Title: Data Efficiency Matters: Performance from 10% → 100% Training Size

Models: Linear Regression, Decision Tree, and Neural Network (MLP) bold text

Dataset: UCI Abalone

```
# ## Step 1: Import Libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

import warnings
warnings.filterwarnings('ignore')
```

```
# ## Step 2: Load Dataset

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/abalone/abalone.data'
columns = ['Sex','Length','Diameter','Height','WholeWeight','ShuckedWeight','VisceraWeight','ShellWeight','Rings']
df = pd.read_csv(url, names=columns)
```

```
print("Dataset shape:", df.shape)
df.head()
```

```
Dataset shape: (4177, 9)
```

	Sex	Length	Diameter	Height	WholeWeight	ShuckedWeight	VisceraWeight	ShellWeight	Rings
0	M	0.455	0.365	0.095	0.5140	0.2245	0.1010	0.150	15
1	M	0.350	0.265	0.090	0.2255	0.0995	0.0485	0.070	7
2	F	0.530	0.420	0.135	0.6770	0.2565	0.1415	0.210	9
3	M	0.440	0.365	0.125	0.5160	0.2155	0.1140	0.155	10
4	I	0.330	0.255	0.080	0.2050	0.0895	0.0395	0.055	7

```
# ## Step 3: Basic Information & Descriptive Statistics

print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4177 entries, 0 to 4176
Data columns (total 9 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Sex            4177 non-null   object
 1   Length         4177 non-null   float64
 2   Diameter       4177 non-null   float64
 3   Height         4177 non-null   float64
 4   WholeWeight    4177 non-null   float64
 5   ShuckedWeight  4177 non-null   float64
 6   VisceraWeight  4177 non-null   float64
 7   ShellWeight    4177 non-null   float64
 8   Rings          4177 non-null   int64
dtypes: float64(7), int64(1), object(1)
memory usage: 293.8+ KB
None
```

```
print(df.describe())
```

```
           Length     Diameter       Height  WholeWeight  ShuckedWeight  \
count  4177.000000  4177.000000  4177.000000  4177.000000    4177.000000
mean      0.523992     0.407881     0.139516     0.828742       0.359367
std       0.120093     0.099240     0.041827     0.490389       0.221963
min       0.075000     0.055000     0.000000     0.002000       0.001000
25%       0.450000     0.350000     0.115000     0.441500       0.186000
50%       0.545000     0.425000     0.140000     0.799500       0.336000
75%       0.615000     0.480000     0.165000     1.153000       0.502000
max       0.815000     0.650000     1.130000     2.825500       1.488000

       VisceraWeight  ShellWeight        Rings
count    4177.000000  4177.000000  4177.000000
```

```
mean        0.180594     0.238831     9.933684
std         0.109614     0.139203     3.224169
min         0.000500     0.001500     1.000000
25%         0.093500     0.130000     8.000000
50%         0.171000     0.234000     9.000000
75%         0.253000     0.329000    11.000000
max         0.760000     1.005000    29.000000
```

```
print("\nMissing Values:\n", df.isnull().sum())
```

