5/5 [==============================] - 0s 32ms/step - loss: 1675.4614 - mae: 31.1750 - val_loss: 601.1045 - val_mae:
Epoch 6/200
5/5 [==============================] - 0s 29ms/step - loss: 1170.8778 - mae: 26.8142 - val_loss: 401.4681 - val_mae:
Epoch 7/200
5/5 [==============================] - 0s 40ms/step - loss: 1663.1354 - mae: 32.8665 - val_loss: 437.7604 - val_mae:
Epoch 8/200
5/5 [==============================] - 0s 30ms/step - loss: 1129.6112 - mae: 26.1211 - val_loss: 606.9973 - val_mae:
Epoch 9/200
5/5 [==============================] - 0s 31ms/step - loss: 1202.5750 - mae: 26.1289 - val_loss: 589.6293 - val_mae:
Epoch 10/200
5/5 [==============================] - 0s 27ms/step - loss: 1204.8440 - mae: 27.6025 - val_loss: 391.7556 - val_mae:
Epoch 11/200
5/5 [==============================] - 0s 31ms/step - loss: 1033.3127 - mae: 24.0702 - val_loss: 359.1985 - val_mae:
Epoch 12/200
5/5 [==============================] - 0s 29ms/step - loss: 941.4180 - mae: 23.8746 - val_loss: 455.6178 - val_mae: 1
Epoch 13/200
5/5 [==============================] - 0s 29ms/step - loss: 915.2896 - mae: 23.9115 - val_loss: 412.8951 - val_mae: 1
Epoch 14/200
5/5 [==============================] - 0s 30ms/step - loss: 798.5001 - mae: 22.4198 - val_loss: 350.2192 - val_mae: 1
Epoch 15/200
5/5 [==============================] - 0s 27ms/step - loss: 814.0058 - mae: 22.9337 - val_loss: 425.4970 - val_mae: 1
Epoch 16/200
5/5 [==============================] - 0s 30ms/step - loss: 842.1879 - mae: 23.7323 - val_loss: 449.0308 - val_mae: 1
Epoch 17/200
```

```
5/5 [==============================] - 0s 29ms/step - loss: 722.1412 - mae: 21.3155 - val_loss: 397.7488 - val_mae: 1
Epoch 18/200
5/5 [==============================] - 0s 34ms/step - loss: 606.5944 - mae: 19.7394 - val_loss: 376.2361 - val_mae: 1
Epoch 19/200
5/5 [==============================] - 0s 29ms/step - loss: 663.0952 - mae: 20.9217 - val_loss: 352.6595 - val_mae: 1
Epoch 20/200
5/5 [==============================] - 0s 30ms/step - loss: 592.7635 - mae: 19.5923 - val_loss: 316.1126 - val_mae: 1
Epoch 21/200
5/5 [==============================] - 0s 28ms/step - loss: 533.0563 - mae: 17.8449 - val_loss: 357.9040 - val_mae: 1
Epoch 22/200
5/5 [==============================] - 0s 27ms/step - loss: 533.1745 - mae: 18.5314 - val_loss: 320.9663 - val_mae: 1
Epoch 23/200
5/5 [==============================] - 0s 27ms/step - loss: 545.1687 - mae: 18.8340 - val_loss: 318.7091 - val_mae: 1
Epoch 24/200
5/5 [==============================] - 0s 27ms/step - loss: 642.5916 - mae: 21.0971 - val_loss: 331.1466 - val_mae: 1
Epoch 25/200
5/5 [==============================] - 0s 24ms/step - loss: 548.3433 - mae: 18.6968 - val_loss: 284.8543 - val_mae: 1
Epoch 26/200
5/5 [==============================] - 0s 30ms/step - loss: 485.8800 - mae: 17.5050 - val_loss: 257.3058 - val_mae: 1
Epoch 27/200
5/5 [==============================] - 0s 28ms/step - loss: 521.7460 - mae: 18.1014 - val_loss: 313.9673 - val_mae: 1
Epoch 28/200
5/5 [==============================] - 0s 28ms/step - loss: 494.4517 - mae: 17.3549 - val_loss: 294.5209 - val_mae: 1
Epoch 29/200
```

```
# Evaluate the model
loss, mae = rnn_model.evaluate(X_test_rnn, y_test)
print("Test Loss:", loss)
print("Test MAE:", mae)
```

