In [4]:
```
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, cross_val_score, KFold
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor, E
from sklearn.linear_model import LinearRegression, Ridge
from sklearn.svm import SVR
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import warnings
warnings.filterwarnings('ignore')

np.random.seed(42)

# Generate realistic phone data
def generate_phone_data(n_samples=1000):
    brands = ["Apple", "Samsung", "OnePlus", "Xiaomi", "Google"]
    conditions = ["Excellent", "Good", "Fair", "Poor"]
    phone_models = {
        "Apple": ["iPhone 15", "iPhone 14", "iPhone 13", "iPhone 12", "iPhone 11
        "Samsung": ["Galaxy S23", "Galaxy S22", "Galaxy S21", "Galaxy A54", "Gal
        "OnePlus": ["OnePlus 11", "OnePlus 10", "OnePlus 9", "OnePlus Nord"],
        "Xiaomi": ["Redmi Note 13", "Redmi Note 12", "Poco F5", "Mi 13"],
        "Google": ["Pixel 8", "Pixel 7", "Pixel 6"]
    }

    data = []
    for i in range(n_samples):
        brand = np.random.choice(brands)
        model = np.random.choice(phone_models[brand])

        if brand == "Apple":
            ram = np.random.choice([4, 6, 8])
            storage = np.random.choice([64, 128, 256, 512])
            battery = np.random.randint(3000, 4500)
            screen_size = round(np.random.uniform(5.4, 6.7), 1)
        elif brand in ["Samsung", "OnePlus"]:
            ram = np.random.choice([6, 8, 12])
            storage = np.random.choice([128, 256, 512])
            battery = np.random.randint(4000, 6000)
            screen_size = round(np.random.uniform(6.1, 6.8), 1)
        else:
            ram = np.random.choice([4, 6, 8])
            storage = np.random.choice([64, 128, 256])
            battery = np.random.randint(4500, 7000)
            screen_size = round(np.random.uniform(6.0, 6.9), 1)

        condition = np.random.choice(conditions, p=[0.2, 0.4, 0.3, 0.1])
        age_months = np.random.randint(0, 36)

        base_price = {
            "Apple": 90000, "Samsung": 70000, "OnePlus": 50000,
            "Xiaomi": 30000, "Google": 60000
        }[brand]

        condition_factor = {"Excellent": 0.75, "Good": 0.60, "Fair": 0.45, "Poor
        ram_factor = 0.8 + (ram / 12) * 0.4
```

```
        storage_factor = 0.7 + (storage / 512) * 0.6
        age_factor = max(0.3, 1 - (age_months / 48))
        battery_factor = min(1.0, 0.7 + (battery / 6000) * 0.3)

        if "Pro" in model or "Ultra" in model:
            model_factor = 1.2
        elif "Lite" in model or "SE" in model:
            model_factor = 0.8
        else:
            model_factor = 1.0

        resale_value = (base_price * condition_factor * ram_factor *
                        storage_factor * age_factor * battery_factor * model_fact

        resale_value *= np.random.uniform(0.9, 1.1)
        resale_value = max(2000, round(resale_value, -2))

        data.append([
            brand, model, ram, storage, battery, screen_size,
            condition, age_months, round(resale_value, 2)
        ])

    return pd.DataFrame(data, columns=[
        "brand", "model", "ram", "storage", "battery", "screen_size",
        "condition", "age_months", "resale_value"
    ])

# Generate realistic laptop data
def generate_laptop_data(n_samples=1000):
    brands = ["Apple", "HP", "Dell", "Lenovo", "Asus"]
    processors = ["i3", "i5", "i7", "i9", "Ryzen 5", "Ryzen 7", "M1", "M2"]
    graphics = ["Integrated", "NVIDIA GTX 1650", "NVIDIA RTX 3050", "NVIDIA RTX
    storage_types = ["HDD", "SSD", "NVMe SSD"]
    conditions = ["Excellent", "Good", "Fair", "Poor"]
    laptop_models = {
        "Apple": ["MacBook Air", "MacBook Pro 13", "MacBook Pro 14"],
        "Dell": ["XPS 13", "XPS 15", "Inspiron 15", "Latitude 14"],
        "HP": ["Spectre x360", "Envy 13", "Pavilion 15", "Omen 16"],
        "Lenovo": ["ThinkPad X1", "ThinkPad T14", "Yoga 9i", "Legion 5"],
        "Asus": ["ZenBook 14", "ROG Zephyrus", "TUF Gaming", "VivoBook"]
    }

    data = []
    for i in range(n_samples):
        brand = np.random.choice(brands)
        model = np.random.choice(laptop_models[brand])