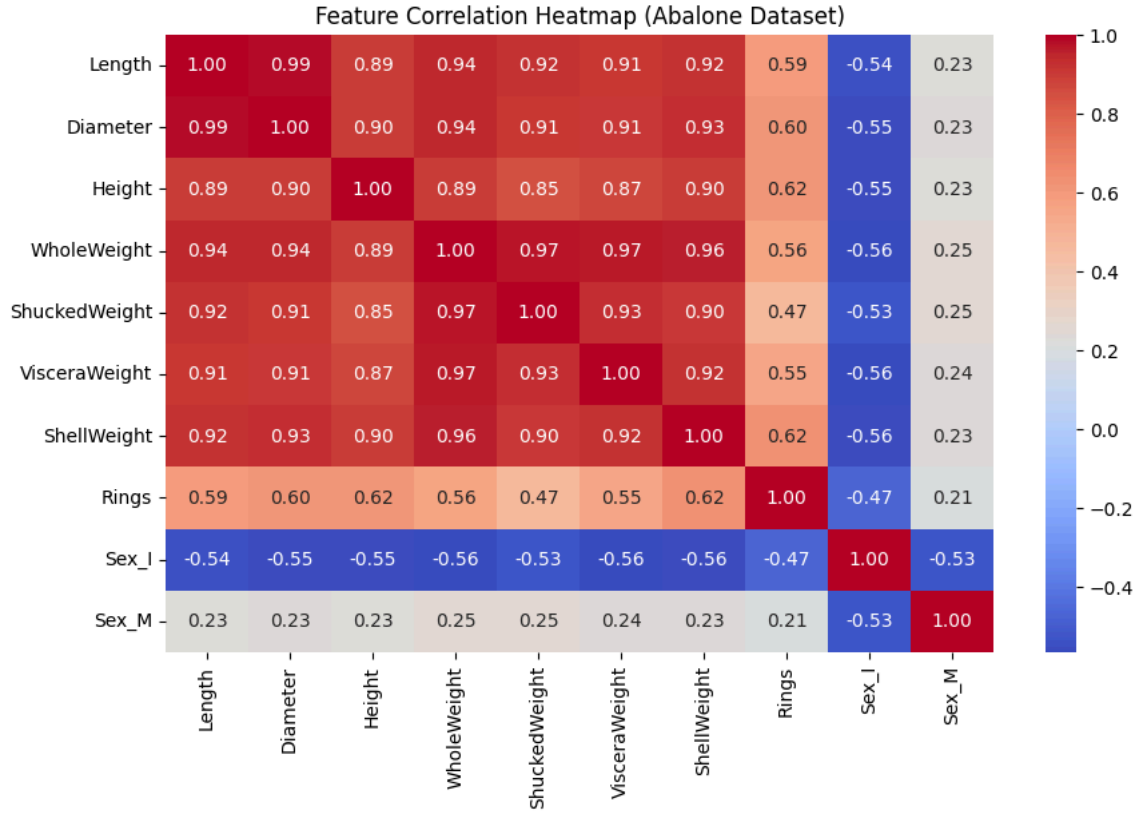
```
Missing Values:
 Sex              0
Length           0
Diameter         0
Height           0
WholeWeight      0
ShuckedWeight    0
VisceraWeight    0
ShellWeight      0
Rings            0
dtype: int64
```

```
# ## Step 4: Handle Categorical Feature & Scaling

df = pd.get_dummies(df, columns=['Sex'], drop_first=True)
X = df.drop('Rings', axis=1)
y = df['Rings']

scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

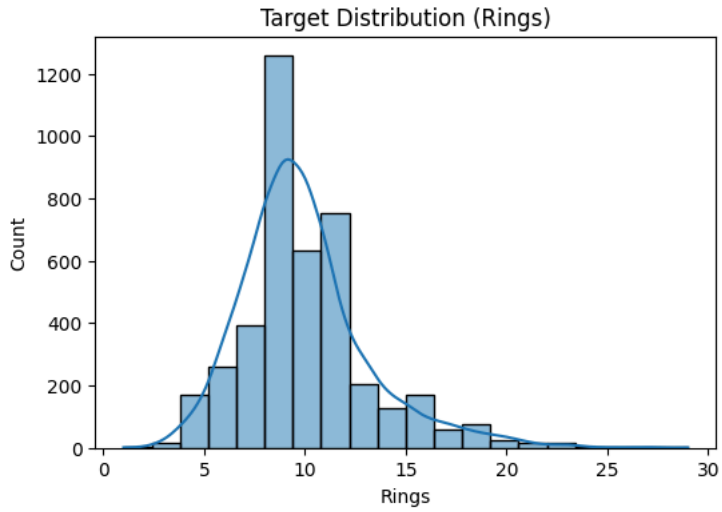
```
# ## Step 5: Outlier Detection (IQR Method)

print("\nOutlier Detection using IQR Method:")

