```
"""# Task
Analyze the data in "/content/paper.xlsx" to predict PCE using a transformer
model combined with 3 or 4 classifiers, with 5-fold cross-validation. Provide
device-wise PCE predictions, feature importance, and explainable AI results.

## Load the data

### Subtask:
Load the data from the "/content/paper.xlsx" file into a pandas DataFrame.
"""

import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

# Advanced Visualizations
print("Generating advanced visualizations...")

# Pairplot
# Exclude non-numeric columns for pairplot if needed, but pairplot often
handles object types gracefully
# Depending on the data and desired output, you might want to select a subset
of columns here too.
sns.pairplot(df, diag_kind='kde')
plt.suptitle('Pairwise Relationships between Features and PCE', y=1.02)
plt.show()

# Correlation Heatmap
plt.figure(figsize=(10, 8))
# Select only numeric columns for correlation calculation
numeric_df = df.select_dtypes(include=np.number)
sns.heatmap(numeric_df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.title('Correlation Heatmap of Features and PCE')
plt.show()

print("Advanced visualizations generated.")

import pandas as pd

file_path = "/content/device.xlsx"
df = pd.read_excel(file_path)

display(df.head())

"""**Reasoning**:
Import pandas and load the Excel file into a DataFrame.


"""
```

```python
# Import necessary libraries
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.preprocessing import MinMaxScaler

# Set plot style
sns.set(style="whitegrid")
plt.rcParams['figure.figsize'] = (10, 6)

# Load the dataset
df = pd.read_excel('/content/device.xlsx')

# Step 1: Basic info & data preview
print("=== Dataset Info ===")
print(df.info())
print("\n=== First 5 rows ===")
print(df.head())

# Step 2: Unique values & distribution in 'Device' column
unique_devices = df['Device'].unique()
print("\nUnique Devices:", unique_devices)
print("\nDevice counts:")
print(df['Device'].value_counts())

# Step 3: Check missing values
print("\nMissing values per column:")
print(df.isnull().sum())

# Optional: Fill missing data if necessary or drop rows
# Example: df = df.dropna()  # or df.fillna(method='ffill')

# Step 4: Summary statistics by Device
print("\nSummary statistics by Device:")
print(df.groupby('Device').describe())

# Step 5: Normalize numeric columns to range [-1, 1]
numeric_cols = df.select_dtypes(include=np.number).columns.tolist()

scaler = MinMaxScaler(feature_range=(-1, 1))
df_normalized = df.copy()
df_normalized[numeric_cols] = scaler.fit_transform(df[numeric_cols])

print("\n=== Normalized Data Preview (range [-1, 1]) ===")
print(df_normalized.head())
```

```
# Step 6: Visualizations on normalized data

## 6.1 PCE distribution by Device (boxplot)
plt.figure()
sns.boxplot(x='Device', y='PCE', data=df_normalized)
plt.title('Normalized PCE Distribution by Device')
plt.show()

## 6.2 Bandgap distribution by Device (histogram + KDE)
plt.figure()
sns.histplot(data=df_normalized, x='Bandgap', hue='Device', multiple='stack',
kde=True)
plt.title('Normalized Bandgap Distribution by Device')
plt.show()

## 6.3 Electron Affinity distribution by Device
plt.figure()
sns.histplot(data=df_normalized, x='Electron Affinity', hue='Device',
multiple='stack', kde=True)
plt.title('Normalized Electron Affinity Distribution by Device')
plt.show()

## 6.4 Scatterplot: Electron Affinity vs Bandgap by Device
plt.figure()
sns.scatterplot(data=df_normalized, x='Bandgap', y='Electron Affinity',
hue='Device', style='Device', s=100)
plt.title('Normalized Electron Affinity vs Bandgap by Device')
plt.show()