        processor = np.random.choice(processors)
        graphics_card = np.random.choice(graphics)
        storage_type = np.random.choice(storage_types, p=[0.2, 0.5, 0.3])

        if brand == "Apple":
            ram = np.random.choice([8, 16, 32])
            storage = np.random.choice([256, 512, 1024])
            screen_size = np.random.choice([13.3, 14.2, 16.2])
        elif "Gaming" in model or "ROG" in model:
            ram = np.random.choice([16, 32])
            storage = np.random.choice([512, 1024])
            screen_size = round(np.random.uniform(15.6, 17.3), 1)
        else:
            ram = np.random.choice([8, 16])
```

```
            storage = np.random.choice([256, 512])
            screen_size = round(np.random.uniform(13.3, 16.0), 1)

        condition = np.random.choice(conditions, p=[0.15, 0.45, 0.3, 0.1])
        age_months = np.random.randint(0, 48)

        base_price = {
            "Apple": 120000, "HP": 80000, "Dell": 85000,
            "Lenovo": 75000, "Asus": 70000
        }[brand]

        condition_factor = {"Excellent": 0.70, "Good": 0.55, "Fair": 0.40, "Poor

        if processor in ["i9", "Ryzen 9", "M2"]:
            cpu_factor = 1.3
        elif processor in ["i7", "Ryzen 7", "M1"]:
            cpu_factor = 1.15
        elif processor in ["i5", "Ryzen 5"]:
            cpu_factor = 1.0
        else:
            cpu_factor = 0.8

        if "RTX 30" in graphics_card:
            gpu_factor = 1.3
        elif "GTX" in graphics_card:
            gpu_factor = 1.2
        else:
            gpu_factor = 1.0

        storage_type_factor = {"HDD": 0.7, "SSD": 1.0, "NVMe SSD": 1.2}[storage_
        ram_factor = 0.8 + (ram / 32) * 0.4
        storage_factor = 0.7 + (storage / 1024) * 0.5
        age_factor = max(0.25, 1 - (age_months / 60))
        screen_factor = 0.9 + (screen_size / 17.3) * 0.2

        if "Pro" in model or "XPS" in model or "Spectre" in model:
            model_factor = 1.25
        elif "Air" in model or "VivoBook" in model:
            model_factor = 0.9
        else:
            model_factor = 1.0

        resale_value = (base_price * condition_factor * cpu_factor * gpu_factor
                        storage_type_factor * ram_factor * storage_factor *
                        age_factor * screen_factor * model_factor)

        resale_value *= np.random.uniform(0.85, 1.15)
        resale_value = max(5000, round(resale_value, -2))

        data.append([
            brand, model, processor, graphics_card, storage, storage_type,
            ram, screen_size, condition, age_months, round(resale_value, 2)
        ])

    return pd.DataFrame(data, columns=[
        "brand", "model", "processor", "graphics", "storage", "storage_type",
        "ram", "screen_size", "condition", "age_months", "resale_value"
    ])

print("Generating datasets...")
```

```
phones_df = generate_phone_data(1000)
laptops_df = generate_laptop_data(1000)

print(f"Phone dataset: {phones_df.shape}")
print(f"Laptop dataset: {laptops_df.shape}")
```

```
Generating datasets...
Phone dataset: (1000, 9)
Laptop dataset: (1000, 11)
```

In [5]:
```
# Data preprocessing and feature engineering
def preprocess_and_engineer(phones_df, laptops_df):
    phones_processed = phones_df.copy()
    laptops_processed = laptops_df.copy()

    # Feature engineering for phones
    phones_processed['storage_per_ram'] = phones_processed['storage'] / phones_p
    phones_processed['battery_to_screen_ratio'] = phones_processed['battery'] /
    phones_processed['is_premium'] = phones_processed['brand'].isin(['Apple', 'S

    # Feature engineering for laptops
    laptops_processed['storage_per_ram'] = laptops_processed['storage'] / laptop
    laptops_processed['is_gaming_gpu'] = laptops_processed['graphics'].str.conta
    laptops_processed['is_premium_brand'] = laptops_processed['brand'].isin(['Ap
    laptops_processed['has_ssd'] = (laptops_processed['storage_type'] != 'HDD').
    laptops_processed['processor_tier'] = laptops_processed['processor'].map({
        'i3': 1, 'Ryzen 5': 2, 'i5': 2, 'i7': 3, 'Ryzen 7': 3, 'i9': 4, 'M1': 3,
