# imldti2ng

August 12, 2024

## 1 Understanding of AI Final Assignment: Analysis of Car sale data

Compare regression models that predict the price of a car based on a single numerical input feature. Based on your results, which numerical variable in th dataset is the best predictor for a car's price, and why? For each numerical inp t feature, is the price better fit by a linear model or by a non-linear (e.g. polynomi l) model?

[16]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score
import matplotlib.pyplot as plt

# Loading the dataset
file_path = 'car_sales_data_24.csv'
car_sales_data = pd.read_csv(file_path)

# Selecting numerical features
numerical_features = ['Engine size', 'Year of manufacture', 'Mileage']

# Preparing the dataset
X = car_sales_data[numerical_features]
y = car_sales_data['Price']

# Splittting 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)

# Function to fit and evaluate models
def fit_and_evaluate_model(feature_name):
    # Linear regression
    linear_model = LinearRegression()
    linear_model.fit(X_train[[feature_name]], y_train)
    y_pred_linear = linear_model.predict(X_test[[feature_name]])
```

```python
    linear_mse = mean_squared_error(y_test, y_pred_linear)
    linear_r2 = r2_score(y_test, y_pred_linear)

    # Polynomial regression (degree 2)
    poly_features = PolynomialFeatures(degree=2)
    X_train_poly = poly_features.fit_transform(X_train[[feature_name]])
    X_test_poly = poly_features.transform(X_test[[feature_name]])
    poly_model = LinearRegression()
    poly_model.fit(X_train_poly, y_train)
    y_pred_poly = poly_model.predict(X_test_poly)
    poly_mse = mean_squared_error(y_test, y_pred_poly)
    poly_r2 = r2_score(y_test, y_pred_poly)

    return {
        'feature': feature_name,
        'linear_mse': linear_mse,
        'linear_r2': linear_r2,
        'poly_mse': poly_mse,
        'poly_r2': poly_r2
    }

# Evaluating models for each numerical feature
results = [fit_and_evaluate_model(feature) for feature in numerical_features]

# Displaying the results
results_df = pd.DataFrame(results)
print(results_df)

# Determining the best predictor and model type
best_linear_model = results_df.loc[results_df['linear_r2'].idxmax()]
best_poly_model = results_df.loc[results_df['poly_r2'].idxmax()]

print("\nBest linear model:")
print(best_linear_model)
print("\nBest polynomial model:")
print(best_poly_model)
```

```
               feature    linear_mse  linear_r2      poly_mse   poly_r2
0          Engine size  2.304992e+08   0.150626  2.303262e+08  0.151263
1  Year of manufacture  1.326790e+08   0.511087  1.059939e+08  0.609419
2              Mileage  1.624686e+08   0.401314  1.296203e+08  0.522358

Best linear model:
feature        Year of manufacture
linear_mse        132678999.947931
linear_r2                 0.511087
poly_mse          105993894.202077
```

```
poly_r2                  0.609419
Name: 1, dtype: object

Best polynomial model:
feature       Year of manufacture
linear_mse       132678999.947931
linear_r2                0.511087
poly_mse         105993894.202077
poly_r2                  0.609419
Name: 1, dtype: object
```

Consider regression models that take multiple numerical variables as input features to predict the price of a car. Does the inclusion of multiple input features improve the accuracy of the model's prediction compared to the single-input feature models that you explored in part (a)?

[17]:
```python
# Fit and evaluate multiple regression model with all numerical features
def fit_and_evaluate_multiple_model():
    # Linear regression with multiple features
    linear_model = LinearRegression()
    linear_model.fit(X_train, y_train)
    y_pred_linear = linear_model.predict(X_test)
    linear_mse = mean_squared_error(y_test, y_pred_linear)
    linear_r2 = r2_score(y_test, y_pred_linear)

    # Polynomial regression (degree 2) with multiple features
    poly_features = PolynomialFeatures(degree=2)
    X_train_poly = poly_features.fit_transform(X_train)
    X_test_poly = poly_features.transform(X_test)
    poly_model = LinearRegression()
    poly_model.fit(X_train_poly, y_train)
    y_pred_poly = poly_model.predict(X_test_poly)
    poly_mse = mean_squared_error(y_test, y_pred_poly)
    poly_r2 = r2_score(y_test, y_pred_poly)

    return {
        'linear_mse': linear_mse,
        'linear_r2': linear_r2,
        'poly_mse': poly_mse,
        'poly_r2': poly_r2
    }

# Evaluate multiple regression models
multiple_model_results = fit_and_evaluate_multiple_model()

# Display the results for multiple regression models
multiple_model_results_df = pd.DataFrame([multiple_model_results])
print("Multiple-Input Feature Models:")
print(multiple_model_results_df)
```

```
Multiple-Input Feature Models:
     linear_mse  linear_r2       poly_mse   poly_r2
0  8.915862e+07   0.671456  2.931149e+07  0.891989
```

In parts (a) and (b) you only considered models that use the numerical variables from the dataset as inputs. However, there are also several categorical variables in the dataset that are likely to affect the price of the car. Now train a regression model that uses all relevant input variables (both categorical and numerical) to predict the price (e.g. a Random Forest Regressor model). Does this improve the accuracy of your results?

[18]:
```python
# Importing the libraries
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import LabelEncoder


# Selecting features
numerical_features = ['Engine size', 'Year of manufacture', 'Mileage']
categorical_features = ['Manufacturer', 'Model', 'Fuel type']

# Preparing the dataset
X = car_sales_data[numerical_features + categorical_features]
y = car_sales_data['Price']

# Encode categorical variables using LabelEncoder
label_encoders = {}
for feature in categorical_features:
    label_encoders[feature] = LabelEncoder()
    X[feature] = label_encoders[feature].fit_transform(X[feature])

# 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)

# Train the Random Forest Regressor
model = RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Predict on the test set
y_pred = model.predict(X_test)

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Random Forest Regressor with all features (using LabelEncoder):")
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")
```

```
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\2931713295.py:18:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\2931713295.py:18:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\2931713295.py:18:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
```

```
Random Forest Regressor with all features (using LabelEncoder):
Mean Squared Error: 475768.78947792
R-squared: 0.9982468229900868
```

Develop an Artificial Neural Network (ANN) model to predict the price of a car based on all the available information from the dataset. How does its performance compare to the other supervised learning models that you have considered? Discuss your choices for the architecture of the neural network that you used, and describe how you tuned the hyperparameters in your model to achieve the best performance.

[19]:
```python
# Importing the libraries
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import r2_score, mean_squared_error
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping

# Define features and target
X = df.drop('Price', axis=1)
y = df['Price']
```

```python
# Preprocessing for numerical and categorical data
numerical_features = ['Engine size', 'Year of manufacture', 'Mileage']
categorical_features = ['Manufacturer', 'Model', 'Fuel type']

numerical_transformer = StandardScaler()
categorical_transformer = OneHotEncoder(drop='first', handle_unknown='ignore')

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numerical_transformer, numerical_features),
        ('cat', categorical_transformer, categorical_features)
    ])

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

# Apply the transformations to the features
X_train = preprocessor.fit_transform(X_train)
X_test = preprocessor.transform(X_test)

# Initialize the ANN
model = Sequential()

# Add input layer and first hidden layer
model.add(Dense(units=128, activation='relu', input_shape=(X_train.shape[1],)))

# Add second hidden layer
model.add(Dense(units=64, activation='relu'))

# Add third hidden layer
model.add(Dense(units=32, activation='relu'))

# Add output layer
model.add(Dense(units=1))

# Compile the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),␣
↪loss='mse')

# Early stopping to prevent overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10,␣
↪restore_best_weights=True)

# Train the model
```

```python
history = model.fit(X_train, y_train, validation_split=0.2, epochs=100,␣
 ↪batch_size=32, callbacks=[early_stopping], verbose=1)

# Predict on the test set
y_pred = model.predict(X_test)

# Calculate R-squared and RMSE
r2 = r2_score(y_test, y_pred)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))

print(f"R-squared: {r2:.4f}")
print(f"RMSE: {rmse:.2f}")
```

```
C:\Users\sleek\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/100
1000/1000          7s 4ms/step -
loss: 277693184.0000 - val_loss: 26553234.0000
Epoch 2/100
1000/1000          3s 3ms/step -
loss: 24479264.0000 - val_loss: 9592022.0000
Epoch 3/100
1000/1000          3s 3ms/step -
loss: 9058254.0000 - val_loss: 4633232.0000
Epoch 4/100
1000/1000          3s 3ms/step -
loss: 5374638.5000 - val_loss: 2639710.5000
Epoch 5/100
1000/1000          3s 3ms/step -
loss: 2901185.5000 - val_loss: 1587659.3750
Epoch 6/100
1000/1000          3s 3ms/step -
loss: 1810975.6250 - val_loss: 1141285.3750
Epoch 7/100
1000/1000          3s 3ms/step -
loss: 1308140.2500 - val_loss: 1062810.6250
Epoch 8/100
1000/1000          3s 3ms/step -
loss: 909141.3750 - val_loss: 664366.8750
Epoch 9/100
1000/1000          3s 3ms/step -
loss: 617248.3750 - val_loss: 542324.1250
Epoch 10/100
1000/1000          3s 3ms/step -
```

```
loss: 565027.6875 - val_loss: 531906.9375
Epoch 11/100
1000/1000            5s 3ms/step -
loss: 459976.5000 - val_loss: 373560.8125
Epoch 12/100
1000/1000            3s 3ms/step -
loss: 420419.0312 - val_loss: 310383.9375
Epoch 13/100
1000/1000            3s 3ms/step -
loss: 281909.8125 - val_loss: 278660.3125
Epoch 14/100
1000/1000            3s 3ms/step -
loss: 246106.5156 - val_loss: 204500.1406
Epoch 15/100
1000/1000            3s 3ms/step -
loss: 185023.2812 - val_loss: 145483.1875
Epoch 16/100
1000/1000            3s 3ms/step -
loss: 159893.9062 - val_loss: 113066.5938
Epoch 17/100
1000/1000            3s 3ms/step -
loss: 119299.4219 - val_loss: 105820.2734
Epoch 18/100
1000/1000            3s 3ms/step -
loss: 114611.1172 - val_loss: 83193.4688
Epoch 19/100
1000/1000            3s 3ms/step -
loss: 86044.9688 - val_loss: 63957.3398
Epoch 20/100
1000/1000            3s 3ms/step -
loss: 64394.3633 - val_loss: 51369.4883
Epoch 21/100
1000/1000            3s 3ms/step -
loss: 62895.9219 - val_loss: 44516.4297
Epoch 22/100
1000/1000            3s 3ms/step -
loss: 45546.1094 - val_loss: 51995.1406
Epoch 23/100
1000/1000            3s 3ms/step -
loss: 59166.5586 - val_loss: 42104.7227
Epoch 24/100
1000/1000            3s 3ms/step -
loss: 48194.7969 - val_loss: 49662.5312
Epoch 25/100
1000/1000            3s 3ms/step -
loss: 42622.3281 - val_loss: 46727.5234
Epoch 26/100
1000/1000            3s 3ms/step -
```

```
loss: 41756.1641 - val_loss: 45528.5898
Epoch 27/100
1000/1000            3s 3ms/step -
loss: 48539.7383 - val_loss: 37525.8281
Epoch 28/100
1000/1000            3s 3ms/step -
loss: 34094.2773 - val_loss: 69808.5312
Epoch 29/100
1000/1000            3s 3ms/step -
loss: 46101.4219 - val_loss: 29629.7773
Epoch 30/100
1000/1000            3s 3ms/step -
loss: 25442.9082 - val_loss: 42937.8438
Epoch 31/100
1000/1000            3s 3ms/step -
loss: 28879.7305 - val_loss: 32125.0566
Epoch 32/100
1000/1000            3s 3ms/step -
loss: 29978.6016 - val_loss: 35532.2461
Epoch 33/100
1000/1000            3s 3ms/step -
loss: 24041.7324 - val_loss: 23923.5898
Epoch 34/100
1000/1000            3s 3ms/step -
loss: 26431.3535 - val_loss: 16014.7275
Epoch 35/100
1000/1000            3s 3ms/step -
loss: 26175.5078 - val_loss: 14637.6465
Epoch 36/100
1000/1000            3s 3ms/step -
loss: 26894.7148 - val_loss: 17936.1758
Epoch 37/100
1000/1000            3s 3ms/step -
loss: 31593.7441 - val_loss: 22432.3926
Epoch 38/100
1000/1000            3s 3ms/step -
loss: 26027.0977 - val_loss: 13853.5234
Epoch 39/100
1000/1000            3s 3ms/step -
loss: 20374.6738 - val_loss: 31748.6953
Epoch 40/100
1000/1000            3s 3ms/step -
loss: 20278.6543 - val_loss: 13893.5430
Epoch 41/100
1000/1000            3s 3ms/step -
loss: 21653.9980 - val_loss: 14998.0342
Epoch 42/100
1000/1000            3s 3ms/step -
```

```
loss: 23695.1816 - val_loss: 14249.8809
Epoch 43/100
1000/1000            3s 3ms/step -
loss: 23998.7793 - val_loss: 23537.8496
Epoch 44/100
1000/1000            6s 3ms/step -
loss: 18038.6699 - val_loss: 15357.3398
Epoch 45/100
1000/1000            3s 3ms/step -
loss: 19063.9629 - val_loss: 11542.7959
Epoch 46/100
1000/1000            4s 3ms/step -
loss: 18774.8867 - val_loss: 21547.0840
Epoch 47/100
1000/1000            3s 3ms/step -
loss: 19994.5156 - val_loss: 15743.9824
Epoch 48/100
1000/1000            4s 3ms/step -
loss: 25567.3047 - val_loss: 11767.8750
Epoch 49/100
1000/1000            4s 4ms/step -
loss: 16597.1250 - val_loss: 28509.2266
Epoch 50/100
1000/1000            4s 4ms/step -
loss: 18066.2832 - val_loss: 33378.5469
Epoch 51/100
1000/1000            3s 3ms/step -
loss: 18958.0879 - val_loss: 10967.9766
Epoch 52/100
1000/1000            3s 3ms/step -
loss: 15097.1221 - val_loss: 11930.5186
Epoch 53/100
1000/1000            5s 3ms/step -
loss: 14814.1367 - val_loss: 11880.3232
Epoch 54/100
1000/1000            3s 3ms/step -
loss: 21722.8379 - val_loss: 14350.3750
Epoch 55/100
1000/1000            5s 3ms/step -
loss: 14925.1611 - val_loss: 11732.0342
Epoch 56/100
1000/1000            6s 4ms/step -
loss: 15154.9756 - val_loss: 13241.9648
Epoch 57/100
1000/1000            5s 3ms/step -
loss: 16712.5215 - val_loss: 11483.2617
Epoch 58/100
1000/1000            4s 3ms/step -
```

```
loss: 23093.5332 - val_loss: 22430.5449
Epoch 59/100
1000/1000            3s 3ms/step -
loss: 13205.5537 - val_loss: 17424.4004
Epoch 60/100
1000/1000            5s 3ms/step -
loss: 17431.6289 - val_loss: 13087.5967
Epoch 61/100
1000/1000            3s 3ms/step -
loss: 13466.2324 - val_loss: 11515.3877
313/313            1s 4ms/step
R-squared: 1.0000
RMSE: 107.80
```

## 2 ANN with a Learning rate of 0.1

[4]:
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Load the dataset
file_path = 'car_sales_data_24.csv'
car_sales_data = pd.read_csv(file_path)

# Selecting features
numerical_features = ['Engine size', 'Year of manufacture', 'Mileage']
categorical_features = ['Manufacturer', 'Model', 'Fuel type']

# Prepare the dataset
X = car_sales_data[numerical_features + categorical_features]
y = car_sales_data['Price']

# Encode categorical variables using LabelEncoder
label_encoders = {}
for feature in categorical_features:
    label_encoders[feature] = LabelEncoder()
    X[feature] = label_encoders[feature].fit_transform(X[feature])

# Scale numerical features
scaler = StandardScaler()
X[numerical_features] = scaler.fit_transform(X[numerical_features])
```

```
# 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)

# Define the architecture of the neural network
def create_model(learning_rate=0.1, dropout_rate=0.2):
    model = Sequential()
    model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(loss='mean_squared_error', optimizer=optimizer,␣
↪metrics=['mse'])
    return model

# Create the model
model = create_model()

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32,␣
↪validation_split=0.2, verbose=1)

# Predict on the test set
y_pred = model.predict(X_test).flatten()

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Artificial Neural Network Model:")
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Plot training & validation loss values
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```

```
Epoch 1/100
```

```
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\736496043.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\736496043.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\736496043.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\736496043.py:30:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[numerical_features] = scaler.fit_transform(X[numerical_features])
C:\Users\sleek\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
1000/1000             3s 2ms/step -
loss: 85700000.0000 - mse: 85700000.0000 - val_loss: 19292022.0000 - val_mse:
19292022.0000
Epoch 2/100
1000/1000             3s 3ms/step -
loss: 25139802.0000 - mse: 25139802.0000 - val_loss: 16458980.0000 - val_mse:
16458980.0000
Epoch 3/100
1000/1000             2s 2ms/step -
loss: 24628988.0000 - mse: 24628988.0000 - val_loss: 14669329.0000 - val_mse:
```

```
14669329.0000
Epoch 4/100
1000/1000            2s 2ms/step -
loss: 23181108.0000 - mse: 23181108.0000 - val_loss: 13318300.0000 - val_mse:
13318300.0000
Epoch 5/100
1000/1000            3s 1ms/step -
loss: 19754624.0000 - mse: 19754624.0000 - val_loss: 12637693.0000 - val_mse:
12637693.0000
Epoch 6/100
1000/1000            2s 2ms/step -
loss: 18740126.0000 - mse: 18740126.0000 - val_loss: 21425308.0000 - val_mse:
21425308.0000
Epoch 7/100
1000/1000            2s 1ms/step -
loss: 17606250.0000 - mse: 17606250.0000 - val_loss: 22533360.0000 - val_mse:
22533360.0000
Epoch 8/100
1000/1000            2s 2ms/step -
loss: 16904096.0000 - mse: 16904096.0000 - val_loss: 20059156.0000 - val_mse:
20059156.0000
Epoch 9/100
1000/1000            1s 1ms/step -
loss: 16704799.0000 - mse: 16704799.0000 - val_loss: 19080252.0000 - val_mse:
19080252.0000
Epoch 10/100
1000/1000            2s 2ms/step -
loss: 15725194.0000 - mse: 15725194.0000 - val_loss: 15075308.0000 - val_mse:
15075308.0000
Epoch 11/100
1000/1000            2s 1ms/step -
loss: 16730637.0000 - mse: 16730637.0000 - val_loss: 28307576.0000 - val_mse:
28307576.0000
Epoch 12/100
1000/1000            2s 1ms/step -
loss: 16421158.0000 - mse: 16421158.0000 - val_loss: 21108136.0000 - val_mse:
21108136.0000
Epoch 13/100
1000/1000            1s 1ms/step -
loss: 13120211.0000 - mse: 13120211.0000 - val_loss: 36914084.0000 - val_mse:
36914084.0000
Epoch 14/100
1000/1000            1s 1ms/step -
loss: 14106537.0000 - mse: 14106537.0000 - val_loss: 23094602.0000 - val_mse:
23094602.0000
Epoch 15/100
1000/1000            2s 2ms/step -
loss: 14686447.0000 - mse: 14686447.0000 - val_loss: 19640822.0000 - val_mse:
```

```
19640822.0000
Epoch 16/100
1000/1000            2s 2ms/step -
loss: 14306306.0000 - mse: 14306306.0000 - val_loss: 22054620.0000 - val_mse:
22054620.0000
Epoch 17/100
1000/1000            1s 1ms/step -
loss: 13150661.0000 - mse: 13150661.0000 - val_loss: 21600496.0000 - val_mse:
21600496.0000
Epoch 18/100
1000/1000            2s 2ms/step -
loss: 12851066.0000 - mse: 12851066.0000 - val_loss: 25286076.0000 - val_mse:
25286076.0000
Epoch 19/100
1000/1000            2s 2ms/step -
loss: 12424508.0000 - mse: 12424508.0000 - val_loss: 27989066.0000 - val_mse:
27989066.0000
Epoch 20/100
1000/1000            2s 2ms/step -
loss: 12683110.0000 - mse: 12683110.0000 - val_loss: 16934806.0000 - val_mse:
16934806.0000
Epoch 21/100
1000/1000            2s 2ms/step -
loss: 12880990.0000 - mse: 12880990.0000 - val_loss: 21505724.0000 - val_mse:
21505724.0000
Epoch 22/100
1000/1000            3s 2ms/step -
loss: 12323287.0000 - mse: 12323287.0000 - val_loss: 21353664.0000 - val_mse:
21353664.0000
Epoch 23/100
1000/1000            2s 1ms/step -
loss: 12370038.0000 - mse: 12370038.0000 - val_loss: 12019086.0000 - val_mse:
12019086.0000
Epoch 24/100
1000/1000            2s 1ms/step -
loss: 12981132.0000 - mse: 12981132.0000 - val_loss: 15818745.0000 - val_mse:
15818745.0000
Epoch 25/100
1000/1000            1s 1ms/step -
loss: 13126239.0000 - mse: 13126239.0000 - val_loss: 22779558.0000 - val_mse:
22779558.0000
Epoch 26/100
1000/1000            1s 1ms/step -
loss: 12039489.0000 - mse: 12039489.0000 - val_loss: 28566190.0000 - val_mse:
28566190.0000
Epoch 27/100
1000/1000            2s 2ms/step -
loss: 11897853.0000 - mse: 11897853.0000 - val_loss: 20679612.0000 - val_mse:
```

