Imports and Dataset Loading

ML Regressions and Their Comparison

In [60]:
```
# === Imports ===
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.datasets import fetch_california_housing
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.preprocessing import StandardScaler, PolynomialFeatures
from sklearn.metrics import mean_squared_error, r2_score

# === Load California Housing dataset ===
data = fetch_california_housing(as_frame=True)
X = data.data   # Features
y = data.target  # Target variable (median house value in $100,000s)
```

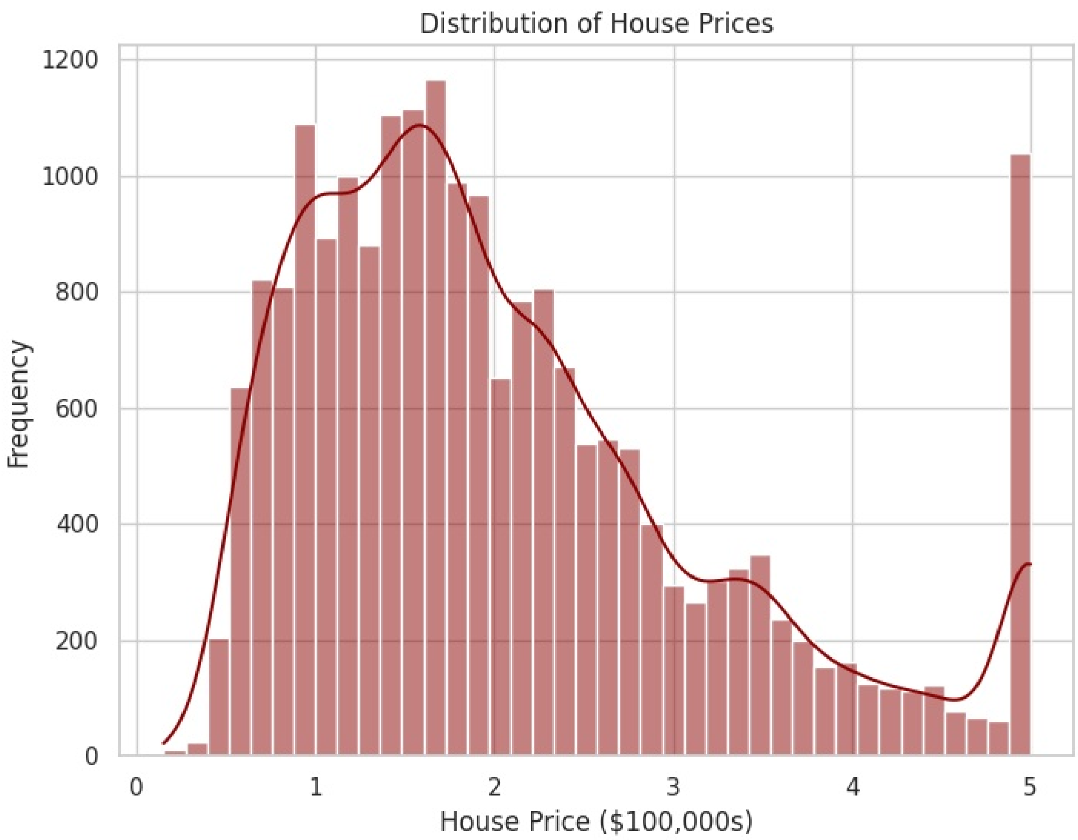
Exploratory Data Analysis (EDA)

In [61]:
```
# === Show first few rows ===
print("First 5 rows of the dataset:")
display(X.head())

# === Dataset description ===
print("\nDataset description:")
print(data.DESCR)

# === Plot house price distribution ===
plt.figure(figsize=(8, 6))
sns.histplot(y, kde=True, bins=40, color='darkred')
plt.title('Distribution of House Prices')
plt.xlabel('House Price ($100,000s)')
plt.ylabel('Frequency')
plt.grid(True)
plt.show()
```

```
First 5 rows of the dataset:
```