# Select only numeric columns for IQR
numeric_cols = df.select_dtypes(include=['float64', 'int64']).columns

# Calculate IQR
Q1 = df[numeric_cols].quantile(0.25)
Q3 = df[numeric_cols].quantile(0.75)
IQR = Q3 - Q1
#Define bounds
lower_bound = Q1 - 1.5 * IQR
upper_bound = Q3 + 1.5 * IQR

# Keep only rows within bounds
df_clean = df[~((df[numeric_cols] < lower_bound) | (df[numeric_cols] > upper_bound)).any(axis=1)]

print(f"Original shape: {df.shape}")
print(f"After removing outliers: {df_clean.shape}")
print(f"Rows removed: {df.shape[0] - df_clean.shape[0]}")
```

```
Outlier Detection using IQR Method:
Original shape: (4177, 10)
After removing outliers: (3781, 10)
Rows removed: 396
```

```
# ## Step 6: Data Visualization

# Correlation Heatmap
plt.figure(figsize=(10,6))
sns.heatmap(df_clean.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Feature Correlation Heatmap (Abalone Dataset)")
plt.show()
```

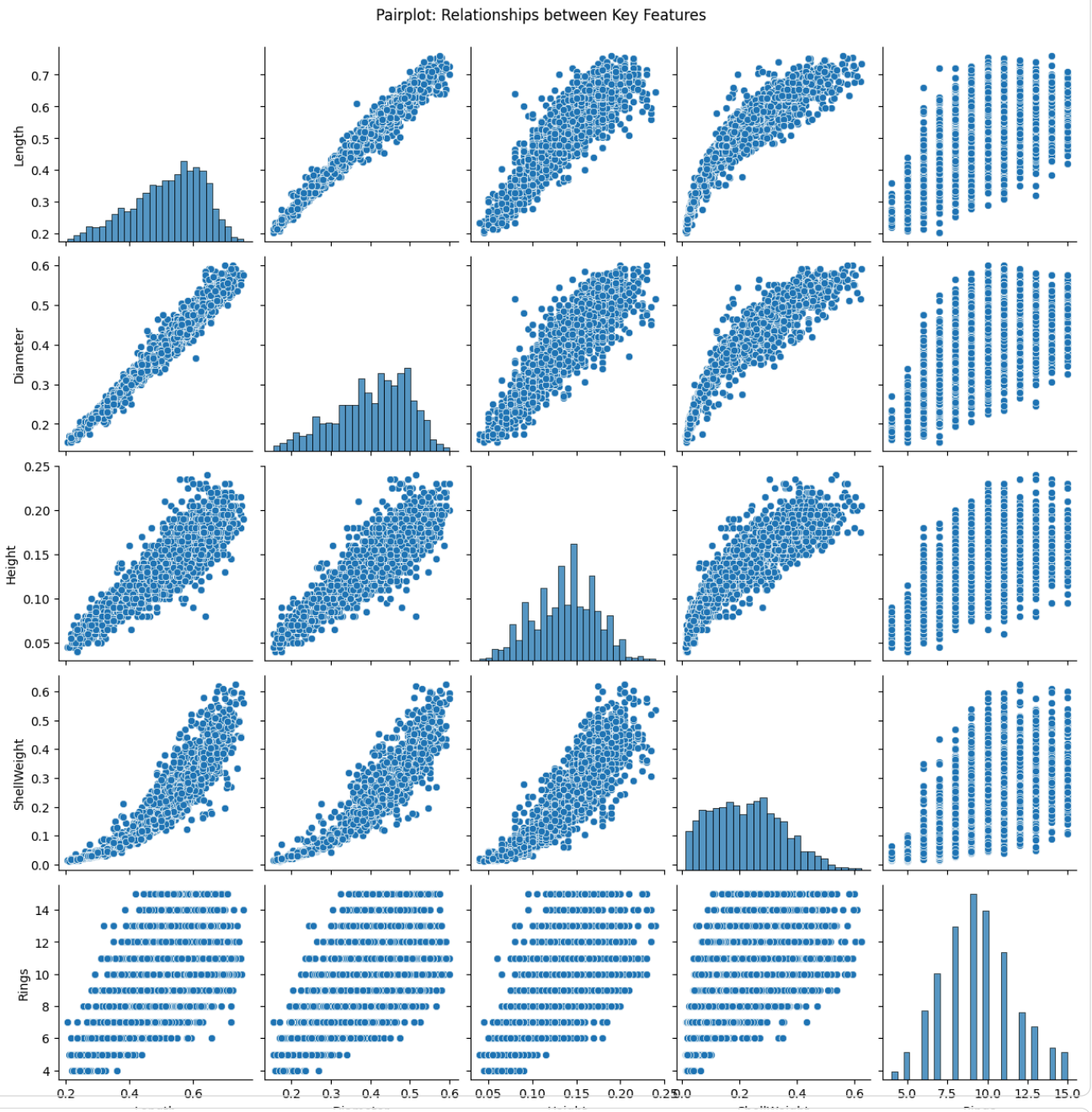