```
20679612.0000
Epoch 28/100
1000/1000            2s 2ms/step -
loss: 11398625.0000 - mse: 11398625.0000 - val_loss: 14879545.0000 - val_mse:
14879545.0000
Epoch 29/100
1000/1000            1s 1ms/step -
loss: 11871618.0000 - mse: 11871618.0000 - val_loss: 17585262.0000 - val_mse:
17585262.0000
Epoch 30/100
1000/1000            1s 1ms/step -
loss: 12247034.0000 - mse: 12247034.0000 - val_loss: 17382844.0000 - val_mse:
17382844.0000
Epoch 31/100
1000/1000            3s 1ms/step -
loss: 12482577.0000 - mse: 12482577.0000 - val_loss: 13042378.0000 - val_mse:
13042378.0000
Epoch 32/100
1000/1000            1s 1ms/step -
loss: 12056432.0000 - mse: 12056432.0000 - val_loss: 24562250.0000 - val_mse:
24562250.0000
Epoch 33/100
1000/1000            1s 1ms/step -
loss: 11505975.0000 - mse: 11505975.0000 - val_loss: 18797066.0000 - val_mse:
18797066.0000
Epoch 34/100
1000/1000            2s 1ms/step -
loss: 11416682.0000 - mse: 11416682.0000 - val_loss: 19969108.0000 - val_mse:
19969108.0000
Epoch 35/100
1000/1000            1s 1ms/step -
loss: 12022302.0000 - mse: 12022302.0000 - val_loss: 14567839.0000 - val_mse:
14567839.0000
Epoch 36/100
1000/1000            1s 1ms/step -
loss: 11711387.0000 - mse: 11711387.0000 - val_loss: 15603911.0000 - val_mse:
15603911.0000
Epoch 37/100
1000/1000            2s 1ms/step -
loss: 11730796.0000 - mse: 11730796.0000 - val_loss: 16716649.0000 - val_mse:
16716649.0000
Epoch 38/100
1000/1000            1s 1ms/step -
loss: 11638209.0000 - mse: 11638209.0000 - val_loss: 23559704.0000 - val_mse:
23559704.0000
Epoch 39/100
1000/1000            1s 1ms/step -
loss: 10590657.0000 - mse: 10590657.0000 - val_loss: 27205868.0000 - val_mse:
```

```
27205868.0000
Epoch 40/100
1000/1000            1s 1ms/step -
loss: 10674844.0000 - mse: 10674844.0000 - val_loss: 13263690.0000 - val_mse:
13263690.0000
Epoch 41/100
1000/1000            2s 2ms/step -
loss: 10207981.0000 - mse: 10207981.0000 - val_loss: 18818298.0000 - val_mse:
18818298.0000
Epoch 42/100
1000/1000            2s 2ms/step -
loss: 9936115.0000 - mse: 9936115.0000 - val_loss: 19995040.0000 - val_mse:
19995040.0000
Epoch 43/100
1000/1000            3s 2ms/step -
loss: 9233119.0000 - mse: 9233119.0000 - val_loss: 13105951.0000 - val_mse:
13105951.0000
Epoch 44/100
1000/1000            2s 2ms/step -
loss: 9574514.0000 - mse: 9574514.0000 - val_loss: 20059166.0000 - val_mse:
20059166.0000
Epoch 45/100
1000/1000            2s 2ms/step -
loss: 9818690.0000 - mse: 9818690.0000 - val_loss: 14899493.0000 - val_mse:
14899493.0000
Epoch 46/100
1000/1000            2s 2ms/step -
loss: 9342079.0000 - mse: 9342079.0000 - val_loss: 15354940.0000 - val_mse:
15354940.0000
Epoch 47/100
1000/1000            2s 2ms/step -
loss: 9429062.0000 - mse: 9429062.0000 - val_loss: 19536068.0000 - val_mse:
19536068.0000
Epoch 48/100
1000/1000            2s 2ms/step -
loss: 9653122.0000 - mse: 9653122.0000 - val_loss: 13495652.0000 - val_mse:
13495652.0000
Epoch 49/100
1000/1000            3s 2ms/step -
loss: 8932151.0000 - mse: 8932151.0000 - val_loss: 16229935.0000 - val_mse:
16229935.0000
Epoch 50/100
1000/1000            2s 2ms/step -
loss: 9173082.0000 - mse: 9173082.0000 - val_loss: 27565804.0000 - val_mse:
27565804.0000
Epoch 51/100
1000/1000            2s 2ms/step -
loss: 9702385.0000 - mse: 9702385.0000 - val_loss: 12620879.0000 - val_mse:
```

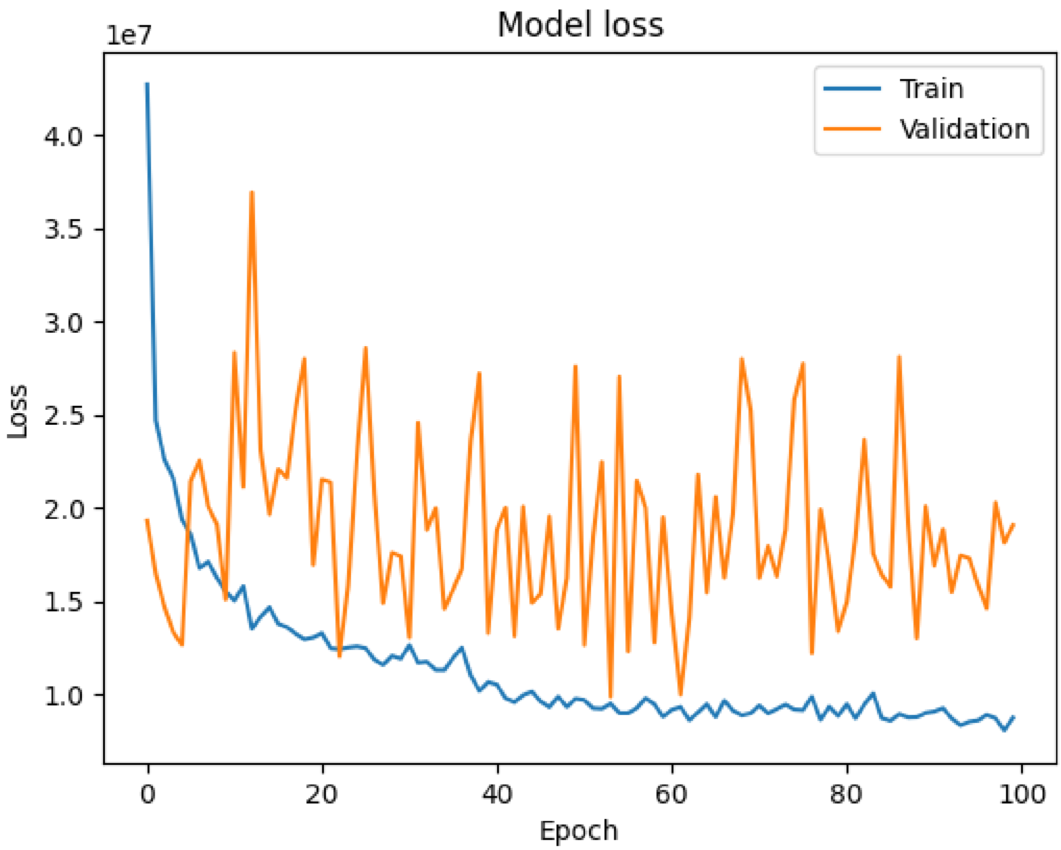
```
12620879.0000
Epoch 52/100
1000/1000            2s 2ms/step -
loss: 8837216.0000 - mse: 8837216.0000 - val_loss: 18454424.0000 - val_mse:
18454424.0000
Epoch 53/100
1000/1000            2s 2ms/step -
loss: 8903914.0000 - mse: 8903914.0000 - val_loss: 22433962.0000 - val_mse:
22433962.0000
Epoch 54/100
1000/1000            2s 2ms/step -
loss: 9572664.0000 - mse: 9572664.0000 - val_loss: 9848017.0000 - val_mse:
9848017.0000
Epoch 55/100
1000/1000            2s 2ms/step -
loss: 8568493.0000 - mse: 8568493.0000 - val_loss: 27019372.0000 - val_mse:
27019372.0000
Epoch 56/100
1000/1000            2s 2ms/step -
loss: 10173640.0000 - mse: 10173640.0000 - val_loss: 12282444.0000 - val_mse:
12282444.0000
Epoch 57/100
1000/1000            2s 2ms/step -
loss: 8739658.0000 - mse: 8739658.0000 - val_loss: 21447458.0000 - val_mse:
21447458.0000
Epoch 58/100
1000/1000            2s 2ms/step -
loss: 9194853.0000 - mse: 9194853.0000 - val_loss: 19975962.0000 - val_mse:
19975962.0000
Epoch 59/100
1000/1000            2s 2ms/step -
loss: 9361551.0000 - mse: 9361551.0000 - val_loss: 12753335.0000 - val_mse:
12753335.0000
Epoch 60/100
1000/1000            2s 2ms/step -
loss: 8247257.0000 - mse: 8247257.0000 - val_loss: 19485294.0000 - val_mse:
19485294.0000
Epoch 61/100
1000/1000            2s 2ms/step -
loss: 9025183.0000 - mse: 9025183.0000 - val_loss: 14355701.0000 - val_mse:
14355701.0000
Epoch 62/100
1000/1000            2s 2ms/step -
loss: 8765830.0000 - mse: 8765830.0000 - val_loss: 9969686.0000 - val_mse:
9969686.0000
Epoch 63/100
1000/1000            2s 2ms/step -
loss: 8254152.5000 - mse: 8254152.5000 - val_loss: 14057155.0000 - val_mse:
```

```
14057155.0000
Epoch 64/100
1000/1000            2s 2ms/step -
loss: 9114152.0000 - mse: 9114152.0000 - val_loss: 21761170.0000 - val_mse:
21761170.0000
Epoch 65/100
1000/1000            2s 2ms/step -
loss: 9877712.0000 - mse: 9877712.0000 - val_loss: 15434034.0000 - val_mse:
15434034.0000
Epoch 66/100
1000/1000            2s 2ms/step -
loss: 8983241.0000 - mse: 8983241.0000 - val_loss: 20565684.0000 - val_mse:
20565684.0000
Epoch 67/100
1000/1000            2s 2ms/step -
loss: 8865960.0000 - mse: 8865960.0000 - val_loss: 16236492.0000 - val_mse:
16236492.0000
Epoch 68/100
1000/1000            2s 2ms/step -
loss: 9759509.0000 - mse: 9759509.0000 - val_loss: 19687428.0000 - val_mse:
19687428.0000
Epoch 69/100
1000/1000            2s 2ms/step -
loss: 8369811.0000 - mse: 8369811.0000 - val_loss: 27985082.0000 - val_mse:
27985082.0000
Epoch 70/100
1000/1000            2s 2ms/step -
loss: 9155815.0000 - mse: 9155815.0000 - val_loss: 25224098.0000 - val_mse:
25224098.0000
Epoch 71/100
1000/1000            2s 2ms/step -
loss: 8564236.0000 - mse: 8564236.0000 - val_loss: 16229114.0000 - val_mse:
16229114.0000
Epoch 72/100
1000/1000            3s 2ms/step -
loss: 8812329.0000 - mse: 8812329.0000 - val_loss: 17931466.0000 - val_mse:
17931466.0000
Epoch 73/100
1000/1000            2s 2ms/step -
loss: 9219502.0000 - mse: 9219502.0000 - val_loss: 16306901.0000 - val_mse:
16306901.0000
Epoch 74/100
1000/1000            2s 2ms/step -
loss: 9310219.0000 - mse: 9310219.0000 - val_loss: 18821020.0000 - val_mse:
18821020.0000
Epoch 75/100
1000/1000            2s 2ms/step -
loss: 8507169.0000 - mse: 8507169.0000 - val_loss: 25779058.0000 - val_mse:
```

```
25779058.0000
Epoch 76/100
1000/1000            2s 2ms/step -
loss: 9576683.0000 - mse: 9576683.0000 - val_loss: 27718504.0000 - val_mse:
27718504.0000
Epoch 77/100
1000/1000            2s 2ms/step -
loss: 10418833.0000 - mse: 10418833.0000 - val_loss: 12181999.0000 - val_mse:
12181999.0000
Epoch 78/100
1000/1000            3s 2ms/step -
loss: 8885427.0000 - mse: 8885427.0000 - val_loss: 19887472.0000 - val_mse:
19887472.0000
Epoch 79/100
1000/1000            2s 2ms/step -
loss: 9124426.0000 - mse: 9124426.0000 - val_loss: 16958748.0000 - val_mse:
16958748.0000
Epoch 80/100
1000/1000            3s 2ms/step -
loss: 8848875.0000 - mse: 8848875.0000 - val_loss: 13360708.0000 - val_mse:
13360708.0000
Epoch 81/100
1000/1000            2s 2ms/step -
loss: 10270699.0000 - mse: 10270699.0000 - val_loss: 14942172.0000 - val_mse:
14942172.0000
Epoch 82/100
1000/1000            2s 2ms/step -
loss: 8886737.0000 - mse: 8886737.0000 - val_loss: 18380050.0000 - val_mse:
18380050.0000
Epoch 83/100
1000/1000            3s 3ms/step -
loss: 9443626.0000 - mse: 9443626.0000 - val_loss: 23640652.0000 - val_mse:
23640652.0000
Epoch 84/100
1000/1000            4s 2ms/step -
loss: 9318567.0000 - mse: 9318567.0000 - val_loss: 17526598.0000 - val_mse:
17526598.0000
Epoch 85/100
1000/1000            2s 2ms/step -
loss: 8775955.0000 - mse: 8775955.0000 - val_loss: 16388337.0000 - val_mse:
16388337.0000
Epoch 86/100
1000/1000            2s 2ms/step -
loss: 8590094.0000 - mse: 8590094.0000 - val_loss: 15750921.0000 - val_mse:
15750921.0000
Epoch 87/100
1000/1000            2s 2ms/step -
loss: 8684017.0000 - mse: 8684017.0000 - val_loss: 28092162.0000 - val_mse:
```

28092162.0000
Epoch 88/100
**1000/1000** **2s** 2ms/step -
loss: 9034208.0000 - mse: 9034208.0000 - val_loss: 19164404.0000 - val_mse:
19164404.0000
Epoch 89/100
**1000/1000** **2s** 2ms/step -
loss: 8346559.5000 - mse: 8346559.5000 - val_loss: 12997876.0000 - val_mse:
12997876.0000
Epoch 90/100
**1000/1000** **2s** 2ms/step -
loss: 9138750.0000 - mse: 9138750.0000 - val_loss: 20081474.0000 - val_mse:
20081474.0000
Epoch 91/100
**1000/1000** **3s** 2ms/step -
loss: 9361158.0000 - mse: 9361158.0000 - val_loss: 16887310.0000 - val_mse:
16887310.0000
Epoch 92/100
**1000/1000** **2s** 2ms/step -
loss: 8740946.0000 - mse: 8740946.0000 - val_loss: 18842112.0000 - val_mse:
18842112.0000
Epoch 93/100
**1000/1000** **2s** 2ms/step -
loss: 9040713.0000 - mse: 9040713.0000 - val_loss: 15454607.0000 - val_mse:
15454607.0000
Epoch 94/100
**1000/1000** **2s** 2ms/step -
loss: 8659189.0000 - mse: 8659189.0000 - val_loss: 17442030.0000 - val_mse:
17442030.0000
Epoch 95/100
**1000/1000** **2s** 2ms/step -
loss: 8371733.5000 - mse: 8371733.5000 - val_loss: 17269622.0000 - val_mse:
17269622.0000
Epoch 96/100
**1000/1000** **2s** 2ms/step -
loss: 8132848.0000 - mse: 8132848.0000 - val_loss: 15841168.0000 - val_mse:
15841168.0000
Epoch 97/100
**1000/1000** **2s** 2ms/step -
loss: 8749248.0000 - mse: 8749248.0000 - val_loss: 14584912.0000 - val_mse:
14584912.0000
Epoch 98/100
**1000/1000** **3s** 2ms/step -
loss: 8786307.0000 - mse: 8786307.0000 - val_loss: 20272144.0000 - val_mse:
20272144.0000
Epoch 99/100
**1000/1000** **2s** 2ms/step -
loss: 7957913.5000 - mse: 7957913.5000 - val_loss: 18109006.0000 - val_mse:

```
18109006.0000
Epoch 100/100
1000/1000            2s 2ms/step -
loss: 8734131.0000 - mse: 8734131.0000 - val_loss: 19055816.0000 - val_mse:
19055816.0000
313/313            0s 1ms/step
Artificial Neural Network Model:
Mean Squared Error: 19503856.4214387
R-squared: 0.9281295537948608
```




## 3 ANN with Learning Rate of 0.01

[3]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
```

```
from tensorflow.keras.optimizers import Adam

# Load the dataset
file_path = 'car_sales_data_24.csv'
car_sales_data = pd.read_csv(file_path)

# Selecting features
numerical_features = ['Engine size', 'Year of manufacture', 'Mileage']
categorical_features = ['Manufacturer', 'Model', 'Fuel type']

# Prepare the dataset
X = car_sales_data[numerical_features + categorical_features]
y = car_sales_data['Price']

# Encode categorical variables using LabelEncoder
label_encoders = {}
for feature in categorical_features:
    label_encoders[feature] = LabelEncoder()
    X[feature] = label_encoders[feature].fit_transform(X[feature])

# Scale numerical features
scaler = StandardScaler()
X[numerical_features] = scaler.fit_transform(X[numerical_features])

# 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)

# Define the architecture of the neural network
def create_model(learning_rate=0.01, dropout_rate=0.2):
    model = Sequential()
    model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(loss='mean_squared_error', optimizer=optimizer,␣
↪metrics=['mse'])
    return model

# Create the model
model = create_model()

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32,␣
↪validation_split=0.2, verbose=1)
```

```python
# Predict on the test set
y_pred = model.predict(X_test).flatten()

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Artificial Neural Network Model:")
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Plot training & validation loss values
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```

```
Epoch 1/100
```

```
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\2825578503.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\2825578503.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\2825578503.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\2825578503.py:30:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[numerical_features] = scaler.fit_transform(X[numerical_features])
C:\Users\sleek\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

1000/1000            3s 2ms/step -
loss: 179778720.0000 - mse: 179778720.0000 - val_loss: 26264268.0000 - val_mse:
26264268.0000
Epoch 2/100
1000/1000            2s 2ms/step -
loss: 30994282.0000 - mse: 30994282.0000 - val_loss: 21996138.0000 - val_mse:
21996138.0000
Epoch 3/100
1000/1000            2s 1ms/step -
loss: 27684426.0000 - mse: 27684426.0000 - val_loss: 20018782.0000 - val_mse:
20018782.0000
Epoch 4/100
1000/1000            2s 1ms/step -
loss: 23651486.0000 - mse: 23651486.0000 - val_loss: 20785324.0000 - val_mse:
20785324.0000
Epoch 5/100
1000/1000            1s 1ms/step -
loss: 23497860.0000 - mse: 23497860.0000 - val_loss: 17479292.0000 - val_mse:
17479292.0000
Epoch 6/100
1000/1000            2s 2ms/step -
loss: 22355170.0000 - mse: 22355170.0000 - val_loss: 17534392.0000 - val_mse:
17534392.0000
Epoch 7/100
1000/1000            2s 2ms/step -
loss: 21825516.0000 - mse: 21825516.0000 - val_loss: 16680743.0000 - val_mse:
16680743.0000
Epoch 8/100
1000/1000            2s 2ms/step -
loss: 20598142.0000 - mse: 20598142.0000 - val_loss: 15559170.0000 - val_mse:
15559170.0000
Epoch 9/100
1000/1000            2s 2ms/step -
```