```
2/2 [==============================] - 0s 14ms/step - loss: 384.1722 - mae: 14.7539
Test Loss: 384.1722412109375
Test MAE: 14.753941535949707
```

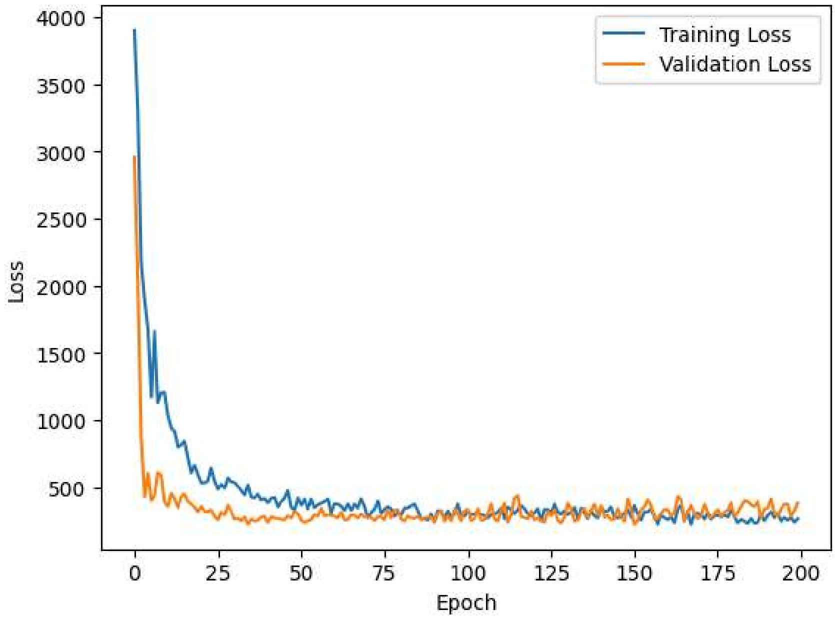
```
predictions=rnn_model.predict(X_test_rnn)
predictions1=rnn_model.predict(X_train_rnn)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test,predictions)
rmse = np.sqrt(mse)
print("RMSE : {:.2f}".format(rmse))
from sklearn.metrics import r2_score
r = r2_score(y_test, predictions)
print("R2 score : {}" . format(r))
```

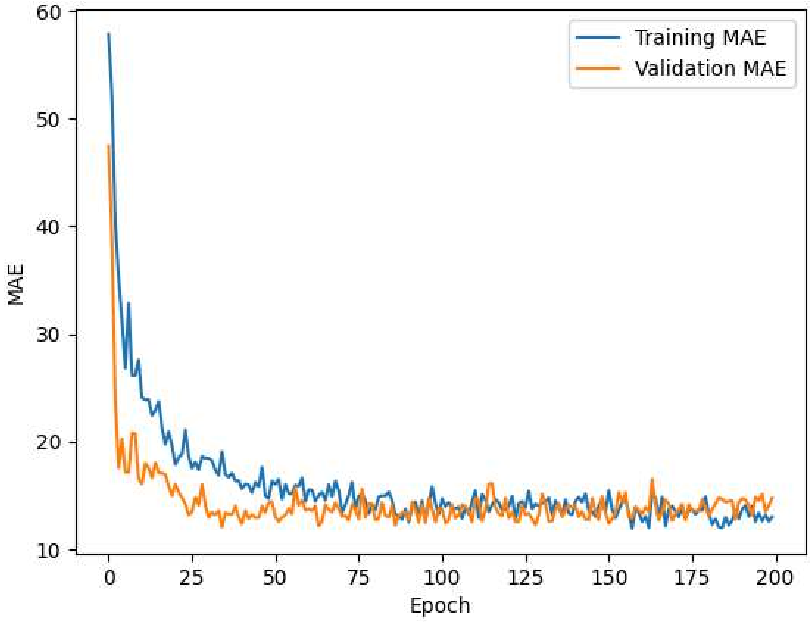
```
2/2 [==============================] - 1s 20ms/step
5/5 [==============================] - 0s 12ms/step
RMSE : 34.15
R2 score : -0.621487877129272
```

```
import matplotlib.pyplot as plt

# Plot training history
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()

plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.legend()
plt.show()
```






```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Reshape the input data for CNN
X_train_cnn = np.expand_dims(X_train.values, axis=2)
X_test_cnn = np.expand_dims(X_test.values, axis=2)



print("Shape of X_train_cnn:", X_train_cnn.shape)
```

```
Shape of X_train_cnn: (157, 8, 1)
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv1D, MaxPooling1D, LSTM, Dense