## 6.5 Scatterplot: Electron Mobility vs Hole Mobility by Device
plt.figure()
sns.scatterplot(data=df_normalized, x='Electron Mobility', y='Hole Mobility',
hue='Device', style='Device', s=100)
plt.title('Normalized Electron Mobility vs Hole Mobility by Device')
plt.show()

# Step 7: Correlation Heatmaps per Device (using normalized data)
for device in unique_devices:
    subset = df_normalized[df_normalized['Device'] ==
device].reset_index(drop=True)
    numeric_subset = subset.select_dtypes(include=np.number)
    plt.figure(figsize=(10, 8))
    sns.heatmap(numeric_subset.corr(), annot=True, cmap='coolwarm', fmt=".2f")
    plt.title(f'Correlation Matrix for Device: {device} (Normalized)')
    plt.show()

# Step 8: Pairplots (scatter matrix) per Device for key features using
normalized data
```

```
features = ['Bandgap', 'Electron Affinity', 'Doping', 'Thickness', 'Electron
Mobility', 'Hole Mobility', 'PCE']
sns.pairplot(df_normalized, hue='Device', vars=features, height=2.5)
plt.suptitle('Pairplot of Key Features by Device (Normalized)', y=1.02)
plt.show()

# Step 9: Outlier detection - Boxplots of numeric columns per Device
(normalized data)
for col in numeric_cols:
    plt.figure()
    sns.boxplot(x='Device', y=col, data=df_normalized)
    plt.title(f'Boxplot of Normalized {col} by Device')
    plt.show()

# Step 10: Summary tables export (optional)
grouped_stats = df_normalized.groupby('Device').agg(['mean', 'std']).round(3)
grouped_stats.to_csv('device_group_summary_normalized.csv')
print("\nNormalized summary statistics saved to
device_group_summary_normalized.csv")

#
# PCE Prediction using Machine Learning Algorithms
#
# Mutliple evaluations and Visual Understanding



import pandas as pd
import numpy as np
import os
import matplotlib.pyplot as plt
import seaborn as sns
import shap

from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import KFold
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import learning_curve
import lightgbm as lgb

from yellowbrick.regressor import PredictionError, ResidualsPlot

# ----------------------------
# CONFIGURATION
# ----------------------------
file_path = '/content/device.xlsx'  # Change this to your actual file path
shap_sample_size = 100              # SHAP sample size per device
```

```
output_dir = "device"               # Main output directory
os.makedirs(output_dir, exist_ok=True)

# ----------------------------
# MODEL DEFINITIONS
# ----------------------------
models = {
    'RandomForest': RandomForestRegressor(random_state=42),
    'GradientBoosting': GradientBoostingRegressor(random_state=42),
    'DecisionTree': DecisionTreeRegressor(random_state=42),
    'LightGBM': lgb.LGBMRegressor(random_state=42)
}

# ----------------------------
# CV METRICS FUNCTION
# ----------------------------
def run_cv_metrics(model, X, y, cv):
    metrics = []
    for train_idx, test_idx in cv.split(X):
        X_train, X_test = X.iloc[train_idx], X.iloc[test_idx]
        y_train, y_test = y.iloc[train_idx], y.iloc[test_idx]

        model.fit(X_train, y_train)
        preds = model.predict(X_test)
        metrics.append({
            'R2': r2_score(y_test, preds),
            'RMSE': np.sqrt(mean_squared_error(y_test, preds)),
            'MAE': mean_absolute_error(y_test, preds)
        })
    return pd.DataFrame(metrics)

# ----------------------------
# MANUAL LEARNING CURVE FUNCTION
# ----------------------------
def plot_learning_curve(estimator, X, y, device_name, device_output_dir):
    train_sizes, train_scores, test_scores = learning_curve(
        estimator, X, y, cv=5, scoring='neg_mean_squared_error', n_jobs=-1,
