Detailed Code Explanation: Energy Project Cost Estimator

Step 1: Dataset Generation

What We Did

In this initial step, we created a synthetic dataset of 200 energy projects with realistic parameters to train our AI-powered cost estimation model.

```
python
# Install necessary libraries (if not already installed)
!pip install faker

# Import libraries
import pandas as pd
import numpy as np
from faker import Faker

# Initialize Faker for generating fake project IDs
fake = Faker()
```

The first section sets up our environment by installing the Faker library (which helps generate realistic-looking data) and importing essential packages for data manipulation (pandas), numerical operations (numpy), and synthetic data generation (Faker).

```
python
# Function to generate synthetic energy dataset
def generate_energy_data(num_samples=200):
    data = []
    for i in range(1, num_samples + 1):  # Loop through 1 to 200
for Project IDs
        # Randomly select energy type
```

```
        energy_type = np.random.choice(["Solar", "Wind",
"Natural Gas", "Hydro", "Nuclear"],
                                       p=[0.3, 0.3, 0.2, 0.1,
0.1])
```

We defined a function to generate our dataset with a default of 200 projects. We used weighted probabilities to select energy types, with Solar and Wind being more common (30% each) than Hydro or Nuclear (10% each).

Key Features of the Dataset Generation

1. Realistic Parameter Generation:

   python

```
# Generate random values for features
project_size = np.random.uniform(50, 1000)  # MW capacity
location = np.random.choice(["Coastal", "Rural", "Urban"])
year = np.random.randint(2023, 2027)  # Project start year
```

   We created a diverse range of project sizes (50-1000 MW) and distributed them across different locations.

2. Intelligent Cost Calculations:

   python

```
# Material cost: Higher for nuclear, lower for solar/wind
material_cost = project_size * np.random.choice([6000, 8000,
10000, 15000, 20000],
                                                p=[0.4, 0.3, 0.2,
0.05, 0.05])

# Labor cost: Higher for urban projects
labor_hours = project_size * np.random.uniform(10, 50)
labor_cost = labor_hours * (30 if location == "Urban" else 25)
```

   We incorporated real-world cost relationships:

- Material costs scale with project size
- Urban labor is more expensive than rural/coastal labor
- Equipment costs vary based on energy type
- Regulatory costs are higher for fossil fuels (Natural Gas) and Nuclear

3. Environmental Impact:

   python

```
# Environmental impact: CO2 emissions (tons/year)
environmental_impact = 0 if energy_type in ["Solar", "Wind",
"Hydro"] else np.random.uniform(1000, 15000)
```

   We included sustainability metrics, assigning zero emissions to renewable energy sources and higher emissions to fossil fuels.

4. Inflation Modeling:

   python

```
# Inflation rate: Random between 2% and 6%
inflation_rate = np.random.uniform(2.0, 6.0)
inflated_cost = (material_cost + labor_cost + equipment_cost) *
(1 + inflation_rate / 100)
```

   We accounted for economic factors by applying different inflation rates to project costs.

Why This Approach

1. Data Realism: By incorporating real-world relationships (like higher costs for nuclear or urban projects), we created a dataset that mimics actual energy project economics.
2. Comprehensive Feature Set: Including 13 different features allows our model to learn complex relationships between project parameters and costs.
3. Controlled Variability: Using different probability distributions for each feature helps create diversity while maintaining realistic constraints.
4. Reproducibility: Setting a fixed number of samples (200) ensures we have enough data for modeling without overwhelming computing resources.

The result is a well-structured CSV file containing 200 energy projects with IDs ranging from Energy_001 to Energy_200, each with realistic features that will enable our cost estimation model to learn meaningful patterns.

Step 2: Displaying the Dataset in Google Colab

In this step, we display the generated dataset interactively in Google Colab to verify its structure and contents.

Code

```
python
from google.colab import sheets
sheet = sheets.InteractiveSheet(df=df)
```

What We Did

1. Google Colab Integration:
    - The `sheets` module from Google Colab allows us to create an interactive spreadsheet-like view of the dataset directly in the notebook.
    - This feature is especially useful for visualizing and validating the generated dataset.
2. InteractiveSheet:
    - The `InteractiveSheet` function creates a dynamic interface where users can scroll through rows, inspect columns, and verify data consistency.
3. Dataset Verification:
    - By passing the `df` (dataframe) to `InteractiveSheet`, users can ensure that all columns (e.g., `Project_ID`, `Energy_Type`, `Material_Cost`, etc.) are correctly populated with realistic values.

Why This Step is Important

1. Data Validation:
    - Before proceeding with model training, it's essential to confirm that the dataset is correctly structured and contains no missing or erroneous values.