```
loss: 20425058.0000 - mse: 20425058.0000 - val_loss: 15478845.0000 - val_mse:
15478845.0000
Epoch 10/100
1000/1000            2s 1ms/step -
loss: 20452468.0000 - mse: 20452468.0000 - val_loss: 15304662.0000 - val_mse:
15304662.0000
Epoch 11/100
1000/1000            2s 2ms/step -
loss: 21142774.0000 - mse: 21142774.0000 - val_loss: 15415751.0000 - val_mse:
15415751.0000
Epoch 12/100
1000/1000            3s 2ms/step -
loss: 19917976.0000 - mse: 19917976.0000 - val_loss: 15502568.0000 - val_mse:
15502568.0000
Epoch 13/100
1000/1000            3s 2ms/step -
loss: 20679968.0000 - mse: 20679968.0000 - val_loss: 14854252.0000 - val_mse:
14854252.0000
Epoch 14/100
1000/1000            2s 2ms/step -
loss: 19927714.0000 - mse: 19927714.0000 - val_loss: 14933572.0000 - val_mse:
14933572.0000
Epoch 15/100
1000/1000            4s 3ms/step -
loss: 19483492.0000 - mse: 19483492.0000 - val_loss: 14627375.0000 - val_mse:
14627375.0000
Epoch 16/100
1000/1000            5s 2ms/step -
loss: 19313598.0000 - mse: 19313598.0000 - val_loss: 14862037.0000 - val_mse:
14862037.0000
Epoch 17/100
1000/1000            2s 2ms/step -
loss: 19216414.0000 - mse: 19216414.0000 - val_loss: 14491583.0000 - val_mse:
14491583.0000
Epoch 18/100
1000/1000            2s 2ms/step -
loss: 17964260.0000 - mse: 17964260.0000 - val_loss: 14138145.0000 - val_mse:
14138145.0000
Epoch 19/100
1000/1000            2s 2ms/step -
loss: 18873064.0000 - mse: 18873064.0000 - val_loss: 13690423.0000 - val_mse:
13690423.0000
Epoch 20/100
1000/1000            2s 2ms/step -
loss: 18361990.0000 - mse: 18361990.0000 - val_loss: 13068014.0000 - val_mse:
13068014.0000
Epoch 21/100
1000/1000            2s 2ms/step -
```

```
loss: 17699300.0000 - mse: 17699300.0000 - val_loss: 13628421.0000 - val_mse:
13628421.0000
Epoch 22/100
1000/1000            1s 1ms/step -
loss: 17166182.0000 - mse: 17166182.0000 - val_loss: 12447859.0000 - val_mse:
12447859.0000
Epoch 23/100
1000/1000            2s 2ms/step -
loss: 18365782.0000 - mse: 18365782.0000 - val_loss: 12997153.0000 - val_mse:
12997153.0000
Epoch 24/100
1000/1000            2s 1ms/step -
loss: 16782014.0000 - mse: 16782014.0000 - val_loss: 12880822.0000 - val_mse:
12880822.0000
Epoch 25/100
1000/1000            3s 2ms/step -
loss: 16701483.0000 - mse: 16701483.0000 - val_loss: 11822283.0000 - val_mse:
11822283.0000
Epoch 26/100
1000/1000            1s 1ms/step -
loss: 16875346.0000 - mse: 16875346.0000 - val_loss: 12003215.0000 - val_mse:
12003215.0000
Epoch 27/100
1000/1000            3s 2ms/step -
loss: 16227864.0000 - mse: 16227864.0000 - val_loss: 10966331.0000 - val_mse:
10966331.0000
Epoch 28/100
1000/1000            2s 1ms/step -
loss: 15827065.0000 - mse: 15827065.0000 - val_loss: 10245960.0000 - val_mse:
10245960.0000
Epoch 29/100
1000/1000            2s 2ms/step -
loss: 15061681.0000 - mse: 15061681.0000 - val_loss: 9999828.0000 - val_mse:
9999828.0000
Epoch 30/100
1000/1000            2s 2ms/step -
loss: 14487789.0000 - mse: 14487789.0000 - val_loss: 9967198.0000 - val_mse:
9967198.0000
Epoch 31/100
1000/1000            2s 2ms/step -
loss: 14828514.0000 - mse: 14828514.0000 - val_loss: 9559266.0000 - val_mse:
9559266.0000
Epoch 32/100
1000/1000            1s 1ms/step -
loss: 15336488.0000 - mse: 15336488.0000 - val_loss: 9126681.0000 - val_mse:
9126681.0000
Epoch 33/100
1000/1000            2s 2ms/step -
```

```
loss: 13772110.0000 - mse: 13772110.0000 - val_loss: 8452438.0000 - val_mse:
8452438.0000
Epoch 34/100
1000/1000            2s 2ms/step -
loss: 13530876.0000 - mse: 13530876.0000 - val_loss: 9000362.0000 - val_mse:
9000362.0000
Epoch 35/100
1000/1000            3s 2ms/step -
loss: 13649129.0000 - mse: 13649129.0000 - val_loss: 8368111.5000 - val_mse:
8368111.5000
Epoch 36/100
1000/1000            2s 2ms/step -
loss: 14013445.0000 - mse: 14013445.0000 - val_loss: 8329083.0000 - val_mse:
8329083.0000
Epoch 37/100
1000/1000            2s 2ms/step -
loss: 14240042.0000 - mse: 14240042.0000 - val_loss: 7931983.5000 - val_mse:
7931983.5000
Epoch 38/100
1000/1000            2s 2ms/step -
loss: 12780744.0000 - mse: 12780744.0000 - val_loss: 9346367.0000 - val_mse:
9346367.0000
Epoch 39/100
1000/1000            2s 2ms/step -
loss: 12101047.0000 - mse: 12101047.0000 - val_loss: 7121877.0000 - val_mse:
7121877.0000
Epoch 40/100
1000/1000            2s 2ms/step -
loss: 12868459.0000 - mse: 12868459.0000 - val_loss: 7308175.5000 - val_mse:
7308175.5000
Epoch 41/100
1000/1000            3s 2ms/step -
loss: 12760709.0000 - mse: 12760709.0000 - val_loss: 7434563.0000 - val_mse:
7434563.0000
Epoch 42/100
1000/1000            2s 2ms/step -
loss: 12197641.0000 - mse: 12197641.0000 - val_loss: 6384237.0000 - val_mse:
6384237.0000
Epoch 43/100
1000/1000            2s 2ms/step -
loss: 12043656.0000 - mse: 12043656.0000 - val_loss: 5986965.5000 - val_mse:
5986965.5000
Epoch 44/100
1000/1000            2s 2ms/step -
loss: 11137700.0000 - mse: 11137700.0000 - val_loss: 5666343.5000 - val_mse:
5666343.5000
Epoch 45/100
1000/1000            2s 2ms/step -
```

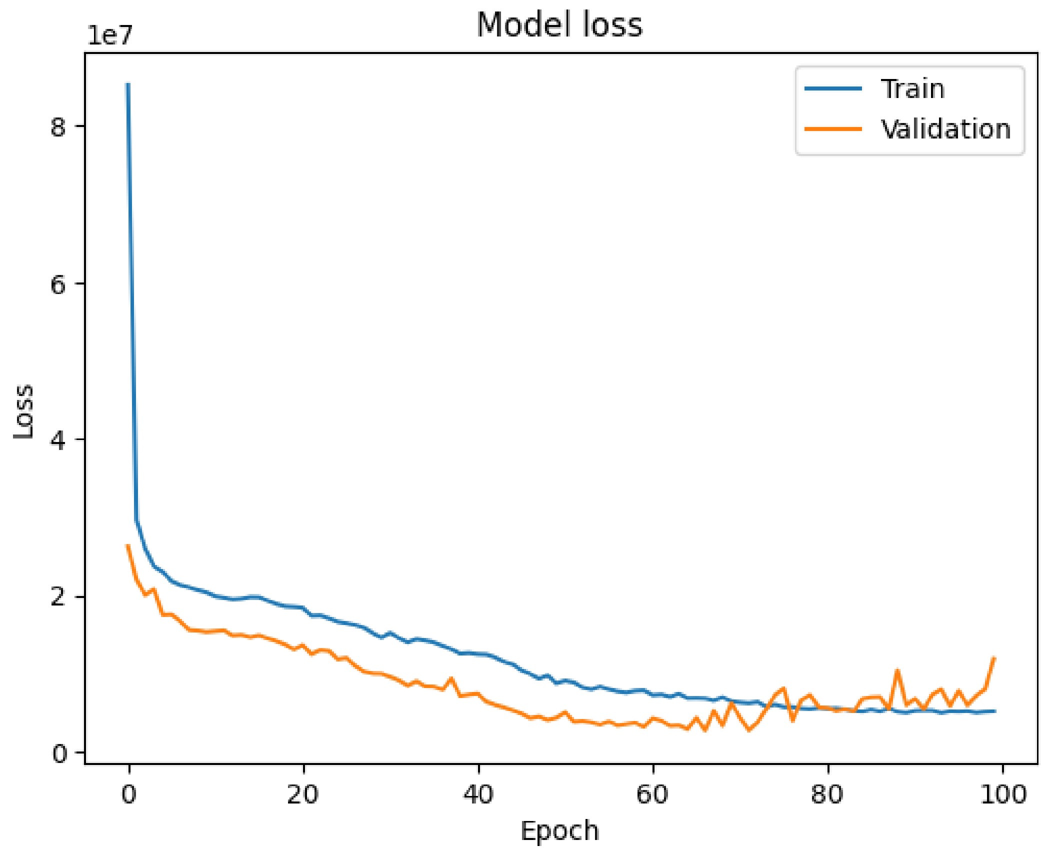
```
loss: 10568357.0000 - mse: 10568357.0000 - val_loss: 5287062.5000 - val_mse:
5287062.5000
Epoch 46/100
1000/1000            3s 2ms/step -
loss: 11304453.0000 - mse: 11304453.0000 - val_loss: 4886361.5000 - val_mse:
4886361.5000
Epoch 47/100
1000/1000            2s 2ms/step -
loss: 10528248.0000 - mse: 10528248.0000 - val_loss: 4285746.5000 - val_mse:
4285746.5000
Epoch 48/100
1000/1000            2s 2ms/step -
loss: 9183731.0000 - mse: 9183731.0000 - val_loss: 4502272.5000 - val_mse:
4502272.5000
Epoch 49/100
1000/1000            2s 2ms/step -
loss: 9376300.0000 - mse: 9376300.0000 - val_loss: 4069047.0000 - val_mse:
4069047.0000
Epoch 50/100
1000/1000            2s 2ms/step -
loss: 8950359.0000 - mse: 8950359.0000 - val_loss: 4337984.0000 - val_mse:
4337984.0000
Epoch 51/100
1000/1000            2s 2ms/step -
loss: 8632782.0000 - mse: 8632782.0000 - val_loss: 5036972.5000 - val_mse:
5036972.5000
Epoch 52/100
1000/1000            3s 2ms/step -
loss: 9433242.0000 - mse: 9433242.0000 - val_loss: 3837807.5000 - val_mse:
3837807.5000
Epoch 53/100
1000/1000            2s 2ms/step -
loss: 8160992.5000 - mse: 8160992.5000 - val_loss: 3926376.5000 - val_mse:
3926376.5000
Epoch 54/100
1000/1000            3s 2ms/step -
loss: 8083785.0000 - mse: 8083785.0000 - val_loss: 3752309.5000 - val_mse:
3752309.5000
Epoch 55/100
1000/1000            2s 2ms/step -
loss: 8015604.0000 - mse: 8015604.0000 - val_loss: 3460813.7500 - val_mse:
3460813.7500
Epoch 56/100
1000/1000            2s 2ms/step -
loss: 8247943.0000 - mse: 8247943.0000 - val_loss: 3857834.2500 - val_mse:
3857834.2500
Epoch 57/100
1000/1000            2s 2ms/step -
```

```
loss: 7220971.5000 - mse: 7220971.5000 - val_loss: 3360703.0000 - val_mse:
3360703.0000
Epoch 58/100
1000/1000            2s 2ms/step -
loss: 7523807.5000 - mse: 7523807.5000 - val_loss: 3540180.2500 - val_mse:
3540180.2500
Epoch 59/100
1000/1000            1s 1ms/step -
loss: 7702352.5000 - mse: 7702352.5000 - val_loss: 3704540.7500 - val_mse:
3704540.7500
Epoch 60/100
1000/1000            2s 2ms/step -
loss: 7993979.0000 - mse: 7993979.0000 - val_loss: 3184316.2500 - val_mse:
3184316.2500
Epoch 61/100
1000/1000            2s 2ms/step -
loss: 7214015.0000 - mse: 7214015.0000 - val_loss: 4281074.5000 - val_mse:
4281074.5000
Epoch 62/100
1000/1000            2s 2ms/step -
loss: 7073404.5000 - mse: 7073404.5000 - val_loss: 3955353.7500 - val_mse:
3955353.7500
Epoch 63/100
1000/1000            2s 2ms/step -
loss: 7370803.5000 - mse: 7370803.5000 - val_loss: 3323260.2500 - val_mse:
3323260.2500
Epoch 64/100
1000/1000            1s 1ms/step -
loss: 7028387.5000 - mse: 7028387.5000 - val_loss: 3381487.7500 - val_mse:
3381487.7500
Epoch 65/100
1000/1000            2s 2ms/step -
loss: 6587038.0000 - mse: 6587038.0000 - val_loss: 2883073.2500 - val_mse:
2883073.2500
Epoch 66/100
1000/1000            1s 1ms/step -
loss: 7000428.5000 - mse: 7000428.5000 - val_loss: 4331236.0000 - val_mse:
4331236.0000
Epoch 67/100
1000/1000            1s 1ms/step -
loss: 7059008.0000 - mse: 7059008.0000 - val_loss: 2737761.2500 - val_mse:
2737761.2500
Epoch 68/100
1000/1000            1s 1ms/step -
loss: 6878546.0000 - mse: 6878546.0000 - val_loss: 5232975.0000 - val_mse:
5232975.0000
Epoch 69/100
1000/1000            1s 1ms/step -
```

```
loss: 6686068.5000 - mse: 6686068.5000 - val_loss: 3306261.0000 - val_mse:
3306261.0000
Epoch 70/100
1000/1000            1s 1ms/step -
loss: 6317791.5000 - mse: 6317791.5000 - val_loss: 6207928.0000 - val_mse:
6207928.0000
Epoch 71/100
1000/1000            1s 1ms/step -
loss: 5969327.0000 - mse: 5969327.0000 - val_loss: 4338555.0000 - val_mse:
4338555.0000
Epoch 72/100
1000/1000            2s 2ms/step -
loss: 6121393.5000 - mse: 6121393.5000 - val_loss: 2732135.0000 - val_mse:
2732135.0000
Epoch 73/100
1000/1000            2s 2ms/step -
loss: 6375708.0000 - mse: 6375708.0000 - val_loss: 3769248.2500 - val_mse:
3769248.2500
Epoch 74/100
1000/1000            1s 1ms/step -
loss: 5602604.5000 - mse: 5602604.5000 - val_loss: 5450787.0000 - val_mse:
5450787.0000
Epoch 75/100
1000/1000            1s 1ms/step -
loss: 5763885.0000 - mse: 5763885.0000 - val_loss: 7230762.5000 - val_mse:
7230762.5000
Epoch 76/100
1000/1000            1s 1ms/step -
loss: 5921578.0000 - mse: 5921578.0000 - val_loss: 8120778.0000 - val_mse:
8120778.0000
Epoch 77/100
1000/1000            1s 1ms/step -
loss: 5603202.5000 - mse: 5603202.5000 - val_loss: 3978172.0000 - val_mse:
3978172.0000
Epoch 78/100
1000/1000            1s 1ms/step -
loss: 5456637.0000 - mse: 5456637.0000 - val_loss: 6584964.5000 - val_mse:
6584964.5000
Epoch 79/100
1000/1000            2s 2ms/step -
loss: 5470249.5000 - mse: 5470249.5000 - val_loss: 7254726.5000 - val_mse:
7254726.5000
Epoch 80/100
1000/1000            4s 4ms/step -
loss: 5591359.5000 - mse: 5591359.5000 - val_loss: 5677412.0000 - val_mse:
5677412.0000
Epoch 81/100
1000/1000            3s 3ms/step -
```

```
loss: 5672366.0000 - mse: 5672366.0000 - val_loss: 5587182.5000 - val_mse:
5587182.5000
Epoch 82/100
1000/1000             2s 2ms/step -
loss: 5246860.5000 - mse: 5246860.5000 - val_loss: 5201082.0000 - val_mse:
5201082.0000
Epoch 83/100
1000/1000             3s 2ms/step -
loss: 5738573.0000 - mse: 5738573.0000 - val_loss: 5443015.5000 - val_mse:
5443015.5000
Epoch 84/100
1000/1000             2s 2ms/step -
loss: 5075452.5000 - mse: 5075452.5000 - val_loss: 5168927.0000 - val_mse:
5168927.0000
Epoch 85/100
1000/1000             2s 2ms/step -
loss: 5236936.5000 - mse: 5236936.5000 - val_loss: 6765882.0000 - val_mse:
6765882.0000
Epoch 86/100
1000/1000             2s 2ms/step -
loss: 5641854.5000 - mse: 5641854.5000 - val_loss: 6949413.5000 - val_mse:
6949413.5000
Epoch 87/100
1000/1000             2s 2ms/step -
loss: 5121403.0000 - mse: 5121403.0000 - val_loss: 6989947.5000 - val_mse:
6989947.5000
Epoch 88/100
1000/1000             2s 2ms/step -
loss: 5438167.5000 - mse: 5438167.5000 - val_loss: 5466829.5000 - val_mse:
5466829.5000
Epoch 89/100
1000/1000             2s 2ms/step -
loss: 4971095.5000 - mse: 4971095.5000 - val_loss: 10409827.0000 - val_mse:
10409827.0000
Epoch 90/100
1000/1000             2s 2ms/step -
loss: 4895359.5000 - mse: 4895359.5000 - val_loss: 5954711.5000 - val_mse:
5954711.5000
Epoch 91/100
1000/1000             2s 2ms/step -
loss: 5429511.0000 - mse: 5429511.0000 - val_loss: 6779120.0000 - val_mse:
6779120.0000
Epoch 92/100
1000/1000             2s 2ms/step -
loss: 5170624.5000 - mse: 5170624.5000 - val_loss: 5457297.5000 - val_mse:
5457297.5000
Epoch 93/100
1000/1000             3s 2ms/step -
```

```
loss: 5243424.0000 - mse: 5243424.0000 - val_loss: 7309073.0000 - val_mse:
7309073.0000
Epoch 94/100
1000/1000            2s 2ms/step -
loss: 4839745.5000 - mse: 4839745.5000 - val_loss: 7984095.5000 - val_mse:
7984095.5000
Epoch 95/100
1000/1000            2s 2ms/step -
loss: 5166984.5000 - mse: 5166984.5000 - val_loss: 5859730.0000 - val_mse:
5859730.0000
Epoch 96/100
1000/1000            2s 2ms/step -
loss: 5174641.5000 - mse: 5174641.5000 - val_loss: 7765691.5000 - val_mse:
7765691.5000
Epoch 97/100
1000/1000            2s 2ms/step -
loss: 4874630.0000 - mse: 4874630.0000 - val_loss: 5939794.5000 - val_mse:
5939794.5000
Epoch 98/100
1000/1000            2s 2ms/step -
loss: 5019232.5000 - mse: 5019232.5000 - val_loss: 7172404.0000 - val_mse:
7172404.0000
Epoch 99/100
1000/1000            2s 2ms/step -
loss: 4980895.5000 - mse: 4980895.5000 - val_loss: 7984255.5000 - val_mse:
7984255.5000
Epoch 100/100
1000/1000            2s 2ms/step -
loss: 5408053.5000 - mse: 5408053.5000 - val_loss: 11881273.0000 - val_mse:
11881273.0000
313/313            0s 1ms/step
Artificial Neural Network Model:
Mean Squared Error: 12126536.54198929
R-squared: 0.9553145170211792
```




# 4 ANN with Learning rate 0.001

[10]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Load the dataset
file_path = 'car_sales_data_24.csv'
car_sales_data = pd.read_csv(file_path)

# Selecting features
numerical_features = ['Engine size', 'Year of manufacture', 'Mileage']
categorical_features = ['Manufacturer', 'Model', 'Fuel type']
```

```python
# Prepare the dataset
X = car_sales_data[numerical_features + categorical_features]
y = car_sales_data['Price']

# Encode categorical variables using LabelEncoder
label_encoders = {}
for feature in categorical_features:
    label_encoders[feature] = LabelEncoder()
    X[feature] = label_encoders[feature].fit_transform(X[feature])

# Scale numerical features
scaler = StandardScaler()
X[numerical_features] = scaler.fit_transform(X[numerical_features])

# 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)

# Define the architecture of the neural network
def create_model(learning_rate=0.001, dropout_rate=0.2):
    model = Sequential()
    model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(loss='mean_squared_error', optimizer=optimizer, metrics=['mse'])
    return model

# Create the model
model = create_model()

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_split=0.2, verbose=1)

# Predict on the test set
y_pred = model.predict(X_test).flatten()

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Artificial Neural Network Model:")
print(f"Mean Squared Error: {mse}")
```

```python
print(f"R-squared: {r2}")

# Plot training & validation loss values
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```

```
C:\Users\sleek\AppData\Local\Temp\ipykernel_11692\3202584909.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_11692\3202584909.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_11692\3202584909.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_11692\3202584909.py:30:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[numerical_features] = scaler.fit_transform(X[numerical_features])
C:\Users\sleek\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
```