	MedInc	HouseAge	AveRooms	AveBedrms	Population	AveOccup	Latitude	Longitude
0	8.3252	41.0	6.984127	1.023810	322.0	2.555556	37.88	-122.23
1	8.3014	21.0	6.238137	0.971880	2401.0	2.109842	37.86	-122.22
2	7.2574	52.0	8.288136	1.073446	496.0	2.802260	37.85	-122.24
3	5.6431	52.0	5.817352	1.073059	558.0	2.547945	37.85	-122.25
4	3.8462	52.0	6.281853	1.081081	565.0	2.181467	37.85	-122.25

```
Dataset description:
.. _california_housing_dataset:

California Housing dataset
--------------------------

**Data Set Characteristics:**

:Number of Instances: 20640

:Number of Attributes: 8 numeric, predictive attributes and the target

:Attribute Information:
    - MedInc        median income in block group
    - HouseAge      median house age in block group
    - AveRooms      average number of rooms per household
    - AveBedrms     average number of bedrooms per household
    - Population    block group population
    - AveOccup      average number of household members
    - Latitude      block group latitude
    - Longitude     block group longitude

:Missing Attribute Values: None

This dataset was obtained from the StatLib repository.
https://www.dcc.fc.up.pt/~ltorgo/Regression/cal_housing.html

The target variable is the median house value for California districts,
expressed in hundreds of thousands of dollars ($100,000).

This dataset was derived from the 1990 U.S. census, using one row per census
block group. A block group is the smallest geographical unit for which the U.S.
Census Bureau publishes sample data (a block group typically has a population
of 600 to 3,000 people).

A household is a group of people residing within a home. Since the average
number of rooms and bedrooms in this dataset are provided per household, these
columns may take surprisingly large values for block groups with few households
and many empty houses, such as vacation resorts.

It can be downloaded/loaded using the
:func:`sklearn.datasets.fetch_california_housing` function.

.. rubric:: References

- Pace, R. Kelley and Ronald Barry, Sparse Spatial Autoregressions,
  Statistics and Probability Letters, 33 (1997) 291-297
```



Data Preprocessing

In [62]:
```python
# === Feature Scaling (Standardization) ===
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# === Train/Test Split ===
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)
```

Linear Regression Model

In [63]:
```python
# === Train Linear Regression ===
lin_reg = LinearRegression()
lin_reg.fit(X_train, y_train)

# === Predict on Test Set ===
y_pred_lin = lin_reg.predict(X_test)

# === Evaluate Performance ===
mse_lin = mean_squared_error(y_test, y_pred_lin)
r2_lin = r2_score(y_test, y_pred_lin)

print(f"Linear Regression MSE: {mse_lin:.4f}")
print(f"Linear Regression R²: {r2_lin:.4f}")
```

```
Linear Regression MSE: 0.5559
Linear Regression R²: 0.5758
```

Polynomial Regression Model (Degree 2)

In [64]:
```python
# === Polynomial Features ===
poly = PolynomialFeatures(degree=2, include_bias=False)
X_poly = poly.fit_transform(X_scaled)

# === Split Polynomial Features ===
X_train_poly, X_test_poly, y_train_poly, y_test_poly = train_test_split(
    X_poly, y, test_size=0.2, random_state=42
)

# === Train Polynomial Regression ===
poly_reg = LinearRegression()
poly_reg.fit(X_train_poly, y_train_poly)

# === Predict and Evaluate ===
y_pred_poly = poly_reg.predict(X_test_poly)
mse_poly = mean_squared_error(y_test_poly, y_pred_poly)
r2_poly = r2_score(y_test_poly, y_pred_poly)

print(f"\nPolynomial Regression (degree=2) MSE: {mse_poly:.4f}")
print(f"Polynomial Regression (degree=2) R²: {r2_poly:.4f}")
```

```
Polynomial Regression (degree=2) MSE: 0.4643
Polynomial Regression (degree=2) R²: 0.6457
```

Random Forest Regressor

In [65]:
```python
from sklearn.ensemble import RandomForestRegressor

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

# === Predict and Evaluate ===
y_pred_rf = rf_reg.predict(X_test)
mse_rf = mean_squared_error(y_test, y_pred_rf)
r2_rf = r2_score(y_test, y_pred_rf)

print(f"\nRandom Forest Regressor MSE: {mse_rf:.4f}")
print(f"Random Forest Regressor R²: {r2_rf:.4f}")
```

```
Random Forest Regressor MSE: 0.2555
Random Forest Regressor R²: 0.8050
```

