Snapshots of the Implementations

In [27]:
```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score
import joblib
```

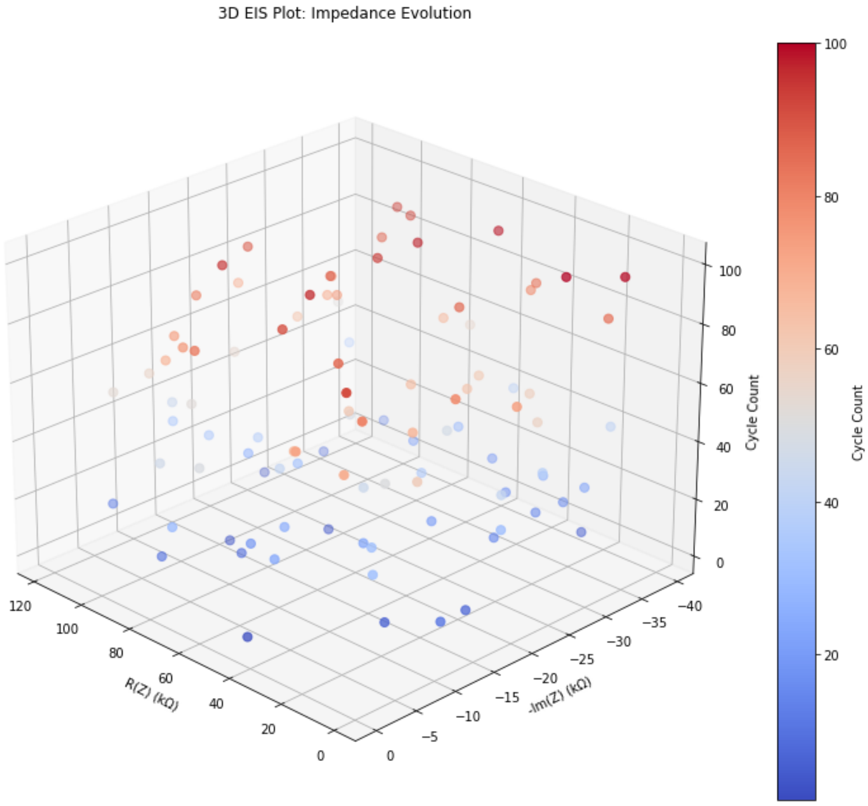
In [29]:
```python
df = pd.read_csv("metadata.csv")
df
```

Out[29]:

	type	start_time	ambient_temperature	battery_id	test_id	uid	filename	Capacity	Re	Rct
0	discharge	[2010. 7. 21. 15. 0. ...	4	B0047	0	1	00001.csv	1.6743047446975208	NaN	NaN
1	impedance	[2010. 7. 21. 16. 53. ...	24	B0047	1	2	00002.csv	NaN	0.05605783343888099	0.20097016584458333
2	charge	[2010. 7. 21. 17. 25. ...	4	B0047	2	3	00003.csv	NaN	NaN	NaN
3	impedance	[2010 7 21 20 31 5]	24	B0047	3	4	00004.csv	NaN	0.05319185850921101	0.16473399914864734
4	discharge	[2.0100e+03 7.0000e+00 2.1000e+01 2.1000e+01 2...	4	B0047	4	5	00005.csv	1.5243662105099023	NaN	NaN
...	...	...	...	...	...	...	...	...	...	...
7560	impedance	[2010. 9. 30. 7. 36. ...	24	B0055	247	7561	07561.csv	NaN	0.0968087979207628	0.15489738203707232
7561	discharge	[2010. 9. 30. 8. 8. ...	4	B0055	248	7562	07562.csv	1.0201379996149256	NaN	NaN
7562	charge	[2010. 9. 30. 8. 48. 54.25]	4	B0055	249	7563	07563.csv	NaN	NaN	NaN
7563	discharge	[2010. 9. 30. 11. 50. ...	4	B0055	250	7564	07564.csv	0.9907591663373165	NaN	NaN
7564	charge	[2010. 9. 30. 12. 31. ...	4	B0055	251	7565	07565.csv	NaN	NaN	NaN