```
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/100
1000/1000            3s 2ms/step -
loss: 405181248.0000 - mse: 405181248.0000 - val_loss: 189311632.0000 - val_mse:
189311632.0000
Epoch 2/100
1000/1000            1s 1ms/step -
loss: 165269728.0000 - mse: 165269728.0000 - val_loss: 73940192.0000 - val_mse:
73940192.0000
Epoch 3/100
1000/1000            1s 1ms/step -
loss: 71540152.0000 - mse: 71540152.0000 - val_loss: 44747792.0000 - val_mse:
44747792.0000
Epoch 4/100
1000/1000            1s 1ms/step -
loss: 47902796.0000 - mse: 47902796.0000 - val_loss: 36838336.0000 - val_mse:
36838336.0000
Epoch 5/100
1000/1000            1s 1ms/step -
loss: 41800536.0000 - mse: 41800536.0000 - val_loss: 33288996.0000 - val_mse:
33288996.0000
Epoch 6/100
1000/1000            1s 1ms/step -
loss: 37093288.0000 - mse: 37093288.0000 - val_loss: 31428334.0000 - val_mse:
31428334.0000
Epoch 7/100
1000/1000            1s 1ms/step -
loss: 35492076.0000 - mse: 35492076.0000 - val_loss: 30141170.0000 - val_mse:
30141170.0000
Epoch 8/100
1000/1000            1s 1ms/step -
loss: 35135848.0000 - mse: 35135848.0000 - val_loss: 29226932.0000 - val_mse:
29226932.0000
Epoch 9/100
1000/1000            1s 1ms/step -
loss: 34647964.0000 - mse: 34647964.0000 - val_loss: 28360142.0000 - val_mse:
28360142.0000
Epoch 10/100
1000/1000            1s 1ms/step -
loss: 33532724.0000 - mse: 33532724.0000 - val_loss: 27341082.0000 - val_mse:
27341082.0000
Epoch 11/100
1000/1000            1s 1ms/step -
loss: 33279254.0000 - mse: 33279254.0000 - val_loss: 26613022.0000 - val_mse:
26613022.0000
```

```
Epoch 12/100
1000/1000           1s 1ms/step -
loss: 30638848.0000 - mse: 30638848.0000 - val_loss: 25991888.0000 - val_mse:
25991888.0000
Epoch 13/100
1000/1000           1s 1ms/step -
loss: 30800308.0000 - mse: 30800308.0000 - val_loss: 25507582.0000 - val_mse:
25507582.0000
Epoch 14/100
1000/1000           1s 1ms/step -
loss: 30555166.0000 - mse: 30555166.0000 - val_loss: 25056594.0000 - val_mse:
25056594.0000
Epoch 15/100
1000/1000           1s 1ms/step -
loss: 29410044.0000 - mse: 29410044.0000 - val_loss: 24479118.0000 - val_mse:
24479118.0000
Epoch 16/100
1000/1000           1s 1ms/step -
loss: 28583482.0000 - mse: 28583482.0000 - val_loss: 24026562.0000 - val_mse:
24026562.0000
Epoch 17/100
1000/1000           1s 1ms/step -
loss: 26838354.0000 - mse: 26838354.0000 - val_loss: 23693914.0000 - val_mse:
23693914.0000
Epoch 18/100
1000/1000           1s 1ms/step -
loss: 26967640.0000 - mse: 26967640.0000 - val_loss: 23428694.0000 - val_mse:
23428694.0000
Epoch 19/100
1000/1000           1s 1ms/step -
loss: 28094100.0000 - mse: 28094100.0000 - val_loss: 22997020.0000 - val_mse:
22997020.0000
Epoch 20/100
1000/1000           1s 1ms/step -
loss: 26735484.0000 - mse: 26735484.0000 - val_loss: 22750500.0000 - val_mse:
22750500.0000
Epoch 21/100
1000/1000           1s 1ms/step -
loss: 27073340.0000 - mse: 27073340.0000 - val_loss: 22384662.0000 - val_mse:
22384662.0000
Epoch 22/100
1000/1000           1s 1ms/step -
loss: 26764780.0000 - mse: 26764780.0000 - val_loss: 22132482.0000 - val_mse:
22132482.0000
Epoch 23/100
1000/1000           1s 1ms/step -
loss: 25815928.0000 - mse: 25815928.0000 - val_loss: 21878694.0000 - val_mse:
21878694.0000
```

```
Epoch 24/100
1000/1000            1s 1ms/step -
loss: 26571166.0000 - mse: 26571166.0000 - val_loss: 21536324.0000 - val_mse:
21536324.0000
Epoch 25/100
1000/1000            1s 1ms/step -
loss: 26012012.0000 - mse: 26012012.0000 - val_loss: 21248786.0000 - val_mse:
21248786.0000
Epoch 26/100
1000/1000            1s 1ms/step -
loss: 26175318.0000 - mse: 26175318.0000 - val_loss: 21012602.0000 - val_mse:
21012602.0000
Epoch 27/100
1000/1000            1s 1ms/step -
loss: 25712768.0000 - mse: 25712768.0000 - val_loss: 20795128.0000 - val_mse:
20795128.0000
Epoch 28/100
1000/1000            2s 2ms/step -
loss: 24746324.0000 - mse: 24746324.0000 - val_loss: 20502062.0000 - val_mse:
20502062.0000
Epoch 29/100
1000/1000            1s 1ms/step -
loss: 24535618.0000 - mse: 24535618.0000 - val_loss: 20258146.0000 - val_mse:
20258146.0000
Epoch 30/100
1000/1000            1s 1ms/step -
loss: 23660612.0000 - mse: 23660612.0000 - val_loss: 20047106.0000 - val_mse:
20047106.0000
Epoch 31/100
1000/1000            1s 1ms/step -
loss: 25031172.0000 - mse: 25031172.0000 - val_loss: 19851626.0000 - val_mse:
19851626.0000
Epoch 32/100
1000/1000            1s 1ms/step -
loss: 23014522.0000 - mse: 23014522.0000 - val_loss: 19652342.0000 - val_mse:
19652342.0000
Epoch 33/100
1000/1000            3s 1ms/step -
loss: 23019880.0000 - mse: 23019880.0000 - val_loss: 19493766.0000 - val_mse:
19493766.0000
Epoch 34/100
1000/1000            1s 1ms/step -
loss: 24317134.0000 - mse: 24317134.0000 - val_loss: 19323462.0000 - val_mse:
19323462.0000
Epoch 35/100
1000/1000            1s 1ms/step -
loss: 24906834.0000 - mse: 24906834.0000 - val_loss: 19165250.0000 - val_mse:
19165250.0000
```

```
Epoch 36/100
1000/1000            1s 1ms/step -
loss: 23003868.0000 - mse: 23003868.0000 - val_loss: 19020682.0000 - val_mse:
19020682.0000
Epoch 37/100
1000/1000            1s 1ms/step -
loss: 23182882.0000 - mse: 23182882.0000 - val_loss: 18856508.0000 - val_mse:
18856508.0000
Epoch 38/100
1000/1000            1s 1ms/step -
loss: 23402208.0000 - mse: 23402208.0000 - val_loss: 18788330.0000 - val_mse:
18788330.0000
Epoch 39/100
1000/1000            3s 1ms/step -
loss: 22662050.0000 - mse: 22662050.0000 - val_loss: 18639790.0000 - val_mse:
18639790.0000
Epoch 40/100
1000/1000            1s 1ms/step -
loss: 22769280.0000 - mse: 22769280.0000 - val_loss: 18520210.0000 - val_mse:
18520210.0000
Epoch 41/100
1000/1000            1s 1ms/step -
loss: 22279908.0000 - mse: 22279908.0000 - val_loss: 18435216.0000 - val_mse:
18435216.0000
Epoch 42/100
1000/1000            1s 1ms/step -
loss: 22506608.0000 - mse: 22506608.0000 - val_loss: 18346780.0000 - val_mse:
18346780.0000
Epoch 43/100
1000/1000            1s 1ms/step -
loss: 21824942.0000 - mse: 21824942.0000 - val_loss: 18256478.0000 - val_mse:
18256478.0000
Epoch 44/100
1000/1000            1s 1ms/step -
loss: 22172012.0000 - mse: 22172012.0000 - val_loss: 18192866.0000 - val_mse:
18192866.0000
Epoch 45/100
1000/1000            1s 1ms/step -
loss: 21479324.0000 - mse: 21479324.0000 - val_loss: 18078384.0000 - val_mse:
18078384.0000
Epoch 46/100
1000/1000            1s 1ms/step -
loss: 21441384.0000 - mse: 21441384.0000 - val_loss: 17965008.0000 - val_mse:
17965008.0000
Epoch 47/100
1000/1000            1s 1ms/step -
loss: 19973974.0000 - mse: 19973974.0000 - val_loss: 17893392.0000 - val_mse:
17893392.0000
```

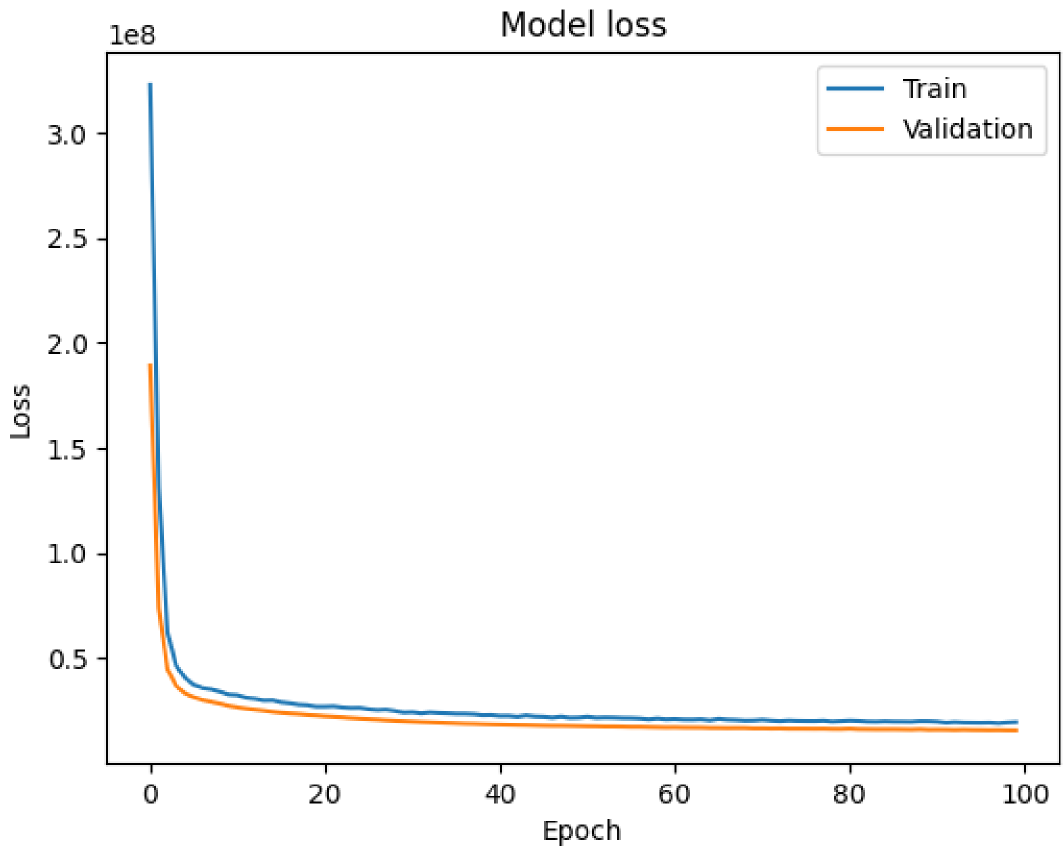
```
Epoch 48/100
1000/1000            1s 1ms/step -
loss: 23554046.0000 - mse: 23554046.0000 - val_loss: 17876882.0000 - val_mse:
17876882.0000
Epoch 49/100
1000/1000            2s 2ms/step -
loss: 22140138.0000 - mse: 22140138.0000 - val_loss: 17811702.0000 - val_mse:
17811702.0000
Epoch 50/100
1000/1000            2s 2ms/step -
loss: 21238756.0000 - mse: 21238756.0000 - val_loss: 17732324.0000 - val_mse:
17732324.0000
Epoch 51/100
1000/1000            1s 1ms/step -
loss: 21949100.0000 - mse: 21949100.0000 - val_loss: 17641438.0000 - val_mse:
17641438.0000
Epoch 52/100
1000/1000            1s 1ms/step -
loss: 21527284.0000 - mse: 21527284.0000 - val_loss: 17596558.0000 - val_mse:
17596558.0000
Epoch 53/100
1000/1000            1s 1ms/step -
loss: 21537488.0000 - mse: 21537488.0000 - val_loss: 17487856.0000 - val_mse:
17487856.0000
Epoch 54/100
1000/1000            1s 1ms/step -
loss: 23674750.0000 - mse: 23674750.0000 - val_loss: 17484144.0000 - val_mse:
17484144.0000
Epoch 55/100
1000/1000            3s 2ms/step -
loss: 21292518.0000 - mse: 21292518.0000 - val_loss: 17424558.0000 - val_mse:
17424558.0000
Epoch 56/100
1000/1000            2s 1ms/step -
loss: 22200922.0000 - mse: 22200922.0000 - val_loss: 17329098.0000 - val_mse:
17329098.0000
Epoch 57/100
1000/1000            3s 1ms/step -
loss: 21058794.0000 - mse: 21058794.0000 - val_loss: 17333082.0000 - val_mse:
17333082.0000
Epoch 58/100
1000/1000            3s 1ms/step -
loss: 21475714.0000 - mse: 21475714.0000 - val_loss: 17227726.0000 - val_mse:
17227726.0000
Epoch 59/100
1000/1000            1s 1ms/step -
loss: 20620654.0000 - mse: 20620654.0000 - val_loss: 17141632.0000 - val_mse:
17141632.0000
```

```
Epoch 60/100
1000/1000           1s 1ms/step -
loss: 20758388.0000 - mse: 20758388.0000 - val_loss: 17058210.0000 - val_mse:
17058210.0000
Epoch 61/100
1000/1000           1s 1ms/step -
loss: 21230986.0000 - mse: 21230986.0000 - val_loss: 17091978.0000 - val_mse:
17091978.0000
Epoch 62/100
1000/1000           3s 1ms/step -
loss: 20354162.0000 - mse: 20354162.0000 - val_loss: 16999836.0000 - val_mse:
16999836.0000
Epoch 63/100
1000/1000           1s 1ms/step -
loss: 19976510.0000 - mse: 19976510.0000 - val_loss: 16974908.0000 - val_mse:
16974908.0000
Epoch 64/100
1000/1000           1s 1ms/step -
loss: 20202470.0000 - mse: 20202470.0000 - val_loss: 16954738.0000 - val_mse:
16954738.0000
Epoch 65/100
1000/1000           1s 1ms/step -
loss: 19819764.0000 - mse: 19819764.0000 - val_loss: 16832092.0000 - val_mse:
16832092.0000
Epoch 66/100
1000/1000           1s 1ms/step -
loss: 20642016.0000 - mse: 20642016.0000 - val_loss: 16808694.0000 - val_mse:
16808694.0000
Epoch 67/100
1000/1000           1s 1ms/step -
loss: 19725990.0000 - mse: 19725990.0000 - val_loss: 16756394.0000 - val_mse:
16756394.0000
Epoch 68/100
1000/1000           1s 1ms/step -
loss: 20565736.0000 - mse: 20565736.0000 - val_loss: 16778998.0000 - val_mse:
16778998.0000
Epoch 69/100
1000/1000           1s 1ms/step -
loss: 20302704.0000 - mse: 20302704.0000 - val_loss: 16775881.0000 - val_mse:
16775881.0000
Epoch 70/100
1000/1000           1s 1ms/step -
loss: 20657646.0000 - mse: 20657646.0000 - val_loss: 16605757.0000 - val_mse:
16605757.0000
Epoch 71/100
1000/1000           1s 1ms/step -
loss: 20876034.0000 - mse: 20876034.0000 - val_loss: 16616390.0000 - val_mse:
16616390.0000
```

```
Epoch 72/100
1000/1000            1s 1ms/step -
loss: 21790510.0000 - mse: 21790510.0000 - val_loss: 16540683.0000 - val_mse:
16540683.0000
Epoch 73/100
1000/1000            1s 1ms/step -
loss: 21222242.0000 - mse: 21222242.0000 - val_loss: 16579475.0000 - val_mse:
16579475.0000
Epoch 74/100
1000/1000            1s 1ms/step -
loss: 21178920.0000 - mse: 21178920.0000 - val_loss: 16467370.0000 - val_mse:
16467370.0000
Epoch 75/100
1000/1000            1s 1ms/step -
loss: 19455134.0000 - mse: 19455134.0000 - val_loss: 16468983.0000 - val_mse:
16468983.0000
Epoch 76/100
1000/1000            1s 1ms/step -
loss: 21104734.0000 - mse: 21104734.0000 - val_loss: 16450309.0000 - val_mse:
16450309.0000
Epoch 77/100
1000/1000            1s 1ms/step -
loss: 20497180.0000 - mse: 20497180.0000 - val_loss: 16422646.0000 - val_mse:
16422646.0000
Epoch 78/100
1000/1000            1s 1ms/step -
loss: 20774236.0000 - mse: 20774236.0000 - val_loss: 16450442.0000 - val_mse:
16450442.0000
Epoch 79/100
1000/1000            3s 1ms/step -
loss: 19656758.0000 - mse: 19656758.0000 - val_loss: 16335507.0000 - val_mse:
16335507.0000
Epoch 80/100
1000/1000            1s 1ms/step -
loss: 20110476.0000 - mse: 20110476.0000 - val_loss: 16310395.0000 - val_mse:
16310395.0000
Epoch 81/100
1000/1000            1s 1ms/step -
loss: 19129436.0000 - mse: 19129436.0000 - val_loss: 16465570.0000 - val_mse:
16465570.0000
Epoch 82/100
1000/1000            1s 1ms/step -
loss: 20463040.0000 - mse: 20463040.0000 - val_loss: 16249046.0000 - val_mse:
16249046.0000
Epoch 83/100
1000/1000            1s 1ms/step -
loss: 20086110.0000 - mse: 20086110.0000 - val_loss: 16208406.0000 - val_mse:
16208406.0000
```

```
Epoch 84/100
1000/1000            1s 1ms/step -
loss: 19713322.0000 - mse: 19713322.0000 - val_loss: 16153833.0000 - val_mse:
16153833.0000
Epoch 85/100
1000/1000            1s 1ms/step -
loss: 19143478.0000 - mse: 19143478.0000 - val_loss: 16163490.0000 - val_mse:
16163490.0000
Epoch 86/100
1000/1000            2s 2ms/step -
loss: 19126396.0000 - mse: 19126396.0000 - val_loss: 16161004.0000 - val_mse:
16161004.0000
Epoch 87/100
1000/1000            2s 2ms/step -
loss: 20843998.0000 - mse: 20843998.0000 - val_loss: 16114629.0000 - val_mse:
16114629.0000
Epoch 88/100
1000/1000            2s 2ms/step -
loss: 18508314.0000 - mse: 18508314.0000 - val_loss: 16046449.0000 - val_mse:
16046449.0000
Epoch 89/100
1000/1000            2s 2ms/step -
loss: 20940534.0000 - mse: 20940534.0000 - val_loss: 16188975.0000 - val_mse:
16188975.0000
Epoch 90/100
1000/1000            2s 1ms/step -
loss: 19167932.0000 - mse: 19167932.0000 - val_loss: 15966461.0000 - val_mse:
15966461.0000
Epoch 91/100
1000/1000            2s 2ms/step -
loss: 20376270.0000 - mse: 20376270.0000 - val_loss: 16000179.0000 - val_mse:
16000179.0000
Epoch 92/100
1000/1000            2s 1ms/step -
loss: 18958204.0000 - mse: 18958204.0000 - val_loss: 15967520.0000 - val_mse:
15967520.0000
Epoch 93/100
1000/1000            1s 1ms/step -
loss: 19359764.0000 - mse: 19359764.0000 - val_loss: 15864368.0000 - val_mse:
15864368.0000
Epoch 94/100
1000/1000            1s 1ms/step -
loss: 19334820.0000 - mse: 19334820.0000 - val_loss: 15964335.0000 - val_mse:
15964335.0000
Epoch 95/100
1000/1000            1s 1ms/step -
loss: 18899866.0000 - mse: 18899866.0000 - val_loss: 15869695.0000 - val_mse:
15869695.0000
```