Gradient Boosting Regressor

In [66]:
```python
from sklearn.ensemble import GradientBoostingRegressor
```

```python
# === Train Gradient Boosting ===
gb_reg = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
gb_reg.fit(X_train, y_train)

# === Predict and Evaluate ===
y_pred_gb = gb_reg.predict(X_test)
mse_gb = mean_squared_error(y_test, y_pred_gb)
r2_gb = r2_score(y_test, y_pred_gb)

print(f"\nGradient Boosting Regressor MSE: {mse_gb:.4f}")
print(f"Gradient Boosting Regressor R²: {r2_gb:.4f}")
```

```
Gradient Boosting Regressor MSE: 0.2940
Gradient Boosting Regressor R²: 0.7756
```

XGBoost Regressor

In [67]:

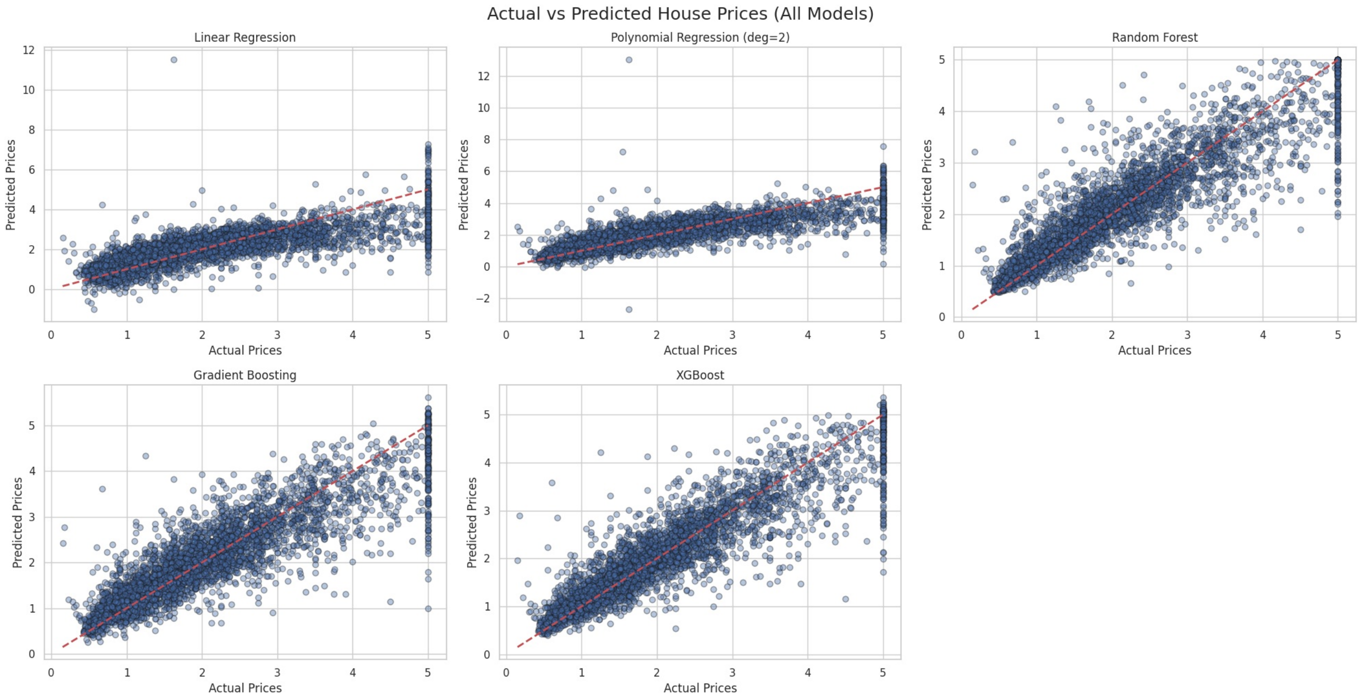
```python
# Optional: Only if xgboost is installed
from xgboost import XGBRegressor

# === Train XGBoost ===
xgb_reg = XGBRegressor(n_estimators=100, learning_rate=0.1, random_state=42)
xgb_reg.fit(X_train, y_train)

# === Predict and Evaluate ===
y_pred_xgb = xgb_reg.predict(X_test)
mse_xgb = mean_squared_error(y_test, y_pred_xgb)
r2_xgb = r2_score(y_test, y_pred_xgb)

print(f"\nXGBoost Regressor MSE: {mse_xgb:.4f}")
print(f"XGBoost Regressor R²: {r2_xgb:.4f}")
```

```
XGBoost Regressor MSE: 0.2273
XGBoost Regressor R²: 0.8266
```