```
plt.figure(figsize=(6,4))
sns.histplot(df['Rings'], kde=True, bins=20)
plt.title('Target Distribution (Rings)')
plt.show()
```

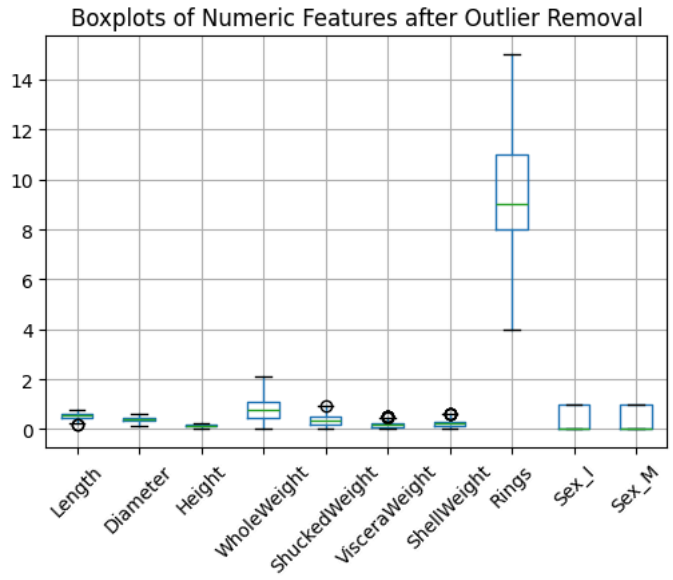