```
Epoch 96/100
1000/1000            2s 2ms/step -
loss: 19469132.0000 - mse: 19469132.0000 - val_loss: 15832176.0000 - val_mse:
15832176.0000
Epoch 97/100
1000/1000            1s 1ms/step -
loss: 19531174.0000 - mse: 19531174.0000 - val_loss: 15779734.0000 - val_mse:
15779734.0000
Epoch 98/100
1000/1000            1s 1ms/step -
loss: 18439458.0000 - mse: 18439458.0000 - val_loss: 15803318.0000 - val_mse:
15803318.0000
Epoch 99/100
1000/1000            2s 2ms/step -
loss: 18851984.0000 - mse: 18851984.0000 - val_loss: 15721355.0000 - val_mse:
15721355.0000
Epoch 100/100
1000/1000            1s 1ms/step -
loss: 19012760.0000 - mse: 19012760.0000 - val_loss: 15695094.0000 - val_mse:
15695094.0000
313/313            0s 1ms/step
Artificial Neural Network Model:
Mean Squared Error: 17244476.519537665
R-squared: 0.936455226886118
```




# 5 ANN with Batch size of 64, and Learning rate of 0.001

[5]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Load the dataset
file_path = 'car_sales_data_24.csv'
car_sales_data = pd.read_csv(file_path)

# Selecting features
numerical_features = ['Engine size', 'Year of manufacture', 'Mileage']
categorical_features = ['Manufacturer', 'Model', 'Fuel type']
```

```python
# Prepare the dataset
X = car_sales_data[numerical_features + categorical_features]
y = car_sales_data['Price']

# Encode categorical variables using LabelEncoder
label_encoders = {}
for feature in categorical_features:
    label_encoders[feature] = LabelEncoder()
    X[feature] = label_encoders[feature].fit_transform(X[feature])

# Scale numerical features
scaler = StandardScaler()
X[numerical_features] = scaler.fit_transform(X[numerical_features])

# 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)

# Define the architecture of the neural network
def create_model(learning_rate=0.001, dropout_rate=0.2):
    model = Sequential()
    model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(loss='mean_squared_error', optimizer=optimizer,␣
↪metrics=['mse'])
    return model

# Create the model
model = create_model()

# Train the model
history = model.fit(X_train, y_train, epochs=100, batch_size=64,␣
↪validation_split=0.2, verbose=1)

# Predict on the test set
y_pred = model.predict(X_test).flatten()

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print("Artificial Neural Network Model:")
print(f"Mean Squared Error: {mse}")
```

```python
print(f"R-squared: {r2}")

# Plot training & validation loss values
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```

```
Epoch 1/100
```

```
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\1093076533.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\1093076533.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\1093076533.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\1093076533.py:30:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[numerical_features] = scaler.fit_transform(X[numerical_features])
C:\Users\sleek\AppData\Local\Programs\Python\Python312\Lib\site-
```

```
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

500/500            2s 3ms/step -
loss: 449192032.0000 - mse: 449192032.0000 - val_loss: 253599984.0000 - val_mse:
253599984.0000
Epoch 2/100
500/500            1s 2ms/step -
loss: 245633984.0000 - mse: 245633984.0000 - val_loss: 170776608.0000 - val_mse:
170776608.0000
Epoch 3/100
500/500            1s 2ms/step -
loss: 157880768.0000 - mse: 157880768.0000 - val_loss: 93806808.0000 - val_mse:
93806808.0000
Epoch 4/100
500/500            1s 3ms/step -
loss: 89700336.0000 - mse: 89700336.0000 - val_loss: 58324280.0000 - val_mse:
58324280.0000
Epoch 5/100
500/500            2s 3ms/step -
loss: 58795536.0000 - mse: 58795536.0000 - val_loss: 45152456.0000 - val_mse:
45152456.0000
Epoch 6/100
500/500            1s 2ms/step -
loss: 46672280.0000 - mse: 46672280.0000 - val_loss: 38520032.0000 - val_mse:
38520032.0000
Epoch 7/100
500/500            1s 2ms/step -
loss: 42838604.0000 - mse: 42838604.0000 - val_loss: 34464640.0000 - val_mse:
34464640.0000
Epoch 8/100
500/500            1s 2ms/step -
loss: 40116508.0000 - mse: 40116508.0000 - val_loss: 32342384.0000 - val_mse:
32342384.0000
Epoch 9/100
500/500            1s 2ms/step -
loss: 37515736.0000 - mse: 37515736.0000 - val_loss: 31080144.0000 - val_mse:
31080144.0000
Epoch 10/100
500/500            1s 2ms/step -
loss: 35255540.0000 - mse: 35255540.0000 - val_loss: 30219114.0000 - val_mse:
30219114.0000
Epoch 11/100
500/500            1s 2ms/step -
loss: 33199372.0000 - mse: 33199372.0000 - val_loss: 29373968.0000 - val_mse:
29373968.0000
```

```
Epoch 12/100
500/500              1s 2ms/step -
loss: 33276128.0000 - mse: 33276128.0000 - val_loss: 28282960.0000 - val_mse:
28282960.0000
Epoch 13/100
500/500              1s 2ms/step -
loss: 32498584.0000 - mse: 32498584.0000 - val_loss: 27514358.0000 - val_mse:
27514358.0000
Epoch 14/100
500/500              1s 2ms/step -
loss: 31927222.0000 - mse: 31927222.0000 - val_loss: 26623656.0000 - val_mse:
26623656.0000
Epoch 15/100
500/500              1s 2ms/step -
loss: 31116958.0000 - mse: 31116958.0000 - val_loss: 25921332.0000 - val_mse:
25921332.0000
Epoch 16/100
500/500              1s 2ms/step -
loss: 30055444.0000 - mse: 30055444.0000 - val_loss: 25309668.0000 - val_mse:
25309668.0000
Epoch 17/100
500/500              1s 2ms/step -
loss: 29076868.0000 - mse: 29076868.0000 - val_loss: 24857822.0000 - val_mse:
24857822.0000
Epoch 18/100
500/500              1s 2ms/step -
loss: 30851870.0000 - mse: 30851870.0000 - val_loss: 24371774.0000 - val_mse:
24371774.0000
Epoch 19/100
500/500              1s 2ms/step -
loss: 30185852.0000 - mse: 30185852.0000 - val_loss: 23926192.0000 - val_mse:
23926192.0000
Epoch 20/100
500/500              1s 2ms/step -
loss: 29086548.0000 - mse: 29086548.0000 - val_loss: 23531302.0000 - val_mse:
23531302.0000
Epoch 21/100
500/500              1s 2ms/step -
loss: 27277752.0000 - mse: 27277752.0000 - val_loss: 23144818.0000 - val_mse:
23144818.0000
Epoch 22/100
500/500              1s 2ms/step -
loss: 27024542.0000 - mse: 27024542.0000 - val_loss: 22839840.0000 - val_mse:
22839840.0000
Epoch 23/100
500/500              1s 2ms/step -
loss: 28392602.0000 - mse: 28392602.0000 - val_loss: 22510546.0000 - val_mse:
22510546.0000
```

```
Epoch 24/100
500/500            1s 2ms/step -
loss: 26462864.0000 - mse: 26462864.0000 - val_loss: 22209132.0000 - val_mse:
22209132.0000
Epoch 25/100
500/500            1s 2ms/step -
loss: 27301492.0000 - mse: 27301492.0000 - val_loss: 21978648.0000 - val_mse:
21978648.0000
Epoch 26/100
500/500            1s 2ms/step -
loss: 27754502.0000 - mse: 27754502.0000 - val_loss: 21665352.0000 - val_mse:
21665352.0000
Epoch 27/100
500/500            1s 2ms/step -
loss: 25084330.0000 - mse: 25084330.0000 - val_loss: 21490014.0000 - val_mse:
21490014.0000
Epoch 28/100
500/500            1s 2ms/step -
loss: 26670700.0000 - mse: 26670700.0000 - val_loss: 21237786.0000 - val_mse:
21237786.0000
Epoch 29/100
500/500            1s 2ms/step -
loss: 25439364.0000 - mse: 25439364.0000 - val_loss: 21057336.0000 - val_mse:
21057336.0000
Epoch 30/100
500/500            1s 2ms/step -
loss: 24849072.0000 - mse: 24849072.0000 - val_loss: 20890566.0000 - val_mse:
20890566.0000
Epoch 31/100
500/500            1s 2ms/step -
loss: 26134836.0000 - mse: 26134836.0000 - val_loss: 20688250.0000 - val_mse:
20688250.0000
Epoch 32/100
500/500            1s 2ms/step -
loss: 24621020.0000 - mse: 24621020.0000 - val_loss: 20560826.0000 - val_mse:
20560826.0000
Epoch 33/100
500/500            1s 2ms/step -
loss: 26036084.0000 - mse: 26036084.0000 - val_loss: 20363006.0000 - val_mse:
20363006.0000
Epoch 34/100
500/500            1s 2ms/step -
loss: 24378916.0000 - mse: 24378916.0000 - val_loss: 20202870.0000 - val_mse:
20202870.0000
Epoch 35/100
500/500            1s 2ms/step -
loss: 26330214.0000 - mse: 26330214.0000 - val_loss: 20047204.0000 - val_mse:
20047204.0000
```

```
Epoch 36/100
500/500            1s 2ms/step -
loss: 24944892.0000 - mse: 24944892.0000 - val_loss: 19907134.0000 - val_mse:
19907134.0000
Epoch 37/100
500/500            1s 2ms/step -
loss: 24238530.0000 - mse: 24238530.0000 - val_loss: 19776088.0000 - val_mse:
19776088.0000
Epoch 38/100
500/500            1s 2ms/step -
loss: 24317518.0000 - mse: 24317518.0000 - val_loss: 19660408.0000 - val_mse:
19660408.0000
Epoch 39/100
500/500            1s 2ms/step -
loss: 24442008.0000 - mse: 24442008.0000 - val_loss: 19649338.0000 - val_mse:
19649338.0000
Epoch 40/100
500/500            1s 2ms/step -
loss: 24246888.0000 - mse: 24246888.0000 - val_loss: 19469892.0000 - val_mse:
19469892.0000
Epoch 41/100
500/500            1s 2ms/step -
loss: 25122258.0000 - mse: 25122258.0000 - val_loss: 19317716.0000 - val_mse:
19317716.0000
Epoch 42/100
500/500            1s 2ms/step -
loss: 24014862.0000 - mse: 24014862.0000 - val_loss: 19232230.0000 - val_mse:
19232230.0000
Epoch 43/100
500/500            1s 2ms/step -
loss: 24297106.0000 - mse: 24297106.0000 - val_loss: 19237118.0000 - val_mse:
19237118.0000
Epoch 44/100
500/500            1s 2ms/step -
loss: 23456706.0000 - mse: 23456706.0000 - val_loss: 19001060.0000 - val_mse:
19001060.0000
Epoch 45/100
500/500            1s 2ms/step -
loss: 22986244.0000 - mse: 22986244.0000 - val_loss: 18962194.0000 - val_mse:
18962194.0000
Epoch 46/100
500/500            1s 2ms/step -
loss: 24704854.0000 - mse: 24704854.0000 - val_loss: 18837300.0000 - val_mse:
18837300.0000
Epoch 47/100
500/500            1s 2ms/step -
loss: 23735722.0000 - mse: 23735722.0000 - val_loss: 18791912.0000 - val_mse:
18791912.0000
```

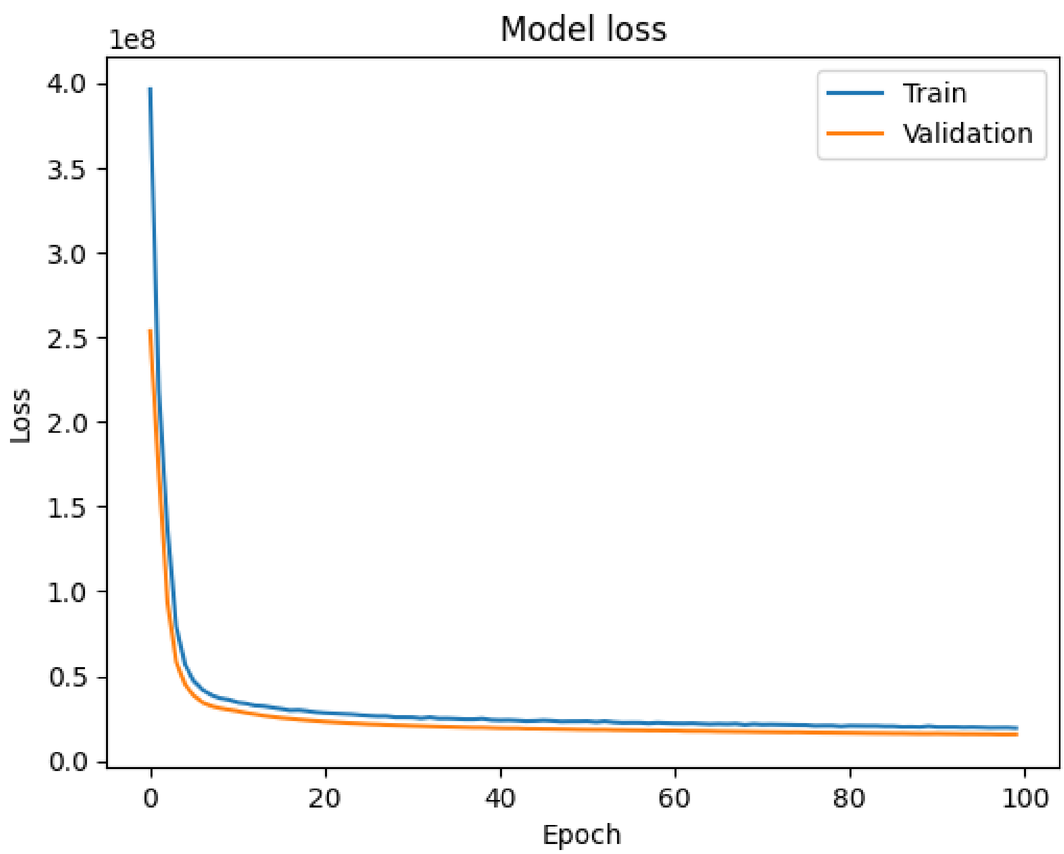
```
Epoch 48/100
500/500            1s 2ms/step -
loss: 24127496.0000 - mse: 24127496.0000 - val_loss: 18681848.0000 - val_mse:
18681848.0000
Epoch 49/100
500/500            1s 2ms/step -
loss: 22940438.0000 - mse: 22940438.0000 - val_loss: 18605536.0000 - val_mse:
18605536.0000
Epoch 50/100
500/500            1s 2ms/step -
loss: 24059278.0000 - mse: 24059278.0000 - val_loss: 18480930.0000 - val_mse:
18480930.0000
Epoch 51/100
500/500            1s 2ms/step -
loss: 24409560.0000 - mse: 24409560.0000 - val_loss: 18408532.0000 - val_mse:
18408532.0000
Epoch 52/100
500/500            1s 2ms/step -
loss: 22904212.0000 - mse: 22904212.0000 - val_loss: 18356518.0000 - val_mse:
18356518.0000
Epoch 53/100
500/500            1s 2ms/step -
loss: 24644972.0000 - mse: 24644972.0000 - val_loss: 18354544.0000 - val_mse:
18354544.0000
Epoch 54/100
500/500            1s 2ms/step -
loss: 22730010.0000 - mse: 22730010.0000 - val_loss: 18183592.0000 - val_mse:
18183592.0000
Epoch 55/100
500/500            1s 2ms/step -
loss: 22902790.0000 - mse: 22902790.0000 - val_loss: 18140276.0000 - val_mse:
18140276.0000
Epoch 56/100
500/500            1s 2ms/step -
loss: 23035396.0000 - mse: 23035396.0000 - val_loss: 18050172.0000 - val_mse:
18050172.0000
Epoch 57/100
500/500            1s 2ms/step -
loss: 21800734.0000 - mse: 21800734.0000 - val_loss: 18010954.0000 - val_mse:
18010954.0000
Epoch 58/100
500/500            1s 2ms/step -
loss: 21990084.0000 - mse: 21990084.0000 - val_loss: 17910966.0000 - val_mse:
17910966.0000
Epoch 59/100
500/500            1s 2ms/step -
loss: 22038176.0000 - mse: 22038176.0000 - val_loss: 17916418.0000 - val_mse:
17916418.0000
```

```
Epoch 60/100
500/500             1s 2ms/step -
loss: 22035224.0000 - mse: 22035224.0000 - val_loss: 17795556.0000 - val_mse:
17795556.0000
Epoch 61/100
500/500             1s 2ms/step -
loss: 21983888.0000 - mse: 21983888.0000 - val_loss: 17813162.0000 - val_mse:
17813162.0000
Epoch 62/100
500/500             1s 2ms/step -
loss: 21619126.0000 - mse: 21619126.0000 - val_loss: 17607018.0000 - val_mse:
17607018.0000
Epoch 63/100
500/500             1s 2ms/step -
loss: 22339060.0000 - mse: 22339060.0000 - val_loss: 17557364.0000 - val_mse:
17557364.0000
Epoch 64/100
500/500             1s 2ms/step -
loss: 21615146.0000 - mse: 21615146.0000 - val_loss: 17543528.0000 - val_mse:
17543528.0000
Epoch 65/100
500/500             1s 2ms/step -
loss: 21195144.0000 - mse: 21195144.0000 - val_loss: 17488960.0000 - val_mse:
17488960.0000
Epoch 66/100
500/500             1s 2ms/step -
loss: 22008154.0000 - mse: 22008154.0000 - val_loss: 17353382.0000 - val_mse:
17353382.0000
Epoch 67/100
500/500             1s 2ms/step -
loss: 21679074.0000 - mse: 21679074.0000 - val_loss: 17305908.0000 - val_mse:
17305908.0000
Epoch 68/100
500/500             1s 2ms/step -
loss: 21648008.0000 - mse: 21648008.0000 - val_loss: 17229720.0000 - val_mse:
17229720.0000
Epoch 69/100
500/500             1s 2ms/step -
loss: 21036152.0000 - mse: 21036152.0000 - val_loss: 17189048.0000 - val_mse:
17189048.0000
Epoch 70/100
500/500             1s 2ms/step -
loss: 23264444.0000 - mse: 23264444.0000 - val_loss: 17094876.0000 - val_mse:
17094876.0000
Epoch 71/100
500/500             1s 2ms/step -
loss: 21140812.0000 - mse: 21140812.0000 - val_loss: 17025420.0000 - val_mse:
17025420.0000
```

```
Epoch 72/100
500/500             1s 2ms/step -
loss: 22034650.0000 - mse: 22034650.0000 - val_loss: 16974516.0000 - val_mse:
16974516.0000
Epoch 73/100
500/500             1s 2ms/step -
loss: 21581056.0000 - mse: 21581056.0000 - val_loss: 16923262.0000 - val_mse:
16923262.0000
Epoch 74/100
500/500             1s 2ms/step -
loss: 21444124.0000 - mse: 21444124.0000 - val_loss: 16864886.0000 - val_mse:
16864886.0000
Epoch 75/100
500/500             1s 2ms/step -
loss: 21724318.0000 - mse: 21724318.0000 - val_loss: 16886090.0000 - val_mse:
16886090.0000
Epoch 76/100
500/500             1s 2ms/step -
loss: 21637444.0000 - mse: 21637444.0000 - val_loss: 16756741.0000 - val_mse:
16756741.0000
Epoch 77/100
500/500             1s 2ms/step -
loss: 21901456.0000 - mse: 21901456.0000 - val_loss: 16690242.0000 - val_mse:
16690242.0000
Epoch 78/100
500/500             1s 2ms/step -
loss: 21295558.0000 - mse: 21295558.0000 - val_loss: 16585650.0000 - val_mse:
16585650.0000
Epoch 79/100
500/500             1s 2ms/step -
loss: 21638094.0000 - mse: 21638094.0000 - val_loss: 16562021.0000 - val_mse:
16562021.0000
Epoch 80/100
500/500             1s 2ms/step -
loss: 20352728.0000 - mse: 20352728.0000 - val_loss: 16495233.0000 - val_mse:
16495233.0000
Epoch 81/100
500/500             1s 2ms/step -
loss: 21416978.0000 - mse: 21416978.0000 - val_loss: 16452518.0000 - val_mse:
16452518.0000
Epoch 82/100
500/500             1s 2ms/step -
loss: 19708178.0000 - mse: 19708178.0000 - val_loss: 16401117.0000 - val_mse:
16401117.0000
Epoch 83/100
500/500             1s 2ms/step -
loss: 21156708.0000 - mse: 21156708.0000 - val_loss: 16323560.0000 - val_mse:
16323560.0000
```