    })

    return phones_processed, laptops_processed

phones_enhanced, laptops_enhanced = preprocess_and_engineer(phones_df, laptops_d

# Encoding categorical variables
def encode_features(phones_df, laptops_df):
    phone_categorical = ["brand", "model", "condition"]
    laptop_categorical = ["brand", "model", "processor", "graphics", "storage_ty

    phones_encoded = phones_df.copy()
    laptops_encoded = laptops_df.copy()

    phone_encoders, laptop_encoders = {}, {}

    for col in phone_categorical:
        le = LabelEncoder()
        phones_encoded[col] = le.fit_transform(phones_encoded[col].astype(str))
        phone_encoders[col] = le

    for col in laptop_categorical:
        le = LabelEncoder()
        laptops_encoded[col] = le.fit_transform(laptops_encoded[col].astype(str)
        laptop_encoders[col] = le

    return phones_encoded, laptops_encoded, phone_encoders, laptop_encoders

phones_encoded, laptops_encoded, phone_encoders, laptop_encoders = encode_featur

# Define feature sets
phone_features = ["brand", "model", "ram", "storage", "battery", "screen_size",
                  "condition", "age_months", "storage_per_ram", "battery_to_scree
```

```
laptop_features = ["brand", "model", "processor", "graphics", "storage", "storag
                   "ram", "screen_size", "condition", "age_months", "storage_per_
                   "is_gaming_gpu", "is_premium_brand", "has_ssd", "processor_tie

print("Preprocessing completed")
print(f"Phone features: {len(phone_features)}")
print(f"Laptop features: {len(laptop_features)}")
```

```
Preprocessing completed
Phone features: 11
Laptop features: 15
```

In [10]:

```
# Prepare data for training
X_phone = phones_encoded[phone_features]
y_phone = phones_encoded["resale_value"]
X_laptop = laptops_encoded[laptop_features]
y_laptop = laptops_encoded["resale_value"]

# Train-test split
X_phone_train, X_phone_test, y_phone_train, y_phone_test = train_test_split(
    X_phone, y_phone, test_size=0.2, random_state=42
)
X_laptop_train, X_laptop_test, y_laptop_train, y_laptop_test = train_test_split(
    X_laptop, y_laptop, test_size=0.2, random_state=42
)

print(f"Phone training set: {X_phone_train.shape}")
print(f"Phone testing set: {X_phone_test.shape}")
print(f"Laptop training set: {X_laptop_train.shape}")
print(f"Laptop testing set: {X_laptop_test.shape}")

# Define models - create fresh instances for each device type
def get_models():
    return {
        'Random Forest': RandomForestRegressor(n_estimators=200, random_state=42
        'Gradient Boosting': GradientBoostingRegressor(n_estimators=200, random_
        'Extra Trees': ExtraTreesRegressor(n_estimators=200, random_state=42, n_
        'Linear Regression': LinearRegression(n_jobs=-1),
        'Ridge Regression': Ridge(alpha=1.0, random_state=42),
        'SVR': SVR(kernel='rbf')
    }

# Train and evaluate models
def train_and_evaluate(X_train, X_test, y_train, y_test, device_name):
    results = {}
    trained_models = {}

    kf = KFold(n_splits=5, shuffle=True, random_state=42)
    models = get_models()  # Get fresh model instances

    for name, model in models.items():
        try:
            model.fit(X_train, y_train)
            y_pred = model.predict(X_test)

            mae = mean_absolute_error(y_test, y_pred)
            rmse = np.sqrt(mean_squared_error(y_test, y_pred))
            r2 = r2_score(y_test, y_pred)
```

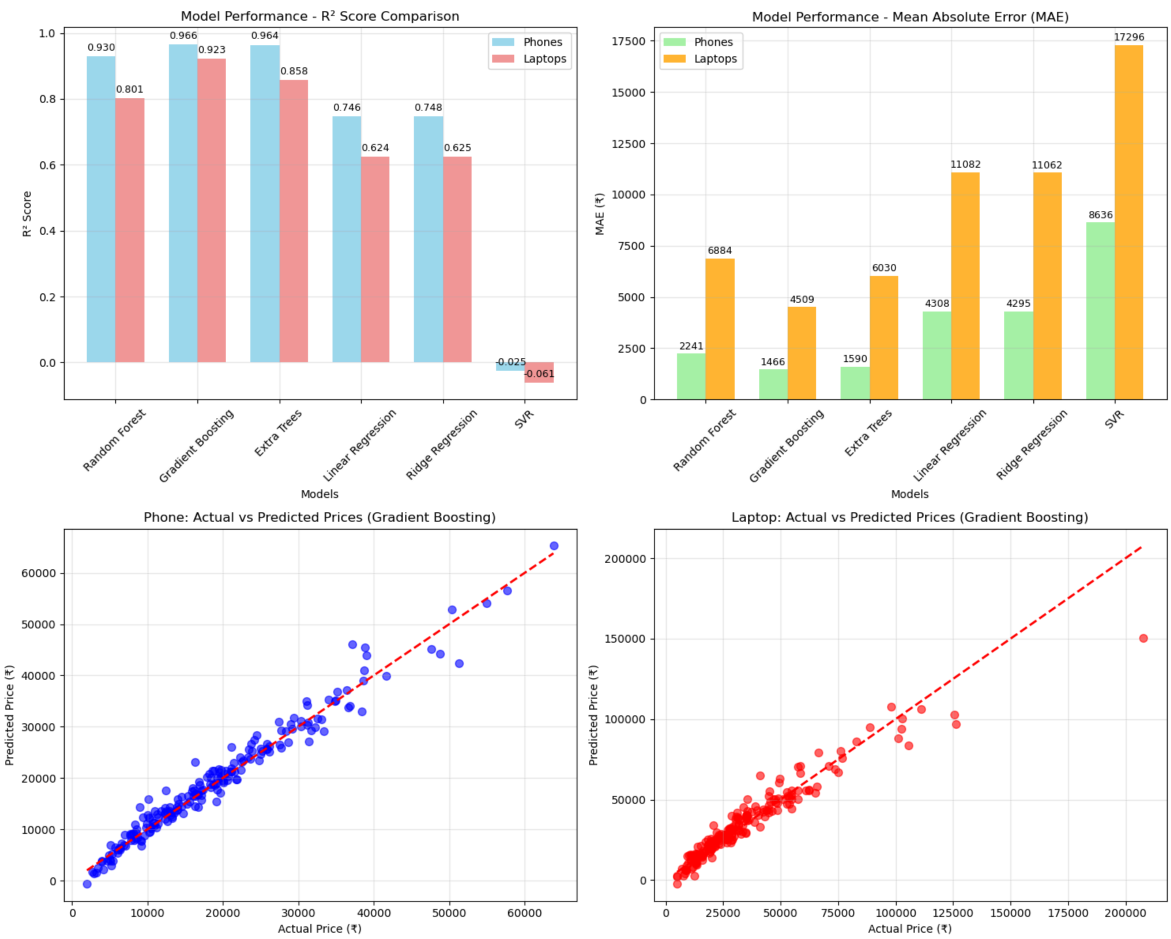
```
            cv_r2 = cross_val_score(model, X_train, y_train, cv=kf, scoring='r2'

            results[name] = {
                'MAE': mae,
                'RMSE': rmse,
                'R2': r2,
                'CV_R2_mean': cv_r2.mean(),
                'CV_R2_std': cv_r2.std()
            }
            trained_models[name] = model

            print(f"{device_name} - {name}: R2={r2:.4f}, MAE={mae:.0f}, CV_R2={c

        except Exception as e:
            print(f"{device_name} - {name}: Error - {str(e)}")
            continue

    return results, trained_models

print("Training phone models...")