```
# Pairplot for a few important columns
plt.figure(figsize=(6,4))
sns.pairplot(df_clean[['Length', 'Diameter', 'Height', 'ShellWeight', 'Rings']])
plt.suptitle("Pairplot: Relationships between Key Features", y=1.02)
plt.show()
```

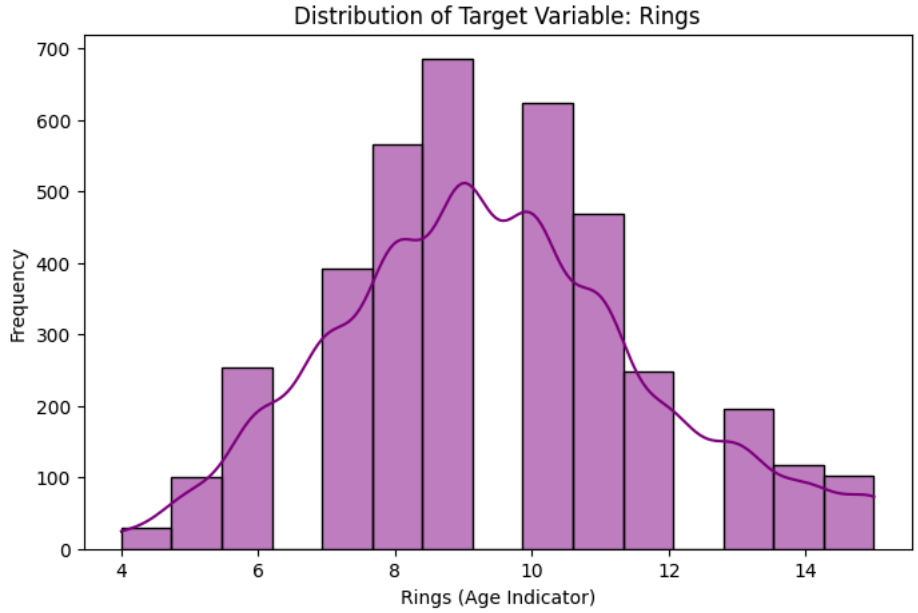
```
<Figure size 600x400 with 0 Axes>
```



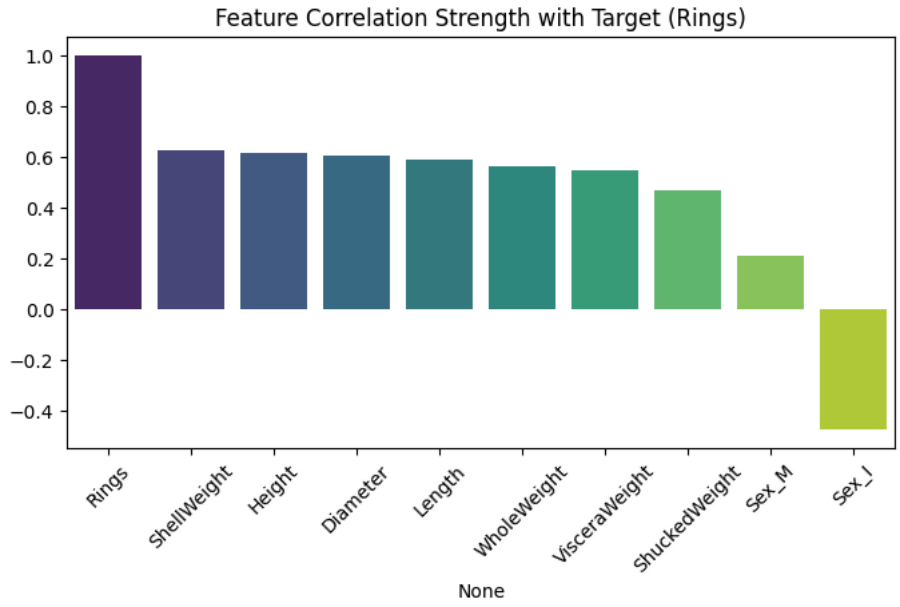

```
# Boxplots for numeric columns
plt.figure(figsize=(6,4))
df_clean.boxplot(rot=45)
plt.title("Boxplots of Numeric Features after Outlier Removal")
plt.show()
```




```
plt.figure(figsize=(8,5))
sns.histplot(df_clean['Rings'], bins=15, kde=True, color='purple')
plt.title("Distribution of Target Variable: Rings")
plt.xlabel("Rings (Age Indicator)")
plt.ylabel("Frequency")
plt.show()
```




```
corr_target = df_clean.corr()['Rings'].sort_values(ascending=False)
plt.figure(figsize=(8,4))
sns.barplot(x=corr_target.index, y=corr_target.values, palette='viridis')
plt.title("Feature Correlation Strength with Target (Rings)")
plt.xticks(rotation=45)
plt.show()
```




```
### Step 7: Train-Test Split

X = df.drop('Rings', axis=1)
y = df['Rings']
X_train_pool, X_test, y_train_pool, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("Train size:", X_train_pool.shape, "Test size:", X_test.shape)
```

```
Train size: (3341, 9) Test size: (836, 9)
```

```
# ## Step 8: Function to Evaluate Models
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

def evaluate_model(y_true, y_pred):
    # Compute metrics manually to avoid version issues
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)  # manually compute root MSE
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)
    return rmse, mae, r2
```

```
# Initialize models
lr = LinearRegression()
dt = DecisionTreeRegressor(random_state=42)
nn = MLPRegressor(hidden_layer_sizes=(64, 32), max_iter=1000, random_state=42)
```