```
Epoch 84/100
500/500              1s 2ms/step -
loss: 20904076.0000 - mse: 20904076.0000 - val_loss: 16273676.0000 - val_mse:
16273676.0000
Epoch 85/100
500/500              1s 2ms/step -
loss: 20257560.0000 - mse: 20257560.0000 - val_loss: 16241580.0000 - val_mse:
16241580.0000
Epoch 86/100
500/500              1s 2ms/step -
loss: 20274636.0000 - mse: 20274636.0000 - val_loss: 16165651.0000 - val_mse:
16165651.0000
Epoch 87/100
500/500              1s 2ms/step -
loss: 19585014.0000 - mse: 19585014.0000 - val_loss: 16161983.0000 - val_mse:
16161983.0000
Epoch 88/100
500/500              1s 2ms/step -
loss: 19904616.0000 - mse: 19904616.0000 - val_loss: 16065154.0000 - val_mse:
16065154.0000
Epoch 89/100
500/500              1s 2ms/step -
loss: 21037400.0000 - mse: 21037400.0000 - val_loss: 16001564.0000 - val_mse:
16001564.0000
Epoch 90/100
500/500              1s 2ms/step -
loss: 20673784.0000 - mse: 20673784.0000 - val_loss: 15991161.0000 - val_mse:
15991161.0000
Epoch 91/100
500/500              1s 2ms/step -
loss: 19914318.0000 - mse: 19914318.0000 - val_loss: 16039917.0000 - val_mse:
16039917.0000
Epoch 92/100
500/500              1s 2ms/step -
loss: 19673772.0000 - mse: 19673772.0000 - val_loss: 15946932.0000 - val_mse:
15946932.0000
Epoch 93/100
500/500              1s 2ms/step -
loss: 19247482.0000 - mse: 19247482.0000 - val_loss: 15901235.0000 - val_mse:
15901235.0000
Epoch 94/100
500/500              1s 2ms/step -
loss: 19503598.0000 - mse: 19503598.0000 - val_loss: 15815375.0000 - val_mse:
15815375.0000
Epoch 95/100
500/500              1s 2ms/step -
loss: 19467006.0000 - mse: 19467006.0000 - val_loss: 15824316.0000 - val_mse:
15824316.0000
```

```
Epoch 96/100
500/500             1s 2ms/step -
loss: 19643810.0000 - mse: 19643810.0000 - val_loss: 15760271.0000 - val_mse:
15760271.0000
Epoch 97/100
500/500             1s 2ms/step -
loss: 20425526.0000 - mse: 20425526.0000 - val_loss: 15757162.0000 - val_mse:
15757162.0000
Epoch 98/100
500/500             1s 2ms/step -
loss: 19663950.0000 - mse: 19663950.0000 - val_loss: 15630959.0000 - val_mse:
15630959.0000
Epoch 99/100
500/500             1s 2ms/step -
loss: 20825090.0000 - mse: 20825090.0000 - val_loss: 15622135.0000 - val_mse:
15622135.0000
Epoch 100/100
500/500             1s 2ms/step -
loss: 18262834.0000 - mse: 18262834.0000 - val_loss: 15653207.0000 - val_mse:
15653207.0000
313/313             0s 1ms/step
Artificial Neural Network Model:
Mean Squared Error: 17347649.265640073
R-squared: 0.9360750317573547
```




# 6 ANN with Learning rate of 0.001, batch size of 32, and 150 Epochs

[6]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.optimizers import Adam

# Load the dataset
file_path = 'car_sales_data_24.csv'
car_sales_data = pd.read_csv(file_path)

# Selecting features
numerical_features = ['Engine size', 'Year of manufacture', 'Mileage']
```

```python
categorical_features = ['Manufacturer', 'Model', 'Fuel type']

# Prepare the dataset
X = car_sales_data[numerical_features + categorical_features]
y = car_sales_data['Price']

# Encode categorical variables using LabelEncoder
label_encoders = {}
for feature in categorical_features:
    label_encoders[feature] = LabelEncoder()
    X[feature] = label_encoders[feature].fit_transform(X[feature])

# Scale numerical features
scaler = StandardScaler()
X[numerical_features] = scaler.fit_transform(X[numerical_features])

# 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)

# Define the architecture of the neural network
def create_model(learning_rate=0.001, dropout_rate=0.2):
    model = Sequential()
    model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
    model.add(Dropout(dropout_rate))
    model.add(Dense(32, activation='relu'))
    model.add(Dense(1))

    optimizer = Adam(learning_rate=learning_rate)
    model.compile(loss='mean_squared_error', optimizer=optimizer,␣
↪metrics=['mse'])
    return model

# Create the model
model = create_model()

# Train the model
history = model.fit(X_train, y_train, epochs=150, batch_size=32,␣
↪validation_split=0.2, verbose=1)

# Predict on the test set
y_pred = model.predict(X_test).flatten()

# Evaluate the model
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

```python
print("Artificial Neural Network Model:")
print(f"Mean Squared Error: {mse}")
print(f"R-squared: {r2}")

# Plot training & validation loss values
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```

```
Epoch 1/150
```

```
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\1419522141.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\1419522141.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\1419522141.py:26:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
  X[feature] = label_encoders[feature].fit_transform(X[feature])
C:\Users\sleek\AppData\Local\Temp\ipykernel_8128\1419522141.py:30:
SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-
docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

```
  X[numerical_features] = scaler.fit_transform(X[numerical_features])
C:\Users\sleek\AppData\Local\Programs\Python\Python312\Lib\site-
packages\keras\src\layers\core\dense.py:87: UserWarning: Do not pass an
`input_shape`/`input_dim` argument to a layer. When using Sequential models,
prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

1000/1000          3s 2ms/step -
loss: 420785984.0000 - mse: 420785984.0000 - val_loss: 209953120.0000 - val_mse:
209953120.0000
Epoch 2/150
1000/1000          2s 2ms/step -
loss: 193934128.0000 - mse: 193934128.0000 - val_loss: 102076160.0000 - val_mse:
102076160.0000
Epoch 3/150
1000/1000          2s 2ms/step -
loss: 87442752.0000 - mse: 87442752.0000 - val_loss: 53048704.0000 - val_mse:
53048704.0000
Epoch 4/150
1000/1000          2s 2ms/step -
loss: 55143692.0000 - mse: 55143692.0000 - val_loss: 40082668.0000 - val_mse:
40082668.0000
Epoch 5/150
1000/1000          2s 1ms/step -
loss: 42176192.0000 - mse: 42176192.0000 - val_loss: 34694916.0000 - val_mse:
34694916.0000
Epoch 6/150
1000/1000          2s 2ms/step -
loss: 41211520.0000 - mse: 41211520.0000 - val_loss: 32138226.0000 - val_mse:
32138226.0000
Epoch 7/150
1000/1000          2s 2ms/step -
loss: 37438496.0000 - mse: 37438496.0000 - val_loss: 30789226.0000 - val_mse:
30789226.0000
Epoch 8/150
1000/1000          2s 2ms/step -
loss: 36211552.0000 - mse: 36211552.0000 - val_loss: 29850648.0000 - val_mse:
29850648.0000
Epoch 9/150
1000/1000          2s 1ms/step -
loss: 34339604.0000 - mse: 34339604.0000 - val_loss: 28998944.0000 - val_mse:
28998944.0000
Epoch 10/150
1000/1000          2s 2ms/step -
loss: 33433270.0000 - mse: 33433270.0000 - val_loss: 28178450.0000 - val_mse:
28178450.0000
Epoch 11/150
1000/1000          2s 2ms/step -
```

```
loss: 32638290.0000 - mse: 32638290.0000 - val_loss: 27363538.0000 - val_mse:
27363538.0000
Epoch 12/150
1000/1000            2s 1ms/step -
loss: 31128602.0000 - mse: 31128602.0000 - val_loss: 26571746.0000 - val_mse:
26571746.0000
Epoch 13/150
1000/1000            2s 2ms/step -
loss: 33361244.0000 - mse: 33361244.0000 - val_loss: 25826418.0000 - val_mse:
25826418.0000
Epoch 14/150
1000/1000            2s 1ms/step -
loss: 30414402.0000 - mse: 30414402.0000 - val_loss: 25143926.0000 - val_mse:
25143926.0000
Epoch 15/150
1000/1000            2s 2ms/step -
loss: 29513794.0000 - mse: 29513794.0000 - val_loss: 24678626.0000 - val_mse:
24678626.0000
Epoch 16/150
1000/1000            2s 1ms/step -
loss: 29739370.0000 - mse: 29739370.0000 - val_loss: 24245726.0000 - val_mse:
24245726.0000
Epoch 17/150
1000/1000            2s 2ms/step -
loss: 27694950.0000 - mse: 27694950.0000 - val_loss: 23839110.0000 - val_mse:
23839110.0000
Epoch 18/150
1000/1000            2s 2ms/step -
loss: 30738828.0000 - mse: 30738828.0000 - val_loss: 23510716.0000 - val_mse:
23510716.0000
Epoch 19/150
1000/1000            2s 2ms/step -
loss: 29708340.0000 - mse: 29708340.0000 - val_loss: 23253110.0000 - val_mse:
23253110.0000
Epoch 20/150
1000/1000            2s 2ms/step -
loss: 28009660.0000 - mse: 28009660.0000 - val_loss: 22991338.0000 - val_mse:
22991338.0000
Epoch 21/150
1000/1000            2s 2ms/step -
loss: 27692324.0000 - mse: 27692324.0000 - val_loss: 22690700.0000 - val_mse:
22690700.0000
Epoch 22/150
1000/1000            2s 2ms/step -
loss: 25955196.0000 - mse: 25955196.0000 - val_loss: 22438056.0000 - val_mse:
22438056.0000
Epoch 23/150
1000/1000            2s 2ms/step -
```

```
loss: 26601906.0000 - mse: 26601906.0000 - val_loss: 22238780.0000 - val_mse:
22238780.0000
Epoch 24/150
1000/1000            2s 2ms/step -
loss: 26502068.0000 - mse: 26502068.0000 - val_loss: 21986476.0000 - val_mse:
21986476.0000
Epoch 25/150
1000/1000            2s 2ms/step -
loss: 26793848.0000 - mse: 26793848.0000 - val_loss: 21739858.0000 - val_mse:
21739858.0000
Epoch 26/150
1000/1000            2s 2ms/step -
loss: 26082400.0000 - mse: 26082400.0000 - val_loss: 21563846.0000 - val_mse:
21563846.0000
Epoch 27/150
1000/1000            2s 2ms/step -
loss: 26835108.0000 - mse: 26835108.0000 - val_loss: 21293548.0000 - val_mse:
21293548.0000
Epoch 28/150
1000/1000            2s 2ms/step -
loss: 26789450.0000 - mse: 26789450.0000 - val_loss: 21139738.0000 - val_mse:
21139738.0000
Epoch 29/150
1000/1000            2s 2ms/step -
loss: 26949368.0000 - mse: 26949368.0000 - val_loss: 20849160.0000 - val_mse:
20849160.0000
Epoch 30/150
1000/1000            2s 2ms/step -
loss: 25742050.0000 - mse: 25742050.0000 - val_loss: 20634096.0000 - val_mse:
20634096.0000
Epoch 31/150
1000/1000            3s 2ms/step -
loss: 26169616.0000 - mse: 26169616.0000 - val_loss: 20450474.0000 - val_mse:
20450474.0000
Epoch 32/150
1000/1000            2s 2ms/step -
loss: 24158398.0000 - mse: 24158398.0000 - val_loss: 20212892.0000 - val_mse:
20212892.0000
Epoch 33/150
1000/1000            2s 2ms/step -
loss: 25422006.0000 - mse: 25422006.0000 - val_loss: 20009176.0000 - val_mse:
20009176.0000
Epoch 34/150
1000/1000            2s 2ms/step -
loss: 25315270.0000 - mse: 25315270.0000 - val_loss: 19858688.0000 - val_mse:
19858688.0000
Epoch 35/150
1000/1000            2s 2ms/step -
```

```
loss: 24912208.0000 - mse: 24912208.0000 - val_loss: 19669466.0000 - val_mse:
19669466.0000
Epoch 36/150
1000/1000            2s 2ms/step -
loss: 23899448.0000 - mse: 23899448.0000 - val_loss: 19515116.0000 - val_mse:
19515116.0000
Epoch 37/150
1000/1000            2s 2ms/step -
loss: 23152680.0000 - mse: 23152680.0000 - val_loss: 19362926.0000 - val_mse:
19362926.0000
Epoch 38/150
1000/1000            2s 2ms/step -
loss: 24680722.0000 - mse: 24680722.0000 - val_loss: 19286888.0000 - val_mse:
19286888.0000
Epoch 39/150
1000/1000            3s 2ms/step -
loss: 23538004.0000 - mse: 23538004.0000 - val_loss: 19050682.0000 - val_mse:
19050682.0000
Epoch 40/150
1000/1000            2s 2ms/step -
loss: 24023872.0000 - mse: 24023872.0000 - val_loss: 18998818.0000 - val_mse:
18998818.0000
Epoch 41/150
1000/1000            2s 2ms/step -
loss: 23731844.0000 - mse: 23731844.0000 - val_loss: 18821910.0000 - val_mse:
18821910.0000
Epoch 42/150
1000/1000            2s 2ms/step -
loss: 22746690.0000 - mse: 22746690.0000 - val_loss: 18728878.0000 - val_mse:
18728878.0000
Epoch 43/150
1000/1000            2s 2ms/step -
loss: 21779926.0000 - mse: 21779926.0000 - val_loss: 18694140.0000 - val_mse:
18694140.0000
Epoch 44/150
1000/1000            3s 2ms/step -
loss: 24351746.0000 - mse: 24351746.0000 - val_loss: 18532254.0000 - val_mse:
18532254.0000
Epoch 45/150
1000/1000            2s 2ms/step -
loss: 23670996.0000 - mse: 23670996.0000 - val_loss: 18523590.0000 - val_mse:
18523590.0000
Epoch 46/150
1000/1000            2s 2ms/step -
loss: 22855176.0000 - mse: 22855176.0000 - val_loss: 18366760.0000 - val_mse:
18366760.0000
Epoch 47/150
1000/1000            3s 2ms/step -
```

```
loss: 21039100.0000 - mse: 21039100.0000 - val_loss: 18301314.0000 - val_mse:
18301314.0000
Epoch 48/150
1000/1000            2s 2ms/step -
loss: 23226072.0000 - mse: 23226072.0000 - val_loss: 18275236.0000 - val_mse:
18275236.0000
Epoch 49/150
1000/1000            2s 1ms/step -
loss: 22926822.0000 - mse: 22926822.0000 - val_loss: 18268824.0000 - val_mse:
18268824.0000
Epoch 50/150
1000/1000            2s 2ms/step -
loss: 22712234.0000 - mse: 22712234.0000 - val_loss: 18154232.0000 - val_mse:
18154232.0000
Epoch 51/150
1000/1000            2s 2ms/step -
loss: 23068420.0000 - mse: 23068420.0000 - val_loss: 18058874.0000 - val_mse:
18058874.0000
Epoch 52/150
1000/1000            2s 2ms/step -
loss: 21255448.0000 - mse: 21255448.0000 - val_loss: 18002926.0000 - val_mse:
18002926.0000
Epoch 53/150
1000/1000            2s 2ms/step -
loss: 24029350.0000 - mse: 24029350.0000 - val_loss: 17947612.0000 - val_mse:
17947612.0000
Epoch 54/150
1000/1000            2s 2ms/step -
loss: 22737858.0000 - mse: 22737858.0000 - val_loss: 17908782.0000 - val_mse:
17908782.0000
Epoch 55/150
1000/1000            2s 2ms/step -
loss: 22055702.0000 - mse: 22055702.0000 - val_loss: 17841156.0000 - val_mse:
17841156.0000
Epoch 56/150
1000/1000            2s 2ms/step -
loss: 21706462.0000 - mse: 21706462.0000 - val_loss: 17726928.0000 - val_mse:
17726928.0000
Epoch 57/150
1000/1000            2s 2ms/step -
loss: 22524058.0000 - mse: 22524058.0000 - val_loss: 17719010.0000 - val_mse:
17719010.0000
Epoch 58/150
1000/1000            2s 2ms/step -
loss: 22515276.0000 - mse: 22515276.0000 - val_loss: 17636288.0000 - val_mse:
17636288.0000
Epoch 59/150
1000/1000            2s 2ms/step -
```

```
loss: 22755590.0000 - mse: 22755590.0000 - val_loss: 17570486.0000 - val_mse:
17570486.0000
Epoch 60/150
1000/1000            2s 2ms/step -
loss: 22090314.0000 - mse: 22090314.0000 - val_loss: 17544880.0000 - val_mse:
17544880.0000
Epoch 61/150
1000/1000            2s 2ms/step -
loss: 21691610.0000 - mse: 21691610.0000 - val_loss: 17471048.0000 - val_mse:
17471048.0000
Epoch 62/150
1000/1000            2s 2ms/step -
loss: 21814960.0000 - mse: 21814960.0000 - val_loss: 17525056.0000 - val_mse:
17525056.0000
Epoch 63/150
1000/1000            2s 2ms/step -
loss: 23101044.0000 - mse: 23101044.0000 - val_loss: 17482804.0000 - val_mse:
17482804.0000
Epoch 64/150
1000/1000            2s 2ms/step -
loss: 21520682.0000 - mse: 21520682.0000 - val_loss: 17570866.0000 - val_mse:
17570866.0000
Epoch 65/150
1000/1000            2s 2ms/step -
loss: 22199682.0000 - mse: 22199682.0000 - val_loss: 17482562.0000 - val_mse:
17482562.0000
Epoch 66/150
1000/1000            3s 2ms/step -
loss: 21831698.0000 - mse: 21831698.0000 - val_loss: 17336850.0000 - val_mse:
17336850.0000
Epoch 67/150
1000/1000            2s 2ms/step -
loss: 21240838.0000 - mse: 21240838.0000 - val_loss: 17387194.0000 - val_mse:
17387194.0000
Epoch 68/150
1000/1000            2s 2ms/step -
loss: 21296444.0000 - mse: 21296444.0000 - val_loss: 17325902.0000 - val_mse:
17325902.0000
Epoch 69/150
1000/1000            2s 2ms/step -
loss: 20368112.0000 - mse: 20368112.0000 - val_loss: 17332160.0000 - val_mse:
17332160.0000
Epoch 70/150
1000/1000            2s 2ms/step -
loss: 21846974.0000 - mse: 21846974.0000 - val_loss: 17307892.0000 - val_mse:
17307892.0000
Epoch 71/150
1000/1000            2s 2ms/step -
```

```
loss: 20946556.0000 - mse: 20946556.0000 - val_loss: 17247642.0000 - val_mse:
17247642.0000
Epoch 72/150
1000/1000            3s 3ms/step -
loss: 21428040.0000 - mse: 21428040.0000 - val_loss: 17183040.0000 - val_mse:
17183040.0000
Epoch 73/150
1000/1000            4s 2ms/step -
loss: 21263584.0000 - mse: 21263584.0000 - val_loss: 17211740.0000 - val_mse:
17211740.0000
Epoch 74/150
1000/1000            3s 2ms/step -
loss: 20911730.0000 - mse: 20911730.0000 - val_loss: 17199826.0000 - val_mse:
17199826.0000
Epoch 75/150
1000/1000            2s 2ms/step -
loss: 20301376.0000 - mse: 20301376.0000 - val_loss: 17109100.0000 - val_mse:
17109100.0000
Epoch 76/150
1000/1000            2s 2ms/step -
loss: 19843918.0000 - mse: 19843918.0000 - val_loss: 17116154.0000 - val_mse:
17116154.0000
Epoch 77/150
1000/1000            3s 2ms/step -
loss: 21831476.0000 - mse: 21831476.0000 - val_loss: 17062178.0000 - val_mse:
17062178.0000
Epoch 78/150
1000/1000            2s 2ms/step -
loss: 21106782.0000 - mse: 21106782.0000 - val_loss: 17017788.0000 - val_mse:
17017788.0000
Epoch 79/150
1000/1000            2s 2ms/step -
loss: 21274218.0000 - mse: 21274218.0000 - val_loss: 16983954.0000 - val_mse:
16983954.0000
Epoch 80/150
1000/1000            3s 2ms/step -
loss: 20566932.0000 - mse: 20566932.0000 - val_loss: 16925476.0000 - val_mse:
16925476.0000
Epoch 81/150
1000/1000            2s 2ms/step -
loss: 20942934.0000 - mse: 20942934.0000 - val_loss: 17013640.0000 - val_mse:
17013640.0000
Epoch 82/150
1000/1000            2s 2ms/step -
loss: 20557900.0000 - mse: 20557900.0000 - val_loss: 16867292.0000 - val_mse:
16867292.0000
Epoch 83/150
1000/1000            2s 2ms/step -
```