# Define the model
cnn_lstm_model = Sequential([
    Conv1D(32, kernel_size=3, activation='relu', input_shape=(X_train_cnn.shape[1], X_train_cnn.shape[2])),
    MaxPooling1D(pool_size=2),
    LSTM(64, activation='relu', return_sequences=True),
    LSTM(32, activation='relu'),
    Dense(64, activation='relu'),
    Dropout(0.5),
    Dense(1)  # Output layer for regression
], name='CNN_LSTM_Model')

# Compile the model
cnn_lstm_model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])

# Print model summary
cnn_lstm_model.summary()
```

```
Model: "CNN_LSTM_Model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv1d_14 (Conv1D)          (None, 6, 32)             128

 max_pooling1d_14 (MaxPooli  (None, 3, 32)             0
 ng1D)

 lstm_49 (LSTM)              (None, 3, 64)             24832

 lstm_50 (LSTM)              (None, 32)                12416

 dense_55 (Dense)            (None, 64)                2112

 dropout_21 (Dropout)        (None, 64)                0

 dense_56 (Dense)            (None, 1)                 65

=================================================================
Total params: 39553 (154.50 KB)
Trainable params: 39553 (154.50 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
# Train the model
history = cnn_lstm_model.fit(X_train_cnn, y_train, epochs=200, batch_size=32, validation_data=(X_test_cnn, y_test), verbose
```

```
Epoch 1/200
5/5 [==============================] - 7s 210ms/step - loss: 4135.6016 - mae: 59.6460 - val_loss: 3457.7727 - val_mae
Epoch 2/200
5/5 [==============================] - 0s 18ms/step - loss: 4027.3416 - mae: 58.8539 - val_loss: 3324.7993 - val_mae
Epoch 3/200
5/5 [==============================] - 0s 21ms/step - loss: 3797.7280 - mae: 57.0247 - val_loss: 3004.6978 - val_mae
Epoch 4/200
5/5 [==============================] - 0s 26ms/step - loss: 3216.6150 - mae: 51.9373 - val_loss: 1968.6804 - val_mae
Epoch 5/200
5/5 [==============================] - 0s 40ms/step - loss: 1974.8208 - mae: 38.1404 - val_loss: 928.8781 - val_mae:
Epoch 6/200
5/5 [==============================] - 0s 33ms/step - loss: 1726.6736 - mae: 33.4788 - val_loss: 672.7242 - val_mae:
Epoch 7/200
5/5 [==============================] - 0s 31ms/step - loss: 1077.3979 - mae: 25.8754 - val_loss: 638.7757 - val_mae:
Epoch 8/200
5/5 [==============================] - 0s 31ms/step - loss: 897.1542 - mae: 23.5530 - val_loss: 582.9091 - val_mae: 1
Epoch 9/200
5/5 [==============================] - 0s 31ms/step - loss: 1008.9902 - mae: 25.7363 - val_loss: 445.6202 - val_mae:
Epoch 10/200
5/5 [==============================] - 0s 35ms/step - loss: 866.2522 - mae: 22.9947 - val_loss: 344.4889 - val_mae: 1
Epoch 11/200
5/5 [==============================] - 0s 32ms/step - loss: 593.8239 - mae: 19.2531 - val_loss: 302.8698 - val_mae: 1
Epoch 12/200
5/5 [==============================] - 0s 34ms/step - loss: 848.7917 - mae: 23.3835 - val_loss: 556.7017 - val_mae: 2
Epoch 13/200
5/5 [==============================] - 0s 33ms/step - loss: 741.9877 - mae: 22.2037 - val_loss: 343.4616 - val_mae: 1
Epoch 14/200
5/5 [==============================] - 0s 34ms/step - loss: 641.7149 - mae: 19.2710 - val_loss: 339.5026 - val_mae: 1
Epoch 15/200
5/5 [==============================] - 0s 36ms/step - loss: 567.3409 - mae: 18.7044 - val_loss: 389.2560 - val_mae: 1
Epoch 16/200
```