phone_results, phone_models = train_and_evaluate(X_phone_train, X_phone_test, y_

print("\nTraining laptop models...")
laptop_results, laptop_models = train_and_evaluate(X_laptop_train, X_laptop_test

# Select best models
if phone_results:
    best_phone_model_name = max(phone_results.items(), key=lambda x: x[1]['R2'])
    best_phone_model = phone_models[best_phone_model_name]
    print(f"\nBest Phone Model: {best_phone_model_name}")
else:
    best_phone_model_name = None
    best_phone_model = None
    print("\nNo phone models were successfully trained")

if laptop_results:
    best_laptop_model_name = max(laptop_results.items(), key=lambda x: x[1]['R2'
    best_laptop_model = laptop_models[best_laptop_model_name]
    print(f"Best Laptop Model: {best_laptop_model_name}")
else:
    best_laptop_model_name = None
    best_laptop_model = None
    print("No laptop models were successfully trained")

# Generate predictions for visualization
if best_phone_model is not None:
    y_phone_pred = best_phone_model.predict(X_phone_test)
    phone_residuals = y_phone_test - y_phone_pred
else:
    y_phone_pred = None
    phone_residuals = None

if best_laptop_model is not None:
    y_laptop_pred = best_laptop_model.predict(X_laptop_test)
    laptop_residuals = y_laptop_test - y_laptop_pred
else:
    y_laptop_pred = None
    laptop_residuals = None

print("\nModel training completed successfully!")
```

```
Phone training set: (800, 11)
Phone testing set: (200, 11)
Laptop training set: (800, 15)
Laptop testing set: (200, 15)
Training phone models...
Phone - Random Forest: R2=0.9300, MAE=2241, CV_R2=0.9036
Phone - Gradient Boosting: R2=0.9659, MAE=1466, CV_R2=0.9599
Phone - Extra Trees: R2=0.9640, MAE=1590, CV_R2=0.9341
Phone - Linear Regression: R2=0.7462, MAE=4308, CV_R2=0.7366
Phone - Ridge Regression: R2=0.7477, MAE=4295, CV_R2=0.7366
Phone - SVR: R2=-0.0250, MAE=8636, CV_R2=-0.0560

Training laptop models...
Laptop - Random Forest: R2=0.8013, MAE=6884, CV_R2=0.7701
Laptop - Gradient Boosting: R2=0.9228, MAE=4509, CV_R2=0.8809
Laptop - Extra Trees: R2=0.8576, MAE=6030, CV_R2=0.7966
Laptop - Linear Regression: R2=0.6242, MAE=11082, CV_R2=0.6337
Laptop - Ridge Regression: R2=0.6249, MAE=11062, CV_R2=0.6337
Laptop - SVR: R2=-0.0612, MAE=17296, CV_R2=-0.0704

Best Phone Model: Gradient Boosting
Best Laptop Model: Gradient Boosting

Model training completed successfully!