```
# ✅ Step 9: Train models on different training sizes and collect results
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np

train_fractions = [0.1, 0.3, 0.5, 0.7, 1.0]

results_lr = []
results_dt = []
results_nn = []

for frac in train_fractions:
    # ✅ Handle 100% case separately (no split)
    if frac < 1.0:
        X_train_frac, _, y_train_frac, _ = train_test_split(X_train, y_train, train_size=frac, random_state=42)
    else:
        X_train_frac, y_train_frac = X_train, y_train

    # --- Linear Regression ---
    lr.fit(X_train_frac, y_train_frac)
    y_pred_lr = lr.predict(X_test)
    rmse_lr = np.sqrt(mean_squared_error(y_test, y_pred_lr))
    r2_lr = r2_score(y_test, y_pred_lr)
    results_lr.append((frac, rmse_lr, len(X_train_frac), r2_lr))

    # --- Decision Tree ---
    dt.fit(X_train_frac, y_train_frac)
```

```
    y_pred_dt = dt.predict(X_test)
    rmse_dt = np.sqrt(mean_squared_error(y_test, y_pred_dt))
    r2_dt = r2_score(y_test, y_pred_dt)
    results_dt.append((frac, rmse_dt, len(X_train_frac), r2_dt))

    # --- Neural Network ---
    nn.fit(X_train_frac, y_train_frac)
    y_pred_nn = nn.predict(X_test)
    rmse_nn = np.sqrt(mean_squared_error(y_test, y_pred_nn))
    r2_nn = r2_score(y_test, y_pred_nn)
    results_nn.append((frac, rmse_nn, len(X_train_frac), r2_nn))

# ✅ Print summary
print("✅ Linear Regression Results:", results_lr)
print("✅ Decision Tree Results:", results_dt)
print("✅ Neural Network Results:", results_nn)
```

```
.7, np.float64(2.2298345191197004), 2338, 0.5406867892879128), (1.0, np.float64(2.211613087121836), 3341, 0.5481628137889265)]
 np.float64(3.129386873008343), 2338, 0.09534668059620688), (1.0, np.float64(3.0439366352199992), 3341, 0.14407663220938294)]
0.7, np.float64(2.151770442364263), 2338, 0.5722839541083458), (1.0, np.float64(2.2058881038283484), 3341, 0.5504990382511511)]
```

```
# Convert results to DataFrames
df_lr = pd.DataFrame(results_lr, columns=['Training Fraction', 'RMSE', 'Samples Used', 'R²'])
df_dt = pd.DataFrame(results_dt, columns=['Training Fraction', 'RMSE', 'Samples Used', 'R²'])
df_nn = pd.DataFrame(results_nn, columns=['Training Fraction', 'RMSE', 'Samples Used', 'R²'])

# Display tables clearly
print("📊 Linear Regression Performance")
display(df_lr.style.background_gradient(cmap='Blues'))

print("\n🌳 Decision Tree Performance")
display(df_dt.style.background_gradient(cmap='Greens'))