```
5/5 [==============================] - 0s 37ms/step - loss: 603.2859 - mae: 20.1766 - val_loss: 305.7768 - val_mae: 1
Epoch 17/200
5/5 [==============================] - 0s 32ms/step - loss: 600.6036 - mae: 19.8751 - val_loss: 319.2968 - val_mae: 1
Epoch 18/200
5/5 [==============================] - 0s 34ms/step - loss: 514.9638 - mae: 18.3868 - val_loss: 334.4675 - val_mae: 1
Epoch 19/200
5/5 [==============================] - 0s 32ms/step - loss: 517.1215 - mae: 18.8208 - val_loss: 319.2922 - val_mae: 1
Epoch 20/200
5/5 [==============================] - 0s 35ms/step - loss: 515.7431 - mae: 17.6412 - val_loss: 309.6314 - val_mae: 1
Epoch 21/200
5/5 [==============================] - 0s 40ms/step - loss: 626.6337 - mae: 19.8402 - val_loss: 307.6683 - val_mae: 1
Epoch 22/200
5/5 [==============================] - 0s 36ms/step - loss: 597.8000 - mae: 19.1396 - val_loss: 300.1831 - val_mae: 1
Epoch 23/200
5/5 [==============================] - 0s 37ms/step - loss: 557.7083 - mae: 18.5409 - val_loss: 300.2847 - val_mae: 1
Epoch 24/200
5/5 [==============================] - 0s 41ms/step - loss: 520.9959 - mae: 18.7248 - val_loss: 294.9996 - val_mae: 1
Epoch 25/200
5/5 [==============================] - 0s 32ms/step - loss: 519.4893 - mae: 18.2673 - val_loss: 304.5624 - val_mae: 1
Epoch 26/200
5/5 [==============================] - 0s 19ms/step - loss: 463.2341 - mae: 17.4720 - val_loss: 275.5735 - val_mae: 1
Epoch 27/200
5/5 [==============================] - 0s 22ms/step - loss: 574.9915 - mae: 18.6342 - val_loss: 273.9836 - val_mae: 1
Epoch 28/200
5/5 [==============================] - 0s 17ms/step - loss: 422.3454 - mae: 16.6018 - val_loss: 284.6069 - val_mae: 1
Epoch 29/200
```

```
# Evaluate the model on the test data
loss, mae = cnn_lstm_model.evaluate(X_test_cnn, y_test, verbose=0)
print("Test Loss:", loss)
print("Test MAE:", mae)
```

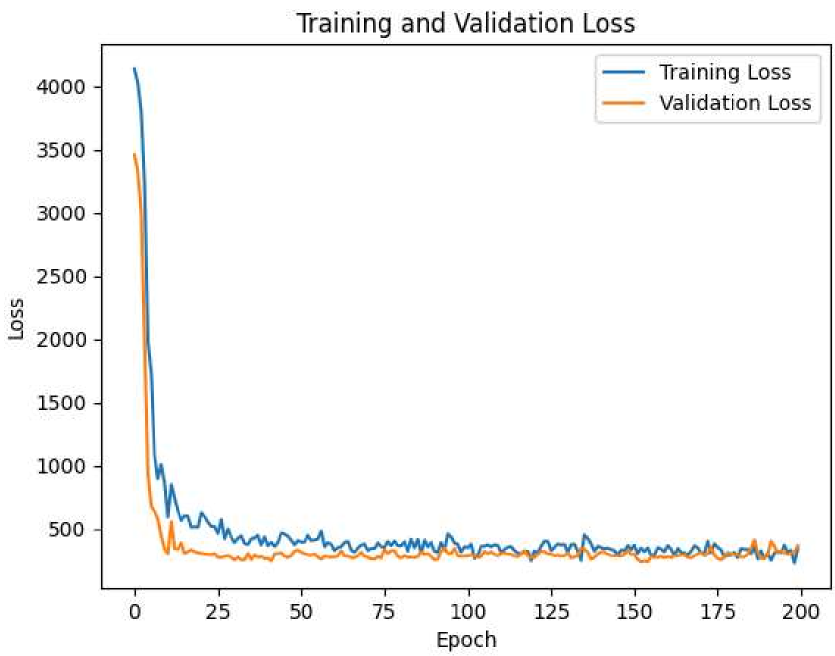
```
Test Loss: 368.05810546875
Test MAE: 15.046069145202637
```

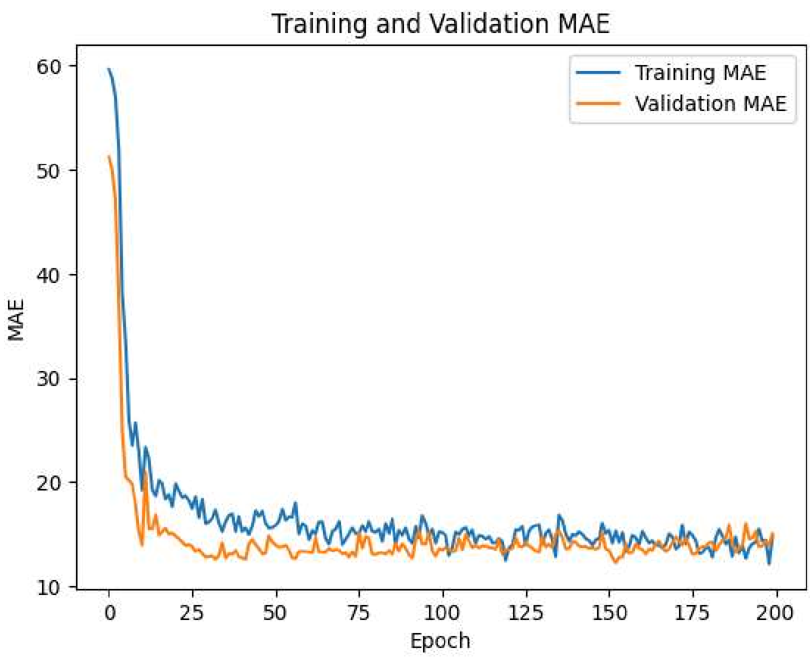
```
predictions=cnn_lstm_model.predict(X_test_cnn)
predictions1=cnn_lstm_model.predict(X_train_cnn)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test,predictions)
rmse = np.sqrt(mse)
print("RMSE : {:.2f}".format(rmse))
from sklearn.metrics import r2_score
r = r2_score(y_test, predictions)
print("R2 score : {}" . format(r))
```