Plot: Actual vs Predicted Prices for All Models

In [68]:

```python
# === Prepare Predictions ===
model_predictions = {
    'Linear Regression': y_pred_lin,
    'Polynomial Regression (deg=2)': y_pred_poly,
    'Random Forest': y_pred_rf,
    'Gradient Boosting': y_pred_gb,
    'XGBoost': y_pred_xgb
}

# === Plot Settings ===
plt.figure(figsize=(20, 10))
for idx, (model_name, y_pred) in enumerate(model_predictions.items(), start=1):
    plt.subplot(2, 3, idx)
    plt.scatter(y_test, y_pred, alpha=0.4, edgecolor='k')
    plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r--', lw=2)
    plt.title(model_name)
    plt.xlabel("Actual Prices")
    plt.ylabel("Predicted Prices")
    plt.grid(True)

plt.tight_layout()
plt.suptitle("Actual vs Predicted House Prices (All Models)", fontsize=18, y=1.02)
plt.show()
```



Deep Learning Regression Model (MLP using TensorFlow/Keras)

Import Required Libraries

In [69]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.callbacks import EarlyStopping
```

Build and Compile the Model

In [70]:
```
# === Define Neural Network ===
model = Sequential([
    Dense(128, activation='relu', input_shape=(X_train.shape[1],)),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dropout(0.2),
    Dense(1)  # Output layer for regression
])

# === Compile Model ===
model.compile(optimizer='adam', loss='mse', metrics=['mae'])
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/layers/core/dense.py:93: UserWarning:

Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Inp
ut(shape)` object as the first layer in the model instead.
```

Train the Model

In [71]:
```
# === Add Early Stopping ===
early_stop = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

# === Train ===
history = model.fit(
    X_train, y_train,
    validation_split=0.2,
    epochs=100,
    batch_size=32,
    callbacks=[early_stop],
    verbose=1
)
```