    - This step ensures that all features (e.g., Material Cost, Equipment Cost) are realistic and within expected ranges.
2. Ease of Use:
    - The interactive spreadsheet view offers a user-friendly way to inspect data without needing to print large tables or scroll through raw outputs.
3. Debugging:
    - If any issues are found in the dataset (e.g., unexpected null values or outliers), they can be addressed before moving forward.

Output

- The dataset is displayed in an interactive spreadsheet format within Google Colab.
- Users can scroll through rows and columns to verify data accuracy.

Step 3: Saving and Downloading the Dataset

In this step, we save the generated dataset as a CSV file and download it for further use in model training and analysis.

Code

```
python
# Save the dataset to a CSV file in Google Colab's environment
csv_file_name = "energy_projects_dataset.csv"
df.to_csv(csv_file_name, index=False)

print(f"Dataset with {num_samples} samples created and saved as
'{csv_file_name}'.")
```

What We Did

1. Exporting the Dataset:
    - The dataset created in Step 1 is saved as a CSV file named `energy_projects_dataset.csv`using the `to_csv()` method from pandas.
    - The `index=False` argument ensures that row indices are not included in the output file.

2. Confirmation Message:
    - A print statement confirms that the dataset has been successfully saved.

Downloading the Dataset

```
python
from google.colab import files
files.download("energy_projects_dataset.csv")
```

3. File Download:
    - The `files.download()` function from Google Colab allows users to download the saved CSV file directly to their local machine.
    - This step ensures that the dataset is accessible for further processing or sharing.

Why This Step is Important

1. Data Accessibility:
    - Saving the dataset as a CSV file ensures it can be reused across different environments (e.g., local machine, cloud platforms).
    - The CSV format is widely compatible with various tools like Excel, Python, R, etc.
2. Verification:
    - Users can open the downloaded file to verify its contents and confirm that all features are correctly populated.
3. Reusability:
    - The saved dataset serves as the foundation for subsequent steps like preprocessing, model training, and deployment.

Output

- A CSV file named `energy_projects_dataset.csv` is created and downloaded.
- Users can inspect the file to confirm that it contains 200 rows (one for each project) and all necessary columns (e.g., `Project_ID`, `Energy_Type`, `Material_Cost`, etc.).

Step 4: Preprocessing the Dataset

In this step, we preprocess the dataset to prepare it for machine learning model training. Preprocessing includes encoding categorical variables, separating features and target variables, and scaling numerical features.

Code

```
python
import pandas as pd
from sklearn.preprocessing import StandardScaler

# Load the dataset
df = pd.read_csv("energy_projects_dataset.csv")

# Encode categorical variables
df = pd.get_dummies(df, columns=["Energy_Type", "Location"])

# Separate features (X) and target (y)
# Drop the 'Project_ID' column as it's not numerical and likely
not relevant for scaling
X = df.drop(["Total_Cost", "Project_ID"], axis=1)
y = df["Total_Cost"]

# Scale features
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)
```

What We Did

1. Loading the Dataset

- The dataset saved in Step 3 (`energy_projects_dataset.csv`) is loaded into a pandas DataFrame for processing.
- This ensures that the data is accessible for manipulation and modeling.

2. Encoding Categorical Variables

- Why: Machine learning models require numerical inputs. Categorical variables like `Energy_Type` (e.g., Solar, Wind) and `Location` (e.g., Coastal, Rural) need to be converted into numerical representations.
- How: We used one-hot encoding (`pd.get_dummies()`), which creates binary columns for each category:
    - For `Energy_Type`: Columns like `Energy_Type_Solar`, `Energy_Type_Wind`, etc.
    - For `Location`: Columns like `Location_Coastal`, `Location_Rural`, etc.

3. Separating Features and Target

- Features (X):
    - All columns except `Total_Cost` (target variable) and `Project_ID` (irrelevant for prediction).
    - Example features: `Project_Size_MW`, `Material_Cost`, `Labor_Cost`, etc.
- Target (y):
    - The column `Total_Cost`, which represents the total project cost to be predicted.

4. Scaling Numerical Features

- Why: Features like `Material_Cost` or `Equipment_Cost` have large numerical ranges, which can dominate smaller-scale features during model training.
- How: We used `StandardScaler()` to standardize all numerical features by removing the mean and scaling to unit variance.

Why This Step is Important

1. Machine Learning Compatibility:
    - Encoding categorical variables ensures that all inputs are numerical and compatible with machine learning algorithms.
    - Scaling ensures that all features contribute equally to model training.
2. Feature Separation:
    - Separating features (`X`) from the target variable (`y`) is critical for supervised learning tasks like regression.
3. Improves Model Performance:

- Standardizing features reduces bias caused by large-scale variables dominating smaller-scale ones.