```

In [11]:

```python
# Model performance comparison graphs
fig, axes = plt.subplots(2, 2, figsize=(15, 12))

# R² Score Comparison
models_list = list(phone_results.keys())
phone_r2 = [phone_results[model]['R2'] for model in models_list]
laptop_r2 = [laptop_results[model]['R2'] for model in models_list]

x = np.arange(len(models_list))
width = 0.35

axes[0,0].bar(x - width/2, phone_r2, width, label='Phones', color='skyblue', alp
axes[0,0].bar(x + width/2, laptop_r2, width, label='Laptops', color='lightcoral'
axes[0,0].set_xlabel('Models')
axes[0,0].set_ylabel('R² Score')
axes[0,0].set_title('Model Performance - R² Score Comparison')
axes[0,0].set_xticks(x)
axes[0,0].set_xticklabels(models_list, rotation=45)
axes[0,0].legend()
axes[0,0].grid(True, alpha=0.3)

for i, v in enumerate(phone_r2):
    axes[0,0].text(i - width/2, v + 0.01, f'{v:.3f}', ha='center', va='bottom',
for i, v in enumerate(laptop_r2):
    axes[0,0].text(i + width/2, v + 0.01, f'{v:.3f}', ha='center', va='bottom',

# MAE Comparison
phone_mae = [phone_results[model]['MAE'] for model in models_list]
laptop_mae = [laptop_results[model]['MAE'] for model in models_list]

axes[0,1].bar(x - width/2, phone_mae, width, label='Phones', color='lightgreen',
axes[0,1].bar(x + width/2, laptop_mae, width, label='Laptops', color='orange', a
axes[0,1].set_xlabel('Models')
axes[0,1].set_ylabel('MAE (₹)')
axes[0,1].set_title('Model Performance - Mean Absolute Error (MAE)')
```

```
axes[0,1].set_xticks(x)
axes[0,1].set_xticklabels(models_list, rotation=45)
axes[0,1].legend()
axes[0,1].grid(True, alpha=0.3)

for i, v in enumerate(phone_mae):
    axes[0,1].text(i - width/2, v + 100, f'{v:.0f}', ha='center', va='bottom', f
for i, v in enumerate(laptop_mae):
    axes[0,1].text(i + width/2, v + 100, f'{v:.0f}', ha='center', va='bottom', f

# Actual vs Predicted for Best Phone Model
axes[1,0].scatter(y_phone_test, y_phone_pred, alpha=0.6, color='blue', s=50)
axes[1,0].plot([y_phone_test.min(), y_phone_test.max()], [y_phone_test.min(), y_
axes[1,0].set_xlabel('Actual Price (₹)')
axes[1,0].set_ylabel('Predicted Price (₹)')
axes[1,0].set_title(f'Phone: Actual vs Predicted Prices ({best_phone_model_name}
axes[1,0].grid(True, alpha=0.3)

# Actual vs Predicted for Best Laptop Model
axes[1,1].scatter(y_laptop_test, y_laptop_pred, alpha=0.6, color='red', s=50)
axes[1,1].plot([y_laptop_test.min(), y_laptop_test.max()], [y_laptop_test.min(),
axes[1,1].set_xlabel('Actual Price (₹)')
axes[1,1].set_ylabel('Predicted Price (₹)')
axes[1,1].set_title(f'Laptop: Actual vs Predicted Prices ({best_laptop_model_nam
axes[1,1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()
```

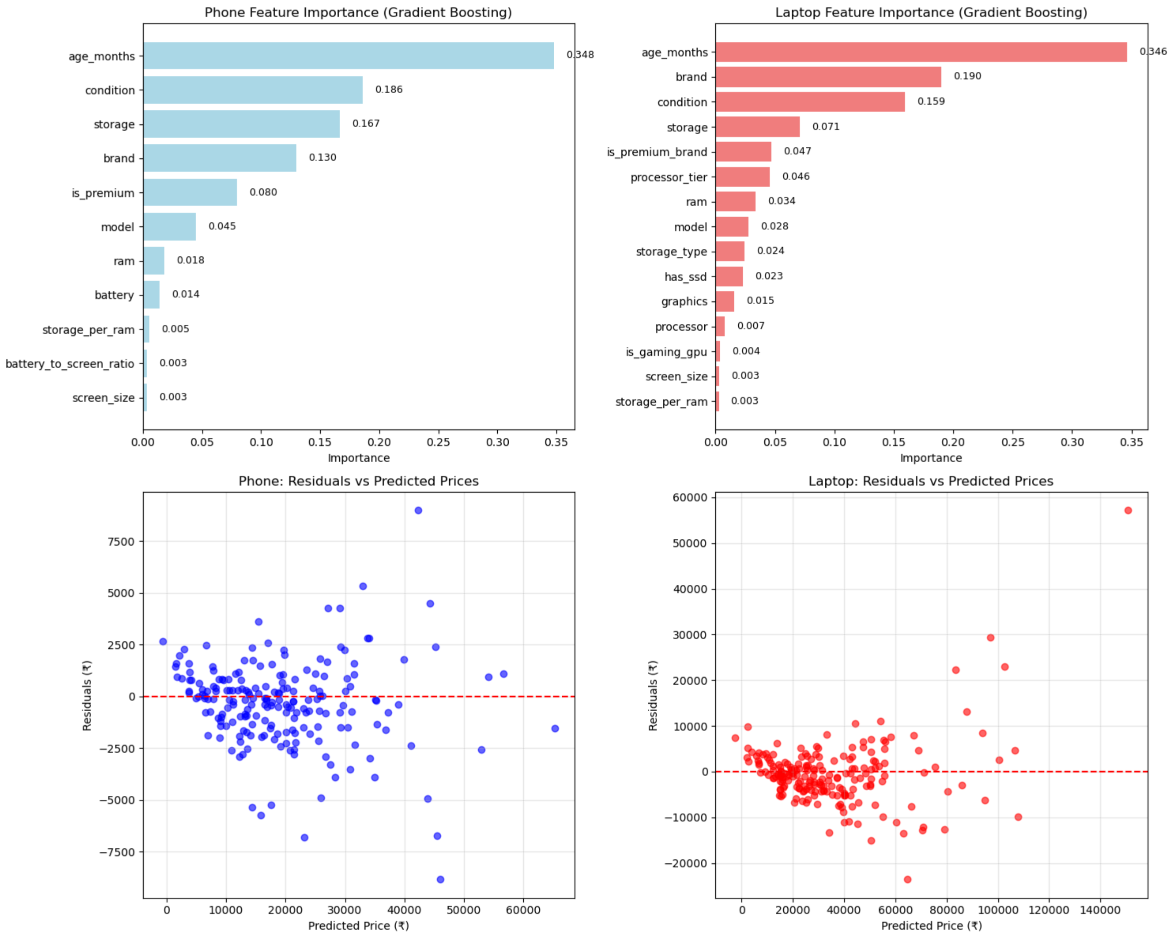


In [14]:

```
# Feature importance and residual analysis
fig, axes = plt.subplots(2, 2, figsize=(15, 12))
```

```
# Phone feature importance
if best_phone_model is not None and hasattr(best_phone_model, 'feature_importanc
    phone_importance = pd.DataFrame({
        'feature': X_phone.columns,
        'importance': best_phone_model.feature_importances_
    }).sort_values('importance', ascending=True)