        train_sizes=np.linspace(0.1, 1.0, 10), random_state=42)

    train_rmse = np.sqrt(-train_scores)
    test_rmse = np.sqrt(-test_scores)

    plt.figure(figsize=(8, 6))
    plt.plot(train_sizes, train_rmse.mean(axis=1), 'o-', color='r',
label='Training RMSE')
    plt.plot(train_sizes, test_rmse.mean(axis=1), 'o-', color='g', label='CV
RMSE')
```

```
    plt.title(f"Learning Curve - {type(estimator).__name__}\nDevice:
{device_name}")
    plt.xlabel("Training Examples")
    plt.ylabel("RMSE")
    plt.legend(loc='best')
    plt.grid(True)
    plt.tight_layout()
    plt.savefig(f"{device_output_dir}/learning_curve_{device_name}.png",
dpi=300)
    plt.close()

# ----------------------------
# LOAD DATA
# ----------------------------
df = pd.read_excel(file_path)

device_list = df['Device'].unique()
features = ['Bandgap', 'Electron Affinity', 'Doping', 'Thickness', 'Defect',
'Electron Mobility', 'Hole Mobility']
target = 'PCE'

# Initialize lists to collect data for global visualizations
all_metrics = []
all_importances = []
actual_vs_predicted_all = []

# Function to format device names for proper subscripts and superscripts
def format_device_name(device_name):
    return (device_name
            .replace("TlPbI3", "$TlPbI_3$")
            .replace("CdS", "-CdS")
            .replace("In3S2", "-In$_2$S$_3$")
            .replace("SnS2", "-SnS$_2$"))

# ----------------------------
# DEVICE-WISE ANALYSIS
# ----------------------------
for device in device_list:
    print(f"\n===== PROCESSING DEVICE: {device} =====")

    # Format the device name for LaTeX rendering
    formatted_device_name = format_device_name(device)

    df_device = df[df['Device'] == device].reset_index(drop=True)
    if len(df_device) < 5:
        print(f"Skipping {device} due to insufficient rows (<5).")
        continue
```

```
    device_name = device.replace(" ", "_")
    device_output_dir = os.path.join(output_dir, device_name)
    os.makedirs(device_output_dir, exist_ok=True)

    X_raw = df_device[features]
    y = df_device[target]

    cv = KFold(n_splits=5, shuffle=True, random_state=42)

    # --- CV METRICS WITHOUT NORMALIZATION ---
    print("\n--- Without Normalization ---")
    results_without_norm = {}
    for name, model in models.items():
        print(f"Running CV for {name}...")
        df_metrics = run_cv_metrics(model, X_raw, y, cv)
        results_without_norm[name] = df_metrics
        print(f"{name} Mean Metrics:\n{df_metrics.mean()}\n")

    # --- CV METRICS WITH NORMALIZATION ---
    print("\n--- With Normalization ---")
    scaler = StandardScaler()
    X_scaled = pd.DataFrame(scaler.fit_transform(X_raw), columns=features)
    results_with_norm = {}
    for name, model in models.items():
        print(f"Running CV for {name} (normalized)...")
        df_metrics = run_cv_metrics(model, X_scaled, y, cv)
        results_with_norm[name] = df_metrics
        print(f"{name} Mean Metrics (Normalized):\n{df_metrics.mean()}\n")

    # --- TRAIN MODELS ON FULL DATA ---
    trained_models = {}
    for name, model in models.items():
        model.fit(X_raw, y)
        trained_models[name] = model

    # --- FEATURE IMPORTANCE PLOT ---
    importance_df = pd.DataFrame({'Feature': features})
    for name, model in models.items():
        if hasattr(model, 'feature_importances_'):
            imp = model.feature_importances_
            if imp.sum() > 0:
                imp = imp / imp.sum()
            importance_df[name] = imp
        else:
            importance_df[name] = np.nan

    importance_df_melt = importance_df.melt(id_vars='Feature',
var_name='Model', value_name='Importance')
```

```
    plt.figure(figsize=(12, 7))
    sns.barplot(data=importance_df_melt.dropna(), x='Feature', y='Importance',
hue='Model')
    plt.title(f"Feature Importance Comparison\nDevice:
{formatted_device_name}")
    plt.xticks(rotation=45)
    plt.tight_layout()
    plt.savefig(f"{device_output_dir}/feature_importance_comparison_{device_na
me}.png", dpi=300)
    plt.close()




    # --- ACTUAL VS PREDICTED PLOTS ---
    plt.figure(figsize=(16, 12))
    for i, (name, model) in enumerate(models.items(), 1):
        preds = model.predict(X_raw)
        plt.subplot(2, 2, i)
        plt.scatter(y, preds, alpha=0.6)
        plt.plot([y.min(), y.max()], [y.min(), y.max()], 'k--')
        plt.title(f"{name}: Actual vs Predicted\nDevice:
{formatted_device_name}")
        plt.xlabel("Actual PCE")
        plt.ylabel("Predicted PCE")
    plt.tight_layout()
    plt.savefig(f"{device_output_dir}/actual_vs_predicted_all_models_{device_n
ame}.png", dpi=300)
    plt.close()

    # --- RESIDUALS PLOTS ---
    plt.figure(figsize=(16, 12))
    for i, (name, model) in enumerate(models.items(), 1):
        preds = model.predict(X_raw)
        residuals = y - preds
        plt.subplot(2, 2, i)
        sns.scatterplot(x=preds, y=residuals)
        plt.axhline(0, color='red', linestyle='--')
        plt.title(f"{name}: Residuals vs Predicted - Device:
{formatted_device_name}")
        plt.xlabel("Predicted PCE")
        plt.ylabel("Residuals (Actual - Predicted)")
    plt.tight_layout()
    plt.savefig(f"{device_output_dir}/residuals_plots_{device_name}.png",
dpi=300)
    plt.close()

    # --- RESIDUALS HISTOGRAM ---
    plt.figure(figsize=(16, 12))
```

```
    for i, (name, model) in enumerate(models.items(), 1):
        preds = model.predict(X_raw)
        residuals = y - preds
        plt.subplot(2, 2, i)
        sns.histplot(residuals, kde=True)
        plt.title(f"{name}: Residuals Histogram - Device:
{formatted_device_name}")
        plt.xlabel("Residual")
    plt.tight_layout()
    plt.savefig(f"{device_output_dir}/residuals_histograms_{device_name}.png",
dpi=300)
    plt.close()

    # --- Q-Q Plots ---
    import scipy.stats as stats
    plt.figure(figsize=(16, 12))
    for i, (name, model) in enumerate(models.items(), 1):
        preds = model.predict(X_raw)
        residuals = y - preds
        plt.subplot(2, 2, i)
        stats.probplot(residuals, dist="norm", plot=plt)
        plt.title(f"{name}: Q-Q Plot of Residuals - Device:
{formatted_device_name}")
    plt.tight_layout()
    plt.savefig(f"{device_output_dir}/qq_plots_{device_name}.png", dpi=300)
    plt.close()

    # --- YELLOWBRICK PREDICTION ERROR PLOT ---
    from sklearn.model_selection import train_test_split
    X_train, X_test, y_train, y_test = train_test_split(X_raw, y,
test_size=0.2, random_state=42)
    for name, model in models.items():
        plt.figure(figsize=(8, 6))
        visualizer = PredictionError(model)
        visualizer.fit(X_train, y_train)
        visualizer.score(X_test, y_test)
        visualizer.poof(outpath=f"{device_output_dir}/yellowbrick_prediction_e
rror_{name.lower()}_{device_name}.png")
        plt.close()

    # --- LEARNING CURVE ---
    for name, model in models.items():
        plot_learning_curve(model, X_raw, y, formatted_device_name,
device_output_dir)

    # --- SHAP EXPLAINABILITY ---
    print("\n--- SHAP EXPLAINABILITY ---")
```

```
    X_shap = X_raw.sample(n=min(shap_sample_size, len(X_raw)),
random_state=42)
    for name, model in models.items():
        print(f"Computing SHAP for {name}...")
        explainer = shap.Explainer(model, X_shap)
        shap_values = explainer(X_shap)

        # SHAP summary plot
        shap.summary_plot(shap_values, X_shap, show=False)
        plt.suptitle(f"SHAP Summary Plot - {name}\nDevice:
{formatted_device_name}", fontsize=14)
        plt.tight_layout()
        plt.savefig(f"{device_output_dir}/shap_summary_{name.lower()}_{device_
name}.png", dpi=300)
        plt.close()

        # SHAP waterfall plot (first sample)
        shap.plots.waterfall(shap_values[0], show=False)
        plt.suptitle(f"SHAP Waterfall Plot - {name}\nDevice:
{formatted_device_name}", fontsize=14)
        plt.tight_layout()
        plt.savefig(f"{device_output_dir}/shap_waterfall_{name.lower()}_{devic
e_name}.png", dpi=300)
        plt.close()

    # Model Metrics - Collect for Global Comparison
    cv_metrics = {}
    for name, model in models.items():
        metrics_df = run_cv_metrics(model, X_scaled, y, cv)
        cv_metrics[name] = metrics_df.mean().to_dict()
        # Append device-wise metrics to the global list
        all_metrics.append({
            'Device': device,
            'Model': name,
            'R2': cv_metrics[name]['R2'],
            'RMSE': cv_metrics[name]['RMSE'],
            'MAE': cv_metrics[name]['MAE']
        })

    # Collect importance and actual vs predicted for Global Comparison
    for name, model in trained_models.items():
        preds = model.predict(X_raw)
        for actual, pred in zip(y, preds):
            actual_vs_predicted_all.append({
                'Device': device,
                'Model': name,
                'Actual': actual,
                'Predicted': pred
```

```
            })
        if hasattr(model, 'feature_importances_'):
            imp = model.feature_importances_
            if imp.sum() > 0:
                imp = imp / imp.sum()
            for f, val in zip(features, imp):
                all_importances.append({
                    'Device': device,
                    'Model': name,
                    'Feature': f,
                    'Importance': val
                })


    print(f"Completed processing for {device}. Results saved in
{device_output_dir}.")

print(f"\n☑ All device-wise evaluation visuals saved in '{output_dir}/'
folder.")

# --------------------------- GLOBAL VISUALIZATIONS & CSV OUTPUT ------------
----------------

# Assuming `all_metrics`, `all_importances`, `actual_vs_predicted_all` are
generated in the device processing loop
metrics_df = pd.DataFrame(all_metrics)
importance_df = pd.DataFrame(all_importances)
preds_df = pd.DataFrame(actual_vs_predicted_all)
preds_df['Residual'] = preds_df['Actual'] - preds_df['Predicted']

# Save Performance Metrics (R2, RMSE, MAE) to CSV
metrics_df.to_csv(f"{output_dir}/global_performance_metrics.csv", index=False)
print(f"☑ Global performance metrics saved as
'global_performance_metrics.csv'.")

# Save Feature Importance to CSV
importance_df.to_csv(f"{output_dir}/global_feature_importance.csv",
index=False)
print(f"☑ Feature importance data saved as
'global_feature_importance.csv'.")

# Save Actual vs Predicted results (including residuals) to CSV
preds_df.to_csv(f"{output_dir}/global_actual_vs_predicted.csv", index=False)
print(f"☑ Actual vs Predicted results saved as
'global_actual_vs_predicted.csv'.")

# --- 1. Performance Metrics (Global)
for metric in ['R2', 'RMSE', 'MAE']:
    plt.figure(figsize=(14, 6))
```

```
    sns.barplot(data=metrics_df, x='Device', y=metric, hue='Model')
    # Format x-axis labels with subscripts
    plt.title(f"{metric} Across Devices and Models")
    plt.xticks(ticks=range(len(metrics_df['Device'].unique())),
               labels=[format_device_name(device) for device in
metrics_df['Device'].unique()],
               rotation=45)
    plt.tight_layout()
    plt.savefig(f"{output_dir}/metric_comparison_{metric.lower()}.png",
dpi=300)
    plt.close()

# --- 2. Residuals Plot (Global)
g = sns.FacetGrid(preds_df, col="Model", row="Device", height=3.5,
margin_titles=True)
g.map_dataframe(sns.scatterplot, x="Predicted", y="Residual", alpha=0.6)
g.map(plt.axhline, y=0, color="red", linestyle="--")
g.set_titles(col_template="{col_name}", row_template="{row_name}")
g.fig.subplots_adjust(top=0.9)
g.fig.suptitle("Residuals Across Devices and Models", fontsize=14)

# Update row and column labels with the formatted device names
for ax in g.axes.flatten():
    ax.set_xlabel('Predicted PCE')
    ax.set_ylabel('Residuals (Actual - Predicted)')
    # The tick labels were already set by FacetGrid, updating them manually is
tricky.