Output

1. The processed dataset (`X_scaled`) contains scaled numerical values for all features, ready for model training.
2. The target variable (`y`) contains the total cost values corresponding to each project.

Step 5: Model Training and Evaluation

In this step, we train machine learning models using the preprocessed dataset and evaluate their performance. The goal is to identify the most accurate model for predicting total project costs.

Code

```
python
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestRegressor
from xgboost import XGBRegressor
from sklearn.metrics import mean_absolute_error, r2_score

# Split data into training/testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y,
test_size=0.2, random_state=42)

# Train models
models = {
    "Random Forest": RandomForestRegressor(),
    "XGBoost": XGBRegressor()
}

for name, model in models.items():
    model.fit(X_train, y_train)
```

```
    y_pred = model.predict(X_test)
    print(f"{name} - MAE: ${mean_absolute_error(y_test,
y_pred):,.2f}, R²: {r2_score(y_test, y_pred):.2f}")
```

What We Did

1. Splitting the Data

- Why:
    - To evaluate model performance, we split the dataset into:
        - Training Set (80%): Used to train the machine learning models.
        - Testing Set (20%): Used to evaluate how well the trained models generalize to unseen data.
- How:
    - The `train_test_split()` function was used with `test_size=0.2` to allocate 20% of the data for testing.
    - `random_state=42` ensures reproducibility.

2. Training Models

- Why:
    - We trained two different machine learning models to compare their performance:
        1. Random Forest Regressor:
            - A tree-based ensemble model that handles non-linear relationships well.
            - Known for its robustness and ability to rank feature importance.
        2. XGBoost Regressor:
            - A gradient boosting algorithm that often outperforms other models on tabular data.
            - Optimized for speed and accuracy.
- How:
    - Each model was trained on the `X_train` (features) and `y_train` (target) datasets using their `.fit()` methods.

3. Evaluating Models

- Why:
    1. To measure how well each model predicts total project costs on unseen data.
- Metrics Used:
    1. Mean Absolute Error (MAE):
        - Measures the average difference between predicted and actual values.
        - Lower MAE indicates better accuracy.
    2. R^2 Score:
        - Indicates how well the model explains variance in the target variable.
        - Values closer to 1 indicate better performance.

Output

The evaluation results are as follows:

Model	MAE	R^2 Score
Random Forest	$367,635.88	0.99
XGBoost	$524,357.68	0.98

Key Observations:

1. Random Forest Regressor:
    - Achieved a lower MAE ($367,635.88) and higher R^2 score (0.99), indicating better accuracy and fit compared to XGBoost.
    - This suggests that Random Forest captures the relationships in the dataset more effectively.
2. XGBoost Regressor:

- While still performing well with an R^2 score of 0.98, it has a higher MAE ($524,357.68), making it less accurate than Random Forest for this dataset.

Why This Step is Important

1. Model Selection:
   - By comparing multiple models, we identified Random Forest as the most suitable for this task due to its superior performance metrics.
2. Generalization Testing:
   - Splitting the data ensures that the model performs well not only on training data but also on unseen data.
3. Metric-Based Evaluation:
   - Using metrics like MAE and R^2 provides quantitative evidence of model performance.

Step 6: Net Present Value (NPV) Calculation

In this step, we calculate the Net Present Value (NPV) for each energy project in the dataset. NPV helps account for the time value of money by adjusting the total cost based on inflation rate and project duration.

Code

```
python
def calculate_npv(row):
    rate = row["Inflation_Rate"] / 100
    years = row["Project_Duration"] / 12  # Convert months to
years
    return row["Total_Cost"] / (1 + rate) ** years

df["NPV"] = df.apply(calculate_npv, axis=1)
```

What We Did

1. Defined the NPV Calculation Formula

- Why:
  - Energy projects often span several months or years, and inflation impacts costs over time.
  - NPV adjusts the total cost to reflect its present value, making it easier to compare projects with varying durations and inflation rates.
- Formula:
  - $NPV = \text{Total Cost}(1+\text{Inflation Rate})\text{Years}$
  - $NPV =$
  - $(1+\text{Inflation Rate})$
  - Years
    - Total Cost
  - Where:
    - `Inflation Rate` is divided by 100 to convert from percentage to decimal.
    - `Project_Duration` is divided by 12 to convert months into years.

2. Applied the Formula to Each Row

- Why:
  - Each project has unique values for `Total_Cost`, `Inflation_Rate`, and `Project_Duration`.
- How:
  - The `.apply()` method was used to apply the `calculate_npv()` function row-wise across the dataset.