    axes[0,0].barh(phone_importance['feature'], phone_importance['importance'],
    axes[0,0].set_xlabel('Importance')
    axes[0,0].set_title(f'Phone Feature Importance ({best_phone_model_name})')

    for i, v in enumerate(phone_importance['importance']):
        axes[0,0].text(v + 0.01, i, f'{v:.3f}', va='center', fontsize=9)
else:
    axes[0,0].text(0.5, 0.5, 'Feature Importance\nNot Available', ha='center', v
    axes[0,0].set_title('Phone Feature Importance (Not Available)')

# Laptop feature importance
if best_laptop_model is not None and hasattr(best_laptop_model, 'feature_importa
    laptop_importance = pd.DataFrame({
        'feature': X_laptop.columns,
        'importance': best_laptop_model.feature_importances_
    }).sort_values('importance', ascending=True)

    axes[0,1].barh(laptop_importance['feature'], laptop_importance['importance']
    axes[0,1].set_xlabel('Importance')
    axes[0,1].set_title(f'Laptop Feature Importance ({best_laptop_model_name})')

    for i, v in enumerate(laptop_importance['importance']):
        axes[0,1].text(v + 0.01, i, f'{v:.3f}', va='center', fontsize=9)
else:
    axes[0,1].text(0.5, 0.5, 'Feature Importance\nNot Available', ha='center', v
    axes[0,1].set_title('Laptop Feature Importance (Not Available)')

# Phone residuals
if y_phone_pred is not None and phone_residuals is not None:
    axes[1,0].scatter(y_phone_pred, phone_residuals, alpha=0.6, color='blue')
    axes[1,0].axhline(y=0, color='red', linestyle='--')
    axes[1,0].set_xlabel('Predicted Price (₹)')
    axes[1,0].set_ylabel('Residuals (₹)')
    axes[1,0].set_title('Phone: Residuals vs Predicted Prices')
    axes[1,0].grid(True, alpha=0.3)
else:
    axes[1,0].text(0.5, 0.5, 'No Residual Data\nAvailable', ha='center', va='cen
    axes[1,0].set_title('Phone: Residuals (No Data)')

# Laptop residuals
if y_laptop_pred is not None and laptop_residuals is not None:
    axes[1,1].scatter(y_laptop_pred, laptop_residuals, alpha=0.6, color='red')
    axes[1,1].axhline(y=0, color='red', linestyle='--')
    axes[1,1].set_xlabel('Predicted Price (₹)')
    axes[1,1].set_ylabel('Residuals (₹)')
    axes[1,1].set_title('Laptop: Residuals vs Predicted Prices')
    axes[1,1].grid(True, alpha=0.3)
else:
    axes[1,1].text(0.5, 0.5, 'No Residual Data\nAvailable', ha='center', va='cen
    axes[1,1].set_title('Laptop: Residuals (No Data)')

plt.tight_layout()
```

```
plt.show()

# Residual Analysis
print("Residual Analysis:")
if phone_residuals is not None:
    print(f"Phone Residuals - Mean: {phone_residuals.mean():.0f}, Std: {phone_re
else:
    print("Phone Residuals - No data available")

if laptop_residuals is not None:
    print(f"Laptop Residuals - Mean: {laptop_residuals.mean():.0f}, Std: {laptop
else:
    print("Laptop Residuals - No data available")
```

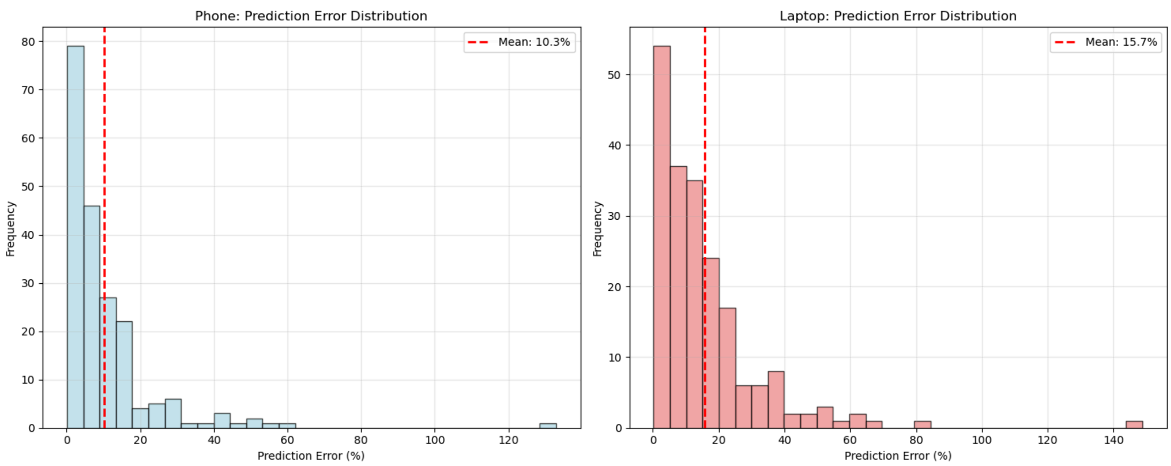



```
Residual Analysis:
Phone Residuals - Mean: -320, Std: 2076
Laptop Residuals - Mean: -417, Std: 7279
```

In [13]:

```
# Prediction error analysis
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Phone prediction error percentage