print("\n🧠 Neural Network Performance")
display(df_nn.style.background_gradient(cmap='Reds'))
```

📊 Linear Regression Performance

	Training Fraction	RMSE	Samples Used	R²
0	0.100000	2.274272	334	0.522197
1	0.300000	2.257619	1002	0.529169
2	0.500000	2.244504	1670	0.534624
3	0.700000	2.229835	2338	0.540687
4	1.000000	2.211613	3341	0.548163

🌳 Decision Tree Performance

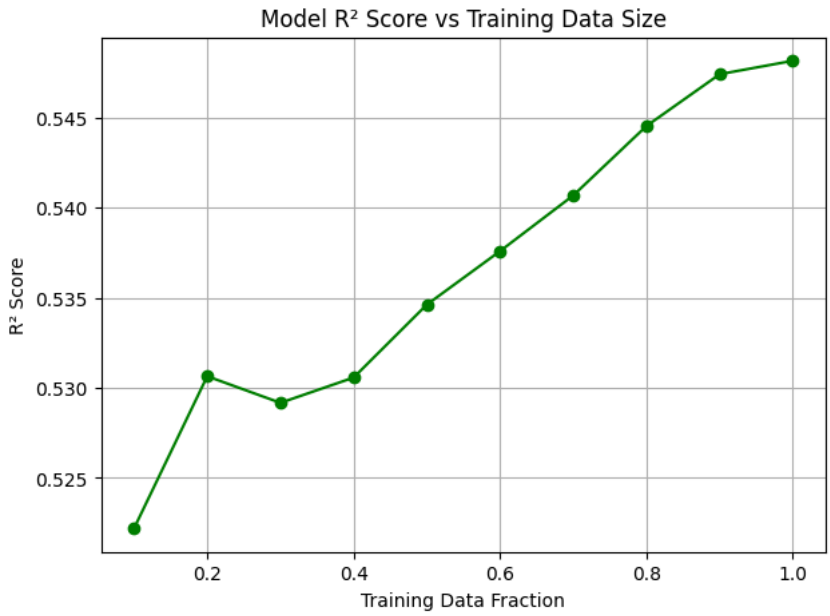
	Training Fraction	RMSE	Samples Used	R²
0	0.100000	3.093248	334	0.116120
1	0.300000	2.987614	1002	0.175458
2	0.500000	2.989215	1670	0.174574
3	0.700000	3.129387	2338	0.095347
4	1.000000	3.043937	3341	0.144077

🧠 Neural Network Performance

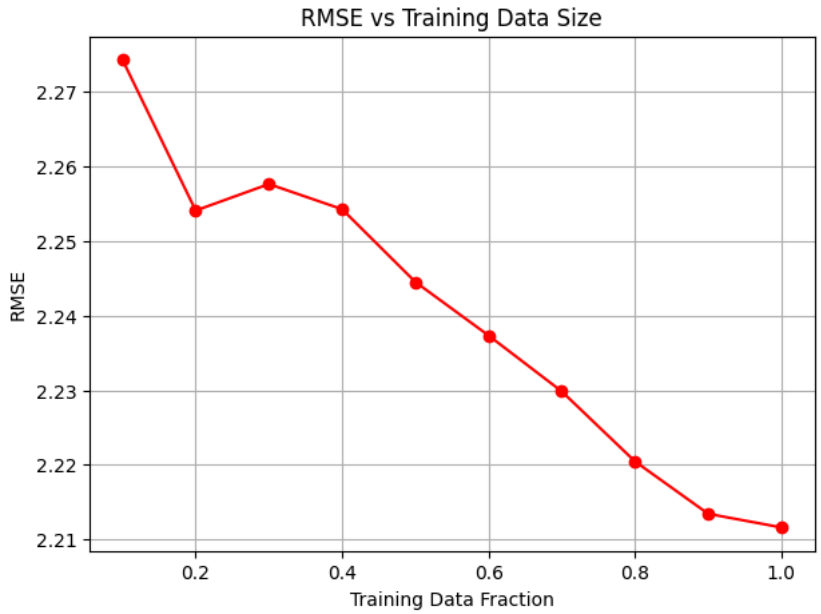
	Training Fraction	RMSE	Samples Used	R²
0	0.100000	2.221576	334	0.544083
1	0.300000	2.289673	1002	0.515704
2	0.500000	2.190666	1670	0.556681
3	0.700000	2.151770	2338	0.572284
4	1.000000	2.205888	3341	0.550499

```
# Step 10: Visualize Model Performance vs Training Size

# Convert results into a DataFrame for easy plotting
train_df = pd.DataFrame(results, columns=['Train Size', 'RMSE', 'MAE', 'R2'])

