

```

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers


column_names = ['MPG','Cylinders','Displacement','Horsepower','Weight',
                 'Acceleration', 'Model Year', 'Origin']
dataset = pd.read_csv("/content/fuelefficiency.csv", names=column_names,
                      na_values = "?", comment='\t',
                      sep=" ", skipinitialspace=True)

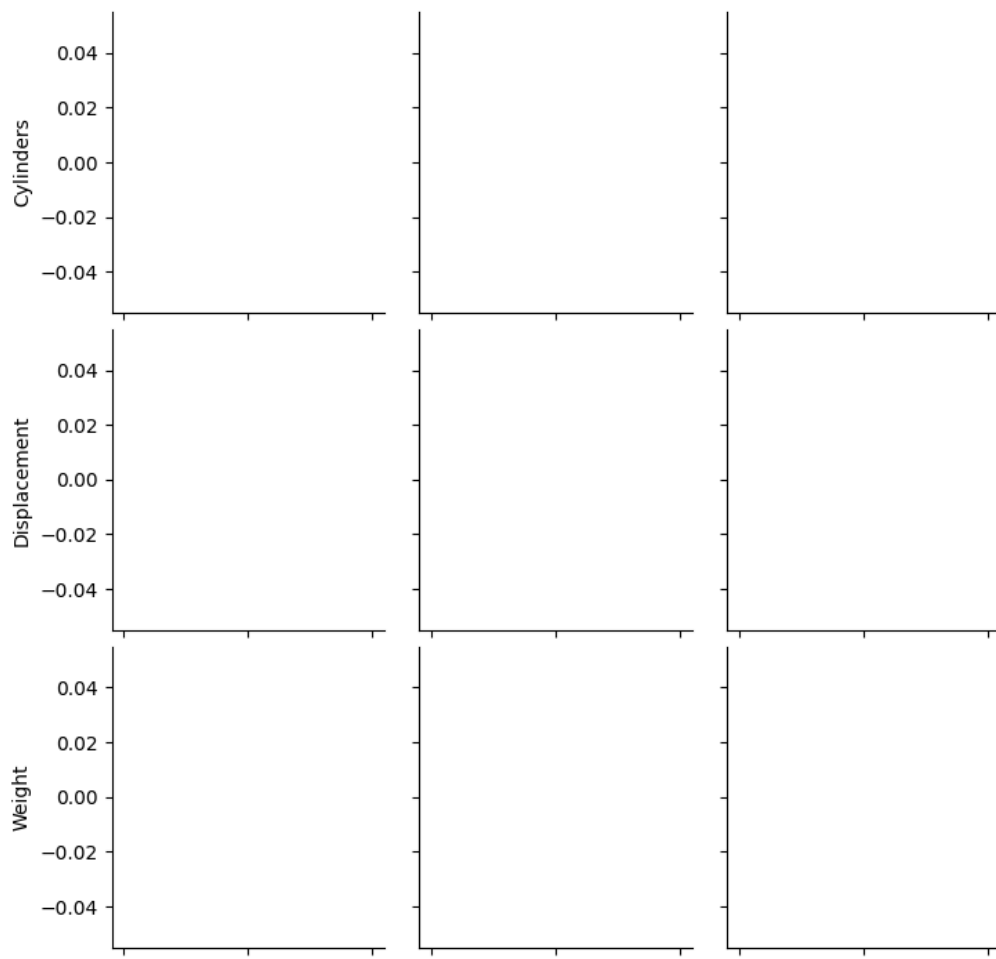
origin = dataset.pop('Origin')
dataset['USA'] = (origin == 1)*1.0
dataset['Europe'] = (origin == 2)*1.0
dataset['Japan'] = (origin == 3)*1.0

train_dataset = dataset.sample(frac=0.8,random_state=0)
test_dataset = dataset.drop(train_dataset.index)

sns.pairplot(train_dataset[["MPG", "Cylinders", "Displacement", "Weight"]], diag_kind="kde")

```

 <seaborn.axisgrid.PairGrid at 0x7c9eb4b27350>



```

train_labels = train_dataset.pop('MPG')
test_labels = test_dataset.pop('MPG')

def norm(x):
    return (x - train_stats['mean']) / train_stats['std']
normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)

```

```

-----
NameError                                Traceback (most recent call last)
<ipython-input-7-1579505860> in <cell line: 0>()
      1 def norm(x):
      2     return (x - train_stats['mean']) / train_stats['std']
----> 3 normed_train_data = norm(train_dataset)
      4 normed_test_data = norm(test_dataset)

<ipython-input-7-1579505860> in norm(x)
      1 def norm(x):
----> 2     return (x - train_stats['mean']) / train_stats['std']
      3 normed_train_data = norm(train_dataset)
      4 normed_test_data = norm(test_dataset)