phone_error_pct = (abs(phone_residuals) / y_phone_test) * 100

# Laptop prediction error percentage
laptop_error_pct = (abs(laptop_residuals) / y_laptop_test) * 100

# Phone error distribution
axes[0].hist(phone_error_pct, bins=30, alpha=0.7, color='lightblue', edgecolor='
axes[0].axvline(x=phone_error_pct.mean(), color='red', linestyle='--', linewidth
                label=f'Mean: {phone_error_pct.mean():.1f}%')
axes[0].set_xlabel('Prediction Error (%)')
```

```python
axes[0].set_ylabel('Frequency')
axes[0].set_title('Phone: Prediction Error Distribution')
axes[0].legend()
axes[0].grid(True, alpha=0.3)

# Laptop error distribution
axes[1].hist(laptop_error_pct, bins=30, alpha=0.7, color='lightcoral', edgecolor
axes[1].axvline(x=laptop_error_pct.mean(), color='red', linestyle='--', linewidt
                label=f'Mean: {laptop_error_pct.mean():.1f}%')
axes[1].set_xlabel('Prediction Error (%)')
axes[1].set_ylabel('Frequency')
axes[1].set_title('Laptop: Prediction Error Distribution')
axes[1].legend()
axes[1].grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Final summary
print("FINAL MODEL PERFORMANCE SUMMARY")
print("="*50)
print(f"Best Phone Model: {best_phone_model_name}")
print(f"  R² Score: {phone_results[best_phone_model_name]['R2']:.4f}")
print(f"  MAE: {phone_results[best_phone_model_name]['MAE']:.0f}")
print(f"  RMSE: {phone_results[best_phone_model_name]['RMSE']:.0f}")
print(f"  CV R²: {phone_results[best_phone_model_name]['CV_R2_mean']:.4f}")

print(f"\nBest Laptop Model: {best_laptop_model_name}")
print(f"  R² Score: {laptop_results[best_laptop_model_name]['R2']:.4f}")
print(f"  MAE: {laptop_results[best_laptop_model_name]['MAE']:.0f}")
print(f"  RMSE: {laptop_results[best_laptop_model_name]['RMSE']:.0f}")
print(f"  CV R²: {laptop_results[best_laptop_model_name]['CV_R2_mean']:.4f}")

print(f"\nAverage Prediction Error:")
print(f"  Phones: {phone_error_pct.mean():.1f}%")
print(f"  Laptops: {laptop_error_pct.mean():.1f}%")

print(f"\nDataset Statistics:")
print(f"  Phone records: {len(phones_df)}")
print(f"  Laptop records: {len(laptops_df)}")
print(f"  Total records: {len(phones_df) + len(laptops_df)}")
```

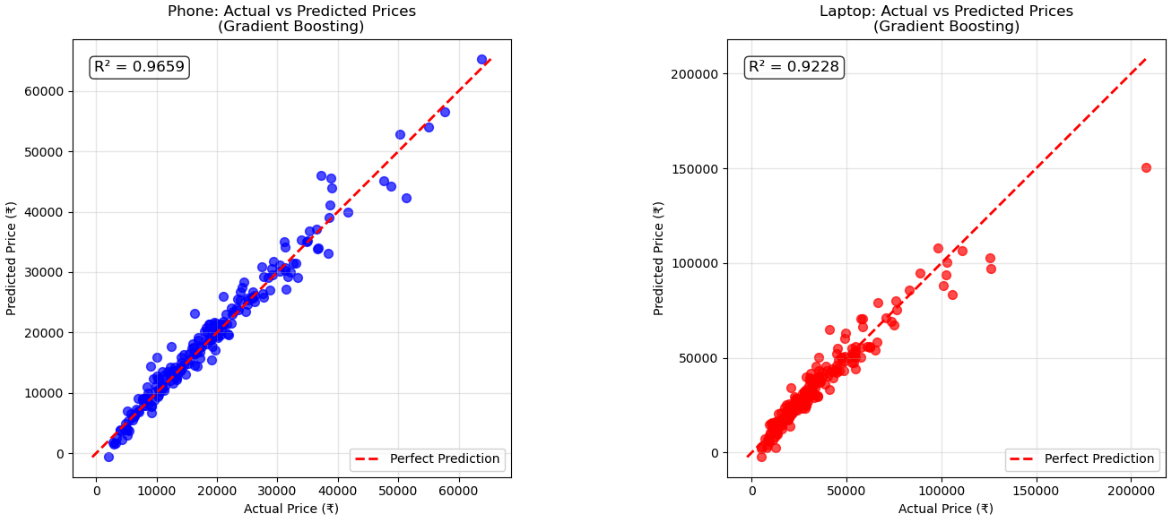


```
FINAL MODEL PERFORMANCE SUMMARY
==================================================
Best Phone Model: Gradient Boosting
  R² Score: 0.9659
  MAE: 1466
  RMSE: 2096
  CV R²: 0.9599

Best Laptop Model: Gradient Boosting
  R² Score: 0.9228
  MAE: 4509
  RMSE: 7273
  CV R²: 0.8809

Average Prediction Error:
  Phones: 10.3%
  Laptops: 15.7%

Dataset Statistics:
  Phone records: 1000
  Laptop records: 1000
  Total records: 2000
```

In [15]:

```python
# Price vs Prediction graph
fig, axes = plt.subplots(1, 2, figsize=(15, 6))

# Phone price vs prediction
if y_phone_pred is not None and y_phone_test is not None:
    axes[0].scatter(y_phone_test, y_phone_pred, alpha=0.7, color='blue', s=50)

    # Perfect prediction line
    max_val = max(y_phone_test.max(), y_phone_pred.max())
    min_val = min(y_phone_test.min(), y_phone_pred.min())