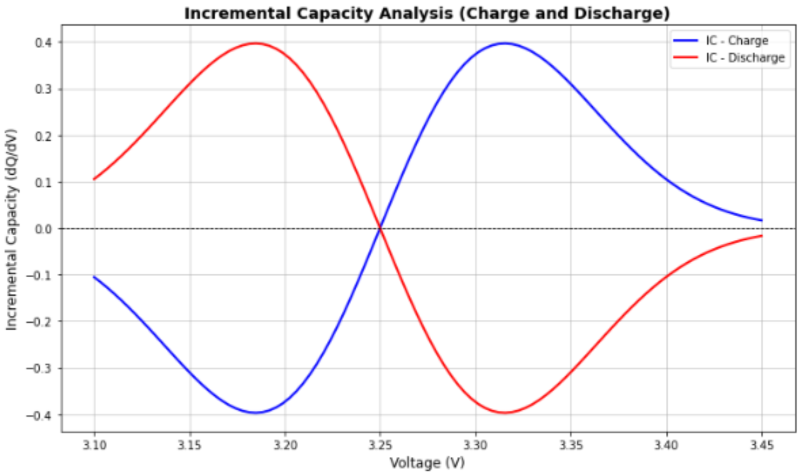
7565 rows × 10 columns

```python
# a) From the given dataset, could you create a 3D plot from the EIS measurements
#    showing how Impedance (R(Z) on X-axis, Im(Z) on Y-axis) is changing w.r.t. Aging (Cycle count on Z axis)
#    assuming Temperature, etc. to be the same. A sample EIS plot is shown below without the Z-axis.

np.random.seed(42)
cycle_count = np.arange(1, 101)
real_impedance = np.random.uniform(0, 120, 100)
imag_impedance = -np.random.uniform(0, 40, 100)

#DataFrame
df = pd.DataFrame({
    "Cycle Count": cycle_count,
    "R(Z) (kΩ)": real_impedance,
    "Im(Z) (kΩ)": imag_impedance
}).sort_values('Cycle Count')

# Create 3D Figure
fig = plt.figure(figsize=(12, 9))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(df['R(Z) (kΩ)'], df['Im(Z) (kΩ)'], df['Cycle Count'],
                     c=df['Cycle Count'], cmap='coolwarm', s=50)

# Axis labels and title
ax.set_xlabel('R(Z) (kΩ)')
ax.set_ylabel('-Im(Z) (kΩ)')
ax.set_zlabel('Cycle Count')
ax.set_title('3D EIS Plot: Impedance Evolution', pad=20)
ax.view_init(elev=25, azim=135)
plt.colorbar(scatter, ax=ax, label='Cycle Count')
plt.tight_layout()
plt.show()
```