```
loss: 21763958.0000 - mse: 21763958.0000 - val_loss: 16907192.0000 - val_mse:
16907192.0000
Epoch 84/150
1000/1000            2s 2ms/step -
loss: 20899530.0000 - mse: 20899530.0000 - val_loss: 16869304.0000 - val_mse:
16869304.0000
Epoch 85/150
1000/1000            2s 2ms/step -
loss: 20030996.0000 - mse: 20030996.0000 - val_loss: 16795130.0000 - val_mse:
16795130.0000
Epoch 86/150
1000/1000            2s 2ms/step -
loss: 21112522.0000 - mse: 21112522.0000 - val_loss: 16962624.0000 - val_mse:
16962624.0000
Epoch 87/150
1000/1000            2s 2ms/step -
loss: 21234752.0000 - mse: 21234752.0000 - val_loss: 16772099.0000 - val_mse:
16772099.0000
Epoch 88/150
1000/1000            2s 2ms/step -
loss: 21209548.0000 - mse: 21209548.0000 - val_loss: 16834948.0000 - val_mse:
16834948.0000
Epoch 89/150
1000/1000            2s 2ms/step -
loss: 21295716.0000 - mse: 21295716.0000 - val_loss: 16676086.0000 - val_mse:
16676086.0000
Epoch 90/150
1000/1000            2s 2ms/step -
loss: 20321186.0000 - mse: 20321186.0000 - val_loss: 16686895.0000 - val_mse:
16686895.0000
Epoch 91/150
1000/1000            2s 2ms/step -
loss: 21263802.0000 - mse: 21263802.0000 - val_loss: 16754633.0000 - val_mse:
16754633.0000
Epoch 92/150
1000/1000            2s 1ms/step -
loss: 21476506.0000 - mse: 21476506.0000 - val_loss: 16652883.0000 - val_mse:
16652883.0000
Epoch 93/150
1000/1000            3s 2ms/step -
loss: 19655446.0000 - mse: 19655446.0000 - val_loss: 16628808.0000 - val_mse:
16628808.0000
Epoch 94/150
1000/1000            2s 2ms/step -
loss: 20467858.0000 - mse: 20467858.0000 - val_loss: 16644930.0000 - val_mse:
16644930.0000
Epoch 95/150
1000/1000            2s 2ms/step -
```

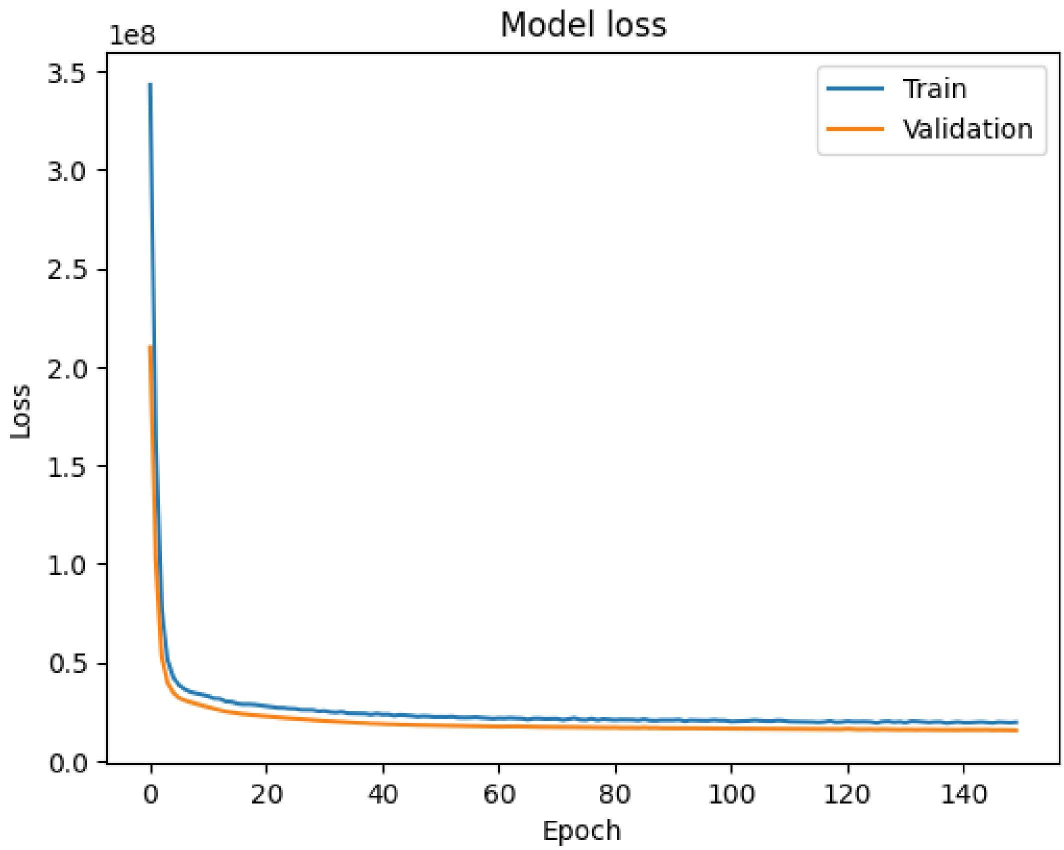
```
loss: 20519026.0000 - mse: 20519026.0000 - val_loss: 16591953.0000 - val_mse:
16591953.0000
Epoch 96/150
1000/1000            2s 2ms/step -
loss: 21752678.0000 - mse: 21752678.0000 - val_loss: 16618843.0000 - val_mse:
16618843.0000
Epoch 97/150
1000/1000            2s 2ms/step -
loss: 20007824.0000 - mse: 20007824.0000 - val_loss: 16577542.0000 - val_mse:
16577542.0000
Epoch 98/150
1000/1000            2s 2ms/step -
loss: 21407676.0000 - mse: 21407676.0000 - val_loss: 16539342.0000 - val_mse:
16539342.0000
Epoch 99/150
1000/1000            3s 2ms/step -
loss: 19850334.0000 - mse: 19850334.0000 - val_loss: 16521512.0000 - val_mse:
16521512.0000
Epoch 100/150
1000/1000            3s 2ms/step -
loss: 21325826.0000 - mse: 21325826.0000 - val_loss: 16458321.0000 - val_mse:
16458321.0000
Epoch 101/150
1000/1000            2s 2ms/step -
loss: 19608568.0000 - mse: 19608568.0000 - val_loss: 16498822.0000 - val_mse:
16498822.0000
Epoch 102/150
1000/1000            2s 2ms/step -
loss: 20043352.0000 - mse: 20043352.0000 - val_loss: 16514002.0000 - val_mse:
16514002.0000
Epoch 103/150
1000/1000            2s 2ms/step -
loss: 19683578.0000 - mse: 19683578.0000 - val_loss: 16405532.0000 - val_mse:
16405532.0000
Epoch 104/150
1000/1000            3s 2ms/step -
loss: 20014912.0000 - mse: 20014912.0000 - val_loss: 16385247.0000 - val_mse:
16385247.0000
Epoch 105/150
1000/1000            2s 2ms/step -
loss: 20844030.0000 - mse: 20844030.0000 - val_loss: 16406302.0000 - val_mse:
16406302.0000
Epoch 106/150
1000/1000            3s 2ms/step -
loss: 21623324.0000 - mse: 21623324.0000 - val_loss: 16432156.0000 - val_mse:
16432156.0000
Epoch 107/150
1000/1000            4s 4ms/step -
```

```
loss: 19882462.0000 - mse: 19882462.0000 - val_loss: 16397047.0000 - val_mse:
16397047.0000
Epoch 108/150
1000/1000            2s 2ms/step -
loss: 19678692.0000 - mse: 19678692.0000 - val_loss: 16310162.0000 - val_mse:
16310162.0000
Epoch 109/150
1000/1000            2s 2ms/step -
loss: 21101156.0000 - mse: 21101156.0000 - val_loss: 16284489.0000 - val_mse:
16284489.0000
Epoch 110/150
1000/1000            3s 2ms/step -
loss: 21425474.0000 - mse: 21425474.0000 - val_loss: 16313745.0000 - val_mse:
16313745.0000
Epoch 111/150
1000/1000            2s 2ms/step -
loss: 21135002.0000 - mse: 21135002.0000 - val_loss: 16254006.0000 - val_mse:
16254006.0000
Epoch 112/150
1000/1000            3s 2ms/step -
loss: 19668250.0000 - mse: 19668250.0000 - val_loss: 16220429.0000 - val_mse:
16220429.0000
Epoch 113/150
1000/1000            2s 2ms/step -
loss: 19406792.0000 - mse: 19406792.0000 - val_loss: 16299833.0000 - val_mse:
16299833.0000
Epoch 114/150
1000/1000            2s 2ms/step -
loss: 19247106.0000 - mse: 19247106.0000 - val_loss: 16229594.0000 - val_mse:
16229594.0000
Epoch 115/150
1000/1000            3s 2ms/step -
loss: 20296516.0000 - mse: 20296516.0000 - val_loss: 16189841.0000 - val_mse:
16189841.0000
Epoch 116/150
1000/1000            2s 2ms/step -
loss: 19551290.0000 - mse: 19551290.0000 - val_loss: 16169573.0000 - val_mse:
16169573.0000
Epoch 117/150
1000/1000            2s 2ms/step -
loss: 20102782.0000 - mse: 20102782.0000 - val_loss: 16167367.0000 - val_mse:
16167367.0000
Epoch 118/150
1000/1000            2s 2ms/step -
loss: 20897908.0000 - mse: 20897908.0000 - val_loss: 16263919.0000 - val_mse:
16263919.0000
Epoch 119/150
1000/1000            3s 2ms/step -
```

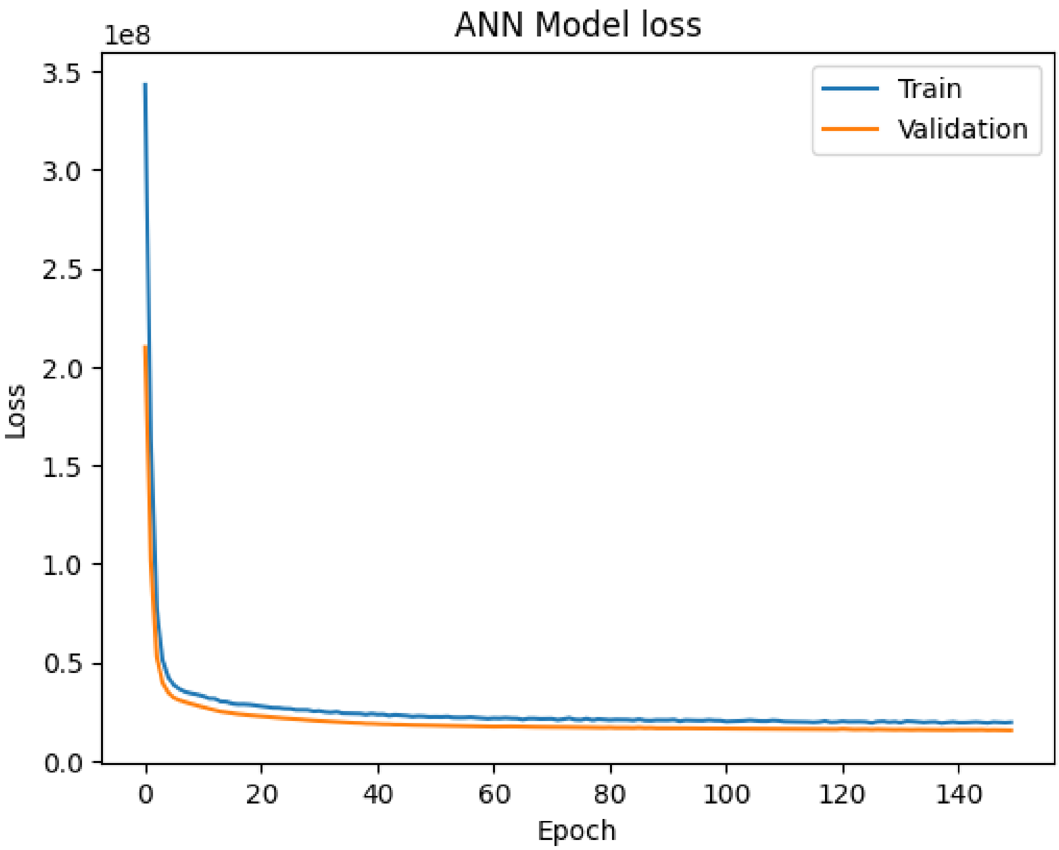
```
loss: 19964512.0000 - mse: 19964512.0000 - val_loss: 16117595.0000 - val_mse:
16117595.0000
Epoch 120/150
1000/1000            3s 2ms/step -
loss: 19342376.0000 - mse: 19342376.0000 - val_loss: 16153577.0000 - val_mse:
16153577.0000
Epoch 121/150
1000/1000            3s 2ms/step -
loss: 20310640.0000 - mse: 20310640.0000 - val_loss: 16354431.0000 - val_mse:
16354431.0000
Epoch 122/150
1000/1000            2s 2ms/step -
loss: 19007940.0000 - mse: 19007940.0000 - val_loss: 16173899.0000 - val_mse:
16173899.0000
Epoch 123/150
1000/1000            2s 2ms/step -
loss: 19425852.0000 - mse: 19425852.0000 - val_loss: 16048699.0000 - val_mse:
16048699.0000
Epoch 124/150
1000/1000            2s 2ms/step -
loss: 20973198.0000 - mse: 20973198.0000 - val_loss: 16073307.0000 - val_mse:
16073307.0000
Epoch 125/150
1000/1000            2s 2ms/step -
loss: 20598108.0000 - mse: 20598108.0000 - val_loss: 16115568.0000 - val_mse:
16115568.0000
Epoch 126/150
1000/1000            2s 2ms/step -
loss: 18690672.0000 - mse: 18690672.0000 - val_loss: 15973716.0000 - val_mse:
15973716.0000
Epoch 127/150
1000/1000            2s 2ms/step -
loss: 20114656.0000 - mse: 20114656.0000 - val_loss: 16101903.0000 - val_mse:
16101903.0000
Epoch 128/150
1000/1000            2s 2ms/step -
loss: 20071688.0000 - mse: 20071688.0000 - val_loss: 16021589.0000 - val_mse:
16021589.0000
Epoch 129/150
1000/1000            2s 2ms/step -
loss: 20084146.0000 - mse: 20084146.0000 - val_loss: 15944250.0000 - val_mse:
15944250.0000
Epoch 130/150
1000/1000            2s 2ms/step -
loss: 19822490.0000 - mse: 19822490.0000 - val_loss: 15906533.0000 - val_mse:
15906533.0000
Epoch 131/150
1000/1000            3s 2ms/step -
```

```
loss: 19288796.0000 - mse: 19288796.0000 - val_loss: 15962459.0000 - val_mse:
15962459.0000
Epoch 132/150
1000/1000            3s 2ms/step -
loss: 21107622.0000 - mse: 21107622.0000 - val_loss: 15896192.0000 - val_mse:
15896192.0000
Epoch 133/150
1000/1000            2s 2ms/step -
loss: 19988694.0000 - mse: 19988694.0000 - val_loss: 15861402.0000 - val_mse:
15861402.0000
Epoch 134/150
1000/1000            3s 2ms/step -
loss: 19722458.0000 - mse: 19722458.0000 - val_loss: 15941513.0000 - val_mse:
15941513.0000
Epoch 135/150
1000/1000            2s 2ms/step -
loss: 19705032.0000 - mse: 19705032.0000 - val_loss: 15860279.0000 - val_mse:
15860279.0000
Epoch 136/150
1000/1000            2s 2ms/step -
loss: 21006418.0000 - mse: 21006418.0000 - val_loss: 15866524.0000 - val_mse:
15866524.0000
Epoch 137/150
1000/1000            2s 2ms/step -
loss: 19811868.0000 - mse: 19811868.0000 - val_loss: 15835150.0000 - val_mse:
15835150.0000
Epoch 138/150
1000/1000            2s 2ms/step -
loss: 18803912.0000 - mse: 18803912.0000 - val_loss: 15811284.0000 - val_mse:
15811284.0000
Epoch 139/150
1000/1000            2s 2ms/step -
loss: 19210722.0000 - mse: 19210722.0000 - val_loss: 15813850.0000 - val_mse:
15813850.0000
Epoch 140/150
1000/1000            2s 2ms/step -
loss: 20500464.0000 - mse: 20500464.0000 - val_loss: 15795413.0000 - val_mse:
15795413.0000
Epoch 141/150
1000/1000            2s 2ms/step -
loss: 19000392.0000 - mse: 19000392.0000 - val_loss: 15869559.0000 - val_mse:
15869559.0000
Epoch 142/150
1000/1000            2s 2ms/step -
loss: 19745134.0000 - mse: 19745134.0000 - val_loss: 15829926.0000 - val_mse:
15829926.0000
Epoch 143/150
1000/1000            2s 2ms/step -
```

```
loss: 18678834.0000 - mse: 18678834.0000 - val_loss: 15759503.0000 - val_mse:
15759503.0000
Epoch 144/150
1000/1000            2s 2ms/step -
loss: 20046330.0000 - mse: 20046330.0000 - val_loss: 15839130.0000 - val_mse:
15839130.0000
Epoch 145/150
1000/1000            2s 2ms/step -
loss: 18749516.0000 - mse: 18749516.0000 - val_loss: 15882867.0000 - val_mse:
15882867.0000
Epoch 146/150
1000/1000            2s 2ms/step -
loss: 20291020.0000 - mse: 20291020.0000 - val_loss: 15723995.0000 - val_mse:
15723995.0000
Epoch 147/150
1000/1000            2s 2ms/step -
loss: 21062922.0000 - mse: 21062922.0000 - val_loss: 15783867.0000 - val_mse:
15783867.0000
Epoch 148/150
1000/1000            2s 2ms/step -
loss: 18967364.0000 - mse: 18967364.0000 - val_loss: 15696787.0000 - val_mse:
15696787.0000
Epoch 149/150
1000/1000            2s 2ms/step -
loss: 19919702.0000 - mse: 19919702.0000 - val_loss: 15660027.0000 - val_mse:
15660027.0000
Epoch 150/150
1000/1000            2s 2ms/step -
loss: 19606590.0000 - mse: 19606590.0000 - val_loss: 15628743.0000 - val_mse:
15628743.0000
313/313            0s 1ms/step
Artificial Neural Network Model:
Mean Squared Error: 17190826.9748952
R-squared: 0.9366528987884521
```




[11]:
```
import matplotlib.pyplot as plt

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('ANN Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.savefig('Best_ANN_model.png')
plt.show()
```

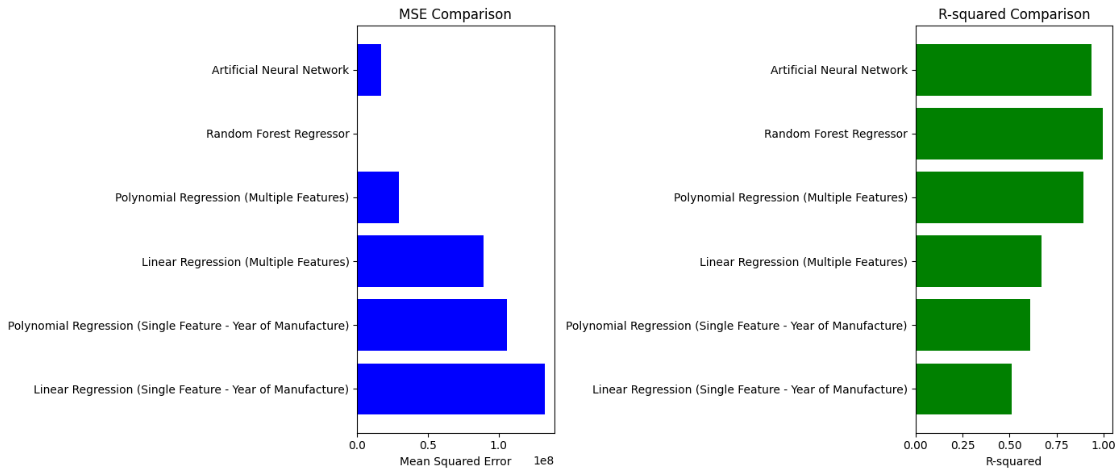



Based on the results of your analysis, what is the best model for predicting the price of a car and why? You should use suitable figures and evaluation metrics to support your conclusions.

[9]:
```
import pandas as pd
import matplotlib.pyplot as plt

# Sample metrics collected from previous models (these should be replaced with␣
↪actual values)
results = {
    'Model': ['Linear Regression (Single Feature - Year of Manufacture)',
              'Polynomial Regression (Single Feature - Year of Manufacture)',
              'Linear Regression (Multiple Features)',
              'Polynomial Regression (Multiple Features)',
              'Random Forest Regressor',
              'Artificial Neural Network'],
    'MSE': [132679000, 105993900, 89158620, 29311490, 475768.9, 17190826.9],
    'R-squared': [0.511087, 0.609419, 0.671456, 0.891989, 0.9982468, 0.936653]
}

# Create a DataFrame to display the results
```

```python
results_df = pd.DataFrame(results)

# Display the DataFrame
print("Model Performance Comparison:")
print(results_df)

# Plot the results
plt.figure(figsize=(14, 6))

# Plot MSE
plt.subplot(1, 2, 1)
plt.barh(results_df['Model'], results_df['MSE'], color='blue')
plt.xlabel('Mean Squared Error')
plt.title('MSE Comparison')