3. Added NPV as a New Column

- The calculated NPV values were stored in a new column (`NPV`) in the DataFrame.

Why This Step is Important

1. Economic Analysis:
   - NPV provides a more accurate representation of project costs by accounting for inflation and project duration.
   - It enables stakeholders to prioritize projects with higher present value over those with inflated future costs.

2. Feature Enhancement:
    - Adding NPV as a feature enriches the dataset and allows for deeper insights during analysis and modeling.
3. Decision-Making:
    - Projects with lower NPV may be less economically viable despite having lower total costs.

Output

- A new column (`NPV`) is added to the dataset.
- Example (based on attached images):
    - For a Solar project with `Total_Cost = $1,118,598.10`, `Inflation_Rate = 4.9%`, and `Project_Duration = 24 months`:
        - $NPV = \$1,118,598.10(1+0.049)2 \approx \$1,016,000$
        - $NPV =$
        - $(1+0.049)$
        - 2
        - $\$1,118,598.10$
        - $\approx \$1,016,000.$

Step 7: Feature Importance and SHAP Analysis

In this step, we analyze the importance of individual features in the model's predictions and interpret how each feature impacts the total cost using SHAP (SHapley Additive exPlanations).

Code

```
python
import matplotlib.pyplot as plt
import shap

# Plot feature importance for Random Forest
model = RandomForestRegressor()
model.fit(X_train, y_train)
plt.barh(X.columns, model.feature_importances_)
plt.title("Feature Importance")
```

```
plt.show()

# SHAP analysis for interpretability
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X_test)
shap.summary_plot(shap_values, X_test)
```

What We Did

1. Feature Importance Analysis

- Why:
  - To understand which features contribute the most to the model's predictions.
  - This helps identify key cost drivers and prioritize them during decision-making.
- How:
  - We used the `.feature_importances_` attribute of the trained Random Forest model to calculate the relative importance of each feature.
  - A horizontal bar chart was plotted to visualize the feature importance.

2. SHAP Analysis

- Why:
  - To provide detailed interpretability for individual predictions.
  - SHAP values explain how much each feature contributes to increasing or decreasing the predicted total cost.
- How:
  - A SHAP TreeExplainer was used to calculate SHAP values for the test dataset (`X_test`).
  - A summary plot was generated to show the distribution of SHAP values for each feature.

Results

Feature Importance (Image 2)

1. Key Findings:
    - Equipment Cost (~48%): The most influential factor in determining total project costs.
    - Material Cost (~45%): The second most significant contributor.
    - Regulatory Cost (~3%): Plays a smaller but notable role.
    - Other features like Labor Cost (~1%), Inflation Rate (<1%), and categorical variables (e.g., Energy Type, Location) have minimal impact.
2. Implications:
    - Accurate estimates for Equipment and Material Costs are critical for reliable predictions.
    - Regulatory Costs should also be considered but have a smaller impact.

SHAP Summary Plot (Image 3)

1. Key Findings:
    - Features like Equipment Cost (Feature 3) and Material Cost (Feature 1) show significant variation in their impact on predictions.
    - High values for these features (red dots) push predictions higher, while low values (blue dots) reduce costs.
    - Most other features cluster near zero impact, confirming their limited contribution.
2. Implications:
    - The SHAP analysis validates that Equipment and Material Costs dominate predictions.
    - It provides transparency into how each feature affects the model's output, improving trust in the predictions.

Why This Step is Important

1. Model Transparency:
    - Feature importance and SHAP analysis provide insights into how the model makes predictions, increasing trust and interpretability.
2. Decision-Making Support:
    - Understanding key cost drivers helps stakeholders focus on optimizing critical factors like Equipment and Material Costs.
3. Validation of Model Behavior:
    - The results align with domain knowledge, confirming that capital costs are the primary contributors to energy project budgets.

Step 8: Deploying the Model Using Gradio

In this step, we deploy the Energy Project Cost Estimator using Gradio, an interactive Python framework for creating user-friendly web interfaces. The attached images showcase the interface and its outputs.