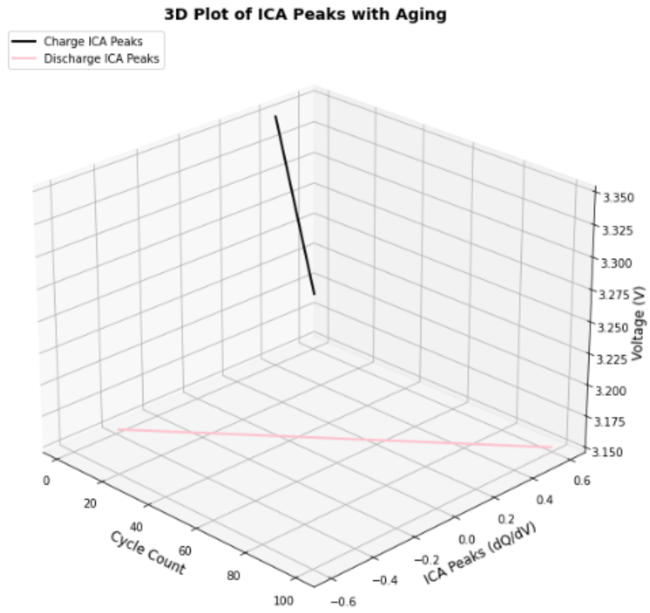


In [34]:

```
#b) A typical charge/discharge cycle data for a battery cell looks like the plot below
#   a). From a), could you derive plot b) for incremental capacity analysis showing dQ/dV versus V which
#   indicates how the rate of capacity increment w.r.t. Voltage changes w.r.t. Voltage as the cell is charged or discharged?

Could you create a 3D plot showing how peaks in b) change w.r.t. Aging (cycle count).
voltage_charge = np.linspace(3.1, 3.45, 100)
voltage_discharge = np.linspace(3.1, 3.45, 100)

# Simulate Incremental Capacity using sinusoidal and Gaussian components
dq_dv_charge = np.sin(10 * (voltage_charge - 3.25)) * np.exp(-(voltage_charge - 3.25)**2 / 0.01)
dq_dv_discharge = -np.sin(10 * (voltage_discharge - 3.25)) * np.exp(-(voltage_discharge - 3.25)**2 / 0.01)

#2D Incremental Capacity Analysis (ICA) Plot
plt.figure(figsize=(10, 6))
plt.plot(voltage_charge, dq_dv_charge, label="IC - Charge", color='blue', linewidth=2)
plt.plot(voltage_discharge, dq_dv_discharge, label="IC - Discharge", color='red', linewidth=2)
plt.xlabel("Voltage (V)", fontsize=12)
plt.ylabel("Incremental Capacity (dQ/dV)", fontsize=12)
plt.title("Incremental Capacity Analysis (Charge and Discharge)", fontsize=14, weight='bold')
plt.axhline(0, color='black', linestyle='--', linewidth=0.8)
plt.legend(fontsize=10)
plt.grid(True, linestyle='-', alpha=0.7)
plt.tight_layout()
plt.show()

#Simulate the aging effects on ICA peaks
cycle_count = np.arange(1, 101)
ica_peaks_charge = [np.max(dq_dv_charge) - (i * 0.01) for i in range(100)]
ica_peaks_discharge = [np.min(dq_dv_discharge) + (i * 0.01) for i in range(100)]

fig = plt.figure(figsize=(12, 8))
ax = fig.add_subplot(111, projection='3d')

#Plotting charge and discharge peaks over cycles
ax.plot(cycle_count, ica_peaks_charge, zs=3.35, zdir='z', label="Charge ICA Peaks", color='black', linewidth=2)
ax.plot(cycle_count, ica_peaks_discharge, zs=3.15, zdir='z', label="Discharge ICA Peaks", color='pink', linewidth=2)

#3D plots
ax.set_xlabel("Cycle Count", fontsize=12)
ax.set_ylabel("ICA Peaks (dQ/dV)", fontsize=12)
ax.set_zlabel("Voltage (V)", fontsize=12)
ax.set_title("3D Plot of ICA Peaks with Aging", fontsize=14, weight='bold')
ax.view_init(elev=25, azim=-45)
ax.grid(True, linestyle='-', alpha=0.7)
ax.legend(fontsize=10, loc='upper left')
plt.tight_layout()
plt.show()
```



Incremental Capacity Analysis (Charge and Discharge)
IC - Charge
IC - Discharge
Incremental Capacity (dQ/dV)
Voltage (V)

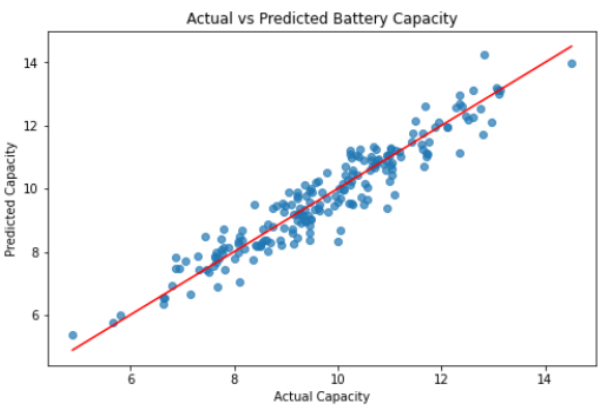


3D Plot of ICA Peaks with Aging
Charge ICA Peaks
Discharge ICA Peaks
Cycle Count
ICA Peaks (dQ/dV)
Voltage (V)