# --- Plot R² Score vs Training Size ---
plt.figure(figsize=(7,5))
```

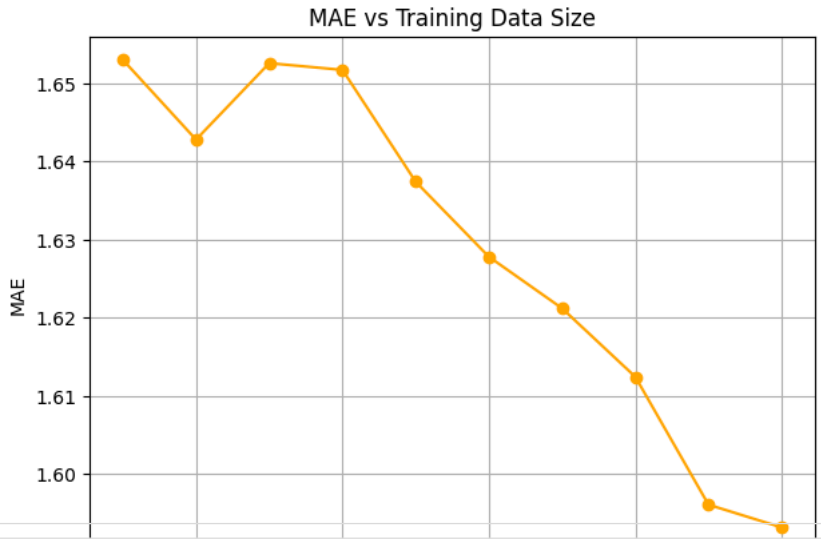
```
plt.plot(train_df['Train Size'], train_df['R2'], marker='o', color='green')
plt.title('Model R² Score vs Training Data Size')
plt.xlabel('Training Data Fraction')
plt.ylabel('R² Score')
plt.grid(True)
plt.show()
```




```
# --- Plot RMSE vs Training Size ---
plt.figure(figsize=(7,5))
plt.plot(train_df['Train Size'], train_df['RMSE'], marker='o', color='red')
plt.title('RMSE vs Training Data Size')
plt.xlabel('Training Data Fraction')
plt.ylabel('RMSE')
plt.grid(True)
plt.show()
```




```
# --- Plot MAE vs Training Size ---
plt.figure(figsize=(7,5))
plt.plot(train_df['Train Size'], train_df['MAE'], marker='o', color='orange')
plt.title('MAE vs Training Data Size')
plt.xlabel('Training Data Fraction')
plt.ylabel('MAE')
plt.grid(True)
plt.show()
```




```
# ✅ Step 10B: Performance Comparison Visualization (RMSE & R² vs Training Size)

import matplotlib.pyplot as plt
import numpy as np

# Ensure results_lr, results_dt, and results_nn exist from Step 9

# Convert results into lists
train_sizes = [r[0]*100 for r in results_lr]  # Convert 0.1 → 10%

rmse_lr = [r[1] for r in results_lr]
rmse_dt = [r[1] for r in results_dt]
rmse_nn = [r[1] for r in results_nn]

r2_lr = [r[3] if not np.isnan(r[3]) else 0 for r in results_lr]
r2_dt = [r[3] if not np.isnan(r[3]) else 0 for r in results_dt]
r2_nn = [r[3] if not np.isnan(r[3]) else 0 for r in results_nn]

# --- Plot RMSE comparison ---
plt.figure(figsize=(7,5))
plt.plot(train_sizes, rmse_lr, marker='o', label='Linear Regression', color='blue')
plt.plot(train_sizes, rmse_dt, marker='s', label='Decision Tree', color='green')
plt.plot(train_sizes, rmse_nn, marker='^', label='Neural Network', color='red')
plt.title("Model Performance (RMSE) vs Training Data Size", fontsize=13)
plt.xlabel("Training Size (%)")
plt.ylabel("Root Mean Squared Error (RMSE)")
plt.legend()
plt.grid(True)
plt.show()

# --- Plot R² comparison ---
plt.figure(figsize=(7,5))
plt.plot(train_sizes, r2_lr, marker='o', label='Linear Regression', color='blue')
plt.plot(train_sizes, r2_dt, marker='s', label='Decision Tree', color='green')
plt.plot(train_sizes, r2_nn, marker='^', label='Neural Network', color='red')
plt.title("Model Performance (R²) vs Training Data Size", fontsize=13)
plt.xlabel("Training Size (%)")
plt.ylabel("R² Score")
plt.legend()
plt.grid(True)
plt.show()

print("✅ Visualization completed — Models compared successfully!")
```


Model Performance (RMSE) vs Training Data Size