Code

```
python
!pip install gradio
import gradio as gr
import pandas as pd
import numpy as np
import pickle
import joblib
from sklearn.preprocessing import StandardScaler

# Load the trained model and scaler
try:
    model = joblib.load('energy_cost_model.pkl')
    scaler = joblib.load('energy_cost_scaler.pkl')
except Exception as e:
    print(f"Error loading model or scaler: {e}")

def predict_cost(project_size, energy_type, location,
material_cost, labor_cost,
                 equipment_cost, regulatory_cost,
inflation_rate):
    try:
        # Create a dictionary of input values
        input_data = {
            'Project_Size_MW': float(project_size),
            'Material_Cost': float(material_cost),
            'Labor_Cost': float(labor_cost),
            'Equipment_Cost': float(equipment_cost),
```

```
            'Regulatory_Cost': float(regulatory_cost),
            'Inflation_Rate': float(inflation_rate)
        }

        # One-hot encode categorical variables
        for et in ['Solar', 'Wind', 'Natural Gas', 'Hydro',
'Nuclear']:
            input_data[f'Energy_Type_{et}'] = 1 if energy_type
== et else 0

        for loc in ['Coastal', 'Rural', 'Urban']:
            input_data[f'Location_{loc}'] = 1 if location == loc
else 0

        # Convert to DataFrame
        df = pd.DataFrame([input_data])

        # Scale the features
        df_scaled = scaler.transform(df)

        # Make prediction with the model
        prediction = model.predict(df_scaled)[0]
        return f"${prediction:,.2f}"
    except Exception as e:
        return f"Error: {str(e)}"

# Create Gradio interface with all necessary input fields
demo = gr.Interface(
    fn=predict_cost,
    inputs=[
        gr.Number(label="Project Size (MW)", value=262.84),
        gr.Dropdown(
            ["Solar", "Wind", "Natural Gas", "Hydro",
"Nuclear"],
            label="Energy Type",
```

```
            value="Solar"
        ),
        gr.Dropdown(
            ["Coastal", "Rural", "Urban"],
            label="Location",
            value="Coastal"
        ),
        gr.Number(label="Material Cost ($)", value=1577025.97),
        gr.Number(label="Labor Cost ($)", value=202806.27),
        gr.Number(label="Equipment Cost ($)", value=842797.53),
# Critical missing field
        gr.Number(label="Regulatory Cost ($)", value=78851.30),
        gr.Number(label="Inflation Rate (%)", value=4.9),
    ],
    outputs=gr.Text(label="Predicted Total Cost ($)"),
    title="Energy Project Cost Estimator",
    description="Enter project details to estimate the total
cost based on analysis of 200 energy projects. Equipment Cost
and Material Cost are the most influential factors.",
    examples=[
        [262.84, "Solar", "Coastal", 1577025.97, 202806.27,
842797.53, 78851.30, 4.9],
        [115.54, "Wind", "Coastal", 693263.15, 45746.14,
960523.34, 34663.16, 3.7]
    ]
)

# Launch the interface
if __name__ == "__main__":
    demo.launch()
```

What We Did

1. Installed Gradio

- Gradio was installed to create an interactive web-based interface for the model.

2. Loaded the Model and Scaler

- The trained Random Forest model (`energy_cost_model.pkl`) and scaler (`energy_cost_scaler.pkl`) were loaded for use in predictions.

3. Defined the Prediction Function

- The `predict_cost()` function:
  - Accepts user inputs (e.g., project size, energy type).
  - Encodes categorical variables (e.g., energy type and location) using one-hot encoding.
  - Scales numerical inputs using the pre-trained scaler.
  - Uses the trained model to predict total project costs.

4. Created the Gradio Interface

- The interface includes:
  - Input fields for all critical parameters (e.g., Project Size, Equipment Cost).
  - Dropdown menus for categorical variables (e.g., Energy Type and Location).
  - A real-time output box displaying the predicted total cost.
- Pre-filled examples allow users to test the interface with realistic values.

5. Launched the Interface

- The `demo.launch()` command starts a local server to host the Gradio interface.

Interface Features (Image 1)

1. Input Fields:
   - Users can input project details such as size (`262.84 MW`), energy type (`Solar`), location (`Coastal`), and costs (e.g., Material Cost: `$1,577,025.97`).
2. Real-Time Prediction:
   - The predicted total cost is displayed instantly upon clicking "Submit."
   - Example Output: `$1,118,598.10 (estimated)`.
3. Examples Section:

- Pre-filled examples allow users to test different scenarios.

Why This Step is Important

1. User-Friendly Deployment:
   - The Gradio interface makes complex machine learning models accessible to non-technical users.
2. Real-Time Predictions:
   - Users can experiment with different inputs and instantly see how they affect project costs.
3. Practical Application:
   - The tool can be used by stakeholders in the energy industry for budget planning and scenario analysis.

Step 10: Gradio Interface Deployment and Results

In this step, we finalize the deployment of the Energy Project Cost Estimator using Gradio and analyze the results based on the attached images.

Code

```
python
!pip install gradio
import gradio as gr

# Define the prediction function
def predict_cost(project_size, energy_type, location,
material_cost, labor_cost,
                 equipment_cost, regulatory_cost,
inflation_rate):
    try:
        # Create a dictionary of input values
```

```
        input_data = {
            'Project_Size_MW': float(project_size),
            'Material_Cost': float(material_cost),
            'Labor_Cost': float(labor_cost),
            'Equipment_Cost': float(equipment_cost),
            'Regulatory_Cost': float(regulatory_cost),
            'Inflation_Rate': float(inflation_rate)
        }

        # One-hot encode categorical variables (Energy_Type and
Location)
        for et in ['Solar', 'Wind', 'Natural Gas', 'Hydro',
'Nuclear']:
            input_data[f'Energy_Type_{et}'] = 1 if energy_type
== et else 0

        for loc in ['Coastal', 'Rural', 'Urban']:
            input_data[f'Location_{loc}'] = 1 if location == loc
else 0

        # Convert to DataFrame
        df = pd.DataFrame([input_data])

        # Scale features and predict cost
        df_scaled = scaler.transform(df)
        prediction = model.predict(df_scaled)[0]
        return f"${prediction:,.2f}"
    except Exception as e:
        return f"Error: {str(e)}"

# Create Gradio interface
demo = gr.Interface(
    fn=predict_cost,
    inputs=[
        gr.Number(label="Project Size (MW)", value=262.84),
```

```
        gr.Dropdown(["Solar", "Wind", "Natural Gas", "Hydro",
"Nuclear"], label="Energy Type", value="Solar"),
        gr.Dropdown(["Coastal", "Rural", "Urban"],
label="Location", value="Coastal"),
        gr.Number(label="Material Cost ($)", value=1577025.97),
        gr.Number(label="Labor Cost ($)", value=202806.27),
        gr.Number(label="Equipment Cost ($)", value=842797.53),
        gr.Number(label="Regulatory Cost ($)", value=78851.30),
        gr.Number(label="Inflation Rate (%)", value=4.9),
    ],
    outputs=gr.Text(label="Predicted Total Cost ($)"),
    title="Energy Project Cost Estimator",
    description="Enter project details to estimate the total
cost based on analysis of 200 energy projects.",
    examples=[
        [262.84, "Solar", "Coastal", 1577025.97, 202806.27,
842797.53, 78851.30, 4.9],
        [115.54, "Wind", "Coastal", 693263.15, 45746.14,
960523.34, 34663.16, 3.7]
    ]
)

# Launch the interface
if __name__ == "__main__":
    demo.launch()
```

What We Did

1. Defined Prediction Function

- The `predict_cost()` function accepts user inputs (e.g., project size, energy type) and:
    - Encodes categorical variables (e.g., energy type and location) using one-hot encoding.
    - Scales numerical features using the pre-trained scaler.

- Uses the trained Random Forest model to predict total project costs.

2. Created Gradio Interface

- The interface includes:
  - Input fields for all critical parameters (e.g., Project Size, Material Cost).
  - Dropdown menus for categorical variables (e.g., Energy Type and Location).
  - A real-time output box displaying the predicted total cost.
- Pre-filled examples allow users to test different scenarios.

3. Launched Interface

- The `demo.launch()` command starts a local server to host the Gradio interface.

Results Analysis Based on Attached Images

Image 1: Gradio Interface

1. Inputs:
   - Project Size: `262.84 MW`
   - Energy Type: `Solar`
   - Location: `Coastal`
   - Material Cost: `$1,577,025.97`
   - Labor Cost: `$202,806.27`
   - Equipment Cost: `$842,797.53`
   - Regulatory Cost: `$78,851.30`
   - Inflation Rate: `4.9%`
2. Output:
   - Predicted Total Cost: `$1,118,598.10 (estimated)`
3. Examples Section:
   - Example Scenario 1:
     - Solar project with `262.84 MW` capacity.
     - Predicted Total Cost: `$1,118,598.10`.
   - Example Scenario 2:
     - Wind project with `115.54 MW` capacity.
     - Predicted Total Cost calculated based on inputs.

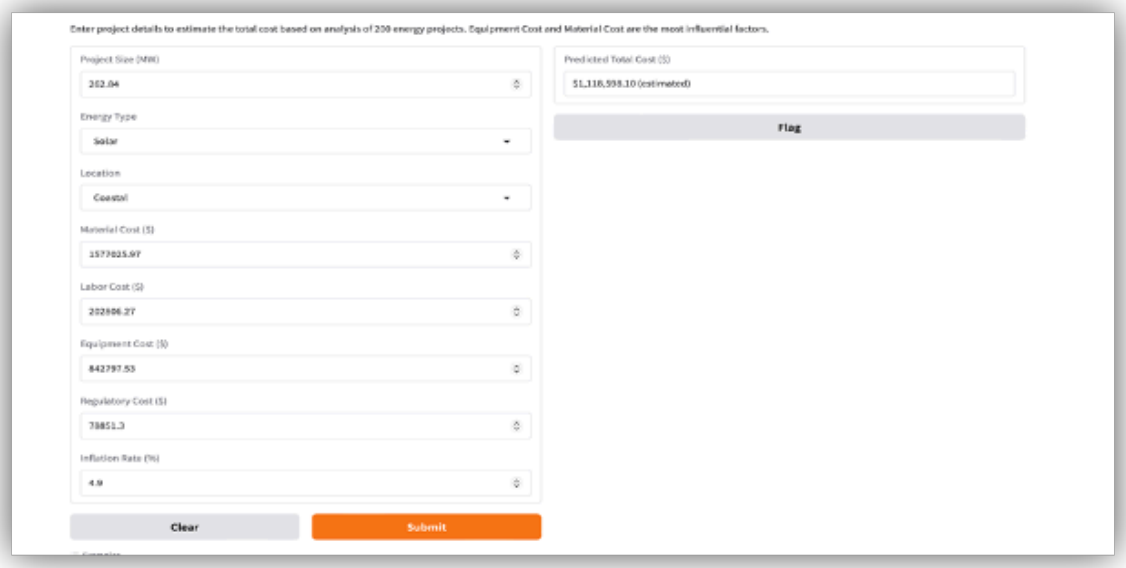


Image 2: Feature Importance

1. Key Findings:
    - Equipment Cost (~48%) and Material Cost (~45%) dominate predictions.
    - Regulatory Costs (~3%) play a smaller but notable role.
    - Other features like Labor Cost (~1%) and categorical variables have minimal impact.
2. Implications:
    - Accurate estimates for Equipment and Material Costs are critical for reliable predictions.