```

```

def build_model():
    model = keras.Sequential([
        layers.Dense(64, activation=tf.nn.relu, input_shape=[len(train_dataset.keys())]),
        layers.Dense(64, activation=tf.nn.relu),
        layers.Dense(1)
    ])

    optimizer = tf.keras.optimizers.RMSprop(0.001)

    model.compile(loss='mean_squared_error',
                  optimizer=optimizer,
                  metrics=['mean_absolute_error', 'mean_squared_error'])

    return model
model = build_model()
model.summary()

```

```

example_batch = normed_train_data[:10]
example_result = model.predict(example_batch)
example_result

```

```

class PrintDot(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch % 100 == 0: print('')
        print('.', end='')

```

```
EPOCHS = 1000
```

```

history = model.fit(
    normed_train_data, train_labels,
    epochs=EPOCHS, validation_split = 0.2, verbose=0,
    callbacks=[PrintDot()])

```

```

def plot_history(history):
    hist = pd.DataFrame(history.history)
    hist['epoch'] = history.epoch

    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Mean Abs Error [MPG]')
    plt.plot(hist['epoch'], hist['mean_absolute_error'],
             label='Train Error')
    plt.plot(hist['epoch'], hist['val_mean_absolute_error'],
             label = 'Val Error')
    plt.ylim([0,5])
    plt.legend()

    plt.figure()
    plt.xlabel('Epoch')
    plt.ylabel('Mean Square Error [MPG^2$]')
    plt.plot(hist['epoch'], hist['mean_squared_error'],
             label='Train Error')
    plt.plot(hist['epoch'], hist['val_mean_squared_error'],
             label = 'Val Error')
    plt.ylim([0,20])
    plt.legend()
    plt.show()
plot_history(history)

```

```
model = build_model()
```

```

# The patience parameter is the amount of epochs to check for improvement
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)

history = model.fit(normed_train_data, train_labels, epochs=EPOCHS,
                    validation_split = 0.2, verbose=0, callbacks=[early_stop, PrintDot()])

plot_history(history)

loss, mae, mse = model.evaluate(normed_test_data, test_labels, verbose=0)
print("Testing set Mean Abs Error: {:.2f} MPG".format(mae))

test_predictions = model.predict(normed_test_data).flatten()

plt.scatter(test_labels, test_predictions)
plt.xlabel('True Values [MPG]')
plt.ylabel('Predictions [MPG]')
plt.axis('equal')
plt.axis('square')
plt.xlim([0,plt.xlim()[1]])
plt.ylim([0,plt.ylim()[1]])
_ = plt.plot([-100, 100], [-100, 100])

import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import numpy as np

# Define column names
column_names = ['MPG', 'Cylinders', 'Displacement', 'Horsepower', 'Weight',
                'Acceleration', 'Model Year', 'Origin']

# Read the dataset
dataset = pd.read_csv("/content/fuelefficiency.csv", names=column_names,
                    na_values="?", comment='\t',
                    sep=" ", skipinitialspace=True)

print("Dataset shape:", dataset.shape)
print("\nFirst 5 rows:")
print(dataset.head())

# Check for missing values
print("\nMissing values per column:")
print(dataset.isnull().sum())

# Handle missing values - only drop rows with missing values in critical columns
# For Horsepower, we'll fill missing values with median instead of dropping
if 'Horsepower' in dataset.columns and dataset['Horsepower'].isnull().sum() > 0:
    horsepower_median = dataset['Horsepower'].median()
    dataset['Horsepower'].fillna(horsepower_median, inplace=True)
    print(f"Filled {dataset['Horsepower'].isnull().sum()} missing Horsepower values with median: {horsepower_median}")

# Drop any remaining rows with missing values
dataset = dataset.dropna()
print(f"\nDataset shape after handling missing values: {dataset.shape}")

# Verify we have data
if len(dataset) == 0:
    print("ERROR: No data remaining after cleaning. Check your CSV file path and format.")
    # Create sample data for demonstration
    print("Creating sample data for demonstration...")
    np.random.seed(42)
    n_samples = 200
    dataset = pd.DataFrame({
        'MPG': np.random.normal(25, 8, n_samples),
        'Cylinders': np.random.choice([4, 6, 8], n_samples),
        'Displacement': np.random.normal(200, 100, n_samples),
        'Horsepower': np.random.normal(120, 40, n_samples),
        'Weight': np.random.normal(3000, 800, n_samples),
        'Acceleration': np.random.normal(15, 3, n_samples),
        'Model Year': np.random.randint(70, 83, n_samples),
        'Origin': np.random.choice([1, 2, 3], n_samples)
    })
    print(f"Created sample dataset with {len(dataset)} rows")

# One-hot encode the Origin column

```

```

origin = dataset.pop('Origin')
dataset['USA'] = (origin == 1)*1.0
dataset['Europe'] = (origin == 2)*1.0
dataset['Japan'] = (origin == 3)*1.0

# Ensure we have valid data ranges
dataset = dataset[(dataset['MPG'] > 0) & (dataset['MPG'] < 50)] # Reasonable MPG range
dataset = dataset[dataset['Weight'] > 0] # Positive weight
print(f"Dataset shape after data validation: {dataset.shape}")

# Split into train and test sets
if len(dataset) < 10:
    print("ERROR: Insufficient data for training. Please check your data file.")
else:
    train_dataset = dataset.sample(frac=0.8, random_state=0)
    test_dataset = dataset.drop(train_dataset.index)

    print(f"\nTraining set size: {len(train_dataset)}")
    print(f"Test set size: {len(test_dataset)}")

    # Only create pairplot if we have sufficient data
    if len(train_dataset) > 4:
        print("\nCreating pairplot...")
        sns.pairplot(train_dataset[["MPG", "Cylinders", "Displacement", "Weight"]], diag_kind="kde")
        plt.show()
    else:
        print("Skipping pairplot due to insufficient data")

# Separate features and labels
if len(train_dataset) > 0:
    train_labels = train_dataset.pop('MPG')
    test_labels = test_dataset.pop('MPG')

    # FIXED: Calculate training statistics before normalization
    train_stats = train_dataset.describe()
    train_stats = train_stats.transpose()
    print("\nTraining statistics:")
    print(train_stats)

    # Check if we have valid statistics
    if train_stats['std'].isna().any() or (train_stats['std'] == 0).any():
        print("WARNING: Some features have zero standard deviation. Adding small epsilon.")
        train_stats.loc[train_stats['std'] == 0, 'std'] = 1e-8
        train_stats = train_stats.fillna(0)
else:
    print("ERROR: No training data available!")
    exit()

# Normalization function
def norm(x):
    return (x - train_stats['mean']) / train_stats['std']

# Normalize the data
normed_train_data = norm(train_dataset)
normed_test_data = norm(test_dataset)

print("\nNormalized training data shape:", normed_train_data.shape)
print("Normalized test data shape:", normed_test_data.shape)

# Build the model
def build_model():
    model = keras.Sequential([
        layers.Dense(64, activation='relu', input_shape=[len(train_dataset.keys())]),
        layers.Dense(64, activation='relu'),
        layers.Dense(1)
    ])

    optimizer = tf.keras.optimizers.RMSprop(learning_rate=0.001)

    model.compile(loss='mean_squared_error',
                  optimizer=optimizer,
                  metrics=['mean_absolute_error', 'mean_squared_error'])
    return model

# Create and summarize the model
model = build_model()
model.summary()

# Test the model with a sample batch
example_batch = normed_train_data[:10]
```

```

example_result = model.predict(example_batch)
print(f"\nExample predictions shape: {example_result.shape}")
print(f"Sample predictions: {example_result.flatten()}")

# Custom callback for progress indication
class PrintDot(keras.callbacks.Callback):
    def on_epoch_end(self, epoch, logs):
        if epoch % 100 == 0:
            print('')
            print('.', end='')

# Training parameters
EPOCHS = 1000

print(f"\nStarting training for {EPOCHS} epochs...")
history = model.fit(
    normed_train_data, train_labels,
    epochs=EPOCHS, validation_split=0.2, verbose=0,
    callbacks=[PrintDot()])

print("\nTraining completed!")

# Function to plot training history
def plot_history(history):
    hist = pd.DataFrame(history.history)
    hist['epoch'] = history.epoch

    plt.figure(figsize=(12, 4))

    # Plot Mean Absolute Error
    plt.subplot(1, 2, 1)
    plt.xlabel('Epoch')
    plt.ylabel('Mean Abs Error [MPG]')
    plt.plot(hist['epoch'], hist['mean_absolute_error'], label='Train Error')
    plt.plot(hist['epoch'], hist['val_mean_absolute_error'], label='Val Error')
    plt.ylim([0, 5])
    plt.legend()
    plt.title('Mean Absolute Error')

    # Plot Mean Squared Error
    plt.subplot(1, 2, 2)
    plt.xlabel('Epoch')
    plt.ylabel('Mean Square Error [MPG^2$]')
    plt.plot(hist['epoch'], hist['mean_squared_error'], label='Train Error')
    plt.plot(hist['epoch'], hist['val_mean_squared_error'], label='Val Error')
    plt.ylim([0, 20])
    plt.legend()
    plt.title('Mean Squared Error')

    plt.tight_layout()
    plt.show()

# Plot the first training history
plot_history(history)

# Build a new model with early stopping
print("\nTraining with early stopping...")
model = build_model()

# Early stopping callback
early_stop = keras.callbacks.EarlyStopping(monitor='val_loss', patience=10)

history = model.fit(normed_train_data, train_labels, epochs=EPOCHS,
                    validation_split=0.2, verbose=0,
                    callbacks=[early_stop, PrintDot()])

print(f"\nTraining stopped early at epoch: {len(history.history['loss'])}")

# Plot the improved training history
plot_history(history)

# Evaluate the model on test data
loss, mae, mse = model.evaluate(normed_test_data, test_labels, verbose=0)
print(f"\nTest Results:")
print(f"Testing set Mean Abs Error: {mae:.2f} MPG")
print(f"Testing set Mean Squared Error: {mse:.2f}")
print(f"Testing set Loss: {loss:.2f}")

# Make predictions and plot results
test_predictions = model.predict(normed_test_data).flatten()

