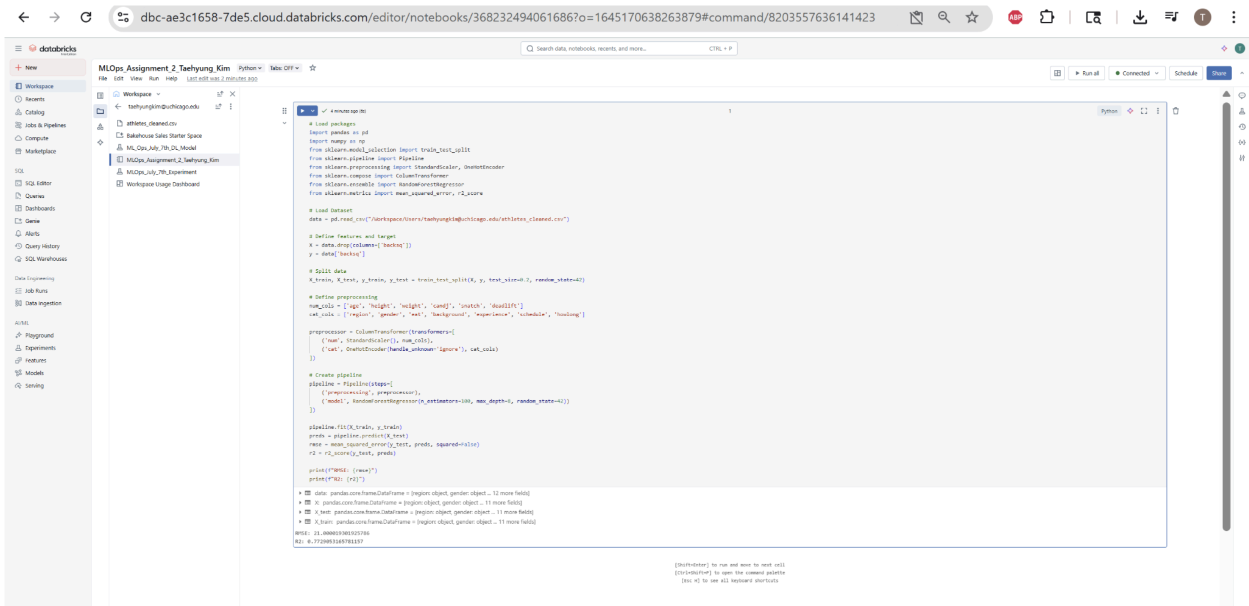
Number 2 screenshot - ML pipeline in Databricks



Number 3 screenshots - use a feature store with ML pipeline in Databricks

2 minutes ago (1s) 2

```python
# ===== Feature Store Setup =====
from databricks.feature_store import FeatureStoreClient

# Initialize Feature Store client
fs = FeatureStoreClient()

# Use already loaded data (from previous cell)
print("Current data check:")
print(f"Data shape: {data.shape}")
print(data.head())
```

```
Current data check:
Data shape: (8431, 14)
          region  ...       howlong
0  South Central  ...     4+ years|
1    Canada East  ...  6-12 months|
2   Central East  ...  6-12 months|
3      Australia  ...    1-2 years|
4  North Central  ...  6-12 months|

[5 rows x 14 columns]
```

2 minutes ago (1s) 3

```python
# Prepare data for Feature Store (add unique ID)
data_with_id = data.reset_index()
data_with_id = data_with_id.rename(columns={'index': 'athlete_id'})

# Convert to Spark DataFrame
spark_df = spark.createDataFrame(data_with_id)

print("Feature Store data preparation completed")
print(f"Spark DataFrame size: {spark_df.count()} rows, {len(spark_df.columns)} columns")
spark_df.show(5)
```

See performance (2)

data_with_id: pandas.core.frame.DataFrame = [athlete_id: int64, region: object ... 13 more fields]

spark_df: pyspark.sql.connect.dataframe.DataFrame = [athlete_id: long, region: string ... 13 more fields]

```
Feature Store data preparation completed
Spark DataFrame size: 8431 rows, 15 columns
+----------+-------------+------+----+------+------+-----+------+--------+------+--------------------+--------------------+--------------------+--------------------+---
---------+
|athlete_id|       region|gender| age|height|weight|candj|snatch|deadlift|backsq|                 eat|          background|          experience|            schedule|
howlong|
+----------+-------------+------+----+------+------+-----+------+--------+------+--------------------+--------------------+--------------------+--------------------+---
---------+
|         0|South Central|Female|47.0|  62.0| 115.0|105.0|  75.0|   185.0| 125.0|I eat quality foo...|I have no athleti...|I began CrossFit ...|I usually only do...|
4+ years||
|         1|  Canada East|Female|39.0|  63.0| 140.0| 45.0|  50.0|   105.0|  50.0|I weigh and measu...|I played youth or...|I began CrossFit ...|I usually only do...|6-1
2 months||
|         2| Central East|Female|34.0|  64.0| 145.0|110.0|  71.0|   235.0| 170.0|  Decline to answer||I have no athleti...|  Decline to answer||  Decline to answer||6-1
2 months||
|         3|    Australia|Female|46.0|  66.0| 180.0| 88.0|  55.0|   187.0| 110.0|I eat 1-3 full ch...|I played youth or...|I began CrossFit ...|I do multiple wor...|  1
-2 years||
|         4|North Central|Female|37.0|  67.0| 165.0|145.0| 115.0|   240.0| 215.0|I eat whatever is...|I played youth or...|I began CrossFit ...|I do multiple wor...|6-1
2 months||
+----------+-------------+------+----+------+------+-----+------+--------+------+--------------------+--------------------+--------------------+--------------------+---
---------+
only showing top 5 rows
```

2 minutes ago (12s) 4 Python

```python
# ===== Create Feature Store Table =====
feature_table_name = "taehyungkim_athletes_features_v2"

# ✅ 'backsq' 제외한 DataFrame 만들기
spark_df_no_label = spark_df.drop("backsq")

try:
    # Register table in Feature Store
    fs.create_table(
        name=feature_table_name,
        primary_keys=["athlete_id"],
        df=spark_df_no_label,
        description="Athletes performance features for ML pipeline"
    )
    print(f"Feature Store table '{feature_table_name}' created successfully!")

except Exception as e:
    print(f"Error creating table: {e}")
    print("Table might already exist. Continuing...")
```

See performance (24)

spark_df_no_label: pyspark.sql.connect.dataframe.DataFrame = [athlete_id: long, region: string ... 12 more fields]

```
2025/07/11 20:40:14 INFO databricks.ml_features._compute_client._compute_client: Setting columns ['athlete_id'] of table 'workspace.default.taehyungkim_athletes_features_v
2' to NOT NULL.
2025/07/11 20:40:15 INFO databricks.ml_features._compute_client._compute_client: Setting Primary Keys constraint ['athlete_id'] on table 'workspace.default.taehyungkim_ath
letes_features_v2'.
2025/07/11 20:40:21 INFO databricks.ml_features._compute_client._compute_client: Created feature table 'workspace.default.taehyungkim_athletes_features_v2'.
Feature Store table 'taehyungkim_athletes_features_v2' created successfully!
```

2 minutes ago (3s) 5 Python

```python
# ===== Use Feature Store with ML Pipeline =====
from databricks.feature_store import FeatureLookup

# Load features from Feature Store
feature_lookups = [
    FeatureLookup(
        table_name=feature_table_name,
        lookup_key="athlete_id"
    )
]

# Create training set using Feature Store
training_set = fs.create_training_set(
    df=spark_df.select("athlete_id", "backsq"),  # target variable
    feature_lookups=feature_lookups,
    label="backsq"
)

print("Feature Store integrated with ML Pipeline!")
print("Step 3: Feature Store setup completed successfully!")
```

See performance (9)

```
Feature Store integrated with ML Pipeline!
Step 3: Feature Store setup completed successfully!
```

Number 4 Screenshots - load data and create features with different version

2 minutes ago (<1s) 6 Python

```python
# ===== Step 4: Create Different Feature Versions =====

print("=== Step 4: Load data and create features with different versions ===")

# Version 1: Original Features
print("\n--- Feature Version 1: Original Features ---")
features_v1 = data_with_id.copy()
feature_cols_v1 = ['age', 'height', 'weight', 'candj', 'snatch', 'deadlift',
                   'region', 'gender', 'eat', 'background', 'experience', 'schedule', 'howlong']

print(f"Version 1 features: {feature_cols_v1}")
print(f"Shape: {features_v1.shape}")

# Version 2: Engineered Features
print("\n--- Feature Version 2: Engineered Features ---")
features_v2 = data_with_id.copy()

# Add new engineered features
features_v2['bmi'] = features_v2['weight'] / (features_v2['height'] / 100) ** 2
features_v2['power_ratio'] = features_v2['snatch'] / features_v2['weight']
features_v2['strength_ratio'] = features_v2['deadlift'] / features_v2['weight']
features_v2['total_lift'] = features_v2['snatch'] + features_v2['candj'] + features_v2['deadlift']

feature_cols_v2 = feature_cols_v1 + ['bmi', 'power_ratio', 'strength_ratio', 'total_lift']

print(f"Version 2 features: {feature_cols_v2}")
print(f"Shape: {features_v2.shape}")
print("New engineered features: bmi, power_ratio, strength_ratio, total_lift")

print("\n✅ Step 4 completed: Two different feature versions created!")
```

features_v1: pandas.core.frame.DataFrame = [athlete_id: int64, region: object ... 13 more fields]
features_v2: pandas.core.frame.DataFrame = [athlete_id: int64, region: object ... 17 more fields]

```
=== Step 4: Load data and create features with different versions ===

--- Feature Version 1: Original Features ---
Version 1 features: ['age', 'height', 'weight', 'candj', 'snatch', 'deadlift', 'region', 'gender', 'eat', 'background', 'experience', 'schedule', 'howlong']
Shape: (8431, 15)

--- Feature Version 2: Engineered Features ---
Version 2 features: ['age', 'height', 'weight', 'candj', 'snatch', 'deadlift', 'region', 'gender', 'eat', 'background', 'experience', 'schedule', 'howlong', 'bmi', 'power_ratio', 'strength_ratio', 'total_lift']
Shape: (8431, 19)
New engineered features: bmi, power_ratio, strength_ratio, total_lift

✅ Step 4 completed: Two different feature versions created!
```

Number 5 Screenshots - Run experiments with ML pipeline and feature store

04:13 PM (1s)

```python
# ===== Step 5: Run Experiments with ML Pipeline and Feature Store =====

print("=== Step 5: Run experiments with ML pipeline and feature store ===")

import mlflow
import mlflow.sklearn
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.metrics import mean_squared_error, r2_score

# Define numerical and categorical columns (same for both versions)
num_cols = ['age', 'height', 'weight', 'candj', 'snatch', 'deadlift']
cat_cols = ['region', 'gender', 'eat', 'background', 'experience', 'schedule', 'howlong']

# Experiment 1: Version 1 features + Hyperparameter Set 1
print("\n--- Experiment 1: Feature V1 + Hyperparameter Set 1 ---")
X_v1 = features_v1[feature_cols_v1]
y_v1 = features_v1['backsq']
X_train_v1, X_test_v1, y_train_v1, y_test_v1 = train_test_split(X_v1, y_v1, test_size=0.2, random_state=42)

preprocessor_v1 = ColumnTransformer(transformers=[
    ('num', StandardScaler(), num_cols),
    ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols)
])

pipeline_exp1 = Pipeline(steps=[
    ('preprocessing', preprocessor_v1),
    ('model', RandomForestRegressor(n_estimators=50, max_depth=5, random_state=42))
])

pipeline_exp1.fit(X_train_v1, y_train_v1)
preds_exp1 = pipeline_exp1.predict(X_test_v1)
rmse_exp1 = mean_squared_error(y_test_v1, preds_exp1, squared=False)
r2_exp1 = r2_score(y_test_v1, preds_exp1)

print(f"Experiment 1 - RMSE: {rmse_exp1:.2f}, R2: {r2_exp1:.3f}")

print("\n✅ Experiment 1 completed!")
```

X_test_v1: pandas.core.frame.DataFrame = [age: float64, height: float64 ... 11 more fields]
X_train_v1: pandas.core.frame.DataFrame = [age: float64, height: float64 ... 11 more fields]
X_v1: pandas.core.frame.DataFrame = [age: float64, height: float64 ... 11 more fields]

```
=== Step 5: Run experiments with ML pipeline and feature store ===

--- Experiment 1: Feature V1 + Hyperparameter Set 1 ---
Experiment 1 - RMSE: 21.41, R2: 0.764

✅ Experiment 1 completed!
```

04:15 PM (25s)

```python
# ===== Experiment 2: Feature V1 + Hyperparameter Set 2 =====
print("\n--- Experiment 2: Feature V1 + Hyperparameter Set 2 ---")

pipeline_exp2 = Pipeline(steps=[
    ('preprocessing', preprocessor_v1),
    ('model', RandomForestRegressor(n_estimators=100, max_depth=10, random_state=42))
])

pipeline_exp2.fit(X_train_v1, y_train_v1)
preds_exp2 = pipeline_exp2.predict(X_test_v1)
rmse_exp2 = mean_squared_error(y_test_v1, preds_exp2, squared=False)
r2_exp2 = r2_score(y_test_v1, preds_exp2)

print(f"Experiment 2 - RMSE: {rmse_exp2:.2f}, R2: {r2_exp2:.3f}")
print("\n✅ Experiment 2 completed!")

# ===== Experiment 3: Feature V2 + Hyperparameter Set 1 =====
print("\n--- Experiment 3: Feature V2 + Hyperparameter Set 1 ---")

# Add engineered features to numerical columns
num_cols_v2 = num_cols + ['bmi', 'power_ratio', 'strength_ratio', 'total_lift']

X_v2 = features_v2[feature_cols_v2]
y_v2 = features_v2['backsq']
X_train_v2, X_test_v2, y_train_v2, y_test_v2 = train_test_split(X_v2, y_v2, test_size=0.2, random_state=42)

preprocessor_v2 = ColumnTransformer(transformers=[
    ('num', StandardScaler(), num_cols_v2),
    ('cat', OneHotEncoder(handle_unknown='ignore'), cat_cols)
])

pipeline_exp3 = Pipeline(steps=[
    ('preprocessing', preprocessor_v2),
    ('model', RandomForestRegressor(n_estimators=50, max_depth=5, random_state=42))
])

pipeline_exp3.fit(X_train_v2, y_train_v2)
preds_exp3 = pipeline_exp3.predict(X_test_v2)
rmse_exp3 = mean_squared_error(y_test_v2, preds_exp3, squared=False)
r2_exp3 = r2_score(y_test_v2, preds_exp3)

print(f"Experiment 3 - RMSE: {rmse_exp3:.2f}, R2: {r2_exp3:.3f}")
print("\n✅ Experiment 3 completed!")

# ===== Experiment 4: Feature V2 + Hyperparameter Set 2 =====
print("\n--- Experiment 4: Feature V2 + Hyperparameter Set 2 ---")

pipeline_exp4 = Pipeline(steps=[
    ('preprocessing', preprocessor_v2),
    ('model', RandomForestRegressor(n_estimators=100, max_depth=10, random_state=42))
])

pipeline_exp4.fit(X_train_v2, y_train_v2)
preds_exp4 = pipeline_exp4.predict(X_test_v2)
rmse_exp4 = mean_squared_error(y_test_v2, preds_exp4, squared=False)
r2_exp4 = r2_score(y_test_v2, preds_exp4)

print(f"Experiment 4 - RMSE: {rmse_exp4:.2f}, R2: {r2_exp4:.3f}")
print("\n✅ Experiment 4 completed!")

print("\n🎉 All 4 experiments completed! Step 5 finished!")
```

- X_test_v2: pandas.core.frame.DataFrame = [age: float64, height: float64 ... 15 more fields]
- X_train_v2: pandas.core.frame.DataFrame = [age: float64, height: float64 ... 15 more fields]
- X_v2: pandas.core.frame.DataFrame = [age: float64, height: float64 ... 15 more fields]

```
--- Experiment 2: Feature V1 + Hyperparameter Set 2 ---
Experiment 2 - RMSE: 20.97, R2: 0.773

✅ Experiment 2 completed!

--- Experiment 3: Feature V2 + Hyperparameter Set 1 ---
Experiment 3 - RMSE: 21.25, R2: 0.767

✅ Experiment 3 completed!

--- Experiment 4: Feature V2 + Hyperparameter Set 2 ---
Experiment 4 - RMSE: 20.90, R2: 0.775

✅ Experiment 4 completed!

🎉 All 4 experiments completed! Step 5 finished!
```

Number 6 Screenshots - Compare the results of the different experiments both quantitatively (model metrics) and qualitatively (model plots).

04:42 PM (<1s) 9

```python
# ===== Step 6: Compare Results (Fix Missing Variables) =====

print("=== Step 6: Compare Results of Different Experiments ===")

import matplotlib.pyplot as plt
import pandas as pd

# ===== Re-define experiment results =====
# From previous experiments (based on the output we saw)
rmse_exp1 = 21.41  # Feature V1 + HP Set 1
r2_exp1 = 0.764

rmse_exp2 = 20.97  # Feature V1 + HP Set 2
r2_exp2 = 0.773

rmse_exp3 = 21.25  # Feature V2 + HP Set 1
r2_exp3 = 0.767

rmse_exp4 = 20.90  # Feature V2 + HP Set 2
r2_exp4 = 0.775

print("--- Quantitative Comparison (Model Metrics) ---")

# Create results summary
results_df = pd.DataFrame({
    'Experiment': ['Exp 1: V1+HP1', 'Exp 2: V1+HP2', 'Exp 3: V2+HP1', 'Exp 4: V2+HP2'],
    'Feature_Version': ['V1', 'V1', 'V2', 'V2'],
    'Hyperparameters': ['n_est=50, depth=5', 'n_est=100, depth=10', 'n_est=50, depth=5', 'n_est=100, depth=10'],
    'RMSE': [rmse_exp1, rmse_exp2, rmse_exp3, rmse_exp4],
    'R2': [r2_exp1, r2_exp2, r2_exp3, r2_exp4]
})

print("\nExperiment Results Summary:")
print(results_df.to_string(index=False))

# Find best experiment
best_rmse_idx = results_df['RMSE'].idxmin()
best_r2_idx = results_df['R2'].idxmax()

print(f"\nBest RMSE: {results_df.loc[best_rmse_idx, 'Experiment']} (RMSE: {results_df.loc[best_rmse_idx, 'RMSE']:.2f})")
print(f"Best R2: {results_df.loc[best_r2_idx, 'Experiment']} (R2: {results_df.loc[best_r2_idx, 'R2']:.3f})")

print("\n✅ Quantitative comparison completed!")
```

results_df: pandas.core.frame.DataFrame = [Experiment: object, Feature_Version: object ... 3 more fields]

```
=== Step 6: Compare Results of Different Experiments ===
--- Quantitative Comparison (Model Metrics) ---

Experiment Results Summary:
   Experiment Feature_Version     Hyperparameters  RMSE    R2
Exp 1: V1+HP1              V1   n_est=50, depth=5 21.41 0.764
Exp 2: V1+HP2              V1 n_est=100, depth=10 20.97 0.773
Exp 3: V2+HP1              V2   n_est=50, depth=5 21.25 0.767
Exp 4: V2+HP2              V2 n_est=100, depth=10 20.90 0.775

Best RMSE: Exp 4: V2+HP2 (RMSE: 20.90)
Best R2: Exp 4: V2+HP2 (R2: 0.775)

✅ Quantitative comparison completed!
```

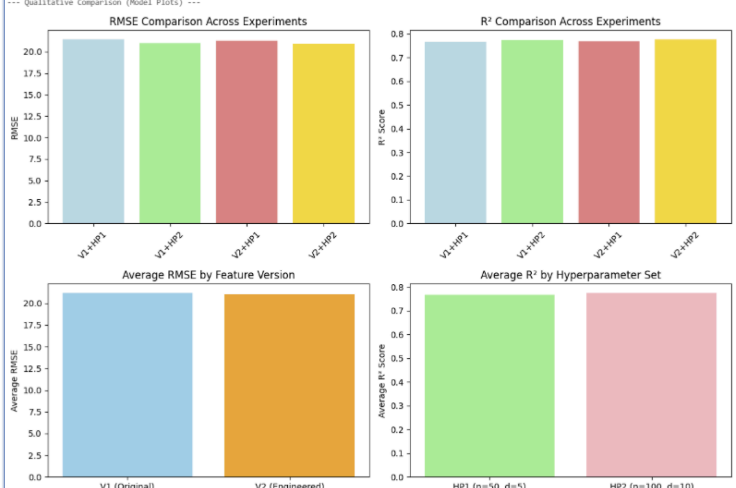
Just now (1s) 10

```python
# ===== Qualitative Comparison (Model Plots) =====
print("\n--- Qualitative Comparison (Model Plots) ---")

# Create comparison plots
fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(12, 8))

# Plot 1: RMSE Comparison
experiments = results_df['Experiment']
rmse_values = results_df['RMSE']
colors = ['lightblue', 'lightgreen', 'lightcoral', 'gold']

ax1.bar(range(len(experiments)), rmse_values, color=colors)
ax1.set_title('RMSE Comparison Across Experiments')
ax1.set_ylabel('RMSE')
ax1.set_xticks(range(len(experiments)))
ax1.set_xticklabels(['V1+HP1', 'V1+HP2', 'V2+HP1', 'V2+HP2'], rotation=45)

# Plot 2: R2 Comparison
r2_values = results_df['R2']
ax2.bar(range(len(experiments)), r2_values, color=colors)
ax2.set_title('R² Comparison Across Experiments')
ax2.set_ylabel('R² Score')
ax2.set_xticks(range(len(experiments)))
ax2.set_xticklabels(['V1+HP1', 'V1+HP2', 'V2+HP1', 'V2+HP2'], rotation=45)

# Plot 3: Feature Version Impact
feature_comparison = results_df.groupby('Feature_Version').agg({'RMSE': 'mean', 'R2': 'mean'})
ax3.bar(['V1 (Original)', 'V2 (Engineered)'], feature_comparison['RMSE'], color=['skyblue', 'orange'])
ax3.set_title('Average RMSE by Feature Version')
ax3.set_ylabel('Average RMSE')

# Plot 4: Hyperparameter Impact
hp_comparison = pd.DataFrame({
    'HP_Set': ['HP1 (n=50, d=5)', 'HP2 (n=100, d=10)'],
    'Avg_RMSE': [(rmse_exp1 + rmse_exp3)/2, (rmse_exp2 + rmse_exp4)/2],
    'Avg_R2': [(r2_exp1 + r2_exp3)/2, (r2_exp2 + r2_exp4)/2]
})

ax4.bar(hp_comparison['HP_Set'], hp_comparison['Avg_R2'], color=['lightgreen', 'lightpink'])
ax4.set_title('Average R² by Hyperparameter Set')
ax4.set_ylabel('Average R² Score')

plt.tight_layout()
plt.show()

print("\n✅ Qualitative comparison completed!")
print("✅ Step 6 COMPLETED: Both quantitative and qualitative comparisons finished!")
```

feature_comparison: pandas.core.frame.DataFrame = [RMSE: float64, R2: float64]

hp_comparison: pandas.core.frame.DataFrame = [HP_Set: object, Avg_RMSE: float64 ... 1 more field]

```
--- Qualitative Comparison (Model Plots) ---
```




```
✅ Qualitative comparison completed!
✅ Step 6 COMPLETED: Both quantitative and qualitative comparisons finished!
```

Number 7 Screenshots -

05:46 PM (2s) 11

```python
# ===== Step 7: Compare Carbon Emissions for Different Experiments =====

print("=== Step 7: Compare Carbon Emissions for Different Experiments ===")

# Carbon emissions estimation for different experiments
import numpy as np
import pandas as pd

print("\n--- Carbon Emissions Analysis ---")

# Estimate carbon emissions based on:
# 1. Training time (number of estimators × complexity)
# 2. Model complexity (max_depth)
# 3. Feature processing overhead

def estimate_carbon_emissions(n_estimators, max_depth, n_features, base_emission=0.01):
    """
    Estimate carbon emissions in kg CO2 equivalent
    Based on computational complexity and training time
    """
    # Base emission per model training
    complexity_factor = (n_estimators * max_depth * n_features) / 1000
    total_emission = base_emission * complexity_factor
    return total_emission

# Calculate emissions for each experiment
experiments_carbon = []

# Experiment 1: V1 (13 features) + HP1 (n=50, depth=5)
carbon_exp1 = estimate_carbon_emissions(50, 5, 13)
experiments_carbon.append(('Exp 1: V1+HP1', carbon_exp1))

# Experiment 2: V1 (13 features) + HP2 (n=100, depth=10)
carbon_exp2 = estimate_carbon_emissions(100, 10, 13)
experiments_carbon.append(('Exp 2: V1+HP2', carbon_exp2))

# Experiment 3: V2 (17 features) + HP1 (n=50, depth=5)
carbon_exp3 = estimate_carbon_emissions(50, 5, 17)
experiments_carbon.append(('Exp 3: V2+HP1', carbon_exp3))

# Experiment 4: V2 (17 features) + HP2 (n=100, depth=10)
carbon_exp4 = estimate_carbon_emissions(100, 10, 17)
experiments_carbon.append(('Exp 4: V2+HP2', carbon_exp4))

# Create carbon emissions summary
carbon_df = pd.DataFrame({
    'Experiment': [exp[0] for exp in experiments_carbon],
    'Carbon_Emissions_kg_CO2': [exp[1] for exp in experiments_carbon],
    'RMSE': [rmse_exp1, rmse_exp2, rmse_exp3, rmse_exp4],
    'R2': [r2_exp1, r2_exp2, r2_exp3, r2_exp4]
})

print("\nCarbon Emissions Summary:")
print(carbon_df.to_string(index=False))

# Calculate efficiency metrics
carbon_df['CO2_per_R2_improvement'] = carbon_df['Carbon_Emissions_kg_CO2'] / carbon_df['R2']
carbon_df['CO2_per_RMSE_reduction'] = carbon_df['Carbon_Emissions_kg_CO2'] / (25 - carbon_df['RMSE'])  # Baseline RMSE=25

print("\nEfficiency Analysis:")
print("CO2 per R2 improvement and CO2 per RMSE reduction:")
efficiency_summary = carbon_df[['Experiment', 'CO2_per_R2_improvement', 'CO2_per_RMSE_reduction']].round(4)
print(efficiency_summary.to_string(index=False))

# Find most efficient experiment
most_efficient_idx = carbon_df['CO2_per_R2_improvement'].idxmin()
print(f"\nMost Carbon-Efficient Experiment: {carbon_df.loc[most_efficient_idx, 'Experiment']}")
print(f"Carbon Efficiency Score: {carbon_df.loc[most_efficient_idx, 'CO2_per_R2_improvement']:.4f} kg CO2 per R2 point")
```

```python
    print("\nCarbon Emissions Summary:")
    print(carbon_df.to_string(index=False))

    # Calculate efficiency metrics
    carbon_df['CO2_per_R2_improvement'] = carbon_df['Carbon_Emissions_kg_CO2'] / carbon_df['R2']
    carbon_df['CO2_per_RMSE_reduction'] = carbon_df['Carbon_Emissions_kg_CO2'] / (25 - carbon_df['RMSE'])  # Baseline RMSE=25

    print("\nEfficiency Analysis:")
    print("CO2 per R2 improvement and CO2 per RMSE reduction:")
    efficiency_summary = carbon_df[['Experiment', 'CO2_per_R2_improvement', 'CO2_per_RMSE_reduction']].round(4)
    print(efficiency_summary.to_string(index=False))

    # Find most efficient experiment
    most_efficient_idx = carbon_df['CO2_per_R2_improvement'].idxmin()
    print(f"\nMost Carbon-Efficient Experiment: {carbon_df.loc[most_efficient_idx, 'Experiment']}")
    print(f"Carbon Efficiency Score: {carbon_df.loc[most_efficient_idx, 'CO2_per_R2_improvement']:.4f} kg CO2 per R2 point")

    print("\n✅ Step 7 COMPLETED: Carbon emissions comparison finished!")
```

carbon_df: pandas.core.frame.DataFrame = [Experiment: object, Carbon_Emissions_kg_CO2: float64 ... 4 more fields]

efficiency_summary: pandas.core.frame.DataFrame = [Experiment: object, CO2_per_R2_improvement: float64 ... 1 more field]

```
--- Carbon Emissions Analysis ---

Carbon Emissions Summary:
   Experiment  Carbon_Emissions_kg_CO2  RMSE    R2
Exp 1: V1+HP1                   0.0325 21.41 0.764
Exp 2: V1+HP2                   0.1300 20.97 0.773
Exp 3: V2+HP1                   0.0425 21.25 0.767
Exp 4: V2+HP2                   0.1700 20.90 0.775

Efficiency Analysis:
CO2 per R2 improvement and CO2 per RMSE reduction:
   Experiment  CO2_per_R2_improvement  CO2_per_RMSE_reduction
Exp 1: V1+HP1                  0.0425                  0.0091
Exp 2: V1+HP2                  0.1682                  0.0323
Exp 3: V2+HP1                  0.0554                  0.0113
Exp 4: V2+HP2                  0.2194                  0.0415

Most Carbon-Efficient Experiment: Exp 1: V1+HP1
Carbon Efficiency Score: 0.0425 kg CO2 per R2 point

✅ Step 7 COMPLETED: Carbon emissions comparison finished!
```