```
2/2 [==============================] - 1s 8ms/step
5/5 [==============================] - 0s 4ms/step
RMSE : 32.20
R2 score : -0.4416882326069915
```

```
# Plot training and validation loss
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Training and Validation Loss')
plt.legend()
plt.show()

# Plot training and validation MAE
plt.plot(history.history['mae'], label='Training MAE')
plt.plot(history.history['val_mae'], label='Validation MAE')
plt.xlabel('Epoch')
plt.ylabel('MAE')
plt.title('Training and Validation MAE')
plt.legend()
plt.show()
```






```
from sklearn.ensemble import RandomForestRegressor
regressor=RandomForestRegressor()
```

```
from sklearn.model_selection import train_test_split #class to divide the data into train and validation set

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
#we divide the data into 2 parts :- 80% train and 20% test data
# and random_state is used to guarantee that same sequence of
#random numbers are generated each time you run the code.
#And unless there is some other randomness present in the process,
#the results produced will be same as always.
```

```
import numpy as np
n_estimators = [int(x) for x in np.linspace(start = 100, stop = 1200, num = 12)]
#n_estimators is a parameter of the random forest regressor which is used to control no of trees in the forest
#so we use 100 200 ....1200 trees for the model
print(n_estimators)
```

```
[100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200]
```

```
#Randomized Search CV
# Number of features to consider at every split
max_features = ['auto', 'sqrt'] # we first consider all the featurees and
#then sqare root number of features to train the model

# Maximum number of levels in tree
max_depth = [int(x) for x in np.linspace(5, 30, num = 6)]
#we create trees with 5 10 15 for each model...and train it

# Minimum number of samples required to split a node
min_samples_split = [2, 5, 10, 15, 100]
# we split as 2 nodes forst then 5 then 10 like that till 100 from the list

# Minimum number of samples required at each leaf node
```

```
from sklearn.model_selection import RandomizedSearchCV
#Randomized search on hyper parameters.
#used to select the best parameter for the model


random_grid = {'n_estimators': n_estimators,
               'max_features': max_features,
               'max_depth': max_depth,
               'min_samples_split': min_samples_split,
               'min_samples_leaf': min_samples_leaf}

print(random_grid)
```

```
{'n_estimators': [100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 1100, 1200], 'max_features': ['auto', 'sqrt'], 'ma
```

```
rf = RandomForestRegressor()
# Random search of parameters, using 3 fold cross validation,
# search across 100 different combinations
rf_random = RandomizedSearchCV(estimator = rf, param_distributions = random_grid,scoring='neg_mean_squared_error', n_iter =


rf_random.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   1
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   1
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   1
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   1
[CV] END max_depth=10, max_features=sqrt, min_samples_leaf=5, min_samples_split=5, n_estimators=900; total time=   1
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=
[CV] END max_depth=15, max_features=sqrt, min_samples_leaf=2, min_samples_split=10, n_estimators=1100; total time=
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has bee
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has bee
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has bee
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has bee
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has bee
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=100, n_estimators=300; total time=
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has bee
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=   0
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has bee
  warn(
[CV] END max_depth=15, max_features=auto, min_samples_leaf=5, min_samples_split=5, n_estimators=400; total time=   0
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_forest.py:413: FutureWarning: `max_features='auto'` has bee
  warn(
```