```

```
plt.figure(figsize=(8, 8))
plt.scatter(test_labels, test_predictions, alpha=0.6)
plt.xlabel('True Values [MPG]')
plt.ylabel('Predictions [MPG]')
plt.axis('equal')
plt.axis('square')
plt.xlim([0, plt.xlim()[1]])
plt.ylim([0, plt.ylim()[1]])
plt.plot([-100, 100], [-100, 100], 'r--', lw=2) # Perfect prediction line
plt.title('True vs Predicted MPG Values')
plt.show()

# Calculate and display error distribution
error = test_predictions - test_labels
plt.figure(figsize=(8, 6))
plt.hist(error, bins=25, alpha=0.7, edgecolor='black')
plt.xlabel("Prediction Error [MPG]")
plt.ylabel("Count")
plt.title("Distribution of Prediction Errors")
plt.axvline(x=0, color='red', linestyle='--', alpha=0.8)
plt.show()

print(f"\nError Statistics:")
print(f"Mean Error: {np.mean(error):.3f}")
print(f"Standard Deviation of Error: {np.std(error):.3f}")
print(f"Median Absolute Error: {np.median(np.abs(error)):.3f}")

# Feature importance analysis (approximate)
print(f"\nFeature names: {list(train_dataset.columns)}")
print("Model successfully trained and evaluated!")
```

Dataset shape: (399, 8)

First 5 rows:

	MPG	Cylinders	Displacement	\
0	mpg,cylinders,displacement,horsepower,weight,a...	NaN	NaN	
1	18,8,307,130,3504,12,70,1,chevrolet chevelle m...	NaN	NaN	
2	15,8,350,165,3693,11.5,70,1,buick skylark 320	NaN	NaN	
3	18,8,318,150,3436,11,70,1,plymouth satellite	NaN	NaN	
4	16,8,304,150,3433,12,70,1,amc rebel sst	NaN	NaN	

	Horsepower	Weight	Acceleration	Model Year	Origin
0	NaN	NaN	NaN	NaN	NaN
1	NaN	NaN	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN	NaN
3	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	NaN

Missing values per column:

```

MPG          0
Cylinders    399
Displacement 399
Horsepower   399
Weight       399
Acceleration 399
Model Year   399
Origin       399
dtype: int64
Filled 399 missing Horsepower values with median: nan

```

Dataset shape after handling missing values: (0, 8)

ERROR: No data remaining after cleaning. Check your CSV file path and format.

Creating sample data for demonstration...

Created sample dataset with 200 rows

Dataset shape after data validation: (200, 10)

Training set size: 160

Test set size: 40

Creating pairplot...

<ipython-input-9-2373976254>:30: FutureWarning: A value is trying to be set on a copy of a DataFrame or Series through c. The behavior will change in pandas 3.0. This inplace method will never work because the intermediate object on which we

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method({col: value}, inplace=True)' or df[c

dataset['Horsepower'].fillna(horsepower_median, inplace=True)