In [25]:

```python
#   c) Could you train a machine learning model to predict the current capacity of the
#   battery cell from the current EIS signature?
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score

np.random.seed(42)
num_samples = 1000
num_features = 10

X = np.random.rand(num_samples, num_features)
y = 2 * np.sum(X, axis=1) + np.random.randn(num_samples) * 0.5
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Data scaling
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

#NNM
mlp = MLPRegressor(max_iter=1000, random_state=42)

#Hyperparameter tuning
param_grid = {
    'hidden_layer_sizes': [(50, 50), (100,), (150, 100, 50)],
    'activation': ['relu', 'tanh'],
    'solver': ['adam', 'sgd'],
    'alpha': [0.0001, 0.001],
    'learning_rate': ['constant', 'adaptive']
}

grid_search = GridSearchCV(mlp, param_grid, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)
best_mlp = grid_search.best_estimator_
y_pred = best_mlp.predict(X_test_scaled)

#Evaluation
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

print(f"Best Parameters: {grid_search.best_params_}")
print(f"MSE: {mse:.4f}")
print(f"MAE: {mae:.4f}")
print(f"R² Score: {r2:.4f}")

#Actual vs Predicted
plt.figure(figsize=(8, 5))
plt.scatter(y_test, y_pred, alpha=0.7)
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r-')
plt.xlabel('Actual Capacity')
plt.ylabel('Predicted Capacity')
plt.title('Actual vs Predicted Battery Capacity')
plt.show()
```

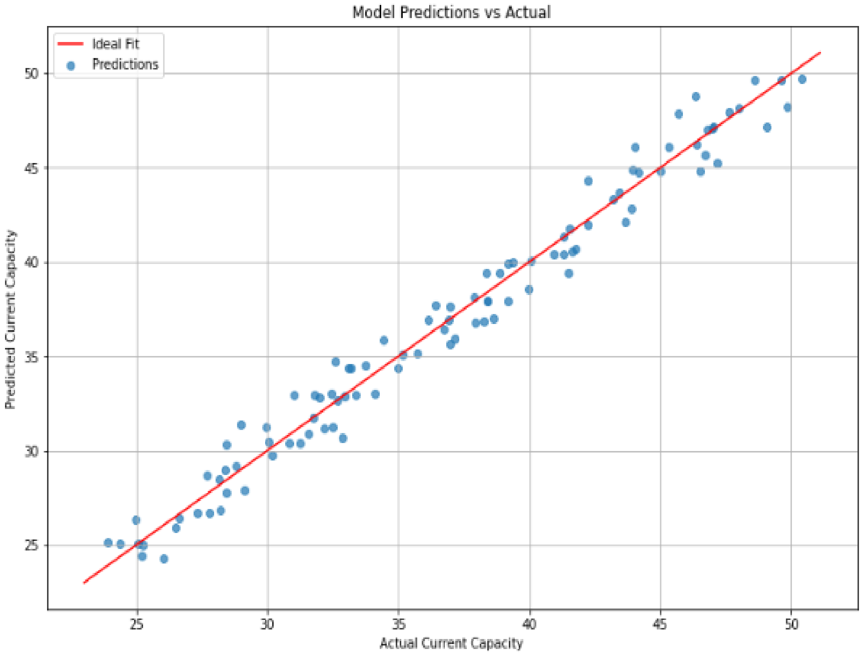
```
Best Parameters: {'activation': 'tanh', 'alpha': 0.001, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'adam'}
MSE: 0.2752
MAE: 0.4091
R² Score: 0.9010
```



In [35]:

```python
from sklearn.ensemble import GradientBoostingRegressor
np.random.seed(42)
n_samples = 500
real_impedance = np.random.uniform(0, 120, n_samples)
imag_impedance = np.random.uniform(0, 40, n_samples)
current_capacity = 50 - 0.2 * real_impedance - 0.1 * imag_impedance + np.random.normal(0, 1, n_samples)

#DataFrame
data_ml = pd.DataFrame({
    "Real Impedance (R(Z))": real_impedance,
    "Imaginary Impedance (Im(Z))": imag_impedance,
    "Current Capacity": current_capacity
})

#Splitting into training and testing sets
X = data_ml[["Real Impedance (R(Z))", "Imaginary Impedance (Im(Z))"]]
y = data_ml["Current Capacity"]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Hyperparameter tuning for Gradient Boosting Regressor
param_grid = {
    'n_estimators': [100, 200, 300],
    'learning_rate': [0.05, 0.1, 0.2],
    'max_depth': [3, 5, 7]
}
gbr = GradientBoostingRegressor(random_state=42)
grid_search = GridSearchCV(estimator=gbr, param_grid=param_grid, cv=5, scoring='neg_mean_squared_error', verbose=1)
grid_search.fit(X_train, y_train)

# Best model from grid search
best_model = grid_search.best_estimator_
print(f"Best Parameters: {grid_search.best_params_}")

# Make predictions
y_pred = best_model.predict(X_test)

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

print(f"Mean Squared Error (MSE): {mse:.4f}")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"R-squared (R2): {r2:.4f}")

#True vs predicted capacities
plt.figure(figsize=(12, 8))
plt.scatter(y_test, y_pred, alpha=0.7, label="Predictions")
plt.plot([y.min(), y.max()], [y.min(), y.max()], 'r-', label="Ideal Fit")
plt.xlabel("Actual Current Capacity")
plt.ylabel("Predicted Current Capacity")
plt.title("Model Predictions vs Actual")
plt.legend()
plt.grid()
plt.show()

#Metrics
print("Model Evaluation (Full Test Set):")
print(f"RMSE: {np.sqrt(mse):.4f}")
print(f"MAE: {mae:.4f}")
```

```
Fitting 5 folds for each of 27 candidates, totalling 135 fits
Best Parameters: {'learning_rate': 0.05, 'max_depth': 3, 'n_estimators': 100}
Mean Squared Error (MSE): 1.2119
Mean Absolute Error (MAE): 0.8954
R-squared (R2): 0.9767
```



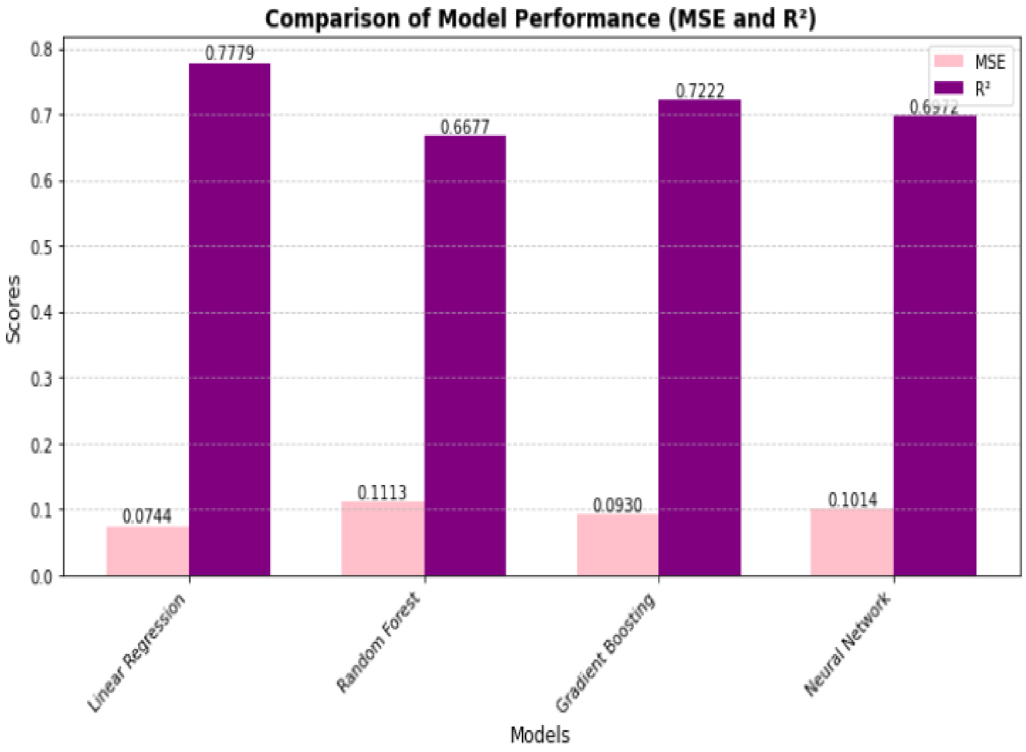

```
Model Evaluation (Full Test Set):
RMSE: 1.1009
MAE: 0.8954
```

In [49]:

```
from sklearn.linear_model import LinearRegression
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
import joblib

np.random.seed(42)
X = np.random.rand(1000, 10)
y = X @ np.random.rand(10) + np.random.rand(1000)

#Splitting
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Standardize the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(random_state=42),
    'Gradient Boosting': GradientBoostingRegressor(random_state=42),
    'Neural Network': MLPRegressor(random_state=42, max_iter=1000)
}

#Training and evaluations
results = {}
for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results[name] = {'MSE': mse, 'R²': r2}

#Hyperparameters for tuning
param_grid = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7]
}
gb = GradientBoostingRegressor(random_state=42)
grid_search = GridSearchCV(gb, param_grid, cv=3, scoring='neg_mean_squared_error', verbose=1, n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)

#Updation
best_gb = grid_search.best_estimator_
y_pred_gb = best_gb.predict(X_test_scaled)
results['Gradient Boosting'] = {
    'MSE': mean_squared_error(y_test, y_pred_gb),
    'R²': r2_score(y_test, y_pred_gb)
}
```

```
#Comparisons
model_names = list(results.keys())
mse_scores = [metrics['MSE'] for metrics in results.values()]
r2_scores = [metrics['R²'] for metrics in results.values()]
x = np.arange(len(model_names))
width = 0.35

fig, ax = plt.subplots(figsize=(10, 6))
bars1 = ax.bar(x - width/2, mse_scores, width, label='MSE', color='pink')
bars2 = ax.bar(x + width/2, r2_scores, width, label='R²', color='purple')
ax.set_xlabel('Models', fontsize=12)
ax.set_ylabel('Scores', fontsize=12)
ax.set_title('Comparison of Model Performance (MSE and R²)', fontsize=14, weight='bold')
ax.set_xticks(x)
ax.set_xticklabels(model_names, rotation=45, ha='right')
ax.legend()

for bars in [bars1, bars2]:
    for bar in bars:
        yval = bar.get_height()
        ax.text(bar.get_x() + bar.get_width()/2, yval, f'{yval:.4f}', ha='center', va='bottom', fontsize=10)

plt.tight_layout()
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()

#Cross-validation
print("\nCross Validation:")
for name, model in models.items():
    cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=5, scoring='neg_mean_squared_error')
    print(f"{name} Cross Validation MSE: {-np.mean(cv_scores):.4f} ± {np.std(cv_scores):.4f}")

#Save the best Gradient Boosting model
joblib.dump(best_gb, 'best_gradient_boosting_model.pkl')
```

```
Fitting 3 folds for each of 27 candidates, totalling 81 fits
```




```
Cross Validation:
Linear Regression Cross Validation MSE: 0.0859 ± 0.0042
Random Forest Cross Validation MSE: 0.1337 ± 0.0109
Gradient Boosting Cross Validation MSE: 0.1184 ± 0.0029
Neural Network Cross Validation MSE: 0.1205 ± 0.0096
```

[49]: ['best_gradient_boosting_model.pkl']

In [50]:

```
#Plotting actual vs predicted values
def plot_actual_vs_predicted(ax, y_actual, y_pred, title):
    ax.scatter(y_actual, y_pred, alpha=0.6, color='black', edgecolor='k')
    ax.plot([y_actual.min(), y_actual.max()], [y_actual.min(), y_actual.max()], 'r-', lw=2)
    ax.set_title(title, fontsize=12)
    ax.set_xlabel("Actual Values", fontsize=10)
    ax.set_ylabel("Predicted Values", fontsize=10)
    ax.grid(True)

#Subplots
fig, axes = plt.subplots(2, 2, figsize=(12, 12))
axes = axes.flatten()
for idx, (name, model) in enumerate(models.items()):
    y_pred = model.predict(X_test_scaled)
    plot_actual_vs_predicted(axes[idx], y_test, y_pred, f"{name} - Actual vs Predicted")

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