```
Epoch 1/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 5s 8ms/step - loss: 2.0275 - mae: 0.9229 - val_loss: 0.4727 - val_mae: 0.4954
Epoch 2/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5450 - mae: 0.5345 - val_loss: 0.4316 - val_mae: 0.4707
Epoch 3/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.4947 - mae: 0.5020 - val_loss: 0.4077 - val_mae: 0.4479
```

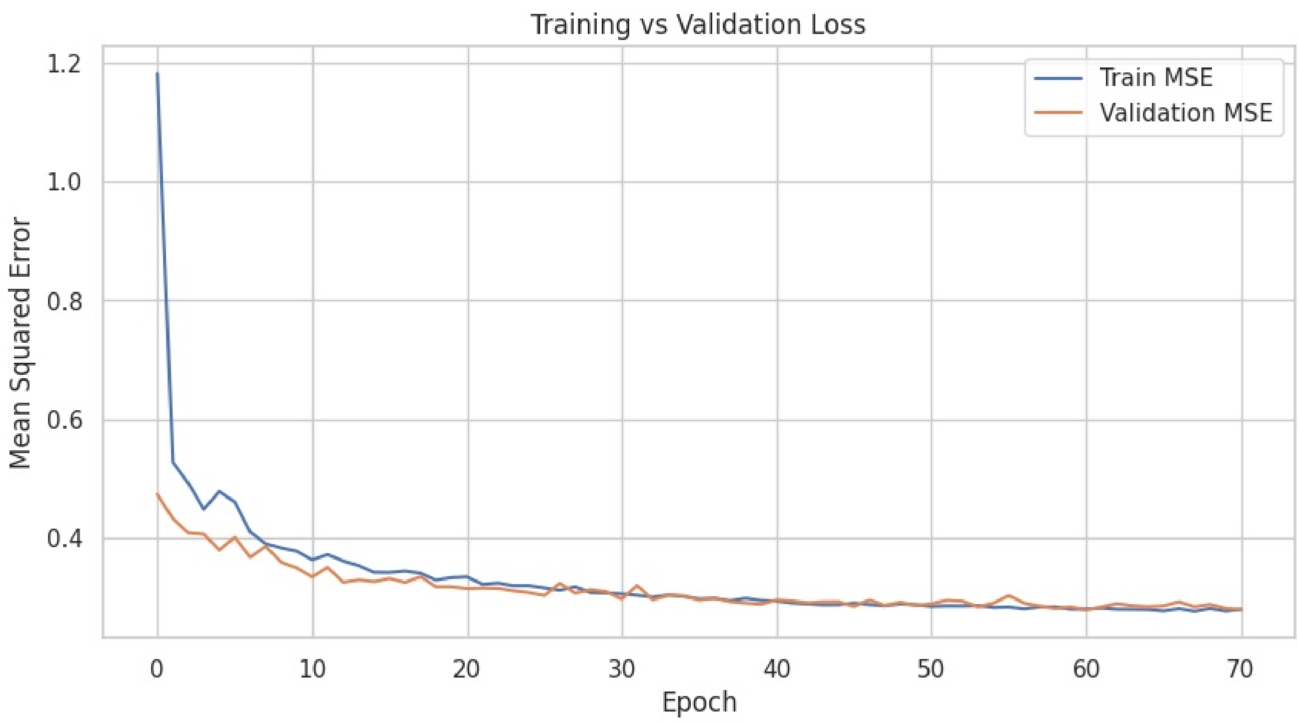
```
Epoch 4/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.4656 - mae: 0.4869 - val_loss: 0.4055 - val_mae: 0.4455
Epoch 5/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.5533 - mae: 0.4820 - val_loss: 0.3785 - val_mae: 0.4267
Epoch 6/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.4750 - mae: 0.4685 - val_loss: 0.4001 - val_mae: 0.4241
Epoch 7/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.4061 - mae: 0.4499 - val_loss: 0.3667 - val_mae: 0.4219
Epoch 8/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.4067 - mae: 0.4504 - val_loss: 0.3848 - val_mae: 0.4114
Epoch 9/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.3993 - mae: 0.4416 - val_loss: 0.3581 - val_mae: 0.4041
Epoch 10/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3692 - mae: 0.4232 - val_loss: 0.3484 - val_mae: 0.4045
Epoch 11/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3607 - mae: 0.4245 - val_loss: 0.3334 - val_mae: 0.3995
Epoch 12/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.3597 - mae: 0.4189 - val_loss: 0.3493 - val_mae: 0.3943
Epoch 13/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3504 - mae: 0.4125 - val_loss: 0.3242 - val_mae: 0.3931
Epoch 14/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.3507 - mae: 0.4171 - val_loss: 0.3283 - val_mae: 0.3976
Epoch 15/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3482 - mae: 0.4136 - val_loss: 0.3254 - val_mae: 0.3905
Epoch 16/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.3491 - mae: 0.4132 - val_loss: 0.3305 - val_mae: 0.3992
Epoch 17/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.3434 - mae: 0.4097 - val_loss: 0.3239 - val_mae: 0.3874
Epoch 18/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.3433 - mae: 0.4074 - val_loss: 0.3337 - val_mae: 0.3839
Epoch 19/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.3212 - mae: 0.3987 - val_loss: 0.3168 - val_mae: 0.3885
Epoch 20/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.3512 - mae: 0.4100 - val_loss: 0.3167 - val_mae: 0.3825
Epoch 21/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3420 - mae: 0.4020 - val_loss: 0.3135 - val_mae: 0.3820
Epoch 22/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.3287 - mae: 0.3977 - val_loss: 0.3144 - val_mae: 0.3756
Epoch 23/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3156 - mae: 0.3909 - val_loss: 0.3138 - val_mae: 0.3768
Epoch 24/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.3179 - mae: 0.3920 - val_loss: 0.3099 - val_mae: 0.3814
Epoch 25/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3120 - mae: 0.3872 - val_loss: 0.3068 - val_mae: 0.3820
Epoch 26/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.3074 - mae: 0.3851 - val_loss: 0.3019 - val_mae: 0.3731
Epoch 27/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3047 - mae: 0.3838 - val_loss: 0.3223 - val_mae: 0.3867
Epoch 28/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.3153 - mae: 0.3898 - val_loss: 0.3060 - val_mae: 0.3786
Epoch 29/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.3192 - mae: 0.3917 - val_loss: 0.3116 - val_mae: 0.3779
Epoch 30/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.2947 - mae: 0.3779 - val_loss: 0.3083 - val_mae: 0.3858
Epoch 31/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.3117 - mae: 0.3860 - val_loss: 0.2961 - val_mae: 0.3714
Epoch 32/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.2951 - mae: 0.3792 - val_loss: 0.3187 - val_mae: 0.3769
Epoch 33/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2931 - mae: 0.3744 - val_loss: 0.2945 - val_mae: 0.3681
Epoch 34/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2947 - mae: 0.3787 - val_loss: 0.3025 - val_mae: 0.3887
Epoch 35/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3033 - mae: 0.3846 - val_loss: 0.3013 - val_mae: 0.3794
Epoch 36/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2985 - mae: 0.3788 - val_loss: 0.2942 - val_mae: 0.3711
Epoch 37/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.3004 - mae: 0.3774 - val_loss: 0.2974 - val_mae: 0.3727
Epoch 38/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step - loss: 0.2871 - mae: 0.3724 - val_loss: 0.2915 - val_mae: 0.3680
Epoch 39/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.3008 - mae: 0.3809 - val_loss: 0.2891 - val_mae: 0.3635
Epoch 40/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2856 - mae: 0.3755 - val_loss: 0.2872 - val_mae: 0.3682
Epoch 41/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2898 - mae: 0.3747 - val_loss: 0.2945 - val_mae: 0.3631
Epoch 42/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2875 - mae: 0.3691 - val_loss: 0.2928 - val_mae: 0.3699
Epoch 43/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2838 - mae: 0.3701 - val_loss: 0.2893 - val_mae: 0.3615
Epoch 44/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.2990 - mae: 0.3781 - val_loss: 0.2911 - val_mae: 0.3656
Epoch 45/100
```

```
413/413 ━━━━━━━━━━━━━━━━━━━━ 4s 6ms/step - loss: 0.2778 - mae: 0.3666 - val_loss: 0.2913 - val_mae: 0.3674
Epoch 46/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2793 - mae: 0.3689 - val_loss: 0.2835 - val_mae: 0.3629
Epoch 47/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2857 - mae: 0.3746 - val_loss: 0.2943 - val_mae: 0.3637
Epoch 48/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.2727 - mae: 0.3627 - val_loss: 0.2852 - val_mae: 0.3626
Epoch 49/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.2777 - mae: 0.3667 - val_loss: 0.2904 - val_mae: 0.3658
Epoch 50/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2971 - mae: 0.3750 - val_loss: 0.2850 - val_mae: 0.3587
Epoch 51/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2857 - mae: 0.3713 - val_loss: 0.2879 - val_mae: 0.3644
Epoch 52/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2806 - mae: 0.3692 - val_loss: 0.2938 - val_mae: 0.3772
Epoch 53/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.2908 - mae: 0.3745 - val_loss: 0.2924 - val_mae: 0.3734
Epoch 54/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2821 - mae: 0.3696 - val_loss: 0.2829 - val_mae: 0.3607
Epoch 55/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2756 - mae: 0.3640 - val_loss: 0.2889 - val_mae: 0.3664
Epoch 56/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - loss: 0.2824 - mae: 0.3693 - val_loss: 0.3017 - val_mae: 0.3628
Epoch 57/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 4ms/step - loss: 0.2718 - mae: 0.3635 - val_loss: 0.2890 - val_mae: 0.3723
Epoch 58/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2918 - mae: 0.3767 - val_loss: 0.2839 - val_mae: 0.3609
Epoch 59/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - loss: 0.2854 - mae: 0.3728 - val_loss: 0.2801 - val_mae: 0.3632
Epoch 60/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.2736 - mae: 0.3624 - val_loss: 0.2826 - val_mae: 0.3567
Epoch 61/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.2766 - mae: 0.3653 - val_loss: 0.2778 - val_mae: 0.3554
Epoch 62/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 2s 6ms/step - loss: 0.2869 - mae: 0.3710 - val_loss: 0.2827 - val_mae: 0.3595
Epoch 63/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2852 - mae: 0.3723 - val_loss: 0.2881 - val_mae: 0.3692
Epoch 64/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2781 - mae: 0.3676 - val_loss: 0.2846 - val_mae: 0.3604
Epoch 65/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.2724 - mae: 0.3612 - val_loss: 0.2831 - val_mae: 0.3660
Epoch 66/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.2763 - mae: 0.3657 - val_loss: 0.2844 - val_mae: 0.3585
Epoch 67/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.2791 - mae: 0.3680 - val_loss: 0.2911 - val_mae: 0.3759
Epoch 68/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 6ms/step - loss: 0.2674 - mae: 0.3610 - val_loss: 0.2830 - val_mae: 0.3607
Epoch 69/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - loss: 0.2777 - mae: 0.3652 - val_loss: 0.2868 - val_mae: 0.3655
Epoch 70/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.2748 - mae: 0.3631 - val_loss: 0.2803 - val_mae: 0.3547
Epoch 71/100
413/413 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - loss: 0.2740 - mae: 0.3643 - val_loss: 0.2783 - val_mae: 0.3607
```

Visualize Training Progress

In [72]:

```
# === Plot Loss Curves ===
plt.figure(figsize=(10, 5))
plt.plot(history.history['loss'], label='Train MSE')
plt.plot(history.history['val_loss'], label='Validation MSE')
plt.title('Training vs Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Mean Squared Error')
plt.legend()
plt.grid(True)
plt.show()
```



Evaluate the Model

In [73]:
```
# === Predict on Test Data ===
y_pred_nn = model.predict(X_test).flatten()

# === Evaluate ===
mse_nn = mean_squared_error(y_test, y_pred_nn)
r2_nn = r2_score(y_test, y_pred_nn)

print(f"\nNeural Network MSE: {mse_nn:.4f}")
print(f"Neural Network R²: {r2_nn:.4f}")
```

```
129/129 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step

Neural Network MSE: 0.2701
Neural Network R²: 0.7939
```

Add NN Results to Comparison Table

In [74]:
```
# === Add Neural Network Results (Pandas 2.0+ Compatible) ===
nn_result = pd.DataFrame([{
    'Model': 'Neural Network (Deep Learning)',
    'MSE': mse_nn,
    'R² Score': r2_nn
}])

results = pd.concat([results, nn_result], ignore_index=True)

# Display updated results
display(results.sort_values(by='R² Score', ascending=False))
```

	Model	MSE	R^2 Score
4	XGBoost	0.227262	0.826571
2	Random Forest	0.255498	0.805024
5	Neural Network (Deep Learning)	0.263983	0.798549
6	Neural Network (Deep Learning)	0.270098	0.793882
3	Gradient Boosting	0.293999	0.775643
1	Polynomial Regression (deg=2)	0.464302	0.645682
0	Linear Regression	0.555892	0.575788

Plot: R^2 Score Comparison (ML vs Neural Network)

In [75]:
```
# === Bar Plot of R² Scores for All Models ===
plt.figure(figsize=(12, 6))
```

```
sns.barplot(data=results.sort_values(by='R² Score', ascending=False),
            x='Model', y='R² Score', palette='viridis')

plt.title('Model Comparison: R² Scores (Higher is Better)', fontsize=16)
plt.xlabel('Regression Model')
plt.ylabel('R² Score')
plt.xticks(rotation=30, ha='right')
plt.ylim(0, 1)  # R² ranges from 0 to 1
plt.grid(True, linestyle='--', alpha=0.6)
plt.tight_layout()
plt.show()
```

```
/tmp/ipython-input-1353875515.py:3: FutureWarning:



Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable
to `hue` and set `legend=False` for the same effect.
```