# Plot R-squared
plt.subplot(1, 2, 2)
plt.barh(results_df['Model'], results_df['R-squared'], color='green')
plt.xlabel('R-squared')
plt.title('R-squared Comparison')

plt.tight_layout()

plt.savefig('Model_performance_comparison.png')
plt.show()
```

```
Model Performance Comparison:
                                              Model          MSE  R-squared
0  Linear Regression (Single Feature - Year of Ma…  132679000.0   0.511087
1  Polynomial Regression (Single Feature - Year o…  105993900.0   0.609419
2             Linear Regression (Multiple Features)   89158620.0   0.671456
3         Polynomial Regression (Multiple Features)   29311490.0   0.891989
4                           Random Forest Regressor     475768.9   0.998247
5                         Artificial Neural Network   17190826.9   0.936653
```




Use the k-Means clustering algorithm to identify clusters in the car sales data. Consider different combinations of the numerical variables in the dataset to use as input features for the clustering algorithm. In each case, what is the optimal number of clusters (k) to use and why? Which combination of variables produces the best clustering results? Use appropriate evaluation metrics to support your conclusions.

Compare the results of the k-Means clustering model from part (f) to at least one other clustering algorithm. Which algorithm produces the best clustering? Use suitable evaluation metrics to justify your answer.

[12]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Load the dataset
file_path = 'car_sales_data_24.csv'
car_sales_data = pd.read_csv(file_path)

# Selecting numerical features
numerical_features = ['Engine size', 'Year of manufacture', 'Mileage', 'Price']

# Scale numerical features
scaler = StandardScaler()
scaled_features = scaler.fit_transform(car_sales_data[numerical_features])

# Function to perform k-Means clustering and evaluate using Elbow method and␣
↪Silhouette Score
def evaluate_kmeans(features, feature_names, max_k=10):
```

```python
    inertia = []
    silhouette_scores = []

    for k in range(2, max_k+1):
        kmeans = KMeans(n_clusters=k, random_state=42)
        kmeans.fit(features)
        inertia.append(kmeans.inertia_)
        silhouette_scores.append(silhouette_score(features, kmeans.labels_))

    # Plot the Elbow method
    plt.figure(figsize=(7, 6))
    plt.plot(range(2, max_k+1), inertia, marker='o')
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Inertia')
    plt.title(f'Elbow Method - {", ".join(feature_names)}')
    plt.savefig(f'Elbow_Method_{"_".join(feature_names)}.png')
    plt.close()

    # Plot the Silhouette Scores
    plt.figure(figsize=(7, 6))
    plt.plot(range(2, max_k+1), silhouette_scores, marker='o')
    plt.xlabel('Number of clusters (k)')
    plt.ylabel('Silhouette Score')
    plt.title(f'Silhouette Scores - {", ".join(feature_names)}')
    plt.savefig(f'Silhouette_Scores_{"_".join(feature_names)}.png')
    plt.close()

    # Return the optimal number of clusters based on Silhouette Score
    optimal_k = np.argmax(silhouette_scores) + 2
    return optimal_k, max(silhouette_scores)

# Evaluate k-Means clustering for different feature combinations
combinations = [
    ['Engine size'],
    ['Year of manufacture'],
    ['Mileage'],
    ['Engine size', 'Year of manufacture'],
    ['Engine size', 'Mileage'],
    ['Year of manufacture', 'Mileage'],
    ['Engine size', 'Year of manufacture', 'Mileage']
]

results = []
for combo in combinations:
    features = scaler.fit_transform(car_sales_data[combo])
    optimal_k, best_silhouette = evaluate_kmeans(features, combo)
```

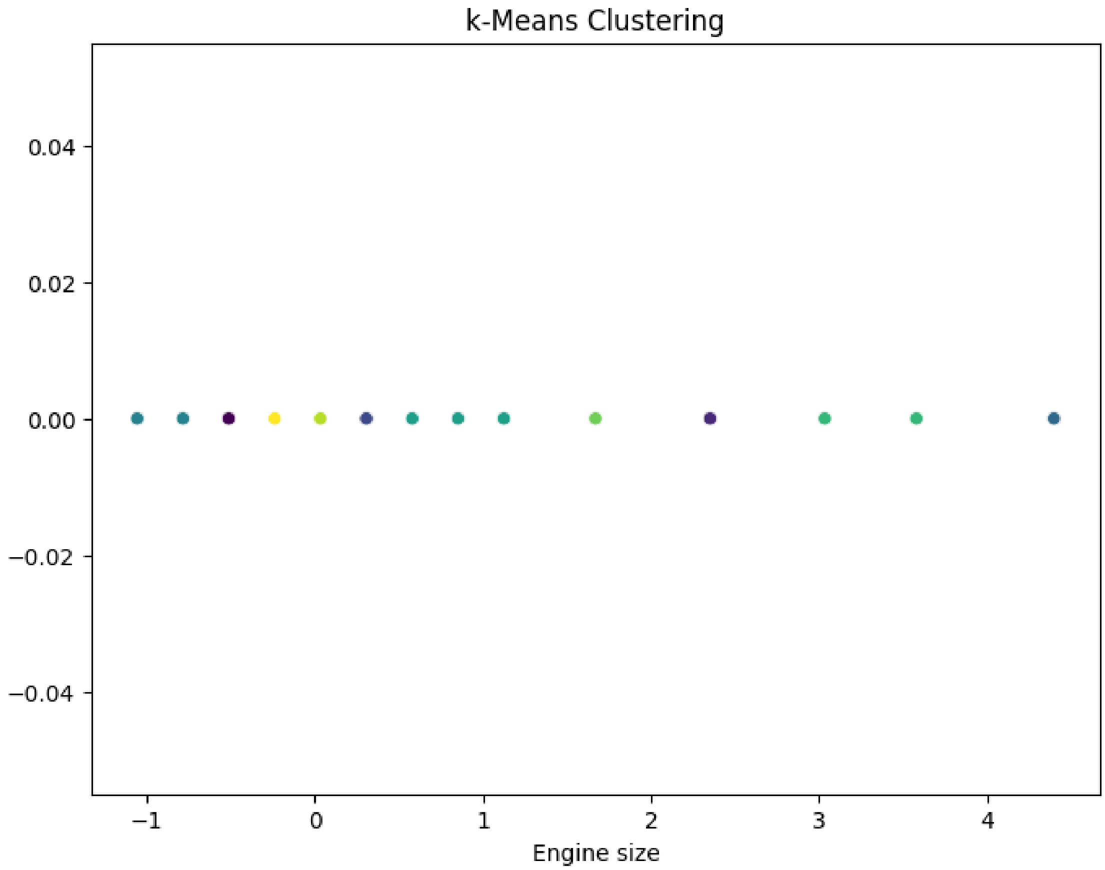
```
    results.append({'Features': combo, 'Optimal k': optimal_k, 'Best Silhouette␣
↪Score': best_silhouette})

# Create a DataFrame to display the results
results_df = pd.DataFrame(results)
print("k-Means Clustering Results:")
print(results_df)

# Determine the best combination of features based on Silhouette Score
best_combination = results_df.loc[results_df['Best Silhouette Score'].idxmax()]
print("\nBest Combination of Features for k-Means Clustering:")
print(best_combination)
```

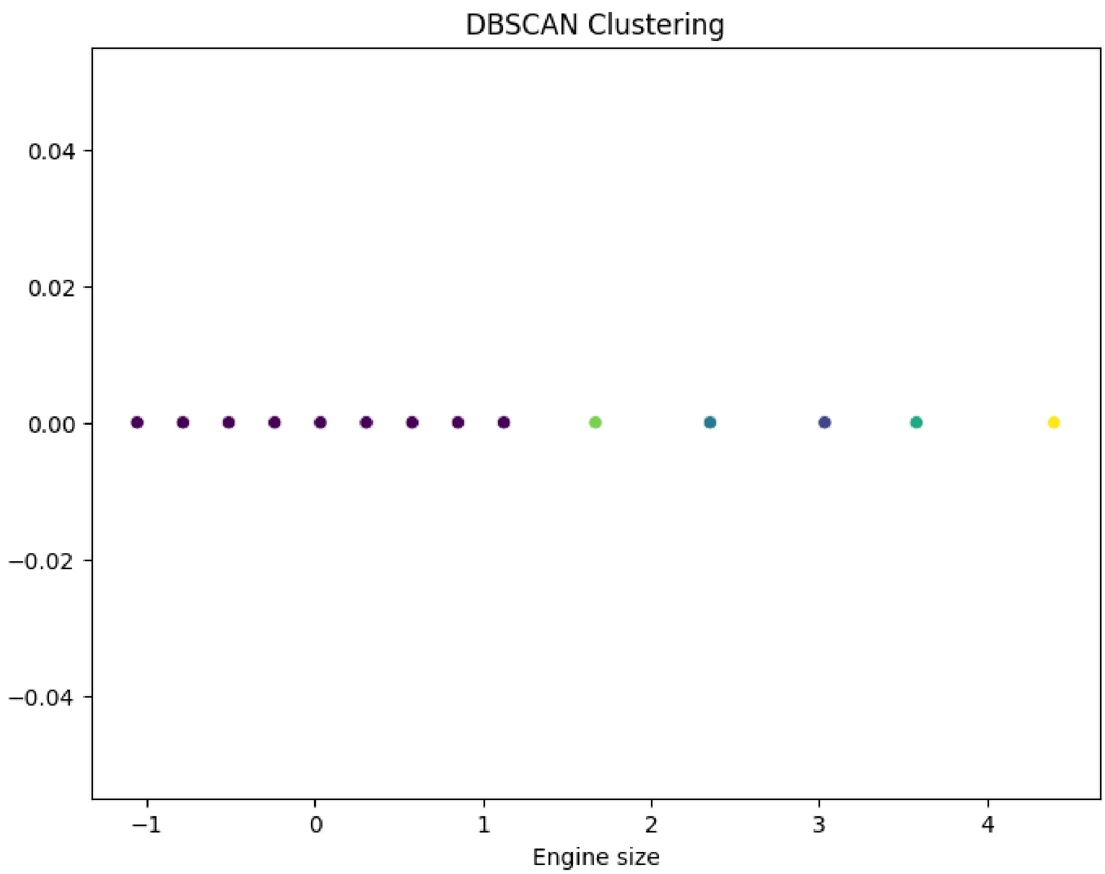
```
k-Means Clustering Results:
                                     Features  Optimal k  \
0                               [Engine size]         10
1                       [Year of manufacture]          2
2                                   [Mileage]          2
3          [Engine size, Year of manufacture]          3
4                      [Engine size, Mileage]          3
5              [Year of manufacture, Mileage]          2
6  [Engine size, Year of manufacture, Mileage]          3

   Best Silhouette Score
0               0.864135
1               0.619056
2               0.603861
3               0.459696
4               0.449840
5               0.533498
6               0.440399

Best Combination of Features for k-Means Clustering:
Features                 [Engine size]
Optimal k                           10
Best Silhouette Score         0.864135
Name: 0, dtype: object
```

[1]:
```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, DBSCAN
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt
import seaborn as sns

# Load the dataset
```

```
file_path = 'car_sales_data_24.csv'
car_sales_data = pd.read_csv(file_path)

# Selecting the best feature
best_feature = ['Engine size']

# Scale the numerical feature
scaler = StandardScaler()
scaled_feature = scaler.fit_transform(car_sales_data[best_feature])

# Perform k-Means clustering
optimal_k = 10
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_labels = kmeans.fit_predict(scaled_feature)
kmeans_silhouette = silhouette_score(scaled_feature, kmeans_labels)

# Perform DBSCAN clustering
dbscan = DBSCAN(eps=0.5, min_samples=5)
dbscan_labels = dbscan.fit_predict(scaled_feature)
# Filter out noise points for silhouette score calculation
dbscan_silhouette = silhouette_score(scaled_feature[dbscan_labels != -1],␣
↪dbscan_labels[dbscan_labels != -1])

# Plot the clusters for visual inspection
def plot_clusters(features, labels, algorithm_name):
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=features[:, 0], y=[0]*len(features), hue=labels,␣
↪palette='viridis', legend=None)
    plt.title(f'{algorithm_name} Clustering')
    plt.xlabel(best_feature[0])
    plt.ylabel('')
    plt.show()

# Plotting k-Means clusters
plot_clusters(scaled_feature, kmeans_labels, 'k-Means')

# Plotting DBSCAN Clusters
plot_clusters(scaled_feature, dbscan_labels, 'DBSCAN')

# Print the evaluation metrics
print("Clustering Evaluation Metrics:")
print(f"k-Means Silhouette Score: {kmeans_silhouette}")
print(f"DBSCAN Silhouette Score (excluding noise): {dbscan_silhouette}")

# Compare the results
if kmeans_silhouette > dbscan_silhouette:
    print("k-Means produces better clustering based on Silhouette Score.")
```

```python
else:
    print("DBSCAN produces better clustering based on Silhouette Score.")
```



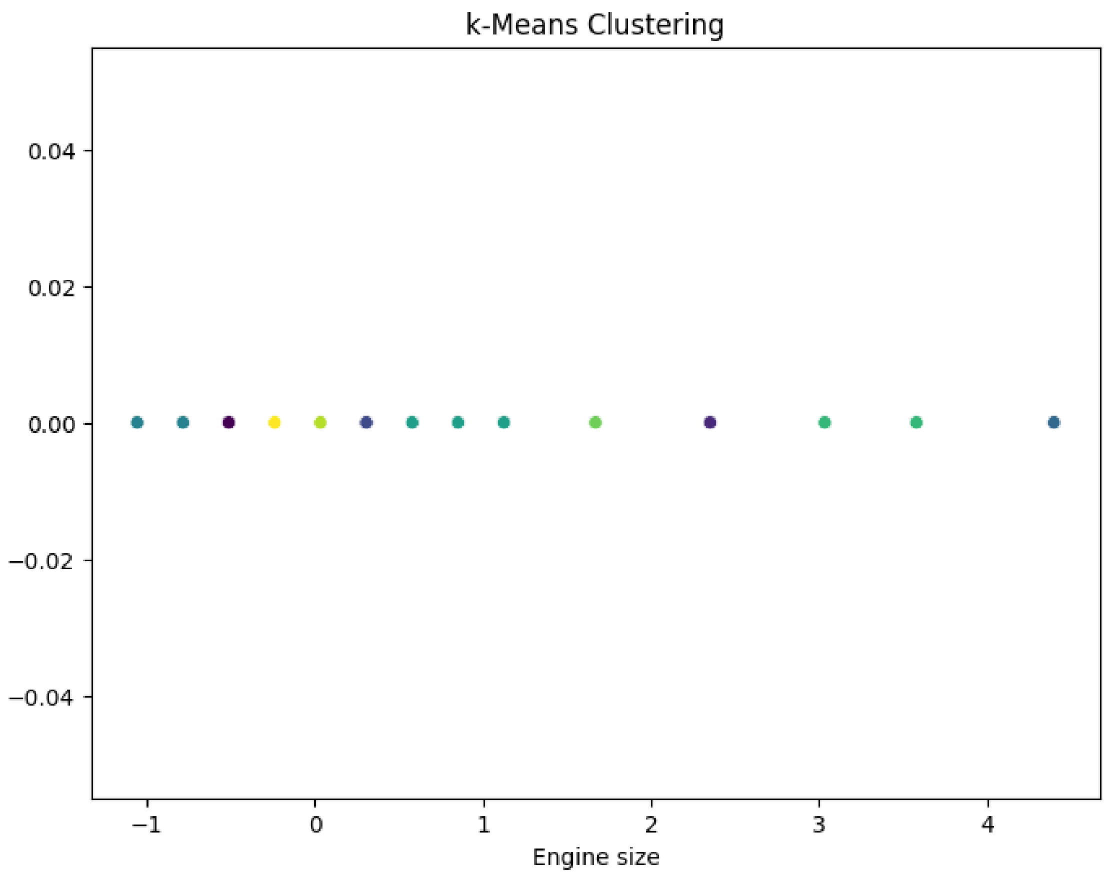


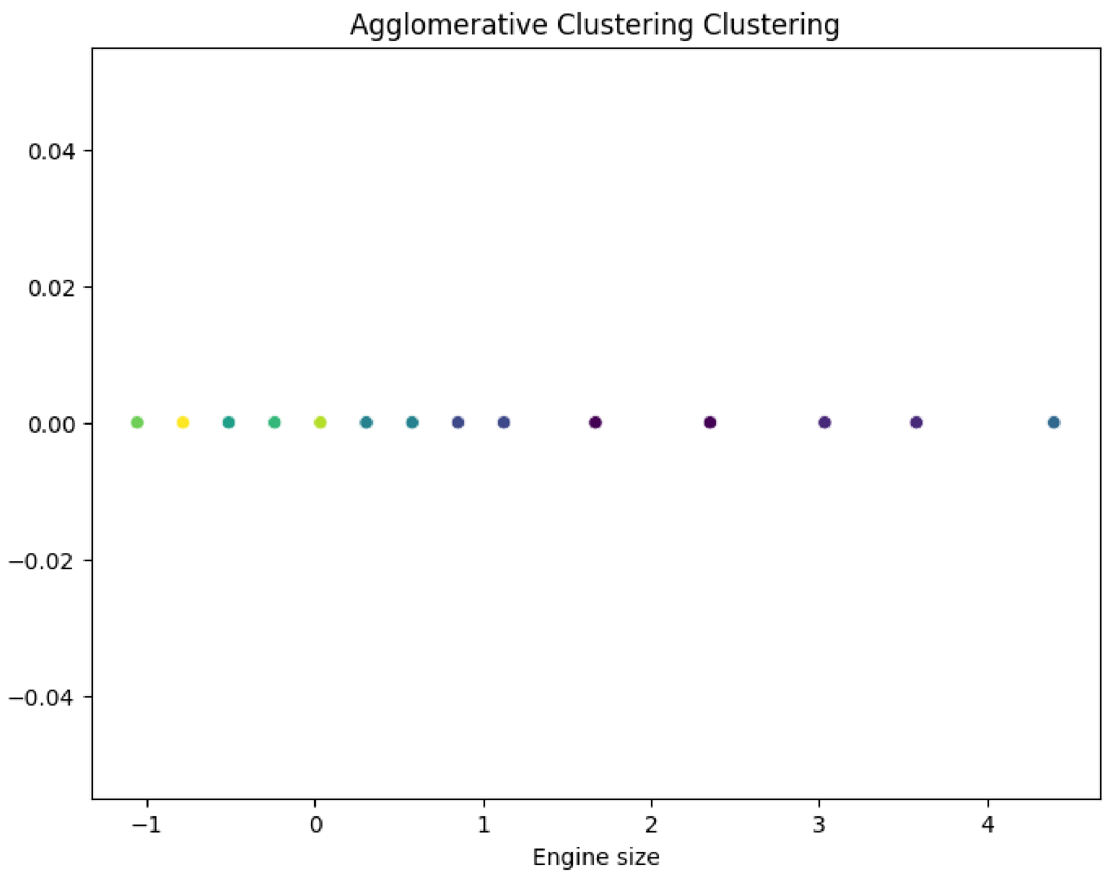

```
Clustering Evaluation Metrics:
k-Means Silhouette Score: 0.8641345201035432
DBSCAN Silhouette Score (excluding noise): 0.6506188805120912
k-Means produces better clustering based on Silhouette Score.
```

[2]:

```python
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans, AgglomerativeClustering
from sklearn.metrics import silhouette_score
import matplotlib.pyplot as plt

# Scale the numerical feature
scaler = StandardScaler()
scaled_feature = scaler.fit_transform(car_sales_data[best_feature])

# Perform k-Means clustering
optimal_k = 10
kmeans = KMeans(n_clusters=optimal_k, random_state=42)
kmeans_labels = kmeans.fit_predict(scaled_feature)
kmeans_silhouette = silhouette_score(scaled_feature, kmeans_labels)
```

```python
# Perform Agglomerative Clustering
agglomerative = AgglomerativeClustering(n_clusters=optimal_k)
agglomerative_labels = agglomerative.fit_predict(scaled_feature)
agglomerative_silhouette = silhouette_score(scaled_feature,␣
↪agglomerative_labels)

# Plot the clusters for visual inspection
def plot_clusters(features, labels, algorithm_name):
    plt.figure(figsize=(8, 6))
    sns.scatterplot(x=features[:, 0], y=[0]*len(features), hue=labels,␣
↪palette='viridis', legend=None)
    plt.title(f'{algorithm_name} Clustering')
    plt.xlabel(best_feature[0])
    plt.ylabel('')
    plt.show()

# Plotting k-Means clusters
plot_clusters(scaled_feature, kmeans_labels, 'k-Means')

# Plotting Agglomerative Clustering clusters
plot_clusters(scaled_feature, agglomerative_labels, 'Agglomerative Clustering')

# Print the evaluation metrics
print("Clustering Evaluation Metrics:")
print(f"k-Means Silhouette Score: {kmeans_silhouette}")
print(f"Agglomerative Clustering Silhouette Score: {agglomerative_silhouette}")

# Compare the results
if kmeans_silhouette > agglomerative_silhouette:
    print("k-Means produces better clustering based on Silhouette Score.")
else:
    print("Agglomerative Clustering produces better clustering based on␣
↪Silhouette Score.")
```


k-Means Clustering
0.04
0.02
0.00
−0.02
−0.04
−1
0
1
2
3
4
Engine size




```
Clustering Evaluation Metrics:
k-Means Silhouette Score: 0.8641345201035432
Agglomerative Clustering Silhouette Score: 0.922945637746698
Agglomerative Clustering produces better clustering based on Silhouette Score.
```

[ ]: