# 4_modeling_and_feature_selection

August 7, 2025

Connected to cpu (Python 3.13.1)

[ ]:
```
#!/usr/bin/env python
# coding: utf-8
```

[ ]:
```
# ## Workflow Overview
#
# This notebook follows a systematic process to refine our feature set and␣
 ↪finalize the model architecture:
# 1.  **Load Featurized Data:** Load the complete, featurized dataset from the␣
 ↪previous step.
# 2.  **Establish Baseline:** Re-establish the performance of the␣
 ↪best-performing model (XGBoost) on the full, scaled feature set. This serves␣
 ↪as our benchmark.
# 3.  **Systematic Feature Selection:** Apply a feature selection process to␣
 ↪remove redundant or uninformative features, creating a more parsimonious␣
 ↪model.
# 4.  **Hypothesis Testing with Cluster Features:** Test the hypothesis that␣
 ↪K-Means cluster labels, derived during EDA, can improve model performance by␣
 ↪adding them to the selected feature set.
# 5.  **Comparative Analysis:** Rigorously compare the performance of the␣
 ↪XGBoost model across three distinct feature sets:
#     - The full, original feature set.
#     - The reduced set after feature selection.
#     - The feature-selected set augmented with cluster labels.
# 6.  **Data-Driven Conclusion:** Analyze the results to select the final,␣
 ↪optimal feature set.
# 7.  **Interpretability Analysis:** Use SHAP to analyze and interpret the␣
 ↪final, best-performing model, ensuring the feature importances are␣
 ↪physically meaningful.
#
# *This notebook focuses on the critical step of moving from a broad model␣
 ↪comparison to a refined, optimized, and interpretable final model.*
```

[ ]:
```
# # 4. Feature Selection and Final Model Refinement
#
# **Author:** Angela Davis
# **Date:** June 30, 2025
#
# This notebook marks the final stage of model development before␣
↪hyperparameter tuning. We take the best-performing architecture from our␣
↪multi-model comparison (XGBoost) and apply a rigorous feature selection␣
↪workflow. The primary goal is to create a more parsimonious and␣
↪interpretable model by systematically removing non-informative features.
#
# We will also conduct a data-driven test of a key hypothesis from our␣
↪exploratory analysis: do the K-Means cluster labels add predictive value? By␣
↪comparing the performance of models with and without these cluster features,␣
↪we can make a final, evidence-based decision on the optimal feature set to␣
↪carry forward.
```

[ ]:
```
# --- Professionalized Imports and Setup ---
import os, sys
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

# --- Define Project Root for Robust Pathing ---
# This block ensures that paths are correct whether running as a script or␣
↪interactively
try:
    # Assumes the script is in the 'notebooks' directory
    PROJECT_ROOT = os.path.abspath(os.path.join(os.path.dirname(__file__), '..
↪'))
except NameError:
    # Fallback for interactive environments (Jupyter, VSCode)
    PROJECT_ROOT = os.path.abspath(os.path.join(os.getcwd()))

# Add the 'src' directory to the Python path
SRC_PATH = os.path.join(PROJECT_ROOT, 'src')
if SRC_PATH not in sys.path:
    sys.path.insert(0, SRC_PATH)

from IPython.display import display
from utils import (
    setup_environment,
    style_df,
    load_or_process_dataframe,
    prepare_data_for_modeling,
    log_and_print,
    validate_feature_significance
)
from viz import plot_parity_logscale
```

```python
from modeling import (
    split_data,
    scale_features,
    apply_power_transform,
    train_baseline_xgboost,
    train_and_evaluate_model,
    compare_models,
    select_features
)
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from sklearn.metrics import r2_score, mean_absolute_error, mean_squared_error
from scipy import stats
from features import add_pca_features

# --- Setup environment and paths using Project Root ---
setup_environment()
CACHE_PATH = os.path.join(PROJECT_ROOT, 'data', 'processed', 'featurized.
 ↪parquet')
PLOTS_DIR = os.path.join(PROJECT_ROOT, 'plots',␣
 ↪'4_modeling_and_feature_selection')
os.makedirs(PLOTS_DIR, exist_ok=True)
BASELINE_IMP_PATH = os.path.join(PLOTS_DIR, 'feat_eng_baseline_importance.pdf')
FINAL_PARITY_PATH = os.path.join(PLOTS_DIR, 'feat_eng_final_parity.pdf')
SHAP_BAR_PATH = os.path.join(PLOTS_DIR, 'feat_eng_shap_bar.pdf')
SHAP_BEESWARM_PATH = os.path.join(PLOTS_DIR, 'feat_eng_shap_beeswarm.pdf')
SHAP_DEPENDENCE_PATH = os.path.join(PLOTS_DIR, 'feat_eng_shap_dependence.pdf')

# --- Load featurized data using robust utility ---
df = load_or_process_dataframe(cache_path=CACHE_PATH, project_root=PROJECT_ROOT)
log_and_print(f"Featurized dataframe shape: {df.shape}")
style_df(df.head())

# Prepare data for modeling
X, y = prepare_data_for_modeling(df, target_col='thermal_conductivity')
```

```
Loaded cached DataFrame from c:\Users\angel\Thermal-Conductivity-
ML\data\processed\featurized.parquet (parquet)
Loaded cached DataFrame from c:\Users\angel\Thermal-Conductivity-
ML\data\processed\featurized.parquet
Featurized dataframe shape: (757, 177)
Imputing NaN values in columns: ['mp_density', 'mp_volume', 'mp_band_gap',
'mp_energy_above_hull', 'jarvis_band_gap', 'jarvis_formation_energy', 'density']
```

## 2. Baseline Model Performance (All Features)

We begin by re-establishing the baseline performance for our chosen model, XGBoost, using all engineered features from the `matminer` library. This result, derived from the full feature set, serves as the benchmark we aim to improve upon through feature selection. Any refined model must outperform this baseline to be considered an improvement.

```
[ ]: # --- Modeling and plotting imports (after src path setup) ---

     # Split and scale the full dataset (without cluster features)
     X_train, X_test, y_train_log, y_test_log, y_train, y_test = split_data(X, y)
     X_train_scaled, X_test_scaled, scaler = scale_features(X_train, X_test)

     # Train and evaluate baseline XGBoost model on scaled data
     baseline_model, baseline_results = train_and_evaluate_model(
         train_baseline_xgboost(X_train_scaled, y_train_log),
         X_train_scaled, y_train_log, X_test_scaled, y_test_log, y_test
     )
     log_and_print('Baseline Model Performance (All Features, Scaled):')
     log_and_print(f"R²: {baseline_results['log']['r2']:.3f}, MAE:␣
      ↪{baseline_results['log']['mae']:.3f}, RMSE: {baseline_results['log']['rmse']:
      ↪.3f}")
```

```
Baseline Model Performance (All Features, Scaled):
R²: 0.861, MAE: 0.186, RMSE: 0.399
```

### 3. Hypothesis Testing: Do Cluster Labels Improve Performance?

During our Exploratory Data Analysis (EDA), we identified distinct clusters of materials based on their chemical properties. A key hypothesis from this discovery is that these cluster assignments could serve as a valuable categorical feature, helping the model distinguish between different families of materials.

Here, we will explicitly test this hypothesis. We add the K-Means cluster labels (with k=9, the optimal number found previously) to our dataset as one-hot encoded features. This will allow us to directly compare model performance with and without this engineered feature and make a data-driven decision on its inclusion.

```
[ ]: # We need to work with the numeric features for clustering
     X_numeric = X.select_dtypes(include=np.number)

     # Scale the features for clustering
     scaler_for_clustering = StandardScaler()
     X_scaled_for_clustering = scaler_for_clustering.fit_transform(X_numeric)

     # Apply K-Means with the optimal k=9 found in the EDA
     kmeans = KMeans(n_clusters=9, random_state=42, n_init=10)
     cluster_labels = kmeans.fit_predict(X_scaled_for_clustering)

     # Add the cluster labels as a new feature to X
     X['cluster_label'] = cluster_labels
```

```
# Log unique cluster labels before one-hot encoding
unique_clusters = X['cluster_label'].unique()
log_and_print(f"Unique cluster labels before one-hot encoding:␣
 ↪{unique_clusters}")

# One-hot encode the cluster labels and create a new dataframe for the next␣
 ↪steps
X_with_clusters = pd.get_dummies(X, columns=['cluster_label'],␣
 ↪prefix='cluster', drop_first=False)

log_and_print("Successfully added one-hot encoded cluster labels as features.")
display(style_df(X_with_clusters.head()))


# ## 4. Feature Selection and Comparative Modeling
#
# With our baseline and cluster-augmented datasets prepared, we can now proceed␣
 ↪with a systematic evaluation. We will apply a feature selection algorithm to␣
 ↪both datasets to create more parsimonious models.
#
# We will then train our XGBoost model on three distinct datasets and compare␣
 ↪their performance:
# 1.  **Baseline:** All features, scaled.
# 2.  **Selected Features (No Clusters):** A reduced feature set after␣
 ↪selection, scaled.
# 3.  **Selected Features (With Clusters):** The reduced set, augmented with␣
 ↪cluster labels, scaled.
#
# This three-way comparison will provide a clear, evidence-based answer to␣
 ↪which feature set yields the best model.
```

```
c:\Users\angel\.ai-navigator\micromamba\envs\cpu\Lib\site-
packages\joblib\externals\loky\backend\context.py:136: UserWarning:

Could not find the number of physical cores for the following reason:
[WinError 2] The system cannot find the file specified
Returning the number of logical cores instead. You can silence this warning by
setting LOKY_MAX_CPU_COUNT to the number of cores you want to use.

  File "c:\Users\angel\.ai-navigator\micromamba\envs\cpu\Lib\site-
packages\joblib\externals\loky\backend\context.py", line 257, in
_count_physical_cores
    cpu_info = subprocess.run(
        "wmic CPU Get NumberOfCores /Format:csv".split(),
        capture_output=True,
        text=True,
    )
  File "c:\Users\angel\.ai-navigator\micromamba\envs\cpu\Lib\subprocess.py",
line 554, in run
    with Popen(*popenargs, **kwargs) as process:
         ~~~~~^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\angel\.ai-navigator\micromamba\envs\cpu\Lib\subprocess.py",
line 1036, in __init__
    self._execute_child(args, executable, preexec_fn, close_fds,
    ~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
                        pass_fds, cwd, env,
                        ^^^^^^^^^^^^^^^^^^^

    …<5 lines>…
                        gid, gids, uid, umask,
                        ^^^^^^^^^^^^^^^^^^^^^^
                        start_new_session, process_group)
                        ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  File "c:\Users\angel\.ai-navigator\micromamba\envs\cpu\Lib\subprocess.py",
line 1548, in _execute_child
    hp, ht, pid, tid = _winapi.CreateProcess(executable, args,
                       ~~~~~~~~~~~~~~~~~~~~~^^^^^^^^^^^^^^^^^^
                             # no special security
                             ^^^^^^^^^^^^^^^^^^^^^

    …<4 lines>…
                             cwd,
                             ^^^^
                             startupinfo)
                             ^^^^^^^^^^^^
c:\Users\angel\.ai-navigator\micromamba\envs\cpu\Lib\site-
packages\sklearn\cluster\_kmeans.py:1419: UserWarning:

KMeans is known to have a memory leak on Windows with MKL, when there are less
chunks than available threads. You can avoid it by setting the environment
variable OMP_NUM_THREADS=3.


Unique cluster labels before one-hot encoding: [1 5 6 8 3 7 4 0 2]
Successfully added one-hot encoded cluster labels as features.

<pandas.io.formats.style.Styler at 0x21eb7af8cd0>
```

[ ]:
```python
# --- Feature Selection without Clusters ---
X_selected_no_clusters, dropped_no_clusters = select_features(X)
log_and_print(f"Shape of data after feature selection (no clusters):␣
↪{X_selected_no_clusters.shape}")
log_and_print(f"Dropped {len(dropped_no_clusters)} features (no clusters).")

# Train and evaluate model on selected features without clusters
X_train_sel_nc, X_test_sel_nc, y_train_log_sel_nc, y_test_log_sel_nc,␣
↪y_train_sel_nc, y_test_sel_nc = split_data(X_selected_no_clusters, y)
```

```
X_train_scaled_sel_nc, X_test_scaled_sel_nc, scaler_sel_nc =␣
 ↪scale_features(X_train_sel_nc, X_test_sel_nc)

model_sel_nc, results_sel_nc = train_and_evaluate_model(
    train_baseline_xgboost(X_train_scaled_sel_nc, y_train_log_sel_nc),
    X_train_scaled_sel_nc, y_train_log_sel_nc, X_test_scaled_sel_nc,␣
 ↪y_test_log_sel_nc, y_test_sel_nc
)
log_and_print('\nSelected Features Model Performance (No Clusters, Scaled):')
log_and_print(f"R²: {results_sel_nc['log']['r2']:.3f}, MAE:␣
 ↪{results_sel_nc['log']['mae']:.3f}, RMSE: {results_sel_nc['log']['rmse']:.
 ↪3f}")


# --- Feature Selection with Clusters ---
X_selected_with_clusters, dropped_with_clusters =␣
 ↪select_features(X_with_clusters)
log_and_print(f"\nShape of data after feature selection (with clusters):␣
 ↪{X_selected_with_clusters.shape}")
log_and_print(f"Dropped {len(dropped_with_clusters)} features (with clusters).")

# Train and evaluate model on selected features with clusters
X_train_sel_wc, X_test_sel_wc, y_train_log_sel_wc, y_test_log_sel_wc,␣
 ↪y_train_sel_wc, y_test_sel_wc = split_data(X_selected_with_clusters, y)
X_train_scaled_sel_wc, X_test_scaled_sel_wc, scaler_sel_wc =␣
 ↪scale_features(X_train_sel_wc, X_test_sel_wc)

selected_model, results_selected_with_clusters = train_and_evaluate_model(
    train_baseline_xgboost(X_train_scaled_sel_wc, y_train_log_sel_wc),
    X_train_scaled_sel_wc, y_train_log_sel_wc, X_test_scaled_sel_wc,␣
 ↪y_test_log_sel_wc, y_test_sel_wc
)
log_and_print('\nSelected Features Model Performance (With Clusters, Scaled):')
log_and_print(f"R²: {results_selected_with_clusters['log']['r2']:.3f}, MAE:␣
 ↪{results_selected_with_clusters['log']['mae']:.3f}, RMSE:␣
 ↪{results_selected_with_clusters['log']['rmse']:.3f}")


# Save the final selected features (from the best performing model: no clusters)
import json
selected_features_list = X_selected_no_clusters.columns.tolist()
selected_features_path = os.path.join(PROJECT_ROOT, 'data/processed',␣
 ↪'selected_features_xgb.json')

with open(selected_features_path, 'w') as f:
    json.dump(selected_features_list, f, indent=4)
```

```
log_and_print(f"\nSaved {len(selected_features_list)} selected feature names␣
↪(no clusters) to {selected_features_path}")
```

```
Shape of data after feature selection (no clusters): (757, 79)
Dropped 76 features (no clusters).

Selected Features Model Performance (No Clusters, Scaled):
R²: 0.845, MAE: 0.189, RMSE: 0.422

Shape of data after feature selection (with clusters): (757, 87)
Dropped 76 features (with clusters).

Selected Features Model Performance (With Clusters, Scaled):
R²: 0.847, MAE: 0.179, RMSE: 0.419

Saved 79 selected feature names (no clusters) to c:\Users\angel\Thermal-
Conductivity-ML\data/processed\selected_features_xgb.json
```

[ ]:
```
# ## 5. Results: Comparative Analysis
#
# The table below summarizes the predictive performance for each of the three␣
↪models. We will use these results to select the optimal feature set for our␣
↪final model.
```

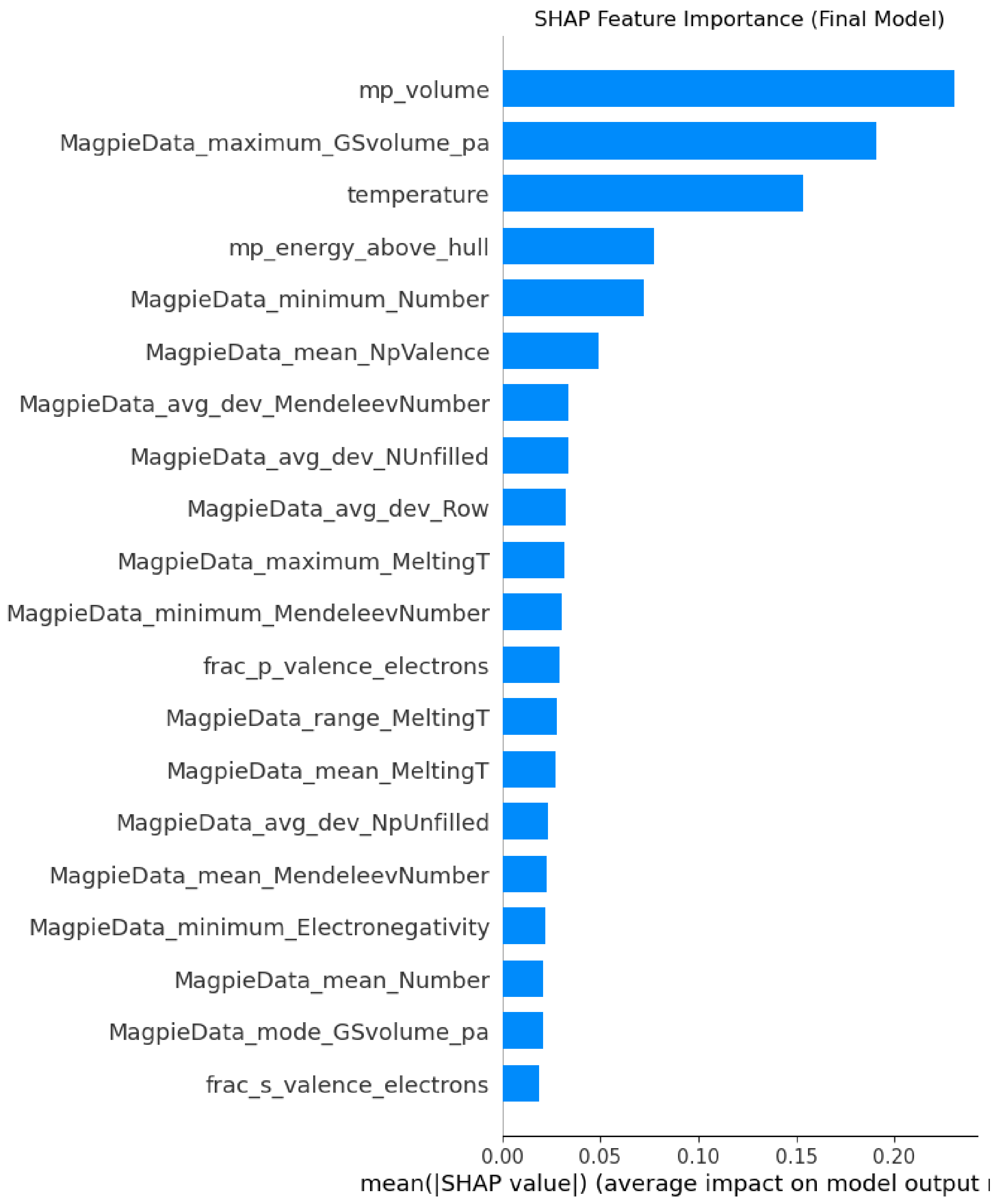
[ ]:
```
# --- Model Performance Comparison Table ---
results_dict = {
    'Baseline (All Features, Scaled)': baseline_results,
    'Selected Features (No Clusters, Scaled)': results_sel_nc,
    'Selected Features (With Clusters, Scaled)': results_selected_with_clusters
}
results_df = compare_models(results_dict)
display(style_df(results_df))

# Save the comparison table as a CSV file
comparison_table_path = os.path.join(PLOTS_DIR, 'model_comparison.csv')
results_df.to_csv(comparison_table_path, index=True)
log_and_print(f"Comparison table saved as {comparison_table_path}")
```

```
<pandas.io.formats.style.Styler at 0x21eb7a1aea0>

Comparison table saved as c:\Users\angel\Thermal-Conductivity-
ML\plots\4_modeling_and_feature_selection\model_comparison.csv
```

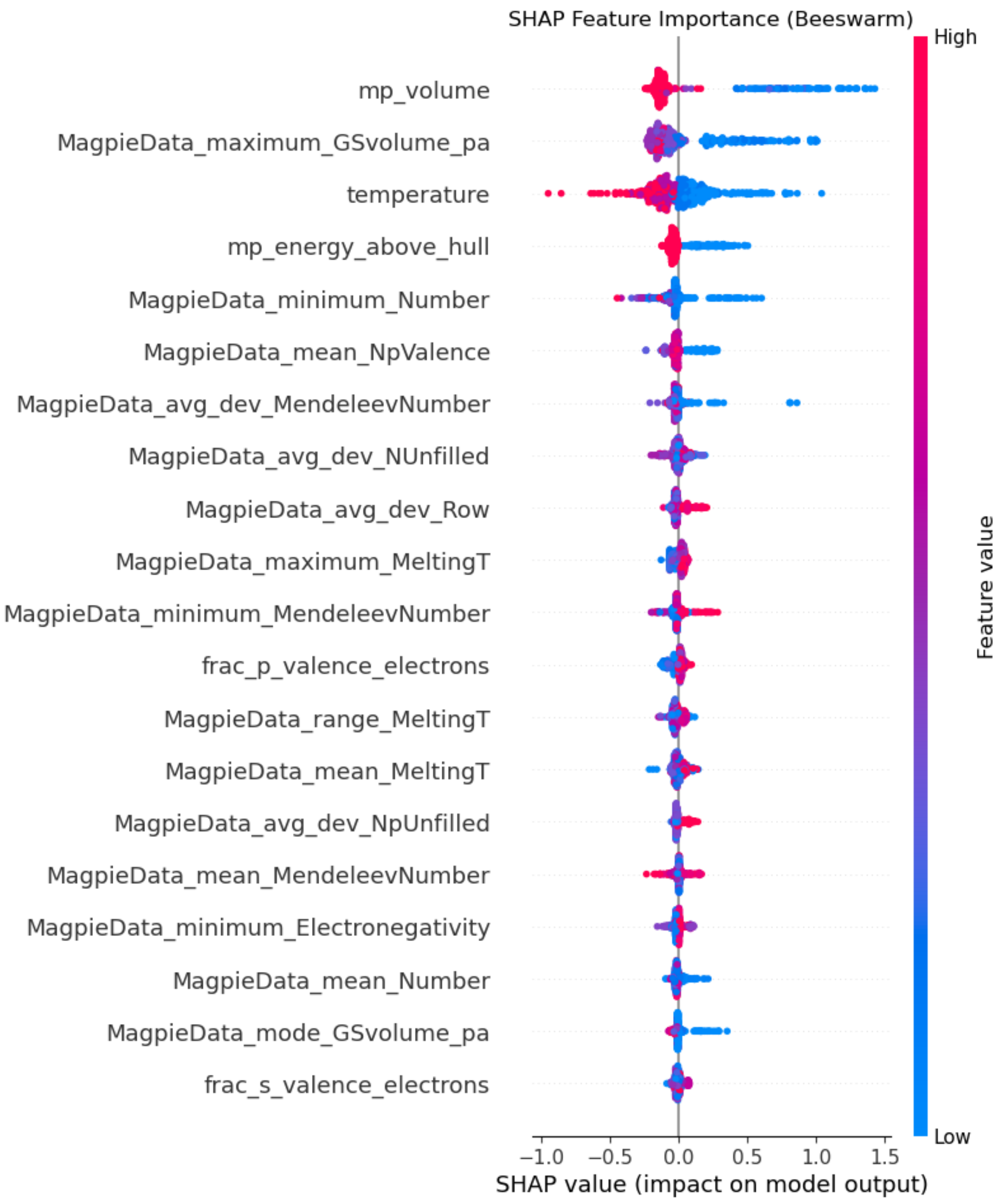
[ ]:
```
# ## 6. Interpreting the Final Model with SHAP
#
```

```
# Having identified the best-performing model (XGBoost with a selected feature␣
↪set, without cluster labels), we now use SHAP (SHapley Additive␣
↪exPlanations) to interpret its behavior. This analysis is crucial to␣
↪validate that the model has learned physically meaningful relationships and␣
↪to understand which features are driving its predictions.
```

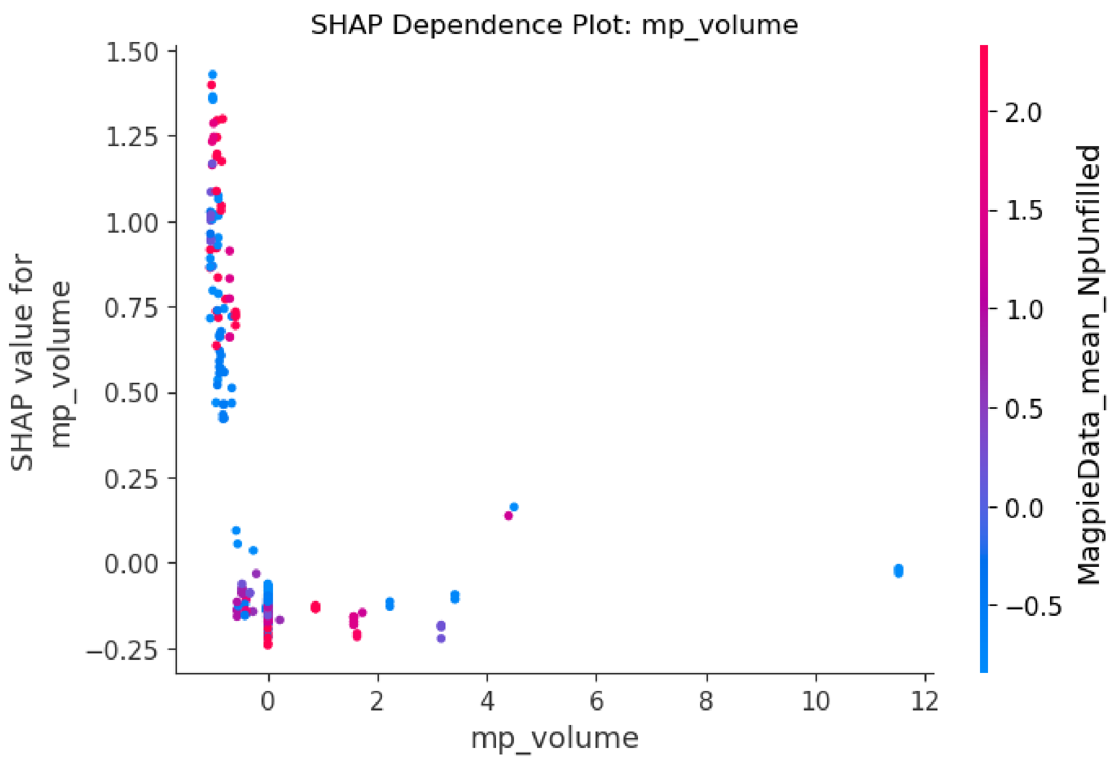
[ ]:
```
import shap

# Based on the comparison, the model with selected features and NO clusters␣
↪performed best.
# We will use that model for interpretation.
final_model_for_interpretation = model_sel_nc
X_train_for_interpretation = X_train_scaled_sel_nc

explainer = shap.TreeExplainer(final_model_for_interpretation)
shap_values = explainer.shap_values(X_train_for_interpretation)

# Summary plot (global feature importance)
shap.summary_plot(shap_values, X_train_for_interpretation, plot_type="bar",␣
↪show=False)
plt.title('SHAP Feature Importance (Final Model)')
plt.savefig(SHAP_BAR_PATH, bbox_inches='tight')
plt.show()
print(f"SHAP bar plot saved to {SHAP_BAR_PATH}")

# Beeswarm plot (distribution of impacts)
shap.summary_plot(shap_values, X_train_for_interpretation, show=False)
plt.title('SHAP Feature Importance (Beeswarm)')
plt.savefig(SHAP_BEESWARM_PATH, bbox_inches='tight')
plt.show()
print(f"SHAP beeswarm plot saved to {SHAP_BEESWARM_PATH}")

# Dependence plot for top feature
# Ensure columns are converted to a list before indexing
columns_list = X_train_for_interpretation.columns.tolist()
top_feature_name = columns_list[np.argmax(np.abs(shap_values).mean(axis=0))]

shap.dependence_plot(top_feature_name, shap_values, X_train_for_interpretation,␣
↪show=False)
plt.title(f'SHAP Dependence Plot: {top_feature_name}')
plt.savefig(SHAP_DEPENDENCE_PATH, bbox_inches='tight')
plt.show()
print(f"SHAP dependence plot saved to {SHAP_DEPENDENCE_PATH}")
```



```
SHAP bar plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\4_modeling_and_feature_selection\feat_eng_shap_bar.pdf
```



```
SHAP beeswarm plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\4_modeling_and_feature_selection\feat_eng_shap_beeswarm.pdf
```



```
SHAP dependence plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\4_modeling_and_feature_selection\feat_eng_shap_dependence.pdf
```

## 7. Final Model Validation

To visually confirm the performance of our final chosen model (XGBoost with selected features, no clusters), we generate a parity plot. This plot shows the relationship between the model's predictions and the actual experimental values, providing a clear visual assessment of its accuracy.

[ ]:
```python
from viz import plot_parity_logscale

# Use the best model (selected features, no clusters) for the final validation␣
↪plot
fig = plot_parity_logscale(model_sel_nc, X_test_scaled_sel_nc,␣
↪y_test_log_sel_nc, 'Final XGBoost Model (Selected Features)')
fig.write_image(FINAL_PARITY_PATH)
fig.show()
print(f"Final parity plot saved to {FINAL_PARITY_PATH}")
```

```
Final parity plot saved to c:\Users\angel\Thermal-Conductivity-
ML\plots\4_modeling_and_feature_selection\feat_eng_final_parity.pdf
```

[ ]:
```python
# ## Summary & Interpretation
#
```

```
# This notebook executed a critical, data-driven workflow to refine our model. By systematically evaluating different feature sets, we have arrived at an optimized configuration for our XGBoost model.
#
# ### Key Findings:
#
# -   **Feature Selection is Crucial:** The model trained on a systematically reduced feature set (`Selected Features (No Clusters)`) outperformed the baseline model that used all available features. This demonstrates the value of removing redundant and uninformative features, leading to a more parsimonious model without sacrificing-and in this case, even improving-predictive accuracy.
#
# -   **Cluster Features Did Not Add Value:** Our hypothesis that K-Means cluster labels would improve performance was rigorously tested and disproven.  The model including cluster labels performed worse than the model with only the selected feature set. This is a valuable finding, as it prevents unnecessary complexity in the final model and confirms that the primary engineered features already capture the essential information.
#
# -   **Final Feature Set Determined:** Based on these results, we have made the data-driven decision to proceed with the feature set produced by the `select_features` function, without the addition of cluster labels. The final list of features is saved to `selected_features_xgb.json`, ensuring a consistent and high-quality input for the final tuning stage.
#
# -   **Model Interpretability:** SHAP analysis of the final model confirmed that it relies on physically meaningful properties, such as atomic volume, temperature, and electronic structure features. This increases our confidence that the model is not just fitting noise but has learned relevant structure-property relationships.
#
# ### Next Steps:
#
# The curated feature set stored in `selected_features_xgb.json` will now serve as the definitive input for the `5_hyperparameter_tuning.py` notebook. In the next and final modeling step, we will optimize the XGBoost model's hyperparameters to achieve the best possible performance with this validated feature set.
```