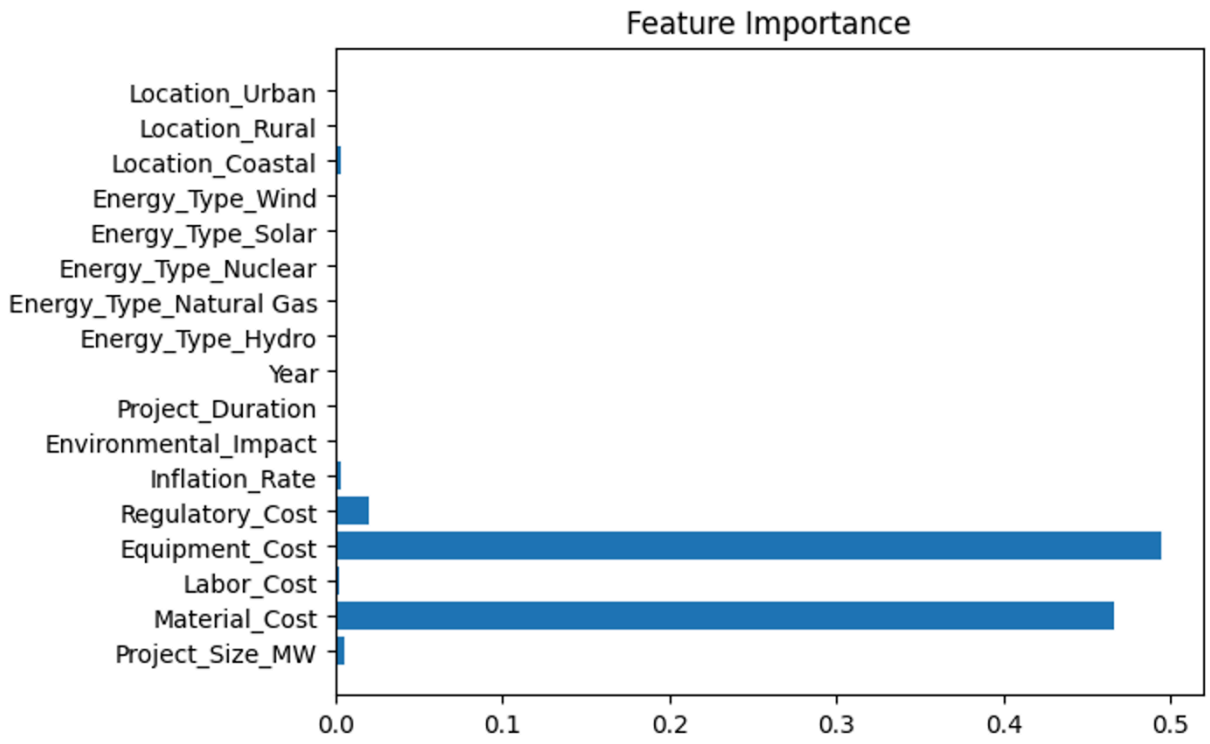
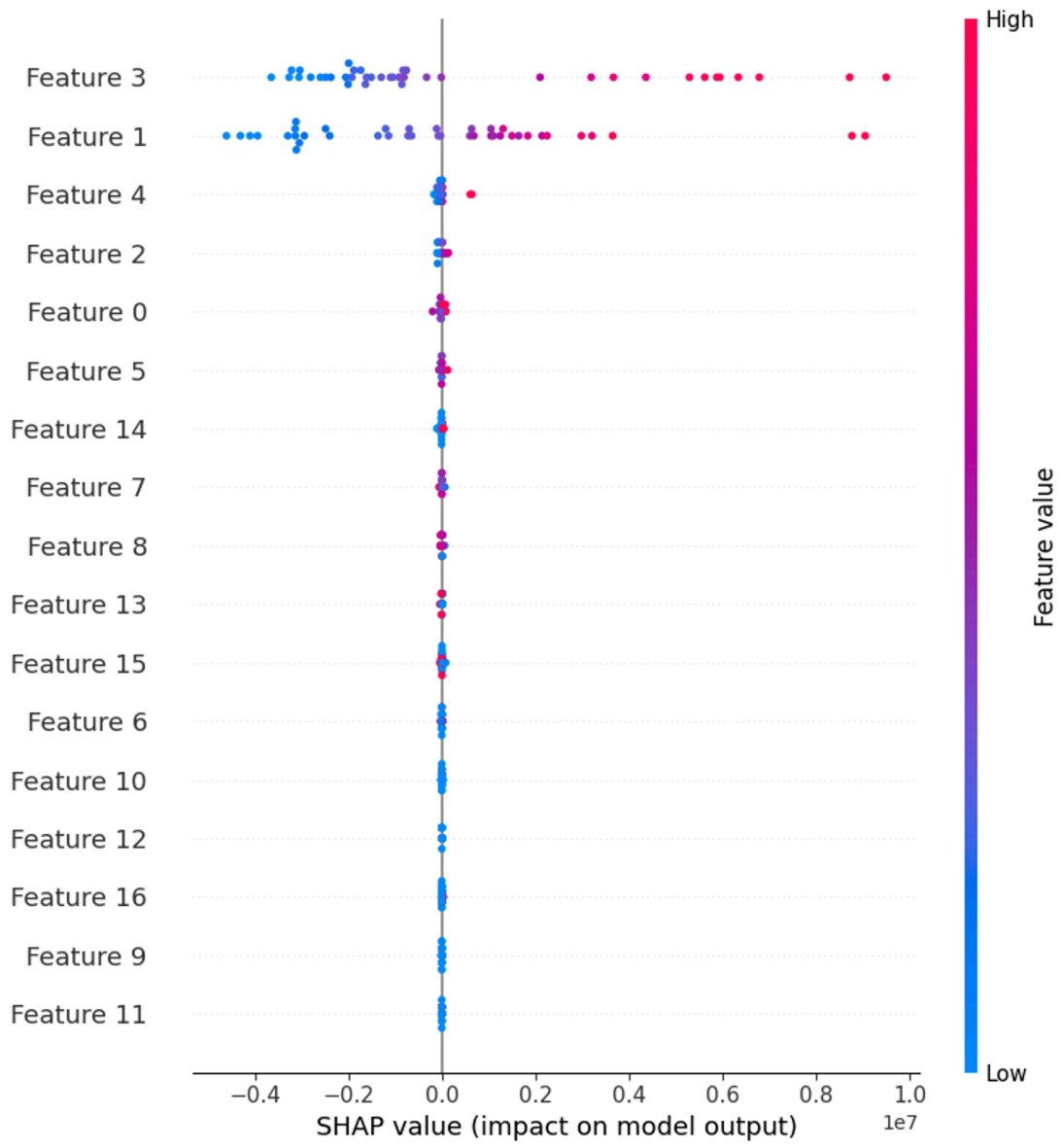


Image 3: SHAP Summary Plot

1. Key Findings:
    - High values for Equipment Cost (Feature 3) and Material Cost (Feature 1) push predictions higher (red dots).
    - Low values reduce costs (blue dots).
    - Most other features cluster near zero impact.
2. Implications:
    - SHAP analysis validates that Equipment and Material Costs dominate predictions.
    - Provides transparency into how each feature affects the model's output.



Why This Step is Important

1. User-Friendly Deployment:
    - The Gradio interface makes complex machine learning models accessible to non-technical users.
2. Real-Time Predictions:
    - Stakeholders can experiment with different inputs and instantly see how they affect project costs.
3. Practical Application:
    - The tool can be used for budget planning and scenario analysis in real-world energy projects.