    # Let's skip manual tick label setting for now to avoid potential issues.
    # If needed, this could be done more robustly by accessing the underlying
axes objects.
    pass # Skipping manual tick label setting


g.savefig(f"{output_dir}/comparative_residuals_all_devices_models.png",
dpi=300)
plt.close()

# --- 3. Feature Distributions Across Devices (Global)
for feature in features:
    plt.figure(figsize=(10, 5))
    sns.boxplot(data=df, x='Device', y=feature)
    plt.title(f"{feature} Distribution Across Devices")
    # Format x-axis labels with subscripts
    plt.xticks(ticks=range(len(df['Device'].unique())),
               labels=[format_device_name(device) for device in
df['Device'].unique()],
               rotation=45)
    plt.tight_layout()
    fname = feature.lower().replace(" ", "_")
```

```
    plt.savefig(f"{output_dir}/feature_distribution_comparison_{fname}.png",
dpi=300)
    plt.close()

# --- 4. Correlation Heatmaps Across Devices (Global)
cols = 3
rows = int(np.ceil(len(device_list) / cols))
plt.figure(figsize=(6 * cols, 5 * rows))
for i, device in enumerate(device_list, 1):
    df_device = df[df['Device'] == device]
    corr = df_device[features + [target]].corr()
    plt.subplot(rows, cols, i)
    sns.heatmap(corr, annot=False, cmap='coolwarm', center=0, cbar=False)
    # Update title with formatted device names
    plt.title(f"Correlation - {format_device_name(device)}")
plt.tight_layout()
plt.savefig(f"{output_dir}/correlation_heatmaps_all_devices.png", dpi=300)
plt.close()

# --- 5. Feature Importance Heatmap (Global)
pivot_imp = importance_df.pivot_table(index='Model', columns=['Device',
'Feature'], values='Importance')
plt.figure(figsize=(18, 8))
sns.heatmap(pivot_imp, cmap="YlGnBu", linewidths=0.5)
plt.title("Feature Importance Heatmap: Models × (Devices × Features)")
plt.xlabel("Device / Feature")
plt.ylabel("Model")
plt.tight_layout()
plt.savefig(f"{output_dir}/comparative_feature_importance_heatmap_all_devices.
png", dpi=300)
plt.close()


# --- FEATURE IMPORTANCE HEATMAP (GLOBAL) ---
# Pivot the DataFrame to create a model vs device-feature matrix
pivot_imp = importance_df.pivot_table(
    index='Model',
    columns=['Device', 'Feature'],
    values='Importance'
)

# Plotting the heatmap
plt.figure(figsize=(18, 8))
sns.heatmap(pivot_imp, cmap="YlGnBu", annot=True, fmt='.2f', linewidths=0.5)
plt.title("Feature Importance Heatmap: Models × (Devices × Features)",
fontsize=16)
plt.xlabel("Device / Feature", fontsize=12)
plt.ylabel("Model", fontsize=12)
plt.tight_layout()
```

```
# Save the heatmap to the output directory
plt.savefig(f"{output_dir}/comparative_feature_importance_heatmap_all_devices.
png", dpi=300)
plt.close()


print(f"\n☑ All global evaluation visuals saved in '{output_dir}/' folder.")

import shutil
from google.colab import files

# Create a zip archive of the folder
shutil.make_archive('/content/device_archive', 'zip', '/content/device')

# Download the zipped archive
files.download('/content/device_archive.zip')

''''
```