    axes[0].plot([min_val, max_val], [min_val, max_val], 'r--', linewidth=2, lab

    # Add R² value to plot
    r2_phone = r2_score(y_phone_test, y_phone_pred)
    axes[0].text(0.05, 0.95, f'R² = {r2_phone:.4f}', transform=axes[0].transAxes
                 fontsize=12, verticalalignment='top', bbox=dict(boxstyle='round'

    axes[0].set_xlabel('Actual Price (₹)')
    axes[0].set_ylabel('Predicted Price (₹)')
    axes[0].set_title(f'Phone: Actual vs Predicted Prices\n({best_phone_model_na
    axes[0].legend()
    axes[0].grid(True, alpha=0.3)

    # Set equal aspect ratio
    axes[0].set_aspect('equal')
else:
    axes[0].text(0.5, 0.5, 'No Phone Prediction Data', ha='center', va='center',
    axes[0].set_title('Phone: Actual vs Predicted (No Data)')

# Laptop price vs prediction
if y_laptop_pred is not None and y_laptop_test is not None:
    axes[1].scatter(y_laptop_test, y_laptop_pred, alpha=0.7, color='red', s=50)

    # Perfect prediction line
    max_val = max(y_laptop_test.max(), y_laptop_pred.max())
    min_val = min(y_laptop_test.min(), y_laptop_pred.min())
    axes[1].plot([min_val, max_val], [min_val, max_val], 'r--', linewidth=2, lab
```

```
    # Add R² value to plot
    r2_laptop = r2_score(y_laptop_test, y_laptop_pred)
    axes[1].text(0.05, 0.95, f'R² = {r2_laptop:.4f}', transform=axes[1].transAxe
                fontsize=12, verticalalignment='top', bbox=dict(boxstyle='round'

    axes[1].set_xlabel('Actual Price (₹)')
    axes[1].set_ylabel('Predicted Price (₹)')
    axes[1].set_title(f'Laptop: Actual vs Predicted Prices\n({best_laptop_model_
    axes[1].legend()
    axes[1].grid(True, alpha=0.3)

    # Set equal aspect ratio
    axes[1].set_aspect('equal')
else:
    axes[1].text(0.5, 0.5, 'No Laptop Prediction Data', ha='center', va='center'
    axes[1].set_title('Laptop: Actual vs Predicted (No Data)')

plt.tight_layout()
plt.show()

# Additional detailed analysis
print("\nPRICE PREDICTION ACCURACY ANALYSIS")
print("="*50)

if y_phone_pred is not None and y_phone_test is not None:
    phone_accuracy = (1 - (abs(y_phone_test - y_phone_pred) / y_phone_test)) * 1
    print(f"Phone Prediction Accuracy:")
    print(f"  Mean Accuracy: {phone_accuracy.mean():.1f}%")
    print(f"  Median Accuracy: {phone_accuracy.median():.1f}%")
    print(f"  Accuracy Range: {phone_accuracy.min():.1f}% to {phone_accuracy.max

if y_laptop_pred is not None and y_laptop_test is not None:
    laptop_accuracy = (1 - (abs(y_laptop_test - y_laptop_pred) / y_laptop_test))
    print(f"\nLaptop Prediction Accuracy:")
    print(f"  Mean Accuracy: {laptop_accuracy.mean():.1f}%")
    print(f"  Median Accuracy: {laptop_accuracy.median():.1f}%")
    print(f"  Accuracy Range: {laptop_accuracy.min():.1f}% to {laptop_accuracy.m
```



```
PRICE PREDICTION ACCURACY ANALYSIS
==================================================
Phone Prediction Accuracy:
  Mean Accuracy: 89.7%
  Median Accuracy: 93.5%
  Accuracy Range: -32.9% to 99.9%

Laptop Prediction Accuracy:
  Mean Accuracy: 84.3%
  Median Accuracy: 88.9%
  Accuracy Range: -48.8% to 99.8%
